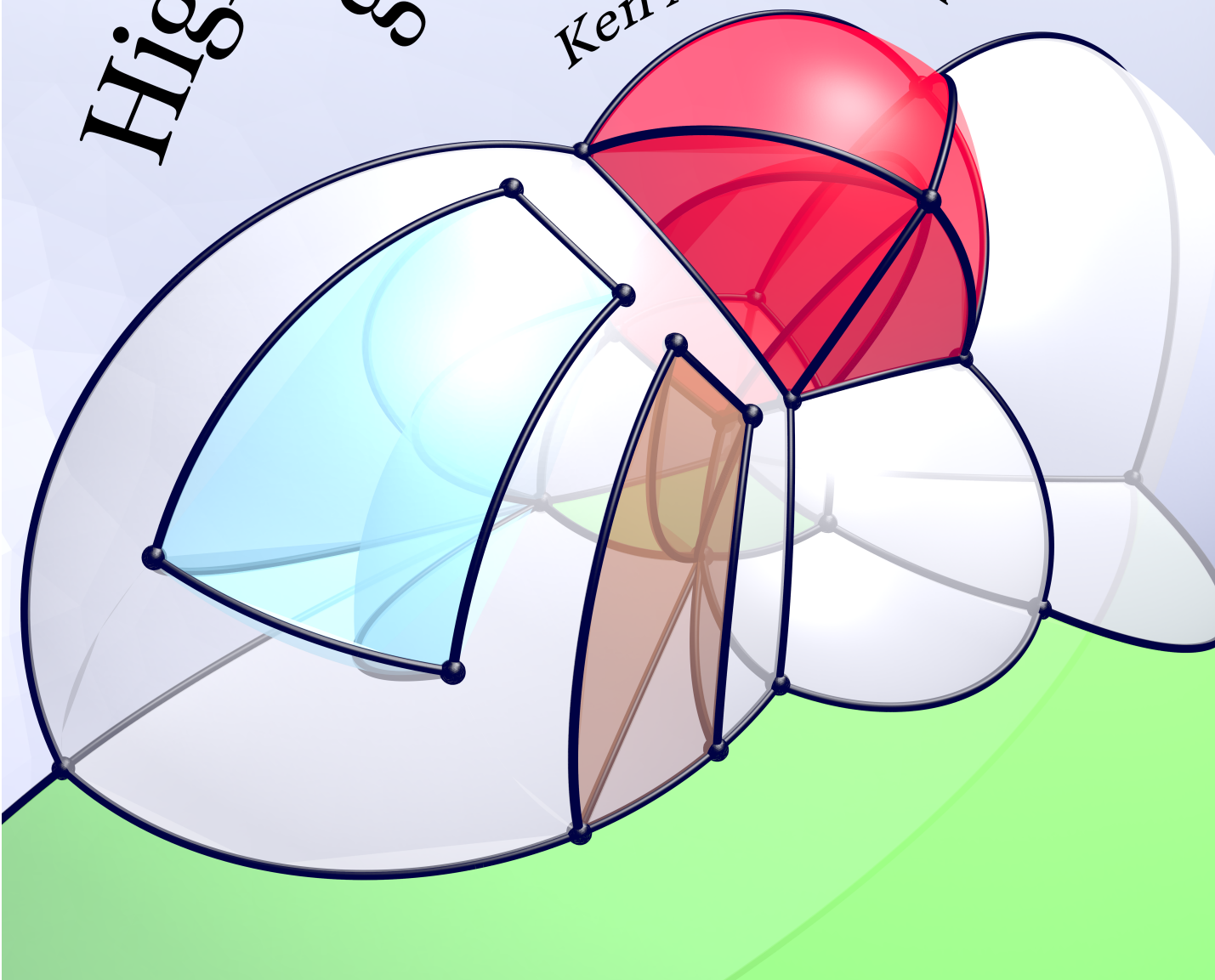
Higher-dimensional modelling of geographic information

Ken Arroyo Ohori



Higher-dimensional modelling of geographic information

Ken Arroyo Ohori

Written in 2014–2016 by Ken Arroyo Ohori.

ISBN: 978-1-326-59638-5

No copyright
This thesis is released into the public domain using the CC0 code. To the extent possible under law, I waive all copyright and related or neighbouring rights to this work.

To view a copy of the CC0 code, visit:
http://creativecommons.org/publicdomain/zero/1.0/

Caveat
I am very happy to release *my own text and figures* without any restrictions whatsoever. As far as I am concerned, any *attribution is very much appreciated* but it is not required. However, this does not except anyone from following the academic practices on attribution as applicable to them. Also, I must note that throughout this thesis I have used several excerpts of others' text, images and code, which I have always been careful to mark as such. While I am myself allowed to use these excerpts under legal *fair use* doctrines in many countries and more specifically by the citation right (*citaatrecht*) in Article 15a of the Dutch Copyright Law (*Auteurswet*), this does not mean that you are also free to use these excerpts for any purpose.

Colophon
This thesis was typeset with XeTeX 3.14159265-2.6-0.99992 (TeX Live 2015) using the Feijoa, GT Pressura and Asana Math typefaces. Most of the figures were created using OmniGraffle, Affinity Designer or Blender, often with the help of LaTeXiT.

The source code of this thesis is available at:
https://github.com/kenohori/thesis

Cover
Model of a 4D house represented as a 4D cell complex. The cells of the model were manually defined and embedded in $\mathbb{R}^4$, projected inwards/outwards to the volume of a 3-sphere (S^3), stereographically projected to $\mathbb{R}^3$ and exported as an .obj file. This was then imported in Blender and rendered using a perspective projection down to 2D. See §9.3 for more details.

Higher-dimensional modelling of geographic information

Proefschrift

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft,
op gezag van de Rector Magnificus prof. ir. K. C. A. M. Luyben,
voorzitter van het College voor Promoties,
in het openbaar te verdedigen op
6 april 2016 om 12.30 uur

door

Gustavo Adolfo Ken Arroyo Ohori

Master of Science in Geomatics
geboren te Mexico-Stad, Mexico.

This dissertation has been approved by the

promotor:	Prof. dr. J. Stoter
copromotor:	Dr. H. Ledoux

Composition of the doctoral committee:

Rector Magnificus	chairperson
Prof. dr. J. Stoter	Delft University of Technology, promotor
Dr. H. Ledoux	Delft University of Technology, copromotor

Independent members:

Prof. dr. E. Eisemann	Delft University of Technology
Prof. dr. M. van Kreveld	Utrecht University
Dr. R. Lindenbergh	Delft University of Technology
Prof. dr. ir. A. van Timmeren	Delft University of Technology

Other member:

Dr. G. Damiand	Claude Bernard University Lyon 1

This research was supported by the Dutch Technology Foundation STW, which is part of the Netherlands Organisation for Scientific Research (NWO), and which is partly funded by the Ministry of Economic Affairs (Project code: 11300).

Contents

1 Introduction 3
- 1.1 Research objective and scope 5
- 1.2 Structure of this thesis 6

I Representing geographic information 9

2 Mathematical foundations of spatial modelling 11
- 2.1 Elementary set theory and mathematical logic 11
- 2.2 Geometry 13
- 2.3 Topology 15

3 Modelling of 2D/3D space, time, scale and attributes 23
- 3.1 Spatial data models and data structures 23
- 3.2 Modelling of 2D and 3D space in practice 40
- 3.3 Spatiotemporal modelling 45
- 3.4 Modelling geographic scale 47
- 3.5 Key characteristics and shortcomings of current approaches 50

4 The higher-dimensional spatial modelling approach 53
- 4.1 Foundations of higher-dimensional modelling 54
- 4.2 The higher-dimensional modelling paradigm 55
- 4.3 Higher-dimensional data models and structures 63
- 4.4 Conclusions and possibilities 80

II Constructing and manipulating objects 83

5 Basic dimension-independent operations 85
- 5.1 Basic operations on certain data structures 86
- 5.2 Basic transformations of an nD scene 91
- 5.3 Spatial indexing 93
- 5.4 Duality in higher dimensions 95
- 5.5 Comparing two objects with and without signatures 97

6 Extrusion 101
- 6.1 Background 101
- 6.2 A dimension-independent extrusion algorithm 104

6.3 Implementation 113
6.4 Experiments 114
6.5 Extrusion-based generalisation operations 118
6.6 Conclusions 118

7 Incremental construction 121
7.1 Background and overall approach 122
7.2 Incremental construction of primitives per dimension 123
7.3 Implementation and complexity analysis 128
7.4 Experiments 130
7.5 Conclusions 132

8 Linking corresponding 3D models 135
8.1 Motivation and background 135
8.2 Suggested methodology and current issues 137
8.3 Use cases 143
8.4 A concrete example 147
8.5 Conclusions 147

9 Extracting information from higher-dimensional models 151
9.1 Background 151
9.2 3D to 2D projections 155
9.3 Higher-dimensional projections 157
9.4 Conclusions and possibilities 160

10 Processing real-world datasets into clean geometric models 163
10.1 Motivation 164
10.2 Creating valid (multi)polygons and planar partitions 166
10.3 Creating valid polyhedra and 3D space partitions 171
10.4 Dimension-independent validity criteria 182

11 Conclusions and future work 185
11.1 An outlook on higher-dimensional GIS 186
11.2 Lessons learned 187
11.3 Contributions 192
11.4 Future work 193

A Implementing higher-dimensional representations and operations 197
A.1 Main libraries used within this thesis 197
A.2 Geometric operations using computer arithmetic 198
A.3 Efficient and flexible dimension-independent programming 199

B A short dictionary of dimension-based GIS terms 205

Bibliography 207

Summary 227
Samenvatting (Dutch summary) 229
Resumen (Spanish summary) 231
Curriculum vitae 233

Figures

0.1 Sphere tries to explain the nature of 3D space to Square 1
0.2 Farnsworth explains why a two-ended digestive system cannot exist in 2D 1

1.1 A polygon as a cycle of points 4
1.2 The polygons in the CORINE dataset around Delft 4
1.3 A cube represented as the 6 square faces that bound it 4
1.4 Five dimensions based on 3D space, time and scale 5
1.5 A non-manifold shape 5
1.6 The top and bottom faces of this torus have holes in them. 5

2.1 A single line passes through two points 13
2.2 A point p in 3D described by a treble (p_x, p_y, p_z) 14
2.3 A plane separates $\mathbb{R}^3$ into two parts on either side of it. 14
2.4 Objects can be defined using Boolean set operations 15
2.5 Two rectangles and two points defined as point sets 16
2.6 Open and closed intervals 16
2.7 The unit open disk 17
2.8 An annulus partitions the Euclidean plane into three parts 17
2.9 A coffee mug and a donut are homeomorphic 17
2.10 The Seven Bridges of Königsberg 18
2.11 0-, 1-, 2- and 3-simplices 19
2.12 The TU Delft campus as a 3D simplicial complex 21
2.13 The TU Delft campus as a 3D cell complex 21
2.14 The Platonic solids 21

3.1 An object represented as a tree of Boolean set operations 28
3.2 A polygon represented as the union of two convex polygons 29
3.3 A Nef polygon represented by a set of local pyramids 29
3.4 A wireframe model can have different interpretations 33
3.5 Pascal's triangle 35
3.6 A triangle-based data structure 35
3.7 A polygonal curve or polyline 36
3.8 The polygon model 36

3.9 The spaghetti model 36
3.10 Storing non-manifold polygons in a half-edge data structure 37
3.11 A 2D cell complex as a DCEL and a 2D combinatorial map 37
3.12 The quad-edge data structure 38
3.13 The facet-edge data structure 38
3.14 A V-map 39
3.15 A selective Nef complex 39
3.16 A triangle strip 41
3.17 Geometry in the Simple Features Specification 42
3.18 Geometry in GML 43
3.19 Sweeps in IFC 43
3.20 The snapshot model 46
3.21 An event-based model 47
3.22 Topographic maps at different scales around Delft 48
3.23 The LODs in CityGML 49

4.1 A tesseract 56
4.2 An object remaining stationary as an extrusion in 2D space+time 57
4.3 Object transformations in 2D space+time 58
4.4 A 3D representation of 2D space+time 58
4.5 A 3D representation of 2D space+scale 59
4.6 A 2D+scale representation of four polygons being generalised 59
4.7 Simple Features representations of a square, cube, and tesseract 61
4.8 The Schönhardt polyhedron 67
4.9 Relationships in a simplex-based data structure 69
4.10 Relationships in an incidence graph 72
4.11 Two different techniques to handle holes in a 3D cell complex 73
4.12 Simplices in a generalised map and a combinatorial map 75
4.13 A cube represented as a 3D combinatorial map 75
4.14 Orientation in a combinatorial map 77
4.15 Representing holes using bridges 78

5.1 Involutions in a 2D generalised map 86
5.2 Partial permutations in a 1D combinatorial map 87
5.3 Partial permutations in a 2D combinatorial map 87
5.4 Darts of a cell 88
5.5 3-sewing two cubes 89
5.6 Boolean set operations on Nef polygons 90
5.7 Duality in a 2D map 95
5.8 A 2D generalised map and its dual 96
5.9 Three pairs of tesseracts 97

6.1 A view of the 3D TOP10NL dataset 102
6.2 Extrusion from 2D to 3D 103
6.3 The Frauenkirche in Dresden 103
6.4 The extrusion intervals of the lower-dimensional cells 105
6.5 Extruding the embeddings 106
6.6 Extruding a 2D cell complex 106
6.7 Propagating the extrusion intervals 107

6.8 The darts in the cell complexes in Figure 6.5 110
6.9 The stack of simplices in an extruded dart 111
6.10 The darts in the cell complexes in Figure 6.6 112
6.11 Extruding the footprint of the Aula Congress Centre in Delft to 3D 116
6.12 Extruding the footprint of the Aula Congress Centre in Delft to 4D 116
6.13 Extruding a dataset of the campus of the Delft University of Technology 117
6.14 Collapsing cells 118

7.1 Two adjacent tetrahedra as a combinatorial map 123
7.2 Constructing the 0-cells of Figure 7.1 124
7.3 Constructing the 2-cells of Figure 7.1 125
7.4 Constructing the 3-cells of Figure 7.1 128
7.5 A tesseract as a combinatorial map 130
7.6 Simple 2D+scale datasets 131
7.7 A large 2D+scale dataset 132
7.8 Construction time speed-up from the use of indices 133

8.1 Two LODs of a building footprint 136
8.2 Two LODs of a building footprint as a single polyhedron 138
8.3 Four linking schemes 141
8.4 Simple linking 144
8.5 Linking by collapsing 144
8.6 Linking using topology modifications 145
8.7 Linking using multiple methods 146
8.8 Linking by matching cells 146
8.9 Programming a complex linking example 148

9.1 Conic sections 152
9.2 Orthographic and perspective projections 153
9.3 Equirectangular projection 153
9.4 Grand Teton National Park map 154
9.5 Perspective projection's frustum and orthographic projection's box 155
9.6 Geometry of an orthographic projection 156
9.7 Geometry of an perspective projection 157
9.8 Stereographic projection 160
9.9 Polyhedron and polychoron in Jenn 3D 160
9.10 4D to 2D projection of a 4D house 161

10.1 Different interpretations of a polygon 165
10.2 Several invalid polygons 166
10.3 Rules used to interpret the interior of a polygon 168
10.4 Defining a snapping threshold 169
10.5 Steps to repair a (multi)polygon using a constrained triangulation 169
10.6 Processing the largest polygon in the CORINE land cover dataset 170
10.7 Steps to repair a planar partition using a constrained triangulation 170
10.8 Various repair methods based on relabelling (sets of) triangles 171
10.9 A planar partition of 16 tiles of the CORINE land cover dataset 171
10.10 Surface-based models vs. volumetric models 172
10.11 An IFC model of the FZK-house 173

10.12 The boundary representation scheme in ISO 19107 173
10.13 Openings in the IfcOpenHouse dataset 175
10.14 The volumes in the IfcOpenHouse dataset do not fit together 176
10.15 A large overlap in the IfcOpenHouse dataset 177
10.16 Computing the best fitting plane of every face 180
10.17 Vertex snapping 180
10.18 The steps of the IfcOpenHouse 181
10.19 A dimension-independent cell harmonised with ISO 19107 183
10.20 Cell complexes in the ISO 19107 standard 184
10.21 An *n*D space subdivision harmonised with ISO 19107 184

A.1 Using C++ templates to convert a `string` into any number type 201
A.2 Creating a dependent type using C++ templates 201
A.3 Using recursive C++ templates to produce dimension independent code 202
A.4 Dimension-independent algorithms using recursive C++ templates 203

1 A cube represented as the 6 square faces that bound it 227
2 3D space, time and scale can be modelled as 5D space. 227
3 3D space subdivision model 227
4 A set of polygons is converted into a set of boxes by 2D-to-3D extrusion. 228
5 Two LODs of a 3D model of a house are linked into a 4D model 228

1 Een kubus wordt gerepresenteerd als 6 vierkante 2D vlakken. 229
2 3D ruimte, tijd en schaal kunnen worden gemodelleerd als 5D ruimte. 229
3 3D ruimtelijke opdeling 229
4 Een set polygonen wordt geconverteerd naar een set blokken. 230
5 Twee detailniveaus van een 3D model worden gelinkt tot een 4D model. 230

1 Un cubo representado por las 6 caras cuadradas en su superficie 231
2 El espacio 3D, el tiempo y la escala modelados como un espacio 5D. 231
3 Partición espacial 3D 231
4 Un conjunto de polígonos se extrude en un conjunto de paralelepípedos. 232
5 Dos niveles de detalle de una casa se enlazan en un modelo 4D 232

Preface

We usually associate the three usual dimensions with length, width and height (in any order), and the fourth dimension with time. However, mathematically, dimensions are essentially a artificial construct and do not have a fixed meaning linked to any specific aspect of reality. By associating each dimension with a variable, they can be used to represent pretty much anything that we can put a parameter to. Within this thesis, representations of any dimension are thus used to model various aspects of geographic information, such as the typical geographic coordinates, but also time and scale.

At the end of the 19th century, a couple of non-scientific books were notable for popularising thinking about dimensions, the satyrical *Flatland: A Romance of Many Dimensions* [Abbott, 1884] (Figure 0.1), and the more serious *A New Era of Thought* [Hinton, 1888]. These books used playful analogies between familiar 0D–3D situations and those in higher dimensions, attempting to give readers an intuitive feeling of what these abstract higher dimensions are like. Plenty of others continue to follow in their footsteps (Figure 0.2).

In order to ensure that all descriptions are formally correct, this thesis necessarily uses some formal descriptions for its data structures and algorithms. However, it also tries to provide intuitive explanations and analogies across different dimensions, offering explanations of higher-dimensional problems based on our intuitive knowledge of similar 2D and 3D cases. I hope that the result is approachable and serves to broach the subject in a practical way, and that potential readers (if any) are not put off by overly technical explanations or by the overly simplistic examples that I am able to draw.

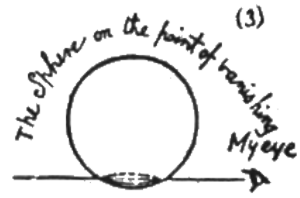


Figure 0.1: As Sphere tries to explain the nature of 3D space to Square by passing through its plane of view: *'You cannot indeed see more than one of my sections, or Circles, at a time; for you have no power to raise your eye out of the plane of Flatland; but you can at least see that, as I rise in Space, so my sections become smaller'* [Abbott, 1884].

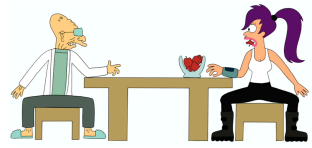


Figure 0.2: As Professor Farnsworth explains why a two-ended digestive system cannot exist in a 2D world: *'As you can see, or rather can't see, but take my word for it, such a digestive system would divide a 2D being into separate pieces.'*. From Futurama season 7 episode 14.

About this thesis

During the past 4+ years I have found many things to love about the scientific process, but many things to hate about how its results are released and published. More than a century after Leo Tolstoy's *Letter to the Free Age Press*[1] and 20 years after Stevan Harnad's *subversive proposal*[2], it is rather sad to see that so much science is still kept behind paywalls or stymied by legal restrictions even when it is taxpayer-funded. This is not only damaging to scientific discourse, but it keeps important knowledge from the public at large, including educators and legislators.

1: https://en.wikisource.org/wiki/Letter_to_the_Free_Age_Press

2: https://en.wikipedia.org/wiki/Subversive_Proposal

At the same time, there is a concerted effort to restrict knowledge creation and creative expression through draconian intellectual property laws, including providing IP holders with special (extra)legal privileges and misappropriating author and user rights while extending already excessive copyright terms. Plenty of laws have been enacted on the matter with a complete lack of transparency, despite obvious harm to users and evidence to a lack of economic benefits [Hargreaves, 2011; Reda, 2014; EFI, 2015].

By releasing this thesis' contents into the public domain and making its source publicly available, I wish to make a small statement about how I believe science should be distributed in the future—openly and without restrictions. Open access journals are a step in the right direction[3], but I hope it is not too long before we are able to get rid of profit-driven journals and publishers altogether.

3: Even as many of them have abusive fees that bear no relation with the marginal cost of internet distribution.

Following in the same spirit, the source code of the main prototype implementations that were developed during this PhD project have also been made publicly available under permissive licences, which are linked to in the corresponding parts of this thesis. As others have pointed out before me[4], there is a great disparity between the alleged importance of scientific evidence and widespread acceptance of not disclosing software implementations.

4: See for instance Morin et al. [2012], Joppa et al. [2013] and Ince et al. [2012]

Acknowledgements

My utmost thanks go to my supervisors, **Hugo** and **Jantien**. They not only provided me with *years* of employment and the wonderful opportunity to work on this PhD, but were always there with good ideas, and all the guidance and support anyone could ever wish. I am also very grateful to **Guillaume** for hosting me in Lyon and helping me to understand the details of his CGAL packages, which were used in large parts of this thesis.

Special thanks go to everyone else in the **3D geoinformation** group during my time there (**Filip**, **Liu**, **Ravi**, **Sisi** and **Zhiyong**). It is truly a fantastic place to work and I feel privileged to have been able to contribute to it with my grain of sand.

I am also thankful to everyone at **GIS technology** (**Edward**, **Elfriede**, **Marian**, **Martijn**, **Radan**, **Theo**, **Tjeu**, **Peter** and **Wilko**). Many of them gave me insightful comments and support, especially so for Peter at the start of this PhD.

Last but not least, I especially thank my mother, **Kimiyo**, who has always been there supporting me, and my **close friends** in Mexico, the Netherlands and elsewhere, who made sure I always had things to enjoy outside work. I could not mention you all by name without making a dubious classification, which I do not want to do, and so I hope you will forgive me for just refusing to do so. *Thank you all.*

1 Introduction

Spatial information describes the location of objects in space and the relationships between them. Within this thesis the emphasis is on *geographic* information as represented in a *computer*, which uses similar techniques but narrows this definition to information that is about the real world and at a human-to-Earth scale, using abstracted digital representations of real-world entities such as terrain, cities, roads and buildings. These entities and the relationships between them are defined using sets of interlinked computer primitives. Because of this, spatial information forms a necessary component of any computer model of the world as we know it at any significant level of detail.

The core of the research carried out in this thesis is concerned with *geographic information systems (GIS)*[6], which produces tools to create, manipulate, analyse and visualise the digital objects that are inherent in these abstract representations of the world. Compared to other software categories that also allow us to model and manipulate objects that are represented geometrically, such as those used in computer-aided design (CAD) and geometric modelling, GIS tools stand out as being *remarkably generic* [Coppock and Rhind, 1991; Gold, 2006]. GIS are used equally to manually build objects by applying interactive drawing operations, to semi-automatically create full models from raw acquired data, to manage large collections of heterogeneous datasets and keep them up to date, or as interactive point-and-click environments to query the attributes of and perform simple calculations on existing datasets. Because of this genericity, GIS are expected to support a large number of different data formats from multiple sources and a wide variety of operations—all while solving problems in a mix of 2D and 3D and preserving the key characteristics of sometimes mutually incompatible computer representations. *Rather than attempting to find the best solution on input fulfilling strict conditions, GIS tools are expected to make a best effort to obtain a good solution on the often invalid data that is available.* This thesis follows this philosophy to a large extent.

For historical and practical reasons, current GIS mostly use simple 2D representations [ESRI, 2005; OGC, 2011], which are relatively easy to use and efficient, essentially consisting of sets of linked points, lines and polygons. Their efficiency is due to the fact that

6: Within this thesis, I always use 'GIS' and not 'GISs'. This is partly because of aesthetics (GISs reads badly and hears worse), but also because I would argue that current GIS are not really systems but disparate collections of methods, tools and processes. As such, when I write about GIS I rarely refer to systems and most often use the word as a modifier rather than a noun.

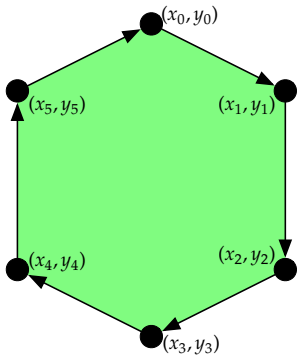


Figure 1.1: A polygon as a cycle of points

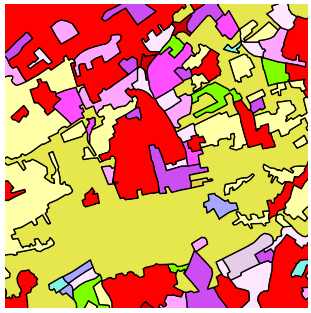


Figure 1.2: The polygons in the CORINE[7] dataset in the area around Delft [CEC, 1995]

7: http://www.eea.europa.eu/publications/CORO-landcover

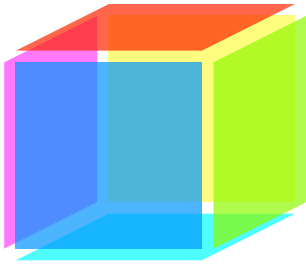


Figure 1.3: A cube represented as the 6 square faces that bound it

they can rely on many strong properties that are intrinsic to 2D objects. For example, one such property is that it is possible to define a natural order for the points around a closed polygonal curve, as shown in Figure 1.1, and so a polygon can be represented as a sequence of points that form a cycle along its boundary [Jordan, 1887], which are implicitly connected by line segments between each consecutive pair. Another such property is based on planar partitions—sets of polygons that form a subdivision of the plane—such as the one in Figure 1.2. Since in a planar partition there are two polygons incident to any edge (except those bordering the exterior), a complete planar partition can be stored easily as a set of edges where every edge is linked to two other edges and records the polygons that lie on each of its two sides [Peucker and Chrisman, 1975].

2D GIS are also able to take advantage of a great number of existing techniques that are based on 2D representations, such as those that are used to model the flow of water over a terrain [van Kreveld, 1997a] or to combine multiple maps into one (i.e. a map overlay) [de Berg et al., 2008, §2.3]. Moreover, these techniques have been developed and improved over decades, whereas any change in representation would require the development of new accompanying techniques in order to be truly useful.

The aforementioned advantages of 2D representations mean that even '3D' GIS usually mimic the third dimension by using a so-called 2.5D structure, essentially treating the third dimension as a simple attribute that is attached to each object [Raper, 1989], or represent individual 3D objects only implicitly through the 2D surface that separates their interior from their exterior [Edmonds, 1960; Baumgart, 1975], as the cube in Figure 1.3, rather than as true solid objects. These solutions are compromises, as they limit the type of geometries that can be represented and complicate the storage of the relations that exist between 3D objects.

Another important consequence is that when non-spatial characteristics that have a strong link to space, such as time and scale, are integrated in a GIS, they are usually implemented using similar adaptations of 2D representations. For example, spatiotemporal GIS keep multiple representations of 2D structures [Armstrong, 1988], each at a different point in time, or a list of changes per object [Worboys, 1992a; Peuquet, 1994], while multi-scale datasets generally consist of independent datasets for each scale with some common identifiers that link objects between datasets [Friis-Christensen and Jensen, 2003; Stoter et al., 2014].

The use of 2D representations also limits the capabilities of GIS software, as the techniques that can be implemented on top of these 2D representations cannot make full use of the potential of 3D spatial information [Zlatanova, 2000, Ch. 3]. For instance, current GIS are largely unable to perform complex manipulations of 3D objects or to compute geometric operations between multiple 3D objects, forcing

users to perform such functions in 3D modelling software. These examples are notable as equivalent functionality in 2D is widespread in 2D GIS and expected by its users.

A potential solution to solve the representation problems of 3D, spatio-temporal and multiscale data is opting out of further ad hoc adaptations of 2D structures. Instead, this thesis shows that it is possible to represent certain parametrisable characteristics as additional dimensions in the geometric sense, as shown in Figure 1.4, such that real-world (0D–3D) entities are modelled as *higher-dimensional objects embedded in higher-dimensional space* [van Oosterom and Stoter, 2010]. A building existing over a time span and stored at a variety of scales on a computer could thus be represented as a single 5D object.

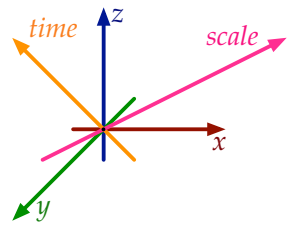


Figure 1.4: Five dimensions based on 3D space, time and scale

The *higher-dimensional geographic information modelling* approach is well grounded in long-standing mathematical theories, such as the setting of coordinates to space [Descartes, 1637], and theories of higher-dimensional geometry [Riemann, 1868] and topology [Poincaré, 1895]. It also opens the door for new, more powerful techniques and practical applications. For example, these n-dimensional representations could be used to ensure that an object is consistent along all dimensions (e.g. a building at different points in time or levels of detail), or to analyse its relations to other objects along all dimensions (e.g. whether two moving objects were ever adjacent).

There is a large body of related work on the representation of abstract n-dimensional objects in mathematics and computer science [Brisson, 1993; Lienhardt, 1994], as well as a few instances of work on its application to GIS [Karimipour et al., 2010]. However, by and large *higher-dimensional representations remain unanalysed in the context of geographic information*. For instance, it is necessary to see how abstract representations can be made fit for use with real-world objects, which have aspects that are difficult to handle, e.g. certain kinds of geometries (Figure 1.5), holes (Figure 1.6) and complex semantics. Additionally, making this approach attractive in practice requires the development of high-level techniques to construct, analyse and visualise higher-dimensional geographic objects. Finally, there are significant technical issues involved in the realisation of these representations and techniques into a computer implementation. All of these aspects, which in short encompass *concepts, representations, operations and visualisation*, are tackled in this thesis.

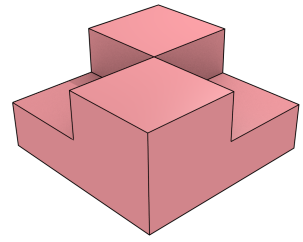


Figure 1.5: This non-manifold shape cannot be handled properly in many representations.

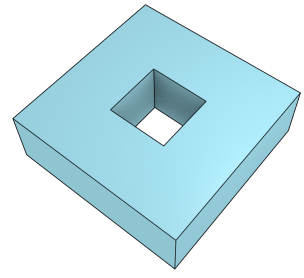


Figure 1.6: The top and bottom faces of this torus have holes in them.

1.1 Research objective and scope

In order to determine whether the *higher-dimensional modelling of geographic information* is worthwhile, and if so, the conditions un-

der which it makes sense to follow it, it is necessary to first gain a greater understanding of the entire modelling process in higher dimensions as well as its technical consequences. The main aim of this thesis is therefore to gain this understanding by **realising the fundamental aspects of a higher-dimensional Geographic Information System**, including the development of the necessary modelling concepts that are analogous to familiar modelling concepts in 2D/3D GIS, the use of appropriate higher-dimensional representations, and the development of simple higher-dimensional operations. While not every aspect of a higher-dimensional GIS can be fully developed within the timeframe allotted for this thesis, prototype-level working software is created whenever possible and tested using real-world higher-dimensional datasets[8]. In this sense, this thesis does not intend to prove that the higher-dimensional spatial modelling approach is better or worse than existing approaches, but instead to highlight its advantages and disadvantages so as to better evaluate it for future applications.

8: In this instance, real-world does not imply tackling all the problems inherent in using the entirety of the very large, complex and invalid datasets that are currently available. Due to the experimental nature of this thesis, small subsets of these real-world datasets that showcase specific problems are necessarily chosen.

1.2 Structure of this thesis

This thesis comprises two main parts, each of which contains a few chapters and which respectively cover methods to solve problems inherent in: (i) representing objects in arbitrary dimensions, and (ii) creating and manipulating such objects. After these, there are some independent chapters that fall outside the two parts, covering practical aspects such as the processing of real-world (invalid) data, implementation details and how the results of all chapters come together. All chapters are described in detail below.

Part I Representing geographic information

Chapter 2 introduces the mathematical concepts and background behind spatial data modelling.

Chapter 3 describes and analyses the state of the art in the 2D and 3D modelling of space, time, scale and attributes. It concludes by listing some of the shortcomings of current approaches.

Chapter 4 presents the higher-dimensional modelling paradigm that is the basis of this thesis, which aims to solve many of the problems alluded to in Chapter 3. It also describes and evaluates the higher-dimensional representations that can be used in order to realise this approach, which take the form of n-dimensional data models and structures.

Part II Constructing and manipulating objects

Chapter 5 presents a few fundamental operations for some of the data structures described in Chapter 4. These are used in order to build the higher level operations described in the other chapters within Part II.

Chapter 6 describes n-dimensional extrusion, an extension of the well-known 2D to 3D extrusion operator in GIS. It can be used to generate simple prism-shaped n-dimensional objects from an $(n-1)$-dimensional space partition by assigning to each $(n-1)$-dimensional object a range along which it exists.

Chapter 7 describes incremental construction, an operation which is able to generate arbitrary n-dimensional objects based on defining their $(n-1)$-dimensional boundary. It is equivalent to the generation of the topological relationships that exist between a set of $(n-1)$-dimensional objects.

Chapter 8 shows how a higher level construction operator that links a series of 2D or 3D models can be created. This enables the construction of 4D models from real-world 3D city models covering the same region.

Chapter 9 explains how 2D and 3D information can be extracted from a higher-dimensional representation by selecting appropriate portions of the data (e.g. cross-sections) and projecting it to 2D/3D space.

Chapter 10 explains the main data validation and repair techniques that are used to create 2D/3D/nD objects and space partitions out of real-world GIS datasets, which often have defects that impede their usage. By obtaining clean geometric models, these techniques enable the use of the higher-dimensional modelling approach in practice.

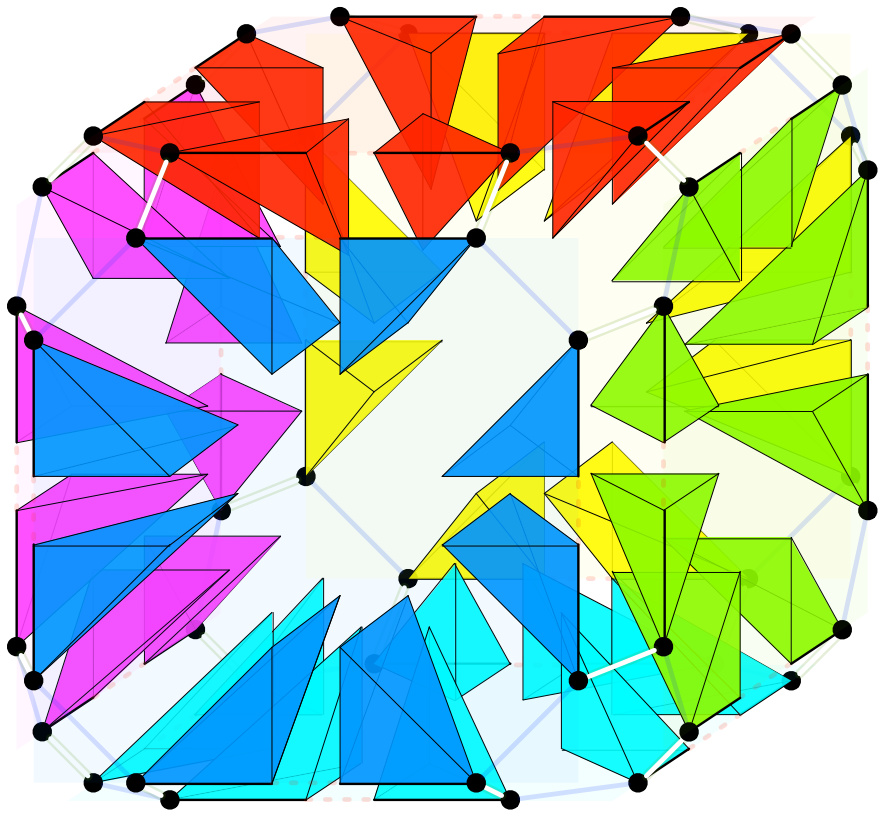
Chapter 11 concludes this thesis by explaining the achievements of this PhD project and highlighting the main challenges to further develop a higher-dimensional GIS. It contains an outlook on how higher-dimensional modelling can be used in GIS as well as some suggestions for future work.

Appendix A describes the most relevant implementation details concerning the representations and operations from the previous chapters, such as the use of various software libraries, robust geometric operations and programming techniques.

Appendix B contains a short dictionary of terms for higher-dimensional GIS. As the terminology used in 2D GIS, 3D GIS, CAD, geometric modelling and other related fields can be different and is often used inconsistently and rather loosely, the appendix is intended for use as a general reference and for the better understanding of the thesis.

Part I

Representing geographic information



Mathematical foundations of spatial modelling

2

Spatial modelling has its origins in the geographical notion of space, which is in turn based on our own observations of the world and empirical experience [Couclelis, 1999]. However, these informal notions are error-prone and differ from person to person. In order to describe space unambiguously, people have thus turned to models that still describe geographical phenomena, but do so using formal notions derived from mathematics. These formal models make it possible to create and store digital representations of the world in a computer, and thus to use the power of a computer to easily solve spatial problems [Burrough, 1986; Bailey and Gatrell, 1995].

The current chapter describes some of these formal notions and their relevant context, which are used to study the spatial modelling approaches presented in the upcoming chapters. §2.1 introduces some concepts of elementary set theory and mathematical logic, which are later used in definitions in this thesis. §2.2 introduces the basic concepts of geometry, which are used to describe the position, shape and orientation of objects. §2.3 builds on these to present topology, which formalises notions such as the boundary and interior of an object or the relationships between multiple objects.

2.1 Elementary set theory and mathematical logic

Set theory is the branch of mathematics that studies *sets*, which are collections of abstract objects. While the study of set theory only formally started with Cantor [1874], its intuitive and minimal concepts were later used in order to give a foundation to almost all areas of mathematics[10]. Since the basic concepts of set theory are used in this thesis in order to describe many other concepts, this section gives a very short primer using the same notation that is used in this thesis. However, it is worth noting that the descriptions used here are reflect the concepts generally used in GIS, and so are meant to be intuitive and not very formal. Perhaps more importantly, these definitions do not reflect modern mathematical thought on the topic[11], which is much more precise but also less accessible, e.g. axiomatic set theory.

10: Even as some mathematicians and philosophers have argued against set theory as a foundation for all of mathematics.

11: In short, these intuitive definitions pretty much assume that anything can be put into a set without leading to paradoxes, which formally is not the case.

Set theory starts by considering the existence of a given domain of objects from which one may build sets, which is known as the *universe set* and denoted as $\mathbb{U}$. These objects can be anything, including other sets. Set theory allows sets to be regarded as single entities and operated upon [Devlin, 1993]. If an object a is part of a set $\mathbb{X}$, it is denoted as $a \in \mathbb{X}$, read as 'a is an *element* of $\mathbb{X}$'. If a is not part of a set $\mathbb{X}$, it is denoted as $a \notin \mathbb{X}$, read as 'a is not an element of $\mathbb{X}$'.

There are two broad ways to describe the elements in a set, both using curly braces, i.e. { and }. One way to do so is to enumerate all the elements of the set one by one. For instance, the set $\{1, 2, 3\}$ is the set containing 1, 2 and 3 as elements (and no others). The other way to do so is to specify one or more rules that the elements of the set need to fulfil. For instance, the set $\{x \mid x \textit{ is a prime number}\}$ consists of all prime numbers. It is read as 'x, such that x is a prime number'.

Sets are by definition unordered and contain unique elements—duplicate items are ignored by convention. A set may contain an infinite number of elements (e.g. as the prime number example above), or no elements at all, in which case it is a special set known as the *null set* and denoted as $\{\}$ or $\varnothing$. Other commonly used sets with a special notation and name are: the natural numbers ($\mathbb{N}$), the real numbers ($\mathbb{R}$), the rational numbers ($\mathbb{Q}$) and the integers ($\mathbb{Z}$).

In order to build more complex sets, the concepts and notation from mathematical logic are used, in particular *propositional logic*. Propositional logic works with *propositions*, which are sentences that are either true or false, but not both. These propositions might be altered and combined using various symbols expressing various notions, such as: *and* ($\wedge$), *or* ($\vee$), *not* ($\neg$), *implies* ($\Rightarrow$), *is implied by* ($\Leftarrow$), *if and only if* ($\Leftrightarrow$), *for all* ($\forall$) and *exists* ($\exists$).

Using these concepts it becomes possible to state relationships between sets. For instance, $\mathbb{A}$ and $\mathbb{B}$ are then equal ($\mathbb{A} = \mathbb{B}$) when an element is in $\mathbb{A}$ if and only if it is also in $\mathbb{B}$, which can be denoted as $\forall x : x \in \mathbb{A} \Leftrightarrow x \in \mathbb{B}$. A set $\mathbb{A}$ is called a subset of a set $\mathbb{B}$ ($\mathbb{A} \subseteq \mathbb{B}$), or $\mathbb{B}$ is a superset of $\mathbb{A}$ ($\mathbb{B} \supseteq \mathbb{A}$), when if an element is in $\mathbb{A}$ then it is also in $\mathbb{B}$, denoted as $\forall x : x \in \mathbb{A} \Rightarrow x \in \mathbb{B}$. If $\mathbb{A} \subseteq \mathbb{B}$ but $\mathbb{A} \neq \mathbb{B}$, i.e. there is at least one extra element in $\mathbb{B}$, then $\mathbb{A}$ is a proper subset of $\mathbb{B}$ ($\mathbb{A} \subset \mathbb{B}$), or alternatively $\mathbb{B}$ is a proper superset of $\mathbb{A}$ ($\mathbb{B} \supset \mathbb{A}$).

It is also possible to use propositional logic to create new sets by defining certain operations between sets, in particular *Boolean set operations*, consisting of intersection, union, difference and complement[12]. The intersection of the sets $\mathbb{A}$ and $\mathbb{B}$, denoted as $\mathbb{A} \cap \mathbb{B}$, consists of all the elements that are both in $\mathbb{A}$ and in $\mathbb{B}$, i.e. $\mathbb{A} \cap \mathbb{B} = \{x \mid x \in \mathbb{A} \wedge x \in \mathbb{B}\}$. The union of the sets $\mathbb{A}$ and $\mathbb{B}$, denoted as $\mathbb{A} \cup \mathbb{B}$, consists of all the elements that are either in $\mathbb{A}$ or in $\mathbb{B}$, i.e. $\mathbb{A} \cup \mathbb{B} = \{x \mid x \in \mathbb{A} \vee x \in \mathbb{B}\}$. The difference between sets $\mathbb{A}$ and $\mathbb{B}$, denoted as $\mathbb{A} - \mathbb{B}$, consists of all the elements that are in $\mathbb{A}$ but not in $\mathbb{B}$, i.e. $\mathbb{A} - \mathbb{B} = \{x \mid x \in \mathbb{A} \wedge x \notin \mathbb{B}\}$. The complement of a set $\mathbb{A}$, denoted as

12: These are the most commonly described basic operations. However, it is possible to define other operations that are equally useful as a base. Either can be used to form other operations by composition.

$\neg\mathbb{A}$, consists of all the elements that are in the universe set but are not in $\mathbb{A}$, i.e. $\neg\mathbb{A} = \{x \mid x \in \mathbb{U} \wedge x \notin \mathbb{A}\}$.

Apart from sets, it is also possible to consider *tuples* of elements, which unlike sets are sequences of ordered elements. A tuple containing exactly two elements is known as a *pair*, one containing three elements is a *treble* and one containing n elements is an n-tuple. Tuples are denoted using parenthesis, i.e. (and).

A common operation that generates tuples is the Cartesian product. The Cartesian product of sets $\mathbb{A}$ and $\mathbb{B}$, denoted as $\mathbb{A} \times \mathbb{B}$, is defined as $\{(a, b) \mid a \in \mathbb{A} \wedge b \in \mathbb{B}\}$. In other words, it is a set of pairs, where the first element of a pair is an element of $\mathbb{A}$ and the second element of the pair is an element of $\mathbb{B}$. This can be generalised to more than two sets, such that the n-fold Cartesian product of n sets is an n-tuple. The n-fold Cartesian product of a set $\mathbb{A}$ with itself, i.e. $\mathbb{A} \times \mathbb{A} \times \ldots \mathbb{A}$, is denoted as $\mathbb{A}^n$.

2.2 Geometry

Geometry is the branch of mathematics concerned with the position of objects in space, a topic that was already formalised by the ancient Greek mathematician Euclid in his textbook *the Elements* around 300 BCE [Fitzpatrick, 2008]. Euclidean geometry consists of a small set of geometric axioms considered to be intuitively obvious, such as the fact that it is possible to draw exactly one line that passes through two points (Figure 2.1), as well as a long series of postulates derived from these and which describe more complex constructions[13].

However, even as Euclidean geometry has the notions of relative distances, angles and areas, objects in Euclidean geometry do not have an absolute position in space. Analytic or Cartesian geometry, developed by Descartes [1637] and de Fermat [1679], significantly changed this by introducing the concept of coordinates. A coordinate system makes it possible to uniquely describe the absolute location of a point as a tuple of real numbers. In particular, the Cartesian coordinate system uses a tuple of perpendicular directed lines as axes, with a positive direction and a negative direction, all of which intersect at a common point known as the origin. A point's coordinates in the system are then given by signed distances to the respective axes[14].

n-dimensional Euclidean space, which can be described by the set of points $\mathbb{R}^n$, has n perpendicular axes intersecting at the origin O, defined by the n-tuple $(0, 0, \ldots, 0)$, and a point p in nD space is thus described by an n-tuple $(p_1, p_2, \ldots, p_n)$, where p_i is the signed distance to the i-th axis. For example, as shown in Figure 2.2, three-dimensional Euclidean space ($\mathbb{R}^3$), has three axes, usually named X, Y and Z, such

13: Non-Euclidean geometry does away with some of these axioms while remaining self-consistent [Bolyai, 1832; Lobachevsky, 1840]. However, it is much less relevant in the context of spatial modelling.

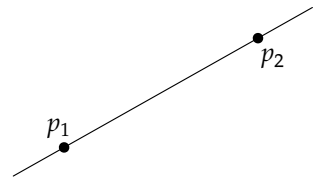


Figure 2.1: There is exactly one line that passes through any pair of points.

14: A more rigorous and correct explanation would be based on a set of linearly independent vectors, but this creates a recursive definition.

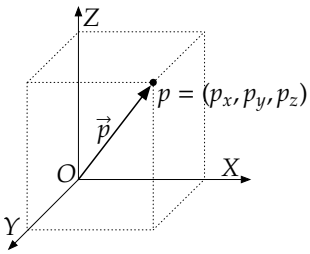


Figure 2.2: A point p in 3D described by a treble (p_x, p_y, p_z)

that a given point p in 3D can be described by a treble (p_x, p_y, p_z), where p_x is the signed distance to the X axis, p_y to the Y axis and p_z to the Z axis.

A point $a = (a_1, a_2, \dots, a_n)$ can also be used to define a vector $\vec{a}$, which goes from the origin to a. The norm, or magnitude of $\vec{a}$, denoted as $\|\vec{a}\|$, gives the length of the line segment between a and the origin and is computed as:

$$\|\vec{a}\| = \sqrt{{a_1}^2 + {a_2}^2 + \cdots + {a_n}^2}$$

This analytic description of objects also enables using algebra to compute properties, such as the Euclidean distance between two points $a = (a_1, a_2, \dots, a_n)$ and $b = (b_1, b_2, \dots, b_n)$, also known as the Euclidean metric. This is given by:

$$\text{distance}(a, b) = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2 + \cdots + (a_n - b_n)^2}$$

Some other objects can be described as a linear combination of linearly independent points (i.e. two different points, three non-collinear points, four non-coplanar points, etc.). Considering the points $p_1, p_2, \dots, p_n$, a linear combination of them takes the form $a_1p_1 + a_2p_2 + \cdots + a_np_n$, where $\sum_{i=1}^{n} a_i = 1$. For every point p_i, a_i is thus a scalar coefficient that determines its *weight*.

If negative weights are allowed, the linear combination of $n + 1$ linearly independent points forms an n-dimensional unbounded linear object, e.g. a line using two points or a plane using three points. All of these points lie exactly on the object. When the weights are instead restricted to the interval $[0, 1]$, the linear combination of $n + 1$ linearly independent points forms an n-dimensional simplex (called an n-simplex)—a convex shape with $n+1$ vertices. A *0-simplex* is thus a point, a *1-simplex* is a line segment, a *2-simplex* is a triangle, a *3-simplex* is a tetrahedron, and so on.

Other, more complex objects can be described using equations, which describe particular subsets of $\mathbb{R}^n$. A hyperplane in $\mathbb{R}^n$, i.e. a space of dimension $\mathbb{R}^{n-1}$ in $\mathbb{R}^n$, can be described by a linear equation of the form $a_1x_1 + a_2x_2 + \cdots + a_nx_n = b$, where $a_1, a_2, \dots, a_n$ are the coefficients of the linear equation. Apart from the points exactly on the hyperplane, as shown in Figure 2.3, such a hyperplane separates $\mathbb{R}^n$ into two parts on either side of it. These are known as open half-spaces and can be obtained by transforming the linear equation into the strict linear inequalities: $a_1x_1 + a_2x_2 + \cdots + a_nx_n < b$ for the half-space below the hyperplane and $a_1x_1 + a_2x_2 + \cdots + a_nx_n > b$ for the one above it. If non-strict linear inequalities are used instead (i.e. using $\leq$ and $\geq$ instead of $<$ and $>$), these *closed half-spaces* also contain the points on the hyperplane.

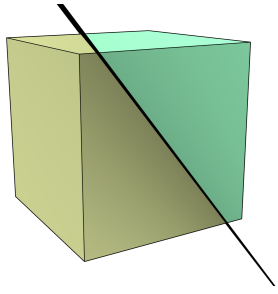
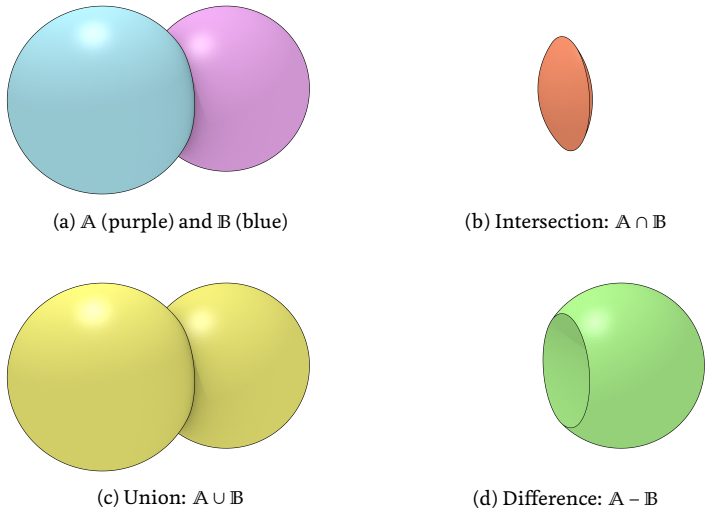


Figure 2.3: A plane separates $\mathbb{R}^3$ into two parts on either side of it.

Considering that a point can be described as a tuple of its coordinates, a hyperplane as a tuple of its coefficients, and similar constructions are possible for many other objects (e.g. a sphere based on a centre point and radius), it becomes possible to have a *computer representation* of these objects simply by storing tuples of numbers in a data structure[15]. Moreover, it becomes possible to use them as a basis to describe other, more complex objects by using them as building blocks, either directly or using some of the topological concepts described in the next section, e.g. them forming the boundary of another object.

15: This hides the fact that using real numbers in a computer is very difficult in practice, thus floating-point approximations are generally used instead [Goldberg, 1991]. The main consequences of this in spatial modelling are discussed in §A.2.

Since analytic geometry allows the description of objects as sets of points in $\mathbb{R}^n$, as shown in Figure 2.4, it is also possible to define objects based on Boolean set operations of their point sets.



(a) $\mathbb{A}$ (purple) and $\mathbb{B}$ (blue)

(b) Intersection: $\mathbb{A} \cap \mathbb{B}$

(c) Union: $\mathbb{A} \cup \mathbb{B}$

(d) Difference: $\mathbb{A} - \mathbb{B}$

Figure 2.4: Based on two balls $\mathbb{A}$ and $\mathbb{B}$, other objects that can be defined using Boolean set operations.

2.3 Topology

Topology is the mathematical study of the shape of objects, growing out of the analysis of certain problems in geometry, such as the boundaries of objects and the different possible notions of connectedness. In particular, it studies the properties of certain objects that are preserved under so-called *topological transformations* or *continuous maps*, which include stretching and bending but exclude tearing or gluing.

There are two branches of topology that are most relevant in the context of spatial modelling, *point-set topology* and *algebraic topol-*

ogy, respectively presented in §2.3.1 and §2.3.2. Point-set topology describes space using concepts derived mainly from set theory, representing objects as continuous sets of points. The properties of these sets and the relationships between multiple sets can then be analysed and described. Algebraic topology adds concepts from abstract algebra as well, representing objects as structured sets of discrete elements, such as points, edges and faces. As these elements and the relationships between them are both discrete, it is possible to use a wide variety of algorithmic methods on them, including graph theory, combinatorics, algorithmic algebra, and computational geometry and topology.

2.3.1 Point-set topology

Point-set topology, also known as general topology, describes objects as sets of points satisfying certain conditions, such as those in the construction in Figure 2.5. These objects can then be analysed based on the properties of the sets that describe them, such as whether a set is bounded or unbounded, has a certain number of holes, or is orientable or unorientable. When multiple objects are present, the relationships between their corresponding sets can be analysed as well, such as whether they are touching or overlapping, or whether it is possible to define a function that maps between these sets.

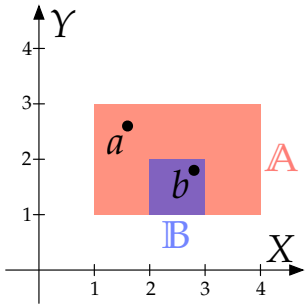


Figure 2.5: The rectangle $\mathbb{A}$ is represented by the set of points where $1 \leq x \leq 4$ and $1 \leq y \leq 3$. In more compact (set builder) notation, $\mathbb{A} = \{(x,y) | x \in [1,4] \wedge y \in [1,3]\}$. For the other objects, rectangle $\mathbb{B} = \{(x,y) | x \in [2,3] \wedge y \in [1,2]\}$, point $a \in \mathbb{A}$, point $b \in \mathbb{B}$ and point $b \in \mathbb{A}$. $\mathbb{B}$ is a subset of $\mathbb{A}$ (i.e. $\mathbb{B} \subset \mathbb{A}$).

Point-set topology works with *topological spaces*, a much more general notion than that of Euclidean space. This allows the description of different types of space with different properties. A topological space consists of a set of points and a *topology* on them satisfying a series of axioms. Edelsbrunner [2014] provides the following simple formulation. Given a set of points $\mathbb{X}$, a topology of $\mathbb{X}$ is a collection of subsets, which are called open sets[16], such that:

- $\mathbb{X}$ is open and the empty set is open;
- the intersection of any two open sets is open;
- the union of any family of open sets is open.

While the definition of an open set for general topological spaces is rather complex, in the context of spatial modelling we are generally interested in Euclidean space, which has a straightforward definition analogous to the concept of an open interval in 1D (Figure 2.6). A point set $\mathbb{S}$ in Euclidean space is open if, given any point $p \in \mathbb{S}$, there exists a real number $\epsilon > 0$ such that, given any point q whose Euclidean distance to p is smaller than ϵ, then $q \in \mathbb{S}$ as well. A point set is closed when the point set formed by its complement is open. Any point on an open interval fulfils these conditions, but the endpoints of a closed interval do not. Note that it is possible for a set to be open *and* closed (e.g. an interval containing only one of its endpoints).

16: Note that there are alternative definitions based on the concepts of closed sets or of neighbourhoods [Hausdorff, 1914]. For a simple definition using neighbourhoods in a GIS context see Worboys and Duckham [2004, §3.2.2].

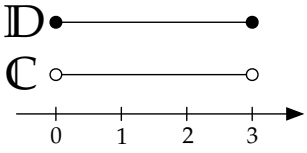


Figure 2.6: An open interval $\mathbb{C} = (0,3) = \{x | 0 < x < 3\}$ does not include its endpoints. By contrast, a closed interval $\mathbb{D} = [0,3] = \{x | 0 \leq x \leq 3\}$ includes its endpoints.

For example, in 2D, the plane together with the topology generated by the Euclidean metric is the topological space known as the *Euclidean topology of the plane*, which can be defined based on *open disks*, which are analogous to 1D open intervals. An open disk is the set of points closer to a point $p \in \mathbb{R}^2$ than a non-zero distance r, such as the unit open disk shown in Figure 2.7. It is easy to see that as these 2D disks do not contain their boundaries, the intersection of any two open disks and the union of any number of disks are both open.

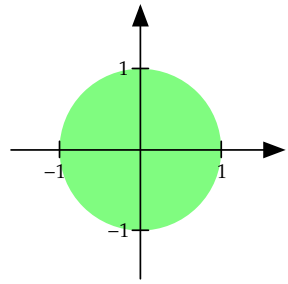


Figure 2.7: The unit open disk, i.e. an open disk of radius 1 centred at the origin can be defined as $\mathbb{D} = \left\{x \in \mathbb{R}^2 \middle| \|x\| < 1\right\}$.

Based on the concepts of open intervals in 1D, open disks in 2D, or open balls when talking about any dimension, it is possible to partition a Euclidean space into three parts: its *interior*, *boundary* and *exterior*. An example of these is shown in Figure 2.8. The *interior* of a point set $\mathbb{S}$ consists of all points where there exists an open ball centred at them such that all the points in the ball are in $\mathbb{S}$, the boundary of $\mathbb{S}$ consists of the points where all possible open balls centred at them have points in $\mathbb{S}$ and out of $\mathbb{S}$, and the exterior of $\mathbb{S}$ consists of all points where there exists an open ball centred at them such that all the points in the ball are out of $\mathbb{S}$. The *closure* of $\mathbb{S}$ is the union of its interior and its boundary. The *regularisation* of $\mathbb{S}$ is the closure of its interior and a point set is thus *regular* when it is equal to its regularisation.

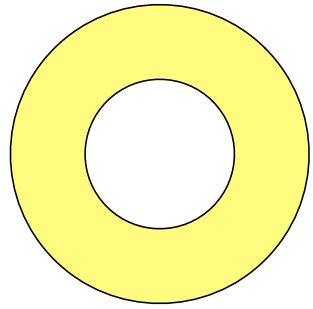
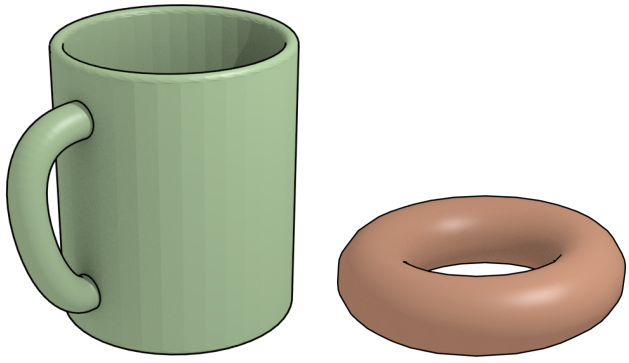


Figure 2.8: An annulus with boundary partitions the Euclidean plane into three parts: its interior (yellow), its boundary (black) and its exterior (the rest of this page). Note that none of these necessarily have to be connected.

Once objects are defined as sets of points, point-set topology works with functions that relate these sets to each other. A function from one point set to another is said to be *continuous* if the preimage (i.e. the inverse image) of every open set is open. If a function is continuous and its inverse function is also continuous, it is known as a *homeomorphism*. When such a function exists between two point sets, they are said to be *homeomorphic* or, more informally, *topologically equivalent*, such as the two objects shown in Figure 2.9.

Another important topological concept is that of a manifold. A manifold is a topological space that is homeomorphic to the Euclidean space of a certain dimension. Intuitively, this means that a



(a) A coffee mug (b) A donut

Figure 2.9: A coffee mug and a donut are homeomorphic[17]. Intuitively, this can be known as it is possible to deform one into the other. The mug was rendered from the model at http://www.thingiverse.com/thing:7953.

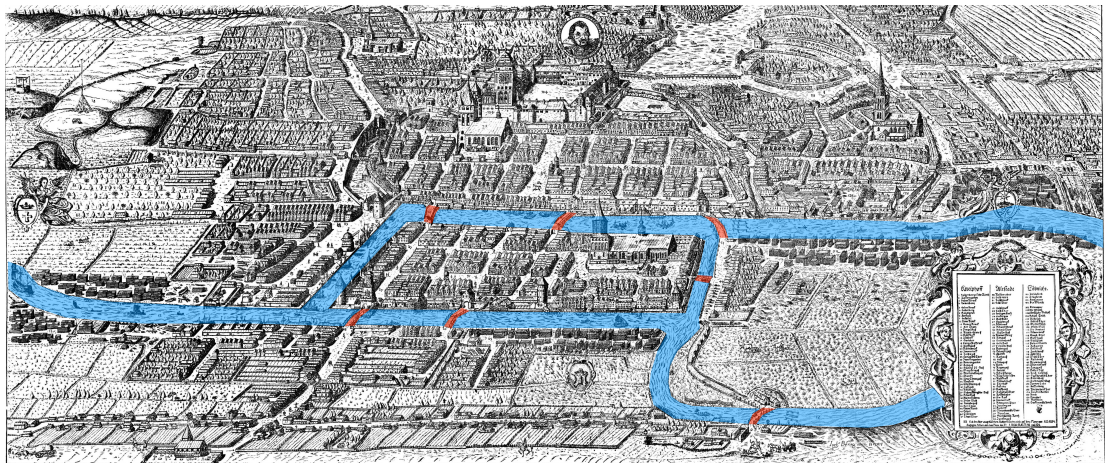


Figure 2.10: The problem of the Seven Bridges of Königsberg asks whether it is possible to find a route through the city that would cross each bridge (highlighted in red) exactly once. Euler [1741] proved that there is no such route in terms of a graph. Whenever one enters a piece of land by a bridge, one has to leave it by another bridge except at the beginning or end of the route. Thus, there must be an *even* number of bridges connected to all but (at most) two pieces of land. Since all pieces of land have an *odd* number of bridges, the problem has no solution. Based on an image from a 1613 engraving by Joachim Bering.

17: Related to the joke: 'A topologist is a mathematician who can't tell the difference between a coffee mug and a donut'.

manifold locally resembles Euclidean space, even if globally it does not. For example, a line and a circle are 1-manifolds, while a plane, a sphere and a torus are 2-manifolds. Generally, when the term manifold is used in GIS it refers to a 2-manifold.

2.3.2 Algebraic topology

Conceptually based on point-set topology, *algebraic topology*, also known as combinatorial topology, uses concepts from abstract algebra in order to analyse topological spaces. A famous early application involved the answer to the problem of the Seven Bridges of Königsberg by Euler [1741], explained in Figure 2.10. However, the real foundations of the field were set when many of its concepts were formalised in algebraic form by Poincaré [1895].

Algebraic topology works by relating topological spaces to groups with specific properties, often by creating combinatorial analogues of such spaces, from which their properties can be extracted using algebraic methods [Henle, 1994], which can be applied algorithmically. As it uses discrete structures rather than continuous point sets, it is often more suited to computer implementations of topological concepts than point-set topology [Worboys and Duckham, 2004, §3.3.5].

Two constructions of algebraic topology are widely used as the basis of GIS: *simplicial complexes* and *cell complexes*. An n-dimensional simplicial complex is a structure made of connected *simplices*, the

simplest objects that can be built in any dimension. As shown in Figure 2.11, an n-dimensional simplex (n-simplex) is a combinatorial primitive made from a set of $n+1$ vertices. Figure 2.12 shows a group of buildings represented as a 3D simplicial complex. A 0D simplicial complex consists of a set of discrete points (i.e. a point cloud) and a 1D simplicial complex is a plane graph. A 2D simplicial complex is known as a *triangulation* and a 3D simplicial complex as a *tetrahedralisation*.

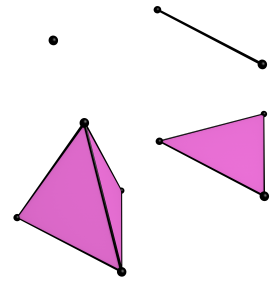


Figure 2.11: An n-dimensional simplex is a combinatorial primitive made from a set of $n+1$ vertices. A 0-simplex is thus a point, a 1-simplex is a line segment, a 2-simplex is a triangle, and a 3-simplex is a tetrahedron. Here they are shown as if embedded in $\mathbb{R}^3$.

A j-dimensional face (j-face) of an i-simplex, $j < i$, is a j-simplex made from a proper subset of its vertices. Sometimes the dimension of the face is omitted and it can be deduced from the context, but in GIS it generally refers to each of the $i+1$ $(i-1)$-faces of an i-simplex. In the context of an i-dimensional simplicial complex, a face refers to each of the i-simplices in the complex, such as the triangles in a triangular mesh.

More formally, a simplicial complex can be defined as a collection of simplices such that:

- every face of a simplex is also in the simplicial complex;
- the intersection of any two simplices is either empty or is a common face of both of them[18].

Based on the set of common vertices shared by two simplices, it is possible to define certain *topological relationships* between them. Two i-simplices are said to be adjacent if they have a common $(i-1)$-face. An i-simplex and a j-simplex, $i \neq j$, are said to be incident if either is a face of the other.

A cell complex is a structure made of connected *cells*, where an i-dimensional cell (i-cell) is an object homeomorphic to an open i-ball (i.e. point, open arc, open disk and open ball). 0-cells are known as vertices, 1-cells as edges, 2-cells as faces and 3-cells as volumes. Considering only linear geometries, 1-cells are thus line segments, 2-cells polygons and 3-cells polyhedra. Figure 2.13 shows a group of buildings represented as a 3D cell complex.

A j-dimensional face (j-face) of an i-cell is a j-cell, $j \leq i$, that lies on the boundary of the i-cell. A facet of an i-cell is an $(n-1)$-face of the cell. As in a simplicial complex, two i-cells are said to be adjacent if they have a common facet, and an i-cell and a j-cell, $i \neq j$, are said to be incident if either is a face of the other. In the context of an i-dimensional cell complex, a face refers to each of the i-cells in the complex, such as the polygons in a polygonal mesh.

More formally, a cell complex can be defined inductively as in Hatcher [2002]. An n-dimensional cell complex is built by starting from a set of isolated vertices, and $\forall 0 < i \leq n$ an i-cell is built by attaching itself to the $(i-1)$-faces (facets) on its boundary, these facets having been previously added to the complex. That is, an edge is built by linking the vertices on its boundary, a surface by linking

18: Note that this implies a definition where a simplex contains its boundary.

the edges on its boundary, a volume by linking the surfaces on its boundary, and so on. Like in a simplicial complex, a facet of a n-cell in an n-dimensional cell complex lies between it and an adjacent n-cell, unless it is on the boundary of the complex.

Apart from the concepts of adjacency, incidence and other relationships between between individual simplices and cells in a complex, it is also possible to define relations and transformations between entire simplicial/cell complexes. The *Poincaré duality* theorem [Poincaré, 1893][19] states that for every n-dimensional simplicial/-cell complex, there exists a *dual simplicial/cell complex* of the same dimension, where for every dimension i, the i-simplices/cells in the original complex are mapped one-to-one to $(n-i)$-simplices/cells in the dual complex. The *duality transformation* is an operation that creates the dual of a simplicial/cell complex. This can be seen as a generalisation of the concept of a dual graph in 2D, where a graph G has a dual G^*, such that vertices in G correspond to faces in G^*, edges in G correspond to edges in G^* and faces in G correspond to vertices in G^*.

19: It was only formulated as an observation without proof in Poincaré [1893]. Poincaré [1895] describes it in more detail but contains a flawed proof. Valid proofs would have to wait until Poincaré [1899, 1900].

For example, considering the Platonic solids in Figure 2.14, the tetrahedron is self-dual (i.e. it is dual to itself), the octahedron is dual to the cube (and vice versa), and the dodecahedron is dual to the icosahedron (and vice versa). This transformation can be seen by creating a new vertex at the centre point of the face of the Platonic solid, connecting these vertices when their dual faces are adjacent. The original vertices become faces whose number of vertices is equal to the number of originally incident faces.

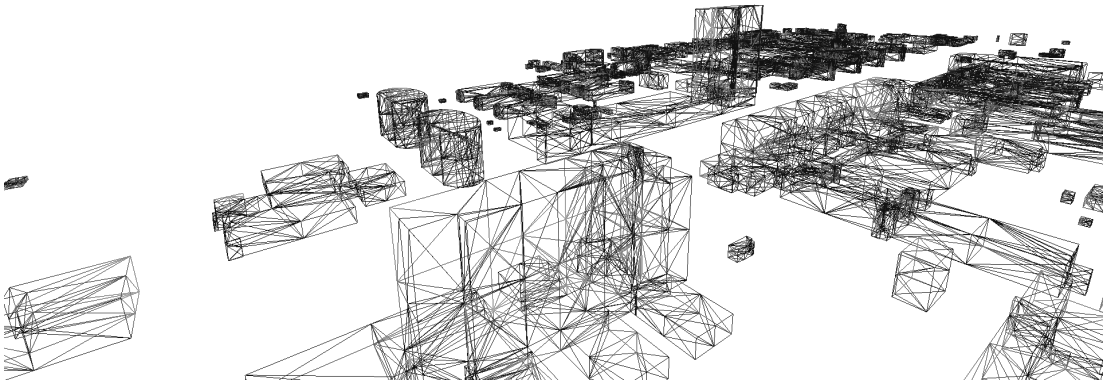


Figure 2.12: The buildings in the TU Delft campus are represented as a 3D simplicial complex, such that each separate building is a set of adjacent tetrahedra. Note that only the tetrahedra's edges are shown here.

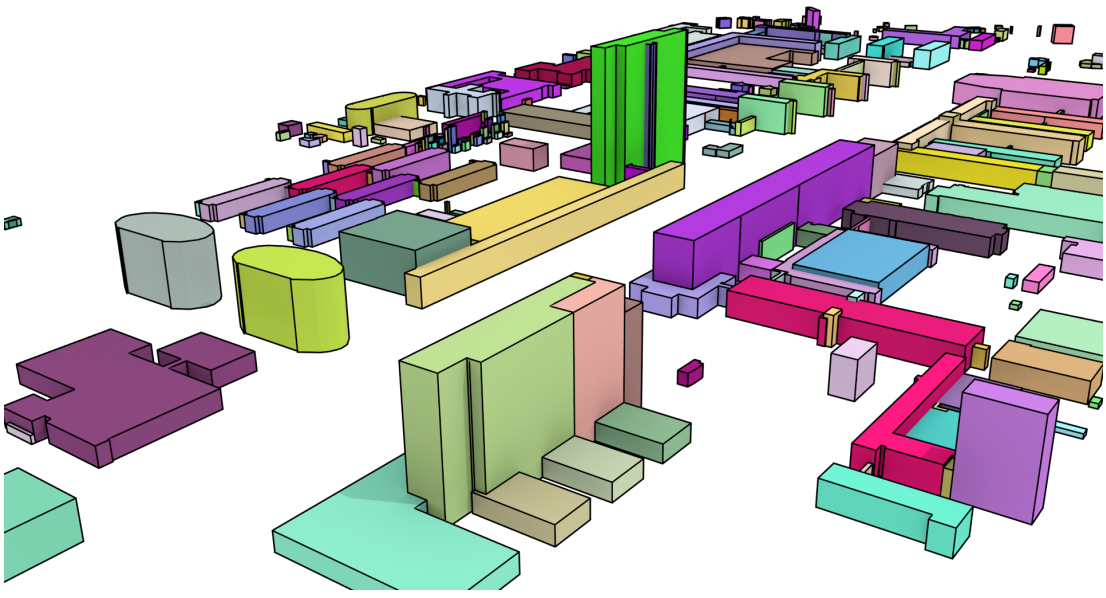


Figure 2.13: The buildings in the TU Delft campus are represented as a 3D cell complex. All the 2-cells of a 3-cell are shown in the same colour, the 1-cells are shown as black lines.

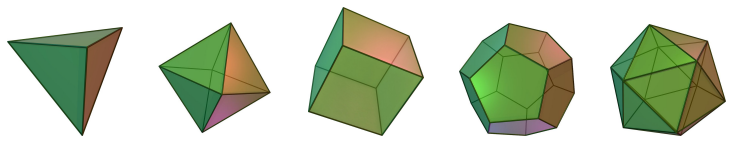


Figure 2.14: The Platonic solids are the five regular polyhedra that have regular polygonal faces: (a) tetrahedron, (b) octahedron, (c) cube or hexahedron, (d) dodecahedron, and (e) icosahedron. From Wikimedia Commons.

Modelling of 2D/3D space, time, scale and attributes

3

This chapter describes how space is usually modelled in 2D/3D GIS and related disciplines, as well as the non-spatial characteristics that are typically used together with it, such as time and scale. §3.1 explains what spatial modelling is and the abstraction process through which it is accomplished, discussing the main approaches that can be followed at three levels in this abstraction process, which take the form of data models, data structures and dual combinatorial/embedding structures. §3.2 describes how 2D and 3D space is currently modelled in a range of representative concrete cases, from the very minimal representations used in exchange file formats, in which simple structures and low storage requirements are preferred, up to those used internally in CAD-like software, in which the complex operations that are required necessitate the explicit storage of precise topological relationships.

Afterwards, this chapter contains sections on how specific non-spatial characteristics—that nevertheless have a strong link to space—are modelled: time in §3.3 and geographic scale or the level of detail of a model in §3.4. Finally, §3.5 concludes the chapter by pointing out some of the shortcomings of the aforementioned approaches to model 2D/3D space, time, scale and attributes. As will be seen in Chapter 4, many of these shortcomings can be resolved by using a higher-dimensional modelling approach.

3.1 Spatial data models and data structures

Spatial modelling aims at the creation of digital representations of real-world objects. However, real-world objects are complex and vaguely defined, while computers can only operate on their heavily abstracted and precisely defined digital counterparts. The spatial modelling process therefore uses a series of progressive abstractions, which start by interpreting reality as a set of high-level concrete entities that still resemble real-world objects and processes, and ultimately aims at creating abstract low-level representations that are close to what is actually stored in a computer, possibly using intermediate levels in the process.

This implies the existence of different levels of abstraction, and at each of these levels different approaches can be followed. While not all of these approaches are compatible with each other, the feasible combinations nevertheless result in a very large number of different computer representations.

As §3.1.1 explains, there are many classification schemes that attempt to group these approaches in a meaningful way according to various criteria, finding similarities and identifying schemes used by more than one representation. However, there is no agreement on the optimum number of levels of abstraction to be used, at which level some of these representational choices fit, nor a comprehensive clear-cut classification of them with no overlapping methods. In fact, as many of these choices only partially solve the difficulties of representing a digital object, a single computer system generally must resort to multiple methods. For instance, implicit (high-level) models are often used as a way to provide easy user interaction in software. However, these generally have to be 'evaluated' into another, more explicit (low-level) model in order for them to be visualised [Mäntylä, 1988] or to perform the type of computations expected in GIS, such as many spatial analysis and geometric operations.

Nevertheless, recognising this layered approach as the basis of the spatial modelling process, this section introduces the process by describing the main approaches that can be followed at three different levels: high-level *data models* using different paradigms to structure and discretise space in §3.1.2, how *data structures* implement these data models to model 2D/3D space in a form that is easy to implement in a computer in §3.1.3, and how *combinatorial and embedding structures* respectively model the topological and geometric information of some of these data structures in §3.1.4.

3.1.1 Classifications of spatial models

Considering that there is an incredible variety of methods that are used to create digital models of the world, but many of these share important parts of their respective approaches, there are various classifications that attempt to group them in a meaningful way.

Some of these are primarily based on human cognition of space. Couclelis [1992] distinguishes views based on empty space populated by discrete *objects* from those based on continuous space-filling *fields*, and Goodchild [1992] links objects and fields to specific computer models that are suitable for them. Freundschuh and Egenhofer [1997] makes distinctions based on the *scale* of such models compared to how people experience space, separating models of objects that are intuitively manipulatable by humans from those that are not.

Others are based on suitability for a particular application or use case. For example, Afyouni et al. [2012] does so for indoor navigation, Gold [2005] for multidimensional GIS, Domínguez et al. [2011] for building interiors and Pelekis et al. [2004] for spatio-temporal models.

However, for the purposes of this thesis, it is more interesting to consider the classifications that reflect a different discretisation of space or mathematical basis, as these will produce significantly different computer representations.

Regarding general classifications in the GIS domain, researchers and practitioners alike generally agree on the existence of two high-level *data models*—formalised structures describing the world as abstract primitives—, which are the *vector* and *raster* models. These are subdivided into a variety of low-level *data structures*, particular implementations of these data models, of which there are many [van Kreveld, 1997b]. This classification is not ideal as GIS data structures are usually specified in an ad hoc manner, sometimes accompanied with added indexing structures that do not neatly fit into the vector or raster approach (e.g. R-trees [Guttman, 1984] and k-d trees [Bentley, 1975]).

Apart from data models and data structures, some authors consider additional levels. Frank [1992] considers *spatial concepts* as well, the human notions used to understand space. Peuquet [1984] considers *reality* itself as a topmost level, including all aspects that may or may not be perceived by individuals, and a lowermost level with the *file structure*, describing how the information is actually represented in hardware. These are certainly interesting and worthy of study, but they are out of the scope of this thesis.

In the domain of geometric modelling and computer graphics, the schemes used are more varied, and thus the classifications used differ significantly. For 3D objects, there is usually a high-level classification with several *solid representation schemes* [Requicha, 1980; Hoffmann, 1992; Foley et al., 1995], general paradigms to model the world using a different approach to discretise and decompose objects. As these normally do not reach the level where it is possible to unambiguously devise a single computer implementation, these are more akin to the data models used in GIS. Mäntylä [1988] follows a similar classification, but groups these schemes into three paradigms: *decomposition models*, *constructive models* and *boundary models*, while recognising the existence of hybrid approaches combining multiple paradigms.

While some of these schemes use 3D primitives stored as simple parametric structures, many of them rely on the specification of 2D primitives using a separate mechanism, which can also be used directly for the description of 2D objects. These are usually described at a low level, generally consisting of 2D cell or simplicial complexes

respectively in the form of *polygon* or *triangle meshes*, which use one of a few *data structures* [Joy et al., 2003; Alumbaugh and Jiao, 2005; Blandford et al., 2005; de Floriani and Hui, 2005].

Some authors follow a more pragmatic approach, directly classifying data structures based on the class of objects that they are capable of representing. Čomić and de Floriani [2012] does so for cell complexes, de Floriani and Hui [2005] for simplicial complexes, Lienhardt [1991] for spaces in different dimensions, de Floriani et al. [2005] for models supporting multiple levels of detail, and Lienhardt et al. [2009] for models for simplicial, simploidal[21] and cell structures.

21: Cartesian products of simplices

Finally, at the lowest level, some authors distinguish between a *topological* or *combinatorial model* [Lienhardt, 1991], which describes the connectivity between a set of predefined elements, and a *geometric* or *embedding model*, which specifies the exact shape and position of individual elements [Mäntylä, 1988].

3.1.2 Data models or representation schemes

Data models are considered as different representation schemes operating at a high-level, differing mostly in their overall approach to decompose and discretise space into a set of elements with certain characteristics. Just as the concepts of geometry and topology seen in Chapter 2, these models are essentially *dimension-independent*—even if in practice they are generally implemented as dimension-specific data structures.

As argued by McKenzie et al. [2001], the ideal data model to be used depends on the application. An ideal data model is expected to be both simple and powerful, but unfortunately these properties tend to conflict. Powerful structures are built upon small and simple primitives of a fixed form with a fixed (or at least bounded) number of links to other related primitives, such as boxes or simplices, enabling them to have a simple but powerful algebra to manipulate them and to navigate between their primitives efficiently, e.g. the quad-edge in 2D [Guibas and Stolfi, 1985] or the facet-edge in 3D [Dobkin and Laszlo, 1987]. However, partitioning complex objects into these simple primitives can be difficult and result in a large number of primitives, requiring significant preprocessing, making them space-intensive and limiting the types of objects that might be stored. In this sense, different data models make different choices and necessarily involve a trade-off.

The main data models that are used in 2D and 3D are briefly described in the following sections, roughly in order of increasing explicitness. A short statement on how well they extend to higher dimensions is also included in each section. The most promising of

these models are further analysed in a higher-dimensional context in §4.3.

Primitive instancing

In *primitive instancing*, the system defines a library of primitives suitable for a particular application. Each of the primitives in the library represents a template for an object that can be parametrised in terms of high-level parameters, such as the number of teeth and the size of a gear or the diameter and thickness of a pipe [Foley et al., 1995]. In 3D city modelling, a common application involves the modelling of roofs using a library of basic roof shapes (e.g. flat, hipped and gabled) [Kada, 2007].

The fact that only a few parameters need to be defined makes it easy to create objects, that is, as long as the desired object resembles one of the templates in the library. Moreover, given these parameters and a primitive's position and orientation, a program can create the geometry of an object that matches those parameters as output using a more explicit representation. Such an object can then be used either directly or as a base for further operations.

Usually, the primitives used in primitive instancing are 3D volumetric objects, but they can actually be objects of any dimension. However, this approach is not very practical for higher-dimensional models of geographic information, as the objects modelled generally do not conform to a set of easily parametrisable primitives[22].

22: There are certainly exceptions, such as 3D+scale models as 4D models where a volume collapses to a point. However, the large number of primitives that would need to be defined for a system of this kind would make it impractical.

Sweep representations

Many objects can be modelled as the space occupied by another object while it is being translated and rotated along a path. Usually a 2D cross-section is used for this purpose, which is either translated or rotated, generating a volume in 3D space. Generalisations where the cross-section is deformed as well are sometimes considered [Blackmore et al., 1994]. When the path is linear and the cross-section is not rotated, it is called a *translational sweep*, or an *extrusion* as it resembles a physical extrusion process as used in manufacturing. When the cross-section does not move and it is merely rotated, it is called a *rotational sweep* or a *solid of revolution*. Among other applications, this representation is particularly convenient for applications that require volume computations as it matches well with calculus techniques to compute integrals, which can be easily solved numerically in a computer [Lee and Requicha, 1982].

Since the description of a 2D cross-section is easier than that of a full 3D volume, this significantly eases the process of modelling certain

types of objects [Weld and Leu, 1990]. In a higher-dimensional context, this property is later used to generate n-dimensional extruded models in Chapter 6. Generating more general higher-dimensional sweeps and revolutions is also possible using similar methods.

Constructive solid geometry

Constructive solid geometry (CSG) [Requicha and Voelcker, 1977, §12.3] represents objects as a tree of regularised Boolean set operations on a set of simple parametrisable primitives. In most instances, these are volumetric primitives such as balls, cylinders, parallelepipeds, cones and tori. Figure 3.1 shows an example of such an object. One of the strengths of CSG is that it allows many operations to be distributed among the primitives in a tree.

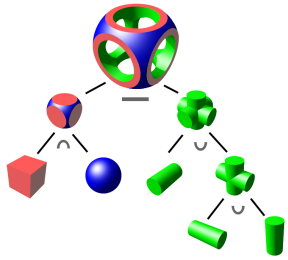


Figure 3.1: An object represented as a tree of Boolean set operations on a sphere, a cube and three cylinders. From Wikimedia Commons.

The regularisation step ensures that the resulting objects of any Boolean set operation are both volumetric and water-tight [Requicha and Tilove, 1978], making many computations easy and robust. It also makes it relatively easy to transform a CSG tree to a more explicit representation. CSG is therefore used in most CAD software and in many game engines.

While CSG extends naturally to higher dimensions, evaluating general CSG trees into more explicit representations is non-trivial in dimensions higher than three. A possible approach to do so would involve higher-dimensional Nef polyhedra, which are described afterwards.

Boolean set operations on half-spaces

There are various representations that store objects using an expression made of Boolean set operations on a set of (open or closed) half-spaces. Any convex object of any dimension can be represented as the intersection of a finite set of half-spaces, a common problem in both linear programming and computational geometry [Shamos and Hoey, 1976; Preparata and Muller, 1979]. As Figure 3.2 shows, if a polyhedron is first decomposed into convex parts [Chazelle and Dobkin, 1979; Bajaj and Dey, 1990], it is then possible to represent an object as a union of these convex parts, which are then represented as intersections of half-spaces.

Each of these half-spaces can then be stored easily, e.g. as a tuple containing the coefficients of a hyperplane[23] equation plus an up-/down direction [Naylor, 1990; Thompson, 2007]. The half-spaces for an object can be stored either all together or separate per convex part. Together, for instance, as a collection of plane equations that represent all the faces of all the convex parts, and each convex part is represented as a set of tuples of Boolean values, where each value

23: i.e. a line in 2D and a plane in 3D.

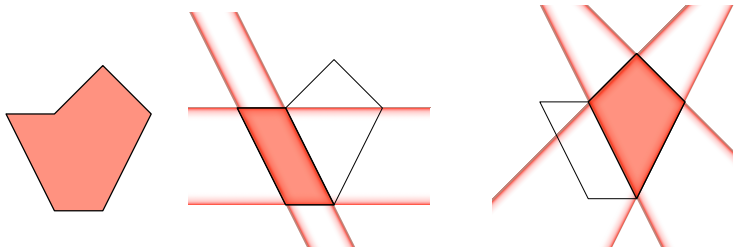


Figure 3.2: (a) A polygon can be represented as the union of its decomposition into the convex polygons (b) and (c). Each of these can then be represented as the intersection of four half-planes.

states whether the part is on the up or down side of a specific plane [Tammik, 2007].

Nef polyhedra

Nef polyhedra [Nef, 1978; Bieri and Nef, 1988] follow a similar approach to half-space models, but extend it with the concept of a local n-dimensional pyramid[24], which stores the neighbourhood information around every vertex. As Figure 3.3 shows, the local pyramid of a vertex thus contains the symbolic intersection of an infinitesimally small sphere (in 3D) or circle (in 2D) with the volumes, faces and edges incident to this vertex. An incident volume thus becomes a face, an incident face becomes an edge, and an incident edge becomes a vertex on the surface of the local pyramid sphere/circle, essentially lowering the dimension of every object by one. An 2D/3D object can be represented as a set of local pyramids, which can individually be stored using simpler 1D/2D data structures.

Nef polyhedra have two properties that make them one of the most promising representations for higher-dimensional models of geographic information: (i) the possibility to reduce the dimensionality of a representation by one, which can be applied recursively in order to model objects of any dimension, and (ii) the possibility to perform operations on the local pyramid level. The former is discussed in more detail in §4.3.3 and the latter in §5.1.2.

24: This is equivalent to a triangle in 2D and a polyhedral pyramid in 3D.

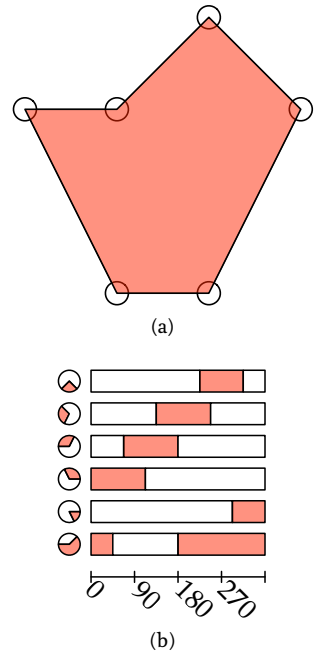


Figure 3.3: (a) A Nef polygon is represented by (b) a set of local pyramids (circles). At every local pyramid, the polygon (red) becomes an angular interval. Incident edges become points at the endpoints of these intervals.

Function representation

Function representation, commonly abbreviated as F-rep, involves representing objects as point sets using an arbitrary continuous function [Pasko et al., 1995], which is represented as an algebraic expression. When a given function $f(x_0, x_1, \ldots, x_n)$ is evaluated at a point with coordinates $(x_0, x_1, \ldots, x_n)$, it is possible to know if the point lies in the interior, boundary or exterior of an object. If it is

positive the point is in the interior of the object, when it is zero it is on the boundary of the object—in 2D known as an isoline and in 3D as an isosurface—, and when it is negative it is outside the object.

By varying the number of input parameters, which in most cases represent a point along an axis, function representations are applicable to any dimension. Moreover, unlike a half-space inequality, such functions do not need to be linear. As with half-space based representations, the spaces described by these functions can be combined using Boolean set operations, which is done on the basis of Rvachev functions (R-functions) [Rvachev, 1963]. However, evaluating these functions in higher dimensions can be complex, and obtaining a continuous function that adequately models a higher-dimensional object is also non-trivial.

Exhaustive enumeration

Starting from the description of a domain that covers the objects that need to be described, a simple deterministic rule is used in order to partition the domain directly into cells, which are of the same dimension as the domain (i.e. they are space-filling). In this manner, the decomposition rule is encoded into the model rather than in the data being represented, and so the cells do not need to store any geometric information of their own. Normally, the domain has a simple shape (e.g. a rectangle, hexagon or box) [ISO, 2007a, §6.8] and the rule partitions it into simple cells of the same size and shape (e.g. rectangles or boxes themselves), in which case it is called a *monohedral* tessellation [Boots, 1999]. If the cells are regular polygons or polyhedra, it is known as as *regular* tessellation.

In most cases, squares or rectangles (in 2D) and cubes or parallelepipeds (in 3D) are used for simplicity, respectively resulting in pixel and voxel structures. When other shapes are used, it is normally desirable to have cells that resemble equally-sized disks or balls as much as possible, something that is related to the sphere packing or kissing number problems in geometry [Conway and Sloane, 1992]. *Semi-regular* tessellations, which consist of more than one type of polygon/polyhedron are also possible but are not widely used, even in 2D. In 2D, the most commonly used other shapes are triangles and hexagons. In 3D, parallelepipeds or cubes are almost always used, but the face-centred cubic or hexagonal close-packed lattices are possibilities that resemble balls more closely. For instance, the new model cycle from the European Centre for Medium-Range Weather Forecasts[25] uses a cube-octahedron honeycomb that wraps around the Earth.

25: http://www.ecmwf.int/en/about/media-centre/news/2016/new-forecast-model-cycle-brings-highest-ever-resolution

These cells are then used to record whether the cell belongs to the interior of the object or not, or in the case of multiple objects, which

objects are present inside the cell. After such a tessellation is defined, a programatic order or path that traverses all cells (i.e. a space filling curve) is defined [Sagan, 1994], and the order of the cells along this curve is used to store them by *enumerating* for each cell whether a cell belongs to the interior of an object.

Exhaustive enumeration schemes are straightforward to extend to higher dimensions, as is further discussed in §4.3.1, with the caveat that their space complexity can increase exponentially on the dimension.

Hierarchical subdivisions using trees

A more general possibility to partition a domain involves using a hierarchical scheme where a space is recursively subdivided according to a particular criterion, resulting in a tree structure where a given node is a partition primitive that divides the space defined by its parent. The leaves of the tree are then used to record whether a cell belongs to the interior of an object or not. Such structures are generally known as space partitioning trees. In this manner, it is possible to efficiently partition spaces of any dimension, including areas and volumes where the objects' sizes are significantly different. However, the shape of a cell in the subdivision is only known after traversing a path of the tree all the way from its root to the leaf that corresponds to the cell.

In the simplest case, the space can be split evenly into halves along all axes at every partition node, resulting in a *quadtree* in 2D [Finkel and Bentley, 1974] or an *octree* in 3D [Meagher, 1980]. Bintrees [Samet and Tamminen, 1985] partition space alternating among dimensions rather than all at once, while k-d trees [Bentley, 1975] do so at an arbitrary point rather than at the midpoint of a space. Many other types of trees however exist. See [Manolopoulos et al., 2006].

As in exhaustive enumeration, this type of representation extends very naturally to higher dimensions (§4.3.2). However, the space used by hierarchical subdivisions can also increase exponentially on the dimension.

More general cell decompositions

Another possibility to represent the geometry of an object involves its decomposition into cells that are more complex than those in the exhaustive enumeration approach, such that they are still of the same dimension as the domain being partitioned but might have

different sizes and/or shapes. Nevertheless, they should still be describable as single computer primitives. This requires geometric information to be stored within each cell (e.g. the coordinates of one or more points).

While an arbitrary number of cell decomposition schemes are possible, the most interesting ones in a dimension-independent context are n-dimensional constrained triangulations and Voronoi diagrams, as these have a simple dimension-independent formulation. 2D constrained triangulations [Chew, 1989; Shewchuk, 1996a] and 3D constrained tetrahedralisations [Si and Gärtner, 2005] are very commonly used in GIS in order to subdivide objects respectively into triangles and tetrahedra, resulting in models that can be represented as simplicial complexes where every simplex can be respectively described by three and four linearly independent points at their vertices. Another common possibility uses Voronoi subdivision described by a single point per cell [Voronoi, 1908], something that is particularly suitable to field-like GIS data [Ledoux, 2006].

More information on how simplicial complexes are stored in 2D and 3D GIS is given in §3.1.3. §4.3.4 later describes how this extends readily to higher dimensions.

Incomplete (implicit) models

Incomplete models attempt to describe a space partition based on implicit rules applied to lower-dimensional primitives only, usually for simplicity purposes. As such, the actual cells forming the partition are created on the fly. Under some circumstances, such as when modelling non-manifolds or when the dimension of the primitives is significantly lower than that of the highest-dimensional cells, they cannot be unambiguously described.

The most typical problematic examples are wireframe models in 3D, which represent volumes using only vertices and edges. While in many cases it is possible to correctly interpret a volume from a wireframe model [Brewer III and Courter, 1986; Hanrahan, 1982], as shown in Figure 3.4 there are cases wireframe representations are ambiguous. They are therefore considered an incomplete representation and are generally not used in current systems.

Despite their drawbacks, incomplete implicit models are however frequently used in 3D GIS and when modelling time, resulting in ill-defined volumes or unclear equivalences of objects across time. Many of these disadvantages are alluded later on in order to justify using more explicit representations in higher dimensions.

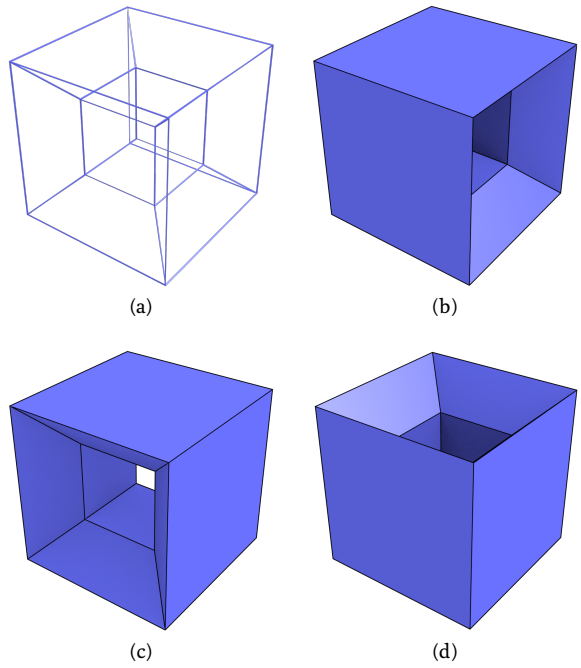


Figure 3.4: The wireframe model (a) can be interpreted as the three different volumes (b), (c) and (d).

Boundary representation

Considering the Jordan curve theorem [Jordan, 1887], which states that a closed curve separates the plane (i.e. $\mathbb{R}^2$) into two parts: an *interior* shape and an *exterior* shape, it is possible to use this curve in order to *implicitly but unambiguously* represent the interior (or exterior) shape. This theorem was generalised to higher dimensions as the Jordan-Brouwer theorem by Lebesgue [1911] and Brouwer [1911][26], implying that a volume in $\mathbb{R}^3$ can be represented by the 2D boundary surface that separates its interior from its exterior. This method, known as *boundary representation*, *B-rep* or *surface modelling*, reduces the complexity of the problem by making it possible to use 2D data models/structures to store 3D objects, which are significantly simpler. Moreover, by decomposing the 2D surface into simple cells, e.g. triangles or polygons, it becomes possible to represent the surface using one of the many data structures capable of representing 2D cell complexes, and if these cells support curved domains (e.g. NURBS), these can be used to represent complex surfaces as well.

26: This is somewhat contentious. See van Dalen [2013, Ch. 5].

While certain problems become more difficult by the use of boundary representation, such as the fact that it is easy to create invalid objects, some others can be performed directly on the 2D cells. For

instance, visibility queries or determining if a point is in the interior of an object can be performed by shooting rays.

As will be discussed in §3.1.3 in 2D/3D and §4.3.5 in nD, cell complexes are effectively represented using boundary representation schemes. §4.3.6 instead shows how an n-dimensional cell complex can be represented using a data structure based on a simplicial complex. The latter approach arguably combines the benefits of both simplicial complexes and cell complexes, and was thus used for most of the methods developed in the latter chapters of this thesis.

3.1.3 Data structures

The terms 'data model' and 'data structure' are often used interchangeably, but within this thesis it is useful to make a distinction between a data model, i.e. a particular *dimension-independent* discretisation of space into abstract primitives of a certain form, from the data structure(s) used to implement it, i.e. their representation in a computer-compatible form [Frank, 1992], which is often *dimension-specific*. A data model broadly defines the shape of the primitives and relationships that exist between them. In contrast, a data structure defines a *concrete representation* of these primitives, deciding to *explicitly* store some of these relations and omit others. Within this thesis, we refer to the omitted relations as being described *implicitly*. Note that even implicit relations can usually be computed from a model, but with an added computational cost.

This distinction is blurred because many data models have very straightforward, natural representations[27]. As shown in §2.2, many geometric primitives can be stored on the basis of a set of points or the coefficients of an equation—all of which can be stored as tuples of numbers. This is also true for the subdivision primitives in cell decomposition schemes and in most cases for the definition of the domain in exhaustive enumeration as well. Meanwhile, primitive instancing schemes and constructive solid geometry models can be stored as tuples of parameters. For instance, one of these parameters is the shape that is used and the other parameters store its parametrisable characteristics.

27: A clear exception are the ad hoc data structures often used in GIS, which usually define a set of fields that is different per dimension, resulting in awkward definitions at the data model level.

However, some other data models work with primitives that are more abstract or complex to represent in a computer. For instance, the schemes used by sweep representations, Nef polyhedra and boundary representations all reduce the dimensionality of the objects to be modelled, but ultimately need either to be applied recursively or to be combined with a different specific method.

This section lists the data structures for subdivisions of 2D and 3D space that have a possible natural extension to higher dimensions,

grouped based on the class of objects that they are capable of representing efficiently, i.e. without requiring brute-force searches or external data structures to navigate between the elements.

2D and 3D simplicial complexes

Simplicial complexes can always be respectively stored using the data structures for cell complexes, which are presented in the following sections. In some cases this is necessary, such as when a simplicial complex is first being created by triangulating a cell complex. However, the simplices in a simplicial complex have a fixed shape, which allows for the usage of more efficient data structures.

An n-dimensional simplex has a fixed number of faces of every dimension up to n, which are given by Pascal's triangle as shown in Figure 3.5. This means that in a 2D simplicial complex, i.e. a triangulation[28], every triangle has three vertices and three edges, and it is adjacent to at most three other triangles. Similarly, in a 3D simplicial complex, i.e. a tetrahedralisation, every tetrahedron has four vertices, six edges and four faces, and it is adjacent to at most four other tetrahedra.

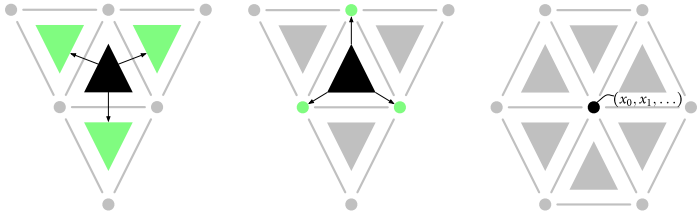
Because of this, 2D simplicial complexes are usually stored using triangle-based data structures and 3D simplicial complexes using tetrahedron-based data structures. These respectively use triangles/tetrahedra as primitives, which can store attributes for themselves (e.g. whether it is part of a given object) and for (some of) their faces, and are linked to their adjacent triangles/tetrahedra. As shown in Figure 3.6, a minimal representation of this type would only consist of triangles/tetrahedra linked to their vertices, either as tuples of coordinates or as pointers to a member of a list of separate point primitives.

Another possibility, also enabled by the fixed form of the simplices in a simplicial complex, is to use compression schemes in order to store them more efficiently. For example, Blandford et al. [2005] show how several of these schemes can be used: storing only some of the vertices in each simplex, difference coding to minimise the

1
1 1
1 2 1
1 3 3 1
1 4 6 4 1

Figure 3.5: The $(i-2)$-th row of Pascal's triangle, which is obtained by adding the numbers in the row above, gives the number of faces of every dimension in an i-simplex. For instance, a tetrahedron (3-simplex) has 4 vertices, 6 edges, 4 faces and 1 volume.

28: even if it is embedded in a dimension higher than two, e.g. the triangulated surfaces embedded in 3D (TINs) that are common in GIS



(a) Adjacent triangles (b) Vertices of a triangle (c) Coordinates of a vertex

Figure 3.6: A simple triangle-based data structure consists of a set of triangles, each of which points to its three adjacent triangles and its three incident vertices. This is accompanied by an indexed list of vertices and their coordinates.

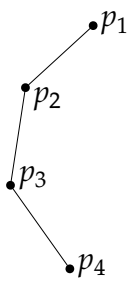


Figure 3.7: A *polygonal curve*, also known as a *polyline*, is a curve made from contiguous line segments. It is commonly represented as a sequence of points connected by implicit line segments, in this case (p_1, p_2, p_3, p_4).

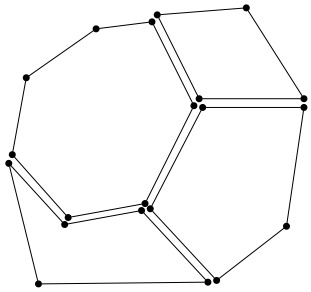


Figure 3.8: In the polygon model, every polygon is represented separately as a list of vertices. Note that every vertex is thus represented once for each polygon where it is used.

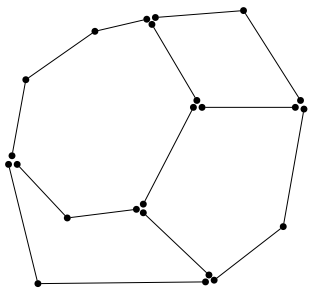


Figure 3.9: In the spaghetti model, common boundaries are identified and represented only once. However, the polygons are only implicitly described.

size of the IDs that are stored, and using collections of adjacent simplices (e.g. the star of a vertex) as primitives. It is also possible to use progressive representations that approximate the simplicial complex incrementally [Popović and Hoppe, 1997], or to compress it as a sequence of operations from which the original structure can be incrementally deduced [Rossignac and Szymczak, 1999].

2D cell complexes

Considering the Jordan curve theorem [Jordan, 1887] and the principles of boundary representation, as presented in §3.1.2, nD cells can be represented based on their $(n-1)$D boundary. 2D cells can be thus be represented based on their 1D boundary. Since linear 1D objects, such as the polygonal curve (polyline) in Figure 3.7, are notably easy to represent, this is the approach that is favoured by most data structures for 2D cell complexes.

The simplest data structures for 2D cell complexes store every 2-cell independently as a list of vertices, as is shown in Figure 3.8. This might be done by storing its coordinates directly in the list, sometimes known as the polygon model, or by keeping an external list of points and referring to point IDs in it, sometimes known as the point dictionary representation [Peucker and Chrisman, 1975].

Another simple approach consists of identifying the polygonal curves that form the common boundary that lies between two polygons (or one polygon and the exterior), splitting them at the vertices that are incident to three or more polygons (or two polygons and the exterior) and storing them *only once*, as is shown in Figure 3.9. This means that most points are only stored once as well (except for those at the beginning or end of these polygonal curves). In GIS, this is sometimes known as the *spaghetti model* and the polygonal curves are known as *chains*[29]. Unlike 3D wireframe models, these polygonal curves are sufficient to identify the polygons in a 2D cell complex[30]—a problem known as polygonisation in GIS, which is related to the computation of all intersections in arrangements of lines in computational geometry [de Berg et al., 2008, Ch. 8]. However, additional processing is required to do so as the polygons are not stored directly.

Most other data structures take into account the fact that every edge or chain in a 2D cell complex lies between two cells, and so it is very convenient to use these edges to store information about the two cells that are incident to them. For instance, the Node-arc-area (NAA) or POLYVRT (Polygon Converter) data structure [Peucker and Chrisman, 1975] also uses chains, but stores for each chain the polygons on the left and right side according to the order in which its vertices are stored.

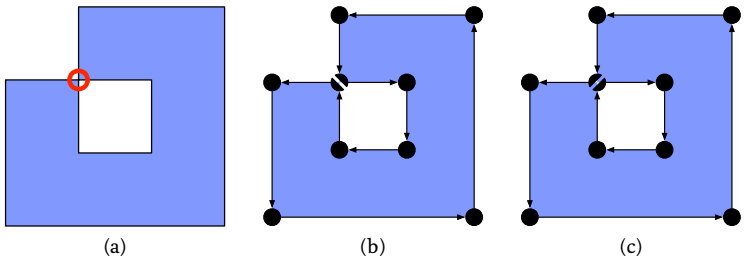


Figure 3.10: (a) A polygon is non-2-manifold as the space around the vertex (highlighted in a red circle) is not homeomorphic to the plane (i.e. $\mathbb{R}^2$). Its linear boundary is non-1-manifold as the space around the vertex is not homeomorphic to the line (i.e. $\mathbb{R}$). (b) & (c) However, the polygon can still be represented using a loop of oriented half-edges by having a duplicate vertex at that location (shown as two half balls), but there are two ways in which this can be done. Note that these are not equally desirable as (c) results in a disconnected graph.

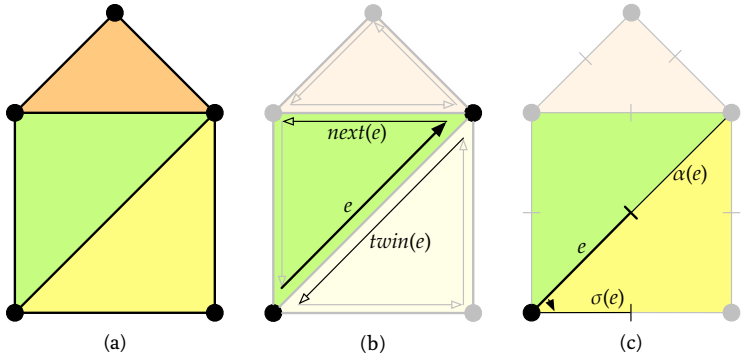


Figure 3.11: (a) A 2D cell complex of three polygons is represented using (b) the DCEL and (c) a 2D combinatorial map. In the DCEL, a half-edge e is meaningfully related to two vertices (the origin and the destination) and one face, and is linked to its next half-edge (on the same face) and its twin half-edge (on the adjacent face). In a 2D combinatorial map, a half-edge e is meaningfully related to one vertex and two faces (on either side), and is linked to the half-edge on the opposite side of the same edge $\alpha(e)$ and the half-edge on the same vertex but on the next edge (as given by a rotation direction).

There are several data structures based on half-edges, oriented edges or vertex-edge pairs. All of these are functionality equivalent and are able to store a 2D cell complex or its dual. However, since cells in a cell complex are supposed to be manifold, special care is usually necessary to represent cells with a non-manifold domain. An example of this shown in Figure 3.10.

Among the half-edge based data structures, the winged-edge data structure [Baumgart, 1975] considers edges as the main primitive, gives them an orientation, and maintains two records for the left and right polygons, as well as four records for the previous and next oriented edges along the boundaries of both of these polygons. As shown in Figure 3.11, the doubly-connected edge list (DCEL) [Muller and Preparata, 1978] and 2D combinatorial maps [Edmonds, 1960], achieve a similar result but have a more elegant and usually more efficient approach, dividing edges into half-edges[31], which makes it easier to follow the cycles representing polygons.

The quad-edge data structure [Guibas and Stolfi, 1985] attempts to unify most of the half-edge-like structures by rigorously naming all the possible relationships between an oriented edge and other

29: Note however that the spaghetti model sometimes refers to other types of related models. The only fact that all references to a spaghetti model have in common is the fact that chains are stored individually.

30: As long as it strictly conforms to the definition of a cell complex, e.g. by having no overlapping polygons and ensuring that polygons are perfectly closed with no overshoots or undershoots.

31: In the case of the DCEL this is conceptually done lengthwise, graphically resulting in side-by-side edges with opposite orientations on either side of the edge. In the case of 2D combinatorial maps it is instead done at the halfway point, graphically resulting in end-to-end edges. However, these are functionally equivalent.

nearby oriented edges or vertices in its *quad-edge algebra*, which moreover includes the dual vertices and edges of the 2D cell complex. However, only a few of these relations need to be stored. For instance, Guibas and Stolfi [1985] proposes a structure of *quads*, which store for a given edge only the previous and next oriented edges for the polygons on both of its sides. As shown in Figure 3.12, these quads store all the necessary relationships to traverse the edges of a polygon in both clockwise and anticlockwise order, as well as to navigate between adjacent polygons that are incident to a given edge or vertex.

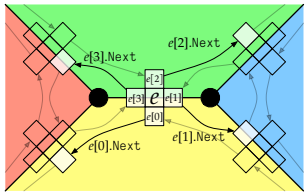


Figure 3.12: In the quad-edge data structure, an edge stores a *quad*, four records pointing to other quads corresponding to the previous and next oriented edges for the polygons on both of its sides.

3D cell complexes

While data structures for 2D cell complexes can rely on the fact that an edge is incident to at most two faces, this is not the case in a 3D cell complex. The data structures for 3D cell complexes are therefore significantly more complex, requiring additional information in order to efficiently traverse the many faces incident to an edge and between different volumes.

The facet-edge data structure [Dobkin and Laszlo, 1987], shown in Figure 3.13, considers an incident face-edge pair as a single primitive, which is known as a *facet-edge*. Groups of eight of these primitives are then stored as a single group, consisting of the four different orientation combinations of a facet-edge and the ones of its dual (consisting of the dual face and dual edge of the facet-edge).

The V-map data structure [Lienhardt, 1988], related to the notion of 2D combinatorial maps [Edmonds, 1960; Cori, 1975], splits edges into half-edges per edge, per face and per volume. These half-edges, called *threads*, result in faces that are modelled as cycles of half-edges, as is shown in Figure 3.14.

The radial edge data structure [Weiler, 1988] follows a similar approach, dividing faces into *face uses* per volume and representing these faces uses as cycles of *edge uses* where every edge use is linked to a *vertex use*. Based on this structure, it is possible to keeps a hierarchy of 3D objects, recursively composed of volumes, shells, faces, loops, edges and vertices.

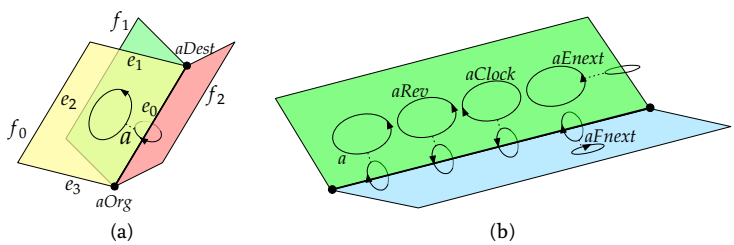
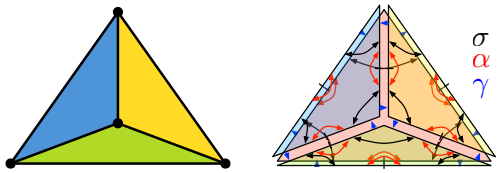


Figure 3.13: (a) The facet-edge data structure considers an incident face f_0 and edge e_0 as a single facet-edge primitive a, each with a predefined orientation. The orientation of the face of a defines the order along its incident edges e_0, e_1, e_2, e_3. The orientation of the edge of a defines the order along its incident faces f_0, f_1, f_2. (b) A set of operations on the facet-edge a is used to traverse the structure.



(a) Tetrahedral 3-cell (b) V-map

Figure 3.14: A tetrahedral 3-cell is represented as a V-map. Since a V-map considers half-edges to be distinct per face and per volume, every edge is here represented by four threads. Within a face, it is possible to switch between the threads of an edge (in French *arête*) with the operator α and between the threads of a vertex (in French *sommet*) with the operator σ. In addition, it is possible to navigate between the threads on different faces or volumes with the operator γ.

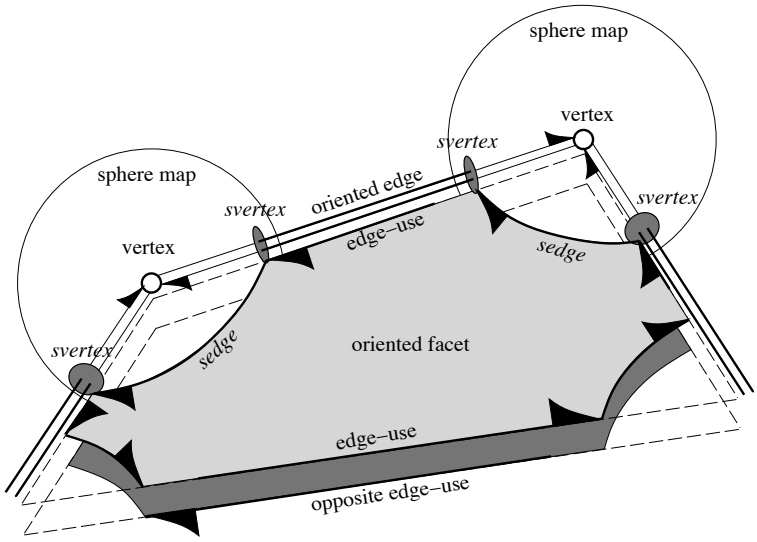


Figure 3.15: A selective Nef complex. The half-edge structure on $\mathbb{R}^3$ uses standard (oriented) faces, edges and vertices. The half-edge structure on the surfaces of the spheres uses *sfaces* (not shown here), *sedges* and *svertices*. Note how sedges are the intersections of a face with the spheres of its incident vertices, and svertices are the intersections of an edge with the spheres of its incident (origin and destination) vertices. From Hachenberger [2006].

Selective Nef complexes [Hachenberger, 2006], shown in Figure 3.15, implement 3D Nef polyhedra using a combination of two half-edge data structures, a common one that represents faces as cycles of *edge-uses*, and one that represents the local pyramids around every vertex as subdivisions on the surfaces of (infinitesimally small) spheres. This combination of data structures significantly reduces the complexity of computing certain operations on Nef polyhedra, including convex decompositions, Minkowski sums and Boolean set operations.

3.1.4 Combinatorial and embedding structures

In many of the data structures presented in §3.1.3, an additional distinction can be made between a *topological* or *combinatorial model* [Lienhardt, 1991], which was the focus of the previous section and

describes the connectivity between a set of predefined elements, and a *geometric* or *embedding model*, which specifies the exact shape and position of individual elements [Mäntylä, 1988]. This embedding model can be as simple as a point in $\mathbb{R}^2$ or $\mathbb{R}^3$ that is assigned to every vertex, resulting in linear geometries when the points for a face are collinear/coplanar.

However, it is often useful to have more powerful embedding information—allowing for more complex shapes (e.g. curved surfaces) or the storage of attributes. Normally, this embedding information is put into *embedding structures* that correspond to simplices or cells and can be used to store geometric and attribute information for themselves and their faces. When a simplex/cell has an explicit representation as a *single* data structure primitive, these structures are not necessary and the information for the simplex/cell can be put into the corresponding primitive. Otherwise, the embedding structures are separate data structures that are linked to related primitives, e.g. the two half-edges of an edge being linked to an embedding structure containing information about the edge, or a half-edge being linked to the structures for its incident faces.

Embedding structures are also used in order to keep semantic information. While the complex structures that are required for complex semantics are outside the scope of this thesis[32], embedding structures can store a cell's attributes as a tuple of fields for a cell. When more complex information is required, external data structures can be kept and linked to/from the embedding structure [Kuhn, 2005].

32: See Biljecki et al. [2014b] for a good description of these.

The geometric information that is necessary to store more complex shapes (e.g. curves and curved surfaces) can be similarly kept as a tuple of parameters. For instance, composite Bézier curves and surfaces [Bézier, 1977] can be stored using a cell complex where the vertices store *control points*. These are accompanied with some external knowledge such as the order of the curve/surface. Non-uniform rational basis splines (NURBS) curves and surfaces [Versprille, 1975] can also be stored as a cell complex of control points with knot vectors stored as lists in the embedding structures used for the edges.

3.2 Modelling of 2D and 3D space in practice

Based on the concepts of the different data models, data structures and combinatorial/embedding structures presented above, this section discusses how these are applied and combined in practice in various international standards, file formats and software. These are ordered from the simple geometric models that allow only for visualisation and simple calculations, up to the complex, more topological models that enable simulations and other complex computations.

3.2.1 Models used in visualisation, computer graphics and gaming

The simplest geometric models are those that are used solely for visualisation. Low-level application programming interfaces (APIs) for 3D graphics, e.g. OpenGL[33], Direct3D[34], Mantle[35], WebGL[36] and Vulkan[37] work with large numbers of simple geometric primitives with 3D coordinates, such as points, line segments and triangles, all of which can be used alone or as structured sequences (such as triangle strips, as shown in Figure 3.16). These can be passed directly, loaded into video RAM, and be processed and rendered in parallel by graphics hardware. Subproblems in this process are therefore offloaded onto techniques that can be easily parallelised and done in hardware. For example, the visibility computations involved when rendering a set of polygons in 3D can be handled using Z-buffers [Straßer, 1974].

33: https://www.opengl.org

34: a subset of DirectX: http://msdn.microsoft.com/directx

35: http://www.amd.com/mantle

36: https://www.khronos.org/webgl/

37: https://www.khronos.org/vulkan/

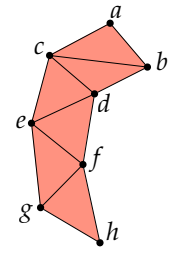


Figure 3.16: A triangle strip is easily defined as a list of vertices (a, b, c, d, e, f, g, h). Every triangle is formed by three consecutive vertices in the list.

3.2.2 Exchange file formats and standards in GIS and BIM

The file formats that are meant for the exchange of spatial objects in GIS—sometimes codified in (international) standards—generally opt for a minimal representation of the geometries involved. Polylines and polygons are thus stored as sequences of points connected by implicit line segments, as this is a more compact and intuitive representation than a set of (unordered) line segments. By contrast, volumes are represented as (unordered) sets of their bounding polygonal faces. As no topological relationships are recorded between these polygons, these types of representations are known as *non-topological models* in GIS.

The Simple Features Specification [OGC, 2011] is an international standard from both the Open Geospatial Consortium and the International Organization for Standardization (as ISO 19125 [ISO, 2006]). It is widely used as the basic geometric model in simple GIS formats and defines a variety of geometry classes, each of which is linked to a reference system. These are shown in Figure 3.17. A `LineString` (or `LinearRing`) is represented as a sequence of two or more `Points`. Apart from this relation and ignoring the generalisation relationships, the classes form a tree structure, with higher-dimensional classes being simple aggregations of lower-dimensional ones. A `Polygon` is made of at least one `LinearRing` (one outer and possibly multiple inner ones defining holes), and a `PolyhedralSurface` is made of a set of patches of `Polygons`. No relationships connect a class with itself or others of the same dimension.

GML [OGC, 2012], is the most widely used international standard to represent 2D and 3D geographic information. It has been further

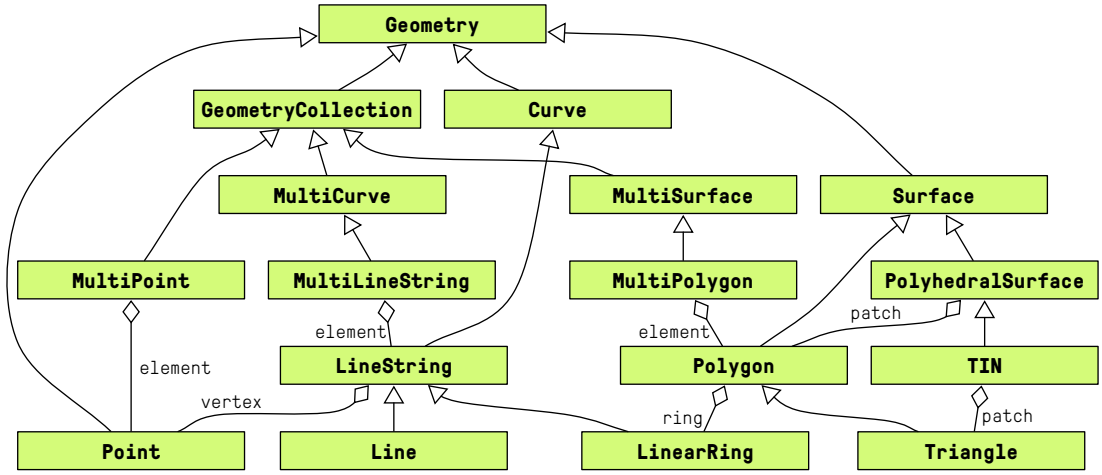


Figure 3.17: The geometry class hierarchy defined in the Simple Features Specification [OGC, 2011].

formalised as ISO 19136 [ISO, 2007b] and is the basis of other standards, such as CityGML [Gröger et al., 2012], which is used for the exchange of 3D city models. In terms of geometry, it implements a subset of the types in the ISO 19107 standard [ISO, 2005a], which are shown in Figure 3.18. It similarly considers `LineStrings` represented as sequences of at least two points, `LinearRings` as sequences of at least four points[38]. `Polygons` consist of one exterior and zero or more interior `Rings`, and various surface types are composed of different kinds of `SurfacePatches`.

The Industry Foundation Classes (IFC)[39] standard is an open data model used in the Building-information modelling (BIM) domain for the exchange of construction models, often including 3D models of buildings. It has also been adapted as the ISO 16739 international standard [ISO, 2013]. Its geometric aspects are however mostly defined or derived from a different standard—ISO 10303 [ISO, 2014]. Unlike the data models originating in GIS (e.g. Simple Features and CityGML), there is an important emphasis on the definition of a local coordinate system per object (as opposed to the national or regional coordinate systems used in GIS). This reflects the fact that in BIM every object is generally modelled independently before later being fitted together. Hierarchical descriptions of objects thus result in hierarchically applied coordinate system transformations. Boundary representation and tessellation geometries are supported as in other models, but IFC supports a much greater variety of geometry classes, which include implicit geometries based on half-space intersections, Constructive Solid Geometry (as a tree of Boolean set operations) and sweeps (based on a cross-section), such as the one shown in Figure 3.19.

38: There is no implicit connection between the first and last points in a CityGML polygon, so the first point needs to be repeated at the end.

39: http://www.buildingsmart-tech.org/specifications/ifc-releases

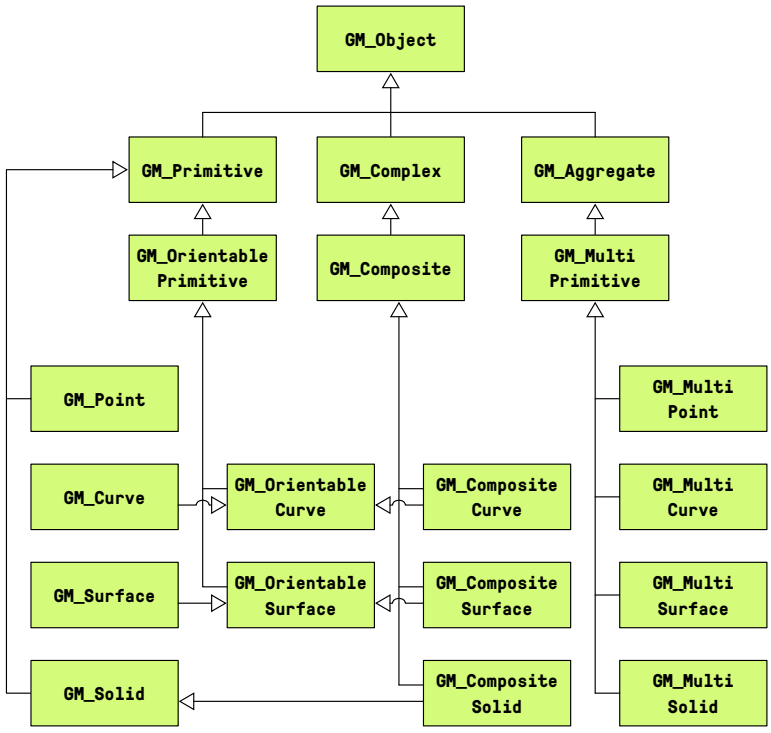


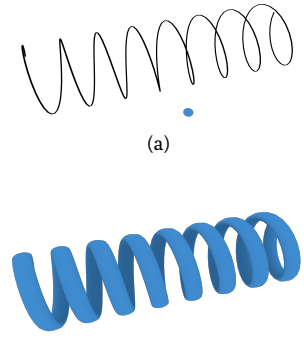
Figure 3.18: The geometric classes from the ISO 19107 standard [ISO, 2005a] that are implemented in the GML standard [OGC, 2007].

3.2.3 Models used in 2D and 3D GIS

When the first GIS were developed in 1960s and 1970s, such as SYMAP [Chrisman, 1988] and the Canada GIS [Tomlinson, 1988], they used boundary representation, depicting 2D areas as polygons defined by a sequence of vertices. By the 1980s, a *topological approach* based on planar partitions was instead used in systems like GRASS[40] and the coverages of ArcInfo [ESRI, 2005]. This was done using an intermediate representation where all input objects are partitioned into homogeneous regions [Rossignac and O'Connor, 1989], such that each region is linked to a set of input objects that are all present in the entire region. This intermediate representation is obtained by performing geometric intersection operations on all of the input objects and allows these partitions to be represented using the types of data structures for 2D cell complexes presented in §3.1.3.

However, current 2D GIS software has reverted to the use of simple data structures with very little or no topology, which are then supplemented by external spatial indices for speed. This type of representation has some drawbacks: it encourages inefficient representations where primitives are represented multiple times (e.g. when a face is part of the boundary of two polyhedra) [Cromley,

40: `http://grass.osgeo.org`



(a) (b)

Figure 3.19: The IFC standard supports objects defined through sweeps, which are defined by (a) an `IfcPCurve` (black spiral) and a `SweptArea` (blue disk), in this case resulting in (b) a screw shape.

1992, Ch. 3], which might not match exactly [de la Losa and Cervelle, 1999], it makes it difficult to check that a set of bounding elements conforms to certain properties (e.g. properly enclosing a space or being manifold), and the topological relationships between many elements need to be recomputed.

However, in the context of current 2D GIS software, in which different data sources are dynamically loaded and used together, it is often more efficient to opt for a representation with little topology, instead computing topological relationships only if and when they are needed, such as is done in ArcGIS [ESRI, 2005] and in some QGIS plug-ins. Many current use cases can actually do without the computation of these topological relationships: as explained in §3.2.1, visualisation does not require explicit topology, today's fast computers can process many datasets using simpler brute force approaches, and algorithms can take advantage of strong properties that are intrinsic to the 2D case, such as that it is possible to define a natural (sequential) order for the points around a closed polygonal curve, or that there can be at most two polygons incident to any edge in a planar partition.

Moreover, it is more robust—and often more convenient—to consider the geometric and topological information that is stored in a file as suspect, programming defensively to avoid errors in software by using repair techniques such as those that will be discussed in Chapter 10, as errors can frequently appear due to a variety of causes, e.g. numerical errors [Goldberg, 1991; Schirra, 1997] or differing interpretations of invalid objects [Ledoux et al., 2014].

The situation regarding 3D GIS is more complex, consisting of many custom made ad hoc systems that are very frequently described in the literature (e.g. the impressive one developed by Zhang et al. [2011]) following both topological and non-topological approaches, but in practice, there are very few general-purpose publicly available 3D GIS. In addition to this, there is limited 3D functionality in existing GIS software, such as 2.5D data containing elevation as an attribute [Raper, 1989], or capabilities for simple storage, visualisation and editing (e.g. moving vertices) of 3D models. This leaves more complex geometric operations, such as the use of Boolean set operations or complex deformations of 3D models within the realm of CAD or 3D modelling software. Partly as a consequence, 3D GIS datasets commonly contain large numbers of invalid geometries [Zhao et al., 2014].

3.2.4 Models used in 3D modelling software and libraries

Compared to GIS software, 3D modelling, CAD and other similar software have a greater emphasis on (interactive) editing and other operations that alter the geometry of a 3D object. However, many (or

most) 3D modelling software[41] represent 3D objects only as closed surfaces, using triangular or polygonal meshes that can be stored using the data structures for 2D simplicial/cell complexes presented in §3.1.3. This is not a limitation for simple 3D editing functionality, such as moving vertices, splitting edges and faces, and adding new vertices/edges/faces. However, more complex geometric operations with and between 3D objects generally require more powerful data structures with an explicit knowledge of volumes.

Nevertheless, some software that is used for this purpose follows a minimal approach similar to the one used for exchange file formats in GIS. The standard *topological data structure* in CAD[42] considers *vertices*, curved *edges* connecting vertices, *wires* consisting of a closed loop of edges, *faces* made from one outer wire and possibly multiple inner wires, closed *shells* made from a set of faces, and *solids* consisting of one outer shell and possibly multiple inner shells. For example, this is the approach that is used in K-3D [Shead, 2010] and openCASCADE library[43], the engine used for geometric computations in CAD software like FreeCAD[44] and the SALOME numerical simulator[45].

This approach often fails when attempting some operations that are notoriously difficult to perform robustly, e.g. Boolean set operations between polyhedra—something that is recognised in the documentation of multiple software packages. Other software thus uses more powerful data structures based on 3D cell complexes, generally achieving better results in the process. Blender 3D[46] uses something akin to the radial edge structure [Weiler, 1988] and Moka[47] uses 3D generalised maps [Lienhardt, 1994].

A different approach is followed by software like BRL-CAD[48], which uses a Constructive Solid Geometry engine and therefore stores objects as trees of Boolean set operations. Finally, the Nef polyhedra implementation in CGAL [Hachenberger, 2006] is able to compute Boolean set operations [Granados et al., 2003], convex decompositions [Chazelle, 1984] and Minkowski sums robustly. It is used in software like OpenSCAD[49].

41: Unfortunately, it is difficult to know which data structures are used internally in most commercial software, so this analysis is necessarily limited.

42: It is important to note the term 'topological data structure' has completely opposite meanings in GIS and CAD. The non-topological (Simple Features) approach in GIS is equivalent to the standard CAD topological data structure; the standard GIS topological data structure is equivalent to a CAD mesh, which is widely considered as the *less* topological approach. Objectively, they both embrace topology in different ways. The GIS topological data structure stores the (algebraic) topological relationships between polygonal areas; the CAD topological data structure stores the (point-set) topological relationships between a polygon/polyhedron and its different rings/shells.

43: `http://www.opencascade.org`

44: `http://www.freecadweb.org`

45: `http://www.salome-platform.org`

46: `http://www.blender.org`

47: `http://moka-modeller.sourceforge.net`

48: `http://brlcad.org`

49: `http://www.openscad.org`

3.3 Spatiotemporal modelling

Among the possible non-spatial characteristics that can be integrated with space, time in particular has long been considered to be interlinked with space [Akhundov, 1986]. Like space, it is considered to have geometry and topology [Earman, 1977], which are often modelled in accordance to the ISO 19108 standard [ISO, 2005b].

Spatiotemporal modelling attempts to create joint models that combine spatial and temporal information. It draws inspiration from

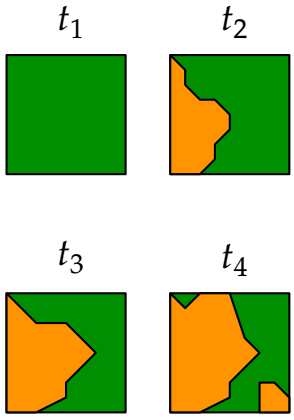


Figure 3.20: The snapshot model stores the state of a map at various moments in time. Based on Langran and Chrisman [1988].

the independent modelling of both space and time. The main models used in spatiotemporal modelling are presented below. For a more thorough review of spatiotemporal models see Al-Taha et al. [1994] and Pelekis et al. [2004].

The simplest and most widespread spatiotemporal model involves the use of timestamps. Known as the *snapshot* model, it was probably first used in the US Historical Boundary File [Basoglu and Morrison, 1978]. In this model, shown in Figure 3.20, every entity represents the state of an object during a specific timeframe. These entities might be objects of any dimension, e.g. polylines as in the US Historical Boundary File or polygons in a cadastral database [Hunter and Williamson, 1990]. An entity is therefore attached with a pair of timestamps, which demarcate the *start* and *end* of the period during which the entity existed as is represented. These models thus keep multiple representations of 2D [Armstrong, 1988] or 3D [Hamre et al., 1997] structures. This approach is simple but not very powerful, containing no direct relations between the objects that were present at the same location but during different periods of time.

There are a few variations of this model. For instance, it is possible to store differential changes only [Langran and Chrisman, 1988], specifying the areas that are added to or removed from an object at a given time. Another related possibility is keeping the current state of the map explicitly as well, which greatly improves the response time of this very common query.

In the *space-time composite* model, objects are first split into homogeneous regions that share the same history, similar to how overlapping objects are handled in topological GIS models [Rossignac and O'Connor, 1989]. This was first described in Chrisman [1983] based on Peucker and Chrisman [1975]. This is more flexible than the snapshot model, but objects can become very fragmented, slowing down many operations. For instance, updating the attributes of an object might involve updating all the regions that the object is split into.

Other models put a greater emphasis on events, such as by keeping a list of changes per object [Worboys, 1992a; Peuquet, 1994]. Event-based models [Peuquet and Duan, 1995] take this a step further, maintaining a main structure consisting of a list of events and a base map. Unlike other models, this makes it possible to identify and attach attributes to individual changes and events. In the *history graph* model [Renolen, 1996], different types of events are supported, which makes it possible to model continuously changing events as well. Similar models are used in computer animations, where a graph of keyframes can identify topological changes in a 2D vector drawing [Dalstein et al., 2015].

A different option is to keep track of space and time independently, linking objects appropriately. So called *three-domain* models are

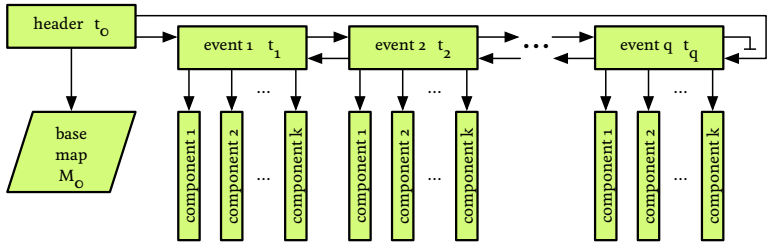


Figure 3.21: An event-based model maintains a list of events and a base map, with each event being linked to all changes that occurred since the last event. Based on Peuquet and Duan [1995].

based on this concept (the third domain being semantics). Examples include Yuan [1994] and Claramunt and Thériault [1995]. In another example, van Oosterom [1997] uses an identifier consisting of both a region identifier and a timestamp to index spatiotemporal objects.

Finally, there are some generic spatiotemporal models described at a more conceptual level [Story and Worboys, 1995], which can be adapted to suit a specific application. For instance, Tryfona and Jensen [1999] describe the space-time entity-relationship model, which is based on the entity-relationship [Chen, 1976] common in the database world. It provides rudimentary support for multi-scale objects by allowing for multiple geometries to be stored in each feature. Claramunt et al. [1999] discusses an object-relational model that is specifically tailored to model change. Additionally, there are a few object-oriented spatiotemporal models [Worboys et al., 1990].

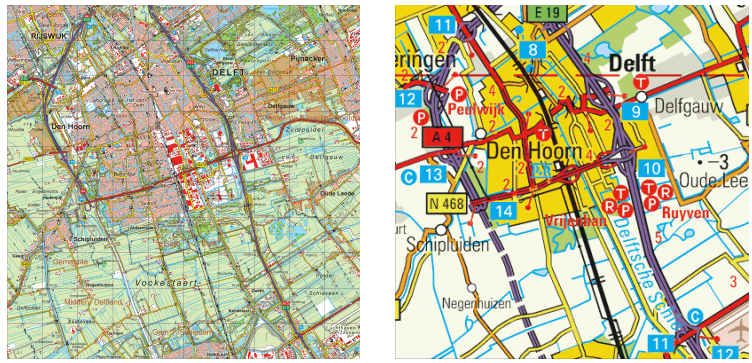
3.4 Modelling geographic scale

Traditional paper maps and physical models have a well-defined *scale*, whose value is a ratio between a linear measure in the model/map with the corresponding linear measure in the real world[50]. Scale is therefore a concept that is fixed and has a clear, measurable value in a model or map.

50: Non-linear measures (e.g. area and volume) in a model/map and reality are related non-linearly to this ratio.

However, apart from this direct meaning, scale also has an indirect relation to the *level of detail* (LOD) that is present in a map/model. Considering limits in resolution in printing and manufacturing technologies, the detail that can be put into a map or model is limited by its scale. Moreover, since humans are only able to reliably distinguish features of a certain size—a commonly accepted measure for maps being 0.5 mm [Goodchild, 2001]—, an adequate representation at a given scale presupposes objects being of a certain minimum size and a minimum distance apart.

The above mentioned factors limit the detail that *can* be present in a model/map at a given scale, but there are additional limits on the detail that *should* be present. Considering that maps are first and fore-



(a) 1:50 000 (b) 1:250 000

Figure 3.22: Fragments of two Dutch topographic maps at different scales around Delft. Note that the region depicted in both maps is the same and maps are (here) presented at the same size, and thus the scale of both in a literal sense is the same. However, the level of detail in each map differs. © Kadaster[51].

most communication tools, good maps need to have content that has an appropriate level of precision and detail for the good conveyance of their intended message [Hardy and Field, 2012].

In the digital realm, where models and maps both become data and the direct meaning of geographic scale as a ratio between a model/map and reality disappears, scale preserves only its indirect meaning as the level of detail that is present in a model or map, and accordingly the term 'level of detail' is more appropriate [Biljecki et al., 2014b]. An increase in the detail of a model enables more applications, but means that its representations occupy larger sizes and their processing involves higher computational costs[52]. When the term 'scale' is used instead, the level of detail that is expected is often presented in relation to its physical predecessor, such as in the map fragments shown in Figure 3.22. In digital models and maps, having an adequate level of detail also becomes important as a higher level of detail entails a correspondingly higher memory and computational requirements [Luebke et al., 2003].

Geographic information at different scales is generally managed as fully independent datasets at each scale [Meijers, 2011], such as in the topographic maps shown in Figure 3.22. In 3D, the CityGML standard [Gröger et al., 2012] defines the five LODs shown in Figure 3.23, which can be stored jointly but are effectively separate datasets except when their geometries are linked (cf. Biljecki et al. [2015]). This is similar to the snapshot model for spatiotemporal information [Basoglu and Morrison, 1978; Langran and Chrisman, 1988] as presented in §3.3, since these datasets are labelled with the scale (or range of scales) that they are associated with. This representation is simple and matches nicely with paper map series—in which a region is depicted at multiple scales in different maps. However, many objects are represented several times (once at each scale in which they are present) [Friis-Christensen and Jensen, 2003], and it is difficult to maintain consistency between these rep-

52: However, the consequences of more detailed maps/models should not been seen as a merely technical issue. High memory usage and processing time directly affect the usability of a model or map, and too much detail can often detract from its intended message.

51: http://www.kadaster.nl

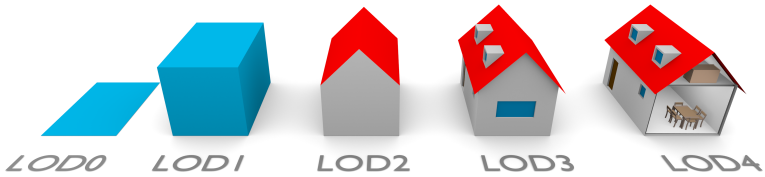


Figure 3.23: The CityGML standard [Gröger et al., 2012] defines five LODs for 3D city models, which range from building footprints to detailed architectural models of their interior and exterior. From Biljecki et al. [2014a].

resentations [Buttenfield and DeLotto, 1989], as the different representations of an object are unlinked or linked only by common identifiers[53].

In order to achieve consistency and complete coverage of a region at a given scale, it becomes important to be able to create a map of that scale using a more detailed map of the same region as input, resulting in maps at different scales covering the same region. This is achieved by the process of cartographic (or map) generalisation. In it, the amount of information in a map is reduced [Töfper and Pillerwizer, 1966], abstracting objects for a given scale using a set of well-defined cartographic rules [SGK, 1975]. Many of these rules can be automated in the form of algorithms (e.g. line simplification [Douglas and Peucker, 1973]), resulting in several automatic *map generalisation algorithms* [Weibel, 1997], and their (much more recent) resulting software implementations [Jones and Ware, 2005; Stoter et al., 2009]. In fact, it is now possible to generalise maps fully automatically in certain cases and with good results [Stoter et al., 2014].

In 3D modelling, similar algorithms are widely used to create simplified versions of complex 3D models [Meng and Forberg, 2007; Kada, 2007; Zhu et al., 2009; Zhao et al., 2012]—using visual or geometric criteria rather than cartographic—based on techniques such as mesh simplification [Garland and Heckbert, 1997; Lindstrom and Turk, 1998]. Considering that these techniques can be applied to the very detailed 3D models that are generated using surface reconstruction [Amenta and Bern, 1998; Kazhdan et al., 2006] from laser-scanned point clouds, or those resulting from the reuse of detailed architectural models [Geiger et al., 2015], they allow the generation of entire series of progressively simpler models up to an arbitrary level of detail in a process that is largely automatic—at least in theory[54].

Using automatic map generalisation, it becomes possible to create digital objects of any given scale, and thus to create and populate structures where an object at many scales is represented as a single entity, usually as a tree where detail is added progressively. For polygonal curves, this is accomplished by structures such as: strip trees [Ballard, 1981], the multi-scale line tree [Jones and Abraham, 1986], the arc tree [Günther, 1988] and the binary line generalisation tree [van Oosterom, 1990]. For planar partitions, it can be done

53: Which can be messy in practice as many times there is not a one-to-one mapping between the different representations.

54: In practice, it is a very challenging process with many engineering stumbling blocks. For one, fully automated surface reconstruction requires a certain density of sampling points (see Cheng et al. [2012, Ch. 13]), which is hard to guarantee. The generalisation of real-world 3D objects is also very challenging in practice due to a variety of reasons, such as the errors caused by the acquisition of data at different times, from different angles, from different sources, or using different methods.

structures such as: hierarchical planar subdivisions [Filho et al., 1995], multi-scale partitions [Rigaux and Scholl, 1995], nested maps [Plümer and Gröger, 1997] and topological generalised area partitioning trees [van Oosterom, 2005].

3.5 Key characteristics and shortcomings of current approaches

As shown in this chapter, there is great variety among the techniques used to model different aspects of 2D and 3D space, time and geographic scale. Considering all the possible ways in which these techniques can be combined together into a complete modelling approach, many different feasible approaches can be devised, which can be further fine-tuned for a particular application or use case. These vary in fundamental aspects, such as their explicitness, their use of geometry and topology, and the class of objects that they are able to handle efficiently.

However, despite these very real and substantial differences, the representation approaches in *2D GIS* have become largely interchangeable in practice. There is a myriad of well-known topological data structures that can be used, such as the DCEL [Muller and Preparata, 1978] and the quad-edge [Guibas and Stolfi, 1985], and even when certain objects are not directly supported in them (e.g. those with a non-manifold shape or with holes), they can usually be nevertheless stored using various simple 'tricks', such as repeated combinatorial primitives (as shown previously in Figure 3.10), special kinds of attributes or external data structures such as indices.

Moreover, many 2D GIS applications do not even require the explicit topology of a topological data structure. Visualisation and simple computations can just as easily do without it, and consequently use data structures with very little or no topology, opting for a Simple Features-like representation [OGC, 2011]. Also, considering that the computation of topological relationships between 2D objects is relatively simple, it can be done on-the-fly only when it is needed [ESRI, 2005]—something that is especially true for the relatively small size of most 2D datasets, which often pale in comparison to the computing power that is now readily available. All of these reasons point towards a similar conclusion: objects of up to two dimensions (i.e. points, line segments and polygons) can be represented using either topological or non-topological approaches with little difference in practice.

In 3D, the situation is more complex and topology brings more significant advantages, even as storing topological relationships comes at a significantly increased cost in terms of memory. Compared to the circumstances in 2D, there are more topological relationships between 3D objects, and these are more difficult and expensive to

compute, so their storage becomes highly desirable in many instances. At the same time, datasets are usually much larger, which makes topological relationships more valuable in order to traverse them efficiently. More complex applications such as simulations greatly benefit from 3D topological representations as well—or at least from data that is known to be valid, for which 3D topological data structures are used [Ledoux, 2013].

Moreover, many of the strong properties that allow for simple but powerful data structures and quick computations in 2D do not work in 3D. For instance, there is a natural order for the vertices or line segments around a polygon—used e.g. to efficiently store a polygon as a sequence of vertices—but there is no similar natural order for the faces around a polyhedron. Similarly, storing a complete planar partition as a set of edge primitives where every edge records the polygons that lie on each of its two sides [Peucker and Chrisman, 1975] is straightforward and efficient, but a 3D space partition stored as a set of faces where every face only knows the volumes that lie on both of its sides is very cumbersome to navigate—even a simple operation such as extracting the volumes in the partition is difficult without the adjacency relationships between the faces.

Partly because of this, as well as the general increase in complexity that comes with an increase in the number of dimensions, the topological data structures that are capable of storing topological relationships between sets of 3D objects are less used in practice. The third spatial dimension, time, and scale are mainly implemented using ad hoc adaptations to 2D data structures, effectively limiting the capabilities of GIS software. 3D GIS often mimic the third dimension by using a so-called 2.5D structure [Raper, 1989], essentially treating the third dimension as an attribute and restricting the geometries that can be represented; or represent 3D objects individually and only implicitly through their 2D boundary, using a 2D data structure with no 3D topological relationships[55]. This implies that many operations are only possible using expensive searches involving many more objects than otherwise needed.

55: i.e. topological relationships involving 3D primitives, such as the adjacencies between two volumes or the incidences between a volume and a vertex/edge/face

Time and scale are considered to be inseparable in the representational process by theorists, since events have an intrinsic place in space and time, as well as specific spatial and temporal resolutions [Raper, 2000]. However, spatiotemporal GIS keep multiple representations of 2D [Armstrong, 1988] or 3D [Hamre et al., 1997] structures, or a list of changes per object [Worboys, 1992a; Peuquet, 1994], limiting combined spatio-temporal analysis of such objects. Multi-scale datasets generally consist of independent datasets for each scale, which are either unconnected or connected only at the object level (e.g. through the use of IDs). This means that complex relationships between objects, such as collapses, aggregations and others that are not one-to-one are difficult to store, which causes, among others, update and maintenance problems as well as incon-

sistencies. It also complicates the storage of semantic information about these relationships.

The higher-dimensional spatial modelling approach

4

As presented in Chapter 3, the most common approaches to model 2D and 3D space are based on ad hoc adaptations of data structures that were originally meant for 2D data. These have many advantages, but are also necessarily limiting when 3D objects are concerned, constraining the class of objects that they are capable of representing efficiently and the operations that can be efficiently performed on them. Even data structures specifically defined for sets of 3D objects have significant limitations, such as not being able to store the incidence and adjacency relationships between objects. Moreover, when the data has to be integrated with non-spatial characteristics (e.g. time and scale), similar ad hoc adaptations to 2D structures are also used. Therefore, the problem of adequately representing 3D, spatio-temporal and multi-scale data is not fully solved even by the most advanced 3D structures as they are currently used in GIS.

This chapter presents the higher-dimensional spatial information modelling approach—the main theme of this thesis—as a solution to all these problems, as well as the higher-dimensional data models and data structures that can be used to realise it. §4.1 introduces the topic by showing the strong theoretical foundations behind higher-dimensional modelling. How higher-dimensional modelling works in general is then explained in §4.2, relating to relevant concepts of 2D/3D GIS whenever possible, as well as importing useful notions from other fields. §4.3 then describes the data models and data structures that can be used to realise the higher-dimensional modelling approach. §4.4 finalises the chapter by presenting some of the new possibilities provided by this approach.

Significant parts of §4.3 and §4.4 and minor parts of the other sections are based on the paper:

- **An evaluation and classification of *n*D topological data structures for the representation of objects in a higher-dimensional GIS**. Ken Arroyo Ohori, Hugo Ledoux and Jantien Stoter. *International Journal of Geographical Information Science* 29(5), May 2015, pp. 825–849.

4.1 Foundations of higher-dimensional modelling

A radical departure from the traditional approaches to model 2D/3D space, time and scale in GIS, as presented in Chapter 3, consists in the representation of any number of *parametrisable characteristics* (e.g. the third spatial dimension, time and scale) as additional dimensions in a geometric sense. These dimensions are modelled so as to be orthogonal to each other, such that real-world 0D–3D entities are modelled as higher-dimensional objects[57] embedded in higher-dimensional space, which are consequently stored using *higher-dimensional data structures*.

57: That is, as long as such objects exist along intervals along these other dimensions. If they only exist at a point along these dimensions (e.g. a moment in time), they are of the same dimension as the original object.

Higher-dimensional modelling is well grounded in long-standing mathematical theories. Descartes [1637] already laid the foundation for nD geometry by putting coordinates to space, allowing the numerical description of geometric primitives and the use of algebraic methods on them, theories of nD geometry were developed by Riemann [1868] among others, and Poincaré [1895] developed algebraic topology with a dimension-independent formulation, stating that even if nD objects could not be [then] represented, they do have a precise topological definition, and consequently properties that can be studied.

This approach opens the door to new possibilities. From an application point of view and as will be discussed in §4.2.3, 4D topological relationships between 4D objects provide insights that 3D topological relationships cannot. Also, McKenzie et al. [2001] contends that weather and groundwater phenomena cannot be adequately studied in less than four dimensions, and van Oosterom and Stoter [2010] argue that the integration of space, time and scale into a 5D model for GIS can be used to ease data maintenance and improve consistency, as algorithms could detect if the 5D representation of an object is self-consistent and does not conflict with other objects.

It is important to note that the higher-dimensional modelling approach—as presented in this thesis—bears no relation to the most common usage of 4D, 5D, 6D, …, nD GIS/BIM in both GIS product descriptions and the scientific literature. There, such terms are generally used as buzzwords that refer to *any* kind of support for the storage of time information, costs, project life cycles, or any other kind of non-spatial information. In most cases, this information is simply stored as a tuple of attributes that are appended to 2D/3D objects (e.g. timestamps), or as external structures that are unlinked to any objects with a geometric description (e.g. IFC scheduling information). If one were to interpret these objects geometrically, they would result in 2D/3D objects embedded in higher-dimensional space[58], and *not in higher-dimensional objects embedded in higher-dimensional space*.

58: Orthogonal and point-like with respect to the higher dimensions.

However, it is worth noting that real strides towards higher-dimensional GIS have been made in various forms. Research in multidimensional GIS aims at the integrated storage and analysis of heterogeneous objects of different dimensions [Raper, 2000; Gold, 2005], usually limited to 0D–3D objects but sometimes conceptually extended to higher dimensions. Storing and manipulating heterogeneous objects and their semantics is a difficult challenge and is a notable weak point of current GIS tools.

The concept of time as a dimension is an established concept with proven applications. As Couclelis [1999] states, time has historically been linked to space and often considered as another dimension, and the 4D spatiotemporal manifold known as the Minkowski space [Minkowski, 1908] is used in the description of special relativity in mathematics and physics[59]. The ISO 19108 standard also states that 'time is a dimension analogous to any of the spatial dimensions' [ISO, 2005b] A representation where time is modelled as one more spatial dimension has been moreover used in several applications, mostly in the form of 2D space+time, such as to analyse paths [Hägerstrand, 1970], to analyse motion in computer vision [Blank et al., 2005] and to visualise geographic information [Kraak, 2003]. 3D+time systems have also been frequently proposed, albeit this usually is meant only at a conceptual (rather than implementation) level [Galton, 2004]. In GIS and other fields, a 2D space+time representation is often called a *space-time cube*[60].

Scale as a dimension has a shorter history, only recently being proposed as way to integrate 2D/3D space, time and scale [van Oosterom and Stoter, 2010]. Döllner and Buchholz [2005] consider the notion of the LOD of a model as a conceptual dimension. Grasset-Simon et al. [2006] discuss generalised map pyramids, which use n-dimensional generalised maps in order to create a hierarchical data structure of 2D/3D/nD images of different resolutions. Untereiner et al. [2015] introduce the Compact Primal Hierarchy, an also hierarchical representation of manifold meshes in any dimension which are generated through successive refinement operations. Finally, van Oosterom and Meijers [2014] discuss the integration of 2D maps at different scales in one 2D+scale model that is conceptually 3D—but is represented instead as a series of linked 2D structures—and is known as the *space-scale cube*[61].

59: Minkowski space is non-Euclidean, but similar representations and techniques can be used with it. For instance, Hanson and Weiskopf [2001] show how computer graphics techniques can be used to visualise it. General relativity is instead described by Riemann space [Riemann, 1868].

60: The use of the term 'cube' is useful for clarity of explanation but not strictly correct since it implies that the analysed region has the same length along every axis. This term is thus not used elsewhere in this thesis. By contrast, Hägerstrand [1970], who is largely credited with popularising the concept, refers to it as the space-time prism.

61: A similar objection could be made to it being called a cube though.

4.2 The higher-dimensional modelling paradigm

Considering the strong foundations of higher-dimensional modelling explained in §4.1, the higher-dimensional modelling of geographic information has great potential to integrate space (of any dimension) with any number of non-spatial characteristics. Such a representation is extremely powerful, capable of handling even

complex situations like objects that are both changing shape (i.e. *morphing*) and moving in time, but it is also complex in several ways.

This section serves as a broad overview of the higher-dimensional modelling paradigm, covering the main conceptual difficulties in how these models work and describing intuitively what they look like. The first fundamental aspect, deciding which characteristics can and should be modelled as dimensions, is described in §4.2.1. Afterwards, considering that understanding how objects in more than three dimensions are hard to visualise, §4.2.2 explains intuitively what these objects look like based on 3D models using 2D+time and 2D+scale, as well as making analogies between the properties of higher-dimensional models and 2D/3D ones.

§4.2.3 discusses the key distinguishing feature of the higher-dimensional approach—the possibility to store and use topological relationships in arbitrary dimensions. Finally, §4.2.4 explains how the standard topological approach in GIS, creating a functional space partition from a set of objects, can also be used in higher dimensions to store these topological relationships in the data structures presented in §4.3.

4.2.1 Defining the characteristics to be modelled as dimensions

As powerful as higher-dimensional representations are, their advantages need to be balanced with the fact that higher-dimensional representations are large and complex. The size of such a model can be expected to increase roughly exponentially with the dimension—although the exact values are highly dependent on the model used and the data being represented. For example, consider the total number of cells of all dimensions in the family of hypercubes. A point has 1 cell, a line segment 3 (2+1) cells, a square 9 (4+4+1) cells, a cube 27 (8 + 12 + 6 + 1) cells and a tesseract, shown in Figure 4.1, 81 (16 + 32 + 24 + 8 + 1) cells.

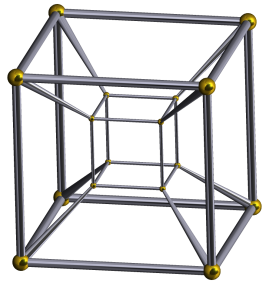


Figure 4.1: A tesseract or 4-cube is the four-dimensional analogue of a square (in 2D) or cube (in 3D). It consists of 16 vertices, 32 edges, 24 faces, 8 volumes and one 4-cell.

Because of this, while many characteristics *can* be used as a dimension, only those characteristics where such modelling would bring significant benefits *should* be modelled in this manner. In fact, while there is no theoretical limit to the number of dimensions that can be used, the increasing space requirements of models with more dimensions generally limit the practical number of dimensions to 6–8.

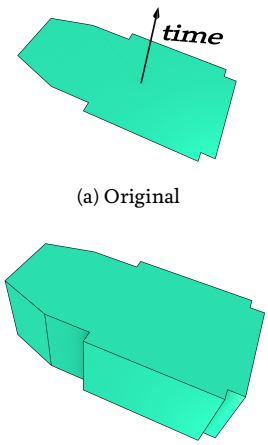
There are various requirements and desirable properties in the characteristics that are to be modelled as additional dimensions. At minimum, such characteristics should be parametrisable (i.e. there should be a way to express them in terms of a parameter or parameters), so that these parameters can be entered into a computer,

and preferably parametrised as numbers that determine the object's position along the axes of the dimensions. Many other properties are not strictly necessary but are still very desirable, including that there is a parametric function that models the characteristic in a natural way, and that this function is invertible so that the object's properties can be known from the inverse function. These are properties that are fulfilled by data that is quantitative in nature, which is related to measurements of an interval or ratio scale [Stevens, 1946][62].

62: Measurements might not be the best term here considering that much data is modelled without any measurements taking place, but this is the original terminology in Stevens [1946].

In addition, there are certain properties of the data to be represented that determine if the higher-dimensional modelling approach is worth the added complexity and size. For this, it is possible to evaluate it by considering whether more compact alternatives are able to depict the situations that are present in the data. For instance, if the value of one of the characteristics can be determined from the others without added knowledge, the former can be simply obtained from the latter. Also, when a given characteristic has a bounded number of values for every possible combination of the other characteristics, the former can be stored more efficiently as an attribute. A typical example of this latter case is a 2.5D terrain model, where there is only one height (z) value for any given combination of (x, y) coordinates. Caves, tunnels, overhangs and arches are thus problematic.

Time and scale—the examples used throughout this thesis—comply with most of the properties mentioned above. They are both intuitively quantifiable in a meaningful way. For instance, a point along a temporal axis defines a moment in time such that points on one side of it occur before it and points on the other side of it occur after it. Similarly, an object at a given LOD is finer than those at a lower LOD and coarser than those at a higher LOD. Moreover, in the general case, an object's position in time or its geographic scale cannot be known based solely on its position in 2D or 3D space, and an object can be at the same position at different moments in time (e.g. by not moving) or at different scales.



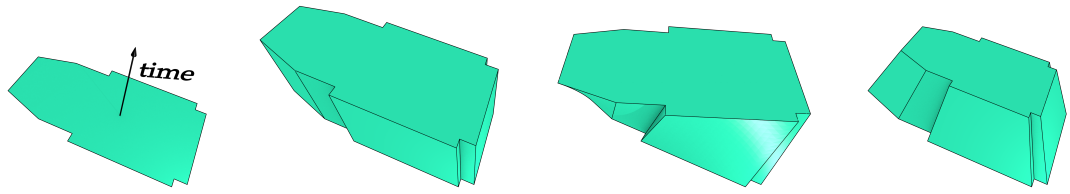
(a) Original

(b) Extrusion

Figure 4.2: In a 2D+time setting, (a) a 2D object remaining stationary becomes (b) a volume where the original object is extruded along the time dimension.

4.2.2 How higher-dimensional data looks

To understand how data modelled using the higher-dimensional modelling paradigm looks, it is easier to first consider a case with a two-dimensional space plane (x, y) where time is added as a third dimension. In this configuration, a set of 2D objects, each of which exists for one or more intervals of time, can be represented as a set of volumes. The simplest case is when these objects remain stationary, which is shown in Figure 4.2. In this case, the shape of the resulting volume is an extrusion of the original 2D shape along the time axis. Such a volume has a prismatic shape and can be generated using the methods that will be described in Chapter 6. As shown in Figure 4.3,



(a) Original (b) Translation (c) Rotation (d) Scaling

Figure 4.3: In a 2D+time setting, applying certain transformations to (a) a polygon during a time interval results in: (b) a parallelepiped in the case of a translation, (c) a screw shape with curved surfaces similar to a *twisted prism* in the case of a rotation, and (d) a *frustum* in the case of scaling.

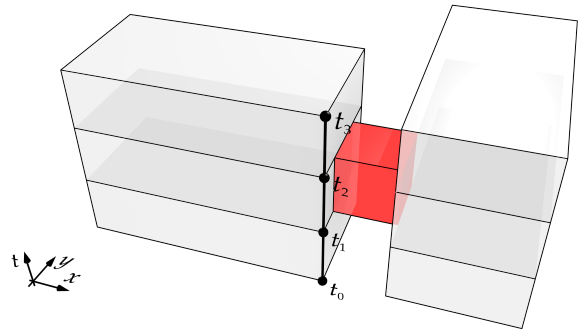


Figure 4.4: A 2D space (x, y) + time (the vertical axis t) view of the footprint of two buildings. These buildings were separate at $t = t_0$, were then connected by a corridor (red) at $t = t_1$, then became disconnected again when the corridor was removed at time t_2. This configuration remains unchanged until time $t = t_3$. The times t_0, t_1, t_2 and t_3 are shown as points along the thick line representing the front right corner of the building on the left.

applying translation, rotation or scale transformations to the 2D object during a time interval yield other volumes that have an intuitive shape.

In the context of geographic information, where many kids of features are represented as planar partitions of polygons, the resulting 2D space+time polyhedra form a 3D space partition—if the polygons do not overlap, the resulting polyhedra do not overlap either. This property is later exploited by the extrusion algorithm in Chapter 6. Moreover, as shown in Figure 4.4, such a representation can represent the state of the partition at any point in time. At any one moment in time (i.e. a slice perpendicular to the 2D space plane), a polygon is still represented as a polygon but one that is embedded in 3D space, parallel to the 2D space plane (x, y) and orthogonal to the time axis. Every object existing (and not moving or changing shape) during a time period would then be prism shaped, with identical base and top faces that are parallel to the 2D space plane and the other faces being orthogonal to it.

Extending this to a 4D representation of 3D space and time, a set of polyhedra can be represented as a set of *polychora*[63]. At any one moment in time, a polyhedron is still represented as a polyhedron

63: A polychoron is the 4D analogue of a 2D polygon or 3D polyhedron.

but one that is embedded in 4D space. If it is not moving or changing shape, it would take the form of a prismatic polychoron. As in the previous cases, if the polyhedra form a 3D space subdivision, the polychora form also a 4D space subdivision.

In the same fashion as the 2D space+time case, it is possible to create similar 3D representations where 2D space is combined with scale as a third dimension. As shown in Figure 4.5, such a representation of a planar partition of polygons at different scales results in a 3D space subdivision where at each scale point there is still a planar partition. As shown in Figure 4.6 and discussed by Meijers [2011], this approach enables the storage of continuous levels of detail—an approach that can be extended to higher dimensions as well [Stoter et al., 2012a,b].

In a general setting, it is then possible to define an n-dimensional model where a set of 0D–3D objects are integrated with any number of additional characteristics and modelled as up-to-n-dimensional objects embedded in n-dimensional space. These higher-dimensional objects can be stored or used for analysis as is, or 0D–3D objects can be extracted from them by looking at specific 'slices' of these objects—intersections of the objects with specific subsets of space (e.g. a plane parallel to the 2D space plane), projections to certain planes or regions of 3D space, or more broadly, transformations to arbitrarily defined 2D/3D coordinate systems. This is described in more detail in Chapter 9.

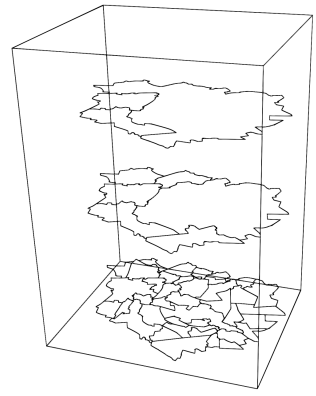


Figure 4.5: A 3D representation of 2D space (horizontal plane) and scale (vertical axis). The vertical edges connecting corresponding features are omitted for legibility. From Meijers [2011].

4.2.3 Topological relationships in higher dimensions

As described in §3.1.4, spatial data structures often consist of two aspects: (i) a combinatorial part, which consists of a set of primitives and some topological relationships between them, and (ii) an embedding that links these primitives to their geometry and attributes. This distinction is useful to give a more formal understanding of what topological relationships in higher dimensions are, as well as to highlight how the storage of higher-dimensional topological relationships in a combinatorial structure is the main aspect that differentiates higher-dimensional structures from 2D/3D ones.

This is clearer when considering the data structures that contain only 1D topological relationships, which are commonly used in GIS. For instance, in the Simple Features Specification [OGC, 2011], independent 2D primitives are defined as sequences of points (with coordinates) that are connected using implicit line segments. When structures constructed from these 2D primitives are described (e.g. planar partitions or polyhedra), they are simply considered as sets of these 2D primitives. Apart from the relations used to define that a primitive belongs to a set, no other information is stored explicitly and the 2D primitives are therefore independent from each other.

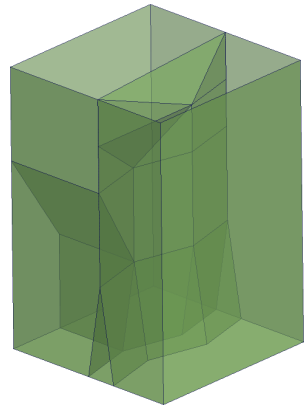


Figure 4.6: A 2D+scale representation of four polygons being smoothly (C^0) generalised. From van Oosterom and Meijers [2011].

Since the implicit line segments in every 2D primitive are known to be adjacent but nothing is known about the relationships between the 2D primitives themselves, such data structures can be considered to store up to 1D topological relationships only (adjacencies between edges and the incidences between edges and vertices). By contrast, a data structure without even these topological relationships could consist of an unordered soup of line segments defined by their endpoints, which are repeated in every line segment in which they are present.

Despite their apparent limitations, these types of structures can be in fact used to store objects of any dimension thanks to the Jordan-Brouwer theorem [Lebesgue, 1911; Brouwer, 1911] described in §3.1.2. Just as independent 2D primitives are defined as sequences of points (with coordinates) that are connected using implicit line segments, 3D primitives can be then represented as sets of such 2D elements, which are otherwise unlinked, and nD objects can be represented as aggregations of their $(n-1)$D-face bounding elements. This property is later exploited in the incremental construction scheme proposed in Chapter 7. Since every point in such an object can have any number of coordinates, an nD object can be embedded in nD space as well.

However, such representations become exponentially more inefficient as the dimension increases: lower-dimensional primitives need to be encoded multiple times, recursively once for each time they appear in a higher-dimensional primitive. This means that they are difficult to navigate and that even simple geometric and topological queries involve searching many objects [Hazelton, 1998]. As an example, it is possible to consider Simple Features-like [OGC, 2011] representations of simple objects of various dimensions, as shown in Figure 4.7. A Simple Features-like representation of a square does not repeat most[64] vertices, while one of a cube repeats every vertex at least three times, one of a tesseract (their four-dimensional analogue) repeats every vertex at least 12 times and one of a 5-cube at least 60 times.

64: The number of times a vertex is repeated in Simple Features can be up to doubled since representing a closed polygon involves repeating its first vertex at the end.

Because of this, in a similar vein to the definition of the dimensionality of a GIS in Hazelton et al. [1990], it is possible to consider that the defining characteristic of a higher-dimensional spatial data structure is not the ability to be embedded into higher-dimensional space, but being able to explicitly store topological relationships in a dimension-independent manner.

Moreover, topological relationships in more than 3D have advantages other than efficiency of representation. For instance, 4D topological relationships between 4D objects provide insights that 3D topological relationships cannot, such as by using the dual graph of such a 4D structure for applications such as space-time wayfinding and analysis. This will be discussed in more detail in §5.4.

```
[[0, 0], [0, 1], [1, 1], [1, 0], [0, 0]]
```

(a) square

```
[[[0, 0, 0], [0, 1, 0], [1, 1, 0], [1, 0, 0], [0, 0, 0]],
 [[0, 0, 0], [0, 1, 0], [0, 1, 1], [0, 0, 1], [0, 0, 0]],
 [[0, 1, 0], [1, 1, 0], [1, 1, 1], [0, 1, 1], [0, 1, 0]],
 [[1, 1, 0], [1, 0, 0], [1, 0, 1], [1, 1, 1], [1, 1, 0]],
 [[0, 0, 1], [0, 1, 1], [1, 1, 1], [1, 0, 1], [0, 0, 1]]]
```

(b) cube

```
[[[[0, 0, 0, 0], [0, 1, 0, 0], [1, 1, 0, 0], [1, 0, 0, 0], [0, 0, 0, 0]],
  [[0, 0, 0, 0], [0, 1, 0, 0], [0, 1, 1, 0], [0, 0, 1, 0], [0, 0, 0, 0]],
  [[0, 1, 0, 0], [1, 1, 0, 0], [1, 1, 1, 0], [0, 1, 1, 0], [0, 1, 0, 0]],
  [[1, 1, 0, 0], [1, 0, 0, 0], [1, 0, 1, 0], [1, 1, 1, 0], [1, 1, 0, 0]],
  [[1, 0, 0, 0], [0, 0, 0, 0], [0, 0, 1, 0], [1, 0, 1, 0], [1, 0, 0, 0]],
  [[0, 0, 1, 0], [0, 1, 1, 0], [1, 1, 1, 0], [1, 0, 1, 0], [0, 0, 1, 0]]],
 [[[0, 0, 0, 0], [0, 1, 0, 0], [1, 1, 0, 0], [1, 0, 0, 0], [0, 0, 0, 0]],
  [[0, 0, 0, 0], [0, 1, 0, 0], [0, 1, 0, 1], [0, 0, 0, 1], [0, 0, 0, 0]],
  [[0, 1, 0, 0], [1, 1, 0, 0], [1, 1, 0, 1], [0, 1, 0, 1], [0, 1, 0, 0]],
  [[1, 1, 0, 0], [1, 0, 0, 0], [1, 0, 0, 1], [1, 1, 0, 1], [1, 1, 0, 0]],
  [[1, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 1], [1, 0, 0, 1], [1, 0, 0, 0]],
  [[0, 0, 0, 1], [0, 1, 0, 1], [1, 1, 0, 1], [1, 0, 0, 1], [0, 0, 0, 1]]],
 [[[0, 0, 0, 0], [0, 1, 0, 0], [0, 1, 1, 0], [0, 0, 1, 0], [0, 0, 0, 0]],
  [[0, 0, 0, 0], [0, 1, 0, 0], [0, 1, 0, 1], [0, 0, 0, 1], [0, 0, 0, 0]],
  [[0, 1, 0, 0], [0, 1, 1, 0], [0, 1, 1, 1], [0, 1, 0, 1], [0, 1, 0, 0]],
  [[0, 1, 1, 0], [0, 0, 1, 0], [0, 0, 1, 1], [0, 1, 1, 1], [0, 1, 1, 0]],
  [[0, 0, 1, 0], [0, 0, 0, 0], [0, 0, 0, 1], [0, 0, 1, 1], [0, 0, 1, 0]],
  [[0, 0, 0, 1], [0, 1, 0, 1], [0, 1, 1, 1], [0, 0, 1, 1], [0, 0, 0, 1]]],
 [[[0, 1, 0, 0], [1, 1, 0, 0], [1, 1, 1, 0], [0, 1, 1, 0], [0, 1, 0, 0]],
  [[0, 1, 0, 0], [1, 1, 0, 0], [1, 1, 0, 1], [0, 1, 0, 1], [0, 1, 0, 0]],
  [[1, 1, 0, 0], [1, 1, 1, 0], [1, 1, 1, 1], [1, 1, 0, 1], [1, 1, 0, 0]],
  [[1, 1, 1, 0], [0, 1, 1, 0], [0, 1, 1, 1], [1, 1, 1, 1], [1, 1, 1, 0]],
  [[0, 1, 1, 0], [0, 1, 0, 0], [0, 1, 0, 1], [0, 1, 1, 1], [0, 1, 1, 0]],
  [[0, 1, 0, 1], [1, 1, 0, 1], [1, 1, 1, 1], [0, 1, 1, 1], [0, 1, 0, 1]]],
 [[[1, 1, 0, 0], [1, 0, 0, 0], [1, 0, 1, 0], [1, 1, 1, 0], [1, 1, 0, 0]],
  [[1, 1, 0, 0], [1, 0, 0, 0], [1, 0, 0, 1], [1, 1, 0, 1], [1, 1, 0, 0]],
  [[1, 0, 0, 0], [1, 0, 1, 0], [1, 0, 1, 1], [1, 0, 0, 1], [1, 0, 0, 0]],
  [[1, 0, 1, 0], [1, 1, 1, 0], [1, 1, 1, 1], [1, 0, 1, 1], [1, 0, 1, 0]],
  [[1, 1, 1, 0], [1, 1, 0, 0], [1, 1, 0, 1], [1, 1, 1, 1], [1, 1, 1, 0]],
  [[1, 1, 0, 1], [1, 0, 0, 1], [1, 0, 1, 1], [1, 1, 1, 1], [1, 1, 0, 1]]],
 [[[1, 0, 0, 0], [0, 0, 0, 0], [0, 0, 1, 0], [1, 0, 1, 0], [1, 0, 0, 0]],
  [[1, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 1], [1, 0, 0, 1], [1, 0, 0, 0]],
  [[0, 0, 0, 0], [0, 0, 1, 0], [0, 0, 1, 1], [0, 0, 0, 1], [0, 0, 0, 0]],
  [[0, 0, 1, 0], [1, 0, 1, 0], [1, 0, 1, 1], [0, 0, 1, 1], [0, 0, 1, 0]],
  [[1, 0, 1, 0], [1, 0, 0, 0], [1, 0, 0, 1], [1, 0, 1, 1], [1, 0, 1, 0]],
  [[1, 0, 0, 1], [0, 0, 0, 1], [0, 0, 1, 1], [1, 0, 1, 1], [1, 0, 0, 1]]],
 [[[0, 0, 1, 0], [0, 1, 1, 0], [1, 1, 1, 0], [1, 0, 1, 0], [0, 0, 1, 0]],
  [[0, 0, 1, 0], [0, 1, 1, 0], [0, 1, 1, 1], [0, 0, 1, 1], [0, 0, 1, 0]],
  [[0, 1, 1, 0], [1, 1, 1, 0], [1, 1, 1, 1], [0, 1, 1, 1], [0, 1, 1, 0]],
  [[1, 1, 1, 0], [1, 0, 1, 0], [1, 0, 1, 1], [1, 1, 1, 1], [1, 1, 1, 0]],
  [[1, 0, 1, 0], [0, 0, 1, 0], [0, 0, 1, 1], [1, 0, 1, 1], [1, 0, 1, 0]],
  [[0, 0, 1, 1], [0, 1, 1, 1], [1, 1, 1, 1], [1, 0, 1, 1], [0, 0, 1, 1]]],
 [[[0, 0, 0, 1], [0, 1, 0, 1], [1, 1, 0, 1], [1, 0, 0, 1], [0, 0, 0, 1]],
  [[0, 0, 0, 1], [0, 1, 0, 1], [0, 1, 1, 1], [0, 0, 1, 1], [0, 0, 0, 1]],
  [[0, 1, 0, 1], [1, 1, 0, 1], [1, 1, 1, 1], [0, 1, 1, 1], [0, 1, 0, 1]],
  [[1, 1, 0, 1], [1, 0, 0, 1], [1, 0, 1, 1], [1, 1, 1, 1], [1, 1, 0, 1]],
  [[1, 0, 0, 1], [0, 0, 0, 1], [0, 0, 1, 1], [1, 0, 1, 1], [1, 0, 0, 1]],
  [[0, 0, 1, 1], [0, 1, 1, 1], [1, 1, 1, 1], [1, 0, 1, 1], [0, 0, 1, 1]]]]
```

(c) tesseract

Figure 4.7: Simple Features-like representations of a unit square, cube, and tesseract. One face (per volume and per 4-cell) is shown in each line. Due to the repetition of structures in this type of representation, a 5-cube, which has only 80 faces, requires 480 lines to be represented, which is 10 times the number of lines and 12.16 times the number of characters used to represent the tesseract in (c).

4.2.4 Representing higher-dimensional objects as a space partition

In order to store the higher-dimensional topological relationships described in §4.2.3, it is a practical requirement is that the objects being represented can be made to functionally form a space partition. For this, it is possible to use the standard topological modelling approach used in GIS—utilising an intermediate type of representation where a set of (possibly overlapping) original objects is transformed into a set of non-overlapping regions, such that each of these regions represents *a set of the original objects* [Rossignac and O'Connor, 1989], which might be the empty set in the case of the regions that are not covered by any of the original objects (e.g. some holes). An original geometry can be obtained by performing a Boolean set union of the regions labelled as belonging to it.

This transformation can be computed in various ways. In 2D and 3D, all objects are generally first overlaid on top of each other, the intersections between their boundaries are computed, and new regions are obtained by finding closed connected regions uncrossed by any other boundaries (e.g. cycles in 2D) [de Berg et al., 2008, §2.3]. It is worth noting however that implementing such a procedure robustly in more than 3D is not a trivial undertaking.

Using this newly created space partition, even complex topological relationships between the original objects can be reduced to a combination of set membership, adjacency and incidence. In fact, many schemes of topology in GIS can be implemented solely on inspecting which objects are in a set [Egenhofer and Franzosa, 1991; Worboys, 1992b; Güting et al., 2000]. These can be very intuitive, as they often have direct correspondences to natural language equivalents (e.g. 'inside', 'touches', 'equals', etc.) [Dube and Egenhofer, 2012].

Moreover, it possible to use the data structures presented in §4.3 to store a traversable structure containing the objects, in which the boundaries of the sets are explicitly represented, and to easily query the topological relationships that exist between the objects.

The most promising structures for higher-dimensional GIS are therefore those that starting from a space subdivision, are capable of storing a well-defined and relatively broad class of objects of arbitrary dimension, embedded into Euclidean space of the same or higher dimension, attributes attached to such objects, and enough topological relationships between them to allow for the quick traversal of the structure and for basic operations, such as testing if two given objects are identical, adjacent or overlapping.

4.3 Higher-dimensional data models and structures

This section presents all the main possibilities to represent dimension-independent objects, which are presented as a series of data models, each with related data structures. These are: exhaustive enumeration (e.g. rasters) in §4.3.1, hierarchical space subdivisions using simple tree-like structures (e.g. BSP-trees) in §4.3.2, more general implicit vector models (e.g. constructive solid geometry) in §4.3.3, geometric simplicial complexes (i.e. triangulations) in §4.3.4, cell complexes in §4.3.5, and ordered topological models (e.g. generalised and combinatorial maps) in §4.3.6. Short notes on other, less relevant vector data structures are discussed in §4.3.7.

4.3.1 Exhaustive enumeration

Exhaustive enumeration in higher dimensions works in a similar manner as in 2D and 3D. The shape of an n-dimensional domain is first specified, generally opting for a simple shape such as an axis-parallel *n-orthotope*, the n-dimensional analogue of a 2D rectangle or a 3D box (more formally a *cuboid*), which can be defined based on two points in $\mathbb{R}^n$ on opposite corners of the orthotope. Afterwards, a simple deterministic rule is then used in order to partition the domain directly into n-cells. A programatic order or path that traverses all cells is defined, and the order of the path along these cells is used to record whether a cell belongs to the interior of the object or not, or in the case of multiple objects, which objects are present inside a cell.

Regular tessellations involve the subdivision of n-dimensional space directly into primitive cells forming *regular n-polytopes*—the n-dimensional analogue of regular polygons and regular polyhedra. A regular n-polytope is one whose bounding faces are regular $(n-1)$-polytopes[65]. As in 2D and 3D, it is possible to create arbitrary rules in order to produce cells of any shape and size, which can be used to create optimal configurations that are related to n-ball packing or kissing number problems in geometry [Conway and Sloane, 1992]. For instance, such cells could be arranged in the form of the optimal 24-cell honeycomb in 4D.

65: More formally, one whose symmetry is transitive on its flags.

However, operations on these optimum configurations can be rather complex. Axis-aligned regular grids consisting of congruent orthotopes, such as those taking the form of pixels in 2D and voxels in 3D, are much simpler in practice and offer a good enough approximation for practical purposes.

As in 2D and 3D, semi-regular tessellations, which consist of more than one type of polytope are also possible but are significantly more complex in practice. Creating optimal configurations of these

tessellations is a problem related to the unequal hypersphere packing, as the general aim is still to create polytopes that are as close to balls (of appropriate sizes) as possible.

Exhaustive enumeration schemes are relatively straightforward even in higher dimensions and can be stored and processed using similar techniques as their 2D and 3D counterparts. However, these representations suffer from the same problems as 2D/3D grids: they are incapable of representing precise boundaries, their contents vary when they are translated, rotated or scaled, and are cumbersome for the representation of objects [Fisher, 1997], especially when they are of mixed dimensions. Moreover, they have extremely high space requirements—increasing *exponentially* with the dimension in grids with a similar number of divisions along each dimension.

Nevertheless, up to 4D grids using time as a dimension have been used in the context of GIS since the 1990s. These are currently used widely for computer simulations in various fields and have recently been implemented in standard GIS software[66]. Much earlier, Mason et al. [1994] implemented a system using a 4D grid of ocean temperatures with support for interpolation and generalisation operations. Bernard et al. [1998] also implemented a 4D grid of atmospheric variables (e.g. temperature, wind or pollution), which can be used for simulations.

66: http://video.esri.com/watch/3653/new-dimensions-with-arcmap

Also worth noting is that outside the domain of GIS, the use of magnetic resonance (MRI) and computer tomography (CT) equipment has greatly increased the availability of acquired 4D raster images on which computational methods are often applied [Lorenzo-Valdés et al., 2004]. Similar methods are also applied on numerical simulations generated in other fields, such as the computation of topological invariants in metallurgy [Kaczynski et al., 2004].

4.3.2 Hierarchical subdivisions using trees

Higher-dimensional hierarchical subdivisions using trees also work in a similar fashion as their 2D and 3D counterparts. An n-dimensional domain is specified, which generally has a simple shape. It is then subdivided using 'partition' primitives, which attempt to split space incrementally in an adaptive manner—each primitive recursively partitioning the space defined by its parent so that the shape of the cells in the subdivision is only known after traversing a path in the tree from its root to a leaf representing the last relevant partition primitive. Even though this means that it is necessary to traverse many nodes in order to extract a single cell, this makes it possible to use a custom partition at every step and so store cells of varying sizes more efficiently.

Many different structures with only small variations have been created for this purpose, of which hyperoctrees, bintrees and k-d trees are by far the most widely used. The simplest possibility is to recursively subdivide space along all dimensions homogeneously at their midpoints at every partition primitive, so that one partition primitive subdivides nD space into 2^n regions. Hyperoctrees [Yau and Srihari, 1983] and k-trees [Jackins and Tanimoto, 1983] do this, being identical dimension-independent generalisations of quadtrees [Finkel and Bentley, 1974] in 2D and octrees [Meagher, 1980] in 3D. Bintrees [Samet and Tamminen, 1985] instead split dimensions alternately rather than all at one, while k-d trees split dimensions at specified points rather than at their midpoints.

Hierarchical subdivisions using trees have also been previously used in a higher-dimensional GIS setting, usually combined with exhaustive enumeration approaches. Varma et al. [1990] defined HHCodes in order to perform efficient topological queries on abstracted 4-orthotopes, such as whether a pair of them overlap both spatially and temporally. O'Conaill et al. [1992] aggregated 4D raster images using trees, permitting the efficient storage of large sparse datasets. Mason et al. [1994] used 4D bintrees in order to index and perform fast queries and operations (e.g. interpolation and generalisation) on the aforementioned 4D grid of ocean temperatures.

4.3.3 (Implicit) vector models

Vector representations of nD objects avoid many of the problems present in exhaustive enumeration and tree-based subdivisions: they are generally more compact, can describe boundaries more precisely and can represent objects with their attributes directly. This makes them particularly interesting for higher-dimensional GIS, even if they have hardly been explored as the data structures commonly used in 2D vector GIS do not extend naturally to higher dimensions.

Among the possible vector-based structures, the ones that most naturally extend to higher dimensions are the more implicit models consisting of primitives that can be defined using a set of simple parameters. This includes many of the models discussed in §3.1.2, such as higher-dimensional versions of primitive instancing, sweep representations, constructive solid geometry, Boolean set operations on half-spaces and Nef polyhedra.

For instance, Paoluzzi et al. [2004] describes a scheme of dimension-independent Boolean operations on half-spaces. It is able to do these operations by combining the half-space constraints of multiple objects. Obtaining such a representation of an object

can be achieved by performing (alternate) decompositions into convex parts [Bulbul and Frank, 2009; Lien and Amato, 2006].

This type of implicit approach might be especially useful in certain contexts, such as production processes and the computation of homologies [Damiand et al., 2008]. However, these are less capable of storing objects and their attributes *directly*, especially those of lower dimensionality, and therefore have to be generally 'evaluated' into another, more explicit model in order for them to be visualised [Mäntylä, 1988] or to perform the type of computations expected in GIS, such as geometric intersection operations. They are therefore less interesting for a higher-dimensional GIS.

Nevertheless, a possible future implementation of Nef polyhedra in arbitrary dimensions deserves an additional mention. As shown in §3.1.2, a set of Nef polygons can be reduced to a structure of sets of circular intervals and a set of Nef polyhedra to a structure of sets of faces on a sphere. This approach extends naturally to higher dimensions as the method can be recursively repeated in decreasing dimension in order to have a complete description of a higher-dimensional Nef polytope. However, the practical benefits of doing do decrease as the dimension increases and the navigation becomes more cumbersome. It is also very complex to implement this in a fully dimension-independent manner, requiring a hyperspherical projective kernel capable of representing cells of any dimension embedded on higher-dimensional spheres which is also capable of computing point-set intersections in any dimension. To the best of my knowledge, no such system has every been implemented, and it is unlikely to be implemented in the near future.

4.3.4 Geometric simplicial complexes

As described in §2.3.2, an i-dimensional simplex (i-simplex) is a combinatorial primitive made from a set of $i + 1$ vertices, and an j-simplex, $j \leq i$, made from a subset of these is an j-dimensional face (j-face) of it. An n-dimensional *geometric* simplicial complex, also known as a Euclidean simplicial complex, is a subdivision of n-dimensional space into a structure of closed n-simplices directly embedded in space by attaching a tuple of coordinates to each vertex, in a manner that their interiors are non-overlapping geometrically (pairwise disjoint). In an n-dimensional simplicial complex, an i-dimensional object, $i \leq n$, can be represented as a set of i-simplices. This allows the representation of objects of any dimension in a geometric simplicial complex.

An i-simplex is easily described by an $(i+1)$-tuple of $i+1$ affinely independent[67] vertices $(v_0, v_1, \ldots, v_i)$. Since they are affinely independent,

67: That is, that none of them are in the affine space described by the others. In practical terms: two points with a different coordinate in 1D, three non-collinear points in 2D, four non-coplanar points in 3D, etc.

the geometric realisation of a simplex is the convex hull of its vertices. A vertex in an n-dimensional simplicial complex can be described by an n-tuple of coordinates $(x_0, x_1, \ldots, x_{n-1})$, representing its embedding into n-dimensional Euclidean space. In a subdivision of n-dimensional space, every simplex in the simplicial complex is a face of at least one n-simplex.

Simplicial complexes have a simple but powerful structure and are easy to navigate. Two i-simplices are adjacent if they have a common face, and an i-simplex and a j-simplex, $i \neq j$, are incident if either is a face of the other. In an n-dimensional simplicial complex, each n-simplex is adjacent to at most $n+1$ other n-simplices and has exactly $n+1$ $(n-1)$-faces on its boundary. To each n-simplex it is therefore possible to apply $n+1$ different boundary and neighbour operators, where the n-th boundary operator returns the $(n-1)$-face of the n-simplex on the opposite side of the n-th vertex—obtained by removing the n-th element of the tuple of vertices—, and the n-th neighbour operator returns the adjacent n-simplex on the opposite side of the n-th vertex, if any, which is also incident to the face given by the n-th boundary operator.

The biggest challenge of using a geometric simplicial complex for the representation of nD objects is the difficulty of triangulating the possibly complex nD objects being modelled, i.e. algorithmically dividing them into non-overlapping geometric simplices so that the union of the simplices representing an object corresponds exactly to its original geometry. While triangulations of sets of points in nD have been studied extensively and there is robust software to compute them, e.g. Quickhull [Barber et al., 1996], such triangulations are generally not applicable to nD objects since a triangulation of the vertices of an object usually results in simplices that do not conform to the object's boundary, i.e. that are partially inside and partially outside the object.

However, it is possible to make sure that the boundaries are part of the triangulation through the use of a constrained or conforming triangulation. Both of these force the triangulation to include certain simplices as part of the triangulation in the form of *constraints*, which in this case would be the objects' boundaries. In a *constrained triangulation*, the constraints are directly part of the structure of the simplicial complex. While the properties of nD constrained triangulations have been described [Shewchuk, 2008] and algorithms to construct them in some cases have been developed [Shewchuk, 2000], their construction in the general case is very difficult in practice, having to deal with robustness issues and possibly requiring the addition of a large number of additional vertices (Steiner points) and subsequently additional simplices, as not every object of dimension three or higher is triangulable without additional vertices [Ruppert and Seidel, 1992], such as the Schönhardt polyhedron shown in Figure 4.8. In fact, it is already an NP-hard

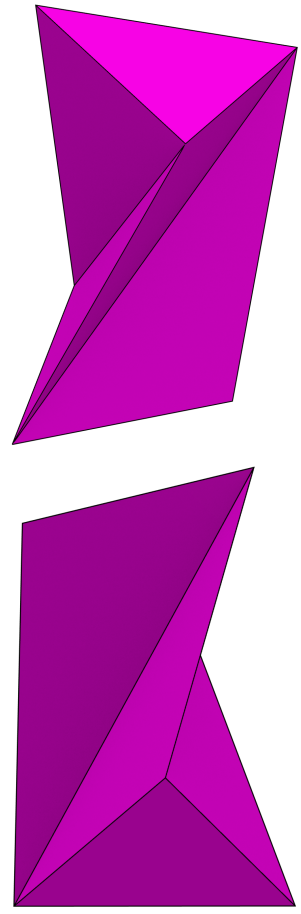


Figure 4.8: Two views of the Schönhardt polyhedron [Schönhardt, 1928]—the archetype of a polyhedron that cannot be tetrahedralised without the addition of Steiner vertices. Starting from a triangular prism whose rectangular faces have been triangulated, it can be created by rotating the triangular faces with respect to each other. Note how any tetrahedron built from the 3 vertices of any of its faces plus any other vertex of the polyhedron necessarily passes through the outside of the Schönhardt polyhedron.

problem to determine whether a polyhedron is tetrahedralisable without adding Steiner points [Ruppert and Seidel, 1992]. Deciding where to best add these additional simplices in nD seems to be the hardest part of this problem[68].

68: In the opinion of Jonathan Shewchuk.

A *conforming triangulation* forces the triangulation to contain some constraints as well, but instead of adding them directly as simplices of the triangulation, it allows them to be split into multiple smaller simplices. This allows a conforming triangulation to fulfil other geometric criteria such as the Delaunay property (i.e. the circumsphere of a simplex does not contain any other vertex).

Unfortunately, while there are robust and reliable constrained and conforming triangulation libraries existing in 2D and 3D, such as Triangle in 2D [Shewchuk, 1996a], CGAL in 2D [Boissonnat et al., 2002], and Tetgen in 3D [Si and Gärtner, 2005], there seems to be no software capable of performing a constrained or conforming triangulation in more than 3D.

Assuming that the objects have been triangulated successfully, managing holes and disconnected components in a simplicial complex becomes trivial since these become integrated into the combinatorial structure of the triangulation, as will be discussed in §10.2. Generally, a triangulation algorithm works initially on the basis of a set of points and ensures that the entire convex hull of all of the points is triangulated, something that can be done using the concept of a 'big triangle' or faraway point [Liu and Snoeyink, 2008], and therefore connects formerly disconnected components or components that were not connected by using new simplices (n-simplices in an nD simplicial complex). Similarly, as the interiors of polytopes with holes are triangulated, additional simplices connect the void regions representing holes with the rest of the structure. In this manner, 2D holes can be represented as sets of triangles labelled as being empty, 3D holes as similar sets of tetrahedra, and so on for higher (and lower) dimensions. This ensures that it is always possible to navigate between all the simplices in the complex.

Another advantage of simplicial complexes is that many operations on them are efficient and easy. For instance, a simplicial complex can be traversed efficiently without any external data structure. Starting from any simplex, it is possible to 'walk' the simplicial complex in the desired direction in order to find any given simplex, e.g. inside which simplex a point is located [Devillers, 2002], with each walking step taking only constant time. Comparing objects composed of sets of simplices is also trivial, assuming that they have been triangulated in a deterministic way[69], since they can be compared one by one (in order) after finding a common simplex, which can be done by walking the respective simplicial complexes. Checking if two objects are adjacent or intersect geometrically are examples of other operations that can be done simply and efficiently simplex by simplex.

69: Note that triangulating objects deterministically is not so simple. If the triangulations are not deterministic, such a comparison could possibly require geometric operations.

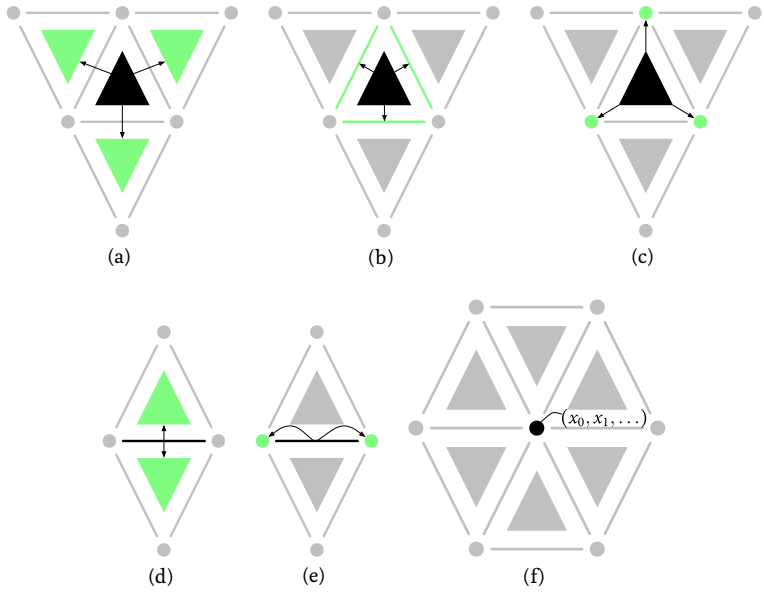


Figure 4.9: Some of the possible relationships between primitives and the storage of a geometry using a 2-simplex (triangle) based data structure. (a) Adjacency relationships of a 2-simplex. (b) Boundary relationships of a 2-simplex. (c) Vertices of a 2-simplex. (d) Star (co-boundary) relationships of a 1-simplex. (e) Boundary relationships of a 1-simplex. (f) Embedding of a 0-simplex. In addition to these, it is also possible to store attributes for the 0-, 1-, and 2-simplices in embedding structures. The adjacency model uses at least (a), (c) and (f), while the incidence model uses at least (b), (e) and (f), plus usually (a) to navigate the structure. Many other combinations are possible.

Since the number of incident vertices, bounding $(n-1)$-faces and adjacent simplices of an n-simplex in an n-dimensional simplicial complex is fixed and known, it is easily represented in a computer using simple arrays, rather than linked lists or other variable-length data structures. This can be done through an n-simplex-based data structure, shown in Figure 4.9 for 2D, where n-simplices are the main primitives. Related data structures are often said to to follow the adjacency or incidence model, depending on the dimensions of the simplices that are represented in the structure—only the highest-dimensional ones in the adjacency model, but possibly all in the incidence model.

An n-dimensional simplicial complex is thus represented as a collection of primitive n-simplices, each containing $n+1$ links to its adjacent n-simplices. For the models described above, it also includes information containing:

Adjacency model For every n-simplex, $n+1$ links to vertex primitives containing the coordinates and attributes of its $n+1$ vertices, or alternatively this information stored directly in the n-simplex. The simplices of dimensions 1 to $(n-1)$ and the incidence relationships between them, are therefore not represented explicitly.

Incidence model For every i-simplex, $\forall 0 < i \leq n$, $i+1$ links to primitives containing the geometry and attributes for its $(i-1)$-faces, or alternatively this information stored directly in the i-simplex. All simplices of every dimension, together with all

their incidence relationships, can therefore be represented explicitly, or as part of a simplex of a higher-dimension.

The adjacency model thus forms an $(n+1)$-regular graph with the n-simplices as nodes, i.e. a graph where every vertex has degree $n+1$. The incidence model is a graph with a maximum degree $n+1$, where the degree of a node representing an i-simplex is $i+1$. Note that the adjacency model is therefore more compact, but only allows for the explicit representation of n-dimensional objects, while the incidence model is more verbose but supports the representation of objects of any dimension. This is done using the embedding information for the simplices of dimension lower than n. Note that non-linear geometries (e.g. NURBS) can be represented as well using the embedding structures for the simplices of dimensions one and higher.

Additionally, since a simplex is also a cell, it is possible to use any data structure meant for more general cell complexes, such as those described in §4.3.5 and §4.3.6. This is commonly used in practice during the process of triangulating a cell complex (since at that point the structure is not a simplicial complex and cannot by definition be stored in a data structure for simplicial complexes only), but it is a very inefficient approach when it has already been triangulated.

Simplicial complexes in more than 3D are commonly described in theory but rarely put in practice. Paoluzzi et al. [1993] described the winged scheme in order to describe arbitrary polytopes as simplicial complexes, and Karimipour et al. [2010] uses lists of vertices to store simplices of any dimension in order to do spatial analysis using Delaunay triangulations as an example.

4.3.5 Cell complexes

For the purposes of a higher-dimensional data structure, cell complexes differ from simplicial complexes in two main related aspects: (i) there is a variable number of facets on the boundary of a cell, each of which has also a different number of facets, continuing recursively down to the vertex level; and (ii) obtaining a geometric interpretation of a cell requires looking recursively at its facets[70]. This causes cell complexes to have a much more complex representation and algebra to manipulate them compared to simplicial complexes. For instance, simplicial complexes can be easily constructed by adding and removing simplices expressed by sets of vertices, or by splitting and merging existing ones, but cell complexes require much more complex operations in order to add or remove cells—something that will be described in detail in Chapter 7. Traversing a cell complex (e.g. to find in which cell a point lies or to see which

70: Or at some similarly complex recursive representation, such as with Nef polyhedra [Bieri and Nef, 1988].

cells intersect a given geometry) is also much more difficult [de Berg et al., 1997].

However, the flexible structure of a cell complex allows it to represent objects either directly or with minimal preprocessing. As long as an i-dimensional object is homeomorphic to an i-ball—or can be modified so that it is functionally so by using simple 'tricks' at the combinatorial level—, it can be modelled as an i-cell in the complex with attached attributes. Also, the lower number of cells in a cell complex compared to the simplices in a simplicial complex representing the same set of objects means that they can store objects more efficiently, as long as an adequate data structure is used.

Cell complexes are usually modelled based on techniques that represent them based on their $(n-1)$-dimensional boundary (i.e. boundary representation), unlike simplicial complexes in which objects are usually modelled using mainly primitives of the same dimension (i.e. object representation, but better known by its 3D name as solid or volume representation). This is a subtle distinction, but it implies that cells are described (and stored) in a more implicit manner based on a recursive definition, unlike simplices which can be stored explicitly as a tuple of vertices. Since this can slow down many operations, access to cells can be improved by adding indexing structures storing links to all the cells of a given dimension and/or to all the cells in a given spatial extent. On the other hand, the use of data structures for the cells of every dimension mean that these can be easily used to store attributes.

The most common data structures for arbitrary cell complexes are incidence graphs and other related structures [Rossignac and O'Connor, 1989; Masuda, 1993; Sohanpanah, 1989], which store all the cells (of every dimension) in a complex as individual embedding primitives containing geometric information and attributes. Just as with simplicial complexes, if only linear geometries are required, the only geometric information needed is the coordinates of the points for the 0-cells. However, since a set of vertices is not enough to describe a cell, all the embedding structures of every dimension need to exist (albeit they can be as simple as a unique ID or memory address per cell). As shown in Figure 4.10, these are connected through a combinatorial structure containing the incidence relationships between the cells, such as an i-cell having an $(i-1)$-cell on its boundary, vice versa (known as the co-boundary or star), or both. Since the number of these incidences is generally not fixed or bounded, unlike in a simplicial complex, they are more difficult to integrate into the cell primitives. One solution involves using variable length data structures (e.g. a linked list of the facets of a cell). Another would be to store instead combinatorial primitives consisting of pairs of an i-cell and an $(i-1)$-cell—an approach well-suited to a database implementation.

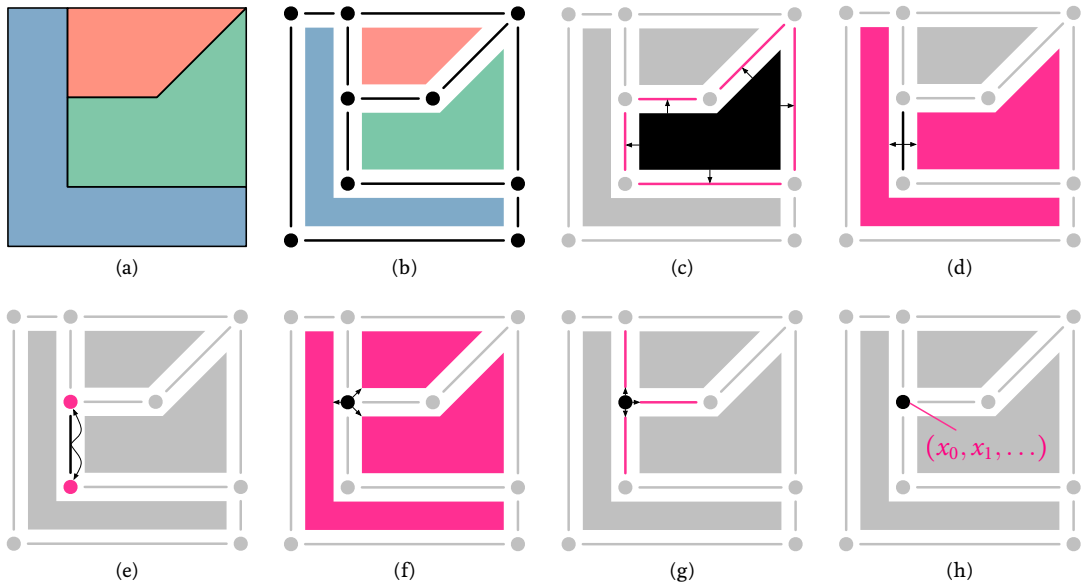


Figure 4.10: The cells in a cell complex, the most common relationships that are be stored in an incidence graph and similar data structures up to 2D, and the embedding of the 0-cells so as to support a geometry. (a) Three adjacent polygons. (b) The 0-, 1- and 2-cells in the cell complex. (c) Boundary relationships of a 2-cell. (d) Co-boundary relationships of a 1-cell. (e) Boundary relationships of a 1-cell. (f) Double co-boundary relationships of a 0-cell. (g) Co-boundary relationships of a 0-cell. (h) Embedding of a 0-cell. Note that (c), (e) and (h) correspond to the incidence graph, a minimal structure where every i-cell is linked to the $(i-1)$-cells on its boundary.

While incidence graphs are efficient in terms of space, they are difficult to navigate and manipulate because of the limited topological information contained in them. This greatly complicates many relatively simple queries that are trivial in simplicial complexes, such as: obtaining the order of a sequence of 0- and 1-cells around a 2-cell; or checking if two n-cells have the same geometry (since their facets, each represented implicitly by its own facets, might be stored in any order).

One more possibility involves limiting the representation to cell complexes with stronger algebraic properties, such as those with a manifold domain. Most importantly for a spatial data model, manifolds have two very useful properties:

- An n-manifold with boundary has the property that its boundary is a $(n-1)$-manifold, ensuring that an n-dimensional object can be represented recursively by its $(n-1)$-dimensional boundary using a manifold data structure.
- An $(i-1)$-cell is incident to at most two i-cells within an $(i+1)$-cell[71], limiting the number of links required between primitives in a boundary representation based model of the complex.

71: This is the definition of a quasi-manifold, a combinatorial interpretation of the topological concept of a manifold. Quasi-manifolds are the same as manifolds in up to two dimensions, but this is not necessarily true in higher dimensions.

Some data structures for cell complexes in 2D and 3D are thus restricted to manifolds, or split objects into manifold sections [Pesco et al., 2004; Lopes and Tavares, 1997], which allows such data structures to be more efficient as they need to handle simpler configurations [Aguila and Ramírez, 2003]. These are subsequently linked in a special structure that allows one to navigate around these 'non-manifold parts' [Weiler, 1988; Gursoz et al., 1990; Lee and Lee, 2001]. In higher dimensions, if the representation is restricted to a n-dimensional cell complex to manifold domains for each cell, an $(n-2)$-cell is incident to at most two $(n-1)$-cells, and thus $(n-2)$-cells can be used as primitives or pointed to while being able to navigate around them.

One major difficulty with data structures based on cell complexes is dealing with holes and disconnected objects, both of which would ordinarily result in a disconnected graph using any of the data structures as described above. Since n-cells in a cell complex are assumed to be homeomorphic to open n-balls, holes violate the mathematical definition of a cell complex. However, an n-dimensional hole that is homeomorphic to an n-ball in an n-dimensional cell can be generally handled in an ad hoc manner using any of techniques shown in Figure 4.11 and described as follows:

Splitting cells The i-cell containing an i-dimensional hole is subdivided into multiple i-cells, all of which are adjacent to it. This solution is conceptually simple and is effectively equivalent to what is done with simplicial complexes. This also best respects the mathematical definition of a cell, since the subdivided i-cells can be homeomorphic to i-balls and the hole is either non represented (when the dimension of the complex is higher than i and the hole is not completely bounded by other i-cells) or is represented as a cell like any other but labelled as being empty. However, finding a way to split a cell accordingly requires geometric operations and can be difficult in practice, especially in dimensions 3 and higher.

Bridge cells A hole is connected to the rest of the combinatorial structure using a 'bridge' cell of lower dimensionality than the hole. This bridge cell usually takes the form of an edge connecting the hole with the cell where it is contained. This solution is generally simpler to implement, usually requiring only the finding of an appropriate bridge (e.g. an edge that lies entirely in the interior of the cell). While the cell containing the hole is no longer homeomorphic to a ball, most computations work with few or no changes [Bryant and Singerman, 1985]. However, some queries might need the addition of special cases to handle it (e.g. not displaying such an edge when visualising the cell or accounting for the fact the bridge will have the same cell on all sides in topological computations).

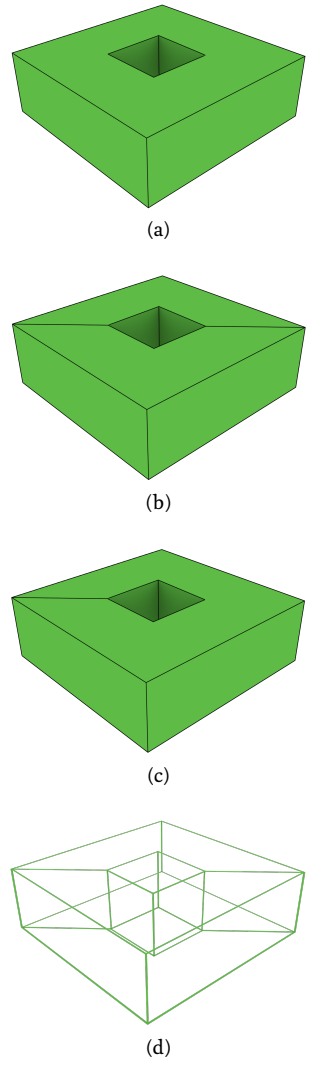


Figure 4.11: Two different techniques to handle holes in (a) a 3D cell complex representing a torus: (b) splitting the 3-cell into two parts and (c) using a bridge edge. (d) Note how these techniques allow for a hole to be representing while the combinatorial structure forms a single connected component (i.e. all cells form a single interlinked group).

Holes as attributes Holes can be stored as fully independent cells, linked as special attributes of the objects inside of which they are, or vice versa, something that can be extended so as to have full hierarchies of objects in a tree [Worboys, 2012]. This solution might be the simplest to implement in terms of storage, but it goes against the mathematical definition of a cell complex and will break or require a complex special treatment for geometric and topological computations. It is also very simple to inadvertently create invalid holes in this manner, such as holes lying (partially) outside the cell that is supposed to contain them.

The latter two described methods can be applied for disconnected components as well. Two disconnected components can be connected by a bridge cell, or all components can be considered as holes of the (empty) universe cell.

It is worth noticing that complex holes that are not homeomorphic to n-balls can exist, e.g. tori-shaped holes inside a volume, and finding a good subdivision that can be used to deal with such a hole might be very complex.

In terms of applications, cell complexes in more than 3D have been rarely described or implemented. Hazelton et al. [1990] described how incidence graphs can be put into a relational database schema in order to store 4D polytopes, on which 4D topological relationships can be queried [Hazelton et al., 1992]. However, to the best of my knowledge, this has never been put into practice in a working system.

4.3.6 Ordered topological models

Ordered topological models are a representation method that combines characteristics of both simplicial complexes and cell complexes, and are therefore commonly grouped with either of them [de Floriani and Hui, 2005; Čomić and de Floriani, 2012]. They work on the basis of a cell complex, but subdivide each cell into *abstract* simplices using a purely combinatorial operation, a concept derived from what is known as an abstract simplicial complex in algebraic topology.

The exact subdivision that is used depends on the data structure, but as shown in Figures 4.12 and 4.13, it is similar to a barycentric triangulation of a convex polytope, consisting of abstract vertices representing cells of different dimensions (i.e. the vertices of such a simplicial complex are not equivalent to only 0-cells), and simplices that connect these vertices in a specific (ordered) manner. The vertices representing the cells of dimensions higher than zero do not have a well-defined location in space (although they are usually depicted lying somewhere in the cells they are supposed to represent),

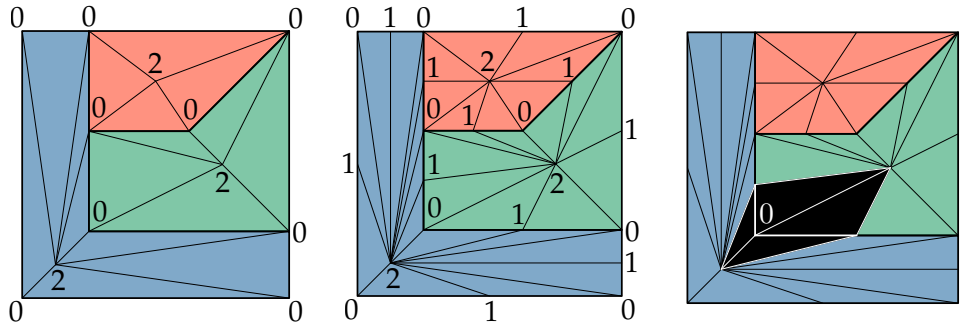


Figure 4.12: The simplicial decomposition applied in order to obtain the simplices for a 2D: (a) generalised map and (b) combinatorial map. (c) All cells are represented as sets of 2-simplices, such as the vertex shown here, which is formed of a set of 4 triangles. Note that the locations of the vertices for the 1- and 2-cells are arbitrary, those for the 1-cells has been defined as the midpoint of the edge, and those for the 2-cells have been set so that the resulting simplices lie entirely in the interior of their corresponding 2-cell.

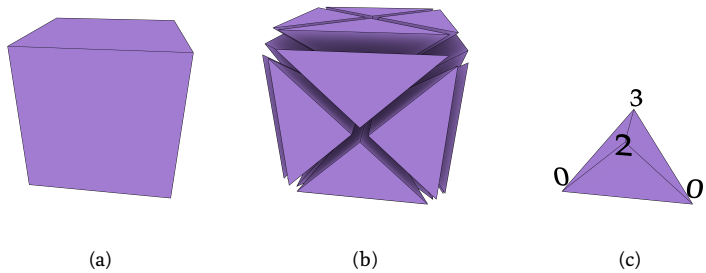


Figure 4.13: (a) A cube is represented as (b) a set of simplices in a 3D combinatorial map. (c) Each simplex has two vertices at two of the original vertices of the cube, as well as two other symbolic vertices that respectively represent the face and volume that they belong to.

but the simplices nevertheless form a combinatorially correct subdivision of the space.

Adjacency and incidence do have a meaning, albeit not the same as in a geometric simplicial complex, even if their mathematical definition is just as in a geometric simplicial complex. Considering that a j-face of an i-simplex, $j \leq i$, is a j-simplex made from a subset of the vertices of the i-simplex, two i-simplices are adjacent if they have a common face, and an i-simplex and a j-simplex, $i \neq j$, are incident if either is a face of the other.

In an ordered topological model, objects are represented by sets of simplices, similar to how they are represented in a geometric simplicial complex. However, unlike in a geometric simplicial or cell complex, where an i-dimensional object is represented as a set of i-simplices or i-cells, in an n-dimensional ordered topological model an object of *any* dimension is represented as a set of n-simplices. More concretely, a cell is represented as the set of simplices that have a symbolic vertex at that cell. In this manner, even vertices

are represented as (possibly large) sets of simplices[72].

72: Geometrically, this set would consist of all the n-simplices incident to the vertex.

Attributes and geometry can be attached to the simplicial complex in an ordered topological model in a similar way as in a geometric simplicial complex. Since the symbolic vertices of each simplex have a known order, and each vertex of each simplex represents a cell, it is possible to link each simplex in an ordered topological model to a tuple of attributes and geometric information for the cells that its vertices represent. Many operations that are possible on a geometric simplicial complex are also possible in an ordered topological model, even considering that many of its vertices do not have a specific location in space. Topological queries can be handled in an identical way, and some geometric operations require minimal changes. For instance, the length, area, volume, etc. of a cell, can be computed using the inclusion-exclusion principle, which works similarly to the computation of the area of a polygon using the 'shoelace formula' [Meister, 1771]. Comparing objects can also be done simplex by simplex [Gosselin et al., 2011] taking advantage of the ordering properties of ordered topological models.

Since this simplicial decomposition performed in an ordered topological model is done only combinatorially, it is not necessary to build a constrained or conforming triangulation of the input, which as stated in §4.3.4, might be very difficult to accomplish. However, using an ordered topological model still allows one to use the stronger algebraic properties present in a simplicial complex, which are used in order to traverse and manipulate a cell complex in a more efficient manner than it would otherwise possible be. The resulting simplicial complex might have a larger number of simplices compared to one triangulated geometrically, but it also has the advantage that the number of vertices and simplices in it is known in advance and does not depend on the geometry.

There are two different simplicial decomposition schemes used in ordered topological models, one for the cell-tuple structure [Brisson, 1989] and generalised maps [Lienhardt, 1994], and another for n-dimensional combinatorial maps [Lienhardt, 1994]. Starting from an input n-dimensional cell complex that is in the form of an incidence graph containing: (i) all the cells (of every dimension) as nodes and (ii) the incidence relationships from each i-cell to the $(i-1)$-faces on its boundary as directed edges, the simplicial decompositions are shown in Figure 4.12 and explained as follows:

Cell-tuple / generalised maps Each possible path in the graph from an n-cell to a 0-cell directly yields the vertices of one simplex. Each n-simplex can be thus defined by a unique n-tuple $(c_0, c_1, \ldots, c_n)$, where c_i is a vertex representation of an i-dimensional cell in the cell complex, all of which are incident to each other. Such an n-tuple is in fact equivalent to an n-simplex. It can be seen as a generalisation of the lath-based structures [Joy et al., 2003] sometimes used in 2D and 3D GIS.

***n*-dimensional combinatorial maps** Each possible path in the graph from an n-cell to a 1-cell yields one simplex, with the vertices for the cells from dimension n to 2 used directly, prepended with the two vertices corresponding to the two 0-cells incident to the 1-cell (i.e. the vertices at the endpoints of the edge). Each n-simplex can be thus defined by an n-tuple $(c_0, c'_0, \ldots, c_n)$, where c_0 and c'_0 are the two 0-cells incident to the 1-cell, and for $i > 0$, c_i is a vertex representation of an i-dimensional cell in the cell complex, all of which are incident to each other and to the two 0-cells. As the two 0-cells could be given in any order, the order is set by specifying an orientation for the entire cell complex (or for each connected component when there are more than one) by specifying which vertices are respectively c_0 and c'_0 in one simplex, and then building the n-tuples for the other simplices so that adjacent simplices have them in the opposite order. As Figure 4.14 shows, this can result in two possible orientations for each cell in a combinatorial map (or connected component). While not shown here, this orientation determination scheme works in any dimension. In practice, this means that if one considers the edges bounded by c_0 and c'_0 as being directed (from c_0 to c'_0), surfaces form loops of directed edges, and opposite surfaces have opposite orientations, just as in most 2D/3D structures that are based on half-edges, e.g. DCEL [Muller and Preparata, 1978].

The cell-tuple structure [Brisson, 1989], and generalised maps and n-dimensional combinatorial maps [Lienhardt, 1994], support this model directly. The **cell-tuple structure** was developed as a generalisation of the quad-edge [Guibas and Stolfi, 1985] data structure in 2D and the facet-edge data structure [Dobkin and Laszlo, 1987] in 3D. In it, an n-simplex is called a *cell-tuple*, the neighbour operator is called *switch*, and the set of cell-tuples that represent a cell c is called the *set of associated cell-tuples* of c. **Generalised maps** and n-dimensional **combinatorial maps** [Lienhardt, 1994] evolved instead as a generalisation of 2D combinatorial maps [Edmonds, 1960]. They were extended to support open (topologically closed) maps in Poudret et al. [2007], so as to support empty space around objects without representing it explicitly. An n-simplex is called a *dart*, the neighbour operator is called α (for generalised maps) or β (for combinatorial maps), and the set of darts that represent a cell is called its *orbit*. These were developed independently, with generalised maps being mathematically shown to be able to represent a wider class of objects than the cell-tuple, but both work out to very similar or identical implementations in practice. Chains of maps [Elter and Lienhardt, 1994] supplement the approach followed by generalised maps with an incidence graph in order to support non-manifolds, but they are rarely used because of their extremely high space requirements.

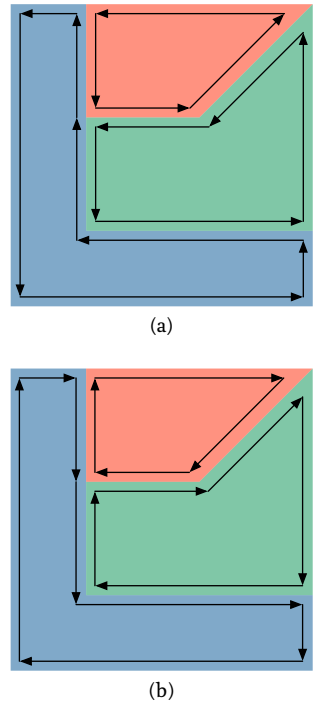


Figure 4.14: There are two possible consistent orientations that can be set for each connected component in a combinatorial map.

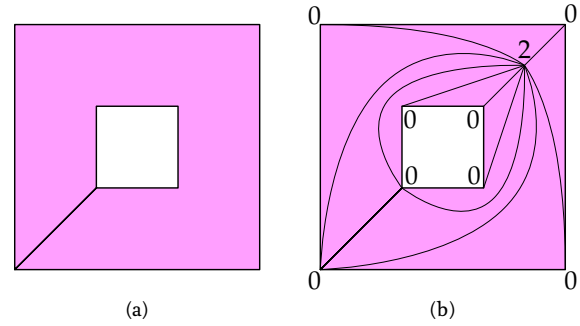


Figure 4.15: (a) A polygon with a hole is represented using a bridge edge to connect the hole with the exterior boundary. (b) The simplices in the combinatorial maps representation of the situation in (a). There are two simplices with the same vertices representing each side of the bridge edge. As in Figure 4.12, the location of the vertex for the 2-cell shown in (b) is arbitrary but chosen so as to lie in the interior of the polygon. The combinatorial triangles are shown with curved lines so as to form a planar embedding to show that they effectively form a partition of the space.

It is worth mentioning that ordered topological models are not significantly more efficient in terms of space than the Simple Features-like [OGC, 2011] approach shown in §4.2.3 and Figure 4.7. Every dart in a combinatorial map is equivalent to a consecutive pair of vertices in the Simple Features approach, while a dart in a combinatorial map is equivalent to two darts in a generalised map or to two cell-tuples. However, ordered topological models consist of combinatorial primitives that can be traversed efficiently, unlike the Simple Features approach in which many operations involve brute force comparisons.

Holes and disconnected components in an ordered topological model can be dealt with any of the techniques described for cell complexes in §4.3.5, with bridge edges modelled as a pair of simplices with the same vertices but opposite orientations as shown in Figure 4.15. Attributes can be handled using an attributes tuple that links each cell of the vertices tuple to a corresponding structure to store an attribute for it, such that if the i-th element of the vertices tuple represents a cell, the i-th element of the attributes tuple can link to a structure storing the attributes of the cell.

While ordered topological models are relatively difficult to implement, they have nevertheless been implemented in several open source libraries and software, including CGAL[73], Moka[74], and CGoGN[75] [Kraemer et al., 2014]. Although not discussed further here, I showed in Arroyo Ohori et al. [2013] in more detail how generalised maps can be implemented to support various characteristics of real-world data, such as such as markers, locks (for parallel algorithms), simple construction operations and indexing of darts and attributes for the objects of all dimensions.

However, even as all of the aforementioned implementations are stable, and the implementation of CGAL Combinatorial Maps is impressively sophisticated in many respects (such as the customisable constant time iterators through darts implemented using C++11

73: http://www.cgal.org

74: http://moka-modeller.sourceforge.net

75: http://cgogn.unistra.fr

variadic templates), it is important to note that they are limited to providing basic navigation and construction functions at the dart/simplex level. The contribution of this thesis thus lies in showing how to use this type of libraries in practice, how to build on them to create higher-dimensional representations of real-world objects, and how one can create more complex algorithms on top of these basic operations in order to implement higher-level (and possibly user-facing) functions.

In terms of applications, Fradin et al. [2002] used generalised maps in architecture building models such that the level of detail is modelled using the same kind of relations that are used to model geometric dimensions. Thomsen et al. [2008] used generalised maps in order to manage topology in 2D/3D GIS in a unified (and dimension-independent) manner. Le [2013] used Gocad[76] (which internally uses generalised maps) to create 4D geological models by interpolating separate 3D models at various points in time.

76: http://www.gocad.org/

4.3.7 Other possible vector models

While this thesis attempts to comprehensively cover the ways in which higher-dimensional objects might be modelled, its principal aim is to *realise* one of these models up to its implementation and basic operations. Because of this—and especially considering the great number of possible models developed in different fields—, there are interesting possibilities that it has not been possible to study in depth within the context of the PhD project corresponding to this thesis. Nevertheless, as a documentation of these preliminary observations, some other representation possibilities are presented in brief in this section.

Models containing only points in nD space [Pasko and Adzhiev, 2001] are widely used in the context of remote sensing and data mining, and are particularly relevant in the representation of nD fields. These can be used together with an appropriate interpolation function [Miller, 1997] or a geometric construction that generates nD cells from the points (e.g. Voronoi diagrams [Ledoux, 2006] and their weighted generalisations [Edelsbrunner, 2014]).

Apart from the hierarchical subdivision approaches presented in §4.3.2, there are other types of hierarchical representations that do not recursively partition space starting from $\mathbb{R}^n$. Čomić et al. [2014] discusses how objects can be described as sequences of operations in the form of *hierarchical cell complexes*, which moreover preserve the homology of the objects. Bulbul and Frank [2009] decomposes objects into simplices using alternate decompositions using convex hulls—an especially interesting approach considering that nD convex hull computations have been extensively described in scientific

literature and implemented in software [Lawson, 1986; Seidel, 1986; Barber et al., 1996].

Algebraic models, such as the function representation (F-rep) scheme [Pasko et al., 1995] described in §3.1.2, also extend well to higher dimensions. For instance, geometric algebra [Artin, 2011] can be used to describe objects directly as vector spaces, eschewing separate combinatorial/embedding structures for a single representation where the geometry and topology of an object is jointly manipulated and queried.

4.4 Conclusions and possibilities

Using higher-dimensional representations of objects offers a solution to the integration of parametrisable characteristics, such as time and scale, into GIS in a generic manner. This makes it possible to store all the topological relationships and the correspondences between objects across (discrete or continuous) time or scale by modelling them as if they were continuous, and can be used for several applications, such as for guaranteeing the internal consistency of a model, for the definition of dimension-independent constraints, or for the automatic update of datasets [van Oosterom and Stoter, 2010].

The approach is well-founded upon long-standing mathematical theories, and so is able to deal with complex cases and with the special requirements of each dimension more robustly than solutions in which parametrisable characteristics are handled in an ad hoc fashion. For instance, if moving objects were to be added to an existing representation (with only static objects), arbitrary motions could be supported rather than those that can be deduced from simple parameters stored in attributes.

While higher-dimensional representations are space intensive, this factor ameliorated by the fact that many GIS applications use relatively sparse data [McKenzie et al., 2001; Mason et al., 1994]. However, analysing the properties of the non-spatial characteristics to be modelled as dimensions is crucial, as there is no advantage in modelling such characteristics as dimensions when attributes clearly suffice.

There are many different data models and data structures that can be used to implement this *higher-dimensional spatial modelling* approach. However, just as in 2D and 3D, there is not a single data structure that works best for all purposes and applications, so choosing an appropriate one is critical.

4D rasters and hierarchies of trees have been used successfully in the past in a GIS context [Mason et al., 1994; Varma et al., 1990; Bernard et al., 1998; O'Conaill et al., 1992]. These can use similar structures

and algorithms as their 2D/3D counterparts and are therefore easy to implement, but they are only capable of achieving rough approximations of the objects' boundaries [Blaschke, 2010] and can grow in size exponentially as the dimension increases. Vector data structures with limited topological information, such as Simple Features [OGC, 2011], possibly combined with the computation of topology on the fly [ESRI, 2005], also explode in terms of size in higher dimensions and are difficult to efficiently traverse. The most promising option is therefore a topological vector approach that allows for the representation of real-world objects of heterogeneous dimensions with holes and attributes, which moreover should be efficient to traverse. This requires the use of higher-dimensional topological vector data structures.

The dimension-independent vector data structures that are able to support this approach can be classified into three broad data models: geometric simplicial complexes, cell complexes and ordered topological models. The data structures belonging to each of these tend to have similar properties in terms of space complexity since they contain a number of primitives of the same order[77]. Operations performed on them also tend to have the same computational complexity since they can have similar topological relationships, and when they do not, their cost is bounded by the conversion cost from one to the other, which is often easy and can be done primitive by primitive.

77: The exact number of primitives might not be the same since a discretised element in the model might be implemented as one or as multiple separate structures.

Since algorithms operate on primitives, which are defined by a data model, this analysis also serves to know which fundamental operations can be applied on a data structure. For instance, Euler operations are defined in terms of cell complexes [Mäntylä, 1988] and stellar operators in terms of simplicial complexes [Velho, 2003]. Therefore, in general terms, any data structure that represents cell complexes can implement Euler operations, and the same for stellar operators and simplicial complexes.

Regarding the three broad data models identified in this thesis, on one end it is possible to recognise geometric simplicial complexes, which have the most powerful algebra and are the easiest to manipulate, but require the subdivision of each object into simplices using a constrained or conforming triangulation, which is very hard to do in more than 3D and for which there is no known available software. Cell complexes are on the other end, as they are trivial to construct but only support a much weaker set of fundamental operations, which increases the complexity of many computations. Ordered topological models, such as the cell-tuple/generalised maps and combinatorial maps, combine characteristics of the previous two and have both a relatively powerful algebra and are easy to construct, although they are also the most space-intensive and the most complex to implement. However, this last point is easily overcome in practice since there are good implementations of them in sev-

eral open source libraries and software, including CGAL, Moka, and CGoGN—although special handling for holes and non-manifold geometries might still be required. Because of this, *ordered topological models are deemed to be the most promising alternatives for higher-dimensional spatial modelling and are used as a base for further operations in the following chapters of this thesis.*

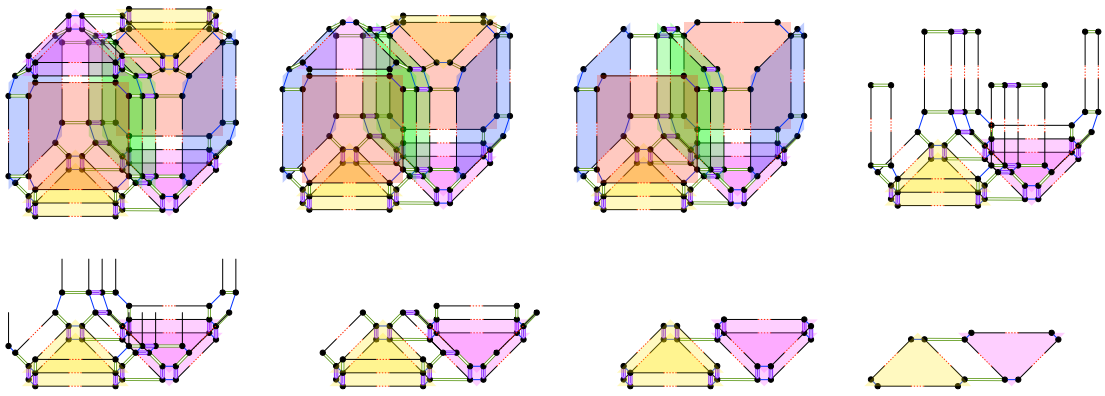
Using a higher-dimensional representation of objects enables combined geometric and topological queries across all dimensions. For instance, applied to 4D BIM[78] models that include the construction process of a building, it is possible to efficiently do safety checks (e.g. dangerous work is not being performed concurrently and close to other activities, or ensuring that all building components are connected properly at all times) by performing well-defined 4D queries rather than by doing many 3D tests that might not encompass all possible undesirable cases. As more higher-dimensional computational geometry and topology algorithms and libraries become available, such as the recent additions in CGAL and Gudhi[79], this approach also makes it possible to utilise them directly, taking advantage of new developments.

78: http://enr.construction.com/technology/bim/2014/0602-Dynamic-Models-for-Safer-Sites.asp?page=2

79: https://project.inria.fr/gudhi/

Part II

Constructing and manipulating objects



Basic dimension-independent operations | 5

Chapter 4 identified the most promising data structures in the context of a higher-dimensional GIS and explained how certain fundamental operations are intimately related to the data structures for which they are defined. This chapter therefore covers a number of basic dimension-independent operations, some of which need to be defined in the context of a specific data structure. These operations are by themselves useful, as they correspond to common simple queries and modification. In addition, some of them are used as building blocks for the more complex operations described in latter chapters of this thesis.

§5.1 introduces basic properties and operations on generalised/-combinatorial maps and on Nef polyhedra, which are later respectively used for more complex higher-dimensional operations (Chapters 6, 7 & 8) and for cleaning 2D and 3D data (Chapter 10). §5.2 describes how to apply basic transformations to an nD simplicial/cell complex, which can be used to manipulate nD objects and to move around an nD scene (Chapter 9).

§5.3 describes some spatial indexing methods which can be applied to higher-dimensional objects, allowing a system to solve two common problems: keeping track of disconnected objects and efficiently obtaining the objects in a particular region. §5.4 discusses how the concept of duality works in higher dimensions, which can be used to characterise the relationships between objects of any dimension. §5.5 puts the concepts of this chapter into practice by providing a few concrete examples of simple dimension-independent operations, including a technique to quickly compare objects in any dimension, which is itself used as the basis of the incremental construction operation in Chapter 7. These examples showcase how operations on an ordered topological model are remarkably efficient compared to the same operations on a Simple Features-like representation.

Most of §5.4 is based on the paper:

- **Representing the dual of objects in a four-dimensional GIS**. Ken Arroyo Ohori, Pawel Boguslawski and Hugo Ledoux. In A. Abdul Rahman, P. Boguslawski, C. Gold and M. N. Said (eds.), *Developments in Multidimensional Spatial Data Models*, Lecture

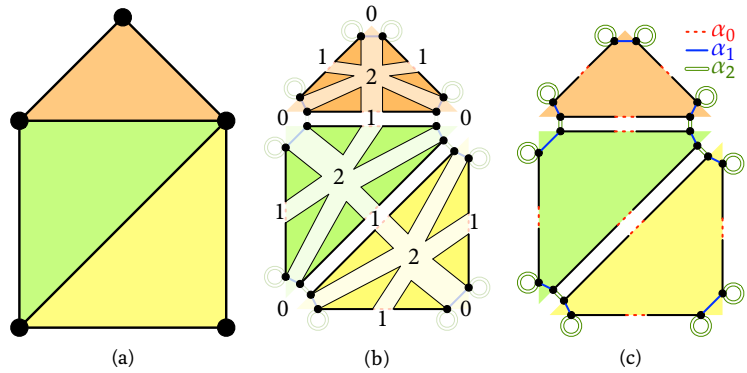


Figure 5.1: (a) A 2D cell complex represented as (b) a 2D generalised map in a simplicial complex-based depiction as used in §4.3.6, and (c) in a simpler depiction using half-edges and showing explicitly the relations between the darts. As involutions are pairwise relations between darts, they are their own inverse, i.e. applying them twice always means returning to the original dart.

Notes in Geoinformation and Cartography, Springer Berlin Heidelberg, Johor Bahru, Malaysia, May 2013, pp. 17–31.

5.1 Basic operations on certain data structures

5.1.1 Generalised and combinatorial maps

As described in §4.3.6, an object in an ordered topological model is described as a set of combinatorial simplices, which are connected by a set of predefined relations. These relations and the operations that use them are described in more detail below, partly based on the definitions used in Damiand and Lienhardt [2014]. Note however that some of the definitions presented here are somewhat simplified, as they are meant only to handle the types of representations and further operations used in this thesis.

In the case of a generalised map, which is shown in Figure 5.1, all the relations between darts take the form of *involutions*—bijective functions that are their own inverse. That is, they are functions f such that $f\big(f(x)\big) = x$, or equivalently $f = f^{-1}$. In addition, certain darts are i-free, which means that for a given dart d, $\alpha_i(d) = d$. These darts represent the boundary of the map.

More formally, a n-dimensional generalised map is defined as a $(n+2)$-tuple $G = (D, \alpha_0, \dots, \alpha_n)$, where:

- D is a finite set of darts;
- $\forall 0 \leq i \leq n$, α_i is an involution on D;
- $\forall 0 \leq i \leq n-2, \forall i+2 \leq j \leq n$, $\alpha_i \circ \alpha_j$ is an involution on D.

In a combinatorial map, the relations between the darts are somewhat different. As shown in Figure 5.2, polygonal curves in a combinatorial map result in some darts that have an undefined β_1 relation. For a given dart d with an undefined relation β_1, this is encoded as $\beta_1(d) = \varnothing$ and the dart is said to be 1-free. Because of this, the β_1 relations in a combinatorial map do not form involutions, but instead form only *partial permutations*. In addition, as shown in Figure 5.3, in a combinatorial map of any dimension, the darts that lie on the boundary of the map have certain undefined β relations, which are set to the special value $\varnothing$. If a given $\beta_i(d) = \varnothing$, the dart d is said to be i-free. Because of the existence of these relations, the β relations of a combinatorial map other than β_1 form *partial involutions*.

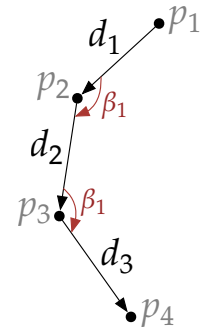


Figure 5.2: The β_1 relations in a 1D combinatorial map are partial permutations. Whereas $\beta_1(d_1) = d_2$, $\beta_1(d_2) = d_3$ and $\beta_1(d_3) = \varnothing$.

More formally, an n-dimensional combinatorial map is defined as an $(n+1)$-tuple $C = (D, \beta_1, \dots, \beta_n)$ where:

- D is a finite set of darts;
- β_1 is a partial permutation on D;
- $\forall 2 \leq i \leq n$, β_i is a partial involution on D;
- $\forall 0 \leq i \leq n-2, \forall 3 \leq j \leq n, i+2 \leq j$, $\beta_i \circ \beta_j$ is a partial involution on D.

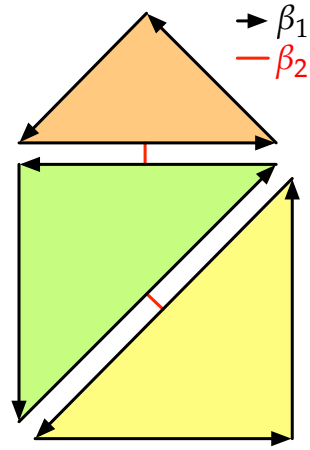


Figure 5.3: The β_1 relations in a 2D combinatorial map are partial permutations, the β_2 relations are partial involutions.

The properties of generalised and combinatorial maps are crucial, as when they are enforced, they ensure that darts form a valid combinatorial structure. They also allow the definition of various operations that operate on the combinatorial structures induced by the darts and their relations. Many of these operations are based on the concept of an *orbit*, a subset of the darts of a generalised/combinatorial map that are connected by certain relations. Starting from a given dart d and a set of relations A, the orbit $\langle A \rangle(d)$ obtains all the darts that can be reached by recursively following all relations in A.

Darts of a cell Considering that every cell in a generalised or combinatorial map is represented by a *set of darts*, an important basic operation is obtaining all the darts that represent a given cell (of any dimension). In a generalised map, from a given dart d known to be part of an i-cell, the darts of the i-cells are those that can be reached by recursively following all relations *except* for α_i. As shown in Figure 5.4, this is given by the orbit $\langle \alpha_0, \dots, \alpha_{i-1}, \alpha_{i+1}, \dots, \alpha_n \rangle(d)$. In a combinatorial map, due to the oriented nature of the β_1 relations, in order to obtain the darts of a 0-cell it is instead necessary to follow a composition of relations given by $\langle \{\beta_j \circ \beta_k \mid \forall 1 \leq k \leq n, \forall 1 \leq j \leq k\} \rangle(d)$. The cells of dimension two and higher are obtained in a similar manner as in a generalised map, by following all relations *except* for β_i, i.e. $\langle \beta_1, \dots, \beta_{i-1}, \beta_{i+1}, \dots, \beta_n \rangle(d)$.

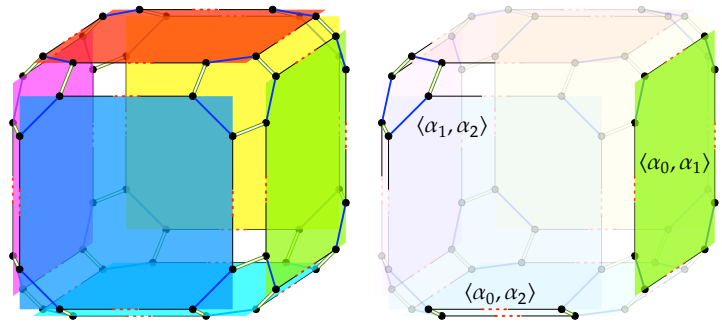


Figure 5.4: Starting from (a) a 2D generalised map representation of a cube, all the darts representing an i-cell of the map can be obtained by starting from any dart known to be part of the i-cell, then using the orbit that contains all relations but α_i. (b) Here, the orbits of a specific 0-cell, 1-cell and 2-cell.

Isomorphism It is possible to check if two maps are *isomorphic* by finding whether there is a one-to-one mapping between the darts in the maps that also preserves the relations between them. That is, given two generalised maps $(D, \alpha_0, \dots, \alpha_n)$ and $(D', \alpha'_0, \dots, \alpha'_n)$, an isomorphism is a function $f : D \to D'$ such that $\forall d \in D, \forall 0 \leq i \leq n, f\big(\alpha_i(d)\big) = \alpha'_i\big(f(d)\big)$. Similarly, given two combinatorial maps $(D, \beta_1, \dots, \beta_n)$ and $(D', \beta'_1, \dots, \beta'_n)$, an isomorphism is a function $f : D \cup \{\varnothing\} \to D' \cup \{\varnothing\}$ such that $f(\varnothing) = \varnothing$, and else $\forall d \in D, \forall 1 \leq i \leq n, f\big(\beta_i(d)\big) = \beta'_i\big(f(d)\big)$.

Sewable darts Starting from isolated darts, these can be assembled together via the sewing operation, described below. However, it is important to first determine if they can be sewed together along a given dimension. Given a generalised map $(D, \alpha_0, \dots, \alpha_n)$ and two darts from the map $d, d' \in D$, d and d' are i-sewable if and only if $d \neq d'$, d and d' are i-free, and there is an isomorphism between $\langle \alpha_0, \dots, \alpha_{i-2}, \alpha_{i+2}, \dots, \alpha_n \rangle(d)$ and $\langle \alpha_0, \dots, \alpha_{i-2}, \alpha_{i+2}, \dots, \alpha_n \rangle(d')$ such that $f(d) = d'$[81]. Similarly, given a combinatorial map $(D, \beta_1, \dots, \beta_n)$ and two darts from the map $d, d' \in D$, d and d' are i-sewable if and only if $d \neq d'$, d and the inverse of d' are i-free, and there is an isomorphism between $\langle \beta_1, \dots, \beta_{i-2}, \beta_{i+2}, \dots, \beta_n \rangle(d)$ and $\langle \beta_1, \dots, \beta_{i-2}, \beta_{i+2}, \dots, \beta_n \rangle(d')$ such that $f(d) = d'$[82].

81: This last condition enforces the last criterion in the definition of a generalised map given above.

82: Similarly, this last condition enforces the last criterion in the definition of a combinatorial map given above.

Sewing The sewing operation is the basic construction method used to link isolated darts in a map in order to form cell complexes. Intuitively, it is possible to link two i-cells along a common $(i-1)$-cell by sewing corresponding darts on the common bounding $(i-1)$-cell around each i-cell. This operation is thus equivalent to a parallel traversal of two maps that links its corresponding darts along an $(i-1)$-cell. Given a generalised map $(D, \alpha_0, \dots, \alpha_n)$ and two darts $d, d' \in D$ that are i-sewable using the isomorphism f, i-sewing d and d' means that $\forall e \in \langle \alpha_0, \dots, \alpha_n \rangle(d)$, $\alpha_i(e)$ should be set to $f(e)$ and $\alpha_i\big(f(e)\big)$ should be set to e. Given a

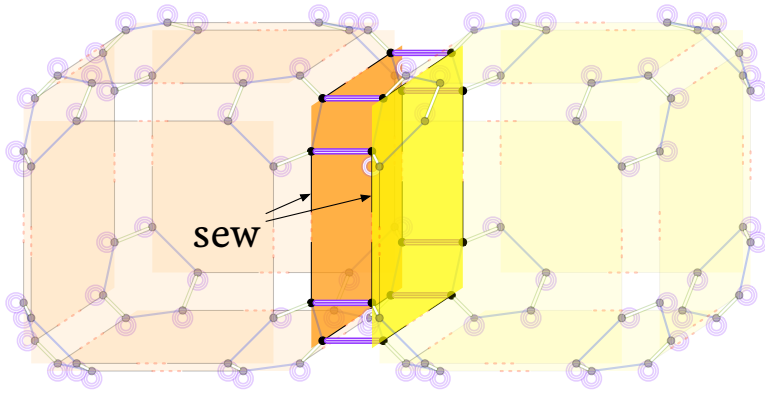


Figure 5.5: Two cubes in a 3D generalised map can be linked together by 3-sewing corresponding darts on the common face (highlighted) on each cube. This operation uses the orbits of each face on each cube in order to link all corresponding darts on both orbits.

combinatorial map $(D, \beta_1, \dots, \beta_n)$ and two darts $d, d' \in D$ that are 1-sewable using the isomorphism f, 1-sewing d and d' means that $\forall e \in \langle\{\beta_i \circ \beta_j \mid \forall 3 \leq k \leq n, \forall 3 \leq j \leq k\}\rangle(d)$, $\beta_1(e)$ should be set to $f(e)$ and $\forall e' \in \langle\{\beta_i \circ \beta_j \mid \forall 3 \leq k \leq n, \forall 3 \leq j \leq k\}\rangle(d')$, $\beta_1(f(e'))$ should be set to e'. For the cells of dimension two and higher, if d and d' are i-sewable ($i \geq 2$), i-sewing them means that $\forall e \in \langle\beta_1, \dots, \beta_n\rangle(d)$, $\beta_i(e)$ should be set to $f(e)$ and $\beta_i\big(f(e)\big)$ should be set to e.

5.1.2 Nef polyhedra

As discussed in §4.3.3, Nef polyhedra [Nef, 1978; Bieri and Nef, 1988] are able to represent polytopes of any dimension based on a set of local pyramids, which contain the neighbourhood information around each vertex. Despite the fact that—to the best of my knowledge—there is no available implementation of Nef polyhedra in more than three dimensions, their dimension-independent formulation and the relative ease of implementing *robust* operations on Nef polyhedra nevertheless make them valuable in a higher-dimensional setting.

In particular, Boolean set operations on 2D Nef polygons and 3D Nef polyhedra are very useful to obtain valid 2D and 3D datasets—something that will be shown in §10.3. These *clean* datasets can then be used as a base for higher-dimensional ones, either by extruding them (Chapter 6), by using them to define parts of a higher-dimensional object's boundary (Chapter 7), or by linking corresponding datasets that represent other points in scale or time (Chapter 8), among other possibilities.

The key advantage of operations on Nef polyhedra is the fact that many of them can be largely implemented at the local pyramid level. As shown by Seel [2001] in 2D and Hachenberger [2006] in 3D,

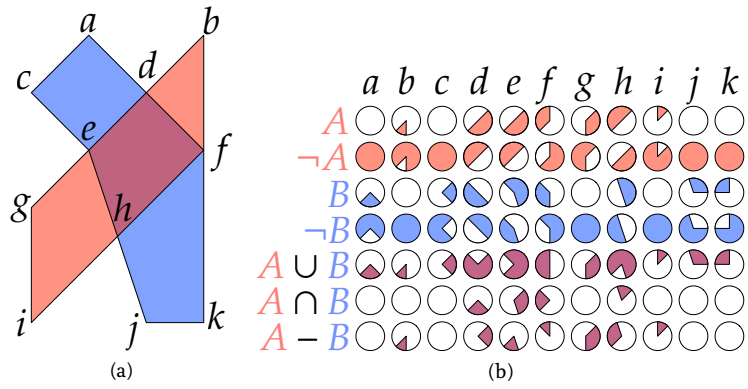


Figure 5.6: Various Boolean set operations on (a) the Nef polygons A (red) and B (blue) that can be performed on (b) their local pyramids: complement ($\neg$), union ($\cup$), intersection ($\cap$) and difference ($-$).

Boolean set operations are one such case. Based on the approach advocated by Rossignac and O'Connor [1989], it is possible to compute these type of binary (or n-ary) operations in three steps: subdivision, selection and simplification. The *subdivision* is the most complex of these. It consists of computing an overlay of the input polyhedra, creating the overall structure where the result will be put (e.g. the vertices, edges, faces and volumes of a cell complex). This can be computed with the methods used for arrangements of line segments in 2D [de Berg et al., 2008, §8.3], or by computing line-segment-to-line-segment and line-segment-to-polygon intersections in 3D [Hachenberger, 2006], which give the locations of the new vertices and thus the new local pyramids.

After this, the *selection* step computes whether the individual cells are to be considered as part of the output or not (i.e. whether they are in the interior or exterior of the Nef polygons/polyhedra). The *simplification* removes unnecessary structures in way that does not alter the point set that is represented, akin to the dissolving operations common in GIS.

Figure 5.6 shows an example of how this works in practice in 2D. A 2D Boolean set operation kernel starts from two Nef polygons A and B—each of which is stored as a set of local pyramids at its corresponding vertices. As shown previously in Figure 3.3 on page 29, each of these 2D local pyramids can be stored as a list of 1D intervals[83]. The kernel first computes the intersections between the line segments (as an overlay problem). The vertices of each polygon and the intersection points between the line segments yield the local pyramids to be considered. The local pyramid intervals for both polygons at all of these locations are computed. A Boolean set operation is then computed by applying it to the local pyramids (i.e. to the intervals). Finally, unnecessary local pyramids can be removed from the output (e.g. in Figure 5.6: f in $A \cup B$; a, b, c, g, i, j and k in $A \cap B$; and a, c, j and k in $A - B$).

83: Note how this essentially reduces the dimensionality of the problem by one.

5.2 Basic transformations of an nD scene

Starting from an n-dimensional simplicial complex or cell complex with linear geometry, where every vertex is embedded in a location in $\mathbb{R}^n$, it is possible to define a set of basic transformations to manipulate nD objects simply by applying them to the coordinates of every vertex. This section thus describes dimension-independent versions of the three most important transformations to an $\mathbb{R}^n$ point set: *translation*, *rotation* and *scaling*. In addition, it explains how to compute the *cross-product* in higher dimensions, which is later used to compute a vector that is orthogonal to all of a set of other vectors.

Translating of a set of points in $\mathbb{R}^n$ can be easy expressed as a sum with a vector $t = [t_0, \dots, t_n]$, or alternatively as a multiplication with a matrix using homogeneous coordinates, which is defined as:

$$T = \begin{bmatrix} 1 & 0 & \cdots & 0 & t_0 \\ 0 & 1 & \cdots & 0 & t_1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & t_n \\ 0 & 0 & \cdots & 0 & 1 \end{bmatrix}$$

In particular, it is often useful to apply a multiplication with a *centering matrix* [Marden, 1996, §3.2], which moves a dataset to a position around the origin. Such a matrix would be defined as $\mathbb{I}_n - \frac{1}{n}\mathbb{1}$, where $\mathbb{I}$ is the identity matrix and $\mathbb{1}$ is a matrix where all entries are set to 1.

Scaling is similarly simple. Given a vector $s = [s_0, s_1, \dots, s_n]$ that defines a scale factor per axis (which in the simplest case can be the same for all axes), it is possible to define a matrix to scale an object as:

$$S = \begin{bmatrix} s_0 & 0 & \cdots & 0 \\ 0 & s_1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & s_n \end{bmatrix}$$

Rotation is somewhat more complex. Rotations in 3D are often conceptualised intuitively as rotations *around an axis*. As there are three degrees of rotational freedom in 3D, combining three such elemental rotations can be used to describe any rotation in 3D space. Most conveniently, these three rotations can be performed respectively around the x, y and z axes, such that a point's coordinate on the axis being rotated remains unchanged. This is a very elegant formulation, but this view of the matter is only valid in 3D.

A more correct way to conceptualise rotations is to consider them as rotations *parallel to a given plane* [Hollasch, 1991], such that a point that is continuously rotated (without changing rotation direction) will form a circle that is parallel to that plane. This view is valid in 2D (where there is only one such plane), in 3D (where a plane is orthogonal to the usually defined axis of rotation) and in any higher dimension. Incidentally, this shows that the degree of rotational freedom in nD is given by the number of possible combinations of two axes (which define a plane) on that dimension [Hanson, 1994], i.e. $\binom{n}{2}$. A general rotation in any dimension can also be seen as a sequence of elementary rotations, although the total number of these rotations that need to be performed increases significantly.

Consider the 2D rotation matrix R_{xy} that rotates points in $\mathbb{R}^2$ parallel to the xy plane:

$$R_{xy} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

Based on it, it is possible to obtain the three 3D rotation matrices to rotate points in $\mathbb{R}^3$ around the x, y and z axes, which correspond to the rotations parallel to the yz, zx and xy planes[84]. These would consist of an identity row and column that preserves the coordinate of a particular axis and rotates the coordinates of the other two, resulting in the following three 3D rotation matrices:

84: Note the order of the three rotation planes given here, which results from *omitting* the axes x, y and z (in that order). Note also the order of the two axes in zx, which follows the right hand rule and defines the signs of the sines in the rotation matrices.

$$R_{yz} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} \qquad R_{zx} = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \qquad R_{xy} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Similarly, in a 4D coordinate system defined by the axes x, y, z and w, it is possible to define six 4D rotation matrices, which correspond to the six rotational degrees of freedom in 4D [Hanson, 1994]. These respectively rotate points in $\mathbb{R}^4$ parallel to the xy, xz, xw, yz, yw and zw planes:

$$R_{xy} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad R_{xz} = \begin{bmatrix} \cos\theta & 0 & -\sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad R_{xw} = \begin{bmatrix} \cos\theta & 0 & 0 & -\sin\theta \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \sin\theta & 0 & 0 & \cos\theta \end{bmatrix}$$

$$R_{yz} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad R_{yw} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & 0 & -\sin\theta \\ 0 & 0 & 1 & 0 \\ 0 & \sin\theta & 0 & \cos\theta \end{bmatrix} \qquad R_{zw} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \cos\theta & -\sin\theta \\ 0 & 0 & \sin\theta & \cos\theta \end{bmatrix}$$

This scheme of a set of elementary rotations can be easily extended to any dimension, always considering a rotation matrix as a transformation that rotates two coordinates of every point and maintains all other coordinates. An alternative to this could be to apply more than one rotation at a time [van Elfrinkhof, 1897]. However, for an application expecting user interaction, it might be more intuitive to rely on an arbitrarily defined rotation plane that does not correspond to specific axes, e.g. by defining such a plane through a triplet of points [Hanson, 1994].

Finally, the **cross-product** is also easier to understand by first considering the lower-dimensional cases. In 2D, it is possible to obtain a normal vector to a 1D line as defined by two (different) points p^0 and p^1, or equivalently a normal vector to a vector from p^0 to p^1. In 3D, it is possible to obtain a normal vector to a 2D plane as defined by three (non-collinear) points p^0, p^1 and p^2, or equivalently a normal vector to a pair of vectors from p^0 to p^1 and from p^0 to p^2. Similarly, in nD it is possible to obtain a normal vector to a $(n-1)$D subspace[85] as defined by n linearly independent points $p^0, p^1, \dots, p^{n-1}$, or equivalently a normal vector to a set of $n-1$ vectors from p^0 to every other point (i.e. $p^1, p^2, \dots, p^{n-1}$) [Massey, 1983; Elduque, 2004].

85: probably easier to picture as an $(n-1)$-simplex

Hanson [1994] follows the latter explanation using a set of $n-1$ vectors all starting from the first point to give an intuitive definition of the n-dimensional cross-product. Assuming that a point p^i in $\mathbb{R}^n$ is defined by a tuple of coordinates denoted as $(p^i_0, p^i_1, \dots, p^i_{n-1})$ and a unit vector along the i-th dimension is denoted as $\hat{x}_i$, the n-dimensional cross-product $\vec{N}$ of a set of points $p^0, p^1, \dots, p^{n-1}$ can be expressed compactly as *the cofactors of the last column* in the following determinant:

$$\vec{N} = \begin{vmatrix} (p^1_0 - p^0_0) & (p^2_0 - p^0_0) & \cdots & (p^{n-1}_0) & \hat{x}_0 \\ (p^1_1 - p^0_1) & (p^2_1 - p^0_1) & \cdots & (p^{n-1}_1) & \hat{x}_1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ (p^1_{n-1} - p^0_{n-1}) & (p^2_{n-1} - p^0_{n-1}) & \cdots & (p^{n-1}_{n-1}) & \hat{x}_{n-1} \end{vmatrix}$$

The components of the normal vector $\vec{N}$ are thus given by the minors of the unit vectors $\hat{x}_0, \hat{x}_1, \dots, \hat{x}_{n-1}$. This vector $\vec{N}$—like all other vectors—can be normalised into a unit vector by dividing it by its norm $\|\vec{N}\|$.

5.3 Spatial indexing

As discussed in §4.4, a topological vector approach seems most promising for a higher-dimensional GIS, as it enables the compact storage of precise object shapes of any dimension together with their attributes. However, vector data structures by themselves lack

a fast access method to access the objects at a certain location or in a certain region (e.g. a bounding box). In addition, in many instances datasets are composed of multiple combinatorially unconnected components, between which it is not possible to navigate using topological relationships. As many algorithms are defined as recursive traversals from a given combinatorial primitive, this lack of connectivity often causes wrong results.

Both of these issues are typically solved in GIS by keeping a *spatial index*—an ancillary data structure that contains links to every object can be efficiently queried [van Oosterom, 1999]. These links can point to embedding structures or to combinatorial primitives.

Some of these spatial indices use the same exhaustive enumeration (i.e. raster) and hierarchical subdivision representations described respectively in §4.3.1 and §4.3.2. Rather than describing objects as a region of a space partition, they store links to the objects that are partially or fully contained in the region. Commonly used structures of this type include: grids, quad/oc/k-trees [Finkel and Bentley, 1974; Meagher, 1980; Yau and Srihari, 1983; Jackins and Tanimoto, 1983] and k-d trees [Bentley, 1975].

Other data structures are also based on hierarchical subdivisions, but they do not contemplate a space partition. Instead, the regions represented by the children of a node can overlap. Data structures of this type include: R-trees [Guttman, 1984], R^+-trees [Sellis et al., 1987] and R^*-trees [Beckmann et al., 1990].

With simple adaptations, these data structures can also be used to index objects in higher dimensions. However, it is important to note that space-partitioning indexing approaches are not faster than brute force searches on a list of objects on *high*-dimensional spaces [Weber et al., 1998], and so the dimension of the data needs to be taken into account. Objects of heterogeneous dimension are also a frequent source of problems, as lower-dimensional objects often result in undesirable edge cases in the heuristics used to keep well-shaped hierarchical subdivisions. For instance, an axis-aligned lower-dimensional object can result in its bounding box having a Lebesgue measure[86] of zero, resulting in long thin boxes that overlap many possible queries.

86: The generalisation of 1D length, 2D area, 3D volume, etc.

Hashing techniques are an interesting alternative, as they allow constant time (or at least low complexity) access to objects. While most forms of hashing are not suitable for spatial objects, there are two relevant exceptions. Geometric hashing [Wolfson and Rigoutsos, 1997] defines a hashing function that is invariant to the (geometric) properties that are required for a particular application, ensuring that objects with a similar shape have similar hash codes. Locality-sensitive hashing [Andoni and Indyk, 2008] instead ensures that objects that are close together also have hash codes that are close together.

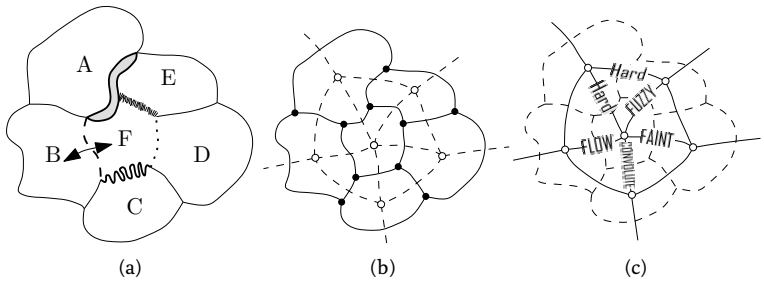


Figure 5.7: (a) Six map objects and their boundaries. (b) The same map stored as a graph and its dual (dotted lines). (c) The dual graph is used to describe the relationships between adjacent polygons. From Ledoux [2006, Ch. 6], itself based on Gold [1991].

A simple indexing approach, which is often used for the work presented in this thesis is to maintain an index on objects based on their *lexicographically smallest vertex*[87]. While this approach is not very useful when trying to obtaining the objects in a region, it is easy to implement and very efficient for many simple operations, such as when objects need to be compared. In fact, by linking directly an index entry to one of a cell's darts which is embedded at the lexicographically smallest vertex (as opposed to an arbitrary dart of the cell), it is possible to significantly limit certain types of search comparisons to a small set of possible starting darts[88]. An extensive example of the use of such an index for comparisons is shown later in §5.5.

87: An idea suggested by Guillaume Damiand.

88: Stricter and more efficient (for a given problem) ordering criteria to choose between all the darts embedded at this vertex could be followed here, but they likely lead to significantly more complex implementations.

5.4 Duality in higher dimensions

The concept of Poincaré duality (§2.3.2) is used in conjunction with spatial information to understand and represent how things are connected. In two dimensions, one application is qualifying the spatial relationships between adjacent objects: as shown in Figure 5.7, Gold [1991] uses the quad-edge data structure [Guibas and Stolfi, 1985] to store simultaneously a map (where each polygon in the map can have certain attributes) and its dual (the boundaries between two adjacent map objects, which can also have certain attributes, e.g. the boundary type or the flow direction).

In three dimensions, duality characterises how volumes are related (e.g. stating that two neighbouring rooms in a building are connected or adjacent). A typical example involves the Delaunay triangulation and the Voronoi diagram, which are dual to each other. Dakowicz and Gold [2003] use them for terrain modelling, Lee and Gahegan [2002] for interactive analysis, and Ledoux and Gold [2008] for three-dimensional fields in the geosciences. In GIS, it is most commonly used to model paths inside 3D buildings, which can be used for navigation computations. Lee and Zlatanova [2008] and Lee and Kwan [2005] extract from a 3D building a graph that can be

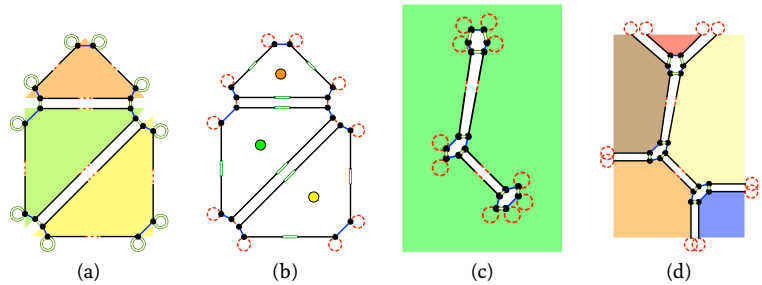


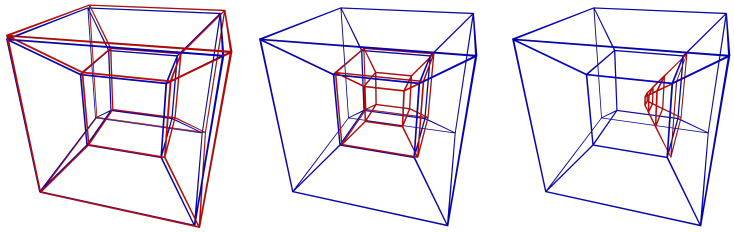
Figure 5.8: The dual of (a) a 2D generalised map can be obtained by (b) computing a point location for the dual 2-cells (e.g. their centroid) and swapping α_0 and α_2, and (c) considering the 2-cells as 0-cells with their new point embeddings. By extending every the exterior of (a) as an additional unbounded face, it is possible to arrive at the alternative dual map in (d).

used in case of emergency, and Boguslawski et al. [2011] and Boguslawski and Gold [2011] perform the same using the *dual half-edge* data structure, which simultaneously represents the buildings (the rooms and their boundaries) and the navigation graph. Liu and Zlatanova [2013] is especially interesting, as it attempts to obtain such a graph from models stored in the 3D standards most widely used in GIS, CityGML and IFC.

In higher-dimensional context of this thesis, duality similarly characterises relationships between objects, albeit its exact meaning depends on the characteristics being modelled as dimensions. For instance, in a 3D space+time setting where a 4-cell represents a volume existing through time, the adjacency relationships between 4-cells represent volumes that were adjacent or connected *during one or more time intervals*. These relationships can thus be used to answer connectivity questions in space-time without the need to add additional semantics in the model. For instance, it would make it possible to create a 3D indoor and outdoor way-finding application, where a user can select any given start and end points and a point in time, and be given the shortest 3D route at that time, taking into account all possible topological changes (e.g. construction work that closes down parts of the building, the dynamic reconfiguration of spaces using plasterboard, or connecting corridors that are only open during office hours). It is worth noting that such a 4D representation effectively stores *the entire history of connectivity in a building* without any added effort.

In a dimension-independent setting, generalised and combinatorial maps are notable because they simultaneously encode a cell complex and its dual. As shown in Figure 5.8, if an n-dimensional cell complex is stored as an n-dimensional generalised map, swapping α_n and α_{n-i} for every i for every dart, the dual of the map is obtained[89]. Note however that as shown in Figure 5.8b, in order to obtain a dual cell complex properly embedded in $\mathbb{R}^n$, it is in practice necessary to compute a new point embedding that lies in the interior of every n-cell.

89: However, does not need to actually swap α_n and α_{n-i}, but merely to *consider* α_n as if it were α_{n-i}.



(a) (b) (c)

Figure 5.9: A stereographic projection of three pairs of tesseracts representing: (a) equality (common 4-cell), (b) adjacency (a common 3-cell), and (c) common 2-cell. Only the edges of each tesseract are shown.

5.5 Comparing two objects with and without signatures

In order to showcase the difference between the topological and non-topological approaches in representing higher-dimensional spatial information, this section compares how efficient the two approaches are in terms of size, and how efficiently three fundamental operations can be performed in either case. The configurations for these tests are shown in Figure 5.9 and respectively involve verifying that two tesseracts are: equal (i.e. that they represent the same 4-cell), adjacent (i.e. that they share a common 3-cell facet), and sharing a common 2-cell ridge. As an example of the topological approach, objects will be represented as a combinatorial map with indices on the lexicographically smallest vertex of every cell. Meanwhile, the non-topological approach will be exemplified with the Simple Features-like [OGC, 2011] representation shown in §4.2.3 and Figure 4.7 on page 61. Indices make little sense in the latter, as there are no combinatorial primitives to navigate and every lower-dimensional cell is represented multiple times. Note that the objects are assumed to have identical coordinates and structure at this point. If this is not the case, cleaning methods such as those mentioned in Chapter 10 or those described by Diakité et al. [2014] need to be applied first.

The tests involving comparisons of combinatorial maps use the method of Gosselin et al. [2011], which can be used to compare two orbits of a map using *signatures*. This method will also be used extensively in Chapter 7. Based on the ordering properties of a map, it is possible to traverse a given orbit from a given dart in a manner that is always consistent, yielding a canonical representation. By following parallel traversals of this type, an algorithm can verify that two cells or maps are isomorphic in $O(n^2)$ time on the number of darts in a cell or cell complex.

Intuitively, the quadratic complexity reflects the fact that two cells or cell complexes are tested for isomorphism by starting one traversal

always at the same dart in one of the cells or cell complexes, possibly including a comparison of up to n darts, while trying all n possible starting darts in the other cell or cell complex. However, it is worth noting that most of the comparisons will stop as soon as one test fails[90], yielding a much better complexity in all but the most pathological cases. Interestingly, Gosselin et al. [2011] also provides a method to verify an isomorphism in $O(n)$ time on a specially generated external signature that uses $O(n^2)$ space, but this method is deemed too space-intensive for the purposes of this thesis.

90: e.g. by differing in a specific cell

All tests are described in detail below.

Size comparison

Comparing the size of each structure is rather difficult as it can vary greatly depending on the specific implementation. The specific implementations described below should nevertheless be roughly comparable, as they are relatively tailored to the kind of object being represented but do not hard-code any of its specific properties (e.g. the number of its bounding 3- or 2-cells).

Combinatorial map A combinatorial maps representation of a tesseract (Figure 7.5 on page 130) is represented as 192 darts and 16 point embeddings. Stored in a 3D combinatorial map[91], every dart contains links to 4 other darts (β_1 to β_3 plus β_1^{-1}) and to its point embedding. As every point is embedded in 4D, every point embedding must store 4 coordinates. Assuming that dart and point embedding links are stored as indices (e.g. of an array) using 1 byte and point coordinates are stored using 4 bytes, a tesseract is stored in exactly **1 Kb** of memory.

91: There is no need for a 4D combinatorial map, as there is only one 4D object.

Simple Features A Simple Features-like binary[92] representation of a tesseract (Figure 4.7c on page 61) contains 8 lists representing its bounding cubes, each of which contains 6 lists representing the bounding squares of a cube, each of which contains 5 lists representing a cycle of points on its boundary, each of which contains the 4 coordinates of a point. Assuming that the first three levels of the lists (i.e. 4D, 3D and 2D) are implemented as arrays that use one more item than their contents[93] and the last level (i.e. the point coordinates) as arrays of four elements and point coordinates are stored using 4 bytes, a tesseract is stored in **5.9 Kb** of memory.

92: An ASCII equivalent (e.g. using well-known text) could easily be an order of magnitude larger.

93: In an implementation of these lists within lists, it is necessary to use this extra item to know their structure. For instance, this item can be used to terminate a list with a special character (e.g. the `'\0'` null terminator used in C strings) or to put a list's size as its first element. Other implementations (e.g. a linked list) would be much less efficient.

Equality test

Combinatorial map In two equal tesseracts, the lexicographically smallest vertex is the same for both of them. Using the index,

it is possible to get a dart at the lexicographically smallest vertex in each. As 4 cubes in each contain this vertex, and 3 faces of each of these cubes contain it, there are 12 darts that are embedded there. Using the isomorphism test using signatures, this would mean 12 possible starting darts in one tesseract's traversal (the other is fixed), each of which can be performed in 2 orientations. As these traversals can be of up to 192 darts, and each dart comparison involves 4 β-links and a point embedding with 4 coordinates, the process might include up to **36 864** comparisons.

Simple Features It is possible to test for equality by traversing one tesseract in order, point by point, while searching for an equivalent point in the other tesseract. As the first two levels of their lists can be in any order (i.e. 4D and 3D) and the third level list (i.e. 2D) can have any starting point, there are 240 possible starting points from which a parallel traversal in the other tesseract can be started, which can be done in 2 possible orientations (forward and backward in every face cycle). These traversals can include up to 240 points, each including up to 4 coordinate comparisons, yielding a total of up to **460 800** comparisons.

Adjacency test

Combinatorial map A tesseract has 8 cubical 3-cells, any of which can be equivalent to a 3-cell in the other tesseract. A worst case test thus involves 32 cube-to-cube comparisons[94]. Comparing two cubes involves a similar process as the one described above for a tesseract. Using indices, it is possible to arrive at the 3 darts at the lexicographically smallest vertex of each cube, from which comparisons can be started in 2 orientations, which involve traversals of up to 24 darts. Each dart comparison involves 3 β-links (β_3 does not need to be tested) and a point embedding with 4 coordinates, yielding 1 008 comparisons per cube-to-cube test and **32 256** total.

94: However, most of these cubes would not have the same lexicographically smallest vertex, but as the number of cubes with the same vertex heavily depends on the configuration, here we use the very conservative absolute worst case of 32.

Simple Features There are also 32 possible cube-to-cube comparisons to be made, each of which is also analogous to the tesseract comparison used above. In a cube (Figure 4.7b on page 61), there are 30 possible starting points for a parallel traversal in 2 possible orientations, each involving up to 30 points with 4 coordinates. Each cube-to-cube comparison thus involves 7 200 comparisons, for a total of **230 400**.

Table 5.1: The relative sizes and maximum number of comparisons in tests with two tesseracts (Figure 5.9) using combinatorial maps (c-maps) and a Simple Features Specification (SFS) representation.

Test	c-maps	SFS	factor
size (Kb)	1 Kb	5.9 Kb	5.9
equality (comparisons)	36 864	460 800	12.5
adjacency (comparisons)	32 256	230 400	7.14
common 2-cell (comparisons)	13 824	57 600	4.17

Common 2-cell test

Combinatorial map Each tesseract has 24 square 2-cells, so there are 288 possible square-to-square comparisons[95]. There is only 1 dart at the lexicographically smallest vertex of each square, from which a comparison can be started in 2 orientations, involving up to 4 darts with 2 β-links and a point with 4 coordinates. There are thus 48 comparisons per square-square combination and **13 824** total.

95: As in the adjacency test, most of these squares would not have the same lexicographically smallest vertex, but since the number of squares with the same vertex depends on the configuration, here we use the very conservative worst case of 288.

Simple Features There are also 288 possible square-to-square comparisons. In a square (Figure 4.7a on page 61), there are 5 possible starting points for a traversal in 2 orientations, each involving 5 other points with 4 coordinates. A square-to-square comparison thus involves 200 comparisons, for a total of **57 600**.

Conclusions and insights into tests in higher dimensions

Table 5.1 summarises the results of the previous tests. As the table shows, a topological approach using combinatorial maps is significantly more efficient than a non-topological Simple Features-like approach. This is true both in terms of size and the number of comparisons involved in basic operations. The difference ranges from a factor of 4.17 for detecting a common 2-cell ridge to a factor of 12.5 for detecting equality (i.e. a common 4-cell).

This difference only becomes more pronounced with more complex objects and those of higher dimensions, as the advantage of having an index pointing to a well-defined starting point for a traversal is inversely proportional to the number of possible starting points.

However, it is worth noting that the differences between the two could be heavily reduced by using a modicum of topology in the Simple Features-like representation[96]. If the lowest-level lists representing the coordinates of a point were exchanged for unique pointers to an external array with the coordinates of all points, many of these searches and the overall storage of the Simple Features approach could be significantly optimised.

96: Even if this is a moot point in practice, as topology seems to be anathema to Simple Features.

6 Extrusion

Extrusion is a commonly used technique in GIS to construct simple 3D models. Starting from a planar partition of polygons and a height interval associated to each of them, it generates a set of space-partitioning box-shaped polyhedra by considering that each polygon exists all along its related interval. For instance, a set of building footprints and associated heights is extruded into a set of simple prismatic buildings. Based on the fundamental operations on generalised and combinatorial maps discussed in Chapter 5, this chapter presents a generalisation of this technique to higher dimensions: an $(n-1)$-dimensional cell complex and a set of associated intervals per $(n-1)$-cell is thus transformed into an n-dimensional cell complex.

The chapter starts by presenting some background on 2D-to-3D extrusion and an intuitive description of dimension-independent extrusion in §6.1. Afterwards, §6.2 describes an dimension-independent extrusion algorithm. §6.3 describes how this algorithm was implemented based on the implementation of combinatorial maps in CGAL. §6.4 summarises experiments using this implementation, which created consistent objects in up to 6D by combining publicly available GIS datasets. §6.5 describes how extrusion can be used not only directly, but as a base to create dimension-independent generalisation operations. Finally, §6.6 concludes the chapter with the main findings and the possibilities to use extrusion for higher-dimensional datasets.

Most of this chapter is based on the paper:

- **A dimension-independent extrusion algorithm using generalised maps**. Ken Arroyo Ohori, Hugo Ledoux and Jantien Stoter. *International Journal of Geographical Information Science* 29(7), July 2015, pp. 1166–1186.

6.1 Background

Extrusion is most commonly used in GIS[98] in order to create simple 3D city models with box-shaped buildings. In such a process, a user takes a set of building footprints, optionally subdivided into

98: This contrasts with the use of the term 'extrusion' in other fields. For instance, in geometric modelling, extrusion is a well-known operation in which all of the objects in the model are 'dragged' along a predefined path given by a curve, more akin to an actual (physical) extrusion process, but whose output topology can be computed without any geometric computations.

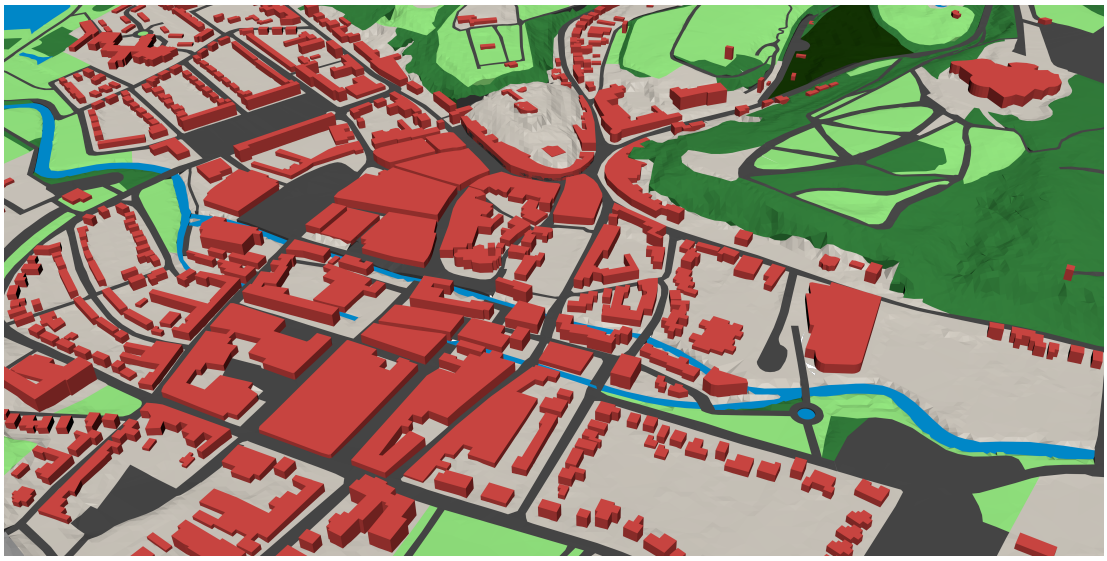


Figure 6.1: A view of the 3D TOP10NL dataset[99] in the area of Valkenburg, the Netherlands.

parts of similar height, and extrudes them using intervals that extend from the ground level to the building height at the location of each polygon. This process thus creates simple 3D representations of each building, where a building is represented by one or more box-shaped polyhedra. Figure 6.1 shows an example of such a model over a triangulated terrain (i.e. a 2D simplicial complex embedded in 3D).

Such an extrusion process is appealing largely because of its simplicity. All the information that is required for the previous example is notoriously easy to obtain: building footprints are widely available—including in open datasets such as OpenStreetMap[100]—or can be easily obtained from satellite imagery, while the height information can be acquired with techniques like airborne laser scanning or photogrammetry. Not coincidentally, this type of 3D models are commonly generated in practice and often appear in standards such as CityGML [Gröger et al., 2012], where it is defined as the level of detail (LOD) 1.

Moreover, it is easy to ensure that extruded models are generated in a topologically consistent manner [Ledoux and Meijers, 2011]. If the input forms a planar partition—or is processed to form a planar partition, e.g. using the method described in §10.2—and the extrusion direction is orthogonal to the planar partition, *the output polyhedra are guaranteed not to overlap*.

While not every possible 3D shape can be generated using extrusion—the top and bottom faces will always be horizontal, and

99: https://www.pdok.nl/nl/producten/pdok-downloads/basis-registratie-topografie/top10nl-3d

100: http://www.openstreetmap.org/

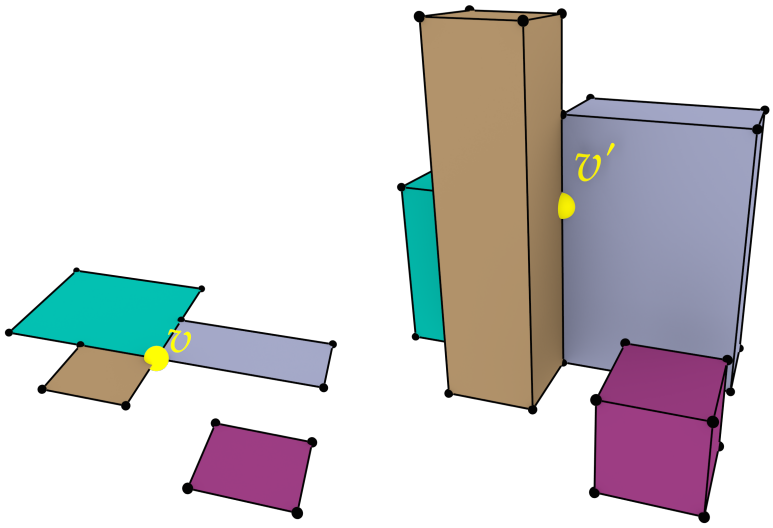


Figure 6.2: (a) A 2D cell complex, where each 2-cell is associated with a height, is extruded to generate (b) a 3D cell complex. Note how the extrusion of vertex v in (a) causes the generation of several vertices and edges. In particular, the vertex v' in (b), which is at the height of the top face of the back left polyhedron. This vertex needs to be used in the representation of the all 4 edges, 5 faces and 3 volumes that are incident to it, not only in those that are part of the back left polyhedron. This ensures that the cells in (b) are pairwise disjoint and have the expected correct topology.

the side faces connecting them will always be vertical—, a remarkable number of shapes can be constructed using this type of method [Ferrucci, 1993]. This is especially true when simple extensions to the method are considered, such as multiple intervals per 2-cell, possibly overlapping 2-cells in the input, or an arbitrary extrusion direction.

Viewed at a more fundamental level, shown in Figure 6.2, extrusion 'lifts' a 2D cell complex to form a 3D cell complex. In order to form a valid cell complex as defined in §2.3.2, cells must not overlap, but instead they should form a structure of cells of dimension from 0 to 3, where i-cells ($\forall i > 0$) are bounded by sets of $(i-1)$-cells. The extruded cells must therefore take into account the incidence and adjacency relationships between cells, splitting cells of every dimension at the places where the common boundaries of higher-dimensional cells start and end.

Extrusion, in the sense of its GIS definition as exposed in the previous paragraph, has a natural extension to higher dimensions. A dimension-independent extrusion operation thus 'lifts' an $(n-1)$-dimensional cell complex, to an n-dimensional cell complex by assigning one or more intervals to each $(n-1)$-cell, i.e. a subset of 1D Euclidean space[101], along which this cell is defined. For instance, just as a polygon representing the footprint of a building is commonly extruded using an interval $(0, height)$, a polyhedron representing a building can be extruded along the time interval $(construction, destruction)$. As Figure 6.3 shows, extrusion using multiple unconnected intervals is also a possible use case.

101: This is equivalent to one or more edges in Lienhardt et al. [2004] or 1-dimensional polyhedra in Ferrucci [1993].

Figure 6.3: The Frauenkirche in Dresden, Germany, was originally built in 1743, destroyed during the bombing of Dresden in 1945, and reconstructed in 2005. A simple 4D model of the building could be made by extruding a 3D model of it along the interval $(1743, 1945) \cup (2005, \infty)$. Photograph by David Müller in Wikimedia Commons.

In the related literature, there are two methods that can be used for a similar purpose. The Cartesian product, a more general operation defined for generalised maps in Lienhardt et al. [2004], generates a combinatorial structure equivalent to the extrusion of all the objects along a single interval by computing the Cartesian product of the original (unextruded) cell complex with an edge. However, since such an operation is limited to a single interval (which is the same for all the input objects), the output cannot be directly applied for modelling real-world datasets. A related possibility, presented by Ferrucci [1993], is to first pre-process all the intervals (for all cells), splitting them into fragments at the other intervals' endpoints so that each fragment intersects another one if and only if they have the same endpoints, and then computing the output purely combinatorially for each possible combination of fragment and for each input cell. While this allows us to model real-world objects, the process of splitting the input intervals might greatly increase the number of output cells since two intersecting intervals are split even if their corresponding cells are far away from each other. The method presented in this chapter follows a similar approach, but instead process the input intervals for each input cell separately based on the incidence relationships with other cells, thus generating fewer total intervals, a smaller number of cells and a considerably smaller combinatorial structure.

6.2 A dimension-independent extrusion algorithm

The dimension-independent extrusion algorithm requires two input arguments: an $(n-1)$-dimensional space partition of $(n-1)$-polytopes embedded into $(n-1)$-dimensional space, stored as an $(n-1)$-dimensional generalised map G; and a map of extrusion intervals ρ[102] that links each $(n-1)$-cell c in G to a set R of 1-dimensional intervals, where every interval r in R is represented as a pair of values $(r_{\min}, r_{\max})$ where c is extruded along the n-th dimension. The intervals in ρ for the cells of lower dimension do not need to be given, since they can be computed by the algorithm based on their incidence relationships to the $(n-1)$-cells. Note that, as Figure 6.4 shows, multiple intervals for the same cell are possible.

102: I originally referred to these as *ranges* and thus used variations of R and ρ (rho). While a kind anonymous reviewer correctly suggested that it is more precise to refer to them as *intervals*, ρ is more distinctive than iota (ι) and upsilon (υ), and so it remains as ρ here.

There are two cases in which a cell has multiple intervals. One of these is that for an $(n-1)$-dimensional cell, multiple intervals may be explicitly provided in the input (e.g. the example previously given in Figure 6.3). The other case is that for a lower-dimensional cell, multiple extrusion intervals may be passed to it by several adjacent higher-dimensional cells.

The result of the extrusion algorithm is an n-dimensional generalised map G' representing a n-dimensional cell complex containing a set of prismatic n-polytopes, the n-dimensional analogue of a

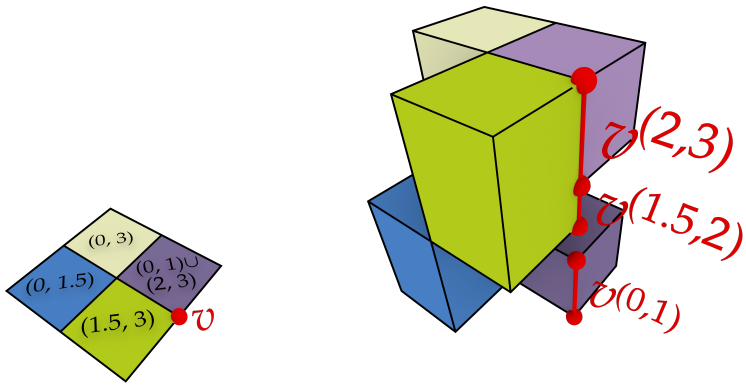


Figure 6.4: All the cells are given extrusion intervals based on the intervals for the $(n-1)$-cells. Note how it is possible to have multiple intervals per cell both in the input, e.g. the purple square on the right is extruded along two intervals, and in the output, e.g. the two edges $v^{(0,1)}$, $v^{(1.5,2)}$ and $v^{(2,3)}$ are the result of the extrusion of a single vertex v whose extrusion interval was not directly given.

set of prisms, which also form a n-dimensional space partition. It is important to note that the output map G' creates entirely new structures, i.e. it does not reuse the darts or embeddings of G, since the $(n-1)$-simplices in G are similar but not identical to the n-simplices in the base of G'. These differ in terms of the highest-dimensional involution α_{n-1} and—importantly for an implementation—in the total number of involutions per dart.

As Figures 6.5 and 6.6 show, the cells in the new n-dimensional cell complex in G' have a direct relation to and can be expressed in terms of the $(n-1)$-cells in G and their extrusion intervals in ρ. This property is used in order to define the cells of the output cell complex, which are equivalent to the embeddings in G'. These consist of:

- 'Base' and 'top' $(n-1)$-cells (i.e. facets), which are constructed from every $(n-1)$-cell c in G at the $r_{\min}$ and $r_{\max}$ end points of every interval in $\rho(c)$.
- A series of prismatic facets linking corresponding $(n-2)$-cells (i.e. ridges) c of the above mentioned $r_{\min}$ and $r_{\max}$ facets for every interval, such that every one of these facets corresponds to the extrusion of the ridge along an interval in $\rho(c)$.

For the combinatorial structure, the algorithm takes advantage of the fact that the darts in the generalised map G can be extruded largely independently based on querying the combinatorial structure of G and simple 1D geometric queries along the n-th dimension. Intuitively, the extrusion of a single $(n-1)$-simplex in G consists of layers of n-simplices that are 'stacked' so as to form one part of the prism-shaped output. Each new layer corresponds to a new n-simplex that shares all but one of the nodes with the n-simplex below it.

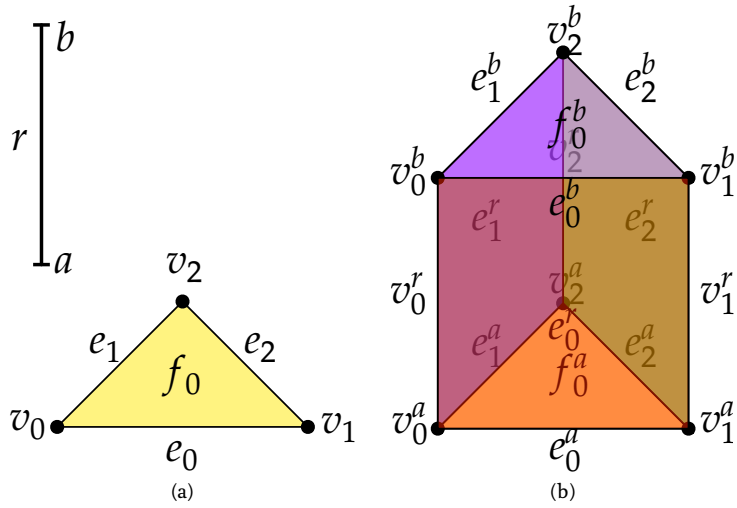


Figure 6.5: Extruding the embeddings of an i-cell c along a single interval $r = (a, b)$ such that $a, b \in \mathbb{R}, a \neq b$, generates the embeddings of three cells: two i-cells c^a and c^b, and an $(i+1)$-cell $c^r = c^{(a,b)}$ lying between them.

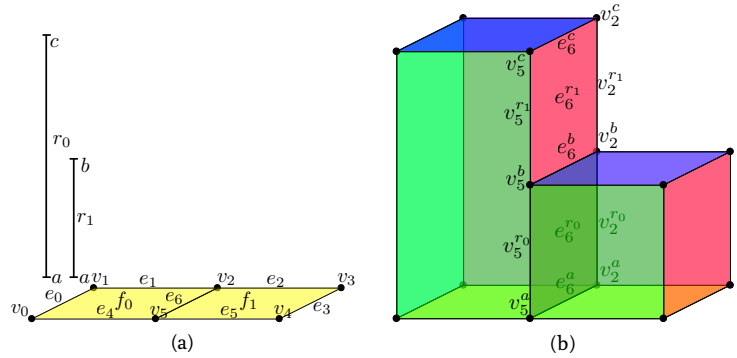


Figure 6.6: Extruding a 2D cell complex using an interval that is defined per 2-cell. The facet f_0 is to be extruded along the interval $r_0 = (a, c)$ and f_1 along $r_1 = (a, b)$. Note how the vertices and edges incident to multiple facets in the input are extruded along the intervals of these facets, generating a series of cells connecting the end points of their intervals.

The extrusion algorithm is thus divided into three parts that are performed sequentially and explained in detail in the following sections: (1) propagating the input intervals to all the cells in the input cell complex, (2) generating the new embeddings (i.e. the attributes and geometry) for each input cell, and (3) generating the combinatorial structure (i.e. the darts and involutions) and linking each dart to its correct embeddings for every dimension.

6.2.1 Propagating the extrusion intervals to all cells

While the extrusion intervals are defined only for the $(n-1)$-cells, the cells of every dimension need to be extruded, and therefore the extrusion intervals need to be propagated from the $(n-1)$-cells to the cells of lower dimension. This is done recursively in decreasing

Algorithm 1: PROPAGATERANGES

Input : generalised map G of the input cell complex,
map ρ with the extrusion intervals of the $(n-1)$-cells in G
Output: map ρ with the extrusion intervals of all the cells in G

```
1  for i ← n − 1 to 0 do
2      foreach i-cell c in G do
3          foreach (i − 1)-cell b on the boundary of c do
4              foreach extrusion interval r = (r_min, r_max) ∈ ρ(c) do
5                  Find the intervals R′ ⊆ ρ(b) whose interiors intersect with r
6                  if r_min is in the interior of an interval r′ = (r′_min, r′_max) ∈ R′ then
7                      Remove r′ from ρ(b)
8                      Add (r′_min, r_min) to ρ(b)
9                      Add (r_min, r′_max) to ρ(b)
10                 if r_min is outside all intervals in R′ then
11                     Add a new interval in R′ from r_min to the minimum of
                         the lowest interval in R′
12                 if r_max is in the interior of an interval r′ = (r′_min, r′_max) ∈ R′ then
13                     Remove r′ from ρ(b)
14                     Add (r′_min, r_max) to ρ(b)
15                     Add (r_max, r′_max) to ρ(b)
16                 if r_max is outside all intervals in R′ then
17                     Add a new interval in R′ from the maximum of the
                         highest interval in R′ to r_max
18                 foreach empty interval (r′_min, r′_max) between consecutive intervals
                     in R′ do
19                     Add (r′_min, r′_max) to ρ(b)
```

dimension, using the incidence relationships between every i-cell and the $(i-1)$-cells on its boundary to pass the intervals of the former to the latter, using the same map of extrusion intervals ρ. Because incidence is a transitive relation, an interval attached to a particular lower dimensional cell thus indicates that it is incident to an $(n-1)$-cell that will be extruded along that interval.

As $(i-1)$-cells can be on the boundary of multiple i-cells, several intersecting intervals can be passed to the map of extrusion intervals of a lower dimensional cell. In order to generate non-intersecting cells, these need to be split into a set of non-intersecting intervals as shown in Figure 6.7, each of which will represent the incidence of this cell to an equal set of higher-dimensional cells along the entirety of the interval. A sketch of a simple process to do this is shown in Algorithm 1, which assumes that the sets of intervals are kept sorted. Note that this is essentially a one dimensional analogue of other operations used in GIS, e.g. vector map overlays [de Berg et al., 2008, §2.3] and time-composites in spatio-temporal modelling [Langran and Chrisman, 1988].

Considering what happens when a single new interval is passed to

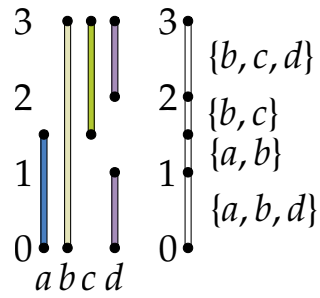


Figure 6.7: The extrusion intervals from the cells incident to the middle vertex of Figure 6.4 in (a), are passed on to it, resulting in a new set of non-intersecting intervals in (b).

a cell in the incremental process shown in Algorithm 1, the total number of intervals of that cell can increase from j to $2j + 1$ in the worst case, which happens when the new interval starts before and ends after all other intervals and these are all not contiguous (i.e. they are all separated by empty intervals). However, when taking into account r intervals passed to the same cell, the total number of intervals for the cell can only increase to $2r - 1$. As each extrusion interval is bounded by two values, r intervals lead to at most $2r$ boundary values and thus the number of intervals cannot be higher than $2r - 1$.

Assuming that a new interval can be added to a cell in $O(\log r)$ time, e.g. by maintaining a sorted list or an augmented red-black tree with the endpoints of the intervals, the total time to construct the ordered set of all intervals for a cell c is $O(r \log r)$, where r is the number of intervals that are passed onto c. The overall computational complexity of this step depends on the incidence relationships between the cells in the complex.

6.2.2 Generating the new embeddings

Intuitively, when a single i-cell is extruded along a single interval $r = (r_{\text{min}}, r_{\text{max}})$, three new cells are generated: two i-cells that correspond to the end points of the interval r_{min} and r_{max}, and a prismatic $(i + 1)$-cell along all of r, i.e. lying between the two i-cells. For instance, extruding a polygon results in 'top' and 'bottom' polygonal faces and a polyhedron that lies between the two polygons.

Analogously, considering the *embedding* structures that store the necessary attributes for each cell, such that an i-dimensional embedding (i-embedding) structure keeps the attributes of an i-cell, *extruding an i-dimensional embedding results in two i-embeddings and one $(i + 1)$-embedding.* In the case of linear geometries, extruding a point entails the creation of two additional points, one with an appended r_{min} coordinate, and one with an r_{max} one.

With multiple intervals, this same procedure can then be used by applying it per embedding and per interval. Notice that some embeddings are shared by multiple intervals (when the minimum of an interval is equal to the maximum of another), but these only need to be created once.

The extrusion algorithm for the embeddings receives the set of input embeddings E and the map of extruded intervals ρ which, being the output of PropagateRanges (Algorithm 1), now contains a set of non-intersecting intervals for the cells of every dimension, and it returns an entirely new set of extruded embeddings E', as well a function $ex(e, v) \to E'$ linking an input embedding $e \in E$ and an interval or end point of an interval v to an extruded (output) embedding $e' \in E'$. For instance, given an interval $r = (r_{\text{min}}, r_{\text{max}}) \in \rho(e)$, v can

Algorithm 2: EMBEDDINGSEXTRUSION

Input : set E of the embeddings in the input cell complex,
map ρ of the extrusion intervals for all cells of the input complex
Output: set E' of the embeddings for the output cell complex,
map ex that links an input embedding to its extruded embeddings

```
1  foreach e ∈ E do
2      foreach r = (r_min, r_max) ∈ ρ(e) do
3          if ex(e, r_min) = ∅ then
4              ex(e, r_min) ← e
5              if e.dimension = 0 then
6                  Append r_min to the coordinates of ex(e, r_min)
7          ex(e, r) ← e
8          ex(e, r).dimension ← ex(e, r).dimension + 1
9          if ex(e, r_max) = ∅ then
10             ex(e, r_max) ← e
11             if e.dimension = 0 then
12                 Append r_max to the coordinates of ex(e, r_max)
13         Put ex(e, r_min), ex(e, r) and ex(e, r_max) in E'
```

be r, $r_{\min}$ or $r_{\max}$, reflecting the fact that extruding an i-embedding results in two i-embeddings (respectively for $r_{\min}$ and $r_{\max}$) and one $(i+1)$-embedding (for r).

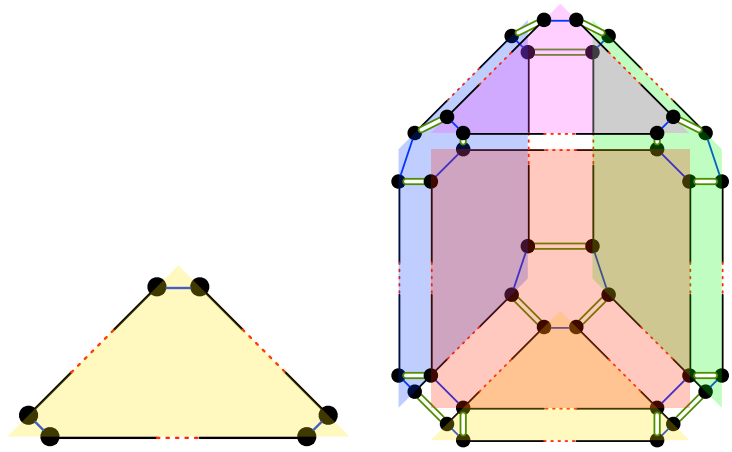
The general procedure to generate the new embeddings and their relation to the old embeddings and the intervals is shown in Algorithm 2. Note that a practical implementation of this algorithm has to deal with the desired attributes for each of the extruded cells rather than simply making a copy of the input ones (lines 4, 6 and 10) and appending one more coordinate to the 0-embeddings (lines 6 and 12).

Since this part of the algorithm iterates through all the cells in the input cell complex, and it generates at most three new embeddings per interval for each cell, the computational complexity can be $O(rn)$ per cell, where r is the total number of intervals and n is the dimension, i.e. as long as the map ex can be queried and the new embeddings can be created in time that is linear on the dimension (which in practice is a relatively small constant).

6.2.3 Generating the new combinatorial structure and linking it to its correct embeddings

The extrusion of a *single* dart in the input map G along a single interval $r = (r_{\min}, r_{\max})$ generates a series of connected darts in G' connected by a sequence of involutions $\alpha_{n-1}, \alpha_{n-2}, \dots, \alpha_1, \alpha_0, \alpha_1, \dots, \alpha_{n-2}, \alpha_{n-1}$. Intuitively, these are equivalent to stacked n-simplices that together form a prism[103]. Figure 6.8

103: This is always true combinatorially, but it might not be true geometrically if the $(n-1)$-simplex is not embedded so as to lie in the interior of the cell, as shown in §4.3.6.



(a) (b)

Figure 6.8: The darts in the cell complexes in Figure 6.5. Note that one can obtain a dart's representation as a simplex by considering additional nodes at its corresponding facet and one in the interior of the volume.

shows a simple extrusion of the cell complex from Figure 6.5 including all its darts.

Since two darts connected by an α_i involution share all but the i-th node, and the i-th node in a generalised map means that a dart belongs to a certain i-cell[104], this series of darts represents a succession of simplices that progressively change from the cells at the 'base' (at $r_{\min}$) to those at the 'side' (at the interval $(r_{\min}, r_{\max})$) to those at the 'top' (at $r_{\max}$). This can be more clearly seen in Figure 6.9. As the pattern that is followed in each stack of darts in G' is the same for all the darts in G, assuming that their extrusion intervals are the same, they can also be expressed in terms of the dart in G being extruded and their position in the stack. The darts at a certain level in the stack are thus called a *layer*, and various functions will map an input dart in G to the equivalent dart at a specific layer in G'.

An intuitive justification for this is that if one considers the darts in the base and top facets in the extrusion of a cell e along a single interval $r = (r_{\min}, r_{\max})$, these belong to different cells of every dimension except for their n-cell—which is $ex(e, r)$. Considering a function $e_i : D \to E$ that links a dart $d \in D$ to its i-embedding in E, for every dimension $i < n$, the darts of the base facet belong to the extruded $ex(e_i(d), r_{\min})$ cells, while the top darts belong to the extruded $ex(e_i(d), r_{\max})$ cells. The darts of the faces on the sides, which connect corresponding ridges of the base and top, belong instead to a mixture of cells in $ex(e_i(d), r_{\min})$, $ex(e_i(d), r_{\max})$ and $ex(e_i(d), r)$. The darts closer to the base face belong to cells in $ex(e_i(d), r_{\min})$ and $ex(e_i(d), r)$, while those closer to the top belong to cells in $ex(e_i(d), r_{\max})$ and $ex(e_i(d), r)$. This is natural when one considers that α_i-linked darts differ only in one of their cells (i.e. the i-cells), all other cells being the same for both darts. This means that there must be a nat-

ural progression of layers of darts: starting from the base, they change their cells one by one from $ex(e_i(d), r_{\min})$ to $ex(e_i(d), r)$ from the highest dimension down, and then change their cells one by one from $ex(e_i(d), r)$ to $ex(e_i(d), r_{\max})$ from the lowest dimension up until reaching the top.

The algorithm to generate the extruded combinatorial structure therefore works by generating layers of darts that follow a pattern based on the darts in the input cell complex, starting from those for the base face, moving on to the $2n-2$ layers for the side faces, and finishing with the top face. The output darts and the involutions between them are expressed in terms of the input generalised map G. For this, a function $cur : G \to G'$ maps a dart in G to the corresponding dart in the currently being generated layer of the output map G'. Note that this means that the algorithm needs to keep track of only two layers of darts at any given time, one in G and one in G', each of which has at most the number of darts in the input map. This bounds the memory usage of this part of the algorithm, which is therefore on the order of $O(d)$, with d the number of darts in the input space partition.

As shown by Ferrucci [1993], when multiple intervals are involved this procedure can simply be repeated for all intervals, assuming that these have all been subdivided so as not to intersect one another. However, it is possible to greatly reduce the number of darts generated by skipping the creation of some of the darts. This is possible because the extrusion intervals have been independently propagated to each cell of every dimension. Based on the algorithm, the lower-dimensional cells in the complex have received all the intervals from their incident higher-dimensional cells so that the intervals in the lower-dimensional cells contain all the endpoints of the intervals of their incidences. However, the same is not true in the opposite direction: the intervals for the higher-dimensional cells have not been subdivided so as to contain all the ones of their incident lower-dimensional cells. Nevertheless, as Figure 6.10 shows, even when an extrusion interval that would be used by the pattern described above is not in a cell, it is possible to map it to a *bigger* interval that contains it. If one considers the darts that would be generated using the above mentioned approach for all the non-intersecting intervals in the cell complex, but at the same time mapping the non-existent extrusion intervals to their immediately bigger containing ones, this can result in many darts that are equivalent (i.e. they have nodes at the same cells). These darts are those that can be skipped during the extrusion process.

Therefore, in order to generate the darts for all intervals of all cells, this procedure is repeated for all the intervals in ρ, after making them non-intersecting using the same procedure delineated in §6.2.1, and skipping all the darts that would be equivalent to those that have already been created. This is done using a sweep-

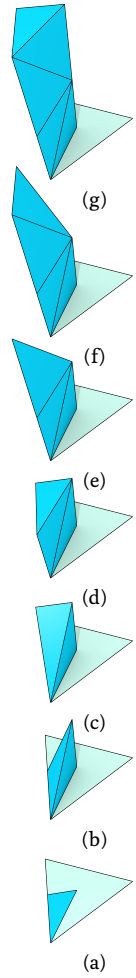


Figure 6.9: The extrusion of a single dart (blue) results in a stack of simplices that together form a prism. In order to visualise it more intuitively, the procedure here is shown from bottom (a) to top (g).

104: In this specific case, where cells have linear geometries.

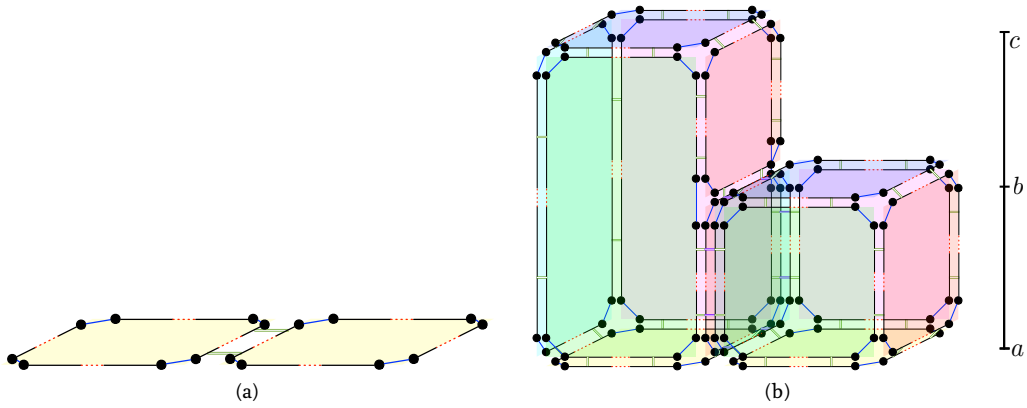


Figure 6.10: The darts in the cell complexes in Figure 6.6. Note that the darts of the right cube in (b), when viewed as individual stacks, are similar to the ones in Figures 6.8 and 6.9. The darts of the left box are however different: those on the left all involve an extrusion along the interval (a, c) due to the fact that there is no vertex, edge or facet extruded to b. On the other hand, those in the right do have a vertex and an edge at b, but not a facet and still belong to the same volume.

hyperplane-like algorithm that generates up to n layers of darts at the events at the beginning or end of an interval, creating darts only when the sweep-hyperplane[105] passes by the beginning or end of their extrusion intervals (i.e. not while it is in their interior). The darts are linked to their appropriate embedding according to the same pattern. The complete procedure to generate the combinatorial structure is presented in Algorithm 3, which uses the Algorithm 4 when the sweep hyperplane passes by the beginning of an interval and the similar Algorithm 5 when the sweep hyperplane by the end of one. If an input dart is denoted as d, its corresponding dart in the currently being generated layer of the output is denoted as $cur(d)$, the dart linked to d by an i-involution as $\alpha_i(d)$, and the i-embedding of d as $e_i(d)$.

105: i.e. a $(n-1)$-dimensional shape that is unbounded along $n-1$ dimensions in nD space, e.g. a line in $\mathbb{R}^2$ or a plane in $\mathbb{R}^3$.

For the latter two algorithms, lines 1–8 show the generation of a new layer of darts and its linking to the previous one, lines 12–17 show how the darts within the layer are linked based on the pattern of the input map, and lines 18–21 show how the darts within the layer are related to their correct embeddings, which are also patterned after the embeddings in the input.

Notice that this method generates layers of darts in a grid-like fashion, each layer containing at most the number of darts in the input map, and calling GMapLayerBegin and GMapLayerEnd to create at most $2n$ layers per non-intersecting interval. The time complexity of computing the set of non-intersecting intervals in Algorithm 1 is $O(r \log r)$ as before, while the number of darts in the output map is bounded by $O(ndr)$, where n is the extrusion dimension, d is the total number of darts in the input map and r is the total number of intervals in the input.

Algorithm 3: GMapExtrusion

Input : generalised map G of the input cell complex,
set E of the embeddings in the input cell complex,
set E' of the embeddings for the output cell complex,
map ρ of the extrusion intervals for all cells of the input complex,
map ex that links an input embedding to its extruded embeddings
Output: generalised map G' of the output cell complex

```
1 Compute an ordered set of non-intersecting intervals r_all using all the
    intervals for all cells in ρ
2 Consider a sweep plane that passes through all the intervals r_all in
    increasing order along dimension n
3 if the sweep plane passes by the beginning of an interval r = (r_min, r_max) ∈ r_all
    then
4     for i ← n to 0 do
5         GMapLayerBegin(G, G', i, E, E', ρ, r, ex)
6 if the sweep plane passes by the end of an interval r = (r_min, r_max) ∈ r_all then
7     for i ← 0 to n do
8         GMapLayerEnd(G, G', i, E, E', ρ, r, ex)
```

6.3 Implementation

The extrusion algorithm has been implemented in C++11 and made available under the open source MIT licence at `https://github.com/kenohori/lcc-tools`. It requires and builds upon the CGAL packages Combinatorial Maps and Linear Cell Complex, among others. The first package provides data structures and algorithms to store and to efficiently iterate over the darts of a combinatorial map, and the second links the 0-embeddings to other CGAL types in order to store the geometry of a model.

As the algorithm is described on the basis of generalised maps while the CGAL packages instead use combinatorial maps, all operations are done through a small wrapper that converts operations for the former to operations for the latter. Since a combinatorial maps dart is equivalent to two generalised map darts (see §4.3.6), some redundant operations are ignored in the process, such as requests to create darts that already exist or to connect darts that are already connected. These redundant operations are instead used as assertions to verify the validity of the map during its creation.

Both these packages and the implementation of the extrusion algorithm make heavy use of the traits programming technique [Myers, 1995] and recursive templates (TMP or template metaprogramming) in order to produce efficient code. These techniques are described in §A.3. One shortcoming of the current prototype implementation is that it uses the C++ type `std::map` in order to link cells to their extrusion intervals, which offers only logarithmic time access [ISO, 2015, §23.4][106], rather than the constant time that would

106: See Austern [2000] for the consequences of this.

Algorithm 4: GMapLayerBegin

Input : generalised map G of the input cell complex,
generalised map G' of the output cell complex,
dimension i of the current layer,
set E of the embeddings in the input cell complex,
set E' of the embeddings of the output cell complex,
map ρ of the extrusion intervals of all cells of the input complex,
current interval r,
map ex that links an input embedding to its extruded embeddings
Output: generalised map G' of the output cell complex

```
1  foreach dart d in G do
2      if ∃r' ∈ ρ(e_{n-1}(d)) | r ⊆ r' then
3          if ∃r'' ∈ ρ(e_i(d)) | r''_min = r_min then
4              last ← cur(d)
5              cur(d) ← new dart
6              Put cur(d) in G'
7              α_{i+1}(cur(d)) ← last
8              α_{i+1}(last) ← cur(d)
9  foreach dart d in G do
10     if ∃r' ∈ ρ(e_{n-1}(d)) | r ⊆ r' then
11         if ∃r'' ∈ ρ(e_i(d)) | r''_min = r_min then
12             for inv ← 0 to i − 1 do
13                 α'_inv(cur(d)) ← cur(α_inv(d))
14                 α'_inv(cur(α_inv(d))) ← cur(d)
15             for inv ← i + 2 to n do
16                 α'_inv(cur(d)) ← cur(α_{inv-1}(d))
17                 α'_inv(cur(α_{inv-1}(d))) ← cur(d)
18             for emb ← 0 to i do
19                 e'_emb(cur(d)) ← ex(e_emb(d), r_min)
20             for emb ← i + 1 to n do
21                 e'_emb(cur(d)) ← ex(e_{emb-1}(d), r)
```

be possible by integrating this into the templated structures. This would involve storing the set of extrusion intervals of a cell directly in the data structure that is used for its attributes.

In order to input and output data, as well as to visualise the results, the implementation uses the OGR Simple Feature Library[107] to read standard GIS data formats. It is also able to output Wavefront OBJ[108] files, in which faces or darts can be directly exported for visualisation. In the latter case, darts are exported as triangles with vertices at the two 0-cells of a dart, as well as an added vertex at the centroid of its 2-cell.

107: http://gdal.org

108: http://www.martinreddy.net/gfx/3d/OBJ.spec

6.4 Experiments

The algorithm has been tested by extruding various 2D datasets to higher dimensions. For this, several free and open datasets in the

Algorithm 5: GMapLayerEnd

Input : generalised map G of the input cell complex,
generalised map G' of the output cell complex,
dimension i of the current layer,
set E of the embeddings in the input cell complex,
set E' of the embeddings of the output cell complex,
map ρ of the extrusion intervals of all cells of the input complex,
current interval r,
map ex that links an input embedding to its extruded embeddings
Output: generalised map G' of the output cell complex

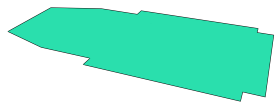
1 **foreach** *dart* d *in* G **do**
2 **if** $\exists r' \in \rho(e_{n-1}(d)) \mid r \subseteq r'$ **then**
3 **if** $\exists r'' \in \rho(e_i(d)) \mid r''_{\max} = r_{\max}$ **then**
4 $last \leftarrow cur(d)$
5 $cur(d) \leftarrow$ new dart
6 Put $cur(d)$ in G'
7 $\alpha_i(cur(d)) \leftarrow last$
8 $\alpha_i(last) \leftarrow cur(d)$
9 **foreach** *dart* d *in* G **do**
10 **if** $\exists r' \in \rho(e_{n-1}(d)) \mid r \subseteq r'$ **then**
11 **if** $\exists r'' \in \rho(e_i(d)) \mid r''_{\max} = r_{\max}$ **then**
12 **for** $inv \leftarrow 0$ **to** $i-1$ **do**
13 $\alpha'_{inv}(cur(d)) \leftarrow cur(\alpha_{inv}(d))$
14 $\alpha'_{inv}(cur(\alpha_{inv}(d))) \leftarrow cur(d)$
15 **for** $inv \leftarrow i+2$ **to** n **do**
16 $\alpha'_{inv}(cur(d)) \leftarrow cur(\alpha_{inv-1}(d))$
17 $\alpha'_{inv}(cur(\alpha_{inv-1}(d))) \leftarrow cur(d)$
18 **for** $emb \leftarrow 0$ **to** i **do**
19 $e'_{emb}(cur(d)) \leftarrow ex(e_{emb}(d), r_{\max})$
20 **for** $emb \leftarrow i+1$ **to** n **do**
21 $e'_{emb}(cur(d)) \leftarrow ex(e_{emb-1}(d), r)$

area of Delft are used, matching the geometries in some with the attributes present in others so as to obtain new attributes and appropriate extrusion intervals for each geometry. The tests were performed on a Mac OS X computer with a 2.7 GHz Intel Core 2 Duo processor and 12 GB of RAM. The main characteristics of the datasets tested are shown in Table 6.1.

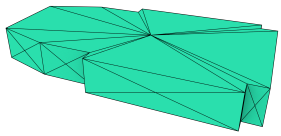
Note however that since the implementation uses an `std::map` to access the extrusion intervals of each cell, the running time is dominated by the large number of times this query is performed (several times per interval and per cell). The times provided in Table 6.1 are therefore not indicative of the theoretical complexity of the algorithm, which is instead dominated by the generation and linking of the darts in the extruded dataset (§6.2.3). The generation of the non-intersecting intervals for all cells (§6.2.1) and the generation of the extruded embeddings (§6.2.2) always ran in under a second, even for very large datasets.

	Input		Output		
Test	darts	cells	darts	cells	time
One GBKN building to 5D	14	29	80 640	783	3 s
370 buildings to 4D	3 268	6 065	123 184	42 552	12 s
TOP10NL to 3D	30 098	48 562	181 640	148 102	2 m 39 s

Table 6.1: Characteristics of the tested datasets. The cell counts represent the total number of cells for all dimensions.



(a)



(b)

Figure 6.11: Extruding the (a) footprint of the Aula Congress Centre in Delft to (b) a 3D representation showing individual darts.

109: http://www.gbkn.nl

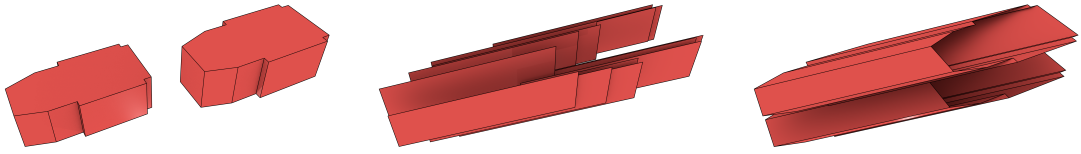
110: http://www.ahn.nl

111: https://www.kadaster.nl/bag

One test involved the Aula Congress Centre in Delft, a building represented by a single polygon with 14 vertices extracted from the GBKN dataset[109], as shown in Figure 6.11. It was extruded from 2D up to 5D using some manually added attributes: its height, construction date and a specified level of detail for the model. Figure 6.12 shows the extruded model at an intermediate stage in this process, in 4D. The end result was a generalised map with 80 640 darts, 112 vertices, 280 edges, 260 facets, 110 volumes, 20 4-cells and 1 5-cell. It was generated in 3 seconds using 15 MB of RAM.

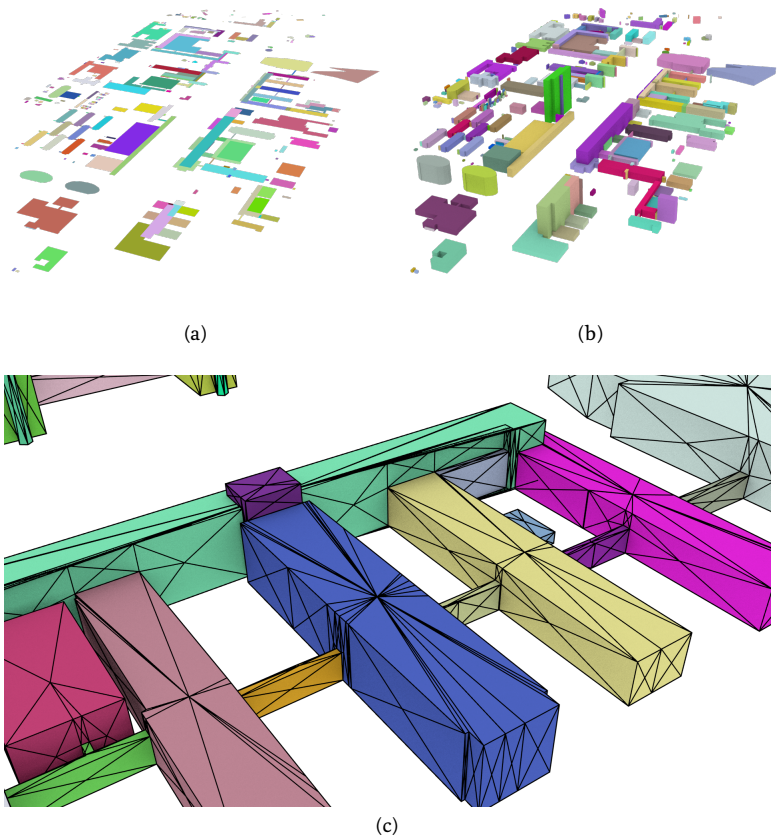
Another test, shown in 2D and 3D in Figure 6.13, involved a previously generated dataset of the campus of the Delft University of Technology (see Ledoux and Meijers [2011]), consisting of 2 749 vertices, 2 946 edges, and 370 facets. This dataset, covering 2.3 km^2, was originally built from the GBKN dataset by manually forming footprint polygons from the lines in the dataset. Building heights for each polygon were obtained from the AHN dataset[110] (airborne laser altimetry), while building dates were obtained from the BAG dataset[111]. This dataset, including the added attributes, is available together with the source code of the program. It was therefore extruded from its original 2D representation to 4D using building heights for the third dimension and dates for the fourth dimension. The result was a generalised map with 123 184 darts, 8 613 vertices, 17 919 edges, 12 471 facets, 3 310 volumes and 239 4-cells. It was generated in 12 seconds using 46 MB of RAM.

One more test used 1 836 buildings from Delft from the TOP10NL



(a) (b) (c)

Figure 6.12: The footprint of the Aula Congress Centre (Figure 6.11) extruded twice to create a 4D model. It is shown here in three parts for clarity: (a) the faces in the two end volumes, (b) the lateral faces connecting corresponding vertical edges, and (c) the top and bottom faces connecting corresponding horizontal edges. Not shown here are the 16 different polyhedra that bound the polychoron and which are all bounded by different sets of these faces.



(a) (b)

(c)

Figure 6.13: Extruding a dataset of the campus of the Delft University of Technology

dataset[112], which was extruded to 3D using intervals from building dates also obtained from the BAG dataset. The result was a generalised map with 181 640 darts, 46 464 vertices, 71 826 edges, 27 975 facets and 1 837 volumes.

An algorithm was used to verify that the constructed datasets conform to the definition of a combinatorial map[113]. Additional tests were made to ensure that all the darts of an i-cell correctly point to it in their i-embeddings, and to verify that all the darts of an n-cell are linked to point embeddings within the extrusion interval given for the n-cell, among other tests. The extruded datasets were also inspected visually in 2D and 3D by exporting 2-cells as polygons and verifying that they form a valid cell complex, i.e. that 2-cells intersect only at their common boundaries, forming 1-cells that are also in the complex. In 2D and 3D, individual darts were also exported as triangles to visually verify that they form a valid generalised map

112: http://www.kadaster.nl/web/artikel/producten/TOP10NL.htm

113: i.e. checking that the links between darts correctly form partial permutations or involutions, and all the darts in the orbit of an i-cell are linked to its correct i-attribute and vice versa.

(as shown in Figures 6.11b and 6.13c).

6.5 Extrusion-based generalisation operations

Apart from using extrusion to directly create finished higher-dimensional datasets, it is also interesting to explore the possibility to use it as a base for further operations. For instance, it can be used as the basis of a simple set of dimension-independent generalisation operations.

Starting from a detailed n-dimensional cell complex, it is possible to first extrude the complex to create an $(n + 1)$-dimensional cell complex, then to obtain the desired geometry by applying certain transformations to the vertices of the 'top' or 'bottom' facet of the extruded model, even if this procedure results in degenerate cells, and finally to remove the degenerate cells from the model.

The transformations that can be applied to the model include all those that can be applied through transformation matrices, such as those shown in Figure 4.3 on page 58 (translation, rotation and scale), as well as others such as reflections and shears. Perhaps more importantly, it is possible to gradually reduce the complexity of the model by collapsing cells of any dimension, which is achieved by simply moving all vertices of a cell to the same location, as is shown in Figure 6.14.

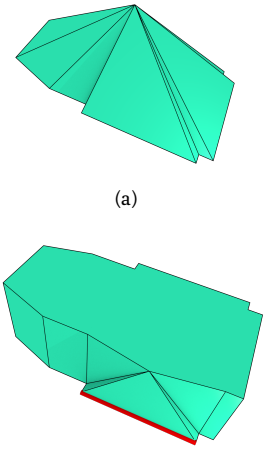


Figure 6.14: Cells can be collapsed by moving all of their vertices to the same location. Shown here are the collapse of: (a) a face and (b) an edge (in red).

After such transformations, it is possible to easily detect and remove degenerate cells, as they can be identified because they have all their vertices at the same location. In the case of a topological model, such as a generalised or combinatorial map, the topological relationships around the collapsed cells will have to be recomputed. However, this is possible to achieve either by finding the changed incidences and adjacencies, or by regenerating all topological relationships using the method described in Chapter 7. For the latter, it is simply necessary to ignore all degenerate cells (as defined by having all their vertices at the same embedding in $\mathbb{R}^n$).

6.6 Conclusions

Extrusion in the GIS sense has a natural extension into a dimension-independent formulation. It can be used to load existing 2D or 3D data into a higher-dimensional structure, either directly or as a base for further (dimension-independent) operations, such as the generalisation operations described in §6.5. The conceptual simplicity of extrusion makes it particularly suitable for higher-dimensional data, as the complex problem of its generation can be reduced to the definition of extrusion intervals per n-cell in an n-dimensional cell

complex. By ensuring that the input n-cells do not overlap—which is a general precondition in a geometric cell complex—, and the extrusion intervals per cell also do not overlap, it is easy to guarantee that the output cell complex forms a valid $(n+1)$-dimensional space partition.

The dimension-independent extrusion formulation has been realised into an algorithm, presented in this chapter, which supports multiple intervals per input object and it is also memory efficient—only three layers of darts of the same size as the input map need to be kept in main memory at the same time. It is also relatively fast, with a worst case complexity of $O(ndr)$ in the main algorithm, where n is the extrusion dimension[114], d is the total number of darts in the input map and r is the total number of intervals in the input, but offers better complexity in practice.

114: In practice a small constant.

Moreover, the algorithm has been implemented using CGAL Combinatorial Maps and the source code has been made publicly available under a permissive licence. It has also been tested with publicly available datasets commonly used in GIS. Apart from being applied cell by cell, it can also be combined with other operations, such as the generalisation operations described here or building certain nD cells incrementally, where more complex models are required only for a subset of the objects in a model.

While the implementation presented here performs well enough for typical GIS datasets, it could be made faster by integrating the extrusion intervals into the embeddings of the cells, allowing for constant time access to the extrusion intervals. This would make it possible to support much larger datasets and make the implementation match the theoretical complexity of the algorithm. In order to improve memory usage, another possibility would be implementing the algorithm in a more compact representation of a simplicial complex, e.g. Boissonnat and Maria [2012].

In the future, it would be interesting to consider other types of models that can be generated using extrusion-like algorithms, such as sweeps and n-dimensional cells of revolution. These are both simple extensions either using either non-linear embeddings or discretised approximations with small cells with linear geometries. A more complex extension would involve starting from a *series of cell complexes* that each form a planar partition, such as series of 3D models of the same region at different levels of detail. While most of the algorithm described here should work directly on this latter example, managing the differing boundaries of the cells along the n-th dimension could be challenging—possibly requiring the computation of all intersections in an arrangement of $(n-1)$-cells.

Incremental construction 7

One more method to the extrusion of Chapter 6, *incremental construction* exploits the property that an n-cell can be defined based on a set of $(n-1)$-cells known to form its complete (closed) boundary. In practice, defining an n-cell in this manner is significantly more complex than doing so based on extrusion. However, unlike extrusion it permits the creation of cells of an *arbitrary shape*. It can also be applied cell by cell in increasing dimension, starting with the construction of isolated 0-cells (embedded in $\mathbb{R}^n$) first, and then constructing 2-cells, 3-cells and further based on their boundary, allowing for the *incremental construction* of objects of any dimension—hence the name given here.

This chapter describes an operation that permits the aforementioned process to be easily applied in practice using a topological data structure. Based on combinatorial maps (§4.3.6) and their fundamental operations (§5.1.1), it connects the separate boundary $(n-1)$-cells together by computing the appropriate adjacency relationships between them, encapsulating low-level details such as the creation of new individual combinatorial elements and setting the correct orientation for the existing ones.

The chapter begins with some background on the operations and a description of the overall approach in §7.1. It then explains the steps needed to create the cells of each dimension in §7.2. §7.3 describes how the approach was implemented based on the Computational Geometry Algorithms Library (CGAL). §7.4 summarises some experiments based on this implementation, generating relatively large objects in up to 4D. Finally, §7.5 concludes the chapter with the possibilities to use incremental construction to build higher-dimensional datasets.

Most of this chapter is based on the paper:

- **Constructing an n-dimensional cell complex from a soup of $(n-1)$-dimensional faces**. Ken Arroyo Ohori, Guillaume Damiand and Hugo Ledoux. In Prosenjit Gupta and Christos Zaroliagis (eds.), *Applied Algorithms. First International Conference, ICAA 2014, Kolkata, India, January 13–15, 2014. Proceedings*, Lecture Notes in Computer Science 8321, Springer International Publishing Switzerland, Kolkata, India, January 2014, pp. 37–48.

7.1 Background and overall approach

Based on the Jordan-Brouwer separation theorem [Lebesgue, 1911; Brouwer, 1911], it is known that a subset of space homeomorphic to an $(n-1)$-dimensional sphere[116] S^{n-1} in the n-dimensional Euclidean space $\mathbb{R}^n$ divides the space into two connected components: the *interior*, which is the region bounded by the sphere, and the *exterior*, which is the unbounded region in which the sphere is a hole. This means that the principles of boundary representation as described in §3.1.2 extend to higher dimensions, and so an n-cell in a cell complex can be described (and therefore constructed) based a set of $(n-1)$-cells that are known to form its complete (closed) boundary[117].

116: A generalisation of the concept of a sphere in arbitrary dimensions. A 0-sphere is the pair of points, a 1-sphere is a circle, and a 2-sphere a sphere. As opposed to disks and balls, circles and spheres are hollow.

117: This is true in practice. However, there are fractal-like pathological objects where this is not the case because their interior or exterior components are not homeomorphic to disks [Alexander, 1924]. While they do exist, this type of objects are certainly out of scope here.

The *incremental construction* method proposed in this chapter exploits this property by applying this process cell by cell in increasing dimension. Isolated vertices are first constructed, which are embedded in $\mathbb{R}^n$ and are uniquely defined based on their coordinates. These vertices can then be connected to form individual edges, or instead to directly form cycles of implicit edges representing faces, as in most of the typical GIS approaches described in §3.2.2. Sets of faces can be connected to form volumes, sets of volumes to form 4-cells and so on.

In a similar manner as extrusion, presented in Chapter 6, this *incremental construction* process significantly reduces the conceptual complexity of defining and creating higher-dimensional objects. The $(n-1)$-dimensional boundary of an n-cell is much easier to conceive than the original n-cell because it can be subdivided into multiple simpler $(n-1)$-cells, which can be individually described and constructed using the same method.

However, applying this incremental construction process using a topological data structure is not that simple, as it involves many intricate small problems: finding the topological relationships between the cells, appropriately connecting them, keeping track of multiple connected components, avoiding the creation of duplicate cells (as part of the boundary of independently-described higher-dimensional cells), changing the orientation of cells on the fly (since those that have been created separately will often have incompatible orientations), and detecting non-manifold shapes (which would result in invalid structures), among others.

The incremental construction operator presented in this chapter solves all of the aforementioned issues efficiently. It is based on n-dimensional combinatorial maps with linear geometries and it is fully dimension-independent. As it generates all the incidence and adjacency relationships between $(n-1)$-cells in an n-dimensional cell complex, it can also be used to obtain these relationships when needed, such as when multiple datasets are combined into one or

when a non-topological representation is instead used (e.g. the Simple Features-like approach shown in §4.2.3).

In order to be efficient, the algorithm uses two basic techniques: (i) indexes on the lexicographically smallest vertex (§5.3) of certain cells, which are used in order to keep track of individual cells (which might be disconnected) and to access cells efficiently; and (ii) signature-generating traversals for specific cells [Gosselin et al., 2011], which are used to efficiently compare whether two cells are equivalent by checking if it is possible to find an isomorphism f that maps corresponding darts with equivalent (β) or reversed (β^{-1}) relations and which are embedded at the same location in $\mathbb{R}^n$, as was shown in §5.5.

The incremental construction operation as described in this chapter is applicable only to perfect data. The $(n-1)$-cells bounding an n-dimensional cell should thus form a manifold and perfectly match each other at their common $(n-2)$-dimensional boundaries. However, the method can be easily extended to handle some more complex configurations, such as by cleaning the input data using the methods described in Chapter 10.

7.2 Incremental construction of primitives per dimension

The idea of the incremental construction algorithm is to construct cells individually, using lower-dimensional cells that have already been constructed in order to describe part of the boundary of a higher-dimensional cell. In terms of the darts of a combinatorial map, this sometimes implies the reuse of existing darts, and sometimes the creation of new ones which are connected to the existing ones. There is however no strict requirement that the cells are created in strictly increasing dimension, and so new cells can be easily added to an existing cell complex.

Due to the need to embed 0-cells into a point in space, as well as the oriented nature of a combinatorial map, the incremental construction method is different for 0-, 1- and 2-cells. 3-cells and those of higher dimensions follow a unified procedure. These cases are therefore described separately below, using as an example the creation of the pair of adjacent tetrahedra shown in Figure 7.1.

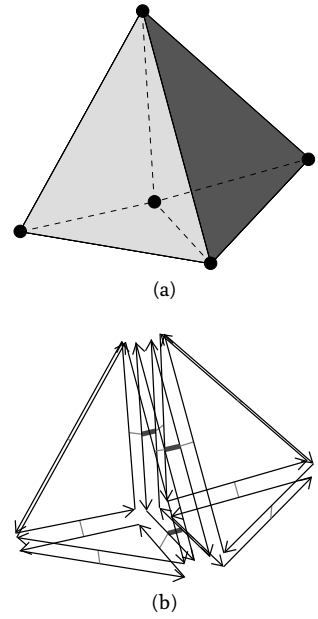


Figure 7.1: (a) Two adjacent tetrahedra and (b) their representation as a combinatorial map.

7.2.1 Vertices (0-cells)

The aim of the process of 0-cell construction is to create an isolated dart and an associated point embedding structure for every 0-cell, while avoiding having duplicate embedding structures having the

same point coordinates. By making point embeddings unique—as defined by their coordinates—, they can be used to compare 0-cells without checking an entire tuple of coordinates. Moreover, by maintaining an index of all point embeddings and links from every point embedding to a dart there, it is possible to use this index to access the combinatorial structure at that point. In this manner, it is possible to find either an already free dart that can be reused, or a non-free dart that is part of a larger combinatorial structure, which can be then copied and its copy used instead.

Thus, calling a function `make_0_cell(`$x_0, \dots, x_n$`)` with the coordinates $x_0, \dots, x_n$ describing an n-dimensional point, should use the index of 0-cells to return an existing dart embedded at that location (using an existing point embedding) if one exists, or a new dart embedded at that location (using a new point embedding) otherwise. In the latter case, the new point embedding and its associated dart should be added to the index. The result of calling this function for all the point coordinates of the vertices in Figure 7.1 is shown in Figure 7.2. Note that the result is identical whether the function is called once for every unique vertex, or multiple times (e.g. once for every vertex in every face or volume).

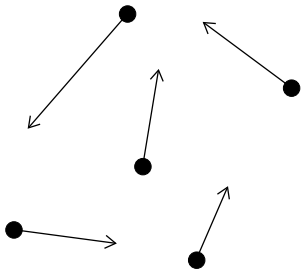


Figure 7.2: Constructing the 0-cells of Figure 7.1 consists of creating exactly one dart embedded at each point. These darts are isolated (i.e. not linked by any β relations to the other darts). The direction of the arrows shown here is arbitrary.

7.2.2 Edges (1-cells)

Generally, it is best to skip the generation of 1-cells and proceed directly to the creation of 2-cells from sequences of points. In order to represent an isolated edge in a combinatorial map not one but *two* darts are required: one embedded at each of the two vertices bounding the edge. More precisely, taking into account the (arbitrary) orientation defined for the edge, the dart embedded at the *origin* of the edge is connected to the *destination* by β_1, and the destination is connected to the origin by β_1^{-1}. Having two darts per edge is not a major problem, but it is wasteful as it often creates unnecessary darts and unnecessary connections between them that would later be deleted. For instance, if a single face that uses all the edges is created from its bounding edges, half of the darts lose their original purpose (i.e. to store a connection to their otherwise unlinked point embeddings) and will be eliminated[118], which is accompanied with having to reset the β relationships pointing to them.

118: These could be those embedded at the edges' origin, destination or a mixture of the two.

In addition, as shown in §3.2.2, polygons can be easily described as an ordered sequence of vertices connected by implicit line segments between each consecutive pair. Therefore, incrementally constructing 1-cells often brings no efficiency gains or practical benefits, and so it is normally best to skip dimension one, constructing 2D facets from a sequence of 0D points, 3D volumes from their 2D faces, 4-cells from their 3-cell faces, and so on.

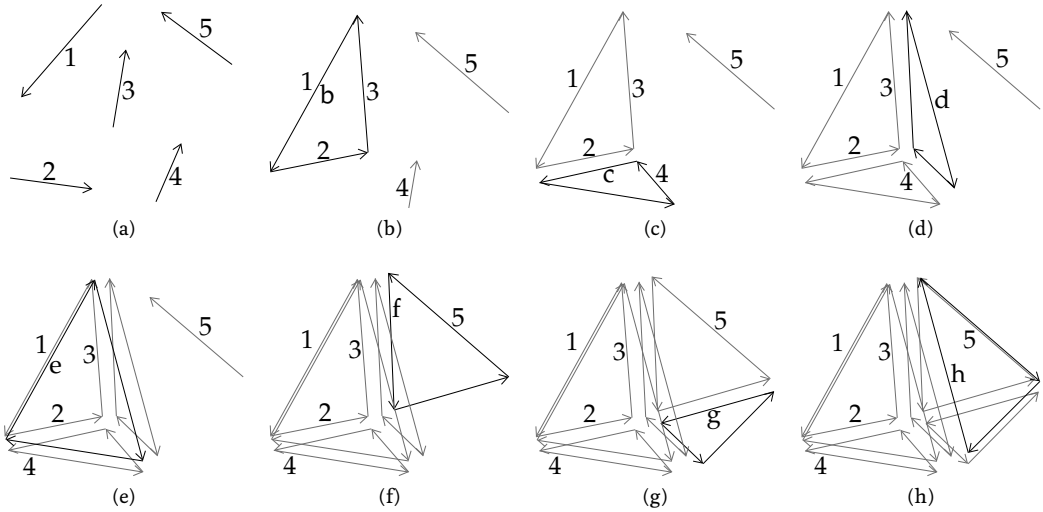


Figure 7.3: The construction of the 2-cells of Figure 7.1. (a) Initial configuration: one dart per vertex. (b) After b = `make_2_cell(1,2,3)`. (c) After c = `make_2_cell(2,4,3)`. (d) After d = `make_2_cell(1,4,3)`. (e) After e = `make_2_cell(1,4,2)`. (f) After f = `make_2_cell(1,3,5)`. (g) After g = `make_2_cell(5,3,4)`. (h) Final result, after h = `make_2_cell(4,5,1)`.

However, there is an exception to this rule, as isolated edges and polygonal lines sometimes do need to be explicitly represented. In these cases, it is possible to simply follow the process described for 2-cells below, omitting sewing the last dart of the line segment or polygonal line to the first dart (and vice versa), and appropriately using the 1-cells index to find if a given 1-cell already exists, or otherwise to index the newly created 1-cell or 1-cells (in the case of a polygonal line).

7.2.3 Faces (2-cells)

Starting from the unique 0-cells obtained from the procedure presented in §7.2.1, in order to create a 2-cell from a sequence of 0-cells, three general steps are needed:

1. The procedure first checks if the 2-cell that is being requested already exists[119]. Just as in the creation of 0-cells, a 2-cell being requested might have already been created, either independently or as part of a 3-cell. This can be easily checked using the index of 2-cells with the lexicographically smallest vertex of the 0-cells being passed, and finding if from a dart starting at the lexicographically smallest vertex and following the β_1 or β_1^{-1} relations, one passes through the same point embeddings as those passed to this construction method.

119: This test can also be done at the end of the process, which makes it possible to use the easy cell comparison method described in §5.5. In that case, the newly created cells (in the form of darts) that are found to be redundant should be deleted and removed from (or not added to) the indices.

2. Each of these 0-cells might be a 1-free dart d (i.e. $\beta_1(d) = \varnothing$) and thus have no dart linked to it by β_1, i.e. $\beta_1^{-1}(d) = \varnothing$ or $\forall d' \nexists \beta_1(d') = d$, or if β_1^{-1} is not stored, $\nexists d' \mid \beta_1(d') = d$, *in which case it can be used directly*, such as in Figure 7.3b. It can also be a non 1-free dart or one that has a dart linked by β_1 to it, which means that it is used as part of a 1-cell (and possibly other higher dimensional cells).

   The darts that are already used as part of 1-cells are more difficult to handle, as they *can be reused only when the 1-cell they are part of is also part of the boundary of a geometrically identical 2-cell as the one that will be constructed using this method*. This needs to be tested in both possible orientations for the sequence of 0-cells, possibly resulting in the reversal of the orientation of some 1-cells. If a given dart *cannot be reused* (because it is part of a 1-cell that is not part of the 2-cell to be created), a copy of it has to be created. The result of this step is thus a list that contains for every 0-cell a reusable existing dart or a newly created dart.

3. The darts obtained in the previous step are 1-sewn sequentially (using β_1 and β_1^{-1}), and the last is 1-sewn to the first, forming a closed cycle. During these sewing operations, the function checks whether every newly created edge (consisting of a pair of linked darts) is present in the index of 1-cells. If a given 1-cell is already there, the new edge's dart is linked to its edge embedding structure. If a 1-cell is not there, the edge is added to the index[120], a new edge embedding is created, and the edge's dart is linked to it. Finally, the newly created 2-cell is then added to the index of 2-cells.

   In order to verify that the 2-cell being generated is a quasi-manifold, it is useful to assert that all darts that are 1-free and have their β_1^{-1} relation set to $\varnothing$ at the beginning of this third step, i.e. those that were not copied, should continue to be 1-free and have a β_1^{-1} set to $\varnothing$ until they are sewn. This condition, which is applied differently to the 1-cell, ensures that a vertex is not used twice within the same face, and as such, that the 2-cell is a quasi-manifold.

120: Here, depending on the kind of expected input data, it might be more convenient to index edges at their origin vertex rather than at their lexicographically smallest.

Figure 7.3 shows the incremental construction procedure for all the 2-cells of Figure 7.1, which consists of 7 2-cells and is thus obtained by 7 calls to the `make_2_cell` method.

7.2.4 Volumes (3-cells) and higher

The method to create i-cells from their $(i-1)$-cell boundaries is identical for all $i \geq 3$, allowing for a fully dimension-independent function to be created. This function consists of four general steps:

1. First of all, the procedure checks whether each $(i-1)$-cell that is passed is $(i-1)$-free or not. If it is $(i-1)$-free, it is reused as part of the i-cell, such as in Figure 7.4a. If it is not $(i-1)$-free, then it is already part of a different i-cell, so a copy of it is made *with a reversed orientation*, as shown in Figure 7.4b. This copy can then be used for the construction of the i-cell.

   For a given dart d that is known to be part of the $(i-1)$-cell, this copy can be made by taking all the darts in the orbit $C = \langle \beta_1, \dots, \beta_{i-2} \rangle (d)$, creating a new set of darts C', and inserting a new corresponding dart in C' for every dart in C. Using a function $f : C \to C'$ that maps a dart in C to its corresponding dart in C', for every dart $c \in C$ the β_1 relations are set as $\beta_1^{-1}\left(f(c)\right) = f\left(\beta_1(c)\right)$ and $\beta_1\left(f(c)\right) = f\left(\beta_1^{-1}(c)\right)$. For the ones of higher dimensions, they are set such that $\forall 2 \leq j \leq i-2$, $\beta_j\left(f(c)\right) = f\left(\beta_j(c)\right)$.

   Note that the combinatorial structures are copied with reversed orientation, as this ensures that the two (old and new) can be directly i-sewn together, if necessary.

2. A temporary *ridge index* of the i-cell is built, containing all the $(i-2)$-cells on the boundary of all $(i-1)$-cells that have been passed to the construction method—not all the $(i-1)$-cells in the cell complex—using their lexicographically smallest vertices, such that their index entries link to a usable dart in the $(i-2)$-cell with a point embedding at the lexicographically smallest vertex. This dart should be one that is $(i-1)$-free, either because it was already free as part of the combinatorial structure of the passed $(i-1)$-cells, or because it is a copy (with reversed orientation) of one that was not $(i-1)$-free.

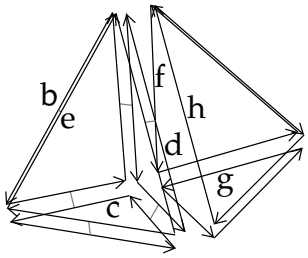
3. Using the ridge index, $(i-1)$-cells (i.e. facets) are $(i-1)$-sewn along their common $(i-2)$-cell boundaries (i.e. ridges). For this, for every $(i-2)$-cell on the boundary of an $(i-1)$-cell passed to the function, exactly one matching $(i-2)$-cell should be found in the index, which should be equivalent as compared using the method described in §5.5 and might be of the same or opposite orientation (but should not be part of the same $(i-1)$-cell or matched to itself). This criterion ensures that a quasi-$(i-1)$-manifold will be constructed.

   If the two $(i-1)$-cells are sewable along their common $(i-2)$-cell boundaries, i.e. they have compatible orientations, they are directly sewn together. Otherwise, the orientation of the entire connected component of either of the two $(i-1)$-cells should be reversed. If the two $(i-1)$-cells have incompatible orientations but are part of the same connected component, the cell that is being requested is unorientable, and thus cannot be represented using combinatorial maps. Although not discussed fur-

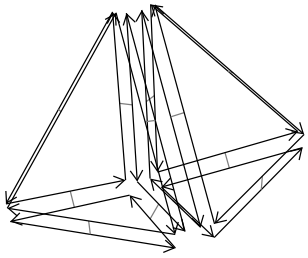
ther here, note that as long as it does forms a quasi-manifold it can however be represented using *generalised maps*.

4. Using the index of i-cells, the newly constructed i-cell is compared to others to check if it had already been created, which is also done using the lexicographically smallest vertex of the cell. If an equivalent i-cell (with the same or opposite orientation) is found, the function deletes the newly created darts of the cell and their corresponding index entries, and instead finally returns the existing i-cell. If an equivalent i-cell is not found, the newly created cell is added to the index and returned.

Figure 7.4 shows the incremental construction procedure for the two 3-cells of Figure 7.1.



(a)



(b)

Figure 7.4: The construction of the 3-cells of Figure 7.1. (a) After `make_3_cell(`b,c,d,e`)`. (b) Final result, after `make_3_cell(`e,f,g,h`)`.

7.3 Implementation and complexity analysis

The incremental construction algorithm has been implemented in C++11 and made available under the open source MIT licence at `https://github.com/kenohori/lcc-tools`. As the extrusion related code (§6.3), it requires and builds upon the CGAL packages Combinatorial Maps and Linear Cell Complex, among others.

For this prototype implementation, the lexicographically smallest vertex indices were implemented as C++ Standard Library `maps`[121] for every dimension, using point embeddings as keys and `lists` of darts as values. Each dart in a `list` represents a separate cell (of the dimension of the index) that has that point as its lexicographically smallest. A custom comparison function is passed as a template so that the points are internally sorted in lexicographical order. As a `map` has $O(\log n)$ search, insertion and deletion times and $O(n)$ space [ISO, 2015, §23.4], all of these operations can be performed efficiently.

If a strictly incremental approach is followed, creating all i-cells before proceeding to the $(i+1)$-cells, it is not necessary to maintain indices for all the cells of all dimensions at the same time. The only ones that are needed are those for: all i- and $(i-1)$-cells, as well as a temporary index for the $(i-2)$-cells on the boundary of the $(i-1)$-cells for the i-cell that is currently being constructed. This means that only three indexes—one of which likely covers only a small part of the dataset—need to be kept at a given time.

Most of these indices can be easily built incrementally, adding new cells as they are created in $O(\log c)$, with c the number of cells of that dimension, assuming that the smallest vertex and a dart embedded there are kept during its construction. The complexity of building any index of cells of any dimension is thus $O(c \log c)$ and it uses $O(c)$ space.

121: The exact implementation depends on the library that is used, but normally they are red-black trees.

Checking whether a given cell already exists in the cell complex is more complex. Finding a list of cells that have a certain smallest vertex is done in $O(\log c)$. In an unrealistically pathological case, all existing cells in the complex could have the same smallest vertex, leading to up to c quadratic time cell-to-cell comparisons just to find whether one cell exists. However, every dart is only part of *a single* cell of any given dimension[122], so while every dart could conceivably be a starting point for the comparison, a single dart cannot be used as a starting point in more than one comparison, and thus a maximum of d_{complex} identity comparisons will be made for *all* cells, with d_{complex} being the total number of darts in the cell complex. From these d_{complex} darts, two identity comparisons are started, one assuming that the two cells (new and existing) have the same orientation, and one assuming opposite orientations. Each of these involves a number of dart-to-dart comparisons in the canonical representations that *cannot* be higher than the number of darts in the smallest of the two cells. The number of darts in the existing cell is unknown, but starting from the number of darts in the newly created cell (d_{cell}), it is safe to say that no more than d_{cell} dart-to-dart comparisons will be made in each identity test, leading to a worst-case time complexity of $O(d_{\text{complex}} d_{\text{cell}})$. Note that this is similar to an isomorphism test starting at every dart of the complex.

122: This is true for the type of combinatorial maps with linear geometries that are handled here, but not necessarily so in the general case.

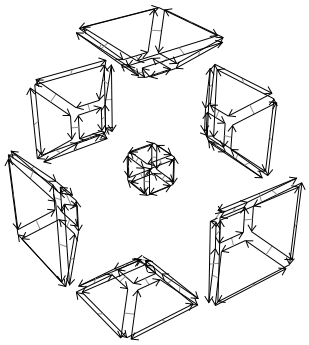
Finally, creating an i-cell from a set of $(i-1)$-cells on its boundary is more expensive, since the $(i-2)$-cell (ridge) index needs to be computed for every $(i-1)$-cell. Following the same reasoning as above, it can be created in $O(r \log r)$ with r the number of ridges in the i-cell, and uses $O(r)$ space. Checking whether a single ridge has a corresponding match in the index is done in $O(d_{\text{cell}} d_{\text{ridge}})$, with d_{cell} the number of darts in the i-cell and d_{ridge} the number of darts in the ridge to be tested. Since this is done for all the ridges in an n-cell, the total complexity of this step, which dominates the running time of the algorithm, is

$$\sum_{\text{ridges}} O(d_{\text{cell}} d_{\text{ridge}}) = O(d_{\text{cell}}^2).$$

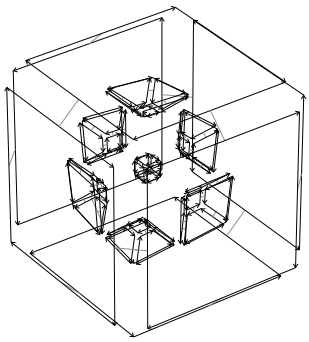
The analyses given above give an indication of the computational and space complexity of the incremental algorithm as a whole. However, it is worth noting that in realistic cases the algorithm fares far better than in these worst-case scenarios: the number of cells that have a certain smallest vertex is normally far lower than the total number of cells in the complex, most of their darts are not embedded at the smallest vertex, and from these darts most identity comparisons will fail long before reaching the end of their canonical representations.

Finally, one more nuance can affect the performance of this approach. When two connected components of darts with incompat-

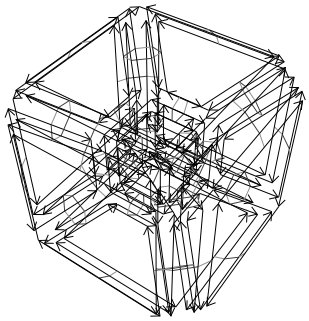
ible orientations have to be joined, the orientation of one of these has to be reversed. This is easily done by obtaining all the connected darts of one of the connected components, preferably the one that is expected to be smaller, and reversing their orientation 2-cell by 2-cell. Every dart d in a 2-cell is then 1-sewn to the previous dart in the polygonal curve of the 2-cell (i.e. $\beta_1^{-1}(d)$). A group of n darts can thus have its orientation reversed in $O(n)$ time. This is not a problem in practice since GIS datasets generally store nearby objects close together, but if a cell complex is incrementally constructed in the worst possible way, i.e. creating as many disconnected groups as possible, this could have to be repeated for *every* cell of every dimension, creating a very inefficient process.



(a)



(b)



(c)

Figure 7.5: A tesseract as a combinatorial map: (a) the darts of its 7 inner cubes, (b) the darts of its 8 cubes, and (c) all of its darts shown together.

7.4 Experiments

Simple comparisons with valid primitives

The CGAL Linear Cell Complex package provides functions to generate a series of primitives (line segments, triangles, quadrangles, tetrahedra and hexahedra) which are known to be created with correct geometry and topology, and can then be sewn together to generate more complex models. Models constructed in this manner were thus created independently and compared to those incrementally constructed using the method presented in this chapter. By using the approach shown in §5.5, it was possible to validate that they were equivalent.

A tesseract

In order to present an example in more than three dimensions, a tesseract was also incrementally constructed using the approach presented in this chapter, which is shown in Figure 7.5. A tesseract is a 4-cell bounded by 8 cubical 3-cells, each of which is bounded by 6 square 2-cells. It thus consists of one 4-cell, 8 3-cells, 24 2-cells, 32 1-cells and 16 0-cells.

Using the approach presented here, an empty 0-cell index is first created. Then, the 16 vertices of the tesseract, each vertex p_i being described by a tuple of coordinates (x_i, y_i, z_i, w_i), were created as p_i = `make_0_cell(`x_i, y_i, z_i, w_i`)`, which returns a unique dart embedded at each location that is also added to the 0-cell index. At this point, the algorithm has built an unconnected cell complex consisting solely of 16 completely free darts.

An empty index of 2-cells is then created. Each of the tesseract's 24 square facets are then built based on their vertices as f_i = `make_2_cell(`p_j, p_k, p_l, p_m`)`, which 1-sews (copies of) these darts in a

loop and returns the dart embedded at the smallest vertex of the facet. These are added to the index of 2-cells. Since every vertex is used in 6 different 2-cells, each dart would be copied 5 times. The cell complex at this point thus consists of 24 disconnected groups of 4 darts each.

Next, an empty index of 3-cells is created and the index of 0-cells can be deleted. For each of the 8 cubical 3-cells, a function call of the form v_i = `make_3_cell(`$f_j, f_k, f_l, f_m, f_n, f_o$`)` is made. At this point, an index of the 1D ridges of each face is built, which is used to find the 12 pairs of corresponding ridges that are then be 2-sewn together. When a 3-cell is created, it is added to the index. Since every face bounds two 3-cells, each dart is duplicated once again, resulting in a cell complex of 8 disconnected groups of 24 darts each.

Finally, the tesseract is created with the function t = `make_4_cell(`$v_1, v_2, \ldots, v_8$`)`. This can use the index of 2-cells to find the 24 corresponding pairs of facets that are then 3-sewn to generate the final cell complex.

The validity of this object was tested by checking whether it formed a valid combinatorial map, testing whether each cube was identical to a cube created with the Linear Cell Complex package, and manually verified the β links of its 192 darts.

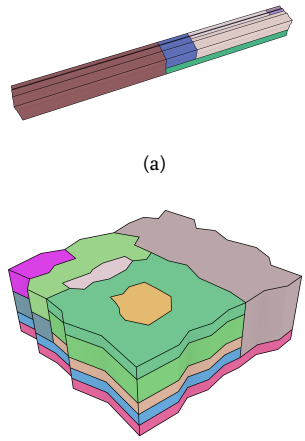
2D+scale data

In order to test the incremental construction algorithm and its applicability to data incorporating non-spatial characteristics, a few 2D+scale datasets from Meijers [2011] using ATKIS data[123], were also incrementally constructed. As shown in Figure 7.6, these datasets model the generalisation of a planar partition as a set of stacked prisms. Both of the simple datasets of this figure were created successfully in under a second.

A larger dataset, shown in Figure 7.7 and consisting of 698 polyhedra with a total of 457 185 faces, was used as a benchmark to test the performance of the algorithm. This dataset was processed without errors in roughly 30 minutes, including validation tests to ensure that every face created was a valid combinatorial map and that the faces of a volume formed a closed quasi-manifold.

In this latter dataset, an additional test was made using its first 250 polyhedra, which lie on the top of Figure 7.7, comparing the time used for the construction of vertices, faces and volumes with and without the use of indices. The results of both methods were equivalent using the approach shown in §5.5. On average, using indices resulted in a faster vertex construction time by a factor of 2 200, faster face construction by a factor of 56 and faster volume construction by a factor of 38. However, as Figure 7.8 shows, these speed gains

123: http://www.bkg.bund.de/nn_147094/SharedDocs/Download/Barrierefreie-Textversionen/EN-InfoMaterial/EN-Text-Vector-Data.html



(a)

(b)

Figure 7.6: Simple 2D+scale datasets from Meijers [2011], which represent the generalisation of a planar partition by merging areas. The dataset shown in (a) has realistic scale intervals such that the generalisation operations are performed at scales that depend on the area of a polygon, while the dataset in (b) uses equally spaced generalisation operations.

are not uniform throughout the 250 polyhedra. The speed gained from using an index on the vertices increases roughly linearly on the number of constructed vertices, while that gained on from the faces index remains roughly constant between a factor of 50 and 60, and that gained from the volumes index tends to slowly increase as well.

7.5 Conclusions

Creating computer representations of higher-dimensional objects can be complex. Common construction methods used in 3D, such as directly manipulating combinatorial primitives, or using primitive-level construction operations (e.g. Euler operators [Mäntylä, 1988]), rely on our intuition of 3D geometry, and thus do not work well in higher dimensions. It is therefore all too easy to create invalid objects, which then cannot be easily interpreted or fixed.

The incremental construction method proposed in this section follows a completely different approach, which has a solid underpinning in the Jordan-Brouwer separation theorem [Lebesgue, 1911; Brouwer, 1911]. By exploiting the principles of boundary representation, it constructs an i-cell based on a set of its bounding $(i-1)$-cells. Since individual $(i-1)$-cells are easier to describe than the i-cell, it thus subdivides a complex representation problem into a set of simpler, more intuitive ones. The method can moreover be incrementally applied to construct cell complexes of any dimension, starting from a set of vertices in $\mathbb{R}^n$ as defined by a n-tuple of their coordinates, and continuing with cells of increasing dimension—optionally creating edges from vertices, then faces from vertices or edges, volumes from faces and so on.

While a set of $(i-1)$-cells bounding an i-cell can be said to already form a complete representation of an i-cell, this is not sufficient for

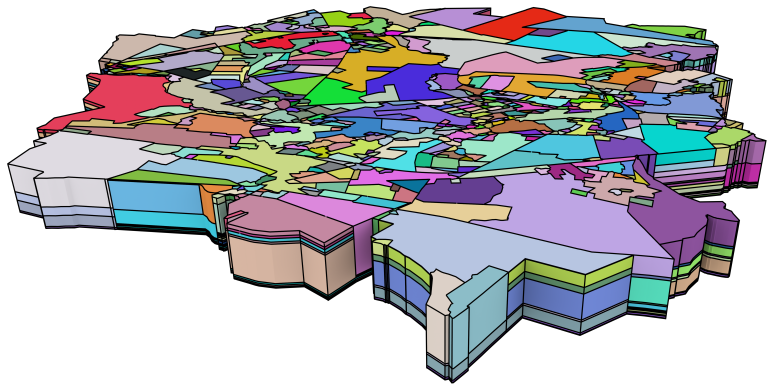


Figure 7.7: A large 2D+scale dataset from Meijers [2011], which uses equally spaced generalisation operations that merge two adjacent polygons.

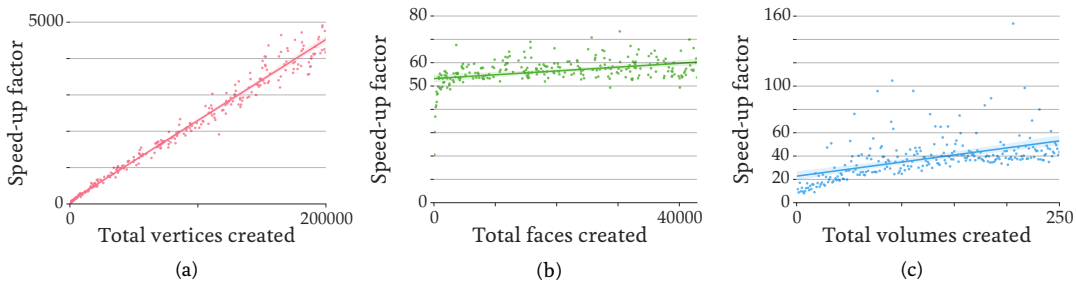


Figure 7.8: Construction time speed-up from the use of indices on the (a) vertices, (b) faces and (c) volumes.

its representation in a topological data structure, which requires the topological relationships between the $(i-1)$-cells to be computed. The incremental construction algorithm proposed in this section thus computes the relationships that are required for two data structures: generalised or combinatorial maps. However, these relationships are also applicable to most other data structures.

The algorithm is efficient due to its use of indices using the lexicographically smallest vertex of every cell per dimension, as well as an added index using the lexicographically smallest vertex of the bounding ridges of the cell that is being built. It generates an i-cell in $O(d^2)$ in the worst case, with d the total number of darts in the cell. However, it fares markedly better in real-world datasets, as cells do not generally share the same lexicographically smallest vertex. By checking all matching ridges within a cell's facets, the algorithm can optionally verify that the cell being constructed forms a combinatorially valid quasi-manifold, avoiding the construction of invalid configurations.

A publicly available implementation of the algorithm has been made based on CGAL Combinatorial Maps, and its source code has been released under a permissive licence. It is worth noting that it is one of very few general-purpose object construction methods that has been described and implemented for four- or higher-dimensional cell complexes.

The implementation has been tested with simple 2D–4D objects, as well as with large 2D+scale datasets from Meijers [2011]. The constructed objects were tested to verify that they form valid combinatorial maps. The small objects were also manually inspected, visualised, and where possible, they were compared with equivalent objects known to be valid using the method discussed in §5.5.

While the incremental construction operation as described in this chapter works only with perfect data, small modifications could make it applicable to additional configurations. For instance, merging adjacent cells that can be perfectly described by a single cell with

linear geometry (e.g. collinear edges and coplanar faces) as a preprocessing step can be used to handle cases where cells are subdivided only on one side of a boundary. Snapping points together can solve several common invalid configurations, such as those described in Chapter 10 and in Diakité et al. [2014].

Finally, the logical step forward at this point would be the use of the incremental construction algorithm for the creation of large higher-dimensional datasets. However, since to the best of my knowledge there are no large space-filling higher-dimensional datasets currently available—without even considering any validity criteria—, such tests could not be conducted within this thesis. The generation of such a higher-dimensional dataset is thus a necessary first step in this direction and is considered as a related piece of future work.

Linking corresponding 3D models | 8

Chapters 6 and 7 presented in detail two low-level construction algorithms for nD objects, both of which operated on level of individual primitives. These are valuable in their own right, but in order to ease the use of higher-dimensional operations in practice, it is necessary to develop high-level operations based on them which can actually be used by practitioners. This chapter thus describes at a conceptual level how one such high-level operator can be defined, which creates a 4D representation from a series of existing 3D representations at different levels of detail.

The chapter starts by introducing some background in §8.1, explaining the motivation behind such an operation and covering the principles of linking operations with 2D/3D objects. §8.2 describes four linking schemes that can be used to construct 4D models from a set of 3D objects, discussing the advantages and disadvantages of each method in terms of their feasibility in practice and of the properties that the 4D model would have. §8.3 presents several use cases to demonstrate how the different schemes result in objects with different characteristics, where the best scheme can be considered to be application-dependent. §8.4 shows a concrete example in detail, implementing a use case that combines most of the linking schemes from §8.2. §8.5 concludes the chapter with a discussion of the further possibilities of this method.

Most of this chapter is based on the paper:

- **Modelling a 3D city model and its levels of detail as a true 4D model**. Ken Arroyo Ohori, Hugo Ledoux, Filip Biljecki and Jantien Stoter. *ISPRS International Journal of Geo-Information*, 4(3), September 2015, pp. 1055–1075.

8.1 Motivation and background

3D city models of the same region are often created at multiple levels of detail (LODs). For instance, in the CityGML standard [Gröger et al., 2012] five discrete LODs are defined (Figure 3.23 on page 49), which range from the 2D footprint of the building up to a representation where the windows, doors and walls and even indoor objects

are all modelled in detail. This allows a user to choose the most appropriate LOD for a given application, balancing the better results that are obtainable using more detailed models with the higher computational requirements that are necessary to obtain them [Biljecki et al., 2014b].

However, the creation of these models is a complex task that needs to be performed continuously, as 3D city models need to be kept up to date [Zlatanova and Holweg, 2004; Kolbe et al., 2005]. Given the large size and complexity of current 3D city models, it can be very beneficial to have incremental updates to a model which affect only a building and its immediate surrounding area [Döllner et al., 2006]. These can take place as buildings and other city objects (e.g. roads, utility infrastructure, city furniture, etc.) are built, modified and destroyed.

In order to apply such incremental update processes to 3D city models at multiple LODs, links between related objects are crucial. Given an object at a certain LOD, links usually point to its incident and adjacent objects at the same LOD (i.e. the topological relationships that are most common in GIS), as well as to its *corresponding* objects at other LODs, even when these objects are of different dimension (e.g. when a thin polyhedron in a higher LOD is collapsed to a polygon in a lower LOD). These links can then be used to propagate changes to other LODs [van Oosterom and Stoter, 2010] or to apply consistency checks to new or newly altered objects [Gröger and Plümer, 2011b], among other operations that are part of a robust update process.

In theory, a series of LODs might be derived from an automatic generalisation process [Weibel, 1997] and thus the exact correspondences between objects can be already known, but as discussed in §3.4, fully automatic 3D generalisation is very complex and has not yet been achieved in practice. The different LODs of a model are therefore usually acquired independently, collected with different techniques, often for different purposes, and thus the resulting representations do not necessarily have easy-to-identify correspondences. The same object can be slightly displaced at different LODs, an object can be an aggregate of other objects (think of a terraced house: either each house is individually represented as a volume or one single volume is used for the whole row), or can be modelled in an entirely different way. In fact, as shown in Figure 8.1, it is possible that there are no common geometries (i.e. 0-, 1-, 2-, or 3-cells with the same geometry) across a series of models at different LODs.

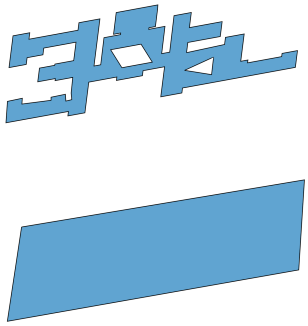


Figure 8.1: Two LODs of a building footprint. Note that there are no vertices, edges or faces with the same geometry in both LODs, and that many primitives in the higher LOD are equivalent to a single one in the lower LOD.

In order to join multiple separate representations, stored as independent datasets, it is therefore first necessary to find the correspondences between (equivalent) objects at different LODs. As described in 2D by Hampe et al. [2003], when an automatic generalisation process is used, the objects can be directly linked as they are being gen-

eralised. Otherwise, the correspondences must be inferred. In 2D, they are usually inferred using *map matching* methods, which can take into account the geometry [Veltkamp and Hagedoorn, 2001], topology and semantics of and between the objects [Devogele et al., 1996]. Devogele et al. [1996] does this in three steps: (1) manually finding correspondences between semantic classes; (2) resolving conflicts; and (3) matching objects using geometry, topology and semantics. Zhang et al. [2014] matches features by computing a compatibility coefficient, derived from the similarity in their geometry and that of their neighbours.

After the correspondences have been found, the corresponding representations of an object are linked together. These links across LODs usually take the form of common IDs at the 2D or 3D object level. However, more advanced linking structures developed for this exact purpose do exist and are often used in 2D—some of these were mentioned in §3.4: hierarchical planar subdivisions [Filho et al., 1995], multi-scale partitions [Rigaux and Scholl, 1995], nested maps [Plümer and Gröger, 1997] and topological generalised area partitioning trees [van Oosterom, 2005].

8.2 Suggested methodology and current issues

The methods used to identify correspondences between 2D objects and the data structures used to store these correspondences, mentioned in §8.1, do not readily extend to higher dimensions. While map matching methods can identify (to a limited extent) the correspondences between 2D or 3D objects, they do not take into account the lower-dimensional correspondences between the 0D–2D cells bounding them. Linking corresponding objects using only common IDs at the 2D or 3D object level is similarly problematic, as it is difficult to store complex correspondence relationships, such as an aggregation of multiple objects into one [Biljecki et al., 2014b], or those connecting the points, line segments and polygons on the boundary of corresponding 2D or 3D objects.

Based on the higher-dimensional data structures presented in Chapter 4, this section presents a sketch of a dimension-independent approach. By considering the LOD of a model as a fully independent dimension in the geometric sense [van Oosterom and Stoter, 2010; Paul et al., 2011; Stoter et al., 2012a], it is possible to store all topological relationships between any related objects across all LODs. A set of connected 2D polygons at multiple LODs are modelled as a single 3D polyhedron[125], as is shown in Figure 8.2, and a set of connected 3D polyhedra at multiple LODs are modelled as a single 4D *polychoron*. Notably, the correspondences between equivalent objects across LODs thus become geometric primitives, making it possible to perform geometric operations

125: Separate polygons might become connected in many different situations, the simplest of which is being joined into a single polygon at one or more LODs.

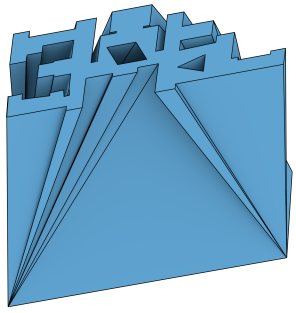


Figure 8.2: Two LODs of a building footprint are stored as a single polyhedron. Note that the correspondences between vertices, edges and faces between the LODs are clearly indicated by the vertical edges and faces, and that holes can be similarly handled with additional edges and faces.

with them (e.g. extracting an intermediate LOD for visualisation purposes) or to attach attributes to them (e.g. the meaning of these correspondences), just as is done to other geometric primitives.

In the specific case of a 4D (3D space+LOD) model, a set of 4D objects is modelled as a 4D cell complex embedded in 4D space, where there is an additional LOD axis l and a point in 4D space is defined by a tuple of coordinates (x, y, z, l). It is worth noting that this implies that the LOD axis should be properly parametrised, defining quantifiable values for every fixed LOD in a model, or alternatively a function that does so.

The 4D space thus defined is filled with a set of non-overlapping polychora, in which a 3D object (e.g. a building) at *all* of its different LODs is represented as a single 4D object. This 4D object is bounded by a set of volumes, two of them being the object at its lowest and highest LOD, and several lateral ones formed by filling the space between corresponding faces across LODs. When a 3D cross-section is extracted from it (Chapter 9), these respectively correspond to the volumes and bounding faces of an extracted 3D model.

A 4D model is constructed from a series of existing LODs of a 3D city model in three steps:

1. identifying corresponding 0D–3D cells;
2. linking them by creating 1D–4D cells connecting them;
3. using the incremental construction algorithm of Chapter 7 to build a 4D cell complex using all 0D–4D cells.

This section describes various methods to identify corresponding cells in different LODs of a 3D object (§8.2.1), and then presents four linking schemes to construct a 4D model (§8.2.2). As §8.3 demonstrates, the 4D cells created by the linking schemes are used to construct 4D cell complexes with different properties and shapes.

8.2.1 Step 1: Identifying corresponding cells in 3D models

Constructing a 4D model from a sequence of 3D models largely depends on the identification of the corresponding 0-, 1-, 2-, and 3-cells between these 3D models. The aim of this identification is to create a mapping between the 3D models that preserves the topological relationships between the elements in the models so as to create a valid 4D model.

Considering 3D models at different LODs, this identification will often result in matching cells of different dimensions, commonly with some cells in the 3D model at the highest LOD being matched to cells of lower dimension in the 3D model at the lowest LOD. Also, these correspondences will often not result in a one-to-one mapping: groups of adjoining cells in one model, most often in the one

at the highest LOD, will commonly be matched to a single cell in the other model.

The identification of matching cells should be done using a combination of the following, arguably in order of preference:

Attributes Using the semantic information stored in the cells, when it is available. For instance, matching two cells that are known to be equivalent through the use of IDs, or if knowledge is kept during the generalisation process, matching a cell with one that is known to be a simplified version of it.

Topology When there is an isomorphism between two cells (§5.1.1) that additionally preserves all the topological relationships between them (such as the facet/ridge comparisons used in Chapter 7), the isomorphism between two cells already gives a matching between them, although it might be important to check that the isomorphism is compatible with the matching of the other cells in the model and with some geometric constraints (e.g. a maximum distance between matched cells). Relevantly and as used elsewhere in this thesis, Gosselin et al. [2011] describe how to whether two cells or maps are isomorphic in any dimension. Another more complex possibility is using subgraph isomorphism on unmatched portions of a generalised map [Eppstein, 1999]. The latter option should lead to partial matches and thus better results, but this problem is known to be NP-complete and so it is unlikely to be applicable to large datasets.

Another relevant way to use topology is to use the topological relationships between cells in order to iteratively infer matchings for the cells that remain unmatched after applying some other algorithm [Hampe et al., 2003]. Such a case is explained more concretely in the example of Figure 8.5.

Geometry Using geometric computations, such as those based on computing similarity metrics, simply matching unmatched cells in one model to their nearest neighbour in another model, or attempting to minimise the Earth Mover's Distance (EMD) [Rubner et al., 1998] between them. It is however important to compute these matches using constraints that generally preserve the relative positions and topological relationships between the cells. For instance, a greedy algorithm could match cells iteratively, cascading these matches to adjacent cells (in all models) or rejecting matches that would violate a geometric or topological constraint.

Another possibility to assist when matching cells is to allow **splitting** an i-cell into multiple i-cells by adding cells of lower dimensions in a manner that does not alter the geometry of the cell. This effectively loosens the requirement that two cells be isomorphic by

modifying their topology, creating a one-to-one mapping between the two cells. For instance, a face can be split into multiple faces by adding a vertex in its interior and creating edges that link it to some of the vertices of the face[126].

126: Similar procedures can be defined for the cells of any dimension.

Holes deserve a special mention, as they can also be used to decide how to match cells. When holes (of any dimension) are present at a higher LOD but disappear at the immediate lower LOD, they can be handled appropriately by collapsing them to a nearby vertex such that they do not create any geometric intersections between the cells. Another option would be to create an additional point at an appropriate location in the lower LOD[127] where it can be collapsed. When holes are present in multiple LODs, they should be handled essentially as if they were independent cells, either matching them directly, splitting them, or collapsing parts of them to match each other.

127: If this location is in its interior, this is essentially equivalent to a 0D hole.

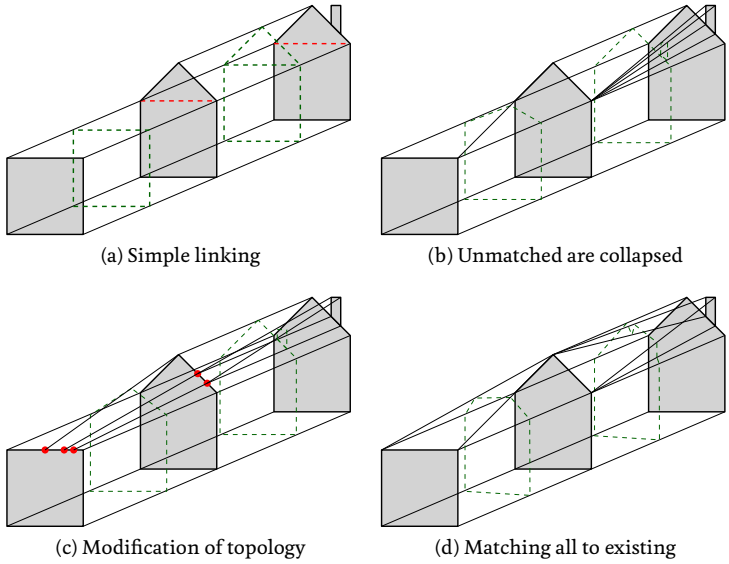
8.2.2 Step 2: Linking corresponding cells

Based on the matches that were found between cells, which mathematically define a map between the 3D cell complexes of the 3D models, they are then linked to construct a 4D cell complex. For this, it might be necessary to create or modify 0D–3D cells in the input cell complexes, as well as to create new 1D–4D cells that lie *between* the 0D–3D cell complexes. The resulting 4D cell complex is then embedded in 4D space by assigning new 4D coordinates for every point. These 4D points are simply the union of all those in the input 3D cell complexes with an appended coordinate that refers to the LOD of their originating model. Four different basic linking schemes are proposed here, which are shown in Figure 8.3.

Method 1: Simple linking of corresponding cells

Links are constructed between the corresponding cells of an object at two different LODs, and if a cell has no corresponding cell then it is ignored. While this makes it possible to easily construct a 4D cell complex in the cases where all cells in the lower LOD model have a corresponding cell in the higher LOD model, when this is not the case, the result will consist of an incomplete 4D cell complex—possibly without any 4-cells.

To ensure a complete one, cells often need to be split (e.g. those separated by the red dotted line in Figure 8.3a), which can be performed using geometric intersections. While this is possible in 3D and tools are readily available (see for instance Granados et al. [2003]; Hachenberger [2006]), the generalisation of this scheme to higher dimensions is not easy in practice since no robust intersection tools



(a) Simple linking (b) Unmatched are collapsed

(c) Modification of topology (d) Matching all to existing

Figure 8.3: The four linking schemes for three LODs of a house, here depicted in 2D. The objects that would obtained by slicing between the LODs can be seen in dashed green contours; the red dashed lines reflect the cells that need to be added and split in order to ensure a valid 3D (2D+LOD) cell complex.

in more than 3D are available. Observe that if a 4D cell complex generated using this method is sliced at an intermediate LOD, the result is exactly that of the lower LOD.

Method 2: Unmatched cells are collapsed to existing ones

No modifications are made to the 3D models, which is in practice a significant advantage since no complex geometric operations need to be performed and the size of the cell complex will be smaller than that of the one where cells are modified. Instead of geometric operations, unmatched cells in the higher LOD model are linked to nearby cells *of a possibly lower dimension* in the lower LOD model while preserving certain geometric and topological constraints (e.g. preserving adjacency and incidence between cells[128]). This implies that some cells will be collapsed (e.g. an edge can be mapped to a vertex), and the cells must be linked with care to ensure that a valid 4D cell complex is created (e.g. no two cells should intersect in 4D space).

128: This might need to be tested geometrically rather than topologically, as degenerate cells will often be created with this process.

For instance, assume that the left eave of the roof of the house in the middle LOD model in Figure 8.3b has been (arbitrarily) matched to the roof of the low LOD model, with the right eave remaining unmatched as no unmatched cells remain in the low LOD model. In this case, using the knowledge that the roof and right wall are adjacent in the low LOD model but their corresponding cells (respectively the left eave and right wall) are separated by the right eave

in the middle LOD model, the right eave can be collapsed to the common vertex lying between the two (upper right). Using such a mapping, the topological relationships between the cells will be preserved, with the exception of those involving the collapsed cells and those incident or adjacent to them.

When the mapping has faults and the resulting 4D cell complex thus has geometric problems (e.g. intersecting cells), the slicing operation might not have any geometric meaning, but the resulting higher-dimensional model can nevertheless offer other benefits, e.g. database consistency and the knowledge of some of the equivalences between cells. Note that ensuring that cells preserve their topological relationships and form a partitioning of space in 4D is challenging and not fully considered in this thesis. Finally, even if a combinatorially and geometrically valid 4D cell complex is constructed, the 3D object obtained by slicing might not be consistent with those in the real world; notice how the chimney in Figure 8.3b becomes increasingly smaller and closer to the right eave of the roof because of the way the cells have been linked.

Method 3: Modifying the topology

To ensure that there is a mapping between all the cells, it is possible to split or merge cells so that the topology (combinatorial structure) of the objects is identical. For instance, operations like removal and contraction [Damiand and Lienhardt, 2003] can be used to simplify the more complex object(s) to make them match the simpler one(s) using an iterative process. On the other direction, it is possible to first identify for every cell in the lower LOD model one or more corresponding cells in the higher LOD model, then split cells in the lower LOD model so that their topology is the same as in the higher LOD model as the one to which it must be linked.

As an example, considering the lowest LOD in Figure 8.3c, this implies first finding multiple matches for the roof cells of the lower LOD models, which then need to be split into multiple cells by the insertion of new vertices. For instance, these can be located at the closest location that lies on the matched lower LOD cell for every higher LOD cell. As this example shows, *all* the representations of an object where this approach is used will result in the same topology. This results in increased storage space and the possibility of degenerate cells e.g. multiple vertices at one location—something that however does not cause any topological problems. The geometric operations necessary to split cells can be rather intricate as well. Observe that using this method, slicing in Figure 8.3c results in a different representation of the object: one where it smoothly morphs into the one at lower LOD (e.g. the tip of the roof is slowly lowered as the LOD decreases).

Method 4: Matching all cells to existing ones

As is the case with Method 2, this method does not require modifying the topology of the objects. The main difference with it is that cells in the higher LOD model are not necessarily collapsed to a lower dimensional cell in the lower LOD model but are instead matched to one or more cells *of any dimension* while also preserving certain geometric and topological constraints. In Figure 8.3d, observe that the tip of the roof of the middle LOD model (a point) is matched to the roof of the lowest LOD (an edge since this is a 2D representation) and that the 2 edges representing the middle-LOD roof are matched to the two corners of the lower-LOD roof (points). Slicing thus creates a truncated roof having 3 edges. This can be achieved by matching all cells that have a clear correspondence first, then attempting to match groups of unmatched cells while preserving the topological relationships between cells.

For instance, in Figure 8.3d it is possible to first match the base and walls of the houses in the lower and middle LOD models, as these have exact equivalences. Then, an algorithm could match the remaining vertex and left/right edges in the middle LOD model respectively to the roof edge and left/right vertices. Observe that in this process, the tip of the roof of the middle LOD model (a point) is matched to the roof of the lowest LOD (an edge) and that the two eaves of the roof (edges) are matched to the two corners of the roof (points) in the lowest LOD. Slicing the resulting 4D cell complex creates a truncated roof having 3 edges. The matches for the chimney to other elements in Figure 8.3d are achieved by matching the chimney top to the right eave, and the remaining vertices and edges on its left and right sides respectively to the roof tip and right eave/wall vertices. The result is that while the chimney looses resemblance to reality, it slowly converges to the roof in the middle LOD model.

8.3 Use cases

This section presents practical examples that describe the matching and the linking of cells for a few simple 3D models representing the same object(s) at different LODs.

8.3.1 Using Method 1: Simple linking

Figure 8.4 shows an example where two LODs for a building are linked in such a way that only matched cells are involved. First observe that since the two objects are not isomorphic, some cells are not matched (the ones representing the roof of the higher-LOD

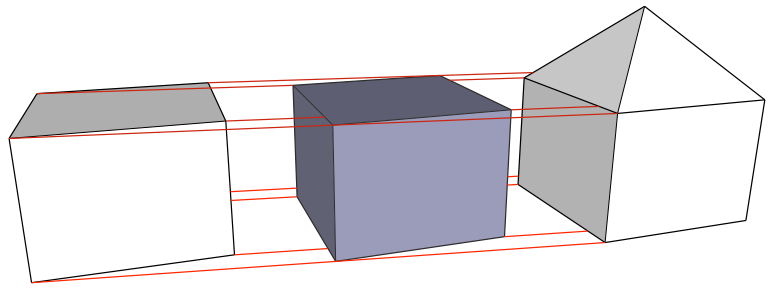


Figure 8.4: Two LODs of a house simply linked and the intermediate LOD obtained.

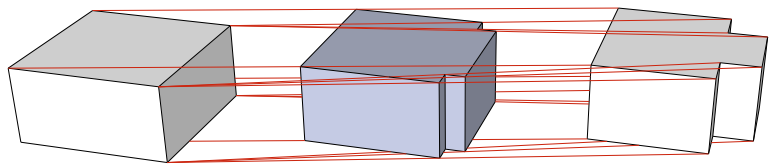


Figure 8.5: Two LODs of a house with differing geometry and topology are integrated into a 4D model by collapsing cells in the model at the highest LOD.

model). Observe also that the roof of the lower-LOD model has no match in the higher-LOD model. Thus, to construct a 4D cell complex, the flat roof geometry has to be added to the higher-LOD model[129]. Then, the corresponding cells can be linked. Although it is possible to generate this 4D model by generating the $(i+1)$-cells that connect a pair of corresponding i-cells and linking all of them together, it is probably easier to extrude one cell complex along the range between the two LODs using the method described in Chapter 6. This method already generates the proper combinatorial structure of the 4D model, and the final cell complex can be obtained by simply moving the vertices of the face representing the model at the other LOD so as to match the geometry of the other model at its LOD. Moreover, when a linear cell complex is used and thus only the vertices are storing the geometry of the model, it is only necessary to move some of the vertices: those in the lowest LOD without a corresponding vertex at the same location in the highest LOD.

129: Otherwise, there could not be a 4-cell in the 4D model as the top of the house would be open and no subset of $\mathbb{R}^4$ would be enclosed in the model.

8.3.2 Using Method 2: Collapsing

Figure 8.5 shows an example with a 3D model at two LODs with differing geometry and topology. The 4D model has been obtained by first matching the 2-cells with known correspondences (the left, right, front and back large faces) and inferring that the other faces in the model at the highest LOD (right) should be collapsed based on their adjacency relationships with the matched faces. For example, since the front and right faces are adjacent in the lowest LOD but not in the highest LOD, the two faces between them[130] should be collapsed to their common boundary (i.e. their intersection: the edge between them). This example also shows that the topological

130: Combinatorially, this involves a search for a path between the front and right faces in the map of the model at the highest LOD. Such a search would be limited to the nodes representing the aforementioned faces and those representing faces that are *not* present in the model at the lowest LOD.

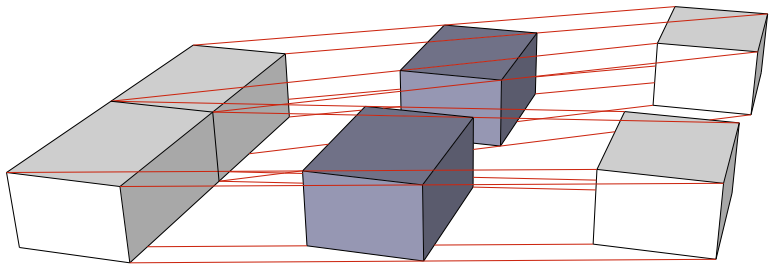


Figure 8.6: Two LODs of two houses being aggregated are integrated into a 4D model by modifying the topology of the model at the lowest LOD so as to match the topology of the model at the highest LOD.

relationships between the cells are nevertheless preserved with the exception of those that involved collapsed cells. The new topological relationships do however connect cells around the former collapsed cells. Note that the 3D model resulting from slicing the 4D model created in this way at an intermediate LOD (middle) will be isomorphic to the model at the highest LOD.

8.3.3 Using Method 3: Modifying the topology

Figure 8.6 shows an example of two 3D models being aggregated. In order to create a 4D model from this situation, the topology of the simpler of the two models is modified, splitting the single volume into equal two adjacent ones, effectively resulting in a cell complex that also has four more vertices, four more edges, and four of its faces split into two. Note that the two models are however not isomorphic since the common face of the two houses in the lowest LOD becomes two disconnected faces in the model at the highest LOD, but that if this topological relationship is disregarded, the two models can be correctly matched independently.

8.3.4 Combination of Methods 2 and 3

Figure 8.7 shows a more complex example with three LODs which are linked using a combination of schemes: collapsing and modifying the topology of one of the models. Most of the cells in the highest LOD can be directly matched to cells in the middle LOD, with the exception of those that are part of the chimney. As these comprise a small object, these are simply collapsed to a single point in the middle LOD. Matching the roof cells in the lowest and middle LODs is however more complex since collapsing it to a point would ignore its adjacency with the body of the house and therefore not preserve its topology. The best solution is therefore to modify the topology of the lowest LOD in order to split the top face of the cubic house (which can be inferred to be a roof based on its attributes) into 4 faces, making the model isomorphic to the middle LOD.

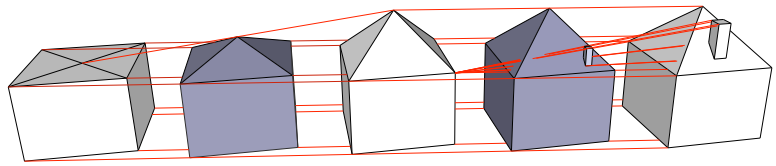


Figure 8.7: Three LODs of a 3D model of a house are integrated into a 4D model by modifying the topology of the model at the lowest LOD and collapsing a part of the model in the highest LOD.

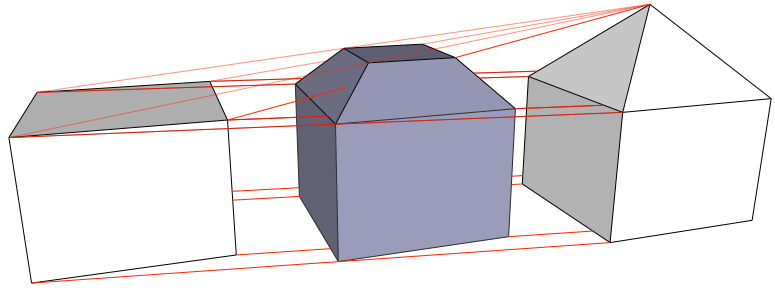


Figure 8.8: Two LODs of a 3D model of a house (left and right) are linked despite not being isomorphic, with an intermediate LOD that shows the result of slicing the construction at an intermediate LOD (centre).

8.3.5 Using Method 4: matching to existing cells

Figure 8.8 shows two of the LODs of the previous example, but matches the cells of the roof of the house to existing cells rather than modifying their topology. After attempting to match corresponding cells, the top face in the lowest LOD and the top four faces in the highest LOD remain unmatched. If each top *face* in the highest LOD is collapsed to the closest top *edge* in the lowest LOD (i.e. the edge that forms the bottom of the triangular face in the highest LOD) and the top *vertex* in the highest LOD (which lies between the faces) is linked to the top *face* in the lowest LOD, a four-sided pyramid is generated. Slices from it are shown as four trapezoidal faces in the sliced intermediate LOD. Then, if the top face in the lowest LOD is collapsed to the top vertex of the highest LOD, another four-sided pyramid is generated. A slice from this one is shown as a square face at the top of the sliced intermediate LOD.

This particular mapping is notable because it correctly preserves all topological relationships between the cells, does not create additional cells at either LOD, and shows that cells are not necessarily only collapsed from higher LODs to lower LODs. Intuitively, the result of this mapping is a set of cells that bound the model along the LOD dimension[131], so that an i-cell and a j-cell that are matched result in a k-cell lying between them, where $k = \max(i, j)+1$. Concretely, if for instance a 0-cell (the tip of the roof) is matched to the flat roof (a 2-cell), then the resulting links will create a tetrahedron (a 3-cell). Admittedly, the rules needed to generate such a mapping can be quite complex, but the cell complex generated is identical in size to the equivalent model according to the scheme in Figure 8.3b.

131: Here is a 3D analogue: considering two arbitrary but parallel 'top' and 'bottom' polygons, an equivalent mapping would create a set of edges and faces connecting them so as to form a closed polyhedron.

8.4 A concrete example

In order to show how the linking methods presented in this chapter work in practice, this section shows an implementation of the model from Figure 8.8. It uses CGAL Linear Cell Complexes and the incremental constructor operator described in Chapter 7. This model was chosen as it uses most of the linking methods discussed in the §8.2.2: the body of the house in both LODs is directly linked (Method 1), the top face of the house in the lower LOD is collapsed to the tip of the roof in the higher LOD (Method 2), and the roof vertices/edges in the lower LOD are connected to existing roof edges/vertices in the higher LOD (Method 4).

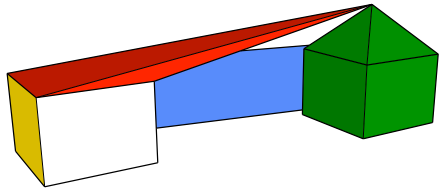
First of all, the 17 vertices of the two 3D models are created as 4D points of the form (x, y, z, l). Afterwards, these are first used to define the 35 faces of the model, the faces are used to define the 12 volumes, and the volumes to define the single 4-cell. Notice that these cells not only include faces and volumes within each of the two volumes of the input 3D models, but also include some faces and volumes that lie *between* the two, i.e. having bounding vertices, edges and faces from both input 3D models. Excerpts of the code to generate the 4D cell complex are shown in Figure 8.9.

The resulting 4D model was then validated by checking the properties of a valid combinatorial map (§5.1.1). In short, these tests involved checking whether the darts (combinatorial simplices) in the map formed correct involutions or partial permutations and whether any darts remained free after the operations. Individual parts of the model (the triangular or square faces and the parallelepiped- or pyramid-shaped volumes) were also validated by verifying that they were isomorphic to similar objects that were known to be valid [Gosselin et al., 2011].

8.5 Conclusions

While integrating different LODs of the same 3D object into a single 4D model is generally considered as complex, the linking schemes proposed in this chapter show that it is possible to define high-level methods to connect different 3D models in a relatively simple manner. The linking schemes operate within a framework of three steps: identifying corresponding elements in different LODs, deciding how these should be linked, and finally linking relevant 3-cells into 4-cells using the incremental construction algorithm described in Chapter 7.

Using a 4D representation means that it is possible to store not only the standard topological relationships (e.g. incidence and adjacency) between objects in one LOD, but also all the correspon-



(a)

```
// 2: Left of first house (yellow)
faceLists.push_back(...);
faceLists.back().push_back(vertices[0]);
faceLists.back().push_back(vertices[2]);
faceLists.back().push_back(vertices[6]);
faceLists.back().push_back(vertices[4]);

//21: Back right vertical edge (blue)
faceLists.push_back(...);
faceLists.back().push_back(vertices[3]);
faceLists.back().push_back(vertices[7]);
faceLists.back().push_back(vertices[15]);
faceLists.back().push_back(vertices[11]);

for (int i = 0; i < 35; ++i)
    faces.push_back(
      builder.get_facet_from_vertices(
        faceLists[i].begin(),
        faceLists[i].end(), false).first);

// 1: Right house (green)
volumeLists.push_back(...);
volumeLists.back().push_back(faces[6]);
volumeLists.back().push_back(faces[7]);
volumeLists.back().push_back(faces[8]);
volumeLists.back().push_back(faces[9]);
volumeLists.back().push_back(faces[10]);
volumeLists.back().push_back(faces[11]);
volumeLists.back().push_back(faces[12]);
volumeLists.back().push_back(faces[13]);
volumeLists.back().push_back(faces[14]);

// 11: Roof from left to right tip (red)
volumeLists.push_back(...);
volumeLists.back().push_back(faces[31]);
volumeLists.back().push_back(faces[32]);
volumeLists.back().push_back(faces[33]);
volumeLists.back().push_back(faces[34]);
volumeLists.back().push_back(faces[5]);

for (int i = 0; i < 12; ++i)
    volumes.push_back(builder.get_cell<3>(
      volumeLists[i].begin(),
      volumeLists[i].end()).first);

LCC::Dart_handle house4d = builder.get_cell<4>(
  volumes.begin(), volumes.end()).first;
```

(b) Constructing other cells

```
float point_coordinates[][4] = {
    // Left house
    {0, 0, 0, 0}, // 0
    {1, 0, 0, 0}, // 1
    {0, 1, 0, 0}, // 2
    {1, 1, 0, 0}, // 3
    {0, 0, 1, 0}, // 4
    {1, 0, 1, 0}, // 5
    {0, 1, 1, 0}, // 6
    {1, 1, 1, 0}, // 7

    // Right house
    {0, 0, 0, 1}, // 8
    {1, 0, 0, 1}, // 9
    {0, 1, 0, 1}, // 10
    {1, 1, 0, 1}, // 11
    {0, 0, 1, 1}, // 12
    {1, 0, 1, 1}, // 13
    {0, 1, 1, 1}, // 14
    {1, 1, 1, 1}, // 15
    {.5, .5, 1.5, 1} // 16
};

for (int i = 0; i < 17; ++i) {
    points.push_back(Point(4,
        point_coordinates[i],
        point_coordinates[i]+4));
    vertices.push_back(
        builder.get_vertex(points[i]));
}
```

(c) Coordinates of the 17 vertices

Figure 8.9: Code excerpts that show how vertices are created based on 4D points, faces as cycles of vertices, volumes as sets of faces, and 4-cells as sets of 3-cells. The colours referred to in (b) correspond to the highlighted faces and volumes in (a).

dences between equivalent objects of any dimension across LODs, even when corresponding objects are of different dimensions or the correspondences between them are not one-to-one. As these correspondences are modelled as geometric primitives, it is easy to perform geometric operations with them (e.g. extracting an intermediate LOD for visualisation purposes) or to attach attributes to them (e.g. the meaning of these correspondences), just as is done to usual geometric primitives.

These topological relationships and correspondences can then be used for multiple applications, such as updating and maintaining series of 3D models at different LODs, or testing the consistency of multi-LOD models (e.g. by using similar validity checks as those in Gröger and Plümer [2011a]).

The different linking schemes presented in §8.2.2 yield 4D models having different properties, such as objects that suddenly appear and disappear, gradually change in size or morph into different objects along the fourth dimension. These different types of 4D models can then be useful for different applications.

The linking schemes described in this chapter are meant for the multi-LOD 3D to 4D case. However, they do have a generic formulation, so they can also be applied to other non-spatial characteristics such as time, enabling them to be used in new applications, such as identifying the motion or change of objects through time. In addition, these schemes are fully dimension-independent, so it is conceivable that they can be applied to linking multiple higher-dimensional models, such as a series of 4D models that reflect a building at different periods of time and at different levels of detail.

While the transitions between LODs as shown in this chapter are exclusively linear, it is worth noting that this does not necessarily have to be the case. Non-linear transformations can be also be defined (such as by ensuring that a series of LODs forms a C^1 or C^2 continuous shape) and their corresponding geometries can be either stored in more complex non-linear embeddings of a combinatorial map or discretised as a series of small linear cells that approximate such a shape up to a given ϵ threshold.

Finally, the clearest next step to be followed here is to realise the method sketched in this chapter by analysing the problem in more detail and implementing the required low-level algorithms. While the linking problem has been described here as a monolithic solution, it is more likely that the different methods presented will result in completely different implementations.

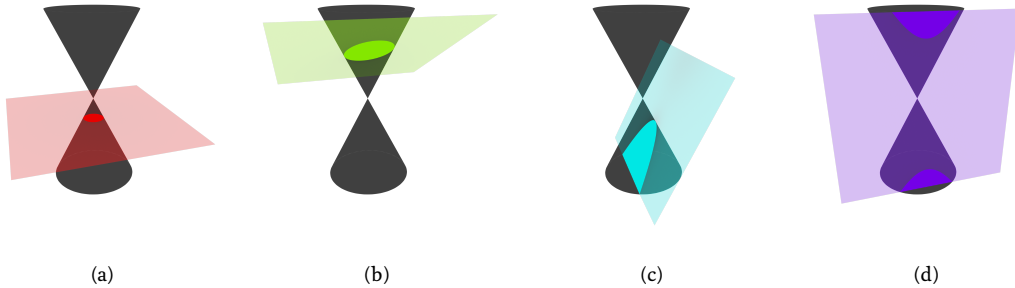
9 Extracting information from higher-dimensional models

The previous chapters have discussed various aspects of higher-dimensional representations and operations. As powerful as they can be, these representations and operations are generally incompatible with current 2D/3D software and are also hard to imagine and visualise. In order for them to be used in practice, it is therefore important to have methods to extract meaningful 2D and 3D subsets from a higher-dimensional model.

While fully developing methods and algorithms to do so is outside the scope of this thesis, steps towards formalising this problem and looking for potential solutions were made within the context of this thesis, and they are thus documented here. The chapter starts by giving some of the background behind the problem in §9.1, explaining at a high level the process to extract lower-dimensional information from a higher-dimensional model. Using a simple camera analogy common in computer graphics, §9.2 explains how a 3D viewpoint is placed in a 3D scene through the application of different transformations, capturing a 2D view of it by applying a given projection. §9.3 uses this analogy to give an intuitive description of how this process extends to higher dimensions, later formulating a simple dimension-independent generalisation of the orthographic and perspective projection methods and explaining other projection possibilities. Finally, §9.4 concludes with some ideas on the value of these methods, what pieces are still missing and how the overall process could be implemented.

9.1 Background

The process to obtain a lower-dimensional subset of a higher-dimensional dataset can be regarded as a function that maps a subset of $\mathbb{R}^n$ to a subset of $\mathbb{R}^m$, $m < n$, which is obtained by cutting through the dataset in a geometrically meaningful way. For instance, it is possible to cut orthogonally to an axis for a snapshot at one point in space/time, or obliquely for a subset that combines different parametrised characteristics, such as the evolution of different parts of an area in time.



(a) (b) (c) (d)

Figure 9.1: Four types of conic sections: (a) circle, (b) ellipse, (c) parabola and (d) hyperbola. These are obtained by the Boolean set intersection of a pair of cones with a plane.

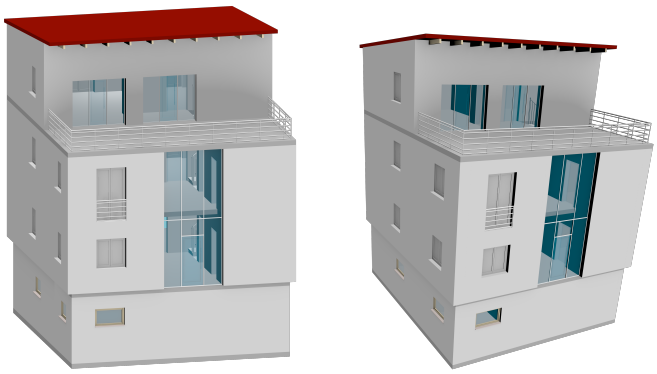
The process to extract these slices can be conceived as consisting of two broad steps: (i) optionally selecting a subset of the objects in the scene, and (ii) applying a transformation that projects this subset to a lower dimension. In computer graphics, these steps would normally be followed by *rendering* the selected and projected objects, obtaining a raster image as an end-product.

Selecting a subset of a dataset is useful to bring certain parts of the dataset into view. For instance, this is commonly applied to parts of a dataset that are normally not visible, such as the interior of an object. It is also widely used to reduce the difficulty of the overall problem and avoiding the need to make unnecessary computations, such as projecting and rendering parts of the dataset that will be clearly out of view.

This selection process can be relatively simple, such as using all objects that are within a certain bounding box, that are within a certain distance of a point, or being of a given dimension. For instance, 3D rendering programs and 3D games will usually only render the 2D faces of the objects lying (approximately) within a region (e.g. frustum), not their (filled-in) volumes or any 2D faces that are clearly out of view.

However, more complex selection processes are useful in many instances, often requiring specialised data structures and geometric algorithms. In particular, a great number of visibility determination algorithms have been developed [Hughes et al., 2014, Ch. 36], which solve various approximations of the surfaces that are visible in a scene. Another example is a selection process is the computation of cross-sections of volumetric 3D objects, which requires the computation of a Boolean set intersection of a point set with a plane, such as the typical *conic sections* shown in Figure 9.1.

Projecting the selected subset yields a given view of the scene and reduces the actual dimension of the objects being represented. Many



(a) (b)

Figure 9.2: The (a) orthographic projection projects objects orthogonally to the projection plane. The (b) perspective projection projects objects in manner such that those that are nearby appear comparatively larger than farther away. Note how parallel lines (e.g. wall edges) remain parallel in the former but not in the latter.

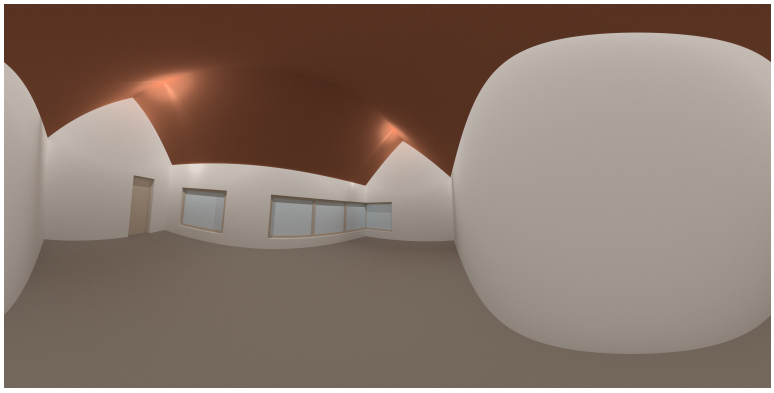


Figure 9.3: The Equirectangular projection directly maps angles to coordinates. A 360° view is here mapped into a rectangular 180°×360° image. Rendered from a viewpoint inside the IfcOpenHouse dataset[133].

types of projections for this purpose can be defined, usually in the form of a transformation that is applied to the objects. For example, orthographic and perspective projections (Figure 9.2) can directly map an nD scene to an mD subspace. However, more complex schemes are also possible, such as projecting first inwards/outwards to an m-sphere (for a hypothetical $(m+1)$D Nef polyhedra implementation), then to an mD subspace (e.g. using an equiangular projection as shown in Figure 9.3).

Many of these projections can be done with techniques that are closely related to computer graphics, which has been dealing with the case of converting a 3D scene into a 2D image for decades, and less frequently covering higher-dimensional cases as well. In this sense, it is useful to consider the same camera analogy that is often used in computer graphics, where a movable camera[134] captures a scene as seen from a particular viewpoint. In a higher-dimensional setting, this analogy would consist of an mD camera capturing an (mD) view of an nD scene. While this is somewhat more difficult to imagine, this analogy emphasises three important aspects that re-

133: http://blog.ifcopenshell.org/2012/11/say-hi-to-ifcopenhouse.html

134: Even if the movable camera is implemented by moving the dataset instead.

main true in any dimension:

- Moving the camera around a scene and orienting it makes it possible to capture the data from any given viewpoint. Such a viewpoint can be parametrised—and thus easily defined and stored—by a set of values containing its *location* and *orientation*.
- It is possible to obtain different views—intuitively corresponding to camera lenses—through various projections. These can also have their own customisable parameters, such as their field of view.
- Despite the fact that the dimension of the data is reduced (from n to m), characteristics that would seem to have been lost in this process can be preserved in the form of attributes and used for further computations. For instance, in computer graphics, the distance from an object to the camera (i.e. the depth) is regularly used for computations of 3D-to-2D projections, either as a simple computation where the objects that are nearer (along a line of sight) occlude those that are farther away, or for more complex effects, such as transparency and reflectivity. In cartography, hypsometric tints, shaded relief, contour lines are all frequently used to encode the height information in a 2D map (Figure 9.4).

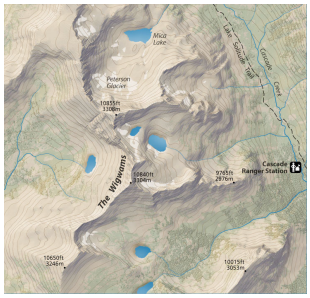


Figure 9.4: An excerpt of the map for the Grand Teton National Park. From Patterson [2002].

Continuing with the camera analogy, it is useful to consider the nD scene as consisting of a set of 0D–nD objects. In the context of this thesis, these would be represented as an n-dimensional simplicial complex or cell complex with linear geometries, such that every vertex in the map is embedded in a location in $\mathbb{R}^n$. The simplices/cells of the complex can be translated, rotated or scaled, if necessary, using the operations described in §5.2. They can then be individually projected by the camera, possibly taking into account occlusion or other visual effects, in order to obtain the mD scene.

This projection is simplest when two conditions are met: (i) the objects have linear geometries that are stored as coordinates attached to their vertices (as in this thesis), and (ii) the projection transformation preserves the linearity of the objects. In this case, it is only necessary to apply the transformation to every vertex individually.

However, even when one or both of these conditions are not met, it is possible to obtain a good approximation by first subdividing a simplex/cell into small simplices/cells (up to an arbitrary ε threshold), assuming that linear geometries will remain approximately linear and projecting only the vertices of the subdivided complex. This subdivision and approximation method is used for the cover of this thesis and the example in Figure 9.10 on page 161.

The following sections will therefore assume that the data to be projected consists only of a set of k points with n coordinates, which

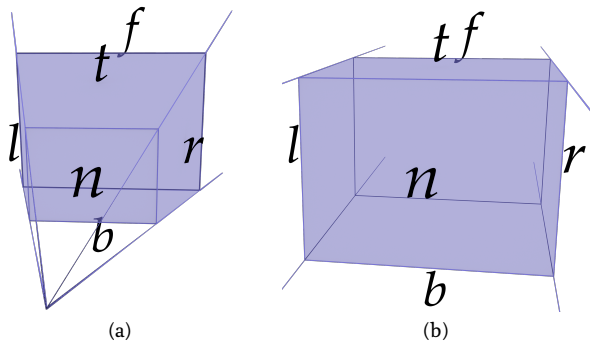


Figure 9.5: The left (l), right (r), bottom (b), top (t), near (n) and far (f) planes that form (a) a perspective projection's frustum and (b) an orthographic projection's box.

is stored as a $k \times n$ matrix. These will be given new coordinates in $\mathbb{R}^m$ by applying one or more operations expressed in terms of linear algebra.

9.2 3D to 2D projections

Projecting a set of 3D objects into a 2D view is one of the most common tasks in computer graphics. As such, most computer graphics books derive their own versions of the transformations required to apply different types of projections, most of which are interchangeable but are often parametrised in different ways.

From a practical perspective, the OpenGL Programming Guide (better known as the Red Book) [Shreiner et al., 2013] provides very intuitive forms of 3D-to-2D perspective and orthographic projections, which are based on defining a *viewing frustum* or *box* where the objects to be viewed should be located/placed. This can be achieved by translating, rotating and scaling the objects as described in §5.2.

Assuming a setup as shown in Figure 9.5, where a camera is placed at $z = 0$ and pointing towards higher values on the z axis, with parameters defining the x coordinates for the left (l) and right (r), y coordinates for the bottom (b) and top (t), and z coordinates for the near (n) and far (f) planes, Shreiner et al. [2013, Ch. 5] define a transformation that applies a perspective projection P_p and an orthographic projection P_o as:

$$P_p = \begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{2(l+r)}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{2(t+b)}{t-b} & 0 \\ 0 & 0 & \frac{f+n}{n-f} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix} \qquad P_o = \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & \frac{l+r}{l-r} \\ 0 & \frac{2}{t-b} & 0 & \frac{b+t}{b-t} \\ 0 & 0 & \frac{-2}{f-n} & \frac{n+f}{n-f} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

However, as this thesis attempts to obtain a dimension-independent formulation, it is more interesting to consider a somewhat more complex method that can be extended more readily to higher dimensions. Foley and Nielson [1992] and Hughes et al. [2014, Ch. 13], among others, start from the definition of a triplet of vectors $\hat{x}$, $\hat{y}$ and $\hat{z}$, which are computed from a point *from* where the camera is located, a point *to* that the camera directly points towards, and an arbitrary vector $\vec{up}$ pointing approximately upwards[135]. The latter vector is necessary because there is still one degree of rotational freedom even as the camera is pointing towards *to*. The unit vectors $\hat{x}$, $\hat{y}$ and $\hat{z}$ correspond to the x, y and z axes from the camera's perspective and are defined as:

135: Strictly, it can be defined pointing toward any direction except along the z axis.

$$\hat{x} = \frac{\hat{z} \times \vec{up}}{\left\| \hat{z} \times \vec{up} \right\|} \qquad \hat{y} = \hat{x} \times \hat{z} \qquad \hat{z} = \frac{to - from}{\left\| to - from \right\|}$$

Note that if $\vec{up}$ is orthogonal to $\hat{z}$, $\hat{y}$ defines the up direction as a normalised $\vec{up}$. If they are not orthogonal, $\hat{y}$ defines the up direction differently from $\vec{up}$.

Based on the three vectors $\hat{x}$, $\hat{y}$ and $\hat{z}$, a $n \times 3$ matrix of n point coordinates in terms of the scene P_{world} (world coordinates) can be converted into a matrix of points coordinates in terms of the camera (eye coordinates). This would be computed as:

$$P_{\text{eye}} = \left[P_{\text{world}} - from \right] \left[\hat{x} \quad \hat{y} \quad \hat{z} \right]$$

In a 3D to 2D **orthographic projection**, the x and y coordinates of this matrix, here denoted as x_{eye} and y_{eye}, are directly usable as the coordinates of the points in $\mathbb{R}^2$. As shown in Figure 9.6, the z_{eye} coordinates represent the distance between the point and the *projection plane*, i.e. the camera, and thus define the *depth*. Points with positive z_{eye} values close to zero are thus close to the camera while points with larger z_{eye} values are farther away. Note that negative z_{eye} values mean that the point lies behind the camera, and so these would be usually omitted.

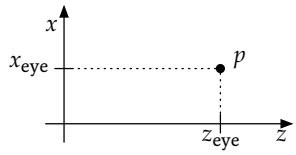


Figure 9.6: The geometry of an orthographic projection for a point p. The situation of the y axis is identical to that of the x axis.

For visualisation purposes, it is often convenient to normalise the coordinates so that the range of the x and y coordinates extend as much as possible along a given frame, resulting in objects that are not too small or too large. This can be accomplished by finding the point that is farthest along the $\hat{x}$ and $\hat{y}$ compared to the desired aspect ratio of the frame in which they should fit. As Foley and Nielson [1992] point out, in the case of the square $[-1, 1] \times [-1, 1]$, which is commonly used, point coordinates can be normalised by dividing them by the longest distance of any point to the *to* point.

In a 3D to 2D **perspective projection**, objects are projected towards a point (the camera viewpoint's coordinates) rather than a plane. This

results in new x_{pers} and y_{pers} coordinates that are scaled inwards in inverse proportion to the depth, which is here defined as the distance between a point and the camera viewpoint's coordinates. Intuitively, this means that if an object is n times farther than another identical object, it is depicted n times smaller, or $\frac{1}{n}$ of its size.

As shown in Figure 9.7, this distance can be easily computed as the hypotenuse of a right-angled triangle, where the adjacent cathetus of an angle ϑ is given by the (orthogonal) distance of the point to the projection plane (z_{eye}), the opposite cathetus is given by the point's world coordinates (x_{eye} or y_{eye}), and ϑ is the viewing angle between $\hat{z}$ (which lies on a line that passes through *from* and *to*) and the line between the *to* point and the current point. New x_{pers} and y_{pers} coordinates are thus computed using this angle as follows:

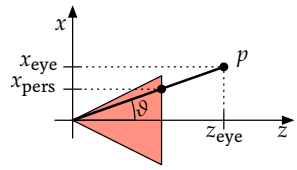


Figure 9.7: The geometry of a perspective projection for a point p. The situation of the y axis is identical to that of the x axis.

$$x_{\text{pers}} = \frac{x_{\text{eye}}}{z_{\text{eye}} \tan(\vartheta/2)} \qquad y_{\text{pers}} = \frac{y_{\text{eye}}}{z_{\text{eye}} \tan(\vartheta/2)}$$

There are many other types of projections that can be defined and can be interesting in higher dimensions, such as the equirectangular projection shown in Figure 9.3 where evenly spaced angles along a *rotation plane* (§5.2) can be directly converted into evenly spaced coordinates. However, they will not be discussed here. See Salomon [2011, Chs. 5–7] for a good reference on how to apply many different types of linear and non-linear projections.

9.3 Higher-dimensional projections

Based on the 3D to 2D projection methods described by Foley and Nielson [1992], Hollasch [1991] extends them to perform 4D to 3D orthographic and perspective projections. This section further extends these methods to describe the n-dimensional to $(n-1)$-dimensional case, changing some aspects to better explain the geometric meaning of each vector.

First, starting from a point $from \in \mathbb{R}^n$ where the camera is located, a point $to \in \mathbb{R}^n$ that the camera directly points towards, and a set of $n-2$ vectors $\vec{v}_1, \dots, \vec{v}_{n-2}$ in $\mathbb{R}^n$ that are all linearly independent from each other and from the vector $to - from$, it is possible to define a set of unit vectors $\hat{x}_0, \dots, \hat{x}_{n-1}$ that define the axes $x_0, \dots, x_{n-1}$ of a coordinate system in $\mathbb{R}^n$ as:

$$\hat{x}_0 = \frac{\vec{v}_1 \times \cdots \times \vec{v}_{n-2} \times \hat{x}_{n-1}}{\left\|\vec{v}_1 \times \cdots \times \vec{v}_{n-2} \times \hat{x}_{n-1}\right\|}$$

$$\hat{x}_i = \frac{\vec{v}_{i+1} \times \cdots \times \vec{v}_{n-2} \times \hat{x}_{n-1} \times \hat{x}_0 \times \cdots \times \hat{x}_{i-1}}{\left\|\vec{v}_{i+1} \times \cdots \times \vec{v}_{n-2} \times \hat{x}_{n-1} \times \hat{x}_0 \times \cdots \times \hat{x}_{i-1}\right\|}, \quad \text{for } 0 < i < n-2$$

$$\hat{x}_{n-2} = \hat{x}_{n-1} \times \hat{x}_0 \times \cdots \times \hat{x}_{n-2}$$

$$\hat{x}_{n-1} = \frac{to - from}{\left\|to - from\right\|}$$

The vector $\hat{x}_{n-1}$ is the first that needs to be computed and is oriented along the line from the camera ($from$) and the point that it is oriented towards (to). Afterwards, the vectors are computed in order from $\hat{x}_0$ to $\hat{x}_{n-2}$ as normalised n-dimensional cross products of $n-1$ vectors. These contain a mixture of the input vectors $\vec{v}_1, \ldots, \vec{v}_{n-2}$ and the computed unit vectors $\hat{x}_0, \ldots, \hat{x}_{n-1}$, starting from $n-2$ input vectors and one unit vector for $\hat{x}_0$, and removing one input vector and adding the previously computed unit vector for the next $\hat{x}_i$ vector. Note that if $\vec{v}_1, \ldots, \vec{v}_{n-2}$ and $\hat{x}_{n-1}$ are all orthogonal to each other, $\forall 0 < i < n-1$, $\hat{x}_i$ is simply a normalised $\vec{v}_i$.

Similarly to the 3D-to-2D case, the vectors $\hat{x}_0, \ldots, \hat{x}_{n-1}$ can be used to transform an $m \times n$ matrix of m nD points in world coordinates P into an $m \times n$ matrix of m nD points in eye coordinates E by applying the following transformation:

$$E = \left[P - from\right]\left[\hat{x}_0 \quad \cdots \quad \hat{x}_{n-1}\right]$$

As before, if E has rows of the form $[e_0 \cdots e_{n-1}]$ representing points, $e_0, \ldots, e_{n-2}$ are directly usable as the coordinates in $\mathbb{R}^{n-1}$ of the projected point in an n-dimensional to $(n-1)$-dimensional **orthographic projection**, while e_{n-1} represents the distance between the point and the projection $(n-1)$-dimensional subspace, which can be used for visual cues[136]. The coordinates along $e_0, \ldots, e_{n-2}$ could be made to fit within a certain bounding box by computing their extent along each axis, then scaling appropriately using the extent that is largest in proportion to the extent of the bounding box's corresponding axis.

136: Visual cues can still be useful in higher dimensions. See http://eusebeia.dyndns.org/4d/vis/08-hsr.

For an n-dimensional to $(n-1)$-dimensional **perspective projection**, it is only necessary to compute the distance between a point and the camera as the hypotenuse of a right-angled triangle by taking into account the viewing angle ϑ between $\hat{x}_{n-1}$ and the line between the to point and every point. This situation is the same as that of the 3D-to-2D case shown previously in Figure 9.7 and results in new $e'_0, \ldots, e'_{n-2}$ coordinates that are shifted inwards. The coordinates are computed as:

$$e'_i = \frac{e_i}{e_{n-1} \tan \vartheta/2}, \quad \text{for } 0 \leq i \leq n-2$$

The $(n-1)$-dimensional coordinates generated by this process can then be recursively projected down to progressively lower dimensions using this method. The objects represented by these coordinates can also be discretised into images of any dimension. For instance, Hanson [1994] describes how to perform many of the operations that would be required, such as dimension-independent clipping tests and ray-tracing methods.

In addition to orthographic and perspective projections, there are other interesting projections that can be applied in higher dimensions. In fact, since we lack an intuitive understanding of higher dimensions, there is little benefit in using projections that work in similar ways as the ways in which we mentally process 3D information. For instance, Jenn 3D[137] visualises polyhedra and polychora by first projecting them inwards/outwards to the volume of a 3-sphere[138], resulting in curved edges, faces and volumes. In a dimension-independent setting, this projection can be easily done by considering the angles $\vartheta_0, \ldots, \vartheta_{n-2}$ in an *n-dimensional spherical coordinate system*. Steeb [2011, §12.2] formulates such a system as:

137: http://www.math.cmu.edu/~fho/jenn/

138: Intuitively, an unbounded volume that wraps around itself, much like a 2-sphere can be seen as an unbounded surface that wraps around itself.

$$\begin{aligned} r &= \sqrt{x_0^2 + \cdots + x_{n-1}^2} \\ \vartheta_i &= \cos^{-1}\left(\frac{x_i}{\sqrt{r^2 - \sum_{j=0}^{i-1} x_j^2}}\right), \quad \text{for } 0 \leq i < n-2 \\ \vartheta_{n-2} &= \tan^{-1}\left(\frac{x_{n-1}}{x_{n-2}}\right) \end{aligned}$$

It is worth to note that the radius r of such a coordinate system is a measure of the depth with respect to the projection $(n-1)$-sphere S^{n-1} and can be used similarly to the previous projection examples. The points can then be converted back into points on the surface of an $(n-1)$-sphere of radius 1 by making $r = 1$ and applying the inverse transformation. Steeb [2011, §12.2] formulates it as:

$$\begin{aligned} x_i &= r \cos \vartheta_i \prod_{j=0}^{i-1} \sin \vartheta_j, \quad \text{for } 0 \leq i < n-2 \\ x_{n-1} &= r \prod_{j=0}^{n-2} \sin \vartheta_j \end{aligned}$$

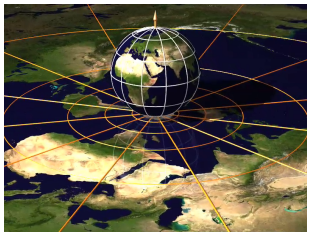


Figure 9.8: A 2-sphere to $\mathbb{R}^2$ stereographic projection can map the surface of the Earth to the plane. Here, every point p on the sphere is projected to the intersection of the plane with a line passing through the North pole and p. From Leys et al. [2008].

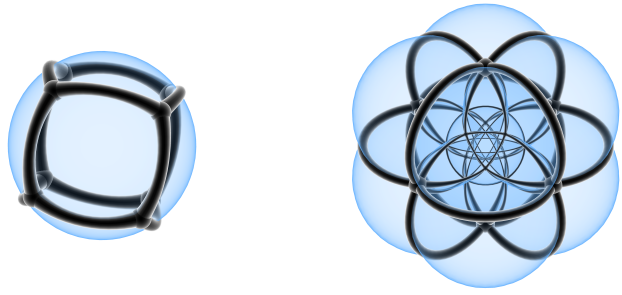
The projections on the 3-sphere used by Jenn 3D are then stereographically projected to $\mathbb{R}^3$, then with a perspective projection (Figure 9.8) down to 2D. The final result of this 4D-to-2D projection in multiple stages is shown in Figure 9.9. A stereographic projection is also easy to apply in higher dimensions, mapping an $(n+1)$-dimensional point $x = (x_0, \dots, x_n)$ on an n-sphere S^n to an n-dimensional point $x' = (x_0, \dots, x_{n-1})$ in the n-dimensional Euclidean space $\mathbb{R}^n$. Chisholm [2000] formulates this projection as:

$$x'_i = \frac{x_i}{x_n - 1}, \quad \text{for } 0 \le i < n$$

Figure 9.10 shows the application of this method to a 4D model of a house. For this, all the cells of the model were manually defined similar to how it was done in §8.4 using their boundary cells and 0-embeddings set in $\mathbb{R}^4$. The 1- and 2-cells were first refined into small line segments and triangles, the 0-, 1- and 2-cells were then projected inwards/outwards to the volume of a 3-sphere (S^3), and then stereographically projected to $\mathbb{R}^3$. The refined 2-cells were exported as to an .obj file with materials that reflect which 3- and 4-cell they belong to. Icospheres with a set radius were generated around every 0-cell and approximations of cylinders were generated around every refined 1-cell. All of these geometries were then imported into Blender 3D and rendered using a perspective projection down to 2D.

9.4 Conclusions and possibilities

Using a combination of data *selection* and *projections*, it is possible to extract meaningful lower-dimensional information from higher-dimensional datasets, making it possible for these datasets to be used in standard software and visualised. Selecting an arbitrary subset of an $\mathbb{R}^n$ point set is very challenging, as this process might require the computation of Boolean set intersection operations in



(a) (b)

Figure 9.9: A polyhedron and a polychoron in Jenn 3D: (a) a cube and (b) a 24-cell.

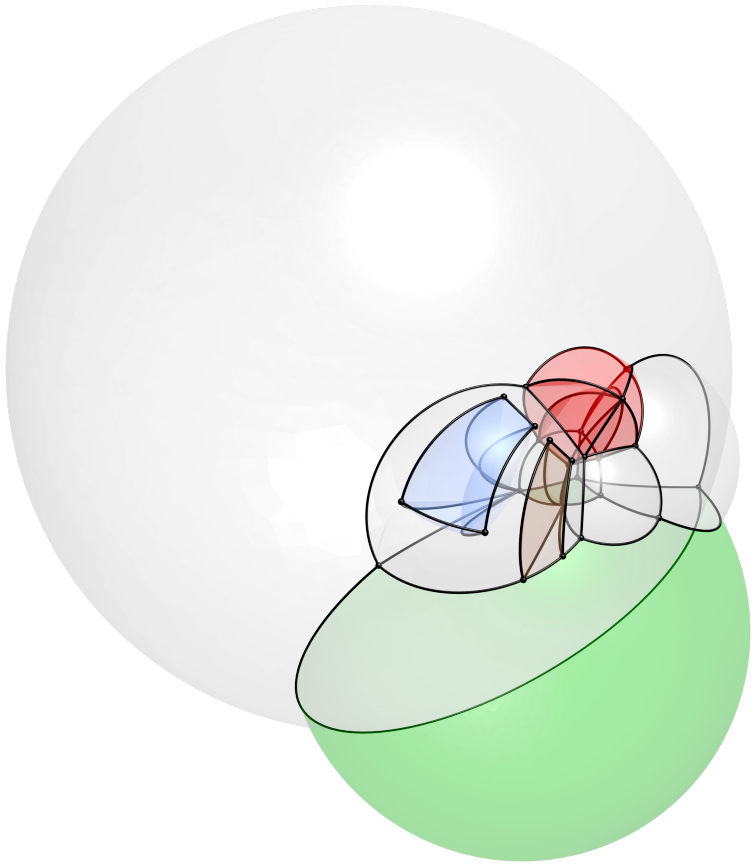


Figure 9.10: A 4D model of a house is projected from $\mathbb{R}^4$ inwards/outwards to the 3-sphere S^3, then stereographically to $\mathbb{R}^3$, finally using a perspective projection down to $\mathbb{R}^2$. A $\pi/5$ rotation along the plane passing through the x_0 and x_3 (LOD) axes makes the bottom and back 2-cells larger than the rest.

any dimension. However, simple selections of the objects within a region or selections involving a finite number of discrete points are relatively straightforward and can be implemented with existing techniques [Hanson, 1994].

On the other hand, projecting higher-dimensional datasets into lower dimensions is simple. As the projection methods explained and developed in this chapter have shown, many projections have dimension-independent formulations using linear algebra that are not that different from their 3D-to-2D versions. They are therefore relatively easy to implement using similar pipelines as current processes [Chu et al., 2009], even if they have not been implemented for general datasets within this thesis due to time constraints.

As this chapter focused on extracting lower-dimensional information from higher-dimensional models from a high-level perspective and in a generic way, many interesting techniques covering useful special cases have not been explored here. For instance, several au-

thors discuss the visualisation of specific classes of objects, generally in 4D. Hoffmann and Zhou [1990] describes methods to visualise surfaces embedded in 4D space. Balsys and Suffern [2007] render implicit (parametrised) surfaces of 4D objects as evaluated sets of points.

Also missing from this chapter was any mention of user interaction and the definition of good camera parameters, both of which are important in order to define values for the input variables described in every projection method. Feiner and Beshers [1990] implemented a system where a user sets such variables using a glove. Zhang and Hanson [2007] uses haptic controllers to explore the 3D shadows casted by 4D objects.

It is good to note that while the techniques mentioned in this chapter are also applicable to different types of objects than those used in GIS and useful for generic applications. Hanson and Weiskopf [2001] uses similar techniques to those mentioned here in order to visualise relativity, Bajaj et al. [1998] does so for n-dimensional scalar fields.

Processing real-world datasets into clean geometric models

10

The representations described in Part I and the operations described in Part II work well on perfectly *valid* data. Objects are assumed not to overlap each other, to be properly closed with no degenerate geometries, and to have perfectly planar faces which are consistently oriented, among many other validity criteria. Unfortunately, as §10.1 explains, GIS processes often fail to clearly specify which criteria are expected and real-world data often fails to meet them, with consequences ranging from the innocuous to a complete inability to use a desired tool, including instances where software gives erroneous results unbeknownst to the user. As GIS datasets can be rather intricate and expensive to acquire, they are not easily replaceable and must therefore be *repaired*, i.e. they must be processed into geometric models that conform to certain validity specifications, so as to make them fit for use.

In the context of this thesis, clean geometric models are important as they are the base for the higher-dimensional models using the representations and operations described in previous chapters. The following sections thus describe the background and a particular solution used in this thesis to obtain valid polygons and planar partitions in §10.2, and valid polyhedra and 3D space partitions in §10.3. The chapter concludes with a generalisation of the definition of validity in arbitrary dimensions in §10.4, which can be used for both higher-dimensional objects and space partitions.

§10.2 is largely based on the papers:

- **Validation and automatic repair of planar partitions using a constrained triangulation**. Ken Arroyo Ohori, Hugo Ledoux and Martijn Meijers. *Photogrammetrie, Fernerkundung, Geoinformation* 5, October 2012, pp. 613–630.
- **A triangulation-based approach to automatically repair GIS polygons**. Hugo Ledoux, Ken Arroyo Ohori and Martijn Meijers. *Computers & Geosciences* 66, May 2014, pp. 121–131.

10.1 Motivation

The representations described in Part I are each able to effectively represent a particular class of objects. For example, most data structures are intended for 3D space partitions whose 3D objects have surfaces that form 2-manifolds, and nD combinatorial maps are capable of representing subdivisions of orientable quasi-manifolds, which within this thesis are generally further limited to having only linear geometries.

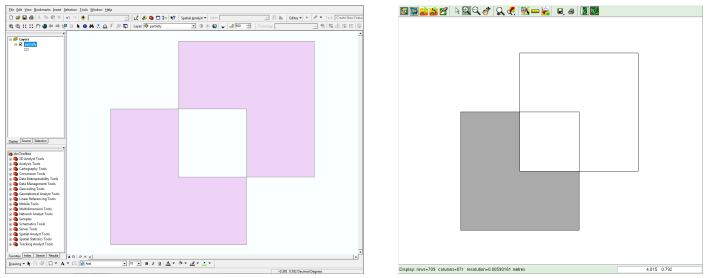
Similarly, the operations described in Part II can only return good results when the input fulfils certain requirements. For instance, the extrusion operation from Chapter 6 requires the input data to form a space partition, the incremental construction algorithm from Chapter 7 requires the ridges of the facets of an object to form matching pairs (i.e. a quasi-manifold), and the linking approach from Chapter 8 will only form a valid 4D cell complex with 4-cells if a matching scheme that preserves all topological relationships between cells can be found or is provided. All these operations are also based on the assumption that the input cells themselves are valid, and are being completely bounded by valid lower-dimensional cells (i.e. facets, ridges, etc.) so as to form a valid cell complex.

The representations and operations presented in this thesis are not special in this sense. Validity assumptions are widely used in all software, especially when complex data is used as input, and are used to make many tasks more manageable, such as to interpret a dataset and load it into a particular data structure for internal usage, as well as for further operations that may be performed using this structure. However, GIS datasets are more complex than most other data[140], and GIS software thus tends to make more assumptions than most other software.

140: For instance, unlike other types of datasets, GIS objects often cannot be stored directly as a plain list of tuples, but are instead decomposed into primitives of a certain shape (§3.1), sometimes recursively, and these have to be defined in terms of its dimension and structure, topological relations, geometry and attributes, sometimes including rich semantics as well.

Moreover, though making sure that these assumptions are true is highly desirable, testing for every possible invalid configuration in a spatial dataset is cumbersome and often unnecessary for a particular task at hand, while testing for only some invalid configurations depending on what needs to be done can easily become intractable and can result in a large number of redundant tests. Running these validation tests can also be difficult and computationally expensive, as many tests take longer to execute than some of the common tasks that a GIS is used for (e.g. visualisation of a dataset, simple statistical analyses, or checking the attributes of some objects).

At the same time, the datasets found in the GIS world are very diverse and their properties vary significantly. They can be generated using a large variety of GIS, CAD and 3D modelling software based on different processes, complying to different specifications and stored in different formats, each of which follows its own internal logic and structure. Most importantly, GIS datasets are created for



(a) (b)

Figure 10.1: Different interpretations of a polygon with a hole that is partly outside its outer boundary (p_3 in Figure 10.2). (a) ArcGIS[141] considers the overlapping region as a hole, but the non-overlapping part of the hole as a new polygon (QGIS[142] and FME[143] do this as well). (b) GRASS[144] removes the overlapping part from the polygon, becoming a new polygon with a different shape. No warning is shown in any software.

different purposes or meant for general-purpose applications (e.g. CityGML [Gröger et al., 2012]). As such, a dataset's specifications are often only vaguely defined, are defined only with regard for a particular application, or are not conformed with in practice.

It is thus perhaps unsurprising that invalid GIS datasets are prevalent [Panigrahi, 2014, Ch. 7] and a major source of problems for those who work with them. As shown in Figure 10.1, invalid datasets can be interpreted inconsistently in different software, leading to inconsistent or erroneous results. They can also make it impossible to perform a certain operation, either due to a failing precondition check or due to software crashes. A partial solution to this issue lies in *validating* these datasets, i.e. identifying the problematic objects in the data so that they can be discarded or (manually) fixed. There are a variety of checklists and (semi) automatic tools for this purpose, such as those provided by SAFE[145] and ESRI[146].

For example, it is possible to incorporate a set of formal preconditions for every operation, as is done in the design by contract software engineering pattern [Meyer, 1986] or the Eiffel programming language [ISO/IEC, 2007]. However, simply discarding problematic (subsets of) datasets is not always feasible, as GIS datasets can be expensive to acquire and thus irreplaceable in practice, and manually fixing errors can be an extremely time-consuming process. In fact, according to McKenney [1998], users of 3D CAD models for finite element analysis—which has similar requirements as certain computations in GIS, such as well-shaped and non-overlapping mesh elements—spend up to 70% of their time fixing the input CAD models. While similar figures for GIS are to the best of my knowledge not available, it is worth noting that CAD software tends to produce better quality models than GIS software[147].

A more complete solution therefore lies in using methods to automatically *repair* a dataset, i.e. to make it conform to a particular set of validity criteria, enabling the full use of many more datasets than would otherwise be possible. As this requires a rather complex defensive programming approach, testing for various types of par-

108: http://www.esri.com/software/arcgis/

109: http://www.qgis.org

110: http://www.safe.com/fme/

111: http://grass.osgeo.org/

145: http://blog.safe.com/2014/11/data-quality-checklist/

146: http://www.esri.com/software/arcgis/extensions/arcgis-data-reviewer/~/media/Files/Pdfs/library/fliers/pdfs/arcgis-data-reviewer-checks.pdf

147: There are many reasons for this. For instance, CAD software makes wider use of topological data structures, and also has topology-aware and smart interactive editing tools (e.g. snapping to guide lines and nearby objects), which help to avoid problems where objects seem to be valid but have small errors, such as sliver polygons and shells that are not properly closed.

tially overlapping cascading errors and fixing them accordingly, it is best performed separately and called as needed rather than integrated into every operation of a GIS. The following sections describe such independent repair methods as were used in this thesis, which involve the creation of valid (multi)polygons and planar partitions from 2D GIS datasets (§10.2), and the creation of valid polyhedra and 3D space partitions from 3D BIM datasets (§10.3). These were then used as input for the different experiments described in Part II.

10.2 Creating valid (multi)polygons and planar partitions

10.2.1 What is a valid (multi)polygon or planar partition?

In most GIS file formats and the software that reads and writes them, polygons and multipolygons are defined in a manner that is consistent with the definitions in the Simple Features Specification [OGC, 2011; ISO, 2006]—an implementation of the ISO 19107 standard [ISO, 2005a]. The specification states that: '*A* `Polygon` *is a planar* `Surface` *defined by 1 exterior boundary and 0 or more interior boundaries. Each interior boundary defines a hole in the* `Polygon`'. Each of these boundaries is described as a `LinearRing` (cf. Figure 3.17 on page 42). According to the specification, an outer ring should be oriented *anticlockwise* when viewed from a predefined *top* direction, which is generally (but not necessarily) the viewing direction in 2D or *outwards* when the polygon specifies part of the boundary of a polyhedron. Inner rings should be oppositely oriented, i.e. generally *clockwise* when viewed from the top direction.

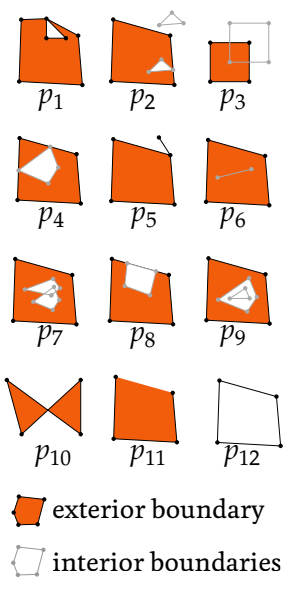


Figure 10.2: Several invalid polygons, with their outer boundaries shown in black and their inner boundaries in grey. The orange areas represent one possible interpretation of the interior of the polygons. Polygon p_{12} has an exterior and an interior boundary with the same geometry.

The Simple Features Specification provides several **validity rules for polygons**, the most relevant of which are described below with examples of invalid polygons provided in Figure 10.2. The rules can be summarised as follows:

- each ring defining the exterior and interior boundaries is *simple*, i.e. non-self-intersecting (p_1 and p_{10});
- each ring is closed (p_{11}), i.e. its first and its last points should be the same;
- the rings of a polygon do not cross (p_3, p_7, p_8 and p_{12}), but they may intersect at one tangent point;
- a polygon does not have cut lines, spikes or punctures (p_5 and p_6);
- the interior of every polygon is a connected point set (p_4);
- each interior ring creates a new area that is disconnected from the exterior (p_2 and p_9).

Similarly, the specification provides a definition and some **validity rules for multipolygons**. A `MultiPolygon` is defined as a `MultiSurface` forming an aggregation of `Polygons`, which also follows certain validity criteria, which can be summarised as follows:

- the interiors of its polygons do not overlap, i.e. their point set intersection should be empty;
- the boundaries of its polygons may only touch at a finite number of points;
- a multipolygon does not have cut lines, spikes or punctures;
- the interior of a multipolygon with more than one polygon is *not* a connected point set.

Intuitively, a *planar partition* is a set of polygons that form a subdivision of a region of the plane. Planar partitions are thus commonly used to model concepts where objects are expected not to overlap, such as land cover, cadastral parcels, or the administrative boundaries of a given country. Despite being a very frequently used representation in GIS, planar partitions are not explicitly defined in the main GIS standards.

Within the classes in the ISO 19107 standard [ISO, 2005a, §6.6], a planar partition could be considered as a `GM_CompositeSurface`, defined in the standard as '*a collection of oriented surfaces that join in pairs on common boundary curves and which, when considered as a whole, form a single surface*'. By following this definition, overlaps between polygons are explicitly forbidden, as a `GM_Complex` (a parent of `GM_CompositeSurface`) is defined as '*a set of primitive geometric objects (in a common coordinate system) whose interiors are disjoint*'. However, a `GM_CompositeSurface` explicitly allows gaps between the surfaces, as these would simply result in inner rings within the overarching single surface.

An alternative definition could be created based on the ISO 19123 standard [ISO, 2007a, §6.8]—a standard focusing on coverages of various types. According to the standard, a planar partition can be considered as a type of `CV_DiscreteSurfaceCoverage` where '*the surfaces that constitute the domain of a coverage are mutually exclusive and exhaustively partition the extent of the coverage*'. Overlapping polygons are disallowed by them being '*mutually exclusive*' and gaps are disallowed by the surfaces '*exhaustively partitioning*' the extent. However, the standard states these conditions as something that occurs '*in most cases*', whereas in a planar partition it should be considered as a strict prerequisite.

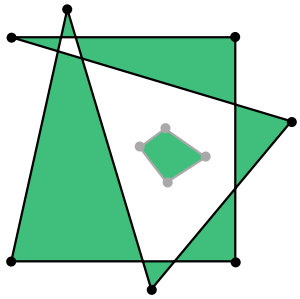
In a **valid planar partition**, there should thus be no overlapping polygons, and no gaps between them either unless these gaps are considered to be outside of the region. These two conditions are covered by the ISO 19107 standard in a different context, when it lists

some possible inconsistencies of 'spaghetti' datasets represented as a `GM_Complex`, stating that '*slivers and gaps are multiple lines that should represent the same geometry, but do not coincide, leaving areas of overlap between two surface boundaries (slivers), and gaps between them*' [ISO, 2005a, §6.2.2.6].

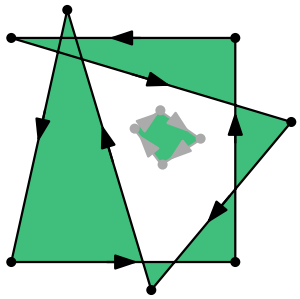
10.2.2 Commonly used validation and repair methods

Starting from an arrangement of line segments that are meant to define the boundary of a (multi)polygon, there are various rules that can be used to define its interior and exterior. Foley et al. [1995] discusses two commonly used sets of rules in vector-based graphic software, which are shown in Figure 10.3[148].

148: Foley et al. [1995] considers only polygons, but the rules as explained here also cover holes in polygons and multipolygons.



(a)



(b)

Figure 10.3: (a) According to the *odd-even rule*, a polygon's interior is the region(s) that can be accessed by passing through an odd number of edges from its exterior. (b) According to the *non-zero winding rule*, a polygon's interior is the region(s) around which the boundaries of a polygon do a non-zero number of revolutions.

In practice, GIS users often repair invalid polygons manually. Among the few documented automatic solutions, it is possible to use a 'buffer-by-0' operation [Ramsey, 2010] or PostGIS 2.0's `ST_MakeValid` function[149].

The validation of planar partitions is usually performed using a checklist of individual tests that together ensure its validity. For instance, Plümer and Gröger [1997] specify that a valid planar partition consists of: no dangling edges, no zero-length edges, planarity, no holes, no self-intersections, no overlaps, and having a connected graph. However, it is worth noting that without the use of a supporting structure, some of these tests can be problematic or computationally expensive. For instance, checking whether any possible pair of polygons overlap can have quadratic behaviour even when heuristics to speed up the process are used [Badawy and Aref, 1999; Kirkpatrick et al., 2000], and robustness issues are significant in polygon intersection tests [Hoffmann, 1988]. Finding the potential gaps in a planar partition is also a problem, as it can require computing the union of the entire set of polygons [Margalit and Knott, 1989; Rivero and Feito, 2000].

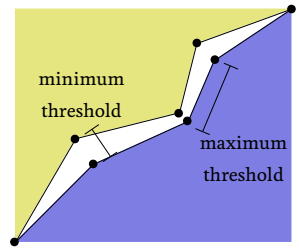
The most common method used to repair a planar partition is based on the assumption that polygons *approximately* match each other at their common boundaries. If the adjacent polygons are within a certain distance *threshold* of each other along their common boundaries (Figure 10.4a), and all parts further apart than this threshold are known not to be common boundaries (Figure 10.4b), it is possible to *snap* together the polygons using this threshold, while in theory keeping the rest untouched. This method of planar partition repair is available in many GIS packages, including ArcGIS, FME, GRASS and Radius Topology[150].

The threshold value for certain input dataset(s) is then usually manually determined, either by trial and error, or by analysing certain properties of the datasets involved (e.g. point spacing, precision, or map scale). However, it is often hard to find an optimal threshold

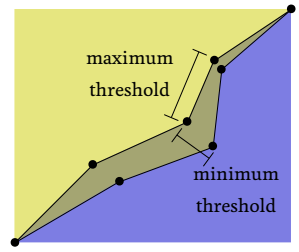
149: `http://postgis.org/documentation/manual-svn/ST_MakeValid.html`

150: `http://www.1spatial.com/software/radius_topology/`

for certain datasets, and sometimes impossible as such a threshold does not even exist (e.g. because point spacing in some places might be smaller than the width of the gaps and overlaps present).



(a)



(b)

Figure 10.4: Defining a snapping threshold taking into account: (a) gaps and (b) overlaps.

10.2.3 Repair using a constrained triangulation

The method developed to repair polygons and planar partitions uses a constrained triangulation of the input polygons as a base structure. Constrained triangulations have distinct properties that make them useful as a base for a repair algorithm. The can be built efficiently with a variety of approaches [Guibas and Stolfi, 1985; Clarkson et al., 1992], can easily be made numerically robust[151], can be used for quick traversal and point location [Mücke et al., 1999], and have fast and robust implementations in several libraries, such as CGAL [Boissonnat et al., 2002], Triangle[152] [Shewchuk, 1997] and GTS[153].

The method used to **repair individual (multi)polygons** exploits these properties and consists of three broad steps, which are shown in Figure 10.5 and described as follows:

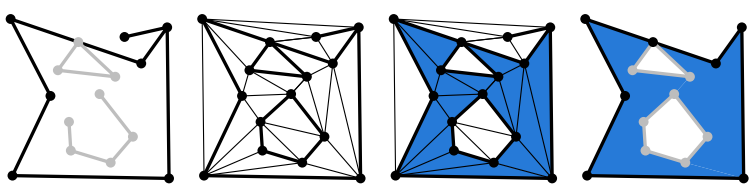
1. construction of the constrained triangulation of the line segments in the input, processing outer and inner rings identically and including an extra edge that connects the first and last vertices of a ring when these are not the same (Figure 10.5b);
2. labelling of each triangle as either *outside* or *inside*, which is based on an extension of the odd-even rule that supports overlapping lines by adding or removing (parts of) constraints in the triangulation (Figure 10.5c), taking only edges that are *constraints* into account;
3. reconstruction of the interior areas as a repaired multipolygon[154] (Figure 10.5d).

This method is remarkably efficient, and its implementation based on CGAL classes is able to process large polygons quickly. As an example, Figure 10.6 shows the process on the largest polygon in the CORINE land cover dataset[155], which consists of almost 1 189 903

151: Based on one robust geometric predicate that tests whether three successive points are collinear, have a clockwise or a anticlockwise orientation (e.g. Shewchuk [1996b]) and the computation of new vertices at the intersections of line segments. A constrained Delaunay triangulation only requires an additional predicate that determines whether a point lies inside, on or outside the circle defined by three other points.

152: `https://www.cs.cmu.edu/~quake/triangle.html`

153: `http://gts.sourceforge.net`



(a) (b) (c) (d)

Figure 10.5: Steps to repair a (multi)polygon using a constrained triangulation: (a) input data, (b) triangulation, (c) labelling, and (d) reconstruction.

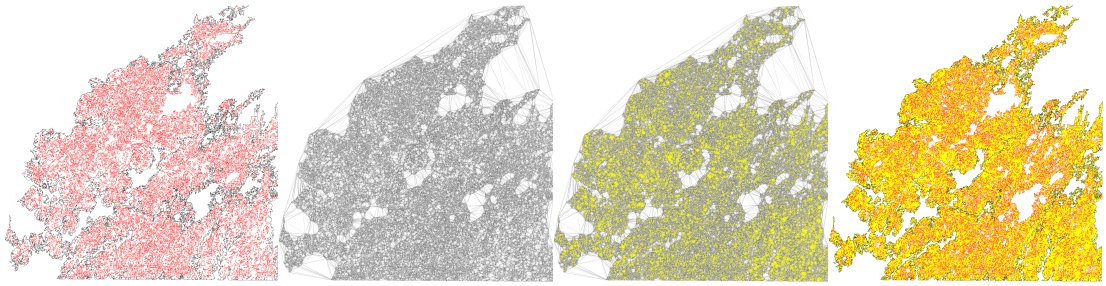


Figure 10.6: Processing the largest polygon in the CORINE land cover dataset: (a) outer and inner boundaries, (b) triangulation, (c) labelling, (d) reconstruction.

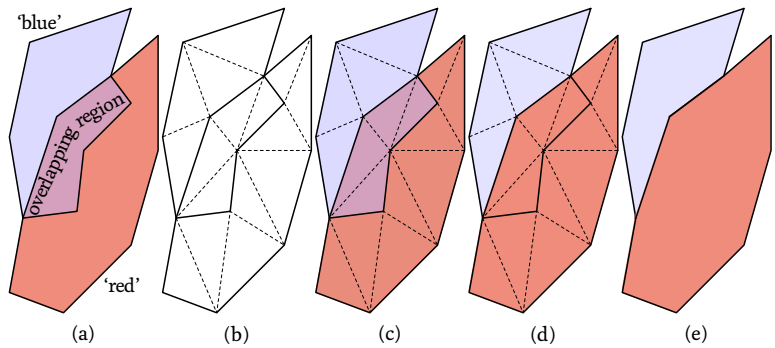


Figure 10.7: Steps to repair a planar partition using a constrained triangulation: (a) input data, (b) triangulation, (c) labelling, (d) re-labelling problematic triangles, (e) reconstruction.

154: The output might only be representable as a multipolygon even if the input was a polygon

155: http://www.eea.europa.eu/publications/CORO-landcover

vertices and 7 672 holes, which is processed in under a second.

The method to **repair a planar partition** then uses polygons which are known to be valid based on the previous method. It consists of four main steps, shown in Figure 10.7, and is as follows:

1. the constrained triangulation of the input segments forming the (now valid) polygons is constructed;
2. each triangle is labelled with the labels of the polygons inside which it is located, such that problems are detected by identifying triangles having no or multiple labels;
3. problems are fixed by re-labelling triangles according to customisable criteria, such that each triangle has exactly one label;
4. the polygons are reconstructed from the triangulation.

Various local repair methods can thus be defined, all of which are based on choosing how to relabel a triangle which no label or with

(a) (b) (c) (d)

Figure 10.8: Various repair methods can be defined based on relabelling triangles or sets of connected triangles. For instance, based on (a) the input data, it is possible to relabel: (b) an invalid triangle based using its longest boundary with a neighbour, (c) an invalid region of connected triangles using its longest boundary with a neighbour, or (d) an invalid region of connected triangles using a random neighbour.

multiple labels. Figure 10.8 shows a few examples of such methods. Within this thesis, planar partitions are obtained by repairing invalid regions using the longest boundary with a neighbour (Figure 10.8c) if possible, as this tends to produce a cartographically more pleasing result, and a random neighbour (Figure 10.8d) otherwise.

This method is able to process large datasets quickly, such as the one shown in Figure 10.9, which consists of 16 tiles of the CORINE land cover dataset in a 4×4 configuration. It has 63 868 polygons with a total of 6 622 133 vertices and was processed in 4 minutes 47 seconds. By comparison, both ArcGIS and GRASS are unable to repair this dataset by snapping geometries, while FME repairs it in 15 minutes 48 seconds[156], also using snapping. Note that apart from the fact that snapping is slower, it does not guarantee a valid result.

156: On tests where ArcGIS and GRASS were able to process the data, they were slower than FME.

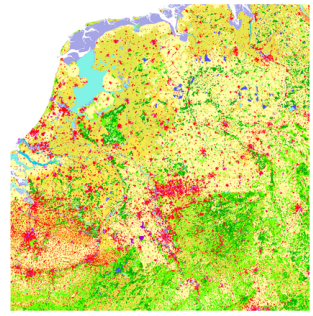


Figure 10.9: A planar partition made from 16 tiles of the CORINE land cover dataset.

10.3 Creating valid polyhedra and 3D space partitions

10.3.1 Motivation: IFC input data

As discussed in §3.2.2, the data models used in 3D GIS have often favoured an approach where objects that are volumetric in reality are modelled as a set of their (visible) surfaces, which can be easily captured after the objects are built. In addition, the volumetric objects that are small, elongated or thin are sometimes not modelled as volumes, but they are instead respectively modelled as points, curves or surfaces. For instance, in 3D models encoded in CityGML [Gröger et al., 2012], all objects are modelled as surfaces that do not necessarily form closed volumes, and thin objects (e.g. roof overhangs, windows and doors) are often modelled as single surfaces.

The volumetric models that are often used in CAD and BIM, such as those in IFC format [ISO, 2013], follow a different approach, which

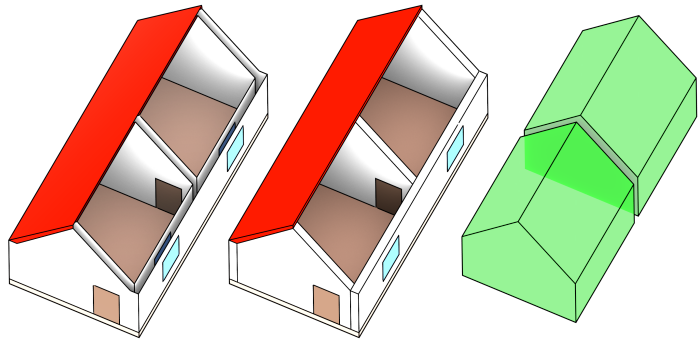


Figure 10.10: A surface-based model (left) consists of a set of semantically labelled surfaces, which are shown here in different colours. A volumetric model (centre+right) instead consists of a set of semantically labelled *volumes*.

is shown in Figure 10.10. In it, almost all real-world volumetric objects, i.e. the objects with a non-zero volume, are modelled as volumes as well. This approach is more expensive in terms of space and makes certain computations more difficult, such as obtaining the volume of a room that is only represented implicitly by a set of (volumetric) walls, doors and windows around it. However, the approach is ultimately a more powerful representation that is closer to reality, enabling more complex operations, such as the structural analysis of a building, and eliminates the ambiguities inherent in interpreting volumetric real-world objects that have been modelled as points, curves or surfaces.

Although the volumes in such a model do not generally fit together perfectly, they can nevertheless be better processed[157] into a 3D space partition consisting of a set of non-overlapping 3D objects, where each of the volumes—and optionally also their vertices, edges and faces—is labelled with appropriate semantic information[158]. The result is thus a *space-partitioning 3D model*, which is analogous to the 2D planar partitions often used to model coverages in GIS. Such a model can be stored in a computer using the topological data structures for 3D (§3.1.3) or nD (§4.3.5) cell complexes.

157: Akin to how a set of polygons can be better processed into a planar partition compared to a rough line drawing with undershoots and overshoots (i.e. a typical spaghetti dataset).

158: As shown in §5.4, these lower-dimensional cells can store important information about the relationships between them objects.

This makes a volumetric model *a better base for a higher-dimensional representation*, as it fully exploits the properties of higher-dimensional data structures and their operations. For instance, taking the examples of this thesis, it can be extruded into higher dimensions (Chapter 6) or multiple such 3D models can be linked into a single 4D model (Chapter 8).

A space-partitioning model is also better able to take advantage of the complex 3D models that are already created during the design and construction processes of a building, such as the faces that form a room, storey or building. This avoids the need to extract or abstract appropriate surfaces for their recreation in a GIS model, such as was done by Donkers [2013]. Since architectural and BIM models have a

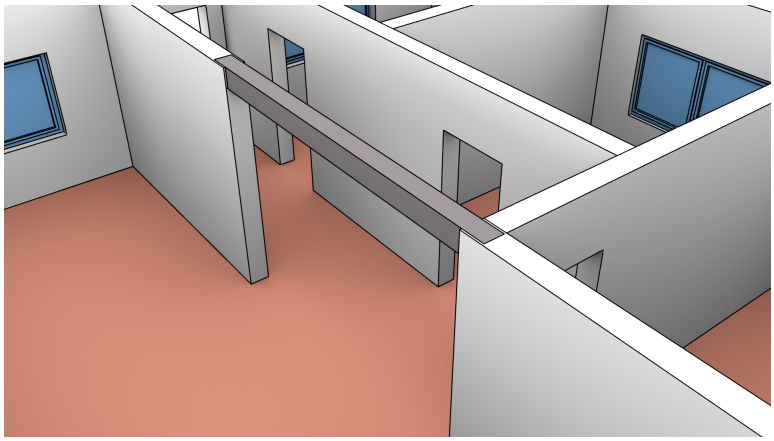


Figure 10.11: An IFC model of the FZK-house[159] contains a complex representation of the structural elements of the building. Note how the volumes in this model fit together, having no overlaps even in places that are normally not visible (e.g. the beam ends which are embedded in the walls).

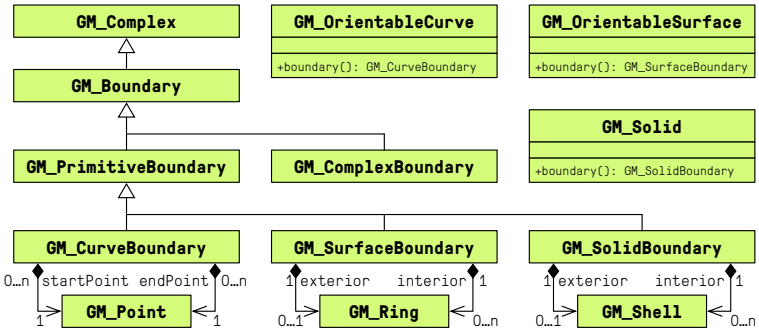


Figure 10.12: The ISO 19107 standard [ISO, 2005a, §6.3.2] is able to specify the boundaries of `GM_Curve`, `GM_Surface` and `GM_Solid` as subclasses of `GM_Boundary`, respectively a `GM_CurveBoundary` linked to a pair of `GM_Point` (the end-points of a line segment), `GM_SurfaceBoundary` linked to a set of instances of `GM_Ring`, and a `GM_SolidBoundary` linked to set of instances of `GM_Shell`.

strong emphasis on representing individual 3D elements that need to be designed, manufactured and put together, such as those shown in Figure 10.11, as well as the fact that these elements generally do not overlap in reality (except as part of hierarchies), they are ideally suited to be used in a space-partitioning model.

159: http://iai-typo3.iai.fzk.de/www-extern/index.php?id=1174&L=1

10.3.2 What is a valid solid or 3D space partition?

The ISO 19107 standard [ISO, 2005a, §6.3.18] defines 3D objects with 3D holes that are known as *solids*, which are specified based on a boundary representation scheme. As shown in Figure 10.12, the standard thus defines a `GM_Solid` with a *boundary* operation returning a `GM_SolidBoundary`, which is a '*sequence of sets of* `GM_Surfaces` *that limit the extent of [the]* `GM_Solid`'. Each of these sets of surfaces describes one of the boundaries of the `GM_Solid` as a `GM_Shell`, corresponding to either the outer boundary for the solid[160] or one of its holes.

A `GM_Shell` [ISO, 2005a, §6.3.8] thus represents '*a single connected*

160: In some cases, there might not be an outer boundary of a solid, such as in non-Euclidean spaces or in the representation of unbounded solids. However, there is nearly always an outer boundary in the context of geographic information.

component of a `GM_SolidBoundary`'. It is known to be *simple*, and consists of a set of oriented instances of `GM_Surface` composed of instances of `GM_SurfacePatch`, which intuitively form a cellular subdivision of the surface and themselves have a `GM_SurfaceBoundary`. A `GM_SurfaceBoundary` represents an area potentially with any number of holes, each of which is stored as a reference to a `GM_Ring`. A `GM_Ring` [ISO, 2005a, §6.3.6] is additionally defined as being *simple*.

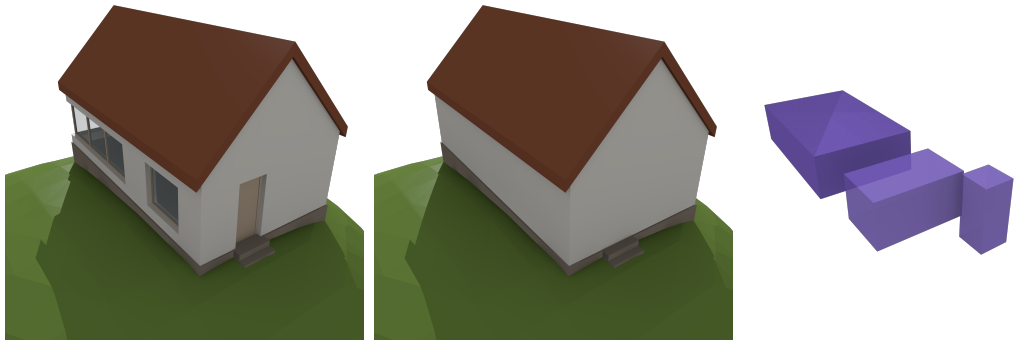
`GM_Object` [ISO, 2005a, §6.2.2], a parent class to all the classes previously mentioned, defines every object as a point set and provides the definition of simple as a '*GM_Object [that] has no interior point of self-intersection or self-tangency. In mathematical formalisms, this means that every point in the interior of the object must have a metric neighborhood whose intersection with the object is isomorphic to an n-sphere, where n is the dimension of this `GM_Object`*'. As discussed by Ledoux [2013], this implies that shells are effectively 2-manifolds. Rings are similarly 1-manifolds.

It is important to note that even though each `GM_Ring` and `GM_Shell` is individually simple, the boundary of the `GM_Surface` or `GM_Solid` that they together describe does not need to be simple. A common example would involve an inner ring/shell tangent to the outer ring/shell containing it. Arguably, the standard does appear to explicitly forbid intersections between the interior of rings or shells as `GM_Complex` is a parent class of `GM_SurfaceBoundary` and `GM_SolidBoundary` and this class requires its composing primitives to be '*geometrically disjoint*'. However, this interpretation is problematic as it would arguably also forbid inner rings being inside their containing outer ring.

Alternatively, it is possible to consider that the standard does not specify *any* restrictions regarding the interactions between rings of a surface or between shells of a solid. As the standard explicitly states that '*implementations may enforce stronger restrictions on the interaction of boundary elements*', it might be the responsibility of other implementing standards to place appropriate restrictions.

Although the GML standard [OGC, 2007] implementing ISO 19107 does not specify such restrictions, it is possible to use those defined in the Simple Features Specification in 2D (§10.2) and define analogous ones in 3D [Ledoux, 2013]. One possible formulation of these could be as follows:

- the shells of a solid do not cross, but the shells on the boundary of a solid may intersect only at a vertex or edge;
- the interior of every solid is a connected point set;
- each interior shell creates a new volume that is disconnected from the exterior.



(a) (b) (c)

Figure 10.13: In the (a) IfcOpenHouse dataset[161], the openings where the windows and door fit are not explicitly carved out from the wall volumes. Instead, the dataset consists of (b) walls with simple shapes and (c) the openings themselves as large boxes. A Boolean point set difference thus needs to be computed in order to obtain the final explicit geometries.

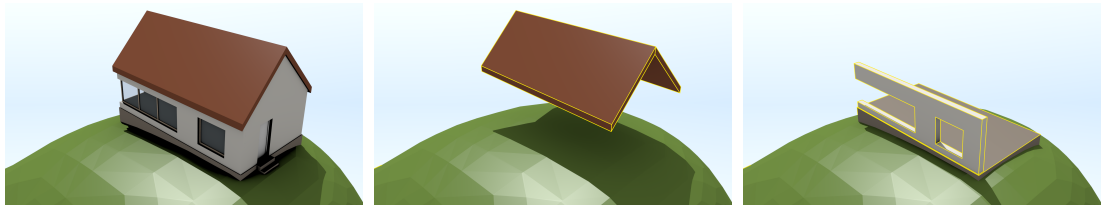
Intuitively, a 3D space partition is a subdivision of a region of 3D space into non-overlapping solids. However, just as with planar partitions, 3D space partitions are usually not strictly defined. Following the same logic as with planar partitions in §10.2, a 3D space partition can be considered as an ISO 19107 `GM_CompositeSolid` [ISO, 2005a, §6.6.13], which is defined in the standard as a '*a set of solids that join in pairs on common boundary surfaces to form a single solid*'. While overlapping solids are explicitly forbidden by a `GM_CompositeSolid` inheriting from `GM_Complex` in which '*[primitive] interiors are disjoint*', gaps between the solids are explicitly allowed.

An alternative definition could also be created based on the ISO 19123 standard by considering a 3D space partition as a type of `CV_DiscreteSolidCoverage` [ISO, 2007a, §6.10], which states that '*generally, the solids that constitute the domain of a coverage are mutually exclusive and exhaustively partition the extent of the coverage*'. While overlaps and gaps are respectively eliminated by the '*mutually exclusive*' and '*exhaustively partition*' conditions, the word '*generally*' implies that these are not always enforced.

10.3.3 Common problems for 3D objects in an IFC file

An IFC model in theory consists of a set of 3D objects that mostly do not overlap. However, there are a few common problems that cause some of these 3D objects to be invalid or prevents them from forming a clean space partition. The most significant of these are the following:

161: http://blog.ifcopenshell.org/2012/11/say-hi-to-ifcopenhouse.html



(a) (b) (c)

Figure 10.14: The volumes in the (a) IfcOpenHouse dataset do not perfectly match each other at their common boundaries. (b) The two volumes forming the roof actually do not touch, causing the house not to be closed. (c) The left wall and the foundation volumes have a small overlapping portion (i.e. the wall sinks inside the foundation).

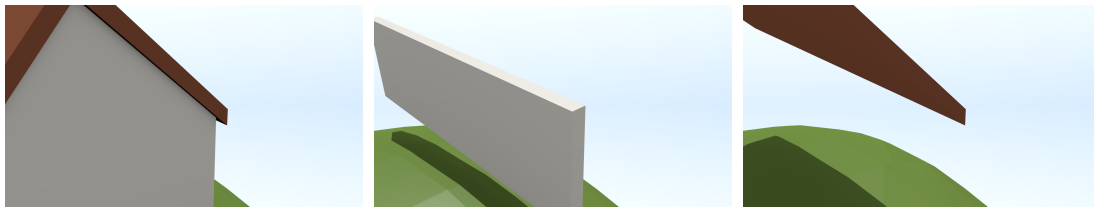
Implicit geometries Objects are often represented using sweeps, intersections of half-spaces and Boolean set operations, such as in the case of openings as shown in Figure 10.13. These need to be converted to explicit (boundary representation) objects that can be made to fit with other objects. As these need to be discretised, they will often not perfectly match the shape of the original objects, sometimes creating problems in their interaction with other objects.

Local coordinate systems Objects are defined using a local coordinate system, which might differ per object. As the parameters of these coordinate systems are stored (and computed) in a finite computer representation, this will cause the objects not to fit together perfectly. In addition, applying a transformation to embed all objects into a unique coordinate system using computed arithmetic will cause additional problems. The result is that objects that visually appear to fit together do not actually do so, having small gaps and overlaps between them as shown in Figure 10.14.

Hidden intentional overlaps Not caring about hidden object intersections is common practice in most 3D modelling approaches. By not caring about these intersections, it is possible to ease and speed up the modelling process by using simpler volumes (e.g. boxes or rectangles) than would otherwise be required and hiding their undesirable parts behind or inside other objects, as is shown in Figure 10.15.

10.3.4 Commonly used validation and repair methods

In order for software to be able to process invalid polyhedra, various automatic validation and repair methods have been developed, detecting and/or fixing some of the possible invalid configurations that can exist. The possible invalid configurations are many and partly overlap or cascade (from lower to higher dimensions), but a



(a) (b) (c)

Figure 10.15: (a) The right wall and the right roof volumes in the IfcOpenHouse dataset visually appear to match each other, but the (b) right wall and (c) right roof are modelled as parallelepipeds, so they have a non-empty intersection (a triangular pyramid).

possible list can be generated by systematically exploring those that occur at the ring, polygon, shell and solid levels[162]. For instance, Wagner et al. [2013] tests for 13 types of invalid configurations while Ledoux [2013] tests for 26, notably including: consecutive points in a ring with the same coordinates, rings that are not closed or self-intersect, polygons with intersecting rings or with an inner ring outside the outer ring, shells that are not closed or are not 2-manifold, and solids with intersecting shells.

162: In theory, errors can occur at the point and edge level, such as a lack or excess of coordinates, but on a typical GIS input file these would be generally considered as syntactic rather than geometric errors.

Repairing individual rings and polygons can be basically done using the processes outlined in §10.2, even if these are embedded in 3D rather than 2D. For this, rings and polygons can be first projected onto a certain plane (e.g. the best-fitting one, or one obtained by disregarding a coordinate of its vertices in a manner that does not create new degenerate shapes) or a restricted triangulation [Cheng et al., 2012, Ch. 13] could be used as a basis for a repair procedure.

Repairing shells and solids are problems that are also partly related to those discussed previously, as a shell can be seen as a planar partition that wraps around an object, i.e. it is watertight, while the constraints that define how the inner and outer shells of a valid solid should interact are similar to those defining the interaction of inner and outer rings within a valid polygon. However, the methods that have been presented previously are much less applicable to shells and solids.

Instead, a shell can often be seen as a mesh that *should* be closed, and it is thus possible to repair individual shells with the procedures used for surface reconstruction and mesh repair. There are various good surveys of the methods that can be used to repair various problems in polygonal meshes, such as Ju [2009] and Attene et al. [2013]. Some of these problems and their respective methods are summarised below. However, it is important to notice that many methods are intended for meshes representing the boundary of a single 'smooth' 2-manifold, making them not applicable to the great majority of BIM and GIS models where perpendicular angles are common (e.g. those between walls and floors/ceilings).

Rossignac and Cardoze [1999] propose a method to make polygonal meshes combinatorial manifolds, determining an appropriate order for a set of edges around a face or for a set of faces bounding a volume such that non-manifolds can be stored using a manifold data structure (Figure 3.10 on page 37). Guéziec and Lazarus [2001] treat the problem from a geometric point of view, converting non-manifold edges into thin volumes by cutting and joining the mesh around them. Attene et al. [2009] propose a method to make tetrahedral meshes[163] manifold—something that can be used also for solids by computing their constrained tetrahedralisation.

163: assuming a tetrahedron-based data structure with adjacency relationships to other tetrahedra

In a similar manner as in GIS, small gaps in a mesh (causing a shell not to enclose any space) can be naively repaired by snapping vertices [Rock and Wozny, 1992], but this requires an error-prone threshold and can lead to topological errors. Iteratively snapping boundary edges together works better [Sheng and Meier, 1995], as it is possible to start from a single corresponding pair of edges, and then iteratively 'zip' together corresponding edges that are adjacent to these. Barequet and Sharir [1995] follows a related approach, matching certain edges and triangulating the remaining gaps. Turk and Levoy [1994] shows how overlapping triangular meshes[164] can be fixed by clipping all but one of a set of overlapping triangles, retriangulating them afterwards.

164: As overlaps are embedded in 3D, they need to be defined from a given point of view.

Holes in a mesh, which tend to be bigger than gaps and reflect missing parts of a surface (e.g. the bottom of a house, the sides of a terraced house, or a surface that is hidden from a typical point of view), require different methods. In many cases, a hole is close to planar and can thus be simply projected to 2D and triangulated [Bøhn and Wozny, 1992]. However, in other cases the holes are far from planar, and it is thus necessary to use more complex methods. For instance, Lévy [2003] fills holes while attempting to minimise a certain objective function representing the energy needed to fill it, Wang and Oliveira [2007] uses moving least squares fitting of the points around it[165] and Podolak and Rusinkiewicz [2005] does so by subdividing space into regions deemed to be completely in or out of the shell. Nooruddin and Turk [2003], Bischoff et al. [2005] and Hétroy et al. [2011] use voxel-based methods to attempt to determine the interior and exterior of a shell and thus close a mesh.

165: Proposed by Lancaster and Salkauskas [1981], it is a widely used surface reconstruction method based on interpolating a set of points.

Specifically in the context of buildings, which have different characteristics from many other meshes such as sharp corners and orthogonal surfaces, Bogdahn and Coors [2010] and Alam et al. [2013] propose two methods that create a smooth 2-manifold, but do not guarantee their results. Zhao et al. [2014] attempts to solve gaps, holes and overlaps in a building mesh simultaneously by using a constrained tetrahedralisation [Si and Gärtner, 2005] of a set of faces, progressively carving away tetrahedra that are deemed as not belonging to its interior based on a set of rules. The possible intersections between the faces are explicitly computed by the constrained

tetrahedralisation, so that the starting tetrahedra form a 3D space subdivision.

Finally, the problem of creating a valid 3D space partition is loosely related to mesh simplification and more closely to the *topological reconstruction* of a 3D model, which sometimes deals with computation of topological relationships from imperfect datasets. Generally, the latter methods work by snapping the geometries that lie within a threshold. For instance, Horna et al. [2006] snaps together generalised map darts that lie within a threshold ε, connecting them by the appropriate involution α, which is based on the steps of the reconstruction process.

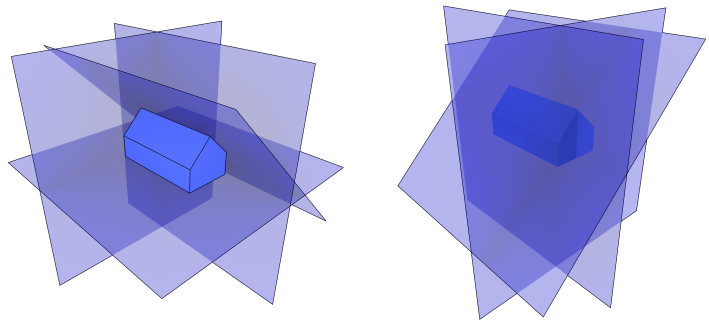
10.3.5 Repair using snapping and Boolean set operations

Despite the fact that the methods presented above fix many of the problems in individual 3D objects, these methods are not always sufficient to create a valid 3D space subdivision. For instance, the methods that are based on snapping are unable to create a space partition when adjacent geometries are farther than the snapping threshold or their overlapping regions have a width larger than the threshold. Similarly, most methods to repair meshes do not guarantee that they form properly enclosed spaces and cannot deal well with solids with inner shells.

A different technique that solves many of these issues was thus developed for this thesis. It starts from a set of separate 3D objects that *approximately* form a 3D space partition, such as those that are commonly found in IFC building models. The method starts by handling every object separately, obtaining a valid interpretation of each that is stored in an exact representation. It then snaps objects together in order to remove small gaps and overlaps at their common boundaries. Finally, it uses Boolean set operations to remove larger overlaps and in order guarantee that a 3D space partition is obtained. Unlike most repair methods used in GIS, it tries to avoid triangulating every face of an object but still manages to obtain perfectly planar faces (in memory). These steps required are described in detail as follows:

1. **Rough individual polyhedra** One or more 'rough' polyhedral representations of every object are first extracted from the input model. These rough representations should be combinatorially valid quasi-manifolds, being composed of patches that join in pairs at their common boundaries. However, at this stage the polygonal patches can have geometric issues, such as not being planar and might intersect geometrically.

   In order to obtain the rough polyhedral representations, every object is first individually parsed, converting implicit geometries into explicit geometries using Boolean set operations



(a) (b)

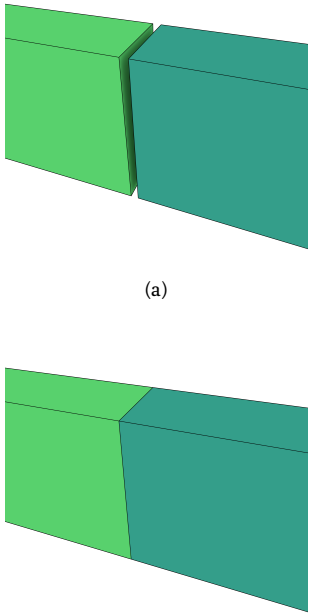
Figure 10.16: The best fitting plane to the vertices of every face is computed, here showing those belonging to (a) the floor, back right wall, back left wall and back eave, and (b) the front left wall, front right wall and front eave.

(Figure 10.13) and triangulating curved surfaces. A transformation is then applied to convert every object's coordinates to a global coordinate system. Finally, the faces of every object are extracted, object by object, and used to incrementally construct a set of polyhedra. This is done by starting from a given face and attempting to add adjacent faces until a closed shell is formed, which is repeated until all faces of an object are processed or the remaining faces cannot form any closed shells.

2. **Clean individual polyhedra** Clean individual polyhedra are then created from the rough polyhedra. In order to do this, the best fitting plane to the vertices of each face is first computed using linear least squares (Figure 10.16) and is stored as a plane equation. Considering these planes as *constraints*, all the vertices of every polyhedron are moved to an exact intersection of as many as possible of its incident face planes using a greedy algorithm unless this would result in a too large shift (as defined by a threshold). If some plane constraints could not be met, the corresponding faces are triangulated, thus becoming perfectly planar. The planes of these new triangular faces are computed.

3. **Snapping vertices together** The vertices of the polyhedra that lie within a threshold are snapped together, which removes most of the small gaps and overlaps in the model (Figure 10.17). This applies to vertices belonging to the same or to different polyhedra. The snapped vertices are now considered *immovable*. Iterating through all the faces of the polyhedra, if a face has at least three immovable non-collinear vertices, the plane passing through these vertices is computed and it is considered as *fixed*. When there are more than three non-coplanar vertices, the face is triangulated and the faces with three immovable vertices are also considered as *fixed*.

4. **Snapping vertices to fixed planes** The vertices that are still



(a) (b)

Figure 10.17: (a) The vertices of different polyhedra that lie within a threshold are snapped together, thus (b) removing a small gap. A small overlap works in the same manner.

considered movable are then snapped to nearby fixed planes, if any, eliminating certain other small gaps and overlaps that do not have vertices in common (e.g. the steps in front of the IfcOpenHouse shown in Figure 10.18, which are actually not touching the house's foundation). For this, iterating through every movable vertex, if it is incident to three or more faces with non-coplanar fixed planes and their intersection lies within a threshold of the vertex's current position, the vertex is moved to the intersection of three of these planes and considered as immovable. If it has more than three incident faces with non-coplanar fixed planes, the faces of the planes that were not used, and are thus now not perfectly planar, are triangulated. This step can be repeated a given number of times, increasing the number of immovable vertices.

5. **Fixing the remaining vertices** The remaining movable vertices are fixed to their incident faces' fixed planes or to their current location. For this, iterating through every movable vertex, the same procedure as the step above is followed. However, if a vertex has less than three incident faces with non-coplanar fixed planes, the vertex is fixed to the position on the intersection of its incident faces' fixed planes that is closest to the vertex's current position. These moved vertices are also considered as fixed and their incident faces' planes are recomputed if necessary.

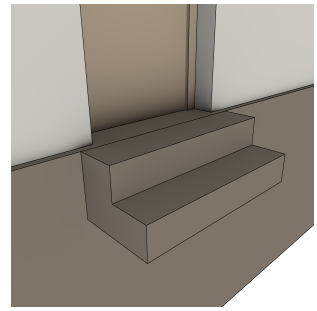


Figure 10.18: The steps of the IfcOpenHouse do not actually touch the house's foundation. Note that it also does not have common vertices with the foundation, so this gap cannot be closed by vertex-to-vertex snapping, but it can be closed by snapping its vertices on the left to the foundation's right plane.

6. **Creating individual Nef polyhedra** A Nef polyhedron is created from every polyhedral representation using the precomputed planes for each face. Note that the exact representations of each plane (in the form of plane equations) are thus kept in this process.

7. **Add the structural types** The Nef polyhedra representing structural types (e.g. walls, slabs and beams) are incrementally added to a model, making sure that a new Nef polyhedron does not intersect the previously added ones. This is done using a Boolean set difference with a Nef polyhedron containing all previously added polyhedra, which is then regularised.

8. **Remove the opening types** The Nef polyhedra representing openings are carved out from the structural types by a Boolean set difference whose result is then regularised. They are also carved out from the Nef polyhedron representing the entire model.

9. **Add the fixture types** The Nef polyhedra representing fixtures (e.g. windows, doors, railings and frames) are incrementally added to the model in the same manner as the structural types. The fixtures will often fit into the openings carved out in the previous step.

The output of these steps is thus a list of Nef polyhedra representing each object using exact arithmetic. As these are regularised, they are known to contain their boundary. Because of this, the common faces of a pair of adjacent polyhedra are easy to obtain through the computation of their Boolean set intersection, which can be implemented quickly. These faces can therefore be used to easily compute a topological representation of the 3D space subdivision.

This 3D repair method was implemented with the help of several libraries: IfcOpenShell is used to parse the IFC file, Open CASCADE is used to triangulate implicit representations and to transform the objects' coordinates to a global coordinate system, CGAL `Polyhedron_3` is used to construct a half-edge representation of every polyhedron, and CGAL `Nef_polyhedron_3` is used to store every Nef polyhedron and to perform the Boolean set operations between them.

10.4 Dimension-independent validity criteria

The previous sections have expanded on the criteria that define what is a valid object or space partition of objects in 2D and 3D. They also described some methods that can be used to make real-world data comply with these criteria, enabling the data to be used for more applications. As this thesis aims at utilising real-world higher-dimensional data, it is important to also consider what criteria can be used to define validity in higher-dimensional data.

The standards for geographic information in 2D and 3D described previously (Simple Features [OGC, 2011], GML [OGC, 2012] and ISO 19107 [ISO, 2005a]) are in theory limited to 2D and 3D. Concretely, the ISO 19107 standard explicitly states that '*this International Standard is restricted to at most three dimensions*'. However, all of these standards are easily extensible to higher dimensions. This would mostly involve the addition of new classes and corresponding definitions. However, the standards do contain minor hard-coded assumptions that are only valid for the 2D and 3D cases, such as how ISO 19107 and GML consider orientable curves and surfaces, but not orientable solids (Figure 3.18 on page 43).

This section therefore defines higher-dimensional objects in a manner that is (mostly) harmonious with the standards used in the GIS world. An n-cell can be thus represented by the set of $(n-1)$-cells in its (outer) boundary, using a similar mechanism as how other boundaries are represented in the ISO 19107 standard, which was shown previously in Figure 10.12.

Following the terminology used in the standard and as shown in Figure 10.19, such an extension of the would mainly entail a `GM_OrientableGeometricPrimitive` with a `dimension` at-

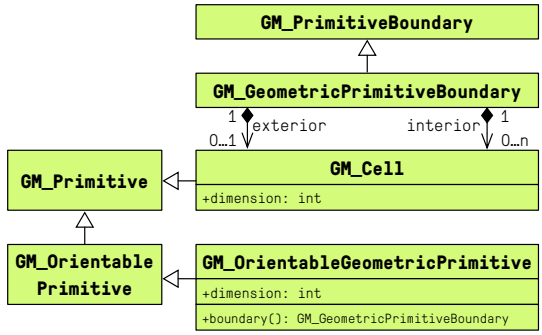


Figure 10.19: A dimension-independent definition of a cell in a harmonised manner with other classes in the ISO 19107 standard [ISO, 2005a].

tribute, which would set to n. This class would be analogous to `GM_OrientableCurve` for `dimension` 1, `GM_OrientableSurface` for `dimension` 2 and a newly created `GM_OrientableSolid` for `dimension` 3, which would be a subclass of `GM_OrientablePrimitive`. The `GM_OrientableGeometricPrimitive` would be bounded by a `GM_GeometricPrimitiveBoundary`, which would be linked to aggregations of $(n-1)$-dimensional instances of a newly created `GM_Cell` (with their `dimension` attribute set to $n-1$). This `GM_Cell` class would be analogous to `GM_Point` for `dimension` 0, `GM_Curve` for `dimension` 1, `GM_Ring` for `dimension` 2, and `GM_Shell` for `dimension` 3. Each of the instances of `GM_Cell` bounding a `GM_OrientablePrimitive` would represent either the outer boundary of the geometric primitive (if any), or one of any number of inner boundaries representing n-dimensional holes. This extension of the standard would seem to follow most in the spirit of ISO 19107.

However, other alternative extensions could be considered. As `GM_Curve`, `GM_Surface`, and `GM_Solid` would essentially be special cases of `GM_Cell`, all of the former could be seen as redundant and eliminated. However, the standard already contains many specialisations that are somewhat redundant but that cover common use cases in geographic information, such as `GM_Triangle` and `GM_Tin`. Another possibility would be considering `GM_Curve`, `GM_Surface`, and `GM_Solid` as subclasses of `GM_Cell` or substituting the abstract `GM_Primitive` for a non-abstract `GM_Cell`, but this would involve a major change in the standard and seems to run counter to the preferred use of abstract top classes in the standard.

The definition of an `GM_OrientableGeometricPrimitive` as explained above also lends itself to the definition of sets of disjoint cells (akin to the `Multi…` classes in the standard) and cell complexes (akin to the `Composite…` classes in the standard), which could also be handled in the same manner as in the ISO 19107 standard. As shown in Figure 10.20, the standard already defines composite curves, surfaces and solids, which are equivalent to 1-, 2- and 3-dimensional cell complexes. A `GM_CompositeCurve` is '*a list of orientable curves (`GM_OrientableCurve`) agreeing in orientation in a man-*

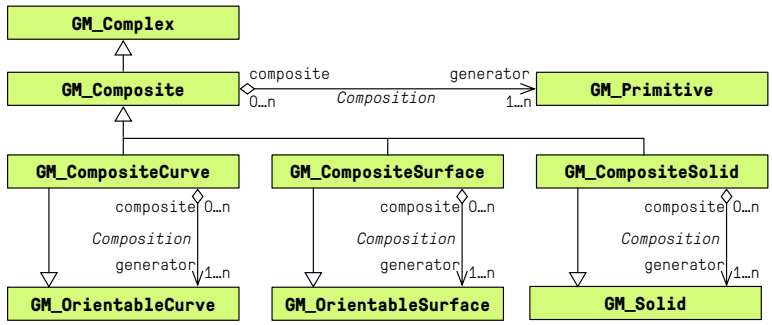


Figure 10.20: The cell complexes of dimension 1, 2 and 3 are respectively defined in the ISO 19107 standard [ISO, 2005a, §6.6.3] as the classes `GM_CompositeCurve`, `GM_CompositeSurface` and `GM_CompositeSolid`.

ner such that each curve (except the first) begins where the previous one ends.', a `GM_CompositeSurface` is '*a collection of oriented surfaces that join in pairs on common boundary curves*', and a `GM_CompositeSolid` is '*a set of solids that join in pairs on common boundary surfaces*'.

A similarly defined `GM_CompositeGeometricPrimitive`, shown in Figure 10.21 which should contain the `dimension` as a parameter, would thus be equivalent to a representation of a space partition of any dimension that allows objects with holes. It could be defined as 'a set of n-dimensional orientable geometric primitives (`GM_OrientableGeometricPrimitive`) that join in pairs on common $(n-1)$-dimensional boundary geometric primitives'. Note that this implies that the primitives combinatorially form an n-quasi-manifold, although geometrically they might not do so due to the presence of holes.

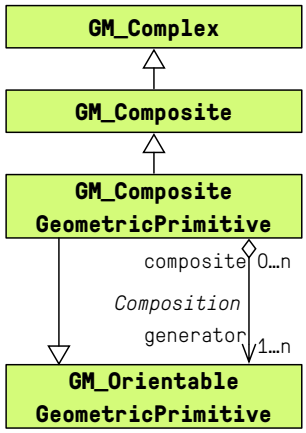


Figure 10.21: A definition of an n-dimensional space subdivision in a harmonised manner with other classes in the ISO 19107 standard [ISO, 2005a].

Following the validity criteria previously described in §10.2 and §10.3, it is possible to define additional validity criteria for an n-dimensional geometric primitive, which would serve to specify the conditions upon which its bounding cells may interact. These would be as follows:

- The bounding n-cells of an n-dimensional geometric primitive do not cross, but they might intersect only at a cell of dimension $n-1$ or lower.
- The interior of every geometric primitive is a connected point set.
- Each interior n-cell creates a new point set in $\mathbb{R}^n$ that is disconnected from the exterior.

Meanwhile, an n-dimensional space subdivision should consist of a set of n-dimensional geometric primitives that are mutually exclusive and exhaustively partition an extent, itself a well-defined subset of $\mathbb{R}^n$. As with the definitions of a planar partition and 3D space subdivision, this implies that there should be no overlapping primitives, and no gaps between them unless these gaps are considered to be outside the extent.

Conclusions and future work | 11

There are a great number of possible representations for spatial information, each of which with its own benefits and drawbacks and only some of which have been discussed in this thesis. From an engineering point of view, choosing an appropriate representation for a given system or task is a fundamental issue, as this choice cascades down to almost every engineering decision and indirectly affects a GIS program's every functionality. Among other aspects, it affects the type of objects that can be efficiently stored and the operations that can be easily performed on them, and it also has important computational consequences in terms of both memory and processing time.

The 2D representations that dominate the GIS world work well for 2D datasets and problems that are essentially two-dimensional, but many problems arise when they are adapted to model 3D, spatiotemporal and multi-scale geographic information. Most research in GIS is devoted to improving these adaptations, as well as to developing new methods that build on them to solve problems both old and new.

This thesis pushes GIS research in a different direction, starting from the assumption that many issues in GIS can probably be better solved by using a new, fundamentally different modelling approach—modelling both spatial and non-spatial characteristics as dimensions in the geometric sense, thus using higher-dimensional representations to create, manipulate and visualise geographic information. Accordingly, this thesis' main research objective was to **realise the fundamental aspects of a higher-dimensional Geographic Information System**, and therefore focusing on the development of higher-dimensional representations and methods for GIS.

This concluding chapter starts with a short outlook on higher-dimensional GIS in §11.1, describing concisely when and where it makes sense to use higher-dimensional models. Afterwards, §11.2 describes in detail the lessons learned by pursuing this thesis' research objective. Finally, §11.3 lists the main contributions of this thesis, and §11.4 discusses the topics that I think would be most useful for future research on this topic.

11.1 An outlook on higher-dimensional GIS

As this thesis has shown, there are many potential advantages to the use of higher-dimensional modelling in GIS. This approach provides *a simple and consistent way to store geometry, attributes and topological relationships between objects of any dimension*. This generic technique can be easily extended to handle other non-spatial characteristics, enabling better data management and more powerful operations. At the same time, higher-dimensional representations are undoubtedly memory-intensive and often hard to work with, both due to their level of abstraction and the unintuitiveness of working with dimensions higher than three. Admittedly, *current GIS use cases do not easily justify the higher-dimensional approach*.

However, a far stronger case for higher-dimensional representations emerges when considering what sort of *new tools could be developed using this type of representations*, both in GIS and in related fields. For instance, 3D modelling software could consider time-varying topology, much as Dalstein et al. [2015] do for 2D vector drawings. 4D topology (as 3D+time) could then be used to automatically generate smooth transitions for animations.

Similarly, CAD tools could use 4D topology to model buildings at different scales and timeframes automatically, keeping track of all relationships between objects and providing immediate user input during interactive editing. For example, a program could display how different changes affect construction time, the size of the model at predefined LODs and when certain safety constraints were violated.

At a lower-level, 4D geometric modelling operators could be developed to operate directly on 4D primitives, such as splitting and merging 4-cells. These could also be used intuitively in an interactive environment, allowing for instance to change a building's configuration by adding and removing walls while always ensuring that the representation remains a valid 4D space partition.

Considering that current GIS are very often used to manage large heterogeneous datasets and keep them up to date, a future system using a higher-dimensional underlying representation could be used to enforce certain validity constraints at the data structure level, such as avoiding 4D intersections or preserving a certain degree of continuity at LOD transitions.

Is the higher-dimensional approach worthwhile?

In short, yes, but only given certain conditions.

Higher-dimensional modelling is advantageous, but only when the added functionality that will be built with them—as compared to the simpler

and more compact 2D/3D models—justifies it, such as when queries across space and time can be implemented as higher-dimensional geometric/topological operations. As discussed in §4.2.1, higher-dimensional modelling makes sense only when the characteristics depicted as dimensions are *parametrisable and independent from each other*, as is the case for space/time/scale, and where objects occur along a dimension as intervals, not as discrete points (which can be easily stored as attributes). Based on current hardware, software and the typical GIS datasets, there is another important practical requirement: the manageable number of dimensions is limited to 6–8. Finally, it is also worth noting that higher-dimensional models are not incompatible with standard 2D/3D data structures and methods—the best tool for the job can be chosen depending on the need at hand and both approaches can be combined.

11.2 Lessons learned

Current representations are not suitable in higher dimensions

The mathematical foundations of spatial data modelling (Chapter 2) are defined in a dimension-independent manner, including all the basic tenets of geometry and topology. This dimensional independence is also true for all spatial data models or representation schemes (Chapter 3)—at least when they are analysed at a high level—but is generally not preserved when they are implemented into more concrete data structures. Most 2D and 3D data structures in GIS thus only encode a few chosen geometric and topological properties (e.g. the coordinates of each point and the adjacencies between polygons), which are often defined with a formulation that is different per dimension.

This modelling approach can make for custom structures that are compact and efficient when used exactly as intended (i.e. for a particular class of objects of a given dimension), but it can limit functionality or introduce inefficiencies when the data structures are adapted to be used under a different set of circumstances. Case in point, the typical data structures of 2D GIS are frequently used with minimal changes for 3D, spatiotemporal and multi-scale GIS. This results various problems, such as inefficient representations (e.g. due to duplicate elements), an inability to represent common 3D objects (e.g. those with a non-2-manifold boundary), difficulties in expressing 3D topological relationships (e.g. adjacencies between solids), and the widespread availability of invalid datasets (e.g. 3D models that do not formally enclose any space), among others.

Higher-dimensional modelling as a solution An alternative to the use of ad hoc adaptations to 2D data structures is to model both spatial and non-spatial characteristics as dimensions in the geometric sense (Chapter 4). While this approach can be memory-intensive, it provides a generic solution that can be applied to the representation of nD space, time, scale and any other parametrisable characteristics.

A tuple of n parametrisable spatial and non-spatial characteristics can thus define a coordinate system in $\mathbb{R}^n$, and lower-dimensional 0D–3D objects existing across these characteristics can thus be modelled as higher-dimensional 0D–nD objects embedded in higher-dimensional space. As GIS objects are usually non-overlapping[167], they should form an nD space partition and can thus be represented using nD topological data structures, reducing the total number of elements and ensuring that it is easy to navigate between the objects. However, even when the objects do not form a space partition, the objects themselves can be partitioned by using an intermediate representation where the original objects are transformed into a set of non-overlapping regions, such that each of these regions represents *a set of the original objects* [Rossignac and O'Connor, 1989].

167: At least in theory.

nD space partitions are also ideal in a practical sense, as they simplify many of the operations that can be defined with them, including simple point-in-polytope queries and all of the constructions methods presented in Chapters 6–8. nD point clouds, while outside the scope of this thesis, are a good complement to nD space partitions, as they are close to data as it is acquired and are relatively easy to store and manipulate.

The most promising higher-dimensional representations Spatial data structures often consist of two aspects: 1. a combinatorial part, which consists of a set of primitives and some topological relationships between them, and 2. an embedding that links these primitives to their geometry and attributes [Lienhardt, 1994]. As argued in this thesis, the knowledge of higher-dimensional topological relationships in a data structure is the main aspect that differentiates higher-dimensional data structures from 2D/3D ones.

Some data structures that are frequently used in 2D/3D GIS have straightforward extensions to higher dimensions, such as nD *rasters* and *hierarchies of trees*. These use similar structures and algorithms as their 2D/3D counterparts and are therefore easy to understand and implement.

Other data structures implement the models of an nD simplicial complex or an nD cell complex. As the simplices in

a simplicial complex have a known number of adjacent simplices and bounding facets, they are most efficiently stored using simplex-based data structures. Meanwhile, cell complexes can be easily stored using incidence graphs and related structures. However, Nef polyhedra [Bieri and Nef, 1988] are probably the most promising representation for an nD cell complex, as they provide a good base to develop Boolean set operations, enabling a wide range of geometric operations.

Ordered topological models such as the cell-tuple [Brisson, 1993] and generalised/combinatorial maps [Lienhardt, 1994] also deserve a special mention. By combining the strong algebra and easy navigation of a simplicial complex with the easy representation of a cell complex, they provide the most important benefits of both. They are rather memory-intensive, but it is important to note that can still be more compact than a non-topological approach (Chapter 5).

Nevertheless, non-topological higher-dimensional representations do have a clear role to play as exchange formats, much as is the case for those based around Simple Features in 2D [OGC, 2011], and CityGML [Gröger et al., 2012] and IFC[168] in 3D.

168: http://www.buildingsmart-tech.org/specifications/ifc-releases

Three construction methods for higher-dimensional objects Creating computer representations of higher-dimensional objects can be complex. Common construction methods used in 2D and 3D, such as directly manipulating combinatorial primitives, or using primitive-level construction operations (e.g. Euler operators [Mäntylä, 1988]), rely on our intuition of 2D/3D geometry, and thus do not work well in higher dimensions. It is therefore all too easy to create invalid objects, which then cannot be easily interpreted or fixed—a problem that is already exceedingly apparent in most 3D datasets. As an alternative to the use of simple operations on combinatorial primitives, this thesis thus proposed three higher-level methods, all of which are relatively easy to use and attempt to create valid output.

Method I. constructing objects using nD extrusion Extrusion as used in GIS has a natural extension into a dimension-independent formulation (Chapter 6). Starting from an $(n-1)$-dimensional space partition as an $(n-1)$-dimensional cell complex and a set of intervals per cell, it is possible to extrude them to create an n-dimensional cell complex. It is the easiest method to load existing 2D or 3D data into a higher-dimensional structure, representing a set of cells that exist along a given dimension, such as a length of time or a range of scales. It is also easy to guarantee that the output cell complex is valid and can be used as a base for further op-

erations, such as dimension-independent generalisation algorithms.

The extrusion algorithm developed in this thesis works on the basis of a generalised map representation of the cell complex and is relatively fast, with a worst case complexity of $O(ndr)$ in the main algorithm, where n is the extrusion dimension, d is the total number of darts in the input map and r is the total number of intervals in the input, but offers better complexity in practice. It is also memory-efficient, as only three layers of darts (of the size of the input cell complex) need to be kept in memory at the same time.

Method II. constructing nD objects incrementally Based on the Jordan-Brouwer separation theorem [Lebesgue, 1911; Brouwer, 1911], it is known that an i-cell can be described based on a set of its bounding $(i-1)$-cells (Chapter 7). Since individual $(i-1)$-cells are easier to describe than the i-cell, this can be used to subdivide a complex representation problem into a set of simpler, more intuitive ones. This method can be incrementally applied to construct cell complexes of any dimension, starting from a set of vertices in $\mathbb{R}^n$ defined by a n-tuple of their coordinates, and continuing with cells of increasing dimension—creating edges from vertices, faces from vertices or edges, volumes from faces and so on.

The incremental construction algorithm developed in this thesis solves this problem in a practical setting by computing the topological relationships connecting the bounding $(i-1)$-cells. It uses indices on the lexicographically smallest vertex of every cell per dimension, as well as an added index using the lexicographically smallest vertex of the ridges around the bounding facets of the cell that is being built. It generates an i-cell in $O(d^2)$ in the worst case, with d the total number of darts in the cell. However, it fares markedly better in real-world datasets, as cells do not generally share the same lexicographically smallest vertex. By checking all matching ridges within a cell's facets, the algorithm can optionally verify that the cell being constructed forms a combinatorially valid quasi-manifold, avoiding the construction of invalid configurations.

Method III. linking 3D models at different LODs into a 4D model As an example high-level higher-dimensional object construction method, a 4D model can be constructed from a series of different 3D models at different LODs (Chapter 8). The method presented in this thesis consists of three steps: identifying corresponding elements in different LODs, deciding how these should be connected according to a linking scheme, and finally linking relevant 3-cells into 4-cells. Different linking schemes yield 4D models having different properties, such as objects that suddenly appear and disappear, gradually

change in size or morph into different objects along the fourth dimension.

By modelling the LOD as a dimension, the correspondences between equivalent objects across LODs become geometric primitives, making it possible to perform geometric operations with them (e.g. extracting an intermediate LOD for visualisation purposes) or to attach attributes to them (e.g. general semantics or the meaning of these correspondences), just as is done to other geometric primitives. These topological relationships and correspondences can then be used for multiple applications, such as updating and maintaining series of 3D models at different LODs, or testing the consistency of multi-LOD models (e.g. by using the validity checks in Gröger and Plümer [2011a]).

Extracting 2D/3D subsets from an nD model The process to obtain a lower-dimensional subset of a higher-dimensional dataset can be regarded as a function that maps a subset of $\mathbb{R}^n$ to a subset of $\mathbb{R}^m$, $m < n$, which is obtained by cutting through the dataset in a geometrically meaningful way (Chapter 10). Broadly, this process consists of two steps: (i) selecting a subset of the objects in the model and (ii) projecting this subset to a lower dimension. Both of these steps can vary substantially. Selecting a subset of the objects can be as simple as obtaining those within a axis-aligned bounding box, or can be as complex as a Boolean set intersection operation, such as for the computation of cross-sections. Meanwhile, there are a wide variety of transformations that apply different projections with different properties, such as the n-dimensional to $(n-1)$-dimensional orthographic and perspective projections derived in this thesis and the $\mathbb{R}^n$ to S^{n-1} spherical projection used in its cover.

Methods to create valid objects and space partitions in 2D and 3D Most algorithms described in computational geometry and GIS assume that their input datasets are flawless and they are processable using real numbers. However, invalid datasets are widespread in GIS (Chapter 10), and they are represented and processed using limited-precision arithmetic (Appendix A).

In order to cope with 2D invalid datasets, this thesis further developed methods to create valid polygons and planar partitions using a constrained triangulation of the input. These were based on the work done in Arroyo Ohori [2010], improving the reconstruction algorithm, fixing edge cases, implementing an odd-even constraint counting mechanism and improving the quality of the implementation. Similarly, a method to repair 3D objects and space subdivisions was developed by snapping together lower-dimensional primitives and removing overlaps using Boolean set operations on Nef

polyhedra [Bieri and Nef, 1988; Hachenberger, 2006]. These methods were used in this thesis in order to use real-world datasets in practice, such as when applying the construction algorithms.

11.3 Contributions

The main contribution of this thesis is the realisation of the fundamental aspects of a higher-dimensional Geographic Information System. By approaching this problem in a practical manner, many of the technical issues of its development were investigated, including an analysis of its possible internal (in-memory) and external (exchange format) representations, the development of basic algorithms for object construction and visualisation, and the development of GIS data repair tools for 2D and 3D datasets. By taking a model that was previously only described at a conceptual level [van Oosterom and Stoter, 2010] and realising it, it is now possible to more fully evaluate the consequences of this higher-dimensional approach.

In more concrete terms, there were several smaller contributions that were necessary to be able to achieve this realisation. The most significant ones are:

Survey and analysis of higher-dimensional models and structures I conducted a survey of all of the main data models and data structures used in GIS, geometric modelling and related fields, considering 2D, 3D and nD data structures. These were analysed in terms of their feasibility for the higher-dimensional modelling of geographic information, including how they could handle different geometry classes, topology and attributes, either in their current form or through modifications, as well as their ease of implementation in practice.

Three construction methods I developed three easy-to-use construction methods for objects of any dimension. The two methods lower-level methods—extrusion and incremental construction—were implemented using CGAL Combinatorial Maps and tested using real-world datasets. The third method has been tested with a few synthetic datasets, but more work is necessary to fully realise it and automate it. All of the implementations were made available publicly under an open source licence.

Higher-dimensional real-world models As part of this thesis, I created higher-dimensional models from real-world 2D and 3D datasets. To the best of my knowledge, these are the only

datasets consisting of realistic higher-dimensional objects in a GIS setting.

Combinatorial map reversal As part of the development of the incremental construction operation, I developed additional functions that were added to CGAL Combinatorial Maps after being approved by the CGAL Editorial Board. These functions involved reversing the orientation of a combinatorial map of any dimension.

Simple formulation of nD to $(n-1)$D projections I developed intuitive formulations of n-dimensional to $(n-1)$-dimensional orthographic and perspective projections. While other formulations exist, my formulations based on normal vectors are in my opinion the easiest to understand and manipulate, at least for a GIS audience.

Repair methods tools I developed methods to automatically repair 2D polygons and planar partitions, as well as 3D polyhedra and space partitions. The 2D methods were also released publicly under an open source licence as `prepair` and `pprepair`. The 3D methods will also be released publicly after further improvements are made, together with more thorough testing and the addition of basic documentation.

11.4 Future work

As this thesis challenges many of the assumptions underpinning current GIS, there are many potential lines of research that can be formulated for higher-dimensional GIS and higher-dimensional modelling in general. Most algorithms for 2D/3D GIS have a dimension-dependent formulation and would result in open problems in an nD context.

However, while these are worthy of attention, I would like to focus on what are still significant gaps in knowledge for the implementation of a higher-dimensional GIS and that could not be solved in this thesis' timeframe. While some of these research topics are not within the main subject matter of GIS research, they are what I consider to be the key steps for a more complete implementation of a working system. They are:

Low-level linking algorithms The linking schemes from Chapter 8 have only been described in terms of high-level algorithms. These should be further developed by finding adequate low-level algorithms to identify matching elements using customisable constraints. For instance, it is reasonable to attempt to minimise a certain distance function between two models (e.g. Earth mover's distance), but it is also important to do so

in a manner that preserves the topological relationships between the objects. The special treatment of holes of different dimensions should also be investigated, together with adequate methods to ensure that holes are linked correctly between themselves and to other primitives.

High-level construction algorithms The three object construction methods described in this thesis operate mostly on lower-dimensional primitives and consequently cover only a few use cases. There is a need to develop intuitive methods that operate directly on higher-dimensional primitives, which should preferably be usable in an interactive environment. Note however that this does not mean that the end user would be viewing the higher-dimensional model directly, as it could be shown in simplified form or as a 2D or 3D representative subset.

***n*D constrained triangulator** There are high-quality robust implementations of constrained Delaunay triangulations in 2D [Shewchuk, 1996a] and 3D [Si and Gärtner, 2005], as well as good descriptions of a constrained Delaunay triangulation in *n*D [Shewchuk, 2008]. However, in order to realise a higher-dimensional GIS based on a simplicial complex model, it is necessary to have a robust *n*D constrained triangulator which should be preferably Delaunay.

Hyperspherical projective geometry kernel I deemed Nef polyhedra one of the most promising models for a higher-dimensional GIS, but so far they have only been implemented in 2D and 3D. In order to implement *n*-dimensional Nef polyhedra, it is necessary to develop a hyperspherical projective geometry kernel. This kernel would then be used to compute the local $(n-1)$-dimensional pyramids around every vertex. While the projective mathematics for this are relatively simple, it is a complex engineering problem to implement it robustly.

***n*D Boolean set operations** Using the above mentioned hyperspherical projective kernel or another method, it is necessary to develop robust algorithms to compute *n*D Boolean set operations. For instance, an *n*D Nef polyhedra implementation using recursive boundary definitions could compute these operations at the local pyramid level. *n*D Boolean set operations would be an excellent base for most geometric operations in a higher-dimensional GIS.

***n*D to/from 2D and 3D projective kernel** Many operations that are required for object manipulation in *n*D are actually well-defined operations applied to 2D and 3D objects that are merely embedded in *n*D. A robust kernel that could handle on-the-fly conversions of *n*D geometries to 2D and 3D without loss

of precision would enable many of these operations to be applied. Relatedly, CGAL currently has simple kernels that apply 3D to 2D orthographic projections using the planes defined by the xy, yz or zx planes. These are useful as they can be wrapped around the basic 2D kernels (i.e. $\mathbb{R}^2$ with floating-point, interval or exact representations) using the traits programming paradigm available throughout CGAL, and so they can be used to apply 2D operations to 2D objects that are embedded in 3D. However, these kernels do not handle the conversions from 2D back to 3D automatically nor are easily extensible to higher dimensions.

Visualisation of higher-dimensional geographic information Chapter 9 described how data selection and projection methods can work in arbitrary dimensions, while §5.2 described the nD mathematics behind the basic manipulation operations for higher-dimensional objects. However, there are significant issues to tackle in order to create a useful visualiser for higher-dimensional datasets. Namely, there are significant hurdles in user interaction, dealing with large datasets, computing higher-dimensional cross-sections and the definition of useful visual cues in higher dimensions. The simple implementation used for the cover of this thesis make several hard-coded assumptions for the dataset that was used and is far from optimal, but it will be published together with the rest of the open source tools developed for this thesis once it is improved to handle more general input in the form of nD linear cell complexes.

2D/3D/nD repair methods with quality guarantees The data repair methods described in Chapter 10 are able to recover from most simple invalid 2D and 3D configurations. However, they are not easily extensible to higher dimensions due to the lack of a robust nD constrained triangulator (see above), they do not provide quality guarantees in their output, and can be numerically unstable when there are many nearly coplanar planes. In order to develop robust systems, it is highly desirable to be able to specify a robustness criterion (e.g. a minimum distance between vertices or ensuring that the geometries are not collapsed/flipped in a floating-point representation), which is guaranteed by a data repair algorithm. Additional geometric constraints could also be implemented, such as guaranteeing that coplanar planes stay coplanar after repair.

Higher-dimensional modification operations By necessity, this thesis focused on operations for object creation in order to generate initial higher-dimensional datasets. Now that it is possible to generate them, it is important to think of intuitive higher-dimensional object modification operations. For in-

stance, how can a 4-cell split operation be intuitively defined, or how can collapsed geometries (e.g. from the extruded and collapsed models in §6.5) be processed to remove combinatorial elements. Ideally, these operations should also be intuitively usable in an interactive environment, such as a geometric modeller.

Real-world 4D spatiotemporal datasets Using timestamped 3D volumetric datasets, it should be possible to create true 4D datasets using spatiotemporal information. However, obtaining reasonably clean volumetric datasets is nearly impossible at this point. Every dataset that was found during this project had only surfaces embedded in 3D, had severe validity problems up to the point that it would require substantial manual work to fix, or was missing the temporal information. Exporting the temporal information that is present in some closed commercial formats is also an issue, as doing do in a naïve way generally means losing the links to the timestamps' corresponding geometries.

Implementing higher-dimensional representations and operations

A

The higher-dimensional representations described in Part I and the operations described in Part II can be difficult to implement, especially when we expect these implementations to be fast, robust, generic, compact and dimension-independent. This is true even when the basic ideas and algorithms are provided, as has been done in this thesis.

For the sake of full reproducibility, this chapter shows some of the key implementation details that are used to efficiently implement the representations and operations described in this thesis. §A.1 lists the main libraries that were used and how they were used. §A.2 explains the main techniques that are used to perform arithmetic and geometric operations robustly. Finally, §A.3 describes the traits programming technique and its use in CGAL and this thesis to produce dimension-independent efficient implementations.

A.1 Main libraries used within this thesis

CGAL[170] (the Computational Geometry Algorithms Library) contains a wide variety of 2D/3D/nD data structures and computational geometry algorithms. Some of its basic packages are directly used within this thesis to store and manipulate numbers and basic shapes, namely: Algebraic Foundations, Number Types, 2D and 3D Linear Geometry Kernel, and dD Geometry Kernel. Most significantly, the packages Combinatorial Maps and Linear Cell Complex are used in most of the implementations described in the previous chapters. Finally, other packages are used as temporary data structures and to process and clean input data: 3D Polyhedral Surface, Halfedge Data Structures, 3D Boolean Operations on Nef Polyhedra, 2D Triangulation, and Principal Component Analysis. A few other packages are used as dependencies of the aforementioned packages.

170: http://www.cgal.org

GDAL[171] (the Geospatial Data Abstraction Library) reads and writes commonly used GIS file formats. Within this thesis, its OGR

171: http://www.gdal.org

vector module is mainly used to read and write polygons described as well-known text [OGC, 2011], or in Esri Shapefile [ESRI, 1998] and FileGDB files.

172: http://www.ifcopenshell.org

IfcOpenShell[172] is a library that is able to read and write IFC files [ISO, 2013]. It internally uses the Open CASCADE geometry types, including to convert implicit geometries (e.g. those built using constructive-solid geometry or sweeps as in Figure 3.19) into explicit ones that can be stored using boundary representation, or to create meshes of a given degree of accuracy from curved surfaces.

173: http://www.opencascade.org

Open CASCADE[173] is a library that is able to manipulate geometric representations in CAD applications. In theory, it supports complex geometric operations between implicit geometries, including Boolean set operations with 3D point sets. However, in practice it performs poorly with GIS data, often failing due to numerical errors or imperfect data.

A.2 Geometric operations using computer arithmetic

Theoretical descriptions of geometric objects and geometric algorithms generally start from the notions of the Euclidean space $\mathbb{R}^n$, in which the coordinates of a point can be described precisely using real numbers ($\mathbb{R}$). However, as real numbers cannot be represented on (digital) computers, implementations usually opt for a combination of *integer numbers* to represent whole numbers of known precision that are known to fall within a given interval and *floating-point numbers* in all other cases. While integers can be precisely expressed as a sequence of binary digits of a given length, floating-point numbers often cannot. The latter are therefore usually[174] expressed using binary numbers with a predefined number of bits.

174: This is only the most common representation among those provided by the much broader IEEE 754 standard [IEEE, 2008], which provides for decimal numbers as well as special values for $\pm\infty$ and NaN (not a number), among other features.

Floating-point numbers can represent a wide range of values and work well in many instances. However, arithmetic performed using floating-point numbers needs special care, as it often leads to a loss of precision [Goldberg, 1991]. While this is a problem for all kinds of algorithms [Hoffmann, 1988], geometric operations are particularly vulnerable as they often rely on getting a correct result for a large number of predicates, which can fail when dealing with edge cases [Kettner et al., 2008], such as almost collinear or coplanar points.

Many alternatives have been developed to deal with various limitations of integer and floating-point numbers. Among these, the ones described below are those that have been used for the implementations related to this thesis. *Multiple precision arithmetic* is a generic solution that can achieve an arbitrary level of precision by

using numbers with a user-definable number of digits. It is widely implemented in libraries such as GMP[175] and MPFR[176].

175: https://gmplib.org/

176: http://www.mpfr.org

Simple arithmetic operations can be computed precisely by using *rational arithmetic*, where a number is stored as a ratio of two other numbers, most commonly integers. In a geometric context, this type of representation is often used in the form of homogeneous coordinates, where a single number is used as a common denominator for all of the coordinates. This common denominator can be used to represent special values, such as a point at infinity by setting it to zero.

In *interval arithmetic* numbers are substituted with intervals. When these are used to represent the error bounds of an operation[177], it is possible to compute arithmetic operations with provably correct results [Ratschek and Rokne, 1988, Ch. 2], such as those provided by the MPFI library[178]. Unfortunately, while this setup using multiple precision interval arithmetic can be applied to most problems with relative ease, it is also very slow. For instance, Held and Mann [2011] reports a factor of 70 for the computation of Voronoi diagrams.

177: combined with correct rounding in the case of floating-point numbers

178: https://perso.ens-lyon.fr/nathalie.revol/software.html

It is possible to go around this problem by fine-tuning a multiple-precision approach to a specific problem. Notably, this is done with very good results for a few geometric predicates by Shewchuk [1997], and the simulation of simplicity paradigm advocated by Edelsbrunner and Mücke [1990]. A more generic and easier to implement solution is provided by the lazy evaluation scheme used in CGAL [Pion and Fabri, 2011], which is based on interval arithmetic and is the one used in this thesis. In it, the computationally expensive multiple precision operations are only computed when floating-point precision is not sufficient. As these cases are important to get correct results, but also relatively rare, it significantly improves the performance of most operations while maintaining their correctness.

A.3 Efficient and flexible dimension-independent programming

Previously, §4.2.1 discussed how higher-dimensional representations have large sizes and methods using them have high computational complexities, which often increase exponentially on the dimension. However, there are also practical obstacles that make it difficult to implement dimension-independent structures and methods efficiently, especially when these have to be used in a generic setting such as in GIS, where varied objects of different dimensions need to be dynamically created and modified, as well as appended with possibly multiple attributes of various types.

One of these obstacles is the need to allocate and use structures that are dimension- and data-independent, and therefore flexible enough to cover all the aforementioned use cases, but at the same time remain compact and allow their contents to be accessed efficiently. These structures can range from simple ones that can be handled by standard data types and containers, to more complex ones that need to be dynamically defined. For instance, some simple types are directly dependent on the dimension, such as n-tuples storing the coordinates of a point in $\mathbb{R}^n$, and can thus be stored as arrays or vectors.

At the opposite end, consider the sets of extrusion intervals that were associated to each cell in Chapter 6, where an unknown number of cells need to be each associated with an unknown number of intervals. As the number of intervals per cell is not known, it is not possible to store the intervals in a fixed-length structure that is integrated into the embedding structure of each cell. Also, while it is possible to directly link a cell to its set of intervals from its embedding structure, these intervals are only temporarily needed, so allocating space for the intervals directly in the embedding structure is wasteful at all other times and thus difficult to justify. The end result was that the intervals per cell were kept in an external structure, where a `map` linked a cell embedding to a `set` of intervals. As Table A.1 shows, this means that accessing a given interval of a given cell—an operation that is performed a very large number of times—, takes logarithmic rather than constant time, significantly slowing the extrusion algorithm in practice.

Table A.1: The typical computational complexity of accessing a given element in common C++ containers [ISO, 2015]

structure	complexity
hard-coded	$O(1)$
array/vector	$O(1)$
map/set	$O(\log n)$
list	$O(n)$

A possible solution to the aforementioned problems is based on *template meta-programming*. Template meta-programming is a technique that uses templates to generate certain data structures or perform certain computations during the compilation of a program rather than during its execution. Templates are normally used as a way to support generic programming, enabling the creation of functions that can deal with different data types indistinctly. A template might thus be instantiated with the dimension of an object or a particular attribute type, thus generating a data structure of the appropriate size and disposition whose members can be accessed in constant time. Figure A.1 shows a slightly more complex example, where a template can be used to convert a string into any number type, which is used in this thesis to parse numbers from various types of files (e.g. coordinates and identifiers).

However, apart from their use in generic programming, templates can also be used to create complex dimension-dependent structures, such as through the use of the traits programming technique [Myers, 1995] used in CGAL, which exploits C++'s `typedef` declarations to create custom dependent types. As an example, Figure A.2 shows how the implementation of the extrusion algorithm defines a combinatorial map that is one dimension higher than the input, which

```
template <typename T>
T string_to_number (const std::string &text, T def_value) {
    std::stringstream ss;
    for (std::string::const_iterator i = text.begin(); i != text.end(); ++i)
        if (isdigit(*i) || *i=='e' || *i=='-' || *i=='+' || *i=='.') ss << *i;
    T result;
    return ss >> result ? result : def_value;
}
```

Figure A.1: Using C++ templates to convert a string containing a number into any number type T, including scientific notation. Adapted from http://www.cplusplus.com/forum/articles/9645/.

```
1  template <unsigned int unextruded_dimension>
2  class Linear_cell_complex_extruder_with_range {
3  public:
4   typedef typename Linear_cell_complex_with_ids<unextruded_dimension>::type Lower_dimensional_cell_complex;
5   typedef typename Linear_cell_complex_with_ids<unextruded_dimension+1>::type Higher_dimensional_cell_complex;
6   typedef Linear_cell_complex_extruder_with_range<unextruded_dimension> Self;
7   typedef typename Lower_dimensional_cell_complex::FT FT;
8
9   typedef typename Extruded_embeddings_with_range_of_dimension_and_lower<Self>::type Extruded_embeddings_with_range;
10  typedef Extrusion_ranges_tuple_per_dimension<Lower_dimensional_cell_complex> Extrusion_ranges;
11
12 private:
13  Extrusion_ranges extrusion_ranges;
14  std::set<typename Lower_dimensional_cell_complex::FT> all_ranges;
15  Higher_dimensional_cell_complex hdcc;
16 };
```

Figure A.2: Using C++ templates it is possible to create dependent types such as the `Higher_dimensional_cell_complex` defined in line 5 and used in line 15.

is created during compilation.

This type of mechanism can be used to a much higher degree by defining *recursive templates*, which are used extensively in the CGAL Combinatorial Maps package. As an example, Figure A.3 shows how this was applied in order to store the extrusion ranges maps for each dimension separately, which is necessary because the embedding structures of each cell are different depending on the dimension.

The same technique can be used to create algorithms that are also fully dimension-independent. In fact, C++ templates are known to be Turing-complete [Veldhuizen, 2003], and thus can be used to compute general-purpose problems. Figure A.4 shows one such example from the implementation of the incremental construction algorithm of Chapter 7, which shows the validation that a set of $(n-1)$-cells form a quasi-n-manifold.

```
1  // Abstracts a map of cell->ranges for a particular dimension
2  template <class LCC, unsigned int dimension>
3  struct Extrusion_ranges_map_of_dimension {
4  public:
5    typedef std::map<typename LCC::template Attribute_const_handle<dimension>::type,
6      Extrusion_ranges<LCC> > type;
7    type ranges_map;
8  };
9
10 // Abstracts a tuple of maps of cell->ranges, each element containing the map of a particular dimension
11 template <class LCC, unsigned int dimension = LCC::dimension, class Result = CGAL::cpp11::tuple<> >
12 struct Extrusion_ranges_tuple_per_dimension_up_to;
13
14 template <class LCC, class ... Result>
15 struct Extrusion_ranges_tuple_per_dimension_up_to<LCC, 0, CGAL::cpp11::tuple<Result ...> > {
16   typedef CGAL::cpp11::tuple<Extrusion_ranges_map_of_dimension<LCC, 0>, Result ...> type;
17 };
18
19 template <class LCC, unsigned int dimension, class ... Result>
20 struct Extrusion_ranges_tuple_per_dimension_up_to<LCC, dimension, CGAL::cpp11::tuple<Result ...> > {
21   typedef typename Extrusion_ranges_tuple_per_dimension_up_to<LCC,dimension - 1,
22     CGAL::cpp11::tuple<Extrusion_ranges_map_of_dimension<LCC, dimension>, Result ...> >::type type;
23 };
24
25 // Abstracts a tuple of maps of cell->ranges, each element containing the map of a particular dimension
26 template <class LCC_>
27 struct Extrusion_ranges_tuple_per_dimension {
28 public:
29   typedef LCC_ LCC;
30   typedef typename Extrusion_ranges_tuple_per_dimension_up_to<LCC>::type type;
31   type ranges;
32 };
```

Figure A.3: Recursive C++ templates can be used to generate dimension independent code. The first structure `Extrusion_ranges_map_of_dimension` (lines 1–8) contains the extrusion map for a single dimension. The second structure `Extrusion_ranges_tuple_per_dimension_up_to` (lines 10–23) uses as a triplet of definitions to create copies of the first for every dimension up to a given one. This is done using a template specialisation for dimension 0 which stops the recursion. The last structure `Extrusion_ranges_tuple_per_dimension` (lines 25–32) creates all structures using the dimension of a passed combinatorial map `LCC`. Note the non-ideal use of `std::map` in line 5.

```
template <unsigned int dimension, class InputIterator>
Dart_handle get_cell(InputIterator begin, InputIterator end) {

  // Validate that this will create a closed (topologically open) quasi-manifold cell
  for (typename std::list<Dart_handle>::iterator dart_in_current_facet_1 = free_facets.begin();
       dart_in_current_facet_1 != free_facets.end(); ++dart_in_current_facet_1) {

    // Go ridge by ridge
    for (auto dart_in_current_ridge_1 = lcc.one_dart_per_incident_cell<dimension-2, dimension-1, dimension-1>(
           *dart_in_current_facet_1).begin();
         dart_in_current_ridge_1 != lcc.one_dart_per_incident_cell<dimension-2, dimension-1, dimension-1>(
           *dart_in_current_facet_1).end();
         ++dart_in_current_ridge_1) {
      int matches = 0;

      // Find possible matches
      for (typename std::list<Dart_handle>::iterator dart_in_current_facet_2 = free_facets.begin();
           dart_in_current_facet_2 != free_facets.end();
           ++dart_in_current_facet_2) {
        if (&*dart_in_current_facet_1 == &*dart_in_current_facet_2) continue; // Compare by memory address

        // Go ridge by ridge in a potential match
        for (auto current_dart_in_facet_2 = lcc.darts_of_cell<dimension-1, dimension-1>(
               *dart_in_current_facet_2).begin();
             current_dart_in_facet_2 != lcc.darts_of_cell<dimension-1, dimension-1>(
               *dart_in_current_facet_2).end();
             ++current_dart_in_facet_2) {

          // Check if it's a complete match (isomorphism test using signatures)
          if (isomorphic<dimension-2>(lcc,dart_in_current_ridge_1, current_dart_in_facet_2)) ++matches;
          if (isomorphic_reversed<dimension-2>(lcc, dart_in_current_ridge_1, current_dart_in_facet_2)) ++matches;
        }
      }

      CGAL_precondition(matches == 1);
    }
  }
}
```

Figure A.4: Recursive C++ templates can also be used to implement dimension-independent algorithms. In this case, the `dimension` that is passed to the templated function is passed on to other functions to obtain appropriate orbits for a given facet (`dimension-1`) or ridge (`dimension-2`), which are then compared dart by dart.

A short dictionary of dimension-based GIS terms

B

ambient space the space in which objects are embedded.

area 1. measure of the 2D extent of an object; 2. 2D combinatorial element; 3. 2D geometric object.

ball 1. (→ same as 3-*ball*) topological definition of the space within a 2-sphere (i.e. a sphere), often defined as the space within a certain distance (the radius) from a point in $\mathbb{R}^3$ (the centre); 2. (→ often n-*ball*) topological definition of the space within an $(n-1)$-sphere, often defined as the space within a certain distance (the radius) from a point in $\mathbb{R}^n$ (the centre); 3. geometric definition of a perfectly round filled 3D object.

body n-dimensional combinatorial element in an nD context.

box 1. cuboid; 2. orthotope of any dimension.

cavity 3D hole.

cell 1. (→ sometimes n-*cell*) nD combinatorial element; 2. 3D combinatorial element.

cell complex topological space formed by a set of cells glued along common faces, all faces of a cell in the complex should be in the complex.

circle 1. topological definition of the space at a certain distance (the radius) from a point in $\mathbb{R}^2$ (the centre); 2. geometric definition of a perfectly round hollow 2D object.

closed 1. topological definition of an object that includes its boundary; 2. geometric definition of an object in nD, usually defined using boundary representation, that encloses an nD subspace.

congruent having the same shape and size.

cube 1. (→ same as 3-*cube*) 3D polyhedron with 6 square facets; 2. (→ usually n-*cube*) n-orthotope with identical $(n-1)$-cube facets, akin to a square in 2D and a cube in 3D.

cuboid 1. box-shaped 3D polyhedron with 6 rectangular facets, its opposite facets are congruent and parallel; 2. parallelotope.

curve 1. 1D geometric object, often with a non-linear geometry; 2. 1D combinatorial element; 3. 1-manifold.

cut-line degenerate part of a 2D shape forming a curve and protruding inwards from its boundary.

disk topological definition of the space within a circle, often defined as the space within a certain distance (the radius) from a point in $\mathbb{R}^2$ (the centre).

edge 1D combinatorial element.

face 1. 2D combinatorial element; 2. (→ sometimes n-*face of* X) nD combinatorial element on the boundary of X; 3. (→ usually *face of* X) $(n-1)$D combinatorial element on the boundary of an nD combinatorial element X.

facet 1. $(n-1)$D combinatorial element in an nD context; 2. (→ often *facet of* X) $(n-1)$D combinatorial element on the boundary of an nD combinatorial element X; 3. 2D combinatorial element.

flat unbounded geometric object with linear geometry of any dimension, such as a point, line or plane.

hole 1. 2D void region; 2. nD void region.

hyperball higher-dimensional ball.

hypercell 1. higher-dimensional combinatorial element; 2. 4D combinatorial element.

hypercube 1. higher-dimensional cube; 2. 4-cube.

hyperplane 1. $(n-1)$D linear subspace in an nD context; 2. plane in a higher-dimensional context.

hyperrectangle higher-dimensional orthotope.

hypersphere higher-dimensional sphere.

hypersurface $(n-1)$D subspace in an nD context, often curved;

interval topological definition of the space between two points in $\mathbb{R}$ (the endpoints).

Lebesgue measure measure of the nD extent of an object, akin to length in 1D, area in 2D and volume in 3D.

length measure of the 1D extent of an object.

line segment 1D geometric object.

manifold (→ sometimes n-*manifold*) topological space that resembles $\mathbb{R}^n$ at every point.

manifold with boundary (→ sometimes n-*manifold with boundary*) topological space that resembles $\mathbb{R}^n$ at every point in its interior.

node 0D combinatorial element.

open 1. topological definition of an object that does not include its boundary; 2. geometric definition of an object in nD, usually defined using boundary representation, that does not enclose any nD subspace.

orthotope (→ sometimes n-*orthotope*) n-polytope with congruent parallel facets forming right angles to each other, akin to a rectangle in 2D and a cuboid in 3D.

parallelepiped 3D polyhedron bounded by 6 parallelogram-shaped 2D faces.

parallelogram polygon bounded by 4 parallel and have the same shape.

parallelotope a polytope whose opposite facets are parallel and have the same shape, akin to a parallelogram in 2D and a parallelepiped in 3D.

peak 1. $(n-3)$D combinatorial element in an nD context; 2. (→ *peak of* X) $(n-3)$D combinatorial element on the boundary of an nD combinatorial element X;

plane 2D unbounded linear subspace.

point 1. 0D geometric element; 2. topological definition of the space at one location.

polychoron 4D geometric object with linear geometry.

polygon 2D geometric object with linear geometry, possibly with holes.

polygonal curve curve formed by a sequence of line segments joined at their endpoints.

polyhedron 1. 3D geometric object with linear geometry, possibly with holes; 2. an nD geometric object with linear geometry.

polyline curve formed by a sequence of line segments joined at their endpoints.

polyteron 5D geometric object with linear geometry.

polytope nD geometric object with linear geometry.

puncture 0D (point) hole, often formed from a degenerate polyline or polygon.

ridge 1. $(n-2)$D combinatorial element in an nD context; 2. (→ *ridge of* X) $(n-2)$D combinatorial element on the boundary of an nD combinatorial element X.

line string curve formed by joining a sequence of vertices by straight line segments, represented by these vertices.

linear ring 2D geometric object with linear geometry represented as a sequence of vertices.

ring 2D combinatorial element represented by its 1D boundary.

subfacet 1. $(n-2)$D combinatorial element in an nD context; 2. (→ *subfacet of* X) $(n-2)$D combinatorial element on the boundary of an nD combinatorial element X.

shell 3D geometric object with linear geometry without 3D holes, usually defined as the volume enclosed by its 2D boundary, often represented as a set of 2D faces.

simplex 1. (→ sometimes *n-simplex*) combinatorial element with $n+1$ vertices and $n+1$ facets; 2. (→ sometimes *n-simplex*) geometric object based on $n+1$ affinely independent vertices, akin to a point in 0D, a line segment in 1D, a triangle in 2D, or a tetrahedron in 3D.

simplicial complex 1. (→ sometimes *abstract simplicial complex*) topological space formed by a set of simplices glued along common faces, all faces of a simplex in the complex should be in the complex; 2. (→ sometimes *geometric simplicial complex*) abstract simplicial complex where simplices are embedded into Euclidean space, their interiors should not intersect geometrically.

solid 1. 3D geometric object with linear geometry, possibly with 3D holes, often represented as a set of shells; 2. object defined based on solid modelling, often as opposed one based on boundary representation; 3. an object containing its interior (as opposed to *hollow*).

spike degenerate part of a 2D shape forming a curve, often protruding outwards from its boundary.

sphere 1. (→ same as 2-*sphere*) topological definition of the space at a certain distance (the radius) from a point in $\mathbb{R}^3$ (the centre); 2. (→ often *n-sphere*) topological definition of the space at a certain distance (the radius) from a point in $\mathbb{R}^{n+1}$ (the centre); 3. geometric definition of a perfectly round hollow 3D object.

surface 1. 2-manifold; 2. 2D combinatorial element.

tesseract 4-cube: polychoron with 8 cubical facets.

vertex 0D combinatorial element.

volume 1. measure of the 3D extent of an object; 2. 3D combinatorial element.

wire 2D combinatorial element represented by its 1D boundary.

Bibliography

Edwin A. Abbott. *Flatland: A Romance of Many Dimensions.* Seely & Co., 1884. Cited on page 1.

Imad Afyouni, Cyril Ray, and Christophe Claramunt. Spatial models for indoor & context-aware navigation systems: A survey. *Journal of Spatial Information Science*, 4:85–123, 2012. Cited on page 25.

Ricardo Pérez Aguila and Antonio Aguilera Ramírez. Classifying edges and faces as manifold or non-manifold elements in 4D orthogonal pseudo-polytopes. In *WSCG'2003*. UNION Agency - Science Press, February 2003. Cited on page 73.

Murad Davudovich Akhundov. *Conceptions of space and time: sources, evolution, directions.* MIT Press, November 1986. Cited on page 45.

Khaled K. Al-Taha, Richard T. Snodgrass, and Michael D. Soo. Bibliography on spatiotemporal databases. *International Journal of Geographical Information Systems*, 8(1):95–103, 1994. Cited on page 46.

N. Alam, D. Wagner, M. Wewetzer, M. Pries, and V. Coors. Towards automatic validation and healing of CityGML models for geometric and semantic consistency. In Umit Isikdag, editor, *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences. Proceedings of the ISPRS 8th 3DGeoInfo Conference & WG II/2 Workshop*, pages 1–6, Istanbul, Turkey, 2013. Cited on page 178.

J. W. Alexander. An example of a simply connected surface bounding a region which is not simply connected. *Proceedings of the National Academy of Sciences of the United States of America*, 10(1):8–10, 1924. Cited on page 122.

Tyler J. Alumbaugh and Xiangmin Jiao. Compact array-based mesh data structures. In *Proceedings of the 14th International Meshing Roundtable*, pages 485–504, 2005. Cited on page 26.

Nina Amenta and Marshall Bern. Surface reconstruction by Voronoi filtering. In *SCG '98 Proceedings of the 14th annual symposium on Computational geometry*, pages 39–48. ACM, 1998. Cited on page 49.

Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Communications of the ACM*, 51(1):117–122, 2008. Cited on page 94.

Marc P. Armstrong. Temporality in spatial databases. In *GIS/LIS '88 : proceedings : accessing the world : third annual International Conference, Exhibits, and Workshops*, pages 880–889. American Society for Photogrammetry and Remote Sensing, 1988. Cited on pages 4, 46, and 51.

Ken Arroyo Ohori. Validation and automatic repair of planar partitions using a constrained triangulation. Master's thesis, Delft University of Technology, August 2010. Cited on page 191.

Ken Arroyo Ohori, Hugo Ledoux, and Jantien Stoter. Modelling higher dimensional data for GIS using generalised maps. In B. Murgante, S. Misra, M. Carlini, C. Torre, H.Q. Nguyen, D. Taniar, B. Apduhan, and O. Gervasi, editors, *Computational Science and Its Applications — ICCSA 2013. 13th International Conference, Ho Chi Minh City, Vietnam, June 24–27, 2013, Proceedings, Part I*, volume 7971 of *Lecture Notes in Computer Science*, pages 526–539. Springer Berlin Heidelberg, Ho Chi Minh City, Vietnam, June 2013. Cited on page 78.

Emil Artin. *Geometric Algebra.* Nabu Press, 2011. Cited on page 80.

Marco Attene, Daniela Giorgi, Massimo Ferri, and Bianca Falcidieno. On converting sets of tetrahedra to combinatorial and PL manifolds. *Computer Aided Geometric Design*, 26(8):850–864, November 2009. Cited on page 178.

Marco Attene, Marcel Campen, and Leif Kobbelt. Polygon mesh repairing: An application perspectice. *ACM Computing Surveys*, 45(2), 2013. Cited on page 177.

Matt Austern. Why you shouldn't use set (and what you should use instead). *C++ Report*, 12(4), 2000. Cited on page 113.

Wael M. Badawy and Walid G. Aref. On local heuristics to speed up polygon-polygon intersection tests. In *Proceedings of the 7th ACM International Symposium on Advances in Geographic Information Systems*, pages 97–102. ACM, 1999. Cited on page 168.

Trevor C. Bailey and Anthony C. Gatrell. *Interactive spatial data analysis*. Longman, 1995. Cited on page 11.

C. L. Bajaj, V. Pascucci, and G. Rabbiolo. Hypervolume visualization: A challenge in simplicity. In *IEEE Symposium on Volume Visualization*. IEEE, 1998. Cited on page 162.

Chanderjit Bajaj and Tamal K. Dey. Convex decomposition of polyhedra and robustness. Technical report, Purdue University, 1990. Cited on page 28.

Dana H. Ballard. Strip trees: A hierarchical representation for curves. *Communications of the ACM*, 24(5): 310–321, May 1981. Cited on page 49.

Ron J. Balsys and Kevin G. Suffern. Point based rendering of implicit 4-dimensional surfaces. In *Computer Graphics, Imaging and Visualisation*. IEEE, 2007. Cited on page 162.

C. Bradford Barber, David P. Dobkin, and Hannu Huhdanpaa. The Quickhull algorithm for convex hulls. *ACM Transactions on Mathematical Software*, 22(4):469–483, December 1996. Cited on pages 67 and 80.

Gill Barequet and Micha Sharir. Filling gaps in the boundary of a polyhedron. *Computer Aided Geometric Design*, 12(2):207–229, March 1995. Cited on page 178.

Umit Basoglu and Joel Morrison. The efficient hierarchical data structure for the US historical boundary file. In Geoffrey Dutton, editor, *Harvard Papers on Geographic Information Systems*, volume 4. Addison-Wesley, 1978. Cited on pages 46 and 48.

Bruce G. Baumgart. A polyhedron representation for computer vision. In *AFIPS '75 Proceedings of the May 19-22, 1975, national computer conference and exposition*, pages 589–596. ACM, 1975. Cited on pages 4 and 37.

Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. The R*-tree: An efficient and robust access method for points and rectangles. In *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data*, number 322–330, 1990. Cited on page 94.

Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975. Cited on pages 25, 31, and 94.

Lars Bernard, Benno Schmidt, and Ulrich Streit. AtmoGIS — integration of atmospheric models and GIS. In T.K. Poiker and N. Chrisman, editors, *Proceedings of the 8th International Symposium on Spatial Data Handling*, 1998. Cited on pages 64 and 80.

Pierre Bézier. *Essai de définition numérique des courbes et des surfaces experimentales: Contribution à l'étude des propriétés des courbes et des surfaces paramétriques polynomiales à coefficients vectoriels*. PhD thesis, Université Pierre et Marie Curie, 1977. Cited on page 40.

H. Bieri and W. Nef. Elementary set operations with d-dimensional polyhedra. In Hartmut Noltemeier, editor, *Computational Geometry and its Applications*, volume 333 of *Lecture Notes in Computer Science*, pages 97–112. Springer Berlin Heidelberg, 1988. Cited on pages 29, 70, 89, 189, and 192.

Filip Biljecki, Hugo Ledoux, and Jantien Stoter. Redefining the level of detail for 3D models. *GIM International*, 28(11):21–23, November 2014a. Cited on page 49.

Filip Biljecki, Hugo Ledoux, Jantien Stoter, and Junqiao Zhao. Formalisation of the level of detail in 3D city modelling. *Computers, Environment and Urban Systems*, 48:1–15, 2014b. Cited on pages 40, 48, 136, and 137.

Filip Biljecki, Hugo Ledoux, and Jantien Stoter. Improving the consistency of multi-LOD CityGML datasets by removing redundancy. In Martin Breunig, Al-Doori Mulhim, Edgar Butwilowski, Paul Vincent Kuper, Joachim Benner, and Karl-Heinz Häfele, editors, *3D Geoinformation Science. The Selected Papers of the 3D GeoInfo 2014*, pages 1–17. Springer International Publishing, 2015. Cited on page 48.

Stephan Bischoff, Darko Pavic, and Leif Kobbelt. Automatic restoration of polygon models. *ACM Transactions on Graphics*, 24(4):1332–1352, 2005. Cited on page 178.

Denis Blackmore, Ming C. Leu, and Frank Shih. Analysis and modelling of deformed swept volumes. *Computer-Aided Design*, 26(4):315–326, April 1994. Cited on page 27.

Daniel K. Blandford, Guy E. Blelloch, David E. Cardoze, and Clemens Kadow. Compact representations of simplicial meshes in two and three dimensions. *International Journal of Computational Geometry and Applications*, 15(1):3–24, 2005. Cited on pages 26 and 35.

Moshe Blank, Lena Gorelick, Eli Shechtman, Michal Irani, and Ronen Basri. Actions as space-time shapes. In *Proceedings of the 10th IEEE International Conference on Computer Vision*, 2005. Cited on page 55.

T. Blaschke. Object based image analysis for remote sensing. *ISPRS Journal of Photogrammetry and Remote Sensing*, 65(1):2–16, January 2010. Cited on page 81.

J. Bogdahn and V. Coors. Towards an automated healing of 3D urban models. In T. H. Kolbe, G. Köning, and C. Nagel, editors, *Proceedings International Conference on 3D Geoinformation*, volume XXXVIII–4/W15, pages 13–17, 2010. Cited on page 178.

Pawel Boguslawski and Christopher Gold. Rapid modelling of complex building interiors. In Thomas H. Kolbe, Gerhard König, and Claus Nagel, editors, *Advances in 3D Geo-Information Sciences*, Lecture Notes in Geoinformation and Cartography, pages 43–56. Springer, 2011. Cited on page 96.

Pawel Boguslawski, Christopher M. Gold, and Hugo Ledoux. Modelling and analysing 3D buildings with a primal/dual data structure. *ISPRS Journal of Photogrammetry & Remote Sensing*, 66:188–197, 2011. Cited on page 96.

Jan Helge Bøhn and Michael J. Wozny. A topology-based approach for shell-closure. In *Selected and Expanded Papers from the IFIP TC5/WG5.2 Working Conference on Geometric Modeling for Product Realization*, pages 297–319. North-Holland Publishing Co, 1992. Cited on page 178.

Jean-Daniel Boissonnat and Clément Maria. The simplex tree: An efficient data structure for general simplicial complexes. In *Algorithms — ESA 2012*, volume 7501 of *Lecture Notes in Computer Science*, pages 731–742. Springer Berlin Heidelberg, 2012. Cited on page 119.

Jean-Daniel Boissonnat, Olivier Devillers, Sylvain Pion, Monique Teillaud, and Mariette Yvinec. Triangulations in CGAL. *Computational Geometry: Theory & Applications*, 22:5–19, 2002. Cited on pages 68 and 169.

János Bolyai. *Appendix Scientiam Spatii Absolute Veram Exhibens*. Maros-Vásárhelyini, 1832. Cited on page 13.

B. Boots. Spatial tessellations. In Paul A. Longley, Michael F. Goodchild, David J. Maguire, and David W. Rhind, editors, *Geographical Information Systems*, chapter 36. John Wiley & Sons, 1999. Cited on page 30.

John A. Brewer III and S. Mark Courter. Automated conversion of curvilinear wire-frame models to surface boundary models; a topological approach. In *SIGGRAPH '86 Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, volume 20. ACM, 1986. Cited on page 32.

Erik Brisson. Representing geometric structures in d dimensions: topology and order. In *Proceedings of the 5th annual symposium on Computational geometry*, pages 218–227, New York, NY, USA, 1989. ACM. Cited on pages 76 and 77.

Erik Brisson. Representing geometric structures in d dimensions: topology and order. *Discrete & Computational Geometry*, 9:387–426, 1993. Cited on pages 5 and 189.

L.E.J. Brouwer. Beweis des Jordanschen Satzes für den n-dimensionalen Raum. *Mathematische Annalen*, 71: 314–319, 1911. Cited on pages 33, 60, 122, 132, and 190.

Robin P. Bryant and David Singerman. Foundations of the theory of maps on surfaces with boundary. *Quarterly Journal of Mathematics*, 36(2):17–41, 1985. Cited on page 73.

Rizwan Bulbul and Andrew U. Frank. AHD: The alternate simplicial decomposition of nonconvex polytopes (generalization of a convex polytope based spatial data model). In *Proceedings of the 17th International Conference on Geoinformatics*, pages 1–6, 2009. Cited on pages 66 and 79.

P.A. Burrough. *Principles of geographical information systems for land resources assessment*. Taylor & Francis, 1986. Cited on page 11.

Barbara P. Buttenfield and Joseph S. DeLotto. Multiple representations: Scientific report for the specialist meeting. Technical Report 89-3, National Center for Geographic Information and Analysis, 1989. Cited on page 49.

Georg Cantor. Ueber eine Eigenschaft des inbegriffs aller reellen algebraischen Zahlen. *Journal für die reine und angewandte Mathematik*, 77:258-262, 1874. Cited on page 11.

CEC. Corine land cover. Technical report, Commission of the European Communities, 1995. Cited on page 4.

Bernard Chazelle. Convex partitions of polyhedra: a lower bound and worst-case optimal algorithm. *SIAM Journal on Computing*, 13(3):488-507, 1984. Cited on page 45.

Bernard Chazelle and David Dobkin. Decomposing a polygon into its convex parts. In *Proceedings of the 11th Annual ACM Symposium on Theory of Computing*, pages 38-48, 1979. Cited on page 28.

Pin-Shan Chen. The entity-relationship model—toward a unified view of data. *ACM Transactions on Database Systems*, 1(1):9-36, 1976. Cited on page 47.

Siu-Wing Cheng, Tamal Krishna Dey, and Jonathan Richard Shewchuk. *Delaunay Mesh Generation*. CRC Press, 2012. Cited on pages 49 and 177.

L. Paul Chew. Constrained Delaunay triangulations. *Algorithmica*, 4(1):97-108, 1989. Cited on page 32.

Matt Chisholm. The sphere in three dimensions and higher: Generalizations and special cases. Available at `https://theory.org/geotopo/3-sphere/3-sphere.ps`, May 2000. Cited on page 160.

Nicholas R. Chrisman. The role of quality information in the long-term functioning of a geographic information system. *Cartographica*, 1983. Cited on page 46.

Nicholas R. Chrisman. The risks of software innovation: A case study of the Harvard lab. *The American Cartographer*, 15(3):291-300, 1988. Cited on page 43.

Alan Chu, Chi-Wing Fu, Andrew J. Hanson, and Pheng-Ann Heng. GL4D: A GPU-based architecture for interactive 4D visualization. In *IEEE Transactions on Visualization and Computer Graphics*, volume 15, pages 1587-1594. IEEE, 2009. Cited on page 161.

Christophe Claramunt and Marius Thériault. Managing time in GIS: An event-oriented approach. In James Clifford and Alexander Tuzhilin, editors, *Proceedings of the International Workshop on Temporal Databases*, pages 23-42, 1995. Cited on page 47.

Christophe Claramunt, Christine Parent, Stefano Spaccapietra, and Marius Thériault. Database modelling for environmental and land use changes. In John Charles Harold Stillwell, Stan Geertman, and Stan Openshaw, editors, *Geographical Information and Planning*, Advances in Spatial Science, chapter 10, pages 181-202. Springer Berlin / Heidelberg, 1999. Cited on page 47.

Kenneth L. Clarkson, Kurt Mehlhorn, and Raimund Seidel. Four results on randomized incremental constructions. In Alain Finkel and Matthias Jantzen, editors, *Proceedings of the 9th Annual Symposium on Theoretical Aspects of Computer Science*, volume 577 of *Lecture Notes in Computer Science*, pages 461-474. Springer Berlin Heidelberg, 1992. Cited on page 169.

Lidija Čomić and Leila de Floriani. *Modeling and Manipulating Cell Complexes in Two, Three and Higher Dimensions*, volume 2 of *Lecture Notes in Computational Vision and Biomechanics*, chapter 4, pages 109-144. Springer, 2012. Cited on pages 26 and 74.

Lidija Čomić, Leila de Floriani, Federico Iuricich, and Ulderico Fugacci. Topological modifications and hierarchical representation of cell complexes in arbitrary dimensions. *Computer Vision and Image Understanding*, 121:2-12, 2014. Cited on page 79.

J.H. Conway and N.J.A. Sloane. *Sphere Packings, Lattices and Groups*. Springer-Verlag, 1992. Cited on pages 30 and 63.

J.T. Coppock and D.W. Rhind. The history of GIS. In D.J. Maguire, M.F. Goodchild, and D.W. Rhind, editors, *Geographical Information Systems: Principles and Applications*, volume 1, chapter 2, pages 21-43. Longman Scientific & Technical, 1991. Cited on page 3.

Robert Cori. Un code pour les graphes planaires et ses applications. Technical report, Société mathématique de France, 1975. Cited on page 38.

H. Couclelis. Space, time, geography. In Paul A. Longley, Michael F. Goodchild, David J. Maguire, and David W. Rhind, editors, *Geographical Information Systems*, chapter 2, pages 29–38. John Wiley & Sons, 1999. Cited on pages 11 and 55.

Helen Couclelis. People manipulate objects (but cultivate fields): Beyond the raster-vector debate in GIS. In A. U. Frank, I. Campari, and U. Formentini, editors, *Theories and Methods of Spatio-Temporal Reasoning in Geographic Space*, volume 639 of *Lecture Notes in Computer Science*, pages 65–77. Springer Berlin Heidelberg, 1992. Cited on page 24.

Robert G. Cromley. *Digital Cartography*. Prentice Hall, 1992. Cited on page 43.

Maciej Dakowicz and Christopher M. Gold. Extracting meaningful slopes from terrain contours. *International Journal of Computational Geometry and Applications*, 13(4):339–357, 2003. Cited on page 95.

Boris Dalstein, Rémi Ronfard, and Michiel van de Panne. Vector graphics animation with time-varying topology. *ACM Transactions of Graphics*, 34(4), 2015. Cited on pages 46 and 186.

Guillaume Damiand and Pascal Lienhardt. Removal and contraction for n-dimensional generalized maps. In *Proceedings of the 11th Discrete Geometry for Computer Imagery*, volume 2886, pages 408–419, 2003. Cited on page 142.

Guillaume Damiand and Pascal Lienhardt. *Combinatorial Maps: Efficient Data Structures for Computer Graphics and Image Processing*. CRC Press, 2014. Cited on page 86.

Guillaume Damiand, Samuel Peltier, and Laurent Fuchs. Computing homology generators for volumes using minimal generalized maps. In Valentin E. Brimkov, Reneta P. Barneva, and Hebert A. Hauptman, editors, *Proceedings of the 12th International Workshop on Combinatorial Image Analysis*, volume 4958 of *Lecture Notes in Computer Science*, pages 63–74. Springer, 2008. Cited on page 66.

Mark de Berg, Marc van Kreveld, René van Oostrum, and Mark Overmars. Simple traversal of a subdivision without extra storage. *International Journal of Geographical Information Science*, 11(4):359–374, 1997. Cited on page 71.

Mark de Berg, Marc van Kreveld, Mark Overmars, and Otfried Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, 3rd edition, 2008. Cited on pages 4, 36, 62, 90, and 107.

Pierre de Fermat. Ad locos planos et solidos isagoge. *Varia opera mathematica*, pages 1–8, 1679. Cited on page 13.

Leila de Floriani and Annie Hui. Data structures for simplicial complexes: an analysis and a comparison. In M. Desbrunn and H. Pottmann, editors, *Eurographics Symposium on Geometry Processing*. The Eurographics Association, 2005. Cited on pages 26 and 74.

Leila de Floriani, Leif Kobbelt, and Enrico Puppo. A survey on data structures for level-of-detail models. In Neil A. Dodgson, Michael S. Floater, and Malcolm A. Sabin, editors, *Advances in Multiresolution for Geometric Modelling. Part II*, Mathematics and Visualization, pages 49–74. Springer Berlin Heidelberg, 2005. Cited on page 26.

Arnaud de la Losa and Bernard Cervelle. 3D topological modeling and visualisation for 3D GIS. *Computers & Graphics*, 23(4):469–468, August 1999. Cited on page 44.

René Descartes. *Discours de la méthode*. Jan Maire, Leyde, 1637. Cited on pages 5, 13, and 54.

Olivier Devillers. Walking in a triangulation. *International Journal of Foundations of Computer Science*, 13(2): 181–199, 2002. Cited on page 68.

Keith Devlin. *The Joy of Sets: Fundamentals of Contemporary Set Theory*. Springer-Verlag, 1993. Cited on page 12.

T. Devogele, J. Trevisan, and L. Raynal. Building a multi-scale database with scale-transition relationships. In *Proceedings of the 7th International Symposium on Spatial Data Handling*, pages 337–351, Delft, Netherlands, 1996. Cited on page 137.

A. A. Diakité, Guillaume Damiand, and D. van Maercke. Topological reconstruction of complex 3d buildings and automatic extraction of levels of detail. In *Proceedings of the 2nd Eurographics Workshop on Urban Data Modelling and Visualisation*, 2014. Cited on pages 97 and 134.

David P. Dobkin and Michael J. Laszlo. Primitives for the manipulation of three-dimensional subdivisions. In *Proceedings of the 3rd Annual Symposium on Computational Geometry*, pages 86–99. ACM, 1987. Cited on pages 26, 38, and 77.

Jürgen Döllner and Henrik Buchholz. Continuous level-of-detail modeling of buildings in 3D city models. In *GIS'05*, pages 173–181. ACM, 2005. Cited on page 55.

Jürgen Döllner, Thomas H. Kolbe, Falko Liecke, Takis Sgouros, and Karin Teichmann. The virtual 3D city model of Berlin — managing, integrating and communicating complex urban information. In *UDMS 2006*, 2006. Cited on page 136.

B. Domínguez, Á.L. García, and F.R. Feito. Semantic and topological representation of building indoors: an overview. In *Proceedings of the Joint ISPRS Workshop on 3D City Modelling & Applications and the 6th 3D GeoInfo Conference*, volume 66, pages 209–222, 2011. Cited on page 25.

Sjors Donkers. Automatic generation of CityGML LoD3 building models from IFC models. Master's thesis, Delft University of Technology, December 2013. Cited on page 172.

David H. Douglas and Thomas K. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 10(2):112–122, December 1973. Cited on page 49.

Matthew P. Dube and Max J. Egenhofer. An ordering of convex topological relations. In Ningchuan Xiao, Mei-Po Kwan, Michael F. Goodchild, and Shashi Shekhar, editors, *Geographic Information Science*, volume 7478 of *Lecture Notes in Computer Science*, pages 72–86. Springer Berlin Heidelberg, 2012. Cited on page 62.

John Earman. How to talk about the topology of time. *Noûs*, 11(3):211–226, 1977. Cited on page 45.

Herbert Edelsbrunner. *A Short Course in Computational Geometry and Topology*. Springer Briefs in Applied Sciences and Technology. Springer, 2014. Cited on pages 16 and 79.

Herbert Edelsbrunner and Ernst P. Mücke. Simulation of simplicity: A technique to cope with degenerate cases in geometric algorithms. *ACM Transactions on Graphics*, 9(1):66–104, 1990. Cited on page 199.

J. Edmonds. A combinatorial representation of polyhedral surfaces. *Notices of the American Mathematical Society*, 7, 1960. Cited on pages 4, 37, 38, and 77.

EFI. Gutachten 2015: zu Forschung, Innovation und technologischer Leistungsfähigkeit Deutschlands. Technical report, Expertenkommission Forschung und Innovation, 2015. Cited on page 2.

Max J. Egenhofer and R. D. Franzosa. Point-set topological spatial relations. *International Journal of Geographical Information Systems*, 5(2):161–174, 1991. Cited on page 62.

Alberto Elduque. Vector cross products. Talk presented at the Seminario Rubio de Francia of the Universidad de Zaragoza on April 1, 2004, April 2004. Cited on page 93.

Herve Elter and Pascal Lienhardt. Cellular complexes as structured semi-simplicial sets. *International Journal of Shape Modeling*, 1(2), 1994. Cited on page 77.

David Eppstein. Subgraph isomorphism in planar graphs and related problems. *Journal of Graph Algorithms and Applications*, 3(3):1–27, 1999. Cited on page 139.

ESRI. *Shapefile Technical Description*. ESRI, July 1998. Cited on page 198.

ESRI. *GIS Topology*. ESRI, July 2005. Cited on pages 3, 43, 44, 50, and 81.

Leonhard Euler. Solutio problematis ad geometriam situs pertinentis. *Commentarii academiae scientiarum Petropolitanae*, 8:128–140, 1741. Cited on page 18.

Steven Feiner and Clifford Beshers. Visualizing n-dimensional virtual worlds with n-vision. In *Proceedings of the 1990 symposium on Interactive 3D graphics*, pages 37–38. ACM, 1990. Cited on page 162.

Vincenzo Ferrucci. Generalised extrusion of polyhedra. In *2nd ACM Solid Modelling '93*, pages 35–42. ACM, 1993. Cited on pages 103, 104, and 111.

Waldemar Celes Filho, Luiz Henrique de Figueiredo, Marcelo Gattass, and Paulo Cezar Carvalho. A topological data structure for hierarchical planar subdivisions. Technical Report CS-95-53, Department of Computer Science, University of Waterloo, 1995. Cited on pages 50 and 137.

Raphael Finkel and J.L. Bentley. Quad trees: A data structure for retrieval on composite keys. *Acta Informatica*, 4(1):1–9, 1974. Cited on pages 31, 65, and 94.

P. Fisher. The pixel: a snare and a delusion. *International Journal of Remote Sensing*, 18(3):679–685, 1997. Cited on page 64.

Richard Fitzpatrick. *Euclid's Elements of Geometry*. Richard Fitzpatrick, 2008. Cited on page 13.

James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer Graphics: Principles and Practice in C*. Addison-Wesley Professional, 1995. Cited on pages 25, 27, and 168.

Thomas A. Foley and Gregory M. Nielson. Practical techniques for producing 3D graphical images. In John Black, editor, *The System Engineer's Handbook: A guide to building VMEbus and VXIbus systems*, chapter 19, pages 223–237. Academic Press, 1992. Cited on pages 156 and 157.

David Fradin, Daniel Meneveaux, and Pascal Lienhardt. Partition de l'espace et hiérarchie de cartes généralisées: application aux complexes architecturaux. In *Actes des XVemes journées de l'Association Française d'Informatique Graphique*, volume 28, 2002. Cited on page 79.

Andrew U. Frank. Spatial concepts, geometric data models, and geometric data structures. *Computers & Geosciences*, 18(4):409–417, 1992. Cited on pages 25 and 34.

Scott M. Freundschuh and Max J. Egenhofer. Human conceptions of spaces: Implications for geographic information systems. *Transactions in GIS*, 2(4):361–375, 1997. Cited on page 24.

Anders Friis-Christensen and Christian S. Jensen. Object-relational management of multiply represented geographic entities. In *Proceedings of the 15th International Conference on Scientific and Statistical Database Management*, pages 150–159. IEEE Computer Society, 2003. Cited on pages 4 and 48.

Antony Galton. Fields and objects in space, time, and space-time. *Spatial Cognition and Computation*, 4(1): 39–68, 2004. Cited on page 55.

Michael Garland and Paul S. Heckbert. Surface simplification using quadric error metrics. In *SIGGRAPH '97 Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 209–216. ACM Press/Addison-Wesley Publishing, 1997. Cited on page 49.

Andreas Geiger, Joachim Benner, and Karl Heinz Haefele. Generalization of 3D IFC building models. In Martin Breunig, Mulhim Al-Doori, Edgar Butwilowski, Paul V. Kuper, Joachim Benner, and Karl Heinz Haefele, editors, *3D Geoinformation Science*, Lecture Notes in Geoinformation and Cartography, pages 19–35. Springer International Publishing Switzerland, 2015. Cited on page 49.

Christopher M. Gold. Problems with handling spatial data—the Voronoi approach. *CISM Journal*, 45(1):65–80, 1991. Cited on page 95.

Christopher M. Gold. Data structures for dynamic and multidimensional GIS. In *Proceedings of the 4th ISPRS Workshop on Dynamic and Multi-dimensional GIS*, pages 36–41, Pontypridd, Wales, UK, 2005. Cited on pages 25 and 55.

Christopher M. Gold. What is GIS and what is not? *Transactions in GIS*, 10(4):505–519, 2006. Cited on page 3.

David Goldberg. What every computer scientist should know about floating-point arithmetic. *Computing Surveys*, 23(1):5–48, March 1991. Cited on pages 15, 44, and 198.

Michael F. Goodchild. Geographical data modeling. *Computers & Geosciences*, 18(4):401–408, 1992. Cited on page 24.

Michael F. Goodchild. Metrics of scale in remote sensing and GIS. *International Journal of Applied Earth Observation and Geoinformation*, 3(2):114–120, 2001. Cited on page 47.

Stéphane Gosselin, Guillaume Damiand, and Christine Solnon. Efficient search of combinatorial maps using signatures. *Theoretical Computer Science*, 412(15):1392–1405, March 2011. Cited on pages 76, 97, 98, 123, 139, and 147.

Miguel Granados, Peter Hachenberger, Susan Hert, Lutz Kettner, Kurt Mehlhorn, and Michael Seel. Boolean operations on 3D selective nef complexes: Data structure, algorithms and implementation. In *Proceedings of the 11th Annual European Symposium on Algorithms*, pages 174–186, September 2003. Cited on pages 45 and 140.

Carine Grasset-Simon, Guillaume Damiand, and Pascal Lienhardt. nd generalized map pyramids: definition, representations and basic operations. *Pattern Recognition*, 39(4):527–538, 2006. Cited on page 55.

Gerhard Gröger and Lutz Plümer. Provably correct and complete transaction rules for updating 3D city models. *Geoinformatica*, 2011a. Cited on pages 149 and 191.

Gerhard Gröger and Lutz Plümer. How to achieve consistency for 3D city models. *Geoinformatica*, 15:137–165, 2011b. Cited on page 136.

Gerhard Gröger, Thomas H. Kolbe, Claus Nagel, and Karl-Heiz Häfele. *OGC City Geography Markup Language (CityGML) Encoding Standard. Version 2.0.0*. Open Geospatial Consortium, April 2012. Cited on pages 42, 48, 49, 102, 135, 165, 171, and 189.

André Guéziec and Francis Lazarus. Cutting and stitching: Converting sets of polygons to manifold surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 7(2):136–151, April–June 2001. Cited on page 178.

Leonidas J. Guibas and Jorge Stolfi. Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams. *ACM Transactions on Graphics*, 4(2):74–123, 1985. Cited on pages 26, 37, 38, 50, 77, 95, and 169.

Oliver Günther. The arc tree: An approximation scheme to represent arbitrary curved shapes. In *Efficient structures for geometric data management*, volume 337 of *Lecture Notes in Computer Science*, chapter 6, pages 85–121. Springer Berlin Heidelberg, 1988. Cited on page 49.

E.L. Gursoz, Y. Choi, and F.B. Prinz. Vertex-based representation of non-manifold boundaries. *Geometric Modeling for Product Engineering*, 1990. Cited on page 73.

Ralf Hartmut Güting, Michael H. Böhlen, Martin Erwig, Christian S. Jensen, Nikos A. Lorentzos, Markus Schneider, and Michalis Vazirgiannis. A foundation for representing and querying moving objects. *ACM Transactions on Database Systems*, 25(1):1–42, March 2000. Cited on page 62.

Antonin Guttman. R-trees: A dynamic index structure for spatial searching. In *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data*, pages 47–57, 1984. Cited on pages 25 and 94.

Peter Hachenberger. *Boolean Operations on 3D Selective Nef Complexes Data Structure, Algorithms, Optimized Implementation, Experiments and Applications*. PhD thesis, Saarland University, 2006. Cited on pages 39, 45, 89, 90, 140, and 192.

Torsten Hägerstrand. What about people in regional science? *Papers of the Regional Science Association*, 24(1): 6–21, 1970. Cited on page 55.

Mark Hampe, Karl-Heinrich Anders, and Monica Sester. MRDB applications for data revision and real-time generalisation. In *Proceedings of the 21st International Cartographic Conference*, pages 192–202, 2003. Cited on pages 136 and 139.

Torill Hamre, Khalid Azim Mughal, and Anita Jacob. A 4D marine data model: Design and application in ice monitoring. *Marine Geodesy*, 20(2–3):121–136, 1997. Cited on pages 46 and 51.

Patrick M. Hanrahan. Creating volume models from edge-vertex graphs. In *SIGGRAPH '82 Proceedings of the 9th annual conference on Computer graphics and interactive techniques*, pages 77–84. ACM, 1982. Cited on page 32.

Andrew J. Hanson. Geometry for n-dimensional graphics. In Paul S. Heckbert, editor, *Graphics Gems IV*, chapter II.6, pages 149–170. Academic Press Professional, 1994. Cited on pages 92, 93, 159, and 161.

Andrew J. Hanson and Daniel Weiskopf. Visualizing relativity. Siggraph 2001 Tutorial, 2001. Cited on pages 55 and 162.

Paul Hardy and Kenneth Field. *Portrayal and Cartography*, chapter 11, pages 323–358. Springer Handbook of Geographic Information. Springer, 2012. Cited on page 48.

Ian Hargreaves. Digital opportunity: A review of intellectural property and growth. Independent report commissioned by United Kingdom Prime Minister David Cameron, May 2011. Cited on page 2.

Allen Hatcher. *Algebraic Topology*. Cambridge University Press, 2002. Cited on page 19.

Felix Hausdorff. *Grundzüge der Mengenlehre*. Verlag von Veit & Comp., Leipzig, 1914. Cited on page 16.

Nicholas W.J. Hazelton, Lisa Marie Bennett, and Joanna Masel. Topological structures for 4-dimensional geographic information systems. *Computers, Environment and Urban Systems*, 16(3):227–237, May–June 1992. Cited on page 74.

N.W.J. Hazelton. Some operations requirements for a multi-temporal 4-D GIS. In M.J. Egenhofer and R.G. Golledge, editors, *Spatial and Temporal Reasoning in Geographic Information Systems*, pages 63–73. Oxford University Press, 1998. Cited on page 60.

N.W.J. Hazelton, F.J. Leahy, and I.P. Williamson. On the design of temporally-referenced, 3-D geographical information systems: development of four-dimensional GIS. In *Proceedings of GIS/LIS'90*, 1990. Cited on pages 60 and 74.

Martin Held and Willi Mann. An experimental analysis of floating-point versus exact arithmetic. In *Proceedings of the 23rd Canadian Conference on Computational Geometry*, 2011. Cited on page 199.

Michael Henle. *A Combinatorial Introduction to Topology*. Dover Publications, 1994. Cited on page 18.

Franck Hétroy, Stéphanie Rey, Carlos Andújar, Pere Brunet, and Àlvar Vinacua. Mesh repair with user-friendly topology control. *Computer-Aided Design*, 43(1):101–113, 2011. Cited on page 178.

Charles Howard Hinton. *A New Era of Thought*. Swan Sonnenschein & Co. Ltd., 1888. Cited on page 1.

Christoph M. Hoffmann. The problems of accuracy and robustness in geometric computation. Computer Science Technical Reports 88–771, Purdue University, 1988. Cited on pages 168 and 198.

Christoph M. Hoffmann. *Geometric and solid modeling*. Morgan Kaufmann Publishers, 1992. Cited on page 25.

Christoph M. Hoffmann and Jianhua Zhou. Visualization of surfaces in four-dimensional space. Computer Science Technical Reports CSD TR–960, Purdue University, May 1990. Cited on page 162.

Steven Richard Hollasch. Four-space visualization of 4D objects. Master's thesis, Arizona State University, 1991. Cited on pages 92 and 157.

Sébastien Horna, Guillaume Damiand, Daniel Meneveaux, and Yves Bertrand. Reconstruction topologique 3D de bâtiments. In *Journées de l'Association Francophone d'Informatique Graphique, Bordeaux, 2006*, 2006. Cited on page 179.

John F. Hughes, Andries van Dam, Morgan McGuire, David F. Sklar, James D. Foley, Steven K. Feiner, and Kurt Akeley. *Computer Graphics: Principles and Practice*. Addison-Wesley, 3rd edition, 2014. Cited on pages 152 and 156.

Gary J. Hunter and Ian P. Williamson. The development of a historical cadastral database. *International Journal of Geographical Information Systems*, 4(2):169–179, 1990. Cited on page 46.

IEEE. *IEEE Standard for Floating-Point Arithmetic*. The Institute of Electrical and Electronics Engineers, 3 Park Avenue, New York, NY 10016-5997, USA, June 2008. Cited on page 198.

Darrel C. Ince, Leslie Hatton, and John Graham-Cumming. The case for open computer programs. *Nature*, 482:487–488, February 2012. Cited on page 2.

ISO. *Geographic Information — Spatial Schema*. International Organization for Standardization, February 2005a. Cited on pages 42, 43, 166, 167, 168, 173, 174, 175, 182, 183, and 184.

ISO. *Geographic information — Temporal schema*. International Organization for Standardization, February 2005b. Cited on pages 45 and 55.

ISO. *Geographic information — Simple feature access — Part 1: Common architecture*. International Organization for Standardization, March 2006. Cited on pages 41 and 166.

ISO. *Geographic information — Schema for coverage geometry and functions*. International Organization for Standardization, 2007a. Cited on pages 30, 167, and 175.

ISO. *Geographic information — Geography Markup Language (GML)*. International Organization for Standardization, 2007b. Cited on page 42.

ISO. *Industry Foundation Classes (IFC) for data sharing in the construction and facility management industries*. International Organization for Standardization, March 2013. Cited on pages 42, 171, and 198.

ISO. *Industrial automation systems and integration - Product data representation and exchange*. International Organization for Standardization, August 2014. Cited on page 42.

ISO. *Information technology — Programming languages — C++*. International Organization for Standardization, 2015. Cited on pages 113, 128, and 200.

ISO/IEC. *Information technology — Eiffel: Analysis, Design and Programming Language*. International Organization for Standardization and International Electrotechnical Commission, 2007. Cited on page 165.

Chris L. Jackins and Steven L. Tanimoto. Quad-trees, oct-trees, and k-trees: A generalized approach to recursive decomposition of euclidean space. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 5(5): 533–539, September 1983. Cited on pages 65 and 94.

C.B. Jones and I.M. Abraham. Design considerations for a scale-independent cartographic database. In D. Marble, editor, *Proceedings of the 2nd International Symposium on Spatial Data Handling*, pages 384–398, 1986. Cited on page 49.

Christopher B. Jones and J. Mark Ware. Map generalisation in the web age. *International Journal of Geographical Information Science*, 19(8–9):859–870, 2005. Cited on page 49.

Lucas N. Joppa, Greg McInerny, Richard Harper, Lara Salido, Kenji Takeda, Kenton O'Hara, David Gavaghan, and Stephen Emmott. Troubling trends in scientific software use. *Science*, 340:814–815, May 2013. Cited on page 2.

M. C. Jordan. *Cours d'Analyse*. Gauthier-Villars, 1887. Cited on pages 4, 33, and 36.

Kenneth I. Joy, Justin Legakis, and Ron MacCracken. Data structures for multiresolution representation of unstructured meshes. In Gerard Farin, Bernd Hamann, and Hans Hagen, editors, *Hierarchical and Geometrical Methods in Scientific Visualization*, Mathematics and Visualization. Springer, 2003. Cited on pages 26 and 76.

Tao Ju. Fixing geometric errors on polygonal models: A survey. *Journal of Computer Science and Technology*, 24 (1):19–29, January 2009. Cited on page 177.

Tomasz Kaczynski, Konstantin Mischaikow, and Marian Mrozek. *Computational Homology*. Springer Berlin / Heidelberg, 2004. Cited on page 64.

Martin Kada. Scale-dependent simplification of 3D building models based on cell decomposition and primitive instancing. In *COSIT 2007*, volume 4736 of *Lecture Notes in Computer Science*, pages 222–237. Springer-Verlag Berlin Heidelberg, 2007. Cited on pages 27 and 49.

Farid Karimipour, Mahmoud R. Delavar, and Andrew U. Frank. A simplex-based approach to implement dimension independent spatial analyses. *Computers & Geosciences*, 36(9):1123–1134, September 2010. Cited on pages 5 and 70.

Michael Kazhdan, Matthew Bolitho1, and Hugues Hoppe2. Poisson surface reconstruction. In Konrad Polthier and Alla Sheffer, editors, *Eurographics Symposium on Geometry Processing*. The Eurographics Association, 2006. Cited on page 49.

Lutz Kettner, Kurt Mehlhorn, Sylvain Pion, Stefan Schirra, and Chee Yap. Classroom examples of robustness problems in geometric computations. *Computational Geometry*, 40(1):61–78, May 2008. Cited on page 198.

David Kirkpatrick, Jack Snoeyink, and Bettina Speckmann. Kinetic collision detection for simple polygons. In *Proceedings of the 16th Annual Symposium on Computational Geometry*, pages 322–330, 2000. Cited on page 168.

Thomas H. Kolbe, Gerhard Gröger, and Lutz Plümer. CityGML: Interoperable access to 3D city models. In Peter van Oosterom, Siyka Zlatanova, and Elfriede M. Fendel, editors, *Geo-information for Disaster Management*, pages 883–899. Springer Berlin Heidelberg, 2005. Cited on page 136.

Menno-Jan Kraak. The space-time cube revisited from a geovisualization perspective. In *Proceedings of the 21st International Cartographic Conference*, pages 1988–1996, 2003. Cited on page 55.

Pierre Kraemer, Lionel Untereiner, Thomas Jund, Sylvain Thery, and David Cazier. CGoGN: n-dimensional meshes with combinatorial maps. In J. Sarrate and M. Staten, editors, *Proceedings of the 22nd International Meshing Roundtable*, pages 485–503. Springer International Publishing Switzerland, 2014. Cited on page 78.

Werner Kuhn. Geospatial semantics: Why, of what, and how? *Journal on Data Semantics*, 2005. Cited on page 40.

P. Lancaster and K. Salkauskas. Surfaces generated by moving least squares methods. *Mathematics of Computation*, 37(155):141–158, July 1981. Cited on page 178.

Gail Langran and Nicholas R. Chrisman. A framework for temporal geographic informations. *Cartographica*, 25(3):1–14, 1988. Cited on pages 46, 48, and 107.

Charles L. Lawson. Properties of n-dimensional triangulations. *Computer-Aided Design*, 3(4):231–246, 1986. Cited on page 80.

Hai Ha Le. Spatio-temporal data construction. *ISPRS International Journal of Geo-Information*, 2:837–853, August 2013. Cited on page 79.

M. Lebesgue. Sur l'invariance du nombre de dimensions d'un espace et sur le theorème de M. Jordan relatif aux varieté fermées. *Comptes rendus de l'Académie des Sciences*, 152:841–844, 1911. Cited on pages 33, 60, 122, 132, and 190.

Hugo Ledoux. *Modelling Three-dimensional Fields in Geoscience with the Voronoi Diagram and its Dual*. PhD thesis, University of Glamorgan, November 2006. Cited on pages 32, 79, and 95.

Hugo Ledoux. On the validation of solids represented with the international standards for geographic information. *Computer-Aided Civil and Infrastructure Engineering*, 28(9):693–706, August 2013. Cited on pages 51, 174, and 177.

Hugo Ledoux and Christopher M. Gold. Modelling three-dimensional geoscientific fields with the Voronoi diagram and its dual. *International Journal of Geographical Information Science*, 22(5):547–574, 2008. Cited on page 95.

Hugo Ledoux and Martijn Meijers. Topologically consistent 3D city models obtained by extrusion. *International Journal of Geographical Information Science*, 25(4):557–574, 2011. Cited on pages 102 and 116.

Hugo Ledoux, Ken Arroyo Ohori, and Martijn Meijers. A triangulation-based approach to automatically repair GIS polygons. *Computers & Geosciences*, 66:121–131, May 2014. Cited on page 44.

Ickjai Lee and Mark Gahegan. Interactive analysis using Voronoi diagrams: Algorithms to support dynamic update from a generic triangle-based data structure. *Transactions in GIS*, 6(2):89–114, 2002. Cited on page 95.

J. Lee and M.-P. Kwan. A combinatorial data model for representing topological relations among 3D geographical features in micro-spatial environments. *International Journal of Geographical Information Science*, 19(10): 1039–1056, 2005. Cited on page 95.

J. Lee and Sisi Zlatanova. A 3D data model and topological analyses for emergency response in urban areas. In S. Zlatanova and Jonathan Li, editors, *Geospatial Information Technology for Emergency Response*, pages 143–168. Taylor & Francis, 2008. Cited on page 95.

Sang Hun Lee and Kunwoo Lee. Partial entity structure: A compact non-manifold boundary representation based on partial topological entities. In David C. Anderson and Kunwoo Lee, editors, *SMA '01 Proceedings of the 6th ACM Symposium on Solid modeling and Applications*, pages 159–170. ACM, 2001. Cited on page 73.

Yong Tsui Lee and Aristides A. G. Requicha. Algorithms for computing the volume and other integral properties of solids. I. Known methods and open issues. *Communications of the ACM*, 25(9):635–641, 1982. Cited on page 27.

Bruno Lévy. Dual domain extrapolation. *ACM Transactions on Graphics*, 22(3):364–369, July 2003. Cited on page 178.

Jos Leys, Étienne Ghys, and Aurelién Alvarez. Dimensions: une promenade mathématique. Available at `http://www.dimensions-math.org/Dim_E.htm`, 2008. Cited on page 160.

Jyh-Ming Lien and Nancy M. Amato. Approximate convex decomposition of polygons. *Computational Geometry: Theory & Applications*, 35:100–123, 2006. Cited on page 66.

Pascal Lienhardt. Extension of the notion of map and subdivisions of a three-dimensional space. In Robert Cori and Martin Wirsing, editors, *Proceedings of the 5th Symposium on the Theoretical Aspects of Computer Science*, volume 294 of *Lecture Notes in Computer Science*, pages 301–311. Springer Berlin / Heidelberg, February 1988. Cited on page 38.

Pascal Lienhardt. Topological models for boundary representation: a comparison with n-dimensional generalized maps. *Computer-Aided Design*, 23(1):59–82, 1991. Cited on pages 26 and 39.

Pascal Lienhardt. n-dimensional generalized combinatorial maps and cellular quasi-manifolds. *International Journal of Computational Geometry and Applications*, 4(3):275–324, 1994. Cited on pages 5, 45, 76, 77, 188, and 189.

Pascal Lienhardt, Xavier Skapin, and Antoine Bergey. Cartesian product of simplicial and cellular structures. *International Journal of Computational Geometry and Applications*, 14(3):115–159, 2004. Cited on pages 103 and 104.

Pascal Lienhardt, Laurent Fuchs, and Yves Bertrand. *Combinatorial Models for Topology-Based Geometric Modeling*, pages 151–198. Theory and applications of proximity, nearness and uniformity. Dipartimento di Matematica della Seconda Università di Napoli, 2009. Cited on page 26.

Peter Lindstrom and Greg Turk. Fast and memory efficient polygonal simplification. In *VIS '98 Proceedings of the conference on Visualization '98*, pages 279–286. IEEE Computer Society Press, 1998. Cited on page 49.

Liu Liu and Sisi Zlatanova. Generating navigation models from existing building data. In *ISPRS Acquisition and Modelling of Indoor and Enclosed Environments 2013*, volume XL-4/W4. ISPRS, 2013. Cited on page 96.

Yuanxin Liu and Jack Snoeyink. Faraway point: A sentinel point for Delaunay computation. *International Journal of Computational Geometry and Applications*, 18(4):343–355, 2008. Cited on page 68.

Nikolai Lobachevsky. *Geometrische Untersuchungen zur Theorie der Parallellinien*. Funcke, Berlin, 1840. Cited on page 13.

Hélio Lopes and Geovan Tavares. Structural operators for modeling 3-manifolds. In Christoph Hoffmann, Wim Bronsvoort, George Allen, Mike Pratt, and David Rosen, editors, *SMA '97, Proceedings of the Fourth Symposium on Solid Modeling and Applications*, pages 10–18. ACM, 1997. Cited on page 73.

María Lorenzo-Valdés, Gerardo I. Sanchez-Ortiz, Andrew G. Elkington, Raad H. Mohiaddin, and Daniel Rueckert. Segmentation of 4D cardiac MR images using a probabilistic atlas and the EM algorithm. *Medical Image Analysis*, 8:255–265, 2004. Cited on page 64.

David Luebke, Martin Reddy, Jonathan D. Cohen, Amitabh Varshney, Benjamin Watson, and Robert Huebner. *Level of Detail for 3D Graphics*. Morgan Kaufmann Publishers, 2003. Cited on page 48.

Yannis Manolopoulos, Alexandros Nanopoulos, Apostolos N. Papadopoulos, and Yannis Theodoridis. *R-Trees: Theory and Applications*. Advanced Information and Knowledge Processing. Springer Berlin Heidelberg, 2006. Cited on page 31.

Martti Mäntylä. *An introduction to solid modeling*. Computer Science Press, New York, USA, 1988. Cited on pages 24, 25, 26, 40, 66, 81, 132, and 189.

John I. Marden. *Analyzing and Modeling Rank Data*, volume 64 of *CRC Monographs on Statistics & Applied Probability*. Chapman & Hall, August 1996. Cited on page 91.

Avraham Margalit and Gary D. Knott. An algorithm for computing the union, intersection or difference of two polygons. *Computers & Graphics*, 13(2):167–183, 1989. Cited on page 168.

N. C. Mason, M. A. O'Conaill, and S. B. M. Bell. Handling four-dimensional geo-referenced data in environmental GIS. *International Journal of Geographical Information Systems*, 8(2):191–215, 1994. Cited on pages 64, 65, and 80.

W. S. Massey. Cross products of vectors in higher dimensional Euclidean spaces. *The American Mathematical Monthly*, 90(10):697–701, December 1983. Cited on page 93.

Hiroshi Masuda. Topological operators and Boolean operations for complex-based non-manifold geometric models. *Computer-Aided Design*, 25(2), 1993. Cited on page 71.

Dan McKenney. Model quality: The key to CAD/CAM/CAE interoperability. Technical report, International TechneGroup Incorporated, 1998. Cited on page 165.

John W. McKenzie, Ian P. Williamson, and N.W.J. Hazelton. 4-D adaptive GIS: Justification and methodologies. Technical report, Department of Geomatics, The University of Melbourne, 2001. Cited on pages 26, 54, and 80.

Donald Meagher. Octree encoding: a new technique for the representation, manipulation and display of arbitrary 3-d objects by computer. Technical report, Rensselaer Polytechnic Institute, 1980. Cited on pages 31, 65, and 94.

Martijn Meijers. *Variable-scale Geo-information*. PhD thesis, Delft University of Technology, December 2011. Cited on pages 48, 59, 131, 132, and 133.

Albrecht Ludwig Friedrich Meister. Generalia de genesi figurarum planarum et inde pendentibus earum affectionibus. *Novi Commentarii Societatis Reglae Scientiarum Gottingensis*, 1:144–180, 1771. Cited on page 76.

Liqiu Meng and Andrea Forberg. 3D building generalisation. In W.A. Mackaness, A. Ruas, and L.T. Sarajkoski, editors, *Generalisation of Geographic Information: Cartographic Modelling and Applications*, pages 211–232. Elsevier, 2007. Cited on page 49.

Bertrand Meyer. Design by contract. Technical Report TR-EI-12/CO, Interactive Software Engineering, 1986. Cited on page 165.

Eric J. Miller. Towards a 4D GIS: Four-dimensional interpolation utilizing Kriging. In Zarine Kemp, editor, *Innovations in GIS*, chapter 13, pages 181–197. Taylor & Francis, 1997. Cited on page 79.

Hermann Minkowski. Die Grundgleichungen für die elektromagnetischen Vorgänge in bewegten Körpern. *Nachrichten von der Gesellschaft der Wissenschaften zu Göttingen, Mathematisch-Physikalische Klasse*, pages 53–111, 1908. Cited on page 55.

A. Morin, J. Urban, I. Foster, A. Sali, D. Baker, and P. Sliz. Shining light into black boxes. *Science*, 336:159–160, April 2012. Cited on page 2.

Ernst P. Mücke, Isaac Saias, and Binhai Zhu. Fast randomized point location without preprocessing in two- and three-dimensional Delaunay triangulations. *Computational Geometry—Theory and Applications*, 12:63–83, 1999. Cited on page 169.

D. E. Muller and F. P. Preparata. Finding the intersection of two convex polyhedra. *Theoretical Computer Science*, 7(2):217–236, 1978. Cited on pages 37, 50, and 77.

Nathan C. Myers. Traits: a new and useful template technique. *C++ Report*, June 1995. Cited on pages 113 and 200.

B. Naylor. Binary space partitioning trees as an alternative representation of polytopes. *Computer-Aided Design*, 22(4), 1990. Cited on page 28.

Walter Nef. *Beiträge zur Theorie der Polyeder: mit Anwendungen in der Computergraphik*. Herbert Lang, Bern, 1978. Cited on pages 29 and 89.

F.S. Nooruddin and G. Turk. Simplification and repair of polygonal models using volumetric techniques. *IEEE Transactions on Visualization and Computer Graphics*, 9(2):191–205, 2003. Cited on page 178.

M. A. O'Conaill, S. B. M. Bell, and N. C. Mason. Developing a prototype 4D GIS on a transputer array. *ITC Journal*, 1:47–54, 1992. Cited on pages 65 and 80.

OGC. *OpenGIS Geography Markup Language (GML) Encoding Standard. Version 3.2.1*. Open Geospatial Consortium, August 2007. Cited on pages 43 and 174.

OGC. *OpenGIS Implementation Specification for Geographic Information - Simple Feature Access - Part 1: Common Architecture. Version 1.2.1*. Open Geospatial Consortium, May 2011. Cited on pages 3, 41, 42, 50, 59, 60, 78, 81, 97, 166, 182, 189, and 198.

OGC. *OGC Geography Markup Language — Extended schemas and encoding rules. Version 3.3.0*. Open Geospatial Consortium, December 2012. Cited on pages 41 and 182.

Narayan Panigrahi. *Computing in Geographic Information Systems*. CRC Press, 2014. Cited on page 165.

A. Paoluzzi, F. Bernardini, C. Cattani, and V. Ferrucci. Dimension-independent modeling with simplicial complexes. *ACM Transactions on Graphics*, 12(1):56–102, January 1993. Cited on page 70.

A. Paoluzzi, V. Pascucci, and G. Scorzelli. Progressive dimension-independent Boolean operations. In G. Elber, N. Patrikalakis, and P. Brunet, editors, *ACM Symposium on Solid Modeling and Applications*. ACM, 2004. Cited on page 65.

A. Pasko, V. Adzhiev, A. Sourin, and V. Savchenko. Function representation in geometric modeling: concepts, implementation and applications. *The Visual Computer*, 11:429–446, 1995. Cited on pages 29 and 80.

Alexander Pasko and Valery Adzhiev. Constructive hypervolume modeling. *Graphical Models*, 63:413–442, 2001. Cited on page 79.

Tom Patterson. Getting real: Reflecting on the new look of national park service maps. *Cartographic Perspectives*, 43, Fall 2002. Cited on page 154.

Norbert Paul, Patrick Erik Bradley, and Martin Breunig. Integrating space, time, version and scale using Alexandrov topologies. Available at `http://arxiv.org/abs/1303.2595`, March 2011. Cited on page 137.

Nikos Pelekis, Babis Theodoulidis, Ioannis Kopanakis, and Yannis Theodoridis. Literature review of spatio-temporal database models. *The Knowledge Engineering Review*, 19(3):235–274, September 2004. Cited on pages 25 and 46.

Sinésio Pesco, Geovan Tavares, and Hélio Lopes. A stratification approach for modeling two-dimensional cell complexes. *Computers & Graphics*, 28:235–247, 2004. Cited on page 73.

Thomas K. Peucker and Nicholas R. Chrisman. Cartographic data strutures. *The American Cartographer*, 2(1): 55–69, 1975. Cited on pages 4, 36, 46, and 51.

Donna J. Peuquet. A conceptual framework and comparison of spatial data models. *Cartographica*, 21(4): 66–113, 1984. Cited on page 25.

Donna J. Peuquet. It's about time: A conceptual framework for the representation of temporal dynamics in geographic information systems. *Annals of the Association of American Geographers*, 84(3):441–461, 1994. Cited on pages 4, 46, and 51.

Donna J. Peuquet and Niu Duan. An event-based spatiotemporal data model (ESTDM) for temporal analysis of geographical data. *International Journal of Geographical Information Science*, 9(1):7–24, 1995. Cited on pages 46 and 47.

Sylvain Pion and Andreas Fabri. A generic lazy evaluation scheme for exact geometric computations. *Science of Computer Programming*, 76(4):307–323, 2011. Cited on page 199.

Lutz Plümer and Gerhard Gröger. Achieving integrity in geographic information systems—maps and nested maps. *GeoInformatica*, 1(4):345–367, 1997. Cited on pages 50, 137, and 168.

Joshua Podolak and Szymon Rusinkiewicz. Atomic volumes for mesh completion. In M. Desbrunn and H. Pottmann, editors, *Eurographics Symposium on Geometry Processing (2005)*, pages 33–42, 2005. Cited on page 178.

H. Poincaré. Complément à l'analysis situs. *Rendiconti del Circolo Matematico di Palermo*, 13:285–343, 1899. Cited on page 20.

H. Poincaré. Second complément à l'analysis situs. *Proceedings of the London Mathematical Society*, 32:277–308, 1900. Cited on page 20.

Henri Poincaré. Sur la généralisation d'un théorème d'Euler relatif aux polyèdres. *Comptes rendus hebdomadaires de l'Académie des sciences de Paris*, 117:144–145, 1893. Cited on page 20.

M.H. Poincaré. Analysis situs. *Journal de l'École polytechnique*, 2(1):1–123, 1895. Cited on pages 5, 18, 20, and 54.

Jovan Popović and Hughes Hoppe. Progressive simplicial complexes. In G. Scott Owen, Turner Whitted, and Barbara Mones-Hattal, editors, *SIGGRAPH '97 Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 217–224. ACM/Addison-Wesley Publishing, 1997. Cited on page 36.

M. Poudret, A. Arnould, Y. Bertrand, and P. Lienhardt. Cartes combinatoires ouvertes. Technical Report 2007-01, Laboratoire SIC, UFR SFA, Université de Poitiers, October 2007. Cited on page 77.

F.P. Preparata and D.E. Muller. Finding the intersection of n half-spaces in time $o(n \log n)$. *Theoretical Computer Science*, 8(1):45–55, 1979. Cited on page 28.

Paul Ramsey. PostGIS: Tips for power users. Presentation at FOSS4G 2010. Available at http://2010.foss4g.org/presentations/3369.pdf, 2010. Cited on page 168.

Jonathan Raper, editor. *Three Dimensional Applications in Geographic Information Systems*. Taylor & Francis, 1989. Cited on pages 4, 44, and 51.

Jonathan Raper. *Multidimensional geographic information science*. Taylor & Francis, 2000. Cited on pages 51 and 55.

Helmut Ratschek and Jon Rokne. *New computer methods for global optimization*. Ellis Horwood, 1988. Cited on page 199.

Julia Reda. Draft report on the implementation of Directive 2001/29/EC of the European Parliament and of the Council of 22 May 2001 on the harmonisation of certain aspects of copyright and related rights in the information society. Technical report, European Parliament Commitee on Legal Affairs, 2014. Cited on page 2.

A. Renolen. History graphs: conceptual modeling of spatio-temporal data. *GIS Frontiers in Business and Science*, 2, 1996. Cited on page 46.

A. A. G. Requicha and H. B. Voelcker. Constructive solid geometry. Technical Memorandum 25, College of Engineering & Applied Science, The University of Rochester, November 1977. Cited on page 28.

Aristides A. G. Requicha. Representations for rigid solids: Theory, methods, and systems. *Computing Surveys*, 12(4):437–464, December 1980. Cited on page 25.

Aristides A. G. Requicha and Robert B. Tilove. Mathematical foundations of constructive solid geometry: General topology of closed regular sets. Production Automation Project Technical Memorandum 27, University of Rochester, 1978. Cited on page 28.

B. Riemann. *Ueber die Hypothesen, welche der Geometrie zu Grunde liegen*. PhD thesis, Abhandlungen der Königlichen Gesellschaft der Wissenschaften zu Göttingen, 1868. Cited on pages 5, 54, and 55.

Philippe Rigaux and Michel Scholl. Multi-scale partitions: Application to spatial and statistical databases. In Max J. Egenhofer and John R. Herring, editors, *Advances in Spatial Databases*, volume 951 of *Lecture Notes in Computer Science*, pages 170–183. Springer Berlin Heidelberg, 1995. Cited on pages 50 and 137.

M. Rivero and F.R. Feito. Boolean operations on general planar polygons. *Computers & Graphics*, 24:881–896, 2000. Cited on page 168.

Stephen J. Rock and Michael J. Wozny. Generating topological information from a "bucket of facets". In *Solid Freeform Fabrication Symposium Proceedings*, pages 251–259, 1992. Cited on page 178.

J. Rossignac and M. O'Connor. SGC: A dimension-independent model for pointsets with internal structures and incomplete boundaries. In M. Wosny, J. Turner, and K. Preiss, editors, *Proceedings of the IFIP Workshop on CAD/CAM*, pages 145–180, 1989. Cited on pages 43, 46, 62, 71, 90, and 188.

Jarek Rossignac and David Cardoze. Matchmaker: Manifold BReps for non-manifold r-sets. Technical Report SM99-020, Georgia Institute of Technology, 1999. Cited on page 177.

Jarek Rossignac and Andrzej Szymczak. Wrap&Zip decompression of the connectivity of triangle meshes compressed with Edgebreaker. *Computational Geometry: Theory & Applications*, 14, 1999. Cited on page 36.

Yossi Rubner, Carlo Tomasi, and Leonidas J. Guibas. A metric for distributions with applications to image databases. In *Proceedings of the 6th International Conference on Computer Vision*, pages 59–66, 1998. Cited on page 139.

Jim Ruppert and Raimund Seidel. On the difficulty of triangulating three-dimensional nonconvex polyhedra. *Discrete & Computational Geometry*, 7(1):227–253, 1992. Cited on pages 67 and 68.

V. L. Rvachev. On the analytical description of certain geometrical objects. *Doklady Akademii Nauk SSSR*, 153 (4):765–767, 1963. Cited on page 30.

Hans Sagan. *Space-Filling Curves*. Springer Science+Business Media, 1994. Cited on page 31.

David Salomon. *The Computer Graphics Manual*, volume 2 of *Texts in Computer Science*. Springer London, 2011. Cited on page 157.

Hanan Samet and Markku Tamminen. Bintrees, CSG trees, and time. In Pat Cole, Robert Heilman, and Brian A. Barsky, editors, *SIGGRAPH '85*, volume 19, pages 121–130. ACM, 1985. Cited on pages 31 and 65.

Stefan Schirra. Precision and robustness in geometric computations. In Marc van Kreveld, Jürg Nievergelt, Thomas Roos, and Peter Widmayer, editors, *Algorithmic Foundations of Geographic Information Systems*, volume 1340 of *Lecture Notes in Computer Science*, pages 255–287. Springer Berlin Heidelberg, 1997. Cited on page 44.

E. Schönhardt. Über die Zerlegung von Dreieckspolyedern in Tetraeder. *Mathematische Annalen*, 1:309–312, 1928. Cited on page 67.

Michael Seel. *Planar Nef Polyhedra and Generic Higher-dimensional Geometry*. PhD thesis, Saarland University, 2001. Cited on page 89.

Raimund Seidel. Constructing higher-dimensional convex hulls at logarithmic cost per face. In *Proceedings of the 18th Annual Symposium on the Theory of Computing*, pages 404–413, 1986. Cited on page 80.

Timos Sellis, Nick Roussopoulos, and Christos Faloutsos. The R+-tree: A dynamic index for multi-dimensional objects. In *Proceedings of 13th International Conference on Very Large Data Bases*, 1987. Cited on page 94.

SGK. *Kartographische Generalisierung*. Der Schweizerischen Gesellschaft für Kartographie, 1975. Cited on page 49.

Michael Ian Shamos and Dan Hoey. Geometric intersection problems. In *Proceedings of the 17th Annual Symposium on the Foundations of Computer Science*, pages 208–215, 1976. Cited on page 28.

Timothy M. Shead. Universal relational storage for geometric primitives. Technical report, The K-3D Project, 2010. Cited on page 45.

Xuejun Sheng and Ingo R. Meier. Generating topological structures for surface models. *IEEE Computer Graphics and Applications*, November 1995. Cited on page 178.

Jonathan Richard Shewchuk. Triangle: Engineering a 2D quality mesh generator and Delaunay triangulator. In Ming C. Lin and Dinesh Manocha, editors, *Applied Computational Geometry: Towards Geometric Engineering*, volume 1148 of *Lecture Notes in Computer Science*, pages 203–222. Springer-Verlag, May 1996a. Cited on pages 32, 68, and 194.

Jonathan Richard Shewchuk. Robust adaptive floating-point geometric predicates. In *Proceedings of the 12th Annual Symposium on Computational Geometry*, pages 141–150, 1996b. Cited on page 169.

Jonathan Richard Shewchuk. Adaptive precision floating-point arithmetic and fast robust geometric predicates. *Discrete & Computational Geometry*, 18:305–363, 1997. Cited on pages 169 and 199.

Jonathan Richard Shewchuk. Sweep algorithms for constructing higher-dimensional constrained Delaunay triangulations. In *Proceedings of the 16th Annual Symposium on Computational Geometry*, pages 350–359, 2000. Cited on page 67.

Jonathan Richard Shewchuk. General-dimensional constrained Delaunay and constrained regular triangulations, I: Combinatorial properties. *Discrete & Computational Geometry*, 39(1003):580–637, March 2008. Cited on pages 67 and 194.

Dave Shreiner, Graham Sellers, John Kessenich, Bill Licea-Kane, and Khronos ARB Working Group. *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 4.3*. Addison-Wesley, 8th edition, 2013. Cited on page 155.

Hang Si and Klaus Gärtner. Meshing piecewise linear complexes by constrained Delaunay tetrahedralizations. In *Proceedings of the 14th International Meshing Roundtable*, September 2005. Cited on pages 32, 68, 178, and 194.

Cathy Sohanpanah. Extension of a boundary representation technique for the description of n dimensional polytopes. *Computers & Graphics*, 13(1):17–23, 1989. Cited on page 71.

Willi-Hans Steeb. *The Nonlinear Workbook*. World Scientific Publishing, 5th edition, 2011. Cited on page 159.

S. S. Stevens. On the theory of scales of measurement. *Science*, 103(2684):677–680, 1946. Cited on page 57.

P. A. Story and Michael F. Worboys. A design support environment for spatio-temporal database applications. In Andrew U. Frank and Werner Kuhn, editors, *Spatial Information Theory: A Theoretical Basis for GIS*, volume 988 of *Lecture Notes in Computer Science*, pages 413–430. Springer Berlin / Heidelberg, 1995. Cited on page 47.

Jantien Stoter, Dirk Burghardt, Cécile Duchêne, Blanca Baella, Nico Bakker, Connie Blok, Maria Pla, Nicolas Regnauld, Guillaume Touya, and Stefan Schmid. Methodology for evaluating automated map generalization in commercial software. *Computers, Environment and Urban Systems*, 33(5):311–324, 2009. Cited on page 49.

Jantien Stoter, Hugo Ledoux, Martijn Meijers, and Ken Arroyo Ohori. Integrating scale and space in 3D city models. In Jacynthe Pouliot, Sylvie Daniel, Frédéric Hubert, and Alborz Zamyadi, editors, *Proceedings of the 7th International 3D GeoInfo Conference*, volume XXXVIII-4/C26 of *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, pages 7–10, Québec City, Canada, May 2012a. ISPRS. Cited on pages 59 and 137.

Jantien Stoter, Hugo Ledoux, Martijn Meijers, Ken Arroyo Ohori, and Peter van Oosterom. 5D modeling - applications and advantages. In *Geospatial World Forum 2012*, Amsterdam, The Netherlands, April 2012b. Cited on page 59.

J.E. Stoter, M. Post, V. van Altena, R. Nijhuis, and B. Bruns. Fully automated generalisation of a 1:50k map from 1:10k data. *Cartography and Geographic Information Science*, 41(1):1–13, January 2014. Cited on pages 4 and 49.

Wolfgang Straßer. *Schnelle Kurven- und Flächendarstellung auf graphischen Sichtgeräten*. PhD thesis, Technische Universität Berlin, 1974. Cited on page 41.

Jeremy Tammik. AutoCAD Nef polyhedron implementation. Available at `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.89.6020&rep=rep1&type=pdf`, September 2007. Cited on page 29.

R.J. Thompson. *Towards a rigorous logic for spatial data representation*. PhD thesis, Delft University of Technology, 2007. Cited on page 28.

Andreas Thomsen, Martin Breunig, and Edgar Butwilowski. Towards a G-map based tool for the modelling and management of topology in multiple representation databases. *Journal of Photogrammetry, Remote Sensing and Geoinformation Processing*, 3:175–186, 2008. Cited on page 79.

F. Töfper and W. Pillerwizer. The principles of selection. *The Cartographic Journal*, 3(1):10–16, May 1966. Cited on page 49.

R.F. Tomlinson. The impact of the transition from analogue to digital cartographic representation. *The American Cartographer*, 15(3):249–261, July 1988. Cited on page 43.

Nectaria Tryfona and Christian S. Jensen. Conceptual data modeling for spatiotemporal applications. *GeoInformatica*, 3(3):245–268, 1999. Cited on page 47.

Greg Turk and Marc Levoy. Zippered polygon meshes from range images. In *SIGGRAPH '94 Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 311–318. ACM, New York, NY, USA, 1994. Cited on page 178.

L. Untereiner, P. Kraemer, D. Cazier, and D. Bechmann. CPH: A compact representation for hierarchical meshes generated by primal refinement. *Computer Graphics Forum*, 2015. Cited on page 55.

Dirk van Dalen. *L.E.J. Brouwer — Topologist, Intuitionist, Philosopher*. Springer Science+Business Media, 2013. Cited on page 33.

L. van Elfrinkhof. Eene eigenschap van de orthogonale substitutie van de vierde orde. In *Handelingen van het 6e Nederlandsch Natuurkundig en Geneeskundig Congres*, pages 237–240, Delft, 1897. Cited on page 93.

Marc van Kreveld. Digital elevation models and TIN algorithms. In Marc van Kreveld, Jürg Nievergelt, Thomas Roos, and Peter Widmayer, editors, *Algorithmic Foundations of Geographic Information Systems*, volume 1340 of *Lecture Notes in Computer Science*, chapter 1, pages 37–78. Springer-Verlag, Berlin, 1997a. Cited on page 4.

Marc van Kreveld. Algorithms for triangulated terrains. In *Proceedings of the 24th International Conference on Current Trends in Theory and Practice of Computer Science*, volume 1338 of *Lecture Notes in Computer Science*, pages 19–36. Springer-Verlag, 1997b. Cited on page 25.

Peter van Oosterom. *Reactive Data Structures for Geographic Information Systems*. PhD thesis, Leiden University, 1990. Cited on page 49.

Peter van Oosterom. Maintaining consistent topology including historical data in a large spatial database. In *Proceedings of Auto-Carto 13*, 1997. Cited on page 47.

Peter van Oosterom. Spatial access methods. In Paul A. Longley, Michael F. Goodchild, David J. Maguire, and David W. Rhind, editors, *Geographical Information Systems: Principlesm Technical Issues, Management Issues and Applications*, volume 1, chapter 2.3, pages 385–400. Wiley, 1999. Cited on page 94.

Peter van Oosterom. Variable-scale topological data structures suitable for progressive data transfer: The GAP-face tree and GAP-edge forest. *Cartography and Geographic Information Science*, 32(4):331–346, 2005. Cited on pages 50 and 137.

Peter van Oosterom and Martijn Meijers. Towards a true vario-scale structure supporting smooth-zoom. In *Proceedings of the 14th ICA/ISPRS Workshop on Generalisation and Multiple Representation, Paris*, 2011. Cited on page 59.

Peter van Oosterom and Martijn Meijers. Vario-scale data structures supporting smooth zoom and progressive transfer of 2D and 3D data. *International Journal of Geographical Information Science*, 28:455–478, 2014. Cited on page 55.

Peter van Oosterom and Jantien Stoter. 5D data modelling: Full integration of 2D/3D space, time and scale dimensions. In Sara Irina Fabrikant, Tumasch Reichenbacher, Marc van Kreveld, and Christoph Schlieder, editors, *Geographic Information Science: 6th International Conference, GIScience 2010, Zurich, Switzerland, September 14-17, 2010. Proceedings*, pages 311–324. Springer Berlin Heidelberg, 2010. Cited on pages 5, 54, 55, 80, 136, 137, and 192.

Herman Varma, H. Boudreau, and W. Prime. A data structure for spatio-temporal databases. In *Proceedings of the IHO Review*, pages 1–10, 1990. Cited on pages 65 and 80.

Todd L. Veldhuizen. C++ templates are turing complete. Available at `http://ubietylab.net/ubigraph/content/Papers/pdf/CppTuring.pdf`, 2003. Cited on page 201.

Luiz Velho. Stellar subdivision grammars. In L. Kobbelt, P. Schröder, and H. Hoppe, editors, *Eurographics Symposium on Geometry Processing*. The Eurographics Association, 2003. Cited on page 81.

Remco C. Veltkamp and Michiel Hagedoorn. State of the art in shape matching. In *Principles of Visual Information Retrieval*, Advances in Pattern Recognition, pages 87–119. Springer London, 2001. Cited on page 137.

Kenneth James Versprille. *Computer-aided design applications of the rational b-spline approximation form*. PhD thesis, Syracuse University, 1975. Cited on page 40.

Georges Voronoi. Nouvelles applications des paramètres continus à la théorie des formes quadratiques. Premier mémoire. Sur quelques propriétés des formes quadratiques positives parfaites. *Journal für die Reine und Angewandte Mathematik*, 133:97–178, 1908. Cited on page 32.

Detlev Wagner, Mark Wewetzer, Jürgen Bogdahn, Nazmul Alam, Margitta Pries, and Volker Coors. Geometric-semantical consistency validation of CityGML models. In J. Pouliot, S. Daniel, F. Hubert, and A. Zamyadi, editors, *Progress and New Trends in 3D Geoinformation Sciences*, Lecture Notes in Geoinformation and Cartography. Springer Berlin Heidelberg, 2013. Cited on page 177.

Jianning Wang and Manuel M. Oliveira. Filling holes on locally smooth surfaces reconstructed from point clouds. *Image and Vision Computing*, 25(1):103–113, 2007. Cited on page 178.

Robert Weber, Hans-J. Schek, and Stephen Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *Proceedings of the 24th VLDB Conference*, pages 194–205. ACM, 1998. Cited on page 94.

Robert Weibel. Generalization of spatial data: Principles and selected algorithms. In Marc van Kreveld, Jürg Nievergelt, Thomas Roos, and Peter Widmayer, editors, *Algorithmic Foundations of Geographic Information Systems*, volume 1340 of *Lecture Notes in Computer Science*, pages 99–152. Springer Berlin Heidelberg, 1997. Cited on pages 49 and 136.

Kevin Weiler. The radial edge data structure: a topological representation for non-manifold geometric boundary modeling. In M.J. Wozny and H.W. McLaughlin, editors, *Geometric modeling for CAD applications: selected and expanded papers from the IFIP WG 5.2 working conference*, pages 3–36. Elsevier, May 1988. Cited on pages 38, 45, and 73.

John D. Weld and Ming C. Leu. Geometric representation of swept volumes with application to polyhedral objects. *The International Journal of Robotics Research*, 9(5):105–117, 1990. Cited on page 28.

Haim J. Wolfson and Isidore Rigoutsos. Geometric hashing: An overview. *IEEE Computational Science and Engineering*, 4(4):10–21, October–December 1997. Cited on page 94.

M.F. Worboys. A model for spatio-temporal information. In *Proceedings of the 5th International Symposium on Spatial Data Handling*, pages 602–611, 1992a. Cited on pages 4, 46, and 51.

Michael Worboys. The maptree: A fine-grained formal representation of space. In Ningchuan Xiao, Mei-Po Kwan, Michael F. Goodchild, and Shashi Shekhar, editors, *Proceedings of the 7th International Conference GIScience 2012*, volume 7478 of *Lecture Notes in Computer Science*, pages 298–310. Springer, 2012. Cited on page 74.

Michael Worboys and Matt Duckham. *GIS: A Computational Perspective*. CRC Press, 2nd edition, 2004. Cited on pages 16 and 18.

Michael F. Worboys. A generic model for planar geographical objects. *International Journal of Geographical Information Systems*, 6(5):353–372, 1992b. Cited on page 62.

Michael F. Worboys, Hilary M. Hearnshaw, and David J. Maguire. Object-oriented data modelling for spatial databases. *International Journal of Geographical Information Systems*, 4(4):369–383, 1990. Cited on page 47.

Mann-May Yau and Sargur N. Srihari. A hierarchical data structure for multidimensional digital images. *Communications of the ACM*, 26(7):504–515, 1983. Cited on pages 65 and 94.

May Yuan. Wildfire conceptual modeling for building GIS space-time models. In *Proceedings of GIS/LIS'94*, pages 860–869, 1994. Cited on page 47.

Hui Zhang and Andrew J. Hanson. Shadow-driven 4D haptic visualization. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1688–1695, November/December 2007. Cited on page 162.

Xiang Zhang, Tinghua Ai, Jantien Stoter, and Xi Zhao. Data matching of building polygons at multiple map scales improved by contextual information and relaxation. *ISPRS Journal of Photogrammetry and Remote Sensing*, 92:147–163, June 2014. Cited on page 137.

Yeting Zhang, Qing Zhu, Gang Liu, Wenting Zheng, Zhonghua Li, and Zhiqiang Du. GeoScope: Full 3D geospatial information system case study. *Geo-Spatial Information Science*, 14(2):150–156, 2011. Cited on page 44.

Junqiao Zhao, Qing Zhu, Zhiqiang Du, Tiantian Feng, and Yeting Zhang. Mathematical morphology-based generalization of complex 3D building models incorporating semantic relationships. *ISPRS Journal of Photogrammetry and Remote Sensing*, 68:95–111, March 2012. Cited on page 49.

Junqiao Zhao, Jantien Stoter, and Hugo Ledoux. A framework for the automatic geometric repair of CityGML models. In Manfred Buchroithner, Nikolas Prechtel, and Dirk Burghardt, editors, *Cartography from Pole to Pole: Selected Contributions to the XXVIth International Conference of the ICA, Dresden 2013*, Lecture Notes in Geoinformation and Cartography, pages 187–202. Springer Berlin Heidelberg, 2014. Cited on pages 44 and 178.

Qing Zhu, Junqiao Zhao, Xiaochun Liu, and Yeting Zhang. Perceptually guided geometrical primitive location method for 3D complex building simplification. In Thomas H. Kolbe, Hao Zhang, and Sisi Zlatanova, editors, *Proceedings of GeoWeb 2009 Academic Track - Cityscapes*, volume XXXVIII-3-4/C3 of *The International Archives of Photogrammetry, Remote Sensing and Spatial Information Science*, pages 74–79. ISPRS, 2009. Cited on page 49.

Siyka Zlatanova. *3D GIS for Urban Development*. PhD thesis, Technische Universität Graz, March 2000. Cited on page 4.

Siyka Zlatanova and Daniel Holweg. 3D geo-information in emergency response: A framework. In *Proceedings of the 4th International Symposium on Mobile Mapping Technology*, 2004. Cited on page 136.

Summary

Our world is three-dimensional and complex, continuously changing over time and appearing different at different scales. Yet, when we model it in a computer using Geographic Information Systems (GIS), we mostly use 2D representations, which essentially consist of linked points, lines and polygons (Figure 1). These representations are relatively easy to use and efficient, and a wide variety of methods is built on top of them. However, 2D representations are necessarily limiting. They force us to reduce problems to two dimensions, limit the type of objects we can represent, and complicate storing the relationships between different objects—especially when these are across time and different scales. Nevertheless, most research in GIS is devoted to improving these 2D representations, as well as to the development of new methods that build on them to solve problems, both old and new.

This thesis explores a new, fundamentally different modelling approach—integrating both spatial and non-spatial characteristics as dimensions in the geometric sense, specifically targeting the cases of time and scale (Figure 2). While this has been proposed before at a conceptual level, this thesis aims to *realise the fundamental aspects of a higher-dimensional GIS* by developing higher-dimensional (nD) representations, as well as new methods operating on them to create, manipulate and visualise geographic information. As this thesis shows, the higher-dimensional approach is undoubtedly memory-intensive, but it is also very powerful, as it provides a simple and consistent way to store geometry, attributes and the topological relationships between objects of any dimension. This generic approach can also be easily extended to handle other non-spatial characteristics, enabling better data management that is consistent across dimensions and more powerful operations, such as checking if two objects are adjacent at any point in time.

In order to model higher-dimensional space, it is best to consider an nD space subdivision as a base (Figure 3), which is conceptualised as an n-dimensional simplicial complex or cell complex. This can then be implemented with a simplex-based data structure, with an incidence graph, as a set of Nef polyhedra, or—as done in this thesis—by using ordered topological models such as the cell-tuple and generalised/combinatorial maps.

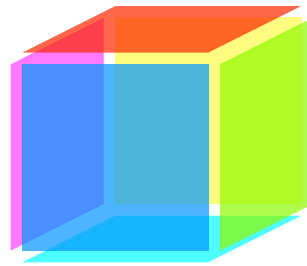


Figure 1: In GIS, a cube is not represented as a 3D solid, but as the 6 square 2D faces that bound it.

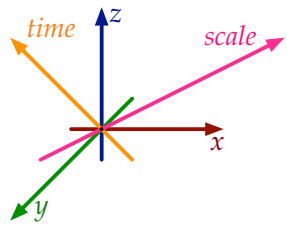


Figure 2: 3D space, time and scale can be modelled as 5D space.

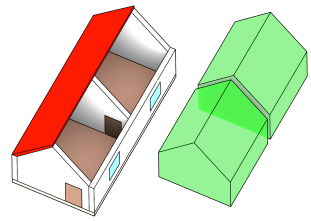
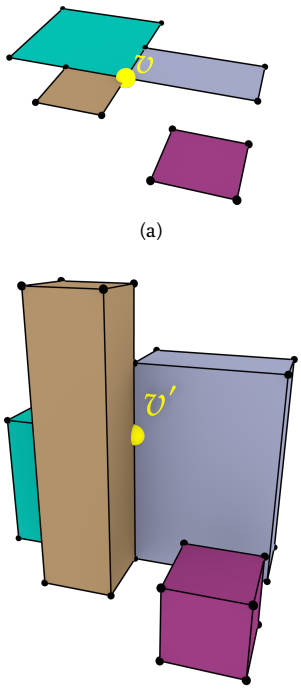


Figure 3: A 3D space subdivision model is composed of a set of space-filling volumes without gaps or overlaps.

Creating computer representations of higher-dimensional objects can be complex. Common construction methods used in 2D and 3D, such as directly manipulating combinatorial primitives, or using primitive-level construction operations (such as Euler operators), rely on our intuition of 2D/3D geometry, and thus do not work well in higher dimensions. It is therefore all too easy to create invalid objects, which then cannot be easily interpreted or fixed—a problem that is already exceedingly apparent in three dimensions.



(a)

(b)

Figure 4: (a) A set of polygons is converted into (b) a set of boxes by 2D-to-3D extrusion.

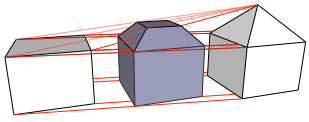


Figure 5: Two LODs of a 3D model of a house (left and right) are linked into a 4D model.

As a way to easily create representations of higher-dimensional objects, this thesis proposes three novel higher-level methods, all of which are intuitive to use and attempt to create valid output. *Extrusion* takes an $(n-1)$-dimensional cell complex and a set of intervals per cell, projecting them parallel to a new axis in order to create an n-dimensional cell complex (Figure 4). *Incremental construction* describes an n-dimensional object based on its $(n-1)$-dimensional boundary, from dimension zero (points) and then upwards. Finally, a 4D model can be constructed from a series of 3D models at different levels of detail (LODs) by *linking them* (Figure 5).

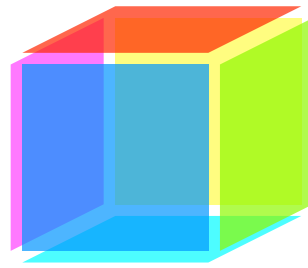
In order to visualise higher-dimensional models, as well as to be able to process them in existing software, it is important to have methods to extract meaningful 2D/3D subsets from them. As a stepping stone towards such methods, this thesis shows how n-dimensional to $(n-1)$-dimensional orthographic and perspective projections can be defined.

Finally, this thesis placed an emphasis on validating the algorithms with real-world datasets, which was only possible by developing methods to repair the invalid datasets that are widespread in practice. This thesis thus contains methods to create valid polygons and planar partitions using a constrained triangulation of the input, as well as a method to repair polyhedra and space subdivisions by snapping together lower-dimensional primitives and removing overlaps using Boolean set operations on Nef polyhedra. This allowed tests with up to 6D datasets based on real-world data—*a good base for higher-dimensional GIS*.

In the future, the work in this thesis will be extended with higher-dimensional modification operations, true 4D spatiotemporal datasets and repair methods with quality guarantees. All implementations made for this thesis are publicly available under open source licences.

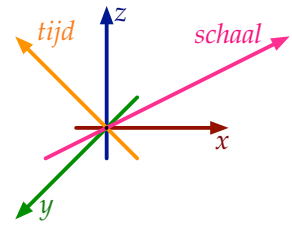
Samenvatting (Dutch summary)

Onze wereld is driedimensionaal en complex, is continue in verandering en heeft verschillende verschijningsvormen op verschillende detailniveaus. Toch modelleren we de werkelijkheid in Geografische Informatie Systemen (GIS) meestal middels 2D representaties. Deze modellen bestaan uit punten, lijnen en polygonen die met elkaar zijn verbonden (Figuur 1). De resulterende representaties zijn efficiënt en relatief eenvoudig te gebruiken en veel methodes en applicaties zijn hierop gebaseerd. Maar 2D representaties kennen hun beperkingen. Problemen moeten worden versimpeld tot 2D; en het beheren van relaties tussen verschillende objecten is complex, vooral als deze relaties een tijdscomponent hebben of over verschillende detail-niveaus gaan. Desalniettemin gaat veel GIS onderzoek over het verbeteren van de 2D representaties alsook het ontwikkelen van nieuwe methoden die op 2D representaties zijn gebaseerd.

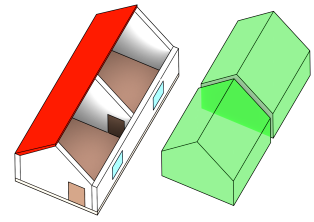


Figuur 1: In GIS wordt een kubus niet gerepresenteerd als een 3D volume maar als 6 vierkante 2D vlakken die tesamen de ruimte van de kubus omvatten.

Deze dissertatie bestudeerd een fundamenteel andere modelleerbenadering waarbij zowel ruimtelijke als niet-ruimtelijke kenmerken worden gemodelleerd als extra geometrische dimensies. De kenmerken "tijd" en "schaal" worden daarbij specifiek bekeken (Figuur 2). Eerdere onderzoeken naar het modelleren van tijd en schaal als extra dimensie zijn nooit verder gekomen dan een conceptuele beschrijving. Dit onderzoek daarentegen beoogt de fundamentele aspecten van een multidimensionaal GIS te realiseren door hoger dimensionale (nD) representaties te ontwikkelen inclusief bewerkingsmethodes om deze nD geografische informatie te creëren, manipuleren en visualiseren. In deze thesis wordt aangetoond dat een nD benadering aan de ene kant veel computergeheugen vraagt om alle relaties over dimensies heen op te slaan. Aan de andere kant is de aanpak zeer krachtig gebleken omdat een nD benadering op een doeltreffende en consistente manier geometrie met haar attributen opslaat. Naast de objecten, kunnen ook alle topologische relaties tussen de objecten worden opgeslagen welke zich kunnen voordoen binnen en tussen iedere dimensie. De in dit onderzoek voorgestelde aanpak is generiek en kan eenvoudig worden uitgebreid om andere niet-ruimtelijke aspecten te modelleren. Hierdoor is data management waarbij de consistentie van geografische informatie wordt gegarandeerd over dimensies heen. Bovendien kunnen hiermee krachtigere bewerkingen worden uitgevoerd zoals controleren



Figuur 2: 3D ruimte, tijd en schaal kunnen worden gemodelleerd als 5D ruimte.

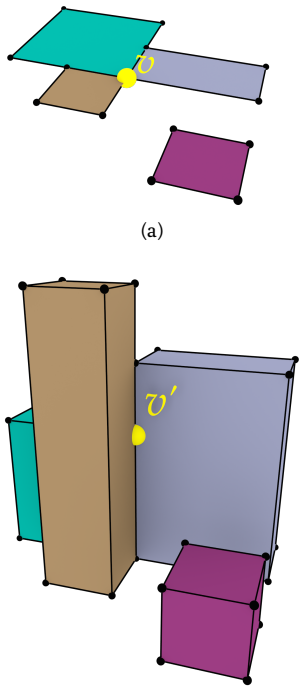


Figuur 3: Een 3D ruimtelijke opdeling bestaat uit een set ruimtevullende volumes zonder gaten en overlap.

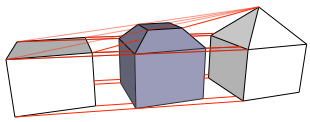
of twee objecten op enig moment in de tijd aangrenzend zijn.

Om een nD ruimte te modelleren kan het best gestart worden met een nD ruimtelijke partitie (zonder gaten en overlap) (Figuur 3). Conceptueel kan dit worden weergegeven als een nD *simplicial complex* of een *cell complex*. Vervolgens kan dit worden geïmplementeerd via een *simplex-based* datastructuur met een *incidence graph*, bijvoorbeeld als een set van *Nef polyhedra* of, zoals dat in dit onderzoek is gedaan, via geordende topologische modellen (*cell-tuple* en *generalised/combinatorial maps*).

Het construeren van computer representaties van nD objecten kan erg complex zijn. Bestaande 2D/3D constructie methodes manipuleren *combinatorial primitives* of maken gebruik van bewerkingen op primitieve-niveau (zoals Euler operatoren). Deze gaan uit van onze intuïtie over 2D en 3D geometrieën en werken daarmee niet goed in hogere dimensies. Hierdoor is het helaas heel makkelijk om invalide nD objecten te creëren die vervolgens moeilijk te bewerken zijn. Dit probleem doet zich ook al voor in drie dimensies.



Figuur 4: (a) een set polygonen wordt geconverteerd naar (b) een set blokken door de polygonen van 2D naar 3D op te trekken.



Figuur 5: Twee detailniveaus van een 3D model van een huis (links en rechts) worden gelinkt tot een 4D model.

In deze dissertatie is een manier onderzocht en ontwikkeld om eenvoudig valide representaties van nD objecten te construeren. Daartoe zijn er drie methodes voorgesteld: *Extrusion* neemt een $(n-1)$D cell complex en een set van intervallen per cell en projecteert deze parallel aan een nieuwe as ten einde een nD cell complex te construeren (Figuur 4). *Incremental construction* construeert een nD object op basis van zijn $(n-1)$-grens. Via het *linken* van verschillende representaties van hetzelfde 3D object op verschillende detailniveaus (LODs) met als resultaat een 4D model (Figuur 5).

Om nD modellen te kunnen visualiseren en te kunnen bewerken in huidige software, is er een methode nodig om valide 2D/3D subsets af te leiden uit de nD data. Als opstap heeft dit onderzoek laten zien hoe een zowel orthografische als een perspectieve projectie van nD naar $(n-1)$D kan worden gedefinieerd.

Tenslotte heeft deze thesis alle hierboven genoemde concepten gevalideerd op nD data over de "echte" wereld. Omdat er veel fouten voorkomen in deze data, zijn er in dit onderzoek ook methodes ontwikkeld om valide polygonen en een plenaire partitie te creëren op basis van *constrained* triangulatie. Daarnaast is er een methode ontwikkeld om polyhedra en ruimtelijke opdelingen te corrigeren door het "snappen" van primitieven op lagere dimensies en het verwijderen van overlap door middel van Boolean set bewerkingen op *Nef polyhedra*. Hierdoor zijn er testen gedaan met data tot 6D gebaseerd op "echte" GIS data, *wat een goede basis is voor nD GIS*.

In de toekomst kan dit onderzoek worden uitgebreid met wijzigingsbewerkingen voor nD objecten, "echte" 4D ruimtelijk-temporele datasets en correctie methodes die de kwaliteit van de nD data garanderen. Alle implementaties van deze dissertatie zijn publiekelijk beschikbaar via open source licenties.

Resumen (Spanish summary)

Nuestro mundo es tridimensional y complejo, cambiando continuamente a través del tiempo y mostrándose distinto a distintas escalas. Aún así, cuando lo modelamos usando sistemas de información geográfica (SIG), generalmente usamos representaciones 2D, las cuales esencialmente consisten en conjuntos de puntos, líneas y polígonos enlazados entre sí (Figura 1). Estas representaciones son relativamente fáciles de usar y eficientes, y una gran variedad de métodos se han construido usándolas. Sin embargo, las representaciones 2D son necesariamente restrictivas. Nos fuerzan a reducir los problemas a dos dimensiones, limitan el tipo de objetos que podemos representar y dificultan guardar las relaciones entre diferentes objetos—especialmente cuando éstas son a través del tiempo y diferentes escalas. A pesar de ello, la mayoría de la investigación en SIG se dedica a mejorar estas representaciones 2D, así como a desarrollar nuevos métodos que las usan para resolver problemas, sean éstos nuevos o antiguos.

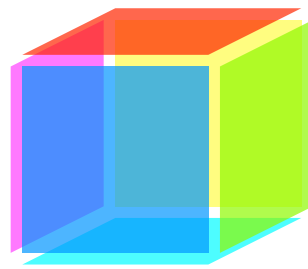


Figura 1: En los SIG, un cubo no se representa como un sólido, sino como las 6 caras cuadradas en su superficie.

Esta tesis explora un nuevo y fundamentalmente distinto paradigma de modelado—integrando características espaciales y espaciales como dimensiones en el sentido geométrico, específicamente apuntando hacia los casos del tiempo y la escala (Figura 2). A pesar de que este paradigma ha sido propuesto antes a un nivel conceptual, esta tesis tiene como objetivo llevarlo a cabo al *implementar los aspectos fundamentales de un SIG de altas dimensiones*, desarrollando representaciones para objetos en altas dimensiones (nD), así como nuevos métodos que operen en ellas para crear, manipular y visualizar información geográfica. Como esta tesis muestra, el paradigma de altas dimensiones tiene un alto consumo de memoria pero también es muy poderoso, proveyendo un método simple y consistente para guardar la geometría, los atributos y las relaciones topológicas entre objetos de cualquier dimensión. Este paradigma genérico puede ser fácilmente extendido a otras características no espaciales, haciendo posible un mejor manejo de datos con consistencia a través de las dimensiones y operaciones más poderosas.

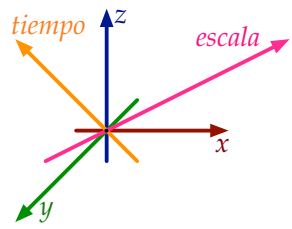


Figura 2: El espacio 3D, el tiempo y la escala modelados como un espacio 5D.

Para poder modelar el espacio de altas dimensiones, es mejor considerar una partición espacial nD como una base (Figura 3), la que es conceptualizada como un complejo simplicial o celular. Éste puede entonces ser implementado con una estructura de datos basada en

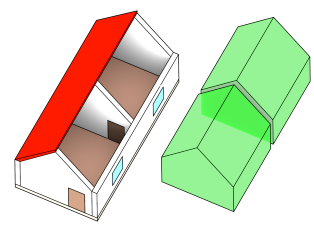


Figura 3: Una partición espacial 3D se compone de un conjunto de volúmenes que llenan el espacio sin huecos y sin solaparse entre ellos.

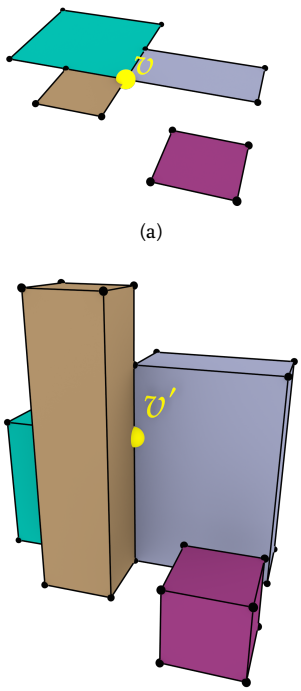


Figura 4: (a) Un conjunto de polígonos se convierte en (b) un conjunto de paralelepípedos al aplicar una extrusión 2D a 3D.

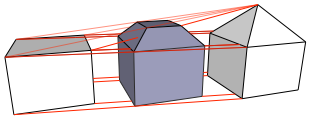


Figura 5: Dos niveles de detalle de una casa (izquierda y derecha) se enlazan en un modelo 4D.

símplices, un grafo de incidencia, como un conjunto de poliedros Nef, o—como en esta tesis—usando modelos topológicos ordenados, como la tupla de células (*cell-tuple*) o los mapas generalizados o combinatorios (*generalised/combinatorial maps*).

Crear representaciones de objetos en altas dimensiones puede ser complejo. Los métodos comunes usados en 2D y 3D, como la manipulación directa de elementos combinatorios primitivos, o el uso de operaciones de construcción que operan al nivel de los elementos primitivos (como las operaciones de Euler), se basan en nuestra intuición de la geometría 2D/3D, y por lo mismo no funcionan bien en altas dimensiones. Esto causa que sea demasiado fácil crear objetos inválidos, los cuales no pueden ser fácilmente interpretados o reparados—un problema que es extremadamente aparente incluso en tres dimensiones.

Esta tesis propone tres métodos novedosos para crear fácilmente representaciones de objetos en altas dimensiones, todos los cuales son intuitivos e intentar crear datos de salida válidos. La *extrusión* recibe un complejo celular $n-1$ dimensional y un conjunto de intervalos por cada célula, proyectándolos de forma paralela a un nuevo eje para crear un complejo celular n dimensional (Figura 4). La *construcción incremental* describe un objeto n dimensional basado en su frontera $n-1$ dimensional, desde la dimensión cero (puntos) y hacia arriba. Finalmente, es posible construir un modelo 4D *enlazando* una serie de modelos 3D a diferentes niveles de detalle (Figura 5).

Para poder visualizar modelos en altas dimensiones, así como poder procesarlos en el software existente, es importante tener métodos para extraer de éstos subconjuntos significativos 2D/3D. Como un paso inicial hacia este tipo de métodos, esta tesis muestra como se pueden definir las proyecciones ortográficas y de perspectiva de n dimensiones a $n-1$.

Finalmente, esta tesis tuvo un enfoque importante en validar los algoritmos con datos reales, lo cual sólo fue posible al desarrollar métodos para reparar datos inválidos, los cuales son comunes en la práctica. Esta tesis por lo tanto contiene métodos para crear polígonos y particiones planares válidas usando una triangulación restringida de los datos de entrada, así como un método para reparar poliedros y particiones 3D juntando objetos (*snapping*) de menores dimensiones y eliminando las partes que se solapan utilizando operaciones booleanas de conjuntos con poliedros Nef. Esto permitió pruebas hasta en seis dimensiones basados en datos reales—*una buena base para un SIG de altas dimensiones.*

En el futuro, el trabajo de esta tesis será extendido con operaciones de modificación en altas dimensiones, datos reales espaciales y temporales 4D, así como métodos de reparación de datos con garantías de calidad. Todas las implementaciones hechas para esta tesis están disponibles públicamente bajo licencias de código abierto.

Curriculum vitae

Gustavo Adolfo **Ken Arroyo Ohori** was born on June 12, 1985 in Mexico City. He graduated in 2003 from the Bicultural High School ('*Preparatoria Bicultural*') programme and in 2007 from the BSc in Computer Science and Technology programme ('*ITC01: Ingeniería en Tecnologías Computacionales*') at the Mexico City campus of the Monterrey Institute of Technology and Higher Education ('*Instituto Tecnológico y de Estudios Superiores de Monterrey*').

Arriving to the Delft University of Technology in 2008, Ken graduated from the MSc in Geomatics programme in 2010 with his thesis on the *Validation and automatic repair of planar partitions using a constrained triangulation*, which later evolved into some of the data repair ideas exposed in Chapter 10 of this thesis.

In 2011, Ken started his PhD project on the *Higher-dimensional modelling of geographic information*, supervised by Jantien Stoter and Hugo Ledoux, working under the umbrella of the project *5D Data Modelling: Full Integration of 2D/3D Space, Time and Scale Dimensions* of the Dutch Technology Foundation (*STW*). In October 2012, he visited Guillaume Damiand, collaborating on the incremental construction method that later became Chapter 7 of this thesis.

In 2016, Ken started working as a postdoc in the same *5D Data Modelling* project, focusing on 4D visualisation and in the integrated modelling of space and time.

Publications

- **Voxelization algorithms for geospatial applications: Computational methods for voxelating spatial datasets of 3D city models containing 3D surface, curve and point data models**. Pirouz Nourian, Romulo Gonçalves, Sisi Zlatanova, Ken Arroyo Ohori and Anh Vu Vo. *MethodsX* 3, January 2016, pp. 69–86.
- **Automatically enhancing CityGML LOD2 models with a corresponding indoor geometry**. Roeland Boeters, Ken Arroyo Ohori, Filip Biljecki and Sisi Zlatanova. *International Journal of Geographical Information Science* 29(12), December 2015, pp. 2248–2268.
- **Automatic semantic-preserving conversion between OBJ and CityGML**. Filip Biljecki and Ken Arroyo Ohori. In F. Biljecki and V. Tourre (eds.), *Eurographics Workshop on Urban Data Modelling and Visualisation*, Eurographics Association, Delft, The Netherlands, November 2015, pp. 25–30.
- **Storing a 3D city model, its levels of detail and the correspondences between objects as a 4D combinatorial map**. Ken Arroyo Ohori, Hugo Ledoux and Jantien Stoter. In Alias Abdul Rahman, Umit Isikdag and Francesc Antón Castro (eds.), Joint International Geoinformation Conference 2015, 28–30 October 2015, Kuala Lumpur, Malaysia, ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences II-2/W2, ISPRS, Kuala Lumpur, Malaysia, October 2015, pp. 1–8.

In Chapter 8
- **Modelling a 3D city model and its levels of detail as a true 4D model**. Ken Arroyo Ohori, Hugo Ledoux, Filip Biljecki and Jantien Stoter. *ISPRS International Journal of Geo-Information*, 4(3), September 2015, pp. 1055–1075.

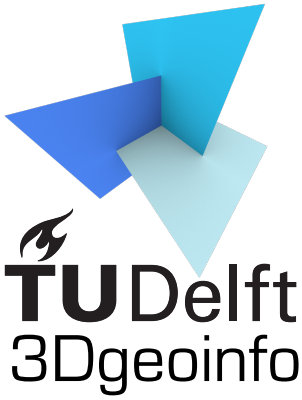
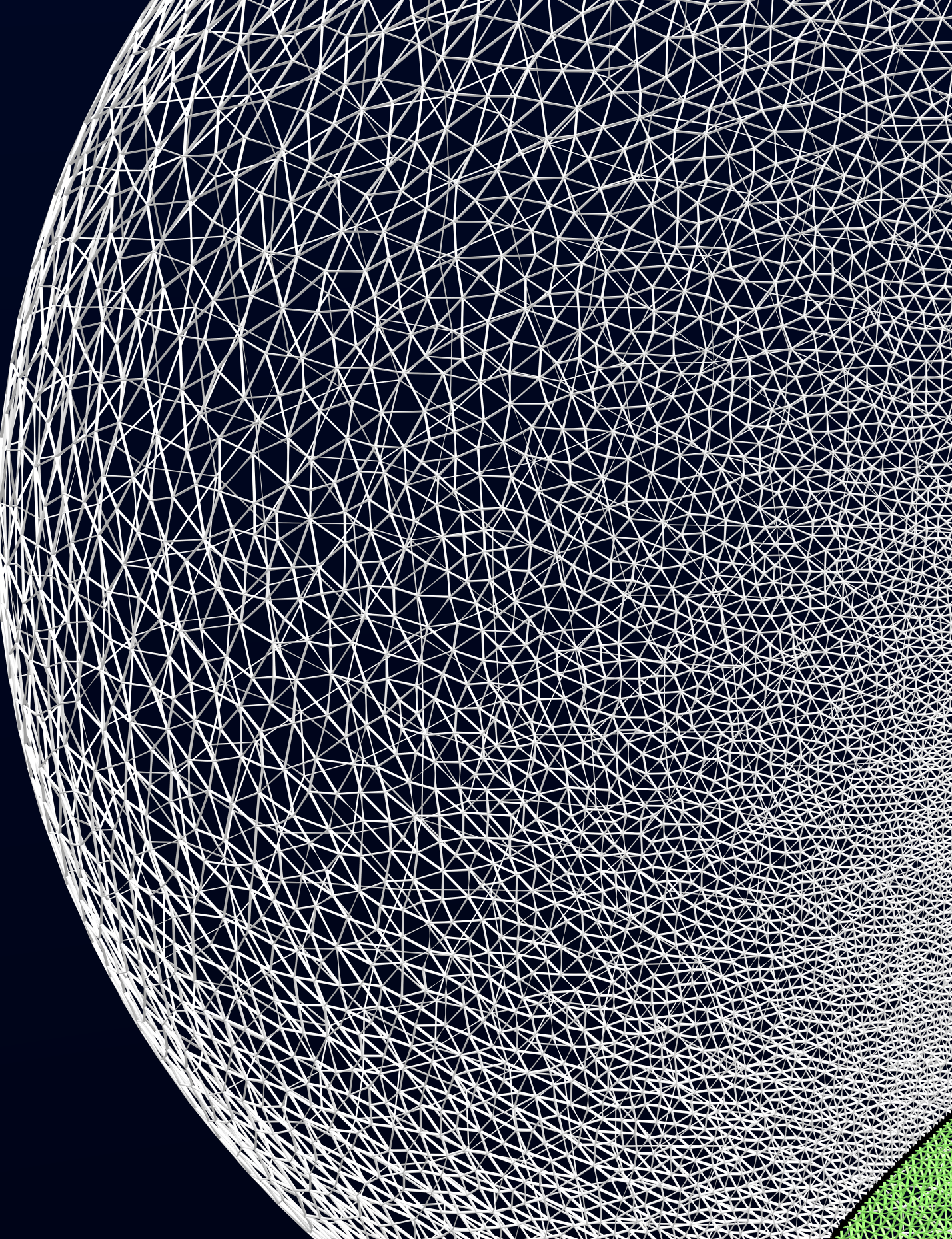
In Chapter 6
- **A dimension-independent extrusion algorithm using generalised maps**. Ken Arroyo Ohori, Hugo Ledoux and Jantien Stoter. *International Journal of Geographical Information Science* 29(7), July 2015, pp. 1166–1186.

In §4.3 & §4.4
- **An evaluation and classification of nD topological data structures for the representation of objects in a higher-dimensional GIS**. Ken Arroyo Ohori, Hugo Ledoux and Jantien Stoter. *International Journal of Geographical Information Science* 29(5), May 2015, pp. 825–849.

In §10.2
- **A triangulation-based approach to automatically repair GIS polygons**. Hugo Ledoux, Ken Arroyo Ohori and Martijn Meijers. *Computers & Geosciences* 66, May 2014, pp. 121–131.

In Chapter 7
- **Constructing an n-dimensional cell complex from a soup of $(n-1)$-dimensional faces**. Ken Arroyo Ohori, Guillaume Damiand and Hugo Ledoux. In Prosenjit Gupta and Christos Zaroliagis (eds.), *Applied Algorithms. First International Conference, ICAA 2014, Kolkata, India, January 13–15, 2014. Proceedings*, Lecture Notes in Computer Science 8321, Springer International Publishing Switzerland, Kolkata, India, January 2014, pp. 37–48.
- **Using extrusion to generate higher-dimensional GIS datasets**. Ken Arroyo Ohori and Hugo Ledoux. In Craig Knoblock, Peer Kröger, John Krumm, Markus Schneider and Peter Widmayer (eds.), *SIGSPATIAL'13: Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, ACM, Orlando, Florida, United States, November 2013, pp. 398–401.

- **Modelling higher dimensional data for GIS using generalised maps**. Ken Arroyo Ohori, Hugo Ledoux and Jantien Stoter. In B. Murgante, S. Misra, M. Carlini, C. Torre, H. Q. Nguyen, D. Taniar, B. Apduhan and O. Gervasi (eds.), *Computational Science and Its Applications — ICCSA 2013. 13th International Conference, Ho Chi Minh City, Vietnam, June 24–27, 2013, Proceedings, Part I*, Lecture Notes in Computer Science 7971, Springer Berlin Heidelberg, June 2013, pp. 526–539.
- **Representing the dual of objects in a four-dimensional GIS**. Ken Arroyo Ohori, Pawel Boguslawski and Hugo Ledoux. In A. Abdul Rahman, P. Boguslawski, C. Gold and M. N. Said (eds.), *Developments in Multidimensional Spatial Data Models*, Lecture Notes in Geoinformation and Cartography, Springer Berlin Heidelberg, Johor Bahru, Malaysia, May 2013, pp. 17–31. In §5.4
- **Manipulating higher dimensional spatial information**. Ken Arroyo Ohori, Filip Biljecki, Jantien Stoter and Hugo Ledoux. In Danny Vandenbroucke, Bénédicte Bucher and Joep Crompvoets (eds.), *Geographic Information Science at the Heart of Europe. Proceedings of the 16th AGILE International Conference on Geographic Information Science*, Leuven, Belgium, May 2013.
- **Validation and automatic repair of planar partitions using a constrained triangulation**. Ken Arroyo Ohori, Hugo Ledoux and Martijn Meijers. *Photogrammetrie, Fernerkundung, Geoinformation* 5, October 2012, pp. 613–630. In §10.2
- **Automatically repairing polygons and planar partitions with prepair and pprepair**. Ken Arroyo Ohori, Hugo Ledoux and Martijn Meijers. *Proceedings of the 4th Open Source GIS UK Conference*, Nottingham, United Kingdom, September 2012.
- **Integrating scale and space in 3D city models**. Jantien Stoter, Hugo Ledoux, Martijn Meijers and Ken Arroyo Ohori. In Jacynthe Pouliot, Sylvie Daniel, Frédéric Hubert and Alborz Zamyadi (eds.), *Proceedings of the 7th International 3D GeoInfo Conference, International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* XXXVIII-4/C26, ISPRS, Québec City, Canada, May 2012, pp. 7–10.
- **Automatically repairing invalid polygons with a constrained triangulation**. Hugo Ledoux, Ken Arroyo Ohori and Martijn Meijers. In Jérôme Gensel, Didier Josselin and Danny Vandenbroucke (eds.), *Multidisciplinary Research on Geographical Information in Europe and Beyond. Proceedings of the 15th AGILE International Conference on Geographic Information Science*, Avignon, France, April 2012, pp. 13–18.
- **Edge-matching polygons with a constrained triangulation**. Hugo Ledoux and Ken Arroyo Ohori. *Proceedings of GIS Ostrava 2011*, Ostrava, Czech Republic, January 2011, pp. 377–390.



ISBN 978-1-326-59638-5
90000
9 781326 596385