Order Number 9108444

Representation of d-dimensional geometric objects

Brisson, Erik, Ph.D.

University of Washington, 1990

U·M·I
300 N. Zeeb Rd.
Ann Arbor, MI 48106

Representation of d-Dimensional Geometric Objects

by

Erik Brisson

A dissertation submitted in partial fulfillment
of the requirements for the degree of

Doctor of Philosophy

University of Washington

1990

Approved by Richard Adey
(Chairperson of Supervisory Committee)

Program Authorized
to Offer Degree Computer Science and Engineering

Date August 8, 1990

Doctoral Dissertation

In presenting this dissertation in partial fulfillment of the requirements for the Doctoral degree at the University of Washington, I agree that the Library shall make its copies freely available for inspection. I further agree that extensive copying of this dissertation is allowable only for scholarly purposes, consistent with "fair use" as prescribed in the U. S. Copyright Law. Requests for copying or reproduction of this dissertation may be referred to University Microfilms, 300 North Zeeb Road, Ann Arbor, Michigan 48106, to whom the author has granted "the right to reproduce and sell (a) copies of the manuscript in microform and/or (b) printed copies of the manuscript made from microform."

Signature Eric Bauer

Date August 8, 1990

University of Washington

Abstract

Representation of d-Dimensional Geometric Objects

by Erik Brisson

Chairperson of the Supervisory Committee: Professor Richard Anderson
Department of Computer Science and Engineering

This work investigates data structures and algorithms for representing and manipulating d-dimensional geometric objects for arbitrary $d \geq 1$. These objects are often described by a set of basic building blocks, together with an incidence relation among the blocks. We give a new representation of such objects, the 'cell-tuple structure', which provides direct access to topological structure, ordering information among cells, the topological dual, and boundaries.

We define 'subdivided d-manifolds', a class of geometric objects which is large enough to encompass most computational geometry applications. Our first main result is that both the incidence graph and the cell-tuple structure are powerful enough to represent subdivided d-manifolds up to topological equivalence. Our second main result is that circular orderings of cells exist in subdivided d-manifolds for all $0 \leq k \leq d$: given a $(k-2)$-dimensional cell which is incident to a $(k+1)$-dimensional cell, there is a simple closed path encountering each of the cells between them exactly once.

Our work generalizes the work of Guibas and Stolfi (the quad-edge data structure, representing subdivisions of 2-manifolds) and the work of Dobkin and Laszlo (the facet-edge data structure, representing subdivisions of 3-manifolds).

The cell-tuple structure gives a simple, uniform representation of subdivided manifolds which unifies the existing work in the field and provides intuitive clarity in all

dimensions. In two and three dimensions it is competitive in terms of size and speed with other methods.

Operators are defined for accessing structure and ordering information. Structure and ordering within the dual of a subdivided manifold may be accessed in a symmetric manner. Constructors are defined for creating and manipulating subdivided manifolds. Implementation issues are discussed, and an actual implementation is described.

Table of Contents

List of Figures

List of Tables

ACKNOWLEDGEMENTS

I would like to express:

gratitude, to my advisor, Professor Richard Anderson, for all of his support and encouragement. He was always willing to consider new problems and listen to ideas, and made many insightful criticisms and suggestions;

thanks, to Professors Paul Beame, Tony DeRose, Richard Ladner, Kenneth Sloan, Steven Tanimoto, and Victor Klee for their help and involvement along the way;

pleasure, for being lucky enough to be part of this very active department, where the graduate students were smart and sociable, the teaching was of high quality, and the professors' doors were open to students from the very first day;

appreciation, to my wife, Eileen Sullivan, for making life complete.

Chapter 1

Introduction

1.1 Introduction

The work to be described in this dissertation falls within the field of computational geometry, which is concerned with representing geometric objects and solving geometric problems on computational machines. The aim of the theoretical work in this field is to solve fundamental computational problems that arise in such areas as computer graphics, computer vision, computer aided design and manufacture, robotics, VLSI layout, visualizing mathematical forms, statistics, pattern matching, etc.

Computer scientists have been modeling 'real-world' phenomena for decades, and a number of schemes for representing 2- and 3-dimensional objects have been proposed and used. The beginnings of a conceptually new approach appeared in the mid-1970's, and took a distinct jump in the mid-1980's. This approach clearly separated the representation of geometry and the representation of topology, was quite simple and elegant, and had a much more mathematical flavor than earlier methods.

Our work continues and extends this new approach, generalizing and unifying the existing work to give a general, coherent representation of higher dimensional objects. We define a very general class of objects, the 'subdivided manifolds', and develop a new data structure, the 'cell-tuple structure', which represents the topological structure and

ordering information which is present in subdivided manifolds. We also define operations for accessing and manipulating them, and consider implementation issues.

1.2 Motivation

Often, the objects occurring in computational geometry may be described as a collection of simple building blocks, along with the relations among these building blocks. We will call such a partition into simpler units a 'subdivision' of an object. (A technical definition for the class of objects we will study will be introduced in Section 2.4.)

To give some support to the claim that such subdivisions arise naturally, we will describe several much-studied problems from the field of computational geometry. These problems will also arise when describing algorithms implemented in Section 6.3.9 and when discussing directions for future work in Section 7.2.

The goal of each of these problems is to construct a geometric object that is defined by a set of input points or hyperplanes. Specifically, these objects are the convex hull, the Voronoi diagram, the Delaunay diagram, and the arrangement. Part of such a construction is to produce the basic building blocks; if such a block is a k-dimensional object, it will be referred to as a 'k-cell'.

The convex hull of a set of points in $\mathbb{R}^d$ is the smallest convex set containing them. Given a finite set of points, we wish to produce their convex hull. The output may be specified to be the set of (d–2)-dimensional facets on the boundary of the hull, or the set of all faces on the boundary of the convex hull, and may require supplying incidence information as well. A related, but computationally easier problem for general d, is to find the set of extreme points, which is the smallest set of input points which define the same convex hull as the input set.

Given n input points, the worst case complexity of the output is in general $\Theta(n^{\lfloor d/2 \rfloor})$ ([Kle64]). For $d = 2$ and 3, there exist algorithms which run in time $O(n \log n)$, which are proven to be optimal by reductions from sorting. For even $d \geq 4$, the 'beneath-beyond' algorithm of Seidel (described in [Ede87]), runs in $O(n^{\lfloor d/2 \rfloor})$. For odd $d \geq 3$,

the best existing algorithm [Sei86] runs in time $O(n^{\lfloor d/2 \rfloor} \log n)$. (To give a few other relevant references, we mention [CK70], [Dwy87], [Edd77], [Efr65], [Für86], [Gra72], [Jar73], [KS86], [Pre79], [Swa85].)

Given a set of points $x_1, \ldots, x_n$ in $\mathbb{R}^d$, define the 'Voronoi region' of x_i as $\mathcal{V}(x_i) = \{y \in \mathbb{R}^d \mid dist(y, x_i) \leq dist(y, x_j)\ \forall j \neq i\}$. These sets are convex and are bounded by a finite number of line segments and rays, as they are the intersections of finite sets of closed half-spaces. The set of such regions defines a subdivision of $\mathbb{R}^d$ (whose d-cells are the interiors of the Voronoi regions, and whose lower dimensional cells are defined by intersections of Voronoi regions). The resulting subdivision is called the 'Voronoi diagram'. The d-dimensional Voronoi diagram has size $\Theta(n^{\lceil d/2 \rceil})$ for n input points ([Kle80]). Besides direct methods, the problem of computing the d-dimensional Voronoi diagram may be reduced by a 'lifting' function to the problem of computing a $(d+1)$-dimensional convex hull of n points ([Bro79], [GS85]). This method gives running times that match the best direct algorithms.

Consider the same set of input points. If a subset of the input points all lie on the surface of a unique d-sphere and there are no other input points in the interior of this d-sphere, then the convex hull of the subset defines a 'Delaunay d-face'. The interiors of all such Delaunay faces define a subdivision of the convex hull of the input set. If the space that they each subdivide is extended appropriately, the Voronoi diagram and the Delaunay diagram become topological duals of each other. Often the Delaunay diagram is built first, and the Voronoi diagram is constructed from it. (To give a few relevant references to work on Voronoi and Delaunay diagrams and algorithms for their construction, we mention [AB83], [Bow81], [Dwy86], [Dwy89], [For86], [GS78], [GS85], [Sib78], [SH75], [Wat81].)

The last problem we will discuss is that of constructing arrangements. Given a set H of n hyperplanes in $\mathbb{R}^d$, where $d \geq 2$, their arrangement $A(H)$ is the subdivision of $\mathbb{R}^d$ they create. That is, if $H = \{h_1, \ldots, h_n\}$, and h_i^- and h_i^+ are the open half-spaces defined by h_i, then the cells of $A(H)$ are $\{\cap_{i=1}^n \tilde{h}_i \mid \tilde{h} = h_i^-, h_i, \text{ or } h_i^+\}$. A description of

an arrangement must include an enumeration of its cells, and may include their incidence relations. If the input hyperplanes are in general position, so that the intersection of any k hyperplanes is a $(d-k)$-dimensional cell, then $A(H)$ is *simple*. The number of k-cells in a general arrangement is $O(n^d)$, and the number of k-cells in a simple arrangement is $\Theta(n^d)$. An optimal algorithm for two dimensions was given independently by Edelsbrunner, O'Rourke and Seidel [EOS86] and by Chazelle, Guibas and Lee [CGL83]. The former paper also gives an optimal algorithm for all $d \geq 2$.

Each of these well-known objects can be described as a set of 'flat' cells, together with the incidence relation among them. Methods of representing objects which can be described this way will be the theme of this dissertation.

1.3 Representing Subdivided Manifolds

The aim of this work is to investigate representations of d-dimensional geometric objects. The class of objects which we will represent are the subdivided manifolds, which are general enough to form a very large class, including most objects which occur in applications, yet which contain enough structure to allow simple representations and to contain valuable ordering information.

The basic building blocks in a subdivision will be referred to as 'faces' or 'cells'. When we say that we are representing the topology of an object, this means that we are representing the relation between the faces – which faces are incident, which are adjacent, etc. This is different from representing geometry, which is concerned with describing the shape of the faces. As an example, Figure 1.1 shows two objects which are topologically equivalent, but not geometrically equivalent. A mathematical definition of topological equivalence of subdivisions will be given in Section 2.4.

The basic requirement of any representation is that it represent objects up to (topological) equivalence. There are several other properties to which it is also useful to have access.

Many computational geometry algorithms use ordering information. For example, the

Figure 1.1: Example of two topologically equivalent objects

only known optimal 3-dimensional convex hull algorithm [PH77]; the divide-and-conquer method (one optimal algorithm) for building the 2-dimensional Voronoi diagram [GS85], [Lee80], [SH75]; and one way of describing the optimal algorithm for constructing the 2-dimensional arrangement [CGL83] (the other method is given in [EOS86]). Rather than giving a formal definition of ordering at this point, we will give some examples of 2- and 3-dimensional subdivisions.

In Figure 1.2 is a portion of a 2-dimensional subdivision. In case (a), we are ordering edges and faces about a vertex, and in case (b) we are ordering the vertices and edges about a face.

In Figure 1.3 is a 3-dimensional subdivision. In case (a), we are ordering the 2- and 3-dimensional faces about an edge, in case (b) we are ordering the edges and 2-faces lying in the boundary of a given 3-face, about a vertex, and in case (c) we are ordering the vertices and edges within a 2-face.

We will generalize this kind of ordering to d-dimensions – given a $(k-2)$-dimensional cell which is incident to a $(k+1)$-dimensional cell ($1 \leq k \leq d$), we will show that there exists an ordering of the $(k-1)$- and k-dimensional cells which lie 'between' them.

Given a subdivision of an object, it is sometimes useful to also have access to that object's topological dual. One classic example in computational geometry is the duality

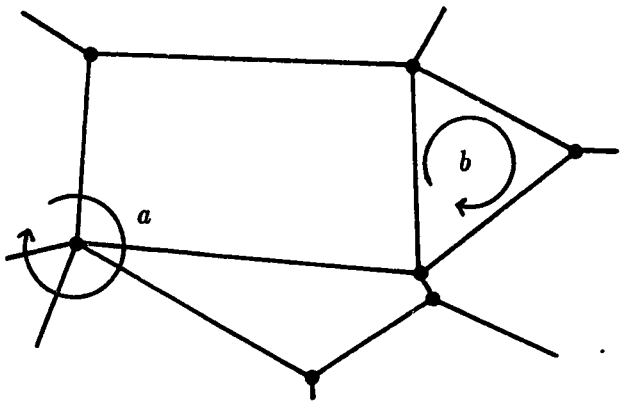


Figure 1.2: Examples of order in two dimensions

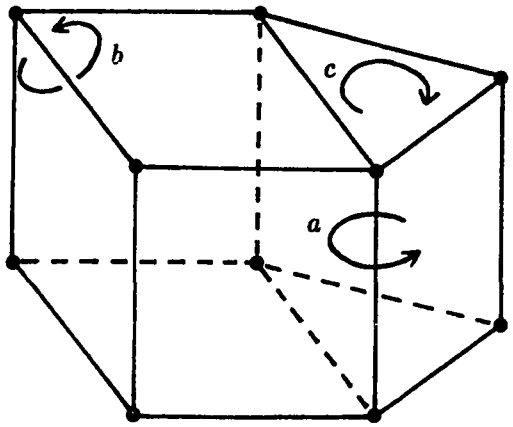


Figure 1.3: Examples of order in three dimensions

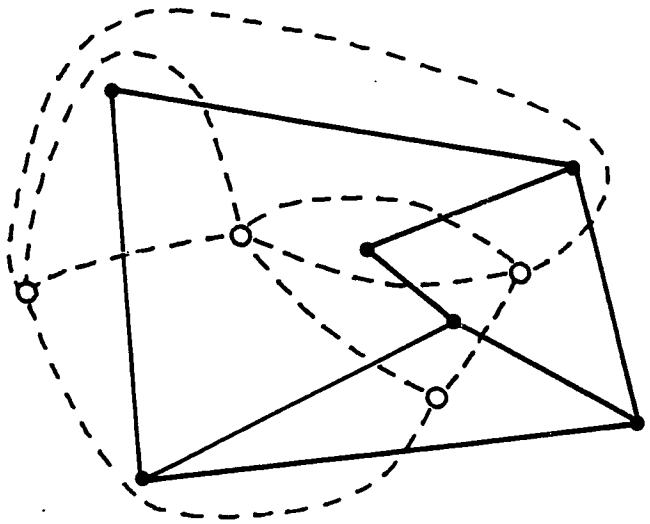
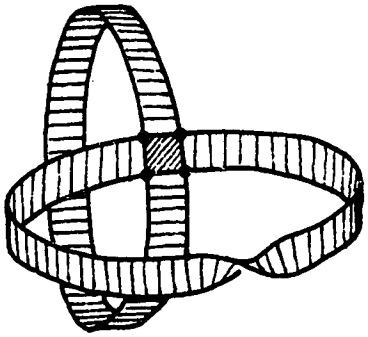


Figure 1.4: Example of the dual

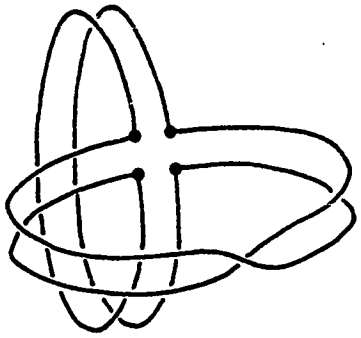
between the Voronoi diagram and the Delaunay diagram. In very rough terms, the dual of a subdivided d-dimensional object is a subdivision of the same object produced by replacing every k-dimensional face by a $(d-k)$-dimensional face, while maintaining the corresponding incidence relations. An example is given in Figure 1.4 (the primal is depicted by solid lines and filled circles, while the dual is shown by dashed lines and open circles).

When modeling real-world phenomena, we are usually modeling objects which have boundaries, so to be useful in practice, a representation must be able to handle objects having boundaries in a fashion consistent with objects not having boundaries. (See Figure 1.5.)

The goal of the investigation leading to this dissertation was to produce a representation of geometric objects which includes the four properties just described – structure, ordering of cells, the dual and boundaries – in a consistent form for arbitrary dimensions. The cell-tuple structure to be described does this, and agrees with existing work.



A subdivided
2-manifold-with-boundary



Its boundary

Figure 1.5: Example of an object and its boundary

1.4 Problems with the Combinatorial Approach in Higher Dimensions

The representations of topological structure considered in this dissertation are combinatorial in nature, in that they are discrete structures representing relations between cells.

While all surfaces may be characterized combinatorially by the classification theorem for surfaces, there is not a combinatorial characterization of all 3-manifolds – in fact, there is not even a combinatorial characterization of S^3, the 3-sphere. A famous problem in mathematics is whether the Poincaré conjecture ([Poi04]) is true or false: that among 3-manifolds the 3-sphere is characterized by having a trivial fundamental group[1].

We will discuss combinatorial characterizations of manifolds briefly here, to give a flavor for the problems involved (our discussion follows the treatment in [Sti84]). Given a polygon, its edges can be given orientations (i.e. one endpoint identified as a 'head', the

[1]The (homotopic) fundamental group is a basic concept in algebraic topology, which may be found in introductory texts on the subject. It will not be used (directly) in this dissertation.

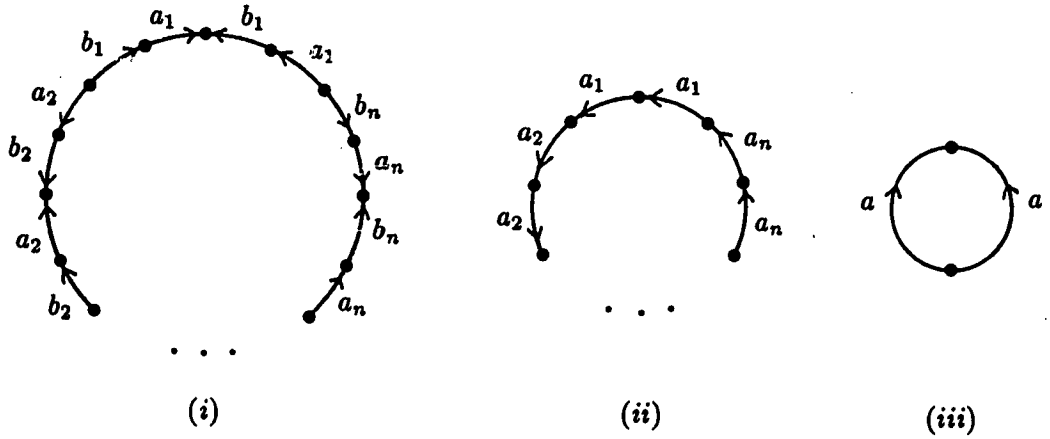


Figure 1.6: The three forms in classification of surfaces

other as a 'tail'), so that each vertex is incident to the head of one edge and the tail of another. A (finite) closed surface (2-manifold) can be defined by a (finite) set of polygons, whose edges are given orientations around the boundary, with edges identified in pairs so that their orientations are preserved (heads are always identified with heads, tails with tails). This is called a 'schema'. Any such schema can be reduced to a schema consisting of a single polygon, without changing the topology of the surface. The classification theorem for surfaces [DH07] says that any closed surface can be reduced to one of the three forms given in Figure 1.6. The first, (i), is a 'sphere with n handles', the second, (ii), is 'a sphere with n crosscaps', and the third, (iii), is a sphere.

The same result goes through for (finite) surfaces with boundary, except that the three forms given above may have an arbitrary (finite) number of holes in them.

It would nice if it was possible to classify a surface from its schema, without having to do any reduction. The first step towards this is to compute the Euler characteristic of the surface: $V - E + F$, where V is the number of vertices, E is the number of edges, and F is the number of faces. The Euler characteristics for the 3 basic forms are:

form (i) $V - E + F = 2 - 2n$

form (ii) $V - E + F = 2 - n$

form (iii) $V - E + F = 2$

The reduction used does not change the Euler characteristic, so this is enough to distinguish the topology if we can distinguish between form (i) and form (ii). If the Euler characteristic is less than 2, then if the schema is 'orientable' it is of form (i), and if not, it is of form (ii). This can be decided by triangulating the schema, and giving an orientation to each triangle. Such an assignment is called 'coherent' if each edge in the triangulation receives opposite orientations from its two incident triangles, and is called 'incoherent' otherwise. The surface is called 'orientable' if it admits a coherent orientation, and 'non-orientable' otherwise.

Thus Euler characteristic and orientability are enough to classify closed surfaces from their schema. Similarly, bounded surfaces are distinguishable by their Euler characteristic, orientability, and the number of boundary components.

It is natural to try to extend this approach to higher dimensions. A first try at giving a definition for 3-dimensional schema would be as:

- a finite set of polyhedra (3-cells, whose boundaries are partitioned into vertices, edges and faces) with disjoint interiors,
- faces of these 3-cells identified in pairs, with edges corresponding to edges and vertices to vertices,
- resulting in a connected complex.

This is not enough to guarantee that the resulting space is a 3-manifold, though it is enough to guarantee that the 3-cells incident to an edge form cycles. A complex satisfying the above is called a (finite) 'pseudo-manifold'.

To make this into a 3-manifold requires adding one more condition:

- the neighborhood surface of each vertex is a 2-sphere.

(The neighborhood surface is the result of intersecting a tiny 2-sphere about a vertex with the complex.) A nice result is that one can decide if a pseudo-manifold is indeed a manifold by computing the Euler characteristic for 3-spaces: $V - E + F - C = 0$ if

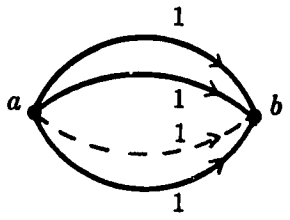


Figure 1.7: Example of a pseudo-manifold

and only if the pseudo-manifold is a manifold (where V, E and F are as before, and C is the number of 3-cells). This is perhaps not too surprising, since testing that the added condition is fulfilled requires checking that a 2-dimensional schema is equivalent to a 2-sphere, which can be done by the classification theorem for surfaces.

What happens when we try to continue this approach in higher dimensions? Since there is currently no way to identify the 3-sphere combinatorially, it is unlikely that there is any simple way to decide if a combinatorial structure is a 4-manifold. The same holds for $d > 4$. Furthermore, two related results show that such an approach is not feasible. Markov [Mar58] showed that the problem of recognizing whether two d-manifolds are homeomorphic (for $d \geq 4$) is recursively unsolvable, and S. P. Novikov (see [VKF74]) showed that the problem of recognizing a d-sphere (for $d \geq 5$) is recursively unsolvable.

We will give an example (taken from [Sti84]) of a very simple pseudo-manifold, to show how badly behaved non-manifolds may be even in three dimensions. The football shaped object in Figure 1.7 is a pseudo-manifold obtained by identifying the top and bottom faces, and identifying the front and back faces; it has two vertices, one edge, two faces and one 3-cell. Note that the Euler characteristic is equal to 2, so this is not a manifold. In fact, the neighborhood surface of each vertex is homeomorphic to a torus.

In this dissertation we will restrict our attention to subdivisions of manifolds and manifolds-with-boundary. It should be kept in mind, following the material in this section, that we do not know how to determine whether or not a combinatorial object may be realized as a subdivided manifold. On the other hand, in practice it is usually possible to make use of properties of the objects being dealt with to ensure that manifold structure is maintained.

1.5 Brief Review of Previous and Related Work

In this section we will give a very short history of representations of subdivisions. More will be given in Chapter 3.

The 'incidence graph' is probably the most widely used representation of topological structure in computational geometry, and is the longest-standing. It was introduced by Sallee [Sal66], and appeared in a classic text of Grünbaum's ([Grü67]), in the late 1960's as a means for representing polyhedra by the facial structure of their boundaries. In the form that it is commonly used in computational geometry, it is a graph with a node for every face in an object, with arcs connecting nodes if the corresponding faces are incident and differ in dimension by one.

The incidence graph does not provide any direct access to ordering information. The ordering information may be derived, but this may be slow and require extra work in an implementation. A natural way to provide such access would be to add a new data structure to the incidence graph (see Figure 1.8). For example, in the 2-dimensional case, each face might have a singly linked list of its incident vertices and edges, and each vertex might have a singly linked list of its incident edges and faces, taken in counterclockwise order. If efficient access to order in both directions is required, the lists could be made doubly linked. If it is necessary to begin at an arbitrary point in an ordering, back-pointers into the lists could be added. There are many variants of increasing complexity and size.

Consider the case of 2-dimensional subdivisions (subdivided surfaces). The extra

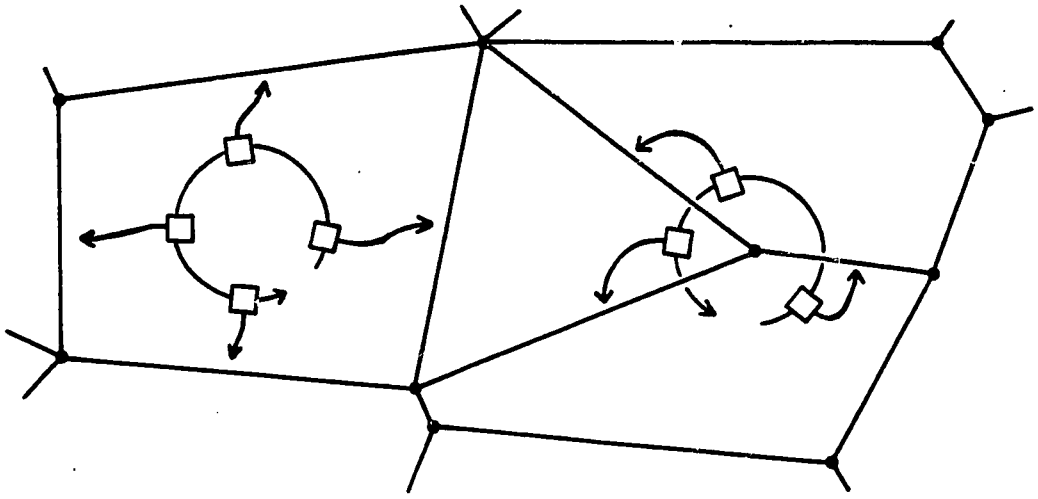


Figure 1.8: A simple representation of ordering in two dimensions

complexity and space requirements necessitated by the straightforward approach given in the previous paragraph may be avoided by taking an approach which attaches all of the ordering information to edges. This works out nicely because an edge is always incident to at most two faces and to at most two vertices. (Edges adjacent to fewer than two faces or vertices arise in representations that allow structures such as self-loops. The presence of a boundary may cause edges adjacent to a single face to occur.) In the mid-seventies, edge-based approaches were given by (among others) Baumgart [Bau75], Danaraj and Klee [DK78] and Muller and Preparata [MP78].

We will give a short description of a generic edge-based representation. Suppose that every edge is incident to two vertices and to two faces. Attach two pointers to each edge, pointing to the two edges which follow this edge in counterclockwise orderings of the boundaries of the two adjacent faces (see Figure 1.9). These pointers also give the edges which follow this edge in clockwise orderings around the adjacent vertices. Extra information may be associated with an edge, such as pointers to descriptors for vertices and faces. To allow efficient access to order in both directions an additional pair of pointers may be attached to each edge, corresponding to clockwise orderings of the boundaries of the adjacent faces. Then by applying tests to the four edges pointed to

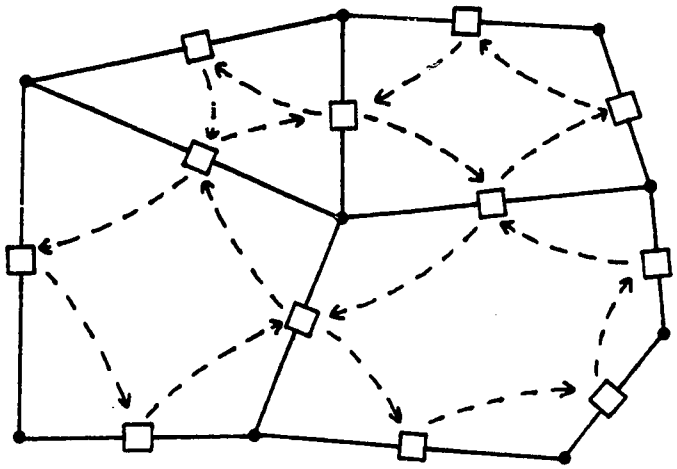


Figure 1.9: A generic edge-based scheme

from an edge, the next edge in either direction about a vertex, or around a face boundary, may be obtained.

Similar work on representing subdivisions of surfaces was also going on at the same time in the constructive solid geometry community, where one method of representing solid (3-dimensional) objects was to represent the boundary (e.g. [BEH79], [BHS80], [ELS75], [MS82]). Other work on representing subdivided surfaces appears in [DH87], [FH89], [Lie88], [Spe88].

For a thorough compendium of work on edge-based representations, see the work of Weiler ([Wei85], [Wei86]).

In 1984 Guibas and Stolfi [GS85] introduced the 'edge algebra', which used a single basic unit (the directed, oriented edge), together with three simple operators, to represent the structure of, and ordering in, subdivisions of 2-manifolds. Faces are represented implicitly by 'rings' of directed, oriented edges. The edge algebra can be characterized in an abstract setting as a finite set with three operators (following certain rules) acting on them. A one-to-one correspondence is shown between subdivided 2-manifolds and abstract edge algebras. The development includes an explicit representation of the dual subdivision, allowing symmetric access to the dual. Two simple primitive operators are

given for creating and combining edge algebras, which are capable of producing the edge algebra corresponding to any subdivided 2-manifold. We will refer to their work as the quad-edge data structure, which is what they called their implementation.

This new approach of representing subdivisions by a single basic element, which gives direct access to order and represents faces implicitly, culminated the edge-based approach and started a new line of research.

In 1987 Dobkin and Laszlo [DL87] gave a generalization of this new idea to the 3-dimensional case. They use a single basic unit, the 'facet-edge', along with a set of five operators for moving within the structure, and four primitive manipulative operators. While their data structure obeys nice algebraic rules, they do not give a combinatorial characterization of 3-manifolds. This is a hard problem (see Section 1.4). A somewhat similar data structure was described by Buckley [Buc88].

The independent work of Lienhardt [Lie89] and Tits [Tit81] in higher dimensions, though attempting to solve different problems, each contain ideas similar to some of those described in this dissertation. Lienhardt deals with a more general class of objects, which does not allow representation by the incidence graph, and which does not contain the ordering information which is present in subdivided manifolds. Tits deals with abstract complexes for an entirely different purpose (proving properties of mathematical objects called 'buildings'), so does not prove anything regarding representation of subdivided manifolds or ordering, and does not consider computational questions.

One of the main properties of this new approach is that it represents cells implicitly (by certain subsets of the basic elements), unlike the other existing approaches. For this reason, the representations developed in this new line of research will be referred to as 'implicit-cell representations', and the more conventional approaches will be referred to as 'explicit-cell representations'. The primary work on implicit-cell representations includes [DL87], [GS85], [Lie89] and this work.

1.6 Overview of this Work

The goal of this work is to generalize the new approach of Guibas and Stolfi, and Dobkin and Laszlo, to d dimensions, for $d \geq 1$, i.e. to find a structure representing topological structure, ordering, the dual, and boundaries of a d-dimensional subdivision using a single basic unit and a small set of operators.

In Chapter 2, we will give necessary mathematical background, beginning with basic topological definitions. Then, we will define CW complexes, which are necessary for the definition of subdivided manifolds, and simplicial complexes, which will play a large role in the proofs of the two main theorems.

We will give a review of previous and similar work in Chapter 3.

Chapter 4 gives the definition of the cell-tuple structure, states and proves the two main theorems, and discusses the dual subdivision and extensions to manifolds-with-boundary. We will define the cell-tuple structure for representing subdivided manifolds, and show that both the incidence graph and the cell-tuple structure are powerful enough to represent subdivided manifolds. The basic units of the cell-tuple structure are $(d+1)$-tuples, and the operators can be defined in terms of their effect on these tuples.

Without giving any details, one way of describing the cell-tuple structure for a subdivided d-manifold is as a $(d+1)$-regular edge-labeled graph, where every node is incident to exactly one edge labeled by each of $0, \ldots, d$. Graphs provide a simple means of operating on objects in higher dimensions, where intuition usually falls away quickly.

We cannot give a one-to-one correspondence between subdivided d-manifolds and cell-tuples structures similar to that given in [GS85] (the reasons follow from the discussion in Section 1.4).

As there is no existing definition of ordering of cells in higher dimensional objects, we will give such a definition, and prove that this ordering exists in all subdivided manifolds. In fact, the proof that this ordering exists uses the cell-tuple structure.

The cell-tuple structure represents the dual implicitly. If desired, an explicit representation of the dual may be included by adding a new set of tuples and an additional

operator. We will discuss both of these cases.

The cell-tuple structure will be initially defined for subdivided manifolds (without boundary). We will extend it to the case of subdivided manifolds-with-boundary, and prove that all of the results go through. This requires a bit more effort than one might suspect.

To be useful in algorithms for building objects, it is necessary to have some sort of operators for creating and combining objects. We will call such operators 'constructors' to distinguish them from operators for accessing topological information. They will be discussed in Chapter 5.

Methods for implementing the cell-tuple structure will be discussed in Chapter 6, considering access to incidence and ordering information, and comparing size and speed of access of the different methods against each other and against the incidence graph. A description will be given of a program written which uses the cell-tuple structure, and several algorithms will be given.

Chapter 7 will give a brief summary of this work, and suggest directions for future work.

Chapter 2

Mathematical Background

2.1 Introduction

In this chapter we will give the mathematical background required to define subdivided manifolds and the cell-tuple structure, as well as prove the two main theorems to be presented in Chapter 4. The material begins with basic topological definitions in Section 2.2. Section 2.3 introduces CW complexes, which will be needed when subdivided manifolds are defined in Section 2.4. Section 2.5 deals with simplicial complexes, which will be needed in Section 4.3 to define the 'generalized barycentric subdivision', which is essential to the proofs of the two main theorems.

2.2 Some Basic Topological Definitions

Before we can define subdivisions and proceed to their representation, we need to establish some topological background. The reader is referred to [Mun75] and [Mun84] for more details.

If Y is a subset of a topological space, we will denote the **closure** of Y by $\overline{Y}$, the **interior** of Y by Int Y and the **boundary** of Y by Bd Y. A **Hausdorff space** is a topological space such that for every pair x, y of distinct points of X, there exists a pair

of disjoint open neighborhoods U and V of x and y, respectively. Let $f : X \to Y$ be a bijection between two topological spaces. Then f is a **homeomorphism** if both f and f^{-1} are continuous. Two spaces X and Y are **homeomorphic** if there exists a homeomorphism from one to the other.

Define $\mathbb{B}^d$, the **open unit d-ball**, by $\mathbb{B}^d = \{(x_1, \ldots, x_d) \in \mathbb{R}^d \mid (x_1^2 + \cdots + x_d^2)^{1/2} < 1\}$. Its closure, $\overline{\mathbb{B}}^d = \{(x_1, \ldots, x_d) \in \mathbb{R}^d \mid (x_1^2 + \cdots + x_d^2)^{1/2} \leq 1\}$, will be called the **closed unit d-ball**. Define $\mathbb{B}^d_{1/2} = \{(x_1, \ldots, x_d) \in \mathbb{R}^d \mid (x_1^2 + \cdots + x_d^2)^{1/2} < 1 \text{ and } x_1 \geq 0\}$. A **closed k-cell** is a space which is homeomorphic with $\overline{\mathbb{B}}^k$, and an **open k-cell** is a space which is homeomorphic with $\mathbb{B}^k$. Both closed and open k-cells are said to have **dimension k**.

A nonempty Hausdorff space X is called a **d-manifold** if each point of X has a neighborhood homeomorphic with $\mathbb{B}^d$. It is called a **d-manifold-with-boundary** if each point of X has a neighborhood homeomorphic with either $\mathbb{B}^d_{1/2}$ or $\mathbb{B}^d$. The class of manifolds-with-boundary include manifolds (without boundaries), but not vice versa. Some authors include in their definition of a manifold that it have a countable basis, or at least that it be metrizable; we do not add either restriction. A point x in a manifold-with-boundary X is called a **boundary point** if its neighborhoods (of the type mentioned in the definition) are homeomorphic to $\mathbb{B}^d_{1/2}$. The set of all boundary points of X is called the **boundary** of X, denoted ∂X. Points of X which are not boundary points are called **interior** points, and the set of all interior points is denoted by Int X.

In this dissertation, we will use the boundary and interiors of k-balls, simplices (to be defined in Section 2.5), and manifolds. When we use a closed k-ball, it will always have the subspace topology as a subspace of $\mathbb{R}^k$. If a closed k-ball is considered as a manifold-with-boundary, its boundary and interior as a manifold-with-boundary agree with its boundary and interior as a subset of $\mathbb{R}^k$. The same holds for k-dimensional simplices. When the interior or boundary are taken of a general manifold-with-boundary, the manifold-with-boundary definition will always apply.

Given a set of points $x_1, \ldots, x_n$ in $\mathbb{R}^d$, their **affine hull** is the set of all points of the

form $c_1x_1 + \cdots + c_nx_n$, where $c_1, \ldots, c_n \in \mathbb{R}$ and $c_1 + \cdots + c_n = 1$. The set $x_1, \ldots, x_n$ is **affinely independent** if the removal of any point changes the affine hull. A **k-flat** in $\mathbb{R}^d$ is a set which is equal to the affine hull of $k+1$ affinely independent points.

Though graphs do not usually fall under the heading of topology, we will mention a convention regarding terminology here. When referring to graphs, we will use the terms **node** and **arc**, to better distinguish between these graphical entities and the vertices and edges that occur in subdivisions.

2.3 CW Complexes

The structures that we are interested in are partitions of manifolds and of manifolds-with-boundary. The notion that will turn out to capture the right combination of generality and structure is the 'finite, regular CW complex'[1]. This will be used in Section 2.4 to define subdivided manifolds.

A **CW complex** is a pair (X, C), where X is a topological space and C is a collection of disjoint open cells $C = \{c_\alpha\}_{\alpha \in A}$ of X whose union is X, such that:

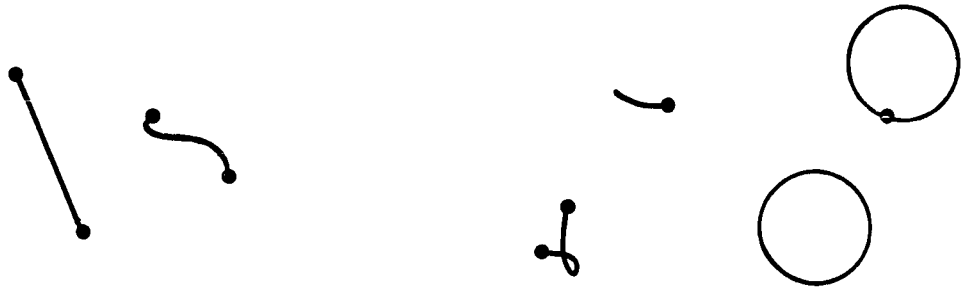
(cw1) X is Hausdorff,

(cw2) For each open k-cell c_α of the collection, there exists a continuous map $f_\alpha : \overline{\mathbb{B}}^k \to X$ that maps $\mathbb{B}^k$ homeomorphically onto c_α and carries Bd $\overline{\mathbb{B}}^k$ into a finite union of open cells of C, each of dimension less than k,

(cw3) A set Y is closed in X if $Y \cap \overline{c}_\alpha$ is closed in $\overline{c}_\alpha$ for each α.

A **finite CW complex** X is a CW complex for which the collection of open cells is finite. In this case, the finiteness part of (cw2) is automatic, and (cw3) is implied by the other conditions.

[1] J. H. C. Whitehead introduced what is now called the CW complex in 1949 [Whi49]. The 'C' stands for 'closure finiteness' (the finiteness part of condition (cw2)), and the 'W' stands for what he called the 'weak topology' (the topology given by condition (cw3)). The reader need not worry about this terminology in this dissertation.



Valid 1-cells

Not valid 1-cells

Figure 2.1: 1-cells in a regular CW complex

The **boundary** of a cell is $\dot{c}_\alpha = \overline{c}_\alpha - c_\alpha$. A CW complex X for which the maps f_α can be taken to be homeomorphisms, and for which each set $\dot{c}_\alpha$ *equals* the union of finitely many open cells of X, is called a **regular CW complex.**

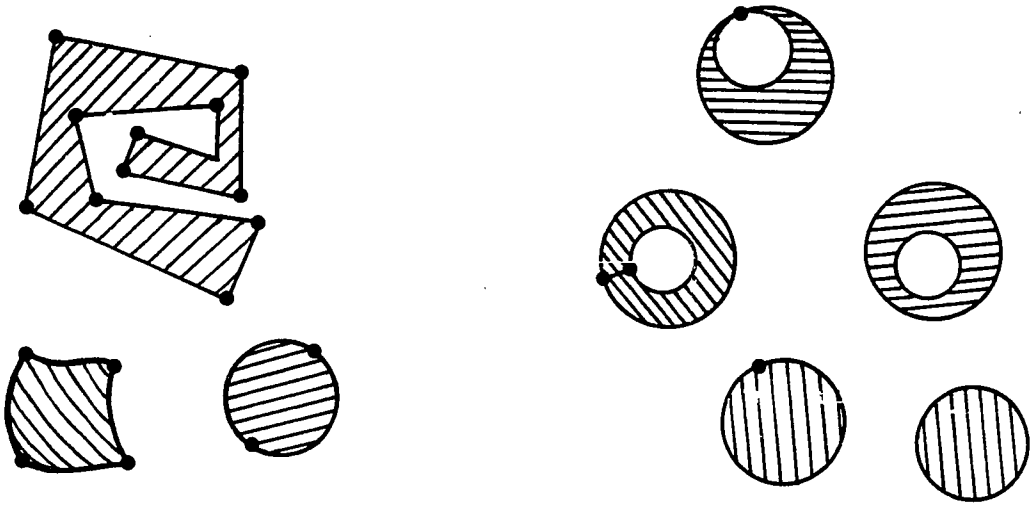
We will consider only CW complexes which are finite and regular. Note that in this case condition (cw1) could be replaced by "X is metrizable", since any finite, regular CW complex is metrizable ([LW69] Proposition II.3.8), and any metric space is Hausdorff ([Mun75], p.126).

Examples of objects which are and are not 1-cells in a regular CW complex are given in Figure 2.1, and similar examples for 2-cells in a regular CW complex are given in Figure 2.2.

If (X,C) is a CW complex, and $D \subset C$, then the **underlying space of** D is $|D| = \bigcup_{d \in D} d$. Note that $X = |C|$.

2.4 Subdivided d-Manifolds

Manifolds and manifolds-with-boundary are general enough to include most objects which are likely to be modeled in practice, so what we will consider are partitions of



Valid 2-cells

Not valid 2-cells

Figure 2.2: 2-cells in a regular CW complex

these. When deciding on what type of partition, the finite, regular CW complex seems to be the right choice, in that it allows the proofs of the two main theorems of this work.

We define a **subdivided d-manifold** (definition 1) to be a pair (M, C), where M is a d-manifold and (M, C) is a finite, regular CW complex.

An alternative definition which eliminates redundant conditions is: if M is a d-manifold and $C = \{c_\alpha\}_{\alpha \in I_C}$ is a finite collection of disjoint open cells whose union is M, then the pair (M, C) is a **subdivided d-manifold** (definition 2) if for each open k-cell c_α of C, there exists a continuous map $f_\alpha : \overline{\mathbb{B}}^k \to X$ such that:

(sm1) f_α maps $\mathbb{B}^k$ homeomorphically onto c_α,

(sm2) f_α maps $\overline{\mathbb{B}}^k$ homeomorphically onto $\overline{c}_\alpha$,

(sm3) $f_\alpha(\text{Bd } \overline{\mathbb{B}}^k)$ is a union of open cells of C, each of dimension less than k.

Claim 2.1 *The two definitions of subdivided d-manifold are equivalent.*

Proof In each definition, M is a d-manifold, and $C = \{c_\alpha\}_{\alpha \in I_C}$ is a finite collection of disjoint open cells.

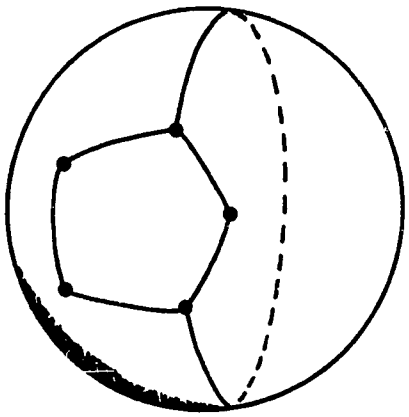


Figure 2.3: A subdivided 2-manifold where M is a 2-sphere

Suppose (M, C) is a subdivided manifold under definition 1. Then (sm1) follows from (cw2), (sm2) follows from regularity and the fact that $f_\alpha(\overline{\mathbb{B}^k}) = \bar{c}_\alpha$ ([Mu 84] pp. 215), and (sm3) follows from (cw2) and the fact that $f_\alpha(\text{Bd } \overline{\mathbb{B}^k}) = \dot{c}_\alpha$ ([Mu 84] pp. 215).

Suppose (M, C) is a subdivided manifold under definition 2. That M is Hausdorff is true by the definition of manifold, so condition (cw1) holds. Condition (cw2) is immediate from (sm1), (sm3) and the fact that C is finite. Condition (cw3) follows from (cw1) and (cw2) ([Mu 84] pp. 215). That (M, C) is regular follows from (sm2), (sm3) and the fact that C is finite. ∎

The definition of subdivided d-manifold-with-boundary is the same as that for subdivided d-manifold, except that M is a d-manifold-with-boundary.

We will refer to the open cells in subdivided manifolds simply as cells.

Figure 2.3 shows a subdivision of a 2-sphere into five vertices (0-cells), six edges (1-cells), and three faces (2-cells). Figure 2.4 shows a subdivision of a Klein bottle, partitioned into four vertices, eight edges, and four faces. Figure 2.5 shows a subdivided 3-manifold with boundary, which might be constructed by identifying one face of a cube with a face of a triangular prism. This object could be made into a subdivided 3-manifold (without boundary) by adding another 3-cell whose boundary is just the boundary of

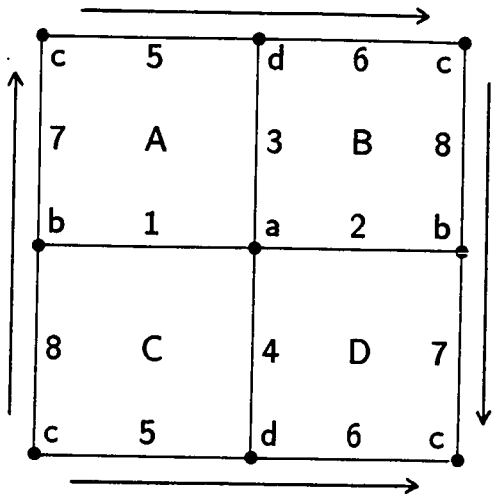


Figure 2.4: A subdivided 2-manifold where M is a Klein bottle

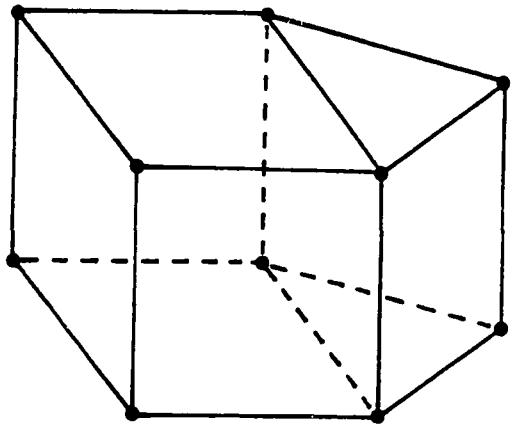


Figure 2.5: A simple subdivided 3-manifold-with-boundary

the given object; this may be embedded in $\mathbb{R}^4$.

The dimension of a cell $c_\alpha \in C$ will be denoted by $\dim(c_\alpha)$. We will write $c_{\alpha_1} < c_{\alpha_2}$ if $c_{\alpha_1} \subseteq \dot{c}_{\alpha_2}$, and will say that c_{α_1} and c_{α_2} are **incident**, and that c_{α_1} is a **subcell** of c_{α_2}. Note that $<$ is a strict partial ordering of C. We will write $c_{\alpha_1} \prec c_{\alpha_2}$ when $c_{\alpha_1} \leq c_{\alpha_2}$ and $\dim(c_{\alpha_2}) = \dim(c_{\alpha_1}) + 1$. Define a function giving the index of a cell by $index(c) = \alpha$ if and only if $c = c_\alpha$. If a k-cell $c_\alpha \subseteq \partial M$, we will say it is a **boundary k-cell.**

For the remainder of this dissertation (M, C) and (N, D) will be subdivided d-manifolds with $C = \{c_\alpha\}_{\alpha \in I_C}$ and $D = \{d_\beta\}_{\beta \in I_D}$, unless specified otherwise.

Two subdivided d-manifolds are **equivalent**, $(M, C) \simeq (N, D)$, if there is a homeomorphism between M and N carrying k-cells onto k-cells. (N, D) is a **refinement** of (M, C) if every cell of D is contained in a cell of C and every cell of C is a finite union of cells of D. If $D \subset C$, $N \subset M$, (and (N, D) is a subdivided k-manifold, where $0 \leq k \leq d$), then (N, D) will be called a **subcomplex** of (M, C).

For notational convenience, we assume the existence of a cell c_{-1} of dimension -1 and a cell c_{d+1} of dimension $d + 1$, such that $c_{-1} < c_\alpha < c_{d+1}$ for all $c_\alpha \in C$. It is stressed that this is notation only, and these cells are never actually used. For example, when we write $c_{\alpha_{k-1}} \prec c_{\alpha_k} \prec c_{\alpha_{k+1}}$, where $\dim(c_{\alpha_i}) = i$, it will be understood that if $k = 0$, this simply means $c_{\alpha_0} \prec c_{\alpha_1}$, and if $k = d$, it simply means $c_{\alpha_{d-1}} \prec c_{\alpha_d}$.

2.5 Simplicial Complexes

Simplicial complexes will be needed when we define the generalized barycentric subdivision in Section 4.3. The generalized barycentric subdivision will give the bridge between the combinatorial representation given by cell-tuple structures and the topological structure of subdivided manifolds.

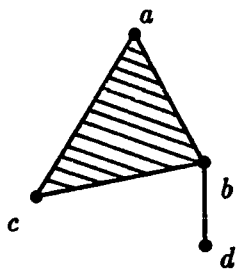
A **k-simplex**, $k \geq 1$, is the convex hull (defined in Section 1.2) of $k + 1$ affinely independent points in $\mathbb{R}^d$. A **face** of a simplex is a simplex formed by a subset of the simplex's vertices. We will use the following notation to identify particular simplices: If $v_{\alpha_0}, \ldots, v_{\alpha_k}$ are points in $\mathbb{R}^d$, then $\sigma(\alpha_0, \ldots, \alpha_k)$ is the simplex defined by $v_{\alpha_0}, \ldots, v_{\alpha_k}$.

$\{ \{a\}, \{b\}, \{c\}, \{d\},$

$\{a,b\}, \{a,c\}, \{b,c\}, \{b,d\},$

$\{a,b,c\} \}$

$\mathcal{A}$



$\mathcal{K}$

Figure 2.6: An abstract simplicial complex and a realization

A **simplicial complex** K in $\mathbb{R}^d$ is a collection of simplices in $\mathbb{R}^d$ such that:

- Every face of a simplex of K is in K.
- The intersection of any two simplices of K is a face of each of them.

A subset of a simplicial complex which is itself a simplicial complex is called a **subcomplex.** The **polytope** (or **underlying space**) of a simplicial complex K is $|K| = \bigcup_{\sigma \in K} \sigma$. (To distinguish between this and the cardinality of a set, we will use $\#(S)$ to denote the cardinality of S.)

If K_1 and K_2 are simplicial complexes, a **simplicial map** between them is a map $f : |K_1| \to |K_2|$ such that f is linear on each simplex of K_1, and f maps simplices to simplices.

An **abstract simplicial complex** $\mathcal{A}$ is a collection of finite non-empty sets, so that if A is an element of the collection, so is every subset of A. The **vertex scheme** of a simplicial complex is the abstract simplicial complex defined by the vertex sets of its simplices. A **geometric realization** of an abstract simplicial complex $\mathcal{A}$ is any simplicial complex whose vertex scheme is $\mathcal{A}$. An example is given in Figure 2.6.

A **triangulation** of a space X is a simplicial complex K together with a homeomorphism $h : |K| \to X$.

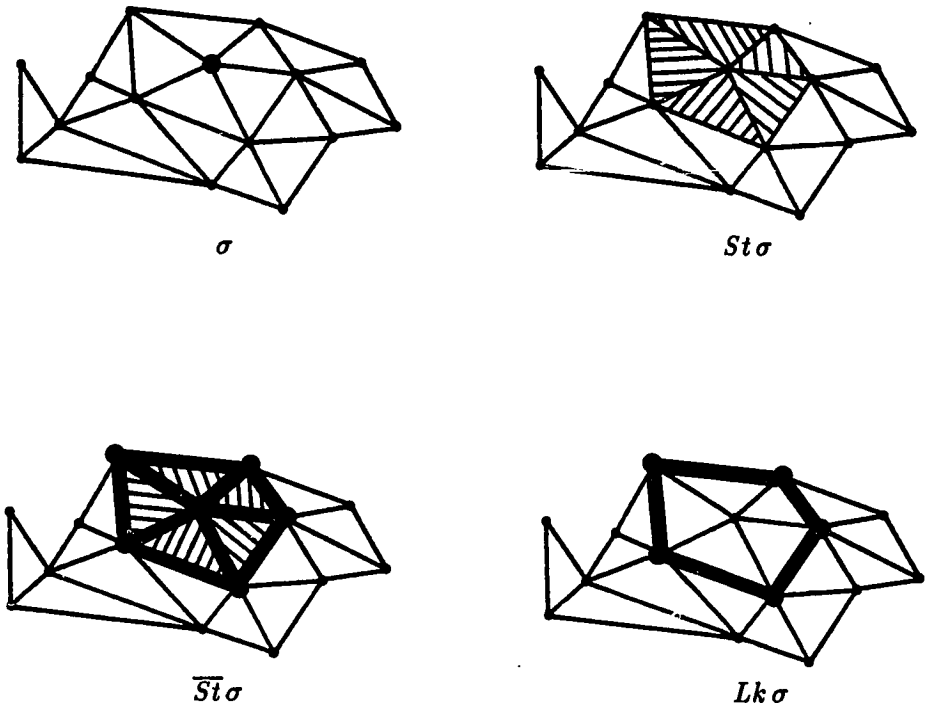


Figure 2.7: Examples of the star, closed star and link

The following definitions of the star, closed star, and link of a simplex will only be used in the proofs of Lemma 4.6 in Section 4.3 and Lemma 4.11 in Section 4.4. If σ is a simplex in K, then the **closed star** of σ is $\overline{\mathbf{St}}\,\sigma = \{\sigma' \in K \mid \exists \sigma'' \in K \text{ s.t. } \sigma \text{ is a face of } \sigma''$ and σ' is a face of $\sigma''\}$. In words, $\overline{St}\,\sigma$ is the set of all faces of simplices that have σ as a face. The **star** of σ is $\mathbf{St}\,\sigma = \{\text{Int } \sigma' \in K \mid \sigma' \in \overline{St}\,\sigma\}$, that is $St\,\sigma$ is the set of interiors of simplices in $\overline{St}\,\sigma$. The **link** of σ is $\mathbf{Lk}\,\sigma = \{\sigma' \in K \mid \sigma' \in \overline{St}\,\sigma$ and $\sigma' \cap \sigma = \emptyset\}$, that is $Lk\,\sigma$ is the set of simplices in $\overline{St}\,\sigma$ which are disjoint from σ. (See Figure 2.7.)

Chapter 3

Review of Previous and Related Work

3.1 Introduction

In this chapter we will review relevant existing work. This includes a general overview of representations of subdivisions in various areas of computer science, descriptions of the representations which use the new (implicit-cell) approach mentioned in the introduction, and some independent work which is similar to the cell-tuple structure. The depth of detail will be only that necessary to give an intuitive idea of the results, or as will be necessary in later discussions.

3.2 The Incidence Graph

The incidence graph mentioned in Section 1.5 is probably the most commonly used representation of topological structure in computational geometry. This is appropriate for any structure having a reasonable notion of incidence. The basic idea is that there is a node for every cell in a complex, and an arc connecting two nodes if the cells they represent are incident and differ in dimension by one. Alternatively, one can use

directed arcs, classified as 'upward-pointing' and 'downward-pointing'. For each node, the upward-pointing arcs point to nodes whose cells are incident to the given node's and have dimension one higher. Similarly, the downward-pointing arcs point to nodes whose cells are incident to the given node's and have dimension one lower. This gives a straight-forward implementation in which the nodes have two associated lists of pointers.

The incidence relation, when taken with the cells' dimensions, may be used to define a partial ordering on the set of cells in a subdivided manifold. This partial order may be represented in exactly the way just described. It is very useful to have this partial ordering available in the work to be described.

Given a subdivided d-manifold (M, C), where $C = \{c_\alpha\}_{\alpha \in I_C}$, define an order relation $<$ on the index set I_C by $\alpha_1 < \alpha_2$ if and only if $c_{\alpha_1} < c_{\alpha_2}$. This is a strict partial order on I_C since it is on C. Define a labeling of the indices by the dimension of their cells, $\dim(\alpha) = \dim(c_\alpha)$. As with cells, define $\prec$ by $\alpha_1 \prec \alpha_2$ if $\alpha_1 < \alpha_2$ and $\dim(\alpha_2) = \dim(\alpha_1)+1$. The resulting partially ordered set should be called the 'incidence poset' of M, but tradition dictates that the it be referred to as the **incidence graph.**

Thus the incidence graph, $\mathcal{I}_M$, of a subdivided manifold (M, C), is the graph whose nodes are the elements of the index set I_C, with directed arcs between them as described above, and the labeling dim and the partial order relation $<$ just described. The incidence graph will be drawn by drawing the Hasse diagram of the partial order. Figure 3.1 gives another depiction of the subdivision shown in Section 2.4. Figure 3.2 shows its incidence graph.

An **ascending chain of length** ℓ in the incidence graph is a sequence $\alpha_{i_1} < \cdots < \alpha_{i_\ell}$. A **full ascending chain** in the incidence graph is an ascending chain of length $d+1$, i.e. a maximal ascending chain $\alpha_{i_0} \prec \cdots \prec \alpha_{i_d}$.

Two incidence graphs are **equivalent**, $\mathcal{I}_M \simeq \mathcal{I}_N$, if there is a one-to-one map from I_C to I_D such that the order relation and dim are preserved.

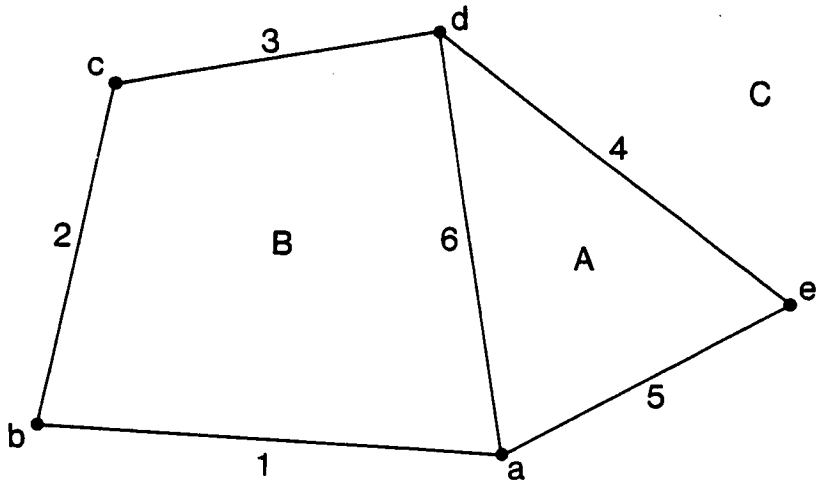


Figure 3.1: A subdivision of S^2

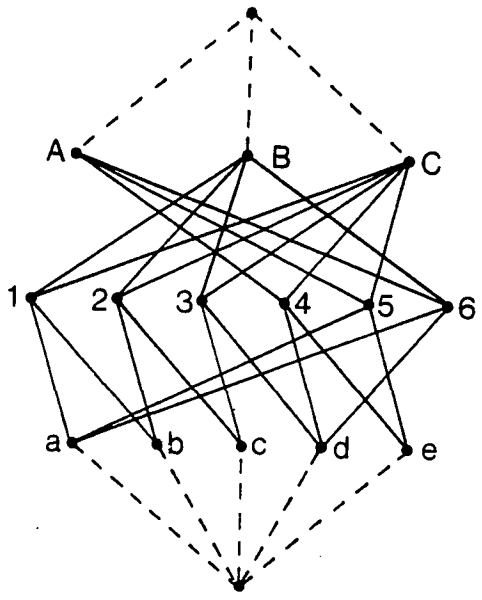


Figure 3.2: Example of the incidence graph

3.3 Representations of Subdivided Surfaces

3.3.1 Introduction

Most work in computational geometry has been concerned with 2-dimensional objects. It is natural that this should be so for representations, as one of the main motivations for this area of research is the modeling of solid (3-dimensional) objects, and the boundaries of such objects are 2-dimensional. It is also natural because they are the most well-understood – the mathematical foundations are strong and long-standing.

3.3.2 Edge-Based Approaches

The idea of the edge-based approach is to associate all of the topological information with edges. As was mentioned in the introduction, edge-based approaches were given by Baumgart [Bau75], Danaraj and Klee [DK78], Muller and Preparata [MP78], and others.

We will give a simple description of Baumgart's edge-based scheme, as this is the most often cited in the literature of which the present work is a part. Any of the other similar methods would serve equally well. Baumgart assumes that the objects being modeled are the surfaces of polyhedra, ensuring that each edge is incident to two distinct vertices and two distinct faces, that the surface is orientable, and that the concepts of inside and outside are well-defined.

To each edge e an arbitrary direction is assigned. Then the incident vertices may be distinguished by which end they lie on with respect to the direction of the edge, and are identified[1] as PVT(e) and NVT(e). Similarly, the two incident faces may be distinguished as each lies to the left or right of the edge (as seen from the outside), identified as NFACE(e) and PFACE(e). There are four edges adjacent to e and also incident to one of these two faces. These are identified as NCW(e), NCCW(e), PCW(e), and PCCW(e). (See Figure 3.3.)

Each of PVT(e) and NVT(e) points to a record containing information about that

[1]Baumgart does not give mnemonics for these identifiers.

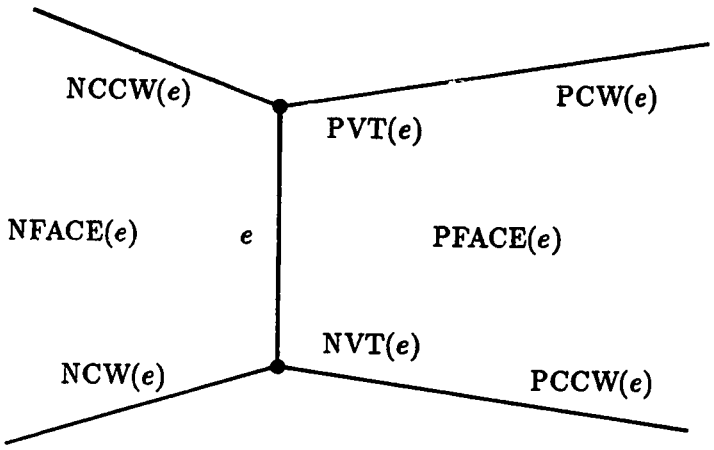


Figure 3.3: Elements associated with an edge in the winged-edge data structure

vertex, such as its coordinates, as well as a pointer to one of its incident edges. Similarly, each of PFACE(e) and NFACE(e) points to a record containing information about that face, such as its normal vector, and a pointer to one of its incident edges.

This allows moving around on the surface by following pointers to adjacent edges, and allows access to all the elements incident to a vertex, edge, or face to be read off in time linear in the number of elements. The only inconvenience is that when moving around from edge to edge, say following the edges incident to a face, tests must be made to decide which of NCW, NCCW, PCW, or PCCW is the desired edge, because of the arbitrary assignment of direction.

For a comprehensive work on the edge-based and similar approaches, the reader is referred to Weiler [Wei85], who analyzes and compares a number of approaches, using the possible adjacency relationships among vertices, edges and faces.

3.3.3 The Quad-Edge Data Structure

Guibas and Stolfi [GS85] introduced the 'edge algebra' for the representation of 'subdivisions' of 2-manifolds. Their definition of a subdivision S is a partition of a 2-manifold into a finite number of disjoint sets – vertices, edges and faces – called the 'elements' of S. A face is required to have the property that its boundary is a closed path of edges and vertices. This means, among other things, that faces are simple disks (i.e. are homeomorphic to $\mathbb{B}^2$), without holes or handles. Note that though the faces themselves are open disks, their closures (the face plus its boundary) may not be closed disks, because the path of edges along a boundary may contain the same edge or vertex more than once. Two subdivisions are 'equivalent' if there is a homeomorphism between the corresponding 2-manifolds which maps vertices to vertices, edges to edges, and faces to faces.

The original subdivision and all of its elements will be called 'primal'. The 'dual subdivision' is produced by adding a new (dual) vertex in the middle of every face, creating a new (dual) edge for every edge and creating a new (dual) face for every vertex. Two elements of the dual subdivision are incident if and only if their corresponding elements in the primal subdivision are incident.

To produce the edge algebra, four new objects are created for every edge in the primal and dual, which correspond to the four ways of giving direction and orientation to the edge. The authors explain this by describing a bug straddling the edge, which may be facing along the edge in either of two ways, and may be on either 'side' of the surface. We will describe it slightly differently. Think of yourself sitting in the plane – the direction tells you which way you are facing, and the orientation tells you which of your two hands is your left and which is your right. (See Figure 3.4.)

The set of directed, oriented edges corresponding to primal edges of S is denoted by ES, and the set of directed, oriented edges of the dual by ES^*. We will use the term do-edge for a directed, oriented edge (Guibas and Stolfi use the term edge for both edges and directed, oriented edges). Given any do-edge in ES or ES^*, the operation *Flip* will

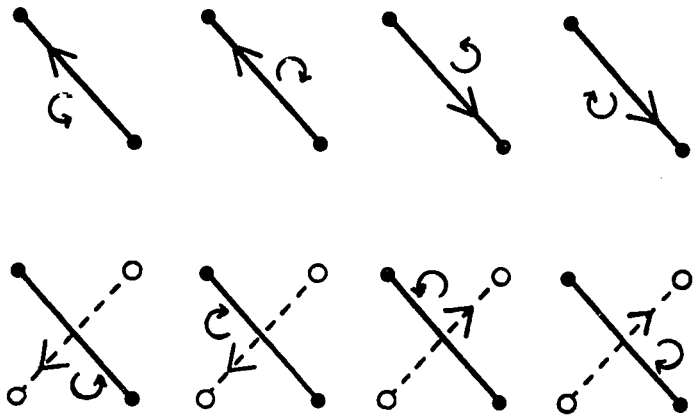


Figure 3.4: The eight directed, oriented edges associated with an edge

produce the do-edge pointing in the same direction but with the reverse orientation. The operation *Rot* will essentially rotate a do-edge (in the direction indicated by the orientation) about its midpoint by 90 degrees, giving one of the dual's do-edges with the same orientation. The operation *Onext* essentially rotates a do-edge (in the direction indicated by the orientation) about its tail onto the first primal do-edge it encounters. (See Figure 3.5.)

Given a do-edge e, let f be the face 'to the left' of e. The next do-edge around f, in the direction given by e, which has the same direction and orientation, is $e\ Rot^2\ Onext^{-1}$. Applying $Rot^2\ Onext^{-1}$ repeatedly allows access to the do-edges incident to f in order around the boundary. Repeated applications of *Onext* gives a similar ordered 'ring' of do-edges incident to a vertex.

It is proven that for any given do-edge e the following rules apply:

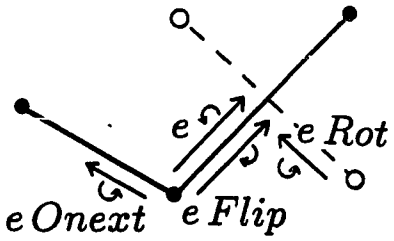


Figure 3.5: Examples of *Rot*, *Flip* and *Onext*

E1)	$eRot^4 = e$,	F1)	$eFlip^2 = e$,
E2)	$eRot\ Onext\ Rot\ Onext = e$,	F2)	$eFlip\ Onext\ Flip\ Onext = e$,
E3)	$eRot^2 \neq e$,	F3)	$eFlip\ Onext^n \neq e$ for any n ,
E4)	$e \in ES \Leftrightarrow eRot \in ES^*$,	F4)	$eFlip\ Rot\ Flip\ Rot = e$,
E5)	$e \in ES \Leftrightarrow eOnext \in ES$,	F5)	$e \in ES \Leftrightarrow eFlip \in ES$.

The 5-tuple $(ES, ES^*, Onext, Rot, Flip)$ obtained from the subdivision is called the 'standard edge algebra' of the subdivision. Given arbitrary finite sets E and E^*, and operations satisfying these rules, $(E, E^*, Onext, Rot, Flip)$ is called an 'edge algebra'.

It is proved that two subdivisions are equivalent if and only if their edge algebras are isomorphic. In addition, they prove that every (abstract) edge algebra has a realization as a subdivision.

There is only one basic element in the edge-algebra, the directed, oriented edge. Vertices are represented by 'double rings' (if v is a vertex, and e is a do-edge incident to v and directed away from v, then there are two rings: one obtained by repeated applications of *Onext* to e, and one obtained by repeated applications of *Onext* to $eFlip$). Edges are represented by four do-edges (given edge e, applying *Flip* and Rot^2 repeatedly gives a set of four do-edges). Faces are represented by 'double rings' (if f is a face, and e is a do-edge incident to v so that f is on e's left side, then there are two rings: one

obtained by repeated applications of $Rot^2\ Onext^{-1}$ to e, and one obtained by repeated applications of $Rot^2\ Onext^{-1}$ to $e Flip\ Rot^2$.

They also introduce two simple operations on subdivisions, '*MakeEdge*' and '*Splice*', for creating and modifying edge algebras. These will be discussed in Section 5.2.2.

A data structure is introduced for representing the edge algebra, and hence for representing the topological structure of subdivided manifolds, which is as simple and clean as the edge algebra itself. The basic unit is called a 'quad-edge', and represents the set of eight do-edges associated with an edge. Since they have proved that the edge algebra captures all of the topological properties of the subdivided manifold, the data structure does also.

One of the nice features of the edge algebra is that the dual subdivision is also represented, and is represented in a completely symmetric fashion. Thus one can ask questions about the dual, or move around in the dual, in exactly the same way as one would in the primal subdivision. When one makes changes to the edge algebra (or the data structure), the dual and the primal subdivision are both updated simultaneously.

The class of subdivisions studied in [GS85] are those whose underlying spaces are manifolds (without boundary), in which the boundaries of cells may intersect (as described earlier in this section). Their definition allows them to prove that every abstract edge algebra has a realization. However, to carry this over when generalizing to higher dimensions, the necessary class of objects includes some bizarre objects, and the ordering results will not apply. The class of subdivisions studied in this dissertation does not allow cell-boundaries to intersect, but does include 2-manifolds-with-boundary.

3.4 Representations of Three-Dimensional Subdivisions

3.4.1 Introduction

This author knows of two generalizations of the quad-edge data structure to representations of 3-dimensional subdivisions. Dobkin and Laszlo [DL87] developed the 'facet-edge

data structure', and Buckley [Buc88] briefly sketches the 'Hexblock' data structure, which he states is similar to the facet-edge data structure. We will describe the former, as their presentation goes into considerably more detail.

3.4.2 The Facet-Edge Data Structure

Dobkin and Laszlo [DL87] created a data structure for representing complexes of 3-cells, in an analogous fashion to that of Guibas and Stolfi's 2-dimensional data structure. The basic units in their work are what they call facet-edge pairs, which are pairs of the form (e, f) such that the edge e is incident to the face f. Since the (topological) dual of an edge is a face, and vice versa, there is a one-to-one correspondence between facet-edges in the primal and dual subdivisions. As in the quad-edge data structure, the idea of orientation is now added. There are two possible directions to rotate around an edge, and two possible directions to traverse the boundary of a face. With each facet-edge pair are associated four objects, corresponding to the four possible ways of assigning these directions.

There are five operators for connecting facet-edges: *Enext*, *Fnext*, *Clock*, *Rev*, and *Sdual*. *Enext* gives a pair with the same face component, but with the next edge around the face in the direction given by the face's orientation, and leaves the orientations unchanged. *Fnext* gives a pair with the same edge component, but with the next face around the edge in the direction given by the edge's orientation, and leaves the orientations unchanged. *Clock* and *Rev* leave both components unchanged. *Clock* changes both orientations, while *Rev* changes only the edge's orientation. *Sdual* gives the corresponding facet-edge in the dual, such that the orientation of the dual face matches that of the primal edge, and the orientation of the dual edge matches that of the primal face. Pictorially, the first four of these operators are shown in Figure 3.6.

The set of all facet-edges, taken with the five operators above, constitutes the facet-edge data structure. Four operators are given for creating and manipulating facet-edge date structures: *make_facet_edge*, *splice_facets*, *splice_edges*, and *transfer*. The class of 3-

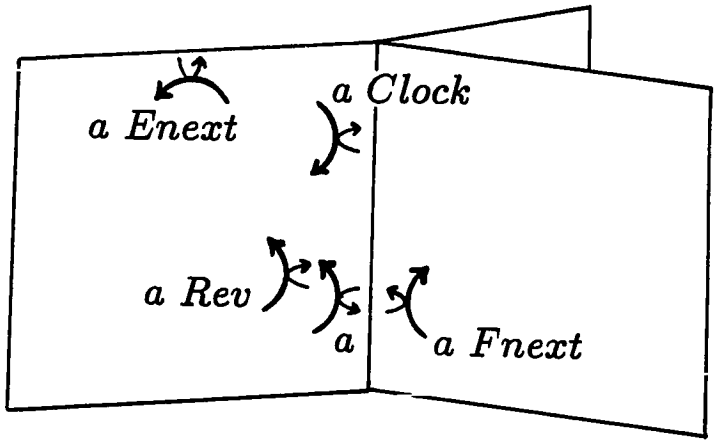


Figure 3.6: A facet-edge pair and primitive operators

complexes explicitly allowed in [DL87] are subdivisions of spaces homeomorphic to $\overline{\mathbb{B}}^3$ or $\mathbb{B}^3$, where the cell boundaries are allowed to self-intersect, under appropriate restrictions. (It should be noted that, unlike the operators in the quad-edge data structure, the four manipulative operators given for the facet-edge data structure may be used to build a much larger class of objects than the two stated above. See Section 5.2.3)

3.5 Related Work in Higher Dimensions

Two other researchers have described, independently of each other and of this work, representations of d-dimensional objects which are similar to the cell-tuple structure. Lienhardt is a computer scientist whose work with 'n-G-maps' is concerned with representations of subdivisions of higher dimensional objects suitable for computational applications, while Tits is a mathematician, who developed 'chamber systems' as a tool for working with mathematical objects called 'buildings'.

3.5.1 n-G-maps

Lienhardt [Lie89] develops 'n-G-maps', which represent a larger class of partitions of topological spaces than subdivided manifolds (note that he also uses the term subdivision). The basic definition (taken directly from his paper) is:

> Definition 1: We define then an n-G-map in a combinatorial way, by: let $n \geq 0$, an *n-G-map* is defined by an $(n+2)$-tuple $G = (B, \alpha_0, \alpha_1, \ldots \alpha_n)$, such as:
>
> - B is a finite, not empty set of *darts*;
> - $\alpha_0, \alpha_1, \ldots \alpha_n$ are permutations on B, such as:
>   - $\forall i \in \{0, \ldots n-1\}, \alpha_i$ is an involution without fixed points, i.e.:
>
> $\forall b \in B, \alpha_i(b) \neq b$, and $\alpha_i^2(b) = b$ (α_i gathers two distinct darts);
>   - α_n is an involution, i.e.:
>
> $\forall b \in B, \alpha_n^2(b) = b$ (α_n gathers at most two darts);
>   - $\forall i \in \{0, \ldots n-2\}, \forall j \in \{i+2, \ldots n\}, \alpha_i\alpha_j$ is an involution (dependence of the involutions: each α_j ($j \geq 2$) gathers two by two (j–2)-dimensional subdivisions).
>
> If α_n is an involution without fixed points, then G is said to be *closed*, else G is said to be *open*.

Here, n is the dimension. To associate an n-G-map with a topological object, first consider $n = 1$. In this case, darts are defined to be half-edges (an edge is split into two halves by a point in the edge's interior). Two half-edges 'gathered' by α_0 form an edge, and two edges are adjacent at an endpoint if the respective half-edges are gathered by α_1. Thus a 1-G-map corresponds to a set of edges forming cycles and paths. A connected closed 1-G-map corresponds to a single cycle, and a connected open 1-G-map corresponds to a single path. (See Figure 3.7.) In two dimensions, a face is defined

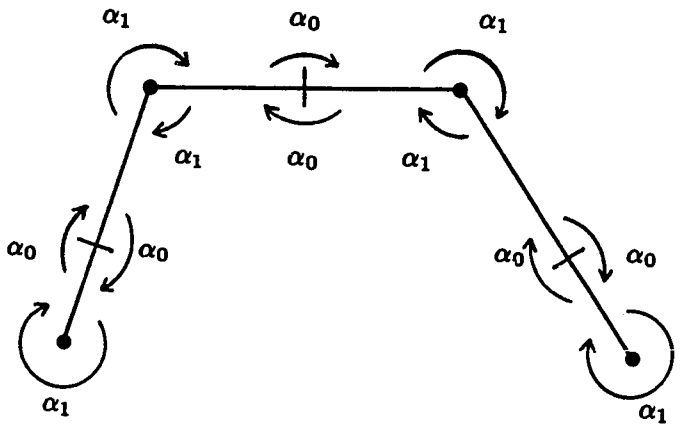


Figure 3.7: Example of a 1-G-map

by 'a simple elementary cycle of edges'. These faces may be 'sewn' together, to form 2-dimensional subdivisions. In general, any connected closed $(n-1)$-G-map is allowed to be the boundary of an n-cell, and such cells may be sewn together to create n-dimensional subdivisions.

Lienhardt discusses the orientability of n-G-maps and gives a definition for the dual of an n-G-map. In [Lie88], he shows how to compute the number of boundaries, the Euler characteristic, orientability factor and genus of 2-G-maps (representing surfaces).

Note that what is referred to as a cell is very general. For instance, a torus or a Klein bottle might be the boundary of a 3-dimensional cell. So the underlying spaces created are very general, i.e. are not manifolds, or any class of 'familiar' space. The advantage to this is that the allowed (gathering) operations always give another valid object. The disadvantage is that less can be said about them, i.e. the ordering results presented in this dissertation do not apply.

3.5.2 Chamber Systems

Tits' work [Tit81] is concerned with the characterization of combinatorial 'buildings', which arose out of the geometric study of semisimple Lie groups and semisimple algebraic groups. We will give a brief description of chamber systems here, presented so as to

facilitate the connection made in Section 4.9.5 between chamber systems and the cell-tuple structure. In particular, his work is developed in a much more general setting than we give here, and is not concerned with subdivided manifolds. He does not consider the representation of subdivided manifolds or ordering, and does not consider computational issues.

In what follows, I is an arbitrary set. A 'geometry over the set I' consists of a set of elements, together with an incidence relation between them, and a labeling of the elements by the members of I. This labeling is called the 'type' of the element. Elements of a geometry are incident to themselves, and may not be incident to any other elements of the same type. The kind of 'complexes' considered in [Tit81] are similar to the abstract simplicial complexes defined in Section 2.5. For the purposes of this discussion, it is enough to let the set I be the dimensions $0, \ldots, d$, and to consider a 'complex over I' to be an abstract simplicial complex whose vertices are labeled from the set I.

To every geometry over I may be associated a complex over I, which is essentially the set of ascending chains of incident elements in the geometry. The 'vertices' of this complex are the ascending chains of length one. Such a vertex is given the same label as the geometric element it was derived from.

A 'chamber system over the set I' is a pair $(\mathcal{C}, (\mathcal{P}_i)_{i \in I})$, where the set $\mathcal{C}$ is an arbitrary set of 'chambers', and $(\mathcal{P}_i)_{i \in I})$ is a family of partitions of $\mathcal{C}$.

The chamber system associated with a complex has as its chambers the set of simplices with vertices labeled by all elements of I, i. e. the simplices are maximal ascending chains. Two chambers are in the same partition $\mathcal{P}_i$ if they have the same face (subsimplex) which is labeled by $I - \{i\}$.

We will see in Section 4.9.5 that the chamber system, when appropriately applied to subdivided manifolds, is essentially the same as the edge-labeled graph characterization of the cell-tuple structure.

Chamber systems are used by Huson and his co-authors [DH87], [FH89] in their study

of tilings of the plane. Huson (who brought Tits' work to this author's attention) has developed an implementation of 2-dimensional chamber systems as part of his work.

Chapter 4

The Cell-Tuple Structure

4.1 Introduction

In this chapter we will define the cell-tuple structure, a simple, general data structure for representing subdivided manifolds. The cell-tuple structure is the major conceptual component of this dissertation work. We will prove two theorems which establish important properties of the cell-tuple structure. They are the major results of the work. Theorem 4.4 shows that the cell-tuple structure is powerful enough to represent subdivided manifolds up to equivalence; and Theorem 4.5 shows the existence of ordering of cells for all dimensions k between 0 and d, and that this ordering is directly accessible using the cell-tuple structure.

We will initially define the cell-tuple structure and prove the two theorems only for manifolds (without boundary), and treat manifolds-with-boundary in a separate section. We will also discuss the dual subdivision, and relate the cell-tuple structure to other implicit-cell representations.

4.2 The Cell-Tuple Structure

In this section we will define the cell-tuple structure of a subdivided d-manifold and state Theorem 4.4 and Theorem 4.5.

If (M, C) is a subdivided d-manifold, let $T_M = \{(c_{\alpha_0}, \ldots, c_{\alpha_d}) \mid c_{\alpha_i} \in C, c_{\alpha_0} \prec \cdots \prec c_{\alpha_d}\}$. The $(d+1)$-tuples of T_M will be called **cell-tuples**. To extract the k-component of a cell-tuple $t \in T_M$, where $t = (c_{\alpha_0}, \ldots, c_{\alpha_d})$, we write $t_k = c_{\alpha_k}$.

The following lemma says that for any given cell-tuple, if one of its components is specified, then there is a unique cell-tuple which differs from the given cell-tuple in the specified component and agrees on all other components. This will be proved in Section 4.4.

Lemma 4.2 *If (M, C) is a subdivided d-manifold, $t \in T_M$ and $0 \leq k \leq d$, then there is a unique $t' \in T_M$ such that $t'_k \neq t_k$ and $t'_i = t_i$ for all $i \neq k$.*

For $0 \leq k \leq d$, define $switch_k : T_M \to T_M$ by $switch_k(t) = t'$, where t' is such that $t'_k \neq t_k$ and $t'_i = t_i$ for $i \neq k$, as given by Lemma 4.2.

When showing examples of the cell-tuple structure, we will draw the cell-tuples as dots located within their d-dimensional component, and near to their other components. See Figure 4.1 for examples of $switch_0(t)$, $switch_1(t)$, and $switch_2(t)$ where $t = (a, 6, B)$.

The **cell-tuple structure** $\mathcal{T}_M$ for (M, C) is the pair $(T_M, \{switch_k\}_{0 \leq k \leq d})$. (There is, of course, redundancy in writing this as a pair, as the $switch_k$ operators are derivable from the set of cell-tuples T_M.) We will usually abuse the notation and simply write $switch$ for the set of operators $\{switch_k\}_{0 \leq k \leq d}$, e.g. the cell-tuple structure will be written as $(T_M, switch)$.

It is useful, and helpful for intuition, to think of the cell-tuple structure as an undirected edge-labeled graph G_M, which has a node for each cell-tuple, and a k-edge between nodes whose cell-tuples are related by $switch_k$. That is, $G_M = (V, E)$, where $V = T_M$, $E_k = \{(t^1, t^2) \mid switch_k(t^1) = t^2\}$, and $E = \cup_{0 \leq k \leq d} E_k$. An edge $e \in E$ is labeled with k if $e \in E_k$. These undirected edges are well-defined, because Lemma 4.2 implies that

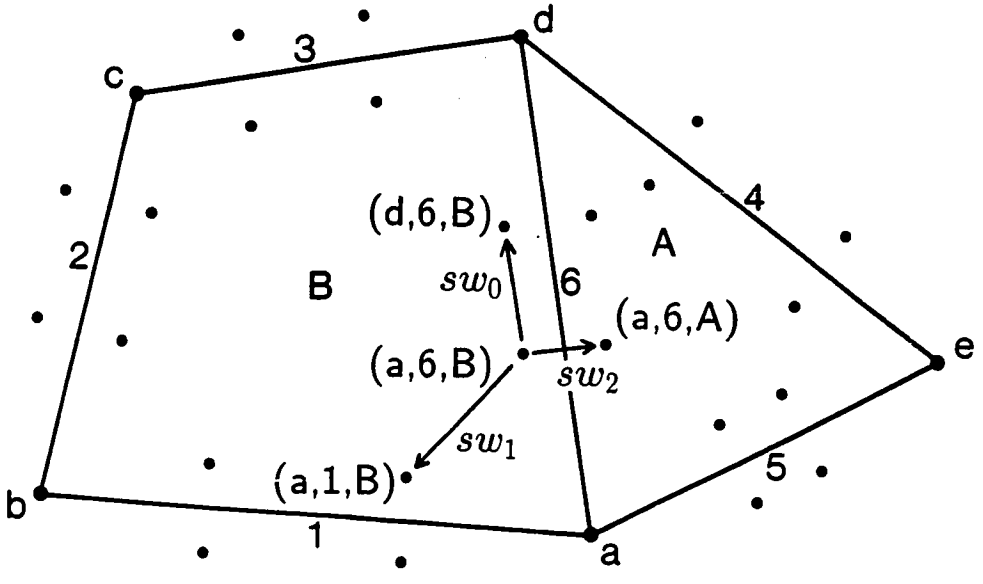


Figure 4.1: Examples of *switch*

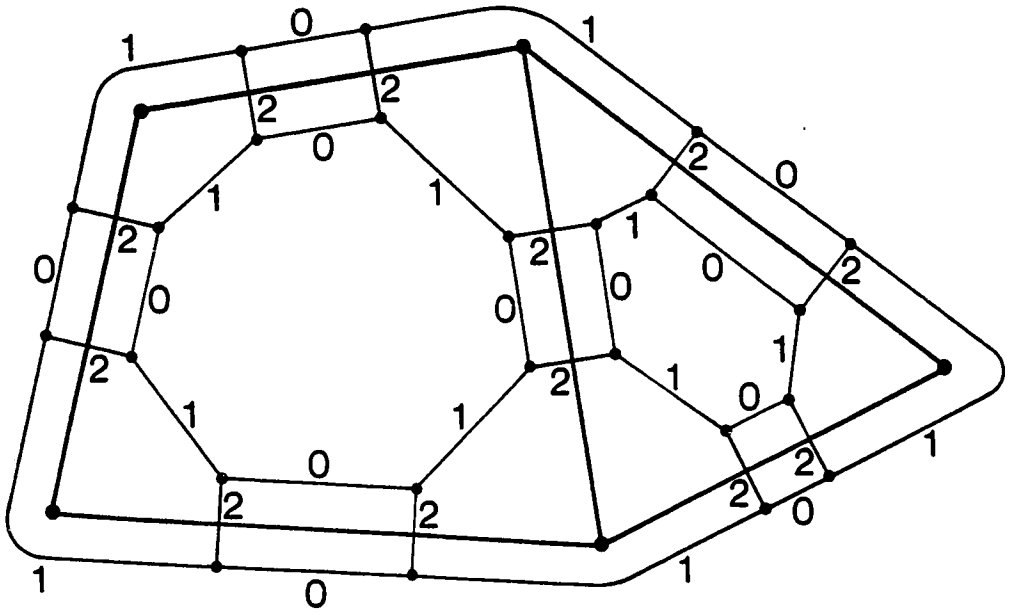


Figure 4.2: Example of G_M

$switch_k(t^1) = t^2$ if and only if $switch_k(t^2) = t^1$. This labeling is well-defined, because the E_k's are disjoint: $k \neq \ell \Rightarrow [switch_k(t)]_k \neq [switch_\ell(t)]_k$.

Note that every vertex of this graph is incident to exactly one edge labeled k for $0 \leq k \leq d$. Figure 4.2 shows G_M for the subdivision given in Figure 4.1.

If $w = w_1 \cdots w_\ell \in \{0, \ldots, d\}^*$, define

$$switch_w(t) = \begin{cases} switch_{w_\ell}(switch_{w_{\ell-1}}(\cdots(switch_{w_2}(switch_{w_1}(t)))\cdots)) & \text{if } w \neq \lambda, \\ t & \text{if } w = \lambda. \end{cases}$$

If $J \subseteq \{0, \ldots, d\}^*$, let $switch_J(t) = \{switch_w(t) \mid w \in J\}$. Let $I \subseteq \{0, \ldots, d\}$. Then $switch_{I^*}(t)$ will be called the I-**orbit**[1] of t. This may be thought of as the set of all cell-tuples obtainable from t by repeated applications of $switch_k$ operations, where the

[1]The term orbit is taken from the similar concept in algebra.

k's are in I. The set of I-orbits for all $t \in T_M$ partitions T_M into a set of equivalence classes.

In terms of the graphical interpretation of the cell-tuple structure the I-orbit may be understood as the set of all cell-tuples in G_M reachable by following paths whose edges are labeled by elements of I. Another way of thinking of this is by considering the subgraph of G_M consisting of all edges whose labels are in I. The I-orbit of t is just the connected component of this subgraph containing t. The connected components of this subgraph are exactly the set of I-orbits of T_M, hence are equal to the equivalence classes generated by the I-orbits.

If c is a k-cell of C, the set of cell-tuples having c as a component, $assoc(c) = \{t \in T_M \mid t_k = c\}$, will be called the set of **associated cell-tuples of** c. If $c_{\alpha_{i_0}} < \cdots < c_{\alpha_{i_\ell}}$, such that $\dim(c_{\alpha_{i_j}}) = i_j$ for $j \in \{0, \ldots, \ell\}$ the set of cell-tuples $assoc(c_{\alpha_{i_0}}, \ldots, c_{\alpha_{i_\ell}}) = \{t \in T_M \mid t_{i_j} = c_{\alpha_{i_j}}, 0 \leq j \leq \ell\}$ will be called the set of **associated cell-tuples** of $c_{\alpha_0}, \ldots, c_{\alpha_\ell}$. This gives a correspondence between cells in C and subsets of cell-tuples. For $0 \leq k \leq d$, the associated sets of the k-cells in C forms a partition of T_M. This will be developed to give a fundamental relationship between cells in C and orbits in T_M, when Lemma 4.11 and Corollary 4.14 are proved in Section 4.4.

Two cell-tuple structures are **equivalent**, $T_M \simeq T_N$, if there is a bijection $\jmath : T_M \to T_N$ which preserves the *switch* operation, by which we mean $switch_k(\jmath(t)) = \jmath(switch_k(t))$ for all $t \in T_M$, $0 \leq k \leq d$.

The following observation will be used in the proof of Theorem 4.4. If $\jmath : T_M \to T_N$ gives an equivalence, and $J \subseteq \{0, \ldots, d\}^*$, then $switch_J(\jmath(t)) = \jmath(switch_J(t))$ for all $t \in T_M$. This can be seen by the following argument. If $w \in J$, then $switch_w(\jmath(t)) = \jmath(switch_w(t))$ by repeated applications of the definition of equivalence. Thus $t' \in switch_J(\jmath(t)) \Leftrightarrow t' = switch_w(\jmath(t))$ for some $w \in J \Leftrightarrow t' = \jmath(switch_w(t))$ for some $w \in J \Leftrightarrow t' \in \jmath(switch_J(t))$.

A similar result to that of Lemma 4.2 can be phrased without reference to the cell-tuple structure. Given an ascending chain of three cells, where the difference in dimension

between each pair of consecutive cells is one, there is a unique cell which is incident to both the first and third cells (taken in ascending order) but is different from the middle cell. This will be proved in Section 4.4.

Corollary 4.3 *If (M,C) is a subdivided d-manifold, $c_{\alpha_{k-1}} \prec c_{\alpha_k} \prec c_{\alpha_{k+1}}$, where $0 \leq k \leq d$, $c_{\alpha_k} \in C$ and $\dim(c_{\alpha_k}) = k$, then there is a unique $c_{\alpha'_k} \neq c_{\alpha_k}$ such that $c_{\alpha_{k-1}} \prec c_{\alpha'_k} \prec c_{\alpha_{k+1}}$.*

It will also be useful to define a *switch* operator on triples, as in Corollary 4.3. Let $S = \{(c_{\alpha_{k-1}}, c_{\alpha_k}, c_{\alpha_{k+1}}) \mid c_{\alpha_{k-1}} \prec c_{\alpha_k} \prec c_{\alpha_{k+1}}, 0 \leq k \leq d, \text{ and } c_{\alpha_{k-1}}, c_{\alpha_k}, c_{\alpha_{k+1}} \in C\}$. Again abusing notation, we define *switch* : $S \rightarrow C$ by $switch(c_{\alpha_{k-1}}, c_{\alpha_k}, c_{\alpha_{k+1}}) = c'$, where c' is $c_{\alpha'_k}$ given by Corollary 4.3.

Now we will state the two main theorems about representations of subdivided manifolds. Their proofs will be given in Sections 4.5 and 4.6.

One of the main objectives when defining a representation of geometric objects is to represent structure. Theorem 4.4 will show that the incidence graph and the cell-tuple structure are each powerful enough to represent subdivided manifolds up to equivalence.

Theorem 4.4 *If (M,C) and (N,D) are subdivided d-manifolds, then the following are equivalent:*

(1) $(M,C) \simeq (N,D)$,

(2) $\mathcal{I}_M \simeq \mathcal{I}_N$,

(3) $\mathcal{T}_M \simeq \mathcal{T}_N$.

Next we will state a theorem about ordering of cells in subdivided manifolds. Recall the examples of ordering given in Section 1.3. Figure 1.2 gave examples in two dimensions: case (a) showed an ordering of the edges and faces incident to a vertex, and case (b) showed an ordering of the vertices and edges incident to a face. Figure 1.3 gave examples in three dimensions: case (a) showed an ordering of the 2- and 3-cells incident

to a 1-cell, case (b) showed an ordering of the 1- and 2-cells incident to both a 0-cell and a 3-cell, and case (c) showed an ordering of the 0- and 1-cells incident to a 2-cell. Observe that in each of these five cases, we are ordering the set of all (k–1)- and k-cells containing a (k–2)-cell, or within a (k+1)-cell, or both.

All of these cases can be put into one framework, by thinking of fixing both a (k–2)-cell and a (k+1)-cell containing it. This will work for $1 \leq k \leq d$. In the case of $k = 1$ the (k–2)-cell is not needed, but for simplicity of notation we specify it to be c_{-1}. In the case of $k = d$ the (k+1)-cell is not needed, but for simplicity of notation we specify it to be c_{d+1}.

If $c_{\alpha_{k-2}} < c_{\alpha_{k+1}}$, where $1 \leq k \leq d$, $c_{\alpha_{k-2}}$ is a (k–2)-cell and $c_{\alpha_{k+1}}$ is a (k+1)-cell, let

$$S(c_{\alpha_{k-2}}, c_{\alpha_{k+1}}) = \{c \mid c_{\alpha_{k-2}} < c < c_{\alpha_{k+1}}\}.$$

When it is clear from the context, we will write S for $S(c_{\alpha_{k-2}}, c_{\alpha_{k+1}})$. It is this set S which can be ordered. The idea is that given a (k–2)-cell contained in the boundary of a (k+1)-cell, all of the cells "between" them may be put into a circular order 'around' the (k–1)-cell, such that this order alternates between (k–1)-cells and k-cells, consecutive pairs of k-cells share the (k–1)-cell between them, and consecutive pairs of (k–1)-cells share the k-cell between them. Furthermore, this order can be produced by choosing an appropriate cell-tuple, and applying a sequence of *switch*'s.

The reason this idea is not more obvious when trying to generalize from two and three dimensions is that of the five cases, namely $d = 2, k = 1, 2$ and $d = 3, k = 1, 2, 3$, only one involves fixing two cells.

Let $m = \#(S)$. A **circular ordering of** S is an ordering $c_{\eta_0}, \ldots, c_{\eta_{m-1}}$ of the elements of S such that:

(1) c_{η_i} is a (k–1)-cell if i is even, and is a k-cell if i is odd,

(2) $c_{\eta_{i-1 \bmod m}}$ and $c_{\eta_{i+1 \bmod m}}$ share c_{η_i}, for $0 \leq i \leq m-1$.

(by *share* we mean 'are each incident to'.)

As proven in Theorem 4.4, the incidence graph is powerful enough to represent the

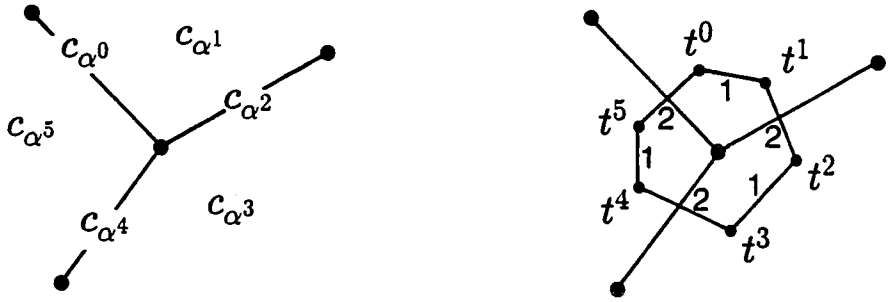


Figure 4.3: Example of Theorem 4.5

topological structure of subdivided manifolds. But the incidence graph doesn't contain ordering information in a readily accessible form. In fact we still don't know what kind of ordering is contained in subdivided manifolds. Theorem 4.5 shows the existence of ordering information for all dimensions from 0 to d, and shows that the cell-tuple structure gives immediate access to it. An example is given in Figure 4.3.

Theorem 4.5 *There exists a circular ordering of $S(c_{\alpha_{k-2}}, c_{\alpha_{k+1}})$. Furthermore, the switch structure directly represents this ordering information:*

A. *There exists a $t^0 \in T_M$ such that $t^0_{k-2} = c_{\alpha_{k-2}}$ and $t^0_{k+1} = c_{\alpha_{k+1}}$. For any such t^0, if the sequence of cell-tuples $t^0, \ldots, t^{m-1}$ is defined by*

$$t^i = \begin{cases} switch_k(t^{i-1}) & i \text{ even}, \\ switch_{k-1}(t^{i-1}) & i \text{ odd}. \end{cases}$$

then the sequence $c_{\eta_0}, \ldots, c_{\eta_{m-1}}$ defined by

$$c_{\eta_i} = \begin{cases} t^i_{k-1} & i \text{ even}, 2 \le i \le m \\ t^i_k & i \text{ odd}, 1 \le i \le m-1 \end{cases}$$

gives a circular ordering of $S(c_{\alpha_{k-2}}, c_{\alpha_{k+1}})$.

B. *There exist $c_{\eta_0}, c_{\eta_1} \in C$ such that $c_{\alpha_{k-2}} \prec c_{\eta_0} \prec c_{\eta_1} \prec c_{\alpha_{k+1}}$. For any such c_{η_0} and c_{η_1}, if the sequence $c_{\eta_0}, \ldots, c_{\eta_{m-1}}$ is defined by*

$$c_{\eta_i} = \begin{cases} switch_{k-1}(c_{\alpha_{k-2}}, c_{\eta_{i-2}}, c_{\eta_{i-1}}) & i \text{ even}, 2 \le i \le m \\ switch_k(c_{\eta_{i-1}}, c_{\eta_{i-2}}, c_{\alpha_{k+1}}) & i \text{ odd}, 1 \le i \le m-1 \end{cases}$$

gives a circular ordering of $S(c_{\alpha_{k-2}}, c_{\alpha_{k+1}})$.

This theorem shows that the circular ordering is always present, and that ordering information is directly accessible via the *switch* operator.

4.3 The Generalized Barycentric Subdivision

In this section we will define and prove some properties of a generalization of the barycentric subdivision of a complex. This is a crucial object in the proofs of the theorems in this dissertation, giving a link between the objects being modeled and their combinatorial representation. An understanding of the generalized barycentric subdivision gives the right vantage point for understanding the mathematics underlying this work, and seeing the underlying similarities among all of the implicit-cell representations.

The generalized barycentric subdivision of a subdivided manifold (M, C) is a triangulation of the manifold M, and is also a refinement of the subdivided manifold. (Recall from Section 2.5 that a triangulation of a manifold is a homeomorphism between a simplicial complex and the manifold, and from Section 2.4 that a refinement of (M, C) is a subdivided manifold (M, D) in which every cell of D is contained in a cell of C.) Having a triangulation of the manifold allows the application of theorems about triangulated manifolds from algebraic topology. The particular refinement given by the generalized barycentric subdivision allows the cells of C to be written as unions of cells of the generalized barycentric subdivision in a convenient way.

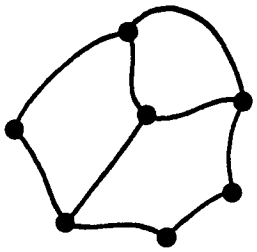
To define the generalized barycentric subdivision, we will first define an abstract simplicial complex $\mathcal{A}_M$ from the incidence relation among the cells of C. A geometric realization of this abstract simplicial complex will then give a simplicial complex K_M. A homeomorphism between this simplicial complex and the manifold M will give the required triangulation. The images in M of the simplices of K_M under the homeomorphism will be the cells in the generalized barycentric subdivision.

We will give an intuitive description of a construction of the generalized barycentric subdivision now. This should give an idea of what the generalized barycentric subdivision is, and should provide the intuition for the construction given in the proof of Lemma 4.6.

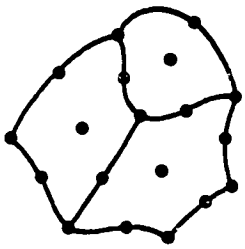
For every cell $c_\alpha \in C$ create a new vertex $c_{sd}(\alpha)$ in the interior of of c_α. (We will assume that the interior of a vertex is equal to the vertex, so that vertices of C have new vertices created 'in their interiors'.) For every pair of cells $c_{\alpha_{k_1}} < c_{\alpha_{k_2}}$ in C, add a new 1-cell in the interior of $c_{\alpha_{k_2}}$ having as endpoints the new 0-cells $c_{sd}(\alpha_{k_1})$ and $c_{sd}(\alpha_{k_2})$. This gives the set of 1-cells of the generalized barycentric subdivision. For every ascending chain $c_{\alpha_{k_1}} < c_{\alpha_{k_2}} < c_{\alpha_{k_3}}$ of three cells, add a 2-cell in the interior of $c_{\alpha_{k_3}}$ which has as its boundary the new 0-cells $c_{sd}(\alpha_{k_1})$, $c_{sd}(\alpha_{k_2})$, and $c_{sd}(\alpha_{k_3})$ and the new 1-cells $c_{sd}(\alpha_{i_1}, \alpha_{i_2})$, $c_{sd}(\alpha_{i_1}, \alpha_{i_3})$, and $c_{sd}(\alpha_{i_2}, \alpha_{i_3})$. Continue 'filling in' cells of increasing dimension in this fashion, until new d-cells are added for all maximal ascending chains of cells in C.

Recall the definitions of abstract simplicial complexes and their geometric realizations from Section 2.5. If (M, C) is a subdivided d-manifold or a subdivided d-manifold-with-boundary, where $C = \{c_\alpha\}_{\alpha \in I_C}$, define the abstract simplicial complex $\mathcal{A}_M = \{\{\alpha_{i_0}, \ldots, \alpha_{i_k}\} \mid \alpha_{i_0} < \cdots < \alpha_{i_k},\ \alpha_{i_j} \in I_C\}$, Let K_M be any geometric realization of $\mathcal{A}_M$. This always exists by [Mun84] Theorem 3.1(a). It may be assumed that K_M is embedded in $\mathbb{R}^n$, for some n (see the proof of [Mun84] Theorem 3.1(a)). K_M will essentially be the generalized barycentric subdivision.

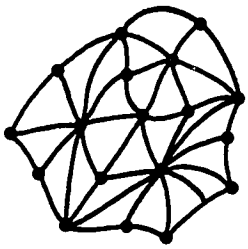
For every cell $c_\alpha \in C$, there is a vertex a in $\mathcal{A}_M$. Let v_α be the vertex in K_M corresponding to a, and define $label(v_\alpha) = \dim(\alpha)$. Denote the simplex defined by $v_{\alpha_{i_1}}, \ldots, v_{\alpha_{i_\ell}}$ by $\sigma(\alpha_{i_1}, \ldots, \alpha_{i_\ell})$. There is a one-to-one correspondence between cell-tuples in T_M and d-simplices in K_M. For $0 \leq k \leq d$, define the operation $switch_k$ mapping the set of d-simplices of K_M onto itself: if $\sigma = \sigma(\alpha_0, \ldots, \alpha_d)$ let $switch_k(\sigma) = \sigma(\alpha_0, \ldots, \alpha_{k-1}, \alpha'_k, \alpha_{k+1}, \ldots, \alpha_d)$, where α'_k is the unique index such that $\alpha'_k \neq \alpha_k$ and $\alpha_{k-1} < \alpha'_k < \alpha_{k+1}$. This corresponds exactly to taking $switch_k(t)$, where $t =$



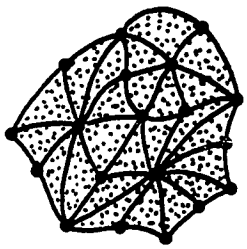
(M, C)



Create 0-cells of (M, C_{sd})



Add 1-cells of (M, C_{sd})



Add 2-cells of (M, C_{sd})

Figure 4.4: Example of construction of the generalized barycentric subdivision

$(c_{\alpha_0}, \ldots, c_{\alpha_d})$.

$\mathcal{A}_M$ and $\mathcal{A}_N$ are **equivalent**, $\mathcal{A}_M \simeq \mathcal{A}_N$, if there exists a bijection $\imath : \mathcal{A}_M \to \mathcal{A}_N$ which preserves dim and simplices. K_M and K_N are **equivalent**, $K_M \simeq K_N$, if there exists a linear simplicial map $f : K_M \to K_N$ which is bijective and preserves *label*.

The following lemma makes a connection between K_M and (M, C) itself, by giving a homeomorphism ψ_M between them. This gives a triangulation of M, which taken as a subdivision of M is a refinement of (M, C) (recall from Section 2.4 that this means that every cell of the new subdivision is contained in a cell of the original subdivision). This allows the cells of C to be described in a very simple way. The refinement defined by ψ_M is the generalized barycentric subdivision.

If $\psi_M : |K_M| \to M$ is a homeomorphism, define $c_{sd}(\alpha_{i_0}, \ldots, \alpha_{i_k}) = \psi_M(\text{Int } \sigma(v_{\alpha_{i_0}}, \ldots, v_{\alpha_{i_k}}))$; thus $c_{sd}(\alpha_{i_0}, \ldots, \alpha_{i_k})$ is the image under ψ_M of the interior of the simplex $\sigma(v_{\alpha_{i_0}}, \ldots, v_{\alpha_{i_k}})$. These $c_{sd}(\alpha_{i_0}, \ldots, \alpha_{i_k})$'s are the cells of the generalized barycentric subdivision.

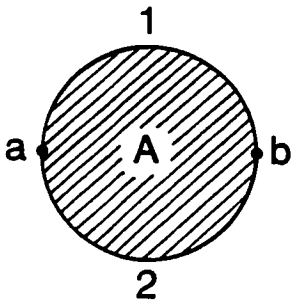
Lemma 4.6 *If (M, C) is a subdivided d-manifold or a subdivided d-manifold-with-boundary, then there exists $\psi_M : |K_M| \to M$ such that*

(a) *ψ_M is a homeomorphism,*

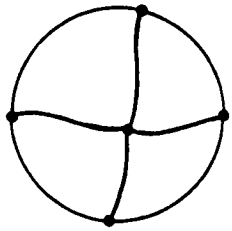
(b) $$c_\alpha = \bigcup_{\alpha_0 < \cdots < \alpha_\ell < \alpha} c_{sd}(\alpha_0, \ldots, \alpha_\ell, \alpha).$$

In the statement of this lemma, and for the remainder of this chapter, the following conventions will hold: the set of sequences over which the union is taken includes the sequence consisting only of α; and we say that the interior of a vertex is equal to the vertex, i.e. Int $\sigma(v) = \sigma(v)$.

Let $C_{sd} = \{c_{sd}(\alpha_{i_0}, \ldots, \alpha_{i_k}) \mid \sigma(\alpha_{i_0}, \ldots, \alpha_{i_k}) \in K_M\}$. (M, C_{sd}) is a subdivided d-manifold-with-boundary itself, which will be called the **generalized barycentric subdivision** of (M, C). (See Figures 4.5 and 4.6.) This definition of the generalized barycentric subdivision is the same as that in [LW 69], but is different from that given in [Mun84].



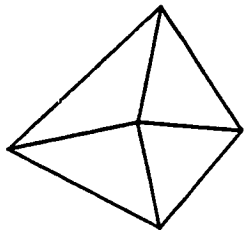
(M, C)



(M, C_{sd})

$\{\ a,\ b,\ 1,\ 2,\ A$
$a1,\ a2,\ b1,\ b2,$
$aA,\ bA,\ 1A,\ 2A,$
$a1A,\ a2A,\ b1A,\ b2A\ \}$

$\mathcal{A}_M$



$|K_M|$

Figure 4.5: Example of $\mathcal{A}_M$, K_M, and the generalized barycentric subdivision

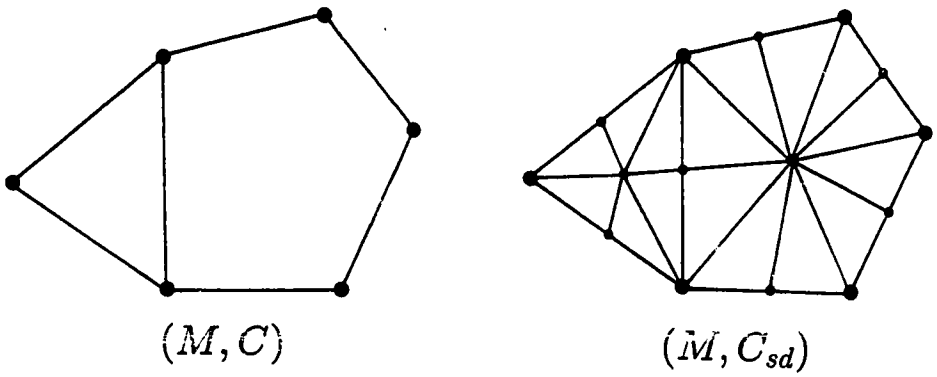


Figure 4.6: Another example of the generalized barycentric subdivision

The generalized barycentric subdivision gives the crucial link between the combinatorial representation of subdivided manifolds and the subdivided manifolds themselves. Theorems about triangulated manifolds from algebraic topology may be used in proofs of the main results, and may be used when extending those results to manifolds-with-boundary. Writing cells in C as unions of cells in C_{sd} is useful in proofs, and allows the definition of the dual in a straightforward manner in Section 4.7.

One possible construction of a ψ_M satisfying the properties of Lemma 4.6 is given in the proof sketch of [LW69] Theorem III.1.7. When taken with [LW69] Theorem III.2.1, this gives the result. We will describe a method for constructing the homeomorphism ψ_M, using the notation of this dissertation, in the following proof. It is basically the same as that in [LW69].

Proof of Lemma 4.6 If (M, C) is a subdivided manifold or a subdivided manifold-with-boundary, let $C_k = \{c \in C \mid \dim(c) = k\}$. The **$k$-skeleton of M**, $C^k = \bigcup_{i=0}^{k} C_k$, is the set of all of the cells in C whose dimension is less than or equal to k. $M^k = |C^k|$ is the subspace of M which equals the union of all cells of C whose dimension is less than or equal to k.

The construction of ψ_M is inductive, on dimension k. An intuitive description of this construction was given earlier in this section. Let $K_M^k = \{\sigma \in K_M \mid \dim(\sigma) \leq k\}$ have the subspace topology from $\mathbb{R}^n$. For each $0 \leq k \leq d$, a homeomorphism $\psi_M^k : |K_M^k| \to M^k$ which satisfies parts (a) and (b) of Lemma 4.6 will be constructed.

For $k = 0$, let $\psi_M^0(v_\alpha) = c_\alpha$. Both $|K_M^0|$ and M^0 have the discrete topology, so ψ_M^0 is a homeomorphism. If $c_\alpha \in C_0$ then $c_\alpha = \psi_M^0(v_\alpha) = c_{sd}(\alpha)$ gives part (b).

For $k > 0$, the assumption is that $\psi_M^{k-1} : |K^{k-1}| \to M^{k-1}$ satisfies parts (a) and (b).

Recall from Section 2.3 that $\dot{c}_\alpha = \overline{c}_\alpha - c_\alpha$ is the boundary of the cell c_α. Also recall the definitions of star, closed star and link from Section 2.5 — if σ is a simplex in a simplicial complex K, then $\overline{St}\,\sigma$ is the set of simplices in K which have σ as a face, $St\,\sigma$ is the set of the interiors of simplices in K which have σ as a face, and $Lk\,\sigma$ is the set of all faces of simplices in K which have σ as a face. The intuition behind the use of the star, closed star, and link in what follows is that if c_α is a k-cell in C, then $|St\,v_\alpha|$ corresponds to c_α, $|\overline{St}\,v_\alpha|$ corresponds to $\overline{c}_\alpha$, and $|Lk\,v_\alpha|$ corresponds to $\dot{c}_\alpha$.

Let $Lk^k\,v_\alpha$ be the link of v_α in K_M^k, and let $\overline{St}^k\,v_\alpha$ be the closed star of v_α in K_M^k.

For each α such that $c_\alpha \in C_k$, restricting ψ_M^{k-1} to $|Lk^k\,v_\alpha|$ gives a homeomorphism mapping $|Lk^k\,v_\alpha|$ onto $\dot{c}_\alpha$. This follows from [Mun75] Chapter 2, Theorem 7.2, once it is shown that $\psi_M(|Lk^k\,v_\alpha|) = \dot{c}_\alpha$:

$$\dot{c}_\alpha = \bigcup_{c_{\alpha'} \subset \dot{c}_\alpha} c_{\alpha'} = \bigcup_{\alpha' < \alpha} \Big(\bigcup_{\alpha_0 < \cdots < \alpha_\ell < \alpha'} c_{sd}(\alpha_0, \ldots, \alpha_\ell, \alpha') \Big)$$

$$= \bigcup_{\alpha_0 < \cdots < \alpha_\ell < \alpha' < \alpha} c_{sd}(\alpha_0, \ldots, \alpha_\ell, \alpha') = \bigcup_{\alpha_0 < \cdots < \alpha_\ell < \alpha' < \alpha} \psi_M(\operatorname{Int} \sigma(\alpha_0, \ldots, \alpha_\ell, \alpha'))$$

$$= \psi_M\Big(\bigcup_{\alpha_0 < \cdots < \alpha_\ell < \alpha' < \alpha} \operatorname{Int} \sigma(\alpha_0, \ldots, \alpha_\ell, \alpha') \Big) = \psi_M\Big(\bigcup_{\sigma \in Lk^k\,v_\alpha} \operatorname{Int} \sigma \Big) = \psi_M\Big(\bigcup_{\sigma \in Lk^k\,v_\alpha} \sigma \Big)$$

$$= \psi_M(|Lk^k\,v_\alpha|)$$

Composing ψ_M restricted to $|Lk^k\,v_\alpha|$ with f_α^{-1} restricted to $\dot{c}_\alpha$, gives a homeomorphism $h : |Lk^k\,v_\alpha| \to \operatorname{Bd} \overline{\mathbb{B}}^k$ defined by $h = f_\alpha^{-1}|_{\dot{c}_\alpha} \circ \psi_M|_{|Lk^k\,v_\alpha|}$.

Next, h can be extended to a homeomorphism $h' : |\overline{St}^k v_\alpha| \to \overline{\mathbb{B}}^k$ as follows:

Every $x \in |\overline{St}^k v_\alpha|$, except $x = v_\alpha$, can be written uniquely as $(1-\gamma)v_\alpha + \gamma y$, where $0 \leq \gamma \leq 1$ and $y \in Lk^k v_\alpha$. Define

$$h'(x) = \begin{cases} \gamma h(y) & x \neq v_\alpha, \\ 0 & x = v_\alpha. \end{cases}$$

For $x \in |\overline{St}^k v_\alpha|$, define $\psi_M^k(x) = f_\alpha(h'(x))$. This is a composition of homeomorphisms, hence is a homeomorphism from $|\overline{St}^k v_\alpha|$ to $\overline{c}_\alpha$.

Now, ψ_M^k as defined for each α agrees on $|K_M^k|$, because each is an extension of ψ_M^{k-1}, and $\psi_M^k(x)$ is defined for all $x \in K_M^k$, so $\psi_M^k : |K_M^k| \to M^k$ is a homeomorphism (by [Mun84] lemma 38.1). Finally, let $\psi_M = \psi_M^d$, to get part (a).

To show part (b), if $c_\alpha \in C_k$:

$$c_\alpha = \psi_M^k(St^k v_\alpha) = \psi_M^k\Big(\bigcup_{\alpha_{i_0} < \cdots < \alpha_{i_\ell} < \alpha} \text{Int } \sigma(\alpha_{i_0}, \ldots, \alpha_{i_\ell}, \alpha)\Big)$$

$$= \bigcup_{\alpha_{i_0} < \cdots < \alpha_{i_\ell} < \alpha} \psi_M^k(\text{Int } \sigma(\alpha_{i_0}, \ldots, \alpha_{i_\ell}, \alpha))$$

$$= \bigcup_{\alpha_{i_0} < \cdots < \alpha_{i_\ell} < \alpha} c_{sd}(\alpha_{i_0}, \ldots, \alpha_{i_\ell}, \alpha)$$

∎

Because the characteristic maps are not part of the definition of subdivided d-manifolds, the barycentric subdivision is not unique, but is unique up to equivalence.

We will state three facts about K_M, for manifolds and manifolds-with-boundary, which will be used at various times in the remainder of this chapter. The first says that in a triangulated manifold or manifold-with-boundary, every simplex is contained in a d-simplex. This follows from [Mun84] Exercise 35.4.

Fact 4.7 *If $\sigma \in K_M$, then:*

(a) *σ is a d-simplex, or*

(b) *σ is contained in a d-simplex.*

Since K_M is homeomorphic to M, and is trivially triangulated, it is a triangulated manifold. Recalling the definition of the boundary of a manifold from Section 2.2, $\partial|K_M|$ is the boundary of $|K_M|$, taken as a manifold-with-boundary. The following fact says that a (d–1)-simplex in K_M is in the boundary of K_M if and only if it is incident to exactly one d-simplex (otherwise it is incident to exactly two d-simplices). This fact follows from [Mun84] Exercise 35.4.

Fact 4.8 *If $\sigma \in K_M$ is a (d–1)-simplex, then:*

(a) *$\sigma \not\subseteq \partial|K_M| \Rightarrow \sigma$ is contained in exactly two d-simplices,*

(b) *$\sigma \subseteq \partial|K_M| \Rightarrow \sigma$ is contained in exactly one d-simplex.*

The third fact is just the statement of [Mun84] Theorem 35.3.

Fact 4.9 *Let M be a d-manifold with boundary; suppose $h : |K| \rightarrow M$ is a triangulation of M. Then $h^{-1}(\partial M)$ is the polytope (underlying space) of a subcomplex of M.*

From this fact it follows that $\partial|K_M|$ is triangulated by a subset of the simplices of K_M.

4.4 A Few Lemmas

The lemmas given in this section will be needed for the proofs of Theorem 4.4 and Theorem 4.5. Taken in and of themselves, several of the lemmas should provide insight into the relation between the cell-tuple structure and subdivided manifolds (i.e. Lemma 4.11, Corollary 4.12, Corollary 4.14, and Lemma 4.15).

We will first prove a useful lemma which shows that for any ascending chain of cells there is a cell-tuple which has all of the cells in that chain as components. This is equivalent to showing that any ascending chain of cells may be extended to a maximal ascending chain.

Lemma 4.10 *If (M,C) is a subdivided d-manifold or a subdivided d-manifold-with-boundary and $c_{\alpha_{i_0}} < \cdots < c_{\alpha_{i_\ell}}$, where $\ell \geq 0$, $c_{\alpha_{i_j}} \in C$ and $\dim(c_{\alpha_{i_j}}) = i_j$, then there exists $t \in T_M$ such that $t_{i_j} = c_{\alpha_{i_j}}$, for $j = 0, \ldots, \ell$.*

Proof If $\ell = d$, let $t = (c_{\alpha_{i_0}}, \ldots, c_{\alpha_{i_\ell}})$. If $\ell < d$, let $\sigma = \sigma(\alpha_{i_0}, \ldots, \alpha_{i_\ell})$. By Fact 4.7 there exists a d-simplex $\sigma(\alpha_0, \ldots, \alpha_d) \in K_M$ containing σ. Let $t = (c_{\alpha_0}, \ldots, c_{\alpha_d})$. ∎

For the remainder of this section M will be a d-manifold (without boundary).

The barycentric subdivision allows us to prove Lemma 4.2 (stated in Section 4.2), which is necessary for the definition of *switch*.

Lemma 4.2 *If (M,C) is a subdivided d-manifold, $t \in T_M$ and $0 \leq k \leq d$, then there is a unique $t' \in T_M$ such that $t'_k \neq t_k$ and $t'_i = t_i$ for all $i \neq k$.*

Proof Let $t \in T_M$ and $0 \leq k \leq d$. If $t = (c_{\alpha_0}, \ldots, c_{\alpha_{k-1}}, c_{\alpha_k}, c_{\alpha_{k+1}}, \ldots, c_{\alpha_d})$, let $\sigma = \sigma(\alpha_0, \ldots, \alpha_{k-1}, \alpha_k, \alpha_{k+1}, \ldots \alpha_d)$, and let $\sigma' = \sigma(\alpha_0, \ldots, \alpha_{k-1}, \alpha_{k+1}, \ldots \alpha_d)$. By Fact 4.8(a), there is a d-simplex $\sigma'' \neq \sigma$ which contains σ' (remember that we are dealing with manifolds without boundary). There must exist a k-cell $c_{\alpha'_k}$ such that $\sigma'' = \sigma(\alpha_0, \ldots, \alpha_{k-1}, \alpha'_k, \alpha_{k+1}, \ldots \alpha_d)$. Let $t' = (c_{\alpha_0}, \ldots, c_{\alpha_{k-1}}, c_{\alpha'_k}, c_{\alpha_{k+1}}, \ldots, c_{\alpha_d})$. ∎

It will be useful to define *switch* on the d-simplices of K_M: given d-simplex $\sigma = \sigma(\alpha_0, \ldots, \alpha_{k-1}, \alpha_k, \alpha_{k+1}, \ldots \alpha_d), 0 \leq k \leq d$, define $switch_k(\sigma)$ to be the unique d-simplex $\sigma(\alpha_0, \ldots, \alpha_{k-1}, \alpha'_k, \alpha_{k+1}, \ldots \alpha_d)$ such that $\alpha'_k \neq \alpha_k$.

Now we are ready to prove Corollary 4.3 (stated in Section 4.2).

Corollary 4.3 *If (M,C) is a subdivided d-manifold, $c_{\alpha_{k-1}} \prec c_{\alpha_k} \prec c_{\alpha_{k+1}}$, where $0 \leq k \leq d$, $c_{\alpha_k} \in C$ and $\dim(c_{\alpha_k}) = k$, then there is a unique $c_{\alpha'_k} \neq c_{\alpha_k}$ such that $c_{\alpha_{k-1}} \prec c_{\alpha'_k} \prec c_{\alpha_{k+1}}$.*

Proof By Lemma 4.10, there is a $t \in T_M$ such that $t_i = c_{\alpha_i}$, for $i \in \{0, \ldots, d\}$. By Lemma 4.2 there exists a unique t' such that $t'_i = t_i \Leftrightarrow i \neq k$. Let $c_{\alpha'_k} = t'_k$. This satisfies the statement of the lemma. We must show that $c_{\alpha'_k}$ is unique. Suppose there exists $c_{\alpha''_k}$ which is not equal to c_{α_k} or $c_{\alpha'_k}$, such that $c_{\alpha_{k-1}} \prec c_{\alpha''_k} \prec c_{\alpha_{k+1}}$. Then the cell-tuple $(t_0, \ldots, t_{k-1}, c_{\alpha''_k}, t_{k+1}, \ldots, t_d)$ is not equal to t or t', contradicting Lemma 4.2. ∎

Next we give a lemma which describes a basic relationship between the cell-tuple structure and the subdivided manifold. Recall from Section 4.2 the definition of the I-orbit, the set of cell-tuples 'reachable' from a given cell-tuple by applying sequences of $switch_k$'s, where the k's are taken from I. If I is a subset of $\{0, \ldots, d\}$, let $\hat{I} = \{0, \ldots, d\} - I$. Thus if $I = \{i_0, \ldots, i_\ell\}$, then $\hat{I}\text{-}orbit(t) = switch_{\{\{0,\ldots,d\}-\{i_0,\ldots,i_\ell\}\}^*}(t)$. If $c_{\alpha_{i_0}} < \cdots < c_{\alpha_{i_\ell}}$ is an ascending chain of cells, recall from Section 4.2 that the associated set of cells in this chain, $assoc(c_{\alpha_{i_0}}, \ldots, c_{\alpha_{i_\ell}})$, is the set of cell-tuples which contain these cells as components. If $I = \{i_0, \ldots, i_\ell\}$ is the set of dimensions of these cells, and t is any cell-tuple in the associated set, then the lemma shows that the associated set is equal to the set of cell-tuples which may be obtained from t by repeated applications of $switch_k$, where $k \notin I$. In graphical terms, consider the subgraph of G_M consisting of edges whose labels are not in I. Then the associated set is equal to the connected component of this subgraph which contains t. Figure 4.7 gives several examples.

Lemma 4.11 *If $c_{\alpha_{i_0}} < \cdots < c_{\alpha_{i_\ell}}$ and $t \in T_M$ such that $t_{i_j} = c_{\alpha_{i_j}}$ for $0 \leq j \leq \ell$, then $assoc(c_{\alpha_{i_0}}, \ldots, c_{\alpha_{i_\ell}}) = switch_{\hat{I}^*}(t)$, where $I = \{i_0, \ldots, i_\ell\}$.*

To show $assoc(c_{\alpha_0}, \ldots, c_{\alpha_\ell}) \subseteq switch_{\hat{I}^*}(t)$ requires showing that the subgraph of G_M induced by $assoc(c_{\alpha_0}, \ldots, c_{\alpha_\ell})$ is connected. This is the only difficult part of the proof. The proof uses the star and link of a simplex defined in Section 2.5.

The proof uses the fact that there is a one-to-one correspodence between the set of ascending chains of length ℓ in C and the ℓ-simplices of K_M, and the similar fact that there is a one-to-one correspondence between cell-tuples in T_M and d-simplices in K_M. These facts both follow from the definition of K_M. The ascending chain $c_{\alpha_{i_0}} < \cdots < c_{\alpha_{i_\ell}}$

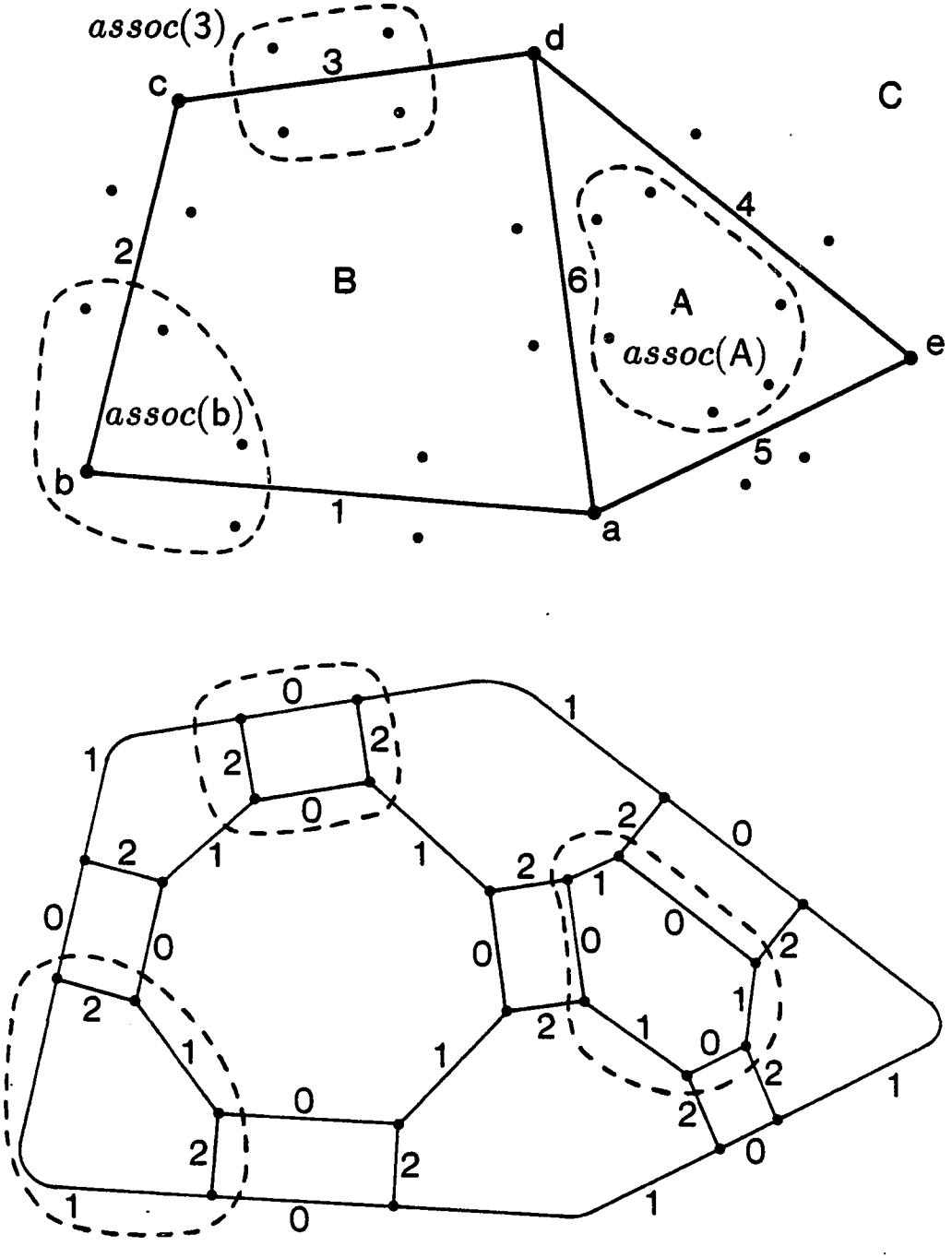


Figure 4.7: Examples of Lemma 4.11

corresponds to $\sigma(\alpha_{i_0},\ldots,\alpha_{i_\ell})$, and the cell-tuples in $assoc(c_{\alpha_0},\ldots,c_{\alpha_\ell})$ correspond to the d-simplices in $\overline{St}\,\sigma(\alpha_{i_0},\ldots,\alpha_{i_\ell})$. Then the problem of showing that the subgraph of G_M induced by $assoc(c_{\alpha_0},\ldots,c_{\alpha_\ell})$ is connected reduces to proving that there is a 'path' of d-simplices in $\overline{St}\,\sigma(\alpha_{i_0},\ldots,\alpha_{i_\ell})$ between any two d-simplices in $\overline{St}\,\sigma(\alpha_{i_0},\ldots,\alpha_{i_\ell})$. This will be proved using the following corollary.

> [Mun84] Corollary 70.3 Let (X,A) be a compact triangulated relative homology d-manifold, with $X-A$ connected. If σ and σ' are two d-simplices of X not in A, there is a sequence
>
> $$\sigma=\sigma_0,\sigma_1,\ldots,\sigma_{m-1}=\sigma'$$
>
> of d-simplices not in A, such that $\sigma_i\cap\sigma_{i+1}$ is a $(d-1)$-simplex not in A, for each i.

Proof To see that $switch_{\hat{I}^*}(t)\subseteq assoc(c_{\alpha_0},\ldots,c_{\alpha_\ell})$ is easy: If $k\in\hat{I}$, then $switch_k$ doesn't change t_j for any $j\in I$. So if $t'=switch_w(t)$, for any $w\in\hat{I}^*$, then $t'_j=t_j$, for all $j\in I$, i.e. $t'\in assoc(t_{i_0},\ldots,t_{i_\ell})$.

What must now be shown is that $assoc(c_{\alpha_0},\ldots,c_{\alpha_\ell})\subseteq switch_{\hat{I}^*}(t)$.

First we will prove the following claim:

> If $\tilde{\sigma}$ is a k-simplex in K_M, $0\le k\le d$, and if σ,σ' are d-simplices in $\overline{St}\,\tilde{\sigma}$, then there is a sequence $\sigma=\sigma_0,\ldots,\sigma_m=\sigma'$ of d-simplices of K_M, contained in $\overline{St}\,\tilde{\sigma}$, such that $\sigma_j\cap\sigma_{j+1}$ is a $(d-1)$-simplex for $0\le j\le m-1$.

To do so, we will make use of [Mun84] Corollary 70.3. We will show that the hypotheses of this corollary are met if $X=|K_M|$ and $A=|K_M-St\,\tilde{\sigma}|$. The claim then follows, since $|St\,\tilde{\sigma}|=X-A$, and the d-simplices in $\overline{St}\,\tilde{\sigma}$ are exactly those which are not in $K_M-St\,\tilde{\sigma}$.

Here the pair notation (X,A) means that X and A are topological spaces such that A is a subspace of X. Since if $X-A$ is a d-manifold, then (X,A) is a relative homology d-manifold (see comment in [Mun84], pp. 374), we only need to show that $|St\,\tilde{\sigma}|$ is a

d-manifold. This will follow from the observation that any open subset Y of a d-manifold X is a d-manifold.

The proof of this observation is easy. To see that Y is Hausdorff, let $x, y \in Y$, and let U, V be disjoint open neighborhoods of x and y, respectively, in X. Then $U \cap Y$ and $V \cap Y$ are disjoint open neighborhoods of x and y, respectively, in Y. To see that every point has an open neighborhood homeomorphic to $\mathbb{B}^d$, let $x \in Y$, and let U be an open neighborhood of x in X such that $h : U \to \mathbb{B}^d$ is a homeomorphism. Now $h(U \cap Y)$ is open in $\mathbb{B}^d$, so there is an $\epsilon > 0$ such that $B_\epsilon(x) \subseteq h(U \cap Y)$, (where $B_\epsilon(x)$ is the open ball of radius ϵ about x). Finally, $h^{-1}(B_\epsilon(x))$ is the desired open neighborhood.

For a pair (X, A) to be compact means that X and A are compact. Since $X = |K_M|$ and $A = |K_M - St\,\tilde{\sigma}|$ are finite unions of d-simplices, they are compact.

For a pair (X, A) to be triangulated means that X is triangulated by $h : |K| \to X$ in such a way that A is triangulated by a subcomplex of K under h. K_M is a simplicial complex, hence X is trivially triangulated, and $K_M - St\,\tilde{\sigma}$ is a subcomplex, so (X, A) is a triangulated pair.

All that remains to be shown is that $X - A = |St\,\tilde{\sigma}|$ is connected. Let $x, y \in |St\,\tilde{\sigma}|$, and let $z \in \text{Int}\ \tilde{\sigma}$ (remember that $z = \tilde{\sigma}$ if $\dim(\tilde{\sigma}) = 0$). Let ℓ_{xz} and ℓ_{yz} be the line segments joining x and y with z, respectively. Both ℓ_{xz} and ℓ_{yz} are contained in $|St\,\tilde{\sigma}|$, so $\ell_{xz} \cup \ell_{yz}$ is a path from x to y contained in $St\,\tilde{\sigma}$, so $St\,\tilde{\sigma}$ is path connected, hence is connected.

This finishes the proof that the hypotheses of [Mun84] Corollary 70.3 are met, and hence the claim is proven. Apply the claim with $\tilde{\sigma} = \sigma(\alpha_{i_0}, \ldots, \alpha_{i_\ell})$. Let σ_j be a simplex in the sequence $\sigma_0, \ldots, \sigma_m$, where $0 \leq j < m$. By the definition of *switch* on the d-simplices of K_M, $\sigma_j \cap \sigma_{j+1}$ is a (d–1)-simplex if and only if $\sigma_{j+1} = switch_k(\sigma_j)$ for some $0 \leq k \leq d$. Since $\sigma_j \subseteq \overline{St}\,\tilde{\sigma}$, it must be that $\tilde{\sigma} \subseteq \sigma_j$, which gives that the vertices of σ_j labeled by $i_0, \ldots, i_\ell$ are also in σ_{j+1}. Therefore, $\sigma_{j+1} = switch_k(\sigma_j)$ for some $k \in \hat{I}$.

Given t as in the statement of the lemma, and a $t' \in assoc(t_{i_0}, \ldots, t_{i_\ell})$, let σ and σ' be the corresponding d-simplices in K_M, respectively. Then

$$
\begin{aligned}
t, t' \in assoc(t_{i_0}, \ldots, t_{i_\ell}) \;&\Rightarrow\; \tilde{\sigma} \subseteq \sigma, \sigma' \\
&\Rightarrow\; \sigma, \sigma' \in \overline{St}\,\tilde{\sigma} \\
&\Rightarrow\; \exists \text{ a sequence } \sigma = \sigma_0, \ldots, \sigma_m = \sigma' \text{ of } d\text{-simplices in } \overline{St}\,\tilde{\sigma} \\
&\qquad\qquad \text{s.t. } \sigma_j \cap \sigma_{j+1} \text{ is a } (d\text{–}1)\text{-face for } j = 0, \ldots, m-1 \\
&\Rightarrow\; \exists w = w_0 \cdots w_{m-1} \in \hat{I}^* \text{ s.t. } \sigma' = switch_w(\sigma) \\
&\Rightarrow\; \exists w = w_0 \cdots w_{m-1} \in \hat{I}^* \text{ s.t. } t' = switch_w(t) \\
&\Rightarrow\; t' \in switch_{\hat{I}^*}(t)
\end{aligned}
$$

Thus $assoc(t_{i_0}, \ldots, t_{i_\ell}) \subseteq switch_{\hat{I}^*}(t)$. ∎

Given an ascending chain of cells, the proof of Lemma 4.11 just given implies the following corollary. It is basically a restatement of Lemma 4.11 in terms of the graphical interpretation of the cell-tuple structure.

Corollary 4.12 *If $c_{\alpha_{i_0}} < \cdots < c_{\alpha_{i_\ell}}$, where $c_{\alpha_{i_j}} \in C$, and $I = \{i_0, \ldots, i_\ell\}$, then the subgraph of G_M induced by $assoc(c_{\alpha_{i_0}}, \ldots, c_{\alpha_{i_\ell}})$ is connected and its edges are all labeled by elements of $\hat{I}$.*

It will be useful in the proof of Theorem 4.4 to think of *assoc* as a function taking ascending chains of cells to sets of cell-tuples. The next corollary shows that this function is injective.

Corollary 4.13 *If $\alpha_{i_0} < \cdots < \alpha_{i_n}$ and $\alpha_{j_0} < \cdots < \alpha_{j_n}$, then*

$$\{\alpha_{i_0}, \ldots, \alpha_{i_n}\} = \{\alpha_{j_0}, \ldots, \alpha_{j_n}\} \Leftrightarrow assoc(c_{\alpha_{i_0}}, \ldots, c_{\alpha_{i_n}}) = assoc(c_{\alpha_{j_0}}, \ldots, c_{\alpha_{j_n}}).$$

Proof ($\Rightarrow$) From the definition of *assoc*.

($\Leftarrow$) The proof is by contradiction. Assume without loss of generality that $\dim(\alpha_{i_k}) = i_k$ and $\dim(\alpha_{j_k}) = j_k$. Suppose $\{\alpha_{i_0}, \ldots, \alpha_{i_n}\} \neq \{\alpha_{j_0}, \ldots, \alpha_{j_n}\}$. Without loss of

generality, assume that there exists a k such that $\alpha_{i_k} \notin \{\alpha_{j_0}, \ldots, \alpha_{j_n}\}$. By Lemma 4.10, there exists $t \in assoc(c_{\alpha_{i_0}}, \ldots, c_{\alpha_{i_n}})$. If $t \notin assoc(c_{\alpha_{j_0}}, \ldots, c_{\alpha_{j_n}})$, we are done. If $t \in assoc(c_{\alpha_{j_0}}, \ldots, c_{\alpha_{j_n}})$, let $t' = switch_{i_k}(t)$. Now, $i_k \in \{i_0, \ldots, i_n\} \Rightarrow t' \notin assoc(c_{\alpha_{i_0}}, \ldots, c_{\alpha_{i_n}})$, but $i_k \notin \{j_0, \ldots, j_n\} \Rightarrow t' \in assoc(c_{\alpha_{j_0}}, \ldots, c_{\alpha_{j_n}})$. $\Rightarrow\Leftarrow$ ∎

If $0 \leq k \leq d$, let $\hat{k} = \{0, \ldots, d\} - \{k\}$. Thus $\hat{k}\text{-}orbit(t) = switch_{\{1,\ldots,k-1,k+1,\ldots,d\}^*}(t)$. The next corollary shows how the cells of a subdivided manifold are represented in $\mathcal{T}_M$. It follows from the preceding one and Lemma 4.11.

Corollary 4.14 *There is a one-to-one correspondence between k-cells and $\hat{k}$-orbits.*

The following lemma, which has an algebraic flavor, is of interest in and of itself, and will be useful when defining constructors and in connecting the cell-tuple structure to others' work. The properties, or rules, given in this lemma are analogous to the rules given for the quad-edge and facet-edge data structures. It is these rules that we will try to preserve when defining constructors. Rule (ct4) essentially shows the existence of circular orderings which is given in full detail by Theorem 4.5.

Lemma 4.15 *If $t \in T_M$ and $i \neq j \in \{0, \ldots, d\}$, then:*

(ct1) $switch_i(t) \neq t$,

(ct2) $switch_{ij}(t) \neq t$,

(ct3) $switch_{i^2}(t) = t$,

(ct4) *if* $j = i \pm 1$, $\exists m \geq 2$ *such that* $switch_{(ij)^m}(t) = t$,

(ct5) *if* $j \neq i \pm 1$, *then* $switch_{(ij)^2}(t) = t$,

Proof Properties (ct1), (ct2), and (ct3) follow from Corollary 4.3 and the definition of *switch*.

Let $t \in T_M$, and $I = \{0, \ldots, d\} - \{i, j\}$. Let $c_{\alpha_{i_k}} = t_{i_k}$ for $i_k \in I$, so that $\{c_{\alpha_{i_0}}, \ldots, c_{\alpha_{i_{d-2}}}\} = \{t_k \mid k \in I\}$. The subgraph of G_M induced by $assoc(c_{\alpha_{i_0}}, \ldots, c_{\alpha_{i_{d-2}}})$ is connected, by Corollary 4.12. Since every vertex is incident to exactly one edge labeled i

and one labeled j, this subgraph is a cycle, with alternating edges labeled i and j. If $j = i \pm 1$, this gives (ct4). If $j \neq i \pm 1$, then there are exactly two i-cells satisfying $t_{i-1} \prec c_\alpha \prec t_{i+1}$, namely $c_\alpha = t_i$ and $c_\alpha = [switch_i(t)]_i$; and there are exactly two j-cells satisfying $t_{j-1} \prec c_{\alpha'} \prec t_{j+1}$, namely $c_{\alpha'} = t_j$ and $c_{\alpha'} = [switch_j(t)]_j$. Thus by the definition of associated sets, $assoc(c_{\alpha_{i_0}}, \ldots, c_{\alpha_{i_{d-2}}})$ has exactly four elements, formed by the four possible choices for c_α and $c_{\alpha'}$. This gives (ct5). ∎

Though the properties given by Lemma 4.15 are useful, they are not enough to ensure that an abstract system having these properties can be realized as a subdivided d-manifold if $d > 3$ (see Section 1.5).

We summarize some correspondences below for easy reference.

In a subdivided d-manifold (M,C), where $C = \{c_\alpha\}_{\alpha \in I_C}$, there are one-to-one correspondences between the elements of each of the following classes:

(1) cell-tuples in $\mathcal{T}_M$,

(2) maximal sequences $c_{\alpha_0} \prec \cdots \prec c_{\alpha_d}$ in C,

(3) full ascending chains $\alpha_0 \prec \cdots \prec \alpha_d$ in $\mathcal{I}_M$,

(4) d-faces in $\mathcal{A}_M$,

(5) d-simplices in K_M,

(6) d-cells in the barycentric subdivision C_{sd}.

Given $I = \{i_0, \ldots, i_\ell\} \subset \{0, \ldots, d\}$, there are one-to-one correspondences between the elements of each of the following classes:

(1) $\hat{I}$-orbits in T_M,

(2) sequences of cells $c_{\alpha_{i_0}} < \cdots < c_{\alpha_{i_\ell}}$,

(3) ascending chains $\alpha_{i_0} < \cdots < \alpha_{i_\ell}$ in $\mathcal{I}_M$,

(4) ℓ-faces in $\mathcal{A}_M$,

(5) ℓ-simplices in K_M,

(6) ℓ-cells in the barycentric subdivision C_{sd}.

Given $0 \leq k \leq d$, there are one-to-one correspondences between the elements of each of the following classes:

(1) $\hat{k}$-orbits in T_M,

(2) k cells in C,

(3) nodes in $\mathcal{I}_M$,

(4) vertices in $\mathcal{A}_M$,

(5) vertices in K_M,

(6) vertices in the barycentric subdivision C_{sd}.

We summarize the definitions of equivalences for the objects we have defined, for easy reference:

$(M, C) \simeq (N, D) \quad \Leftrightarrow \quad \exists\, h : M \to N$ s.t.

- h is a homeomorphism,
- h takes k-cells of C onto k-cells of D.

$\mathcal{I}_M \simeq \mathcal{I}_N \quad \Leftrightarrow \quad \exists\, \imath : I_C \to I_D$ s.t.

- $\imath$ is bijective,
- $\imath$ preserves $<$,
- $\imath$ preserves dim.

$\mathcal{A}_M \simeq \mathcal{A}_N \quad \Leftrightarrow \quad \exists\, \imath : I_C \to I_D$ s.t.

- $\imath$ is bijective,
- $\imath$ preserves dim,
- $\imath$ preserves simplices.

$K_M \simeq K_N \quad \Leftrightarrow \quad \exists\, f : |K_M| \to |K_N|$ s.t.

- f is a (linear) simplicial map,
- f is bijective,
- f preserves *label*.

$\mathcal{T}_M \simeq \mathcal{T}_N \quad \Leftrightarrow \quad \exists\, \jmath : T_M \to T_N$ s.t.

- $\jmath$ is bijective,
- $switch_k(\jmath(t)) = \jmath(switch_k(t)) \;\; \forall t \in T_M, \;\; 0 \leq k \leq d.$

4.5 Proof of Theorem 4.4

Recall the statement of Theorem 4.4 from Section 4.2:

Theorem 4.4 *If (M,C) and (N,D) are subdivided d-manifolds, then the following are equivalent:*

(1) $(M,C) \simeq (N,D)$,

(2) $\mathcal{I}_M \simeq \mathcal{I}_N$,

(3) $\mathcal{T}_M \simeq \mathcal{T}_N$.

Proof The proof is organized as follows:

$$\begin{array}{ccccc} (M,C) \simeq (N,D) & \Rightarrow & \mathcal{I}_M \simeq \mathcal{I}_N & \Leftrightarrow & \mathcal{T}_M \simeq \mathcal{T}_N \\ \Uparrow & & \Downarrow & & \\ K_M \simeq K_N & \Leftarrow & \mathcal{A}_M \simeq \mathcal{A}_N & & \end{array}$$

In the rest of the proof, we will assume that $C = \{c_\alpha\}_{\alpha \in I_C}$ and $D = \{d_\beta\}_{\beta \in I_D}$.

$\underline{(M,C) \simeq (N,D) \Rightarrow \mathcal{I}_M \simeq \mathcal{I}_N}$: Let $h : M \to N$ give an equivalence between (M,C) and (N,D), i.e. h is a homeomorphism which takes k-cells onto k-cells. Define $\imath : I_C \to I_D$ by $\imath(\alpha) = index(h(c_\alpha))$. (The *index* function was defined in Section 2.4. In this context, $index(h(c_\alpha)) = \beta \Leftrightarrow h(c_\alpha) = d_\beta$.)

It must be shown that $\imath$ is well-defined, bijective, preserves $<$, and preserves dim.

$\imath$ is well-defined because h takes k-cells onto k-cells.

$\imath$ is injective: $\alpha_1 \neq \alpha_2 \Rightarrow c_{\alpha_1} \neq c_{\alpha_2} \Rightarrow h(c_{\alpha_1}) \neq h(c_{\alpha_2}) \Rightarrow \imath(\alpha_1) \neq \imath(\alpha_2)$. $\imath$ is surjective: if $\beta \in I_D$, let $\alpha = index(h^{-1}(d_\beta))$. Then $\imath(\alpha) = index(h(h^{-1}(d_\beta))) = \beta$.

$\imath$ preserves dim : $\dim(\alpha) = \dim(c_\alpha) = \dim(h(c_\alpha)) = \dim(\imath(\alpha))$.

$\imath$ preserves $<$: $\alpha_1 < \alpha_2 \Leftrightarrow c_{\alpha_1} < c_{\alpha_2} \Leftrightarrow h(c_{\alpha_1}) < h(c_{\alpha_2}) \Leftrightarrow \imath(\alpha_1) < \imath(\alpha_2)$.

$\underline{\mathcal{I}_M \simeq \mathcal{I}_N \Rightarrow \mathcal{A}_M \simeq \mathcal{A}_N}$: Let $\imath : I_C \to I_D$ give an equivalence between $\mathcal{I}_M$ and $\mathcal{I}_N$, i.e. $\imath$ is bijective, and preserves $<$ and dim. We must show that $\imath$ preserves simplices, hence gives an equivalence between $\mathcal{A}_M$ and $\mathcal{A}_N$. If $\{\alpha_0, \ldots, \alpha_k\} \in \mathcal{A}_M$, then there exists an ordering $\alpha_{i_0}, \ldots, \alpha_{i_k}$ of the elements of $\{\alpha_0, \ldots, \alpha_k\}$ so that $\alpha_{i_0} < \cdots < \alpha_{i_k}$. Since $\imath$ preserves $<$, this implies that $\imath(\alpha_{i_0}) < \cdots < \imath(\alpha_{i_k})$. So $\{\imath(\alpha_{i_0}), \ldots, \imath(\alpha_{i_k})\} \in \mathcal{A}_N$. The same argument applied to $\imath^{-1}$ shows that $\{\beta_0, \ldots, \beta_k\} \in \mathcal{A}_N$ implies $\{\imath^{-1}(\beta_0), \ldots, \imath^{-1}(\beta_k)\} \in \mathcal{A}_M$. Thus $\{\alpha_0, \ldots, \alpha_k\} \in \mathcal{A}_M$ if and only if $\{\imath(\alpha_{i_0}), \ldots, \imath(\alpha_{i_k})\} \in \mathcal{A}_N$.

$\underline{\mathcal{A}_M \simeq \mathcal{A}_N \Rightarrow K_M \simeq K_N}$: This follows from [Mun84] Theorem 3.1(b), when it is noted that the isomorphism produced there preserves labels: if $\imath : I_C \to I_D$ gives an equivalence between $\mathcal{A}_M$ and $\mathcal{A}_N$, then the isomorphism $j : |K_M| \to |K_N|$ produced by the theorem has the property that $j(v_\alpha) = v_{\imath(\alpha)}$, so $label(v_\alpha) = \dim(\alpha) = \dim(\imath(\alpha)) = label(v_{\imath(\alpha)}) = label(j(v_\alpha))$.

$\underline{K_M \simeq K_N \Rightarrow (M,C) \simeq (N,D)}$: Let $f : |K_M| \to |K_N|$ give an equivalence between $|K_M|$ and $|K_N|$, i.e. f is a bijective simplicial map, which preserves *label*. Let $\psi_M : |K_M| \to M$ and $\psi_N : |K_N| \to N$ be maps giving barycentric subdivisions. These exist by Lemma

4.6. Define $h : M \to N$ by $h = \psi_N \circ f \circ \psi_M^{-1}$. This is a composition of homeomorphisms, so is a homeomorphism. We must show that h takes k-cells onto k-cells.

Define $\imath : I_C \to I_D$ by $\imath(\alpha) = \beta \Leftrightarrow f(v_\alpha) = v_\beta$. This is well-defined since f is an equivalence. Let c_α be a k-cell in C. Then

$$\begin{aligned}
h(c_\alpha) &= h\Big(\bigcup_{\alpha_0<\cdots<\alpha_\ell<\alpha} c_{sd}(\alpha_0,\ldots,\alpha_\ell,\alpha)\Big) \\
&= \bigcup_{\alpha_0<\cdots<\alpha_\ell<\alpha} \psi_N \circ f \circ \psi_M^{-1}(c_{sd}(\alpha_0,\ldots,\alpha_\ell,\alpha)) \\
&= \bigcup_{\alpha_0<\cdots<\alpha_\ell<\alpha} \psi_N \circ f(\operatorname{Int}\sigma(\alpha_0,\ldots,\alpha_\ell,\alpha)) \\
&= \bigcup_{\imath(\alpha_0)<\cdots<\imath(\alpha_\ell)<\imath(\alpha)} \psi_N(\operatorname{Int}\sigma(\imath(\alpha_0),\ldots,\imath(\alpha_\ell),\imath(\alpha))) \\
&= \bigcup_{\imath(\alpha_0)<\cdots<\imath(\alpha_\ell)<\imath(\alpha)} c_{sd}(\imath(\alpha_0),\ldots,\imath(\alpha_\ell),\imath(\alpha)) \\
&= d_{\imath(\alpha)},
\end{aligned}$$

which is a k-cell in D.

The first and last equalities use property (b) of Lemma 4.6. The rest follow from the definition of h, and the fact that f is an equivalence.

$\underline{\mathcal{I}_M \simeq \mathcal{I}_N \Rightarrow \mathcal{T}_M \simeq \mathcal{T}_N}$: Let $\imath : I_C \to I_D$ be an equivalence between $\mathcal{I}_M$ and $\mathcal{I}_N$, i.e. $\imath$ is bijective, and preserves $<$ and dim. Define $\jmath : T_M \to T_N$ by $\jmath((c_{\alpha_0},\ldots,c_{\alpha_d})) = (d_{\imath(\alpha_0)},\ldots,d_{\imath(\alpha_d)})$. We must show that $\jmath$ is well-defined, bijective and preserves *switch*.

$\jmath$ is well-defined:

$$\begin{aligned}
(c_{\alpha_0},\ldots,c_{\alpha_d}) \in T_M &\Leftrightarrow \alpha_0 < \cdots < \alpha_d \\
&\Leftrightarrow \imath(\alpha_0) < \ldots < \imath(\alpha_d) \\
&\Leftrightarrow (d_{\imath(\alpha_0)},\ldots,d_{\imath(\alpha_d)}) \in T_N.
\end{aligned}$$

$\jmath$ is injective: If $t \neq t' \in T_M$, then there is a k such that $t_k \neq t'_k$, and $\imath(index(t_k)) \neq \imath(index(t'_k)) \Rightarrow [\jmath(t)]_k \neq [\jmath(t')]_k \Rightarrow \jmath(t) \neq \jmath(t')$.

$\jmath$ is surjective: If $(d_{\beta_0}, \ldots, d_{\beta_d}) \in T_N$, then $\beta_0 < \cdots < \beta_d \Rightarrow \imath^{-1}(\beta_0) < \cdots < \imath^{-1}(\beta_d) \Rightarrow (c_{\imath^{-1}(\beta_0)}, \ldots, c_{\imath^{-1}(\beta_d)}) \in T_M$, and $\jmath((c_{\imath^{-1}(\beta_0)}, \ldots, c_{\imath^{-1}(\beta_d)})) = (d_{\beta_0}, \ldots, d_{\beta_d})$.

To see that $\jmath$ preserves *switch*: Let $0 \leq k \leq d$ and $t \in T_M$. Then

$$\begin{aligned} t' = switch_k(t) &\Leftrightarrow (t_i = t'_i \Leftrightarrow i \neq k) \\ &\Leftrightarrow (index(t_i) \neq index(t'_i) \Leftrightarrow i \neq k) \\ &\Leftrightarrow (\imath(index(t_i)) \neq \imath(index(t'_i)) \Leftrightarrow i \neq k) \\ &\Leftrightarrow ([\jmath(t)]_i = [\jmath(t')]_i \Leftrightarrow i \neq k) \\ &\Leftrightarrow \jmath(t') = switch_k(\jmath(t)). \end{aligned}$$

$\underline{\mathcal{T}_M \simeq \mathcal{T}_N \Rightarrow \mathcal{I}_M \simeq \mathcal{I}_N}$: Let $\jmath : T_M \to T_N$ give an equivalence between $\mathcal{T}_M$ and $\mathcal{T}_N$ i.e. $\jmath$ is bijective, and $switch_k(\jmath(t)) = \jmath(switch_k(t))$ $\forall t \in T_M$, $0 \leq k \leq d$. We must define an equivalence $\imath : I_C \to I_D$.

Let c_α be a k-cell, and $t \in T_M$ a cell-tuple such that $t_k = c_\alpha$ (which exists by Lemma 4.10). Now, $assoc(c_\alpha) = switch_{\hat{k}}(t)$, by Lemma 4.11. It was shown in Section 4.2 that if $J \in \{0, \ldots, d\}^*$ and $\jmath$ is an equivalence between cell-tuple structures, then $switch_J(\jmath(t)) = \jmath(switch_J(t))$. Putting this together with Corollary 4.13 gives $\jmath(assoc(c_\alpha)) = \jmath(switch_{\hat{k}^*}(t)) = switch_{\hat{k}^*}(\jmath(t)) = assoc(d_\beta)$ for a unique k-cell $d_\beta \in D$.

By this argument, we can define $\imath : I_C \to I_D$ by $\imath(\alpha) = index(assoc^{-1}(\jmath(assoc(c_\alpha))))$. This also gives that dim is preserved ($\dim(\alpha) = k = \dim(\beta)$).

We must show that $\imath$ is bijective and preserves $<$.

$\imath$ is bijective: recall from Corollary 4.13 that *assoc* (as a function from I_C to 2^{T_M}) is bijective with its range, by definition *index* is bijective, and it is given that $\jmath$ is bijective.

$\imath$ preserves $<$: first note that if $\beta = \imath(\alpha)$, then $\beta = index(assoc^{-1}(\jmath(assoc(c_\alpha))))$ implies $assoc(d_\beta) = \jmath(assoc(c_\alpha))$. Now suppose $\alpha_1, \alpha_2 \in I_C, k_1 = \dim(\alpha_1)$ and $k_2 =$

$\dim(\alpha_2)$. Let $\beta_1 = \imath(\alpha_1)$ and $\beta_2 = \imath(\alpha_2)$. Then

$$\begin{aligned}
\alpha_1 < \alpha_2 &\Leftrightarrow k_1 < k_2 \quad \text{and} \quad \exists t \in T_M \text{ s.t. } t_{k_1} = c_{\alpha_1}, t_{k_2} = c_{\alpha_2} \\
&\Leftrightarrow k_1 < k_2 \quad \text{and} \quad assoc(c_{\alpha_1}) \cap assoc(c_{\alpha_2}) \neq \emptyset \\
&\Leftrightarrow k_1 < k_2 \quad \text{and} \quad \jmath(assoc(c_{\alpha_1})) \cap \jmath(assoc(c_{\alpha_2})) \neq \emptyset \\
&\Leftrightarrow k_1 < k_2 \quad \text{and} \quad assoc(d_{\beta 1}) \cap assoc(d_{\beta 2}) \neq \emptyset \\
&\Leftrightarrow k_1 < k_2 \quad \text{and} \quad \exists t \in T_N \text{ s.t. } t_{k_1} = d_{\beta 1}, t_{k_2} = d_{\beta 2} \\
&\Leftrightarrow \beta_1 < \beta_2
\end{aligned}$$

■

4.6 Proof of Theorem 4.5

Recall the statement of Theorem 4.5 from Section 4.2:

Theorem 4.5 *There exists a circular ordering of $S(c_{\alpha_{k-2}}, c_{\alpha_{k+1}})$. Furthermore, the switch structure directly represents this ordering information:*

A. *There exists a $t^0 \in T_M$ such that $t^0_{k-2} = c_{\alpha_{k-2}}$ and $t^0_{k+1} = c_{\alpha_{k+1}}$. For any such t^0, if the sequence of cell-tuples $t^0, \ldots, t^{m-1}$ is defined by*

$$t^i = \begin{cases} switch_k(t^{i-1}) & i \text{ even}, \\ switch_{k-1}(t^{i-1}) & i \text{ odd}. \end{cases}$$

then the sequence $c_{\eta_0}, \ldots, c_{\eta_{m-1}}$ defined by

$$c_{\eta_i} = \begin{cases} t^i_{k-1} & i \text{ even}, 2 \leq i \leq m \\ t^i_k & i \text{ odd}, 1 \leq i \leq m-1 \end{cases}$$

gives a circular ordering of $S(c_{\alpha_{k-2}}, c_{\alpha_{k+1}})$.

B. *There exist $c_{\eta_0}, c_{\eta_1} \in C$ such that $c_{\alpha_{k-2}} \prec c_{\eta_0} \prec c_{\eta_1} \prec c_{\alpha_{k+1}}$. For any such c_{η_0} and c_{η_1}, if the sequence $c_{\eta_0}, \ldots, c_{\eta_{m-1}}$ is defined by*

$$c_{\eta_i} = \begin{cases} switch_{k-1}(c_{\alpha_{k-2}}, c_{\eta_{i-2}}, c_{\eta_{i-1}}) & i \text{ even}, 2 \leq i \leq m \\ switch_k(c_{\eta_{i-1}}, c_{\eta_{i-2}}, c_{\alpha_{k+1}}) & i \text{ odd}, 1 \leq i \leq m-1 \end{cases}$$

gives a circular ordering of $S(c_{\alpha_{k-2}}, c_{\alpha_{k+1}})$.

Proof Let $t^0 = (c_{\alpha_0}, \ldots, c_{\alpha_d}) \in T_M$ such that $t^0_{k-2} = c_{\alpha_{k-2}}$ and $t^0_{k+1} = c_{\alpha_{k+1}}$, which is possible by Lemma 4.10. Let $T' = assoc(c_{\alpha_0}, \ldots, c_{\alpha_{k-2}}, c_{\alpha_{k+1}}, \ldots, c_{\alpha_d})$. Then $T' = switch_{\{k-1,k\}^*}(t^0)$ by Lemma 4.11.

Recall from Section 4.2 the definition of G_M, the graph whose nodes are cell-tuples and whose labeled edges represent the $switch_k$ operators. Let G be the subgraph of G_M induced by T', i.e. $G = (V, E)$, where $V = T'$ and $E = E_{k-1} \cup E_k$:

$$E = \{(t^1, t^2) \mid t^1, t^2 \in T' \text{ and } e \text{ is labeled with } k-1 \text{ or } k\}.$$

Every $t \in V$ is incident to exactly one edge labeled $k-1$ and one labeled k, and G is connected, because $T' = switch_{\{k-1,k\}^*}(t^0)$. So G is a simple circuit, whose edges are labeled alternately by $k-1$ and k.

Let

$$\begin{aligned}
t^1 &= switch_{k-1}(t^0), \\
t^2 &= switch_k(t^1), \\
t^3 &= switch_{k-1}(t^2), \\
t^4 &= switch_k(t^3), \\
&\vdots \\
t^{m-2} &= switch_k(t^{m-3}) \\
t^{m-1} &= switch_{k-1}(t^{m-2}).
\end{aligned}$$

Then $T' = \{t^0, \ldots, t^{m-1}\}$.

Let

$$\begin{aligned}
c_{\eta_0} &= t^0_{k-1} \\
c_{\eta_1} &= t^1_k \\
c_{\eta_2} &= t^2_{k-1} \\
c_{\eta_3} &= t^3_k \\
&\vdots \\
c_{\eta_{m-2}} &= t^{m-2}_{k-1} \\
c_{\eta_{m-1}} &= t^{m-1}_k
\end{aligned}$$

Let $S = S(c_{\alpha_{k-2}}, c_{\alpha_{k+1}})$. We need to show that $S = \{c_{\eta_0}, \ldots, c_{\eta_{m-1}}\}$. If $c \in S$, then $c_{\alpha_{k-2}} < c < c_{\alpha_{k+1}}$ implies that there exists a $t \in T'$ such that either $t_{k-1} = c$ or $t_k = c$, so $c \in \{c_{\eta_0}, \ldots, c_{\eta_{m-1}}\}$. This shows that $S \subset \{c_{\eta_0}, \ldots, c_{\eta_{m-1}}\}$. If $0 \leq i \leq m-1$, then $c_{\alpha_{k-2}} < c_{\eta_i} < c_{\alpha_{k+1}}$ implies $c_{\eta_i} \in S$, i.e. $\{c_{\eta_0}, \ldots, c_{\eta_{m-1}}\} \subset S$.

The two properties of a circular ordering are:

(1) c_{η_i} is a (k–1)-cell if i is even, and is a k-cell if i is odd,

(2) $c_{\eta_{i-1 \bmod m}}$ and $c_{\eta_{i+1 \bmod m}}$ share c_{η_i}, for $0 \leq i \leq m-1$.

Property (1) follows from $c_{\eta_i} = t^i_{k-1}$ if i is even, $c_{\eta_i} = t^i_k$ if i is odd. Property (2) follows from the definition of *switch*. Parts (A) and (B) follow from the proof. ∎

Note that the lemmas and corollaries used in the proofs of Theorem 4.4 and Theorem 4.5 were Lemma 4.6, Lemma 4.10, Lemma 4.11, and Corollary 4.13. When we extend the two theorems to manifolds-with-boundary, we will have to reprove these.

4.7 The Dual

It is useful to have access to the dual and original subdivisions. For instance, one might produce the d-dimensional Voronoi diagram by first constructing the d-dimensional Delaunay diagram, and then use this to build the Voronoi diagram.

The dual is represented implicitly by the incidence graph (simply 'turn it upside-down'). The quad-edge data structure and the face-edge data structure represent the dual in exactly the same manner as the primal – there is an explicit set of dual quad-edges and dual facet-edges in each, respectively. In fact, this is not necessary – the primal elements actually hold the dual implicitly.

Recall from Lemma 4.6:

$$\text{(b) } c_\alpha = \bigcup_{\alpha_0 < \cdots < \alpha_\ell < \alpha} c_{sd}(\alpha_0, \ldots, \alpha_\ell, \alpha),$$

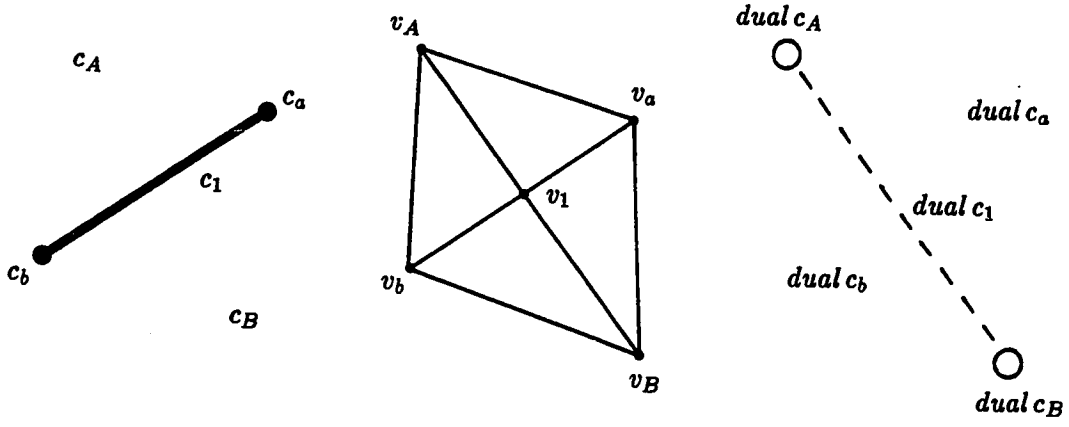


Figure 4.8: Example of dual cells

A **dual complex** to C can now be defined easily: The dual face for c_α is

$$dual\ c_\alpha = \bigcup_{\alpha<\alpha_{i_0}<\cdots<\alpha_{i_\ell}} c_{sd}(\alpha, \alpha_{i_0}, \ldots, \alpha_{i_\ell}).$$

(See Figure 4.8.) Let $C^{dual} = \{dual\ c_\alpha\}_{\alpha \in I_C}$. Of course this is not unique, since the characteristic functions are not unique, and the cells in C_{sd} are not unique, but it is unique up to equivalence.

There is a one-to-one correspondence between the original cells and the dual cells, and incidence and the orderings are preserved. The cell-tuple structure maintains both the original and dual complexes simultaneously. To operate in the dual, rather than the original, note that $switch_k$ in the original subdivided manifold is the same thing as $switch_{d-k}$ in the dual. Thus the same ordering results apply in the dual, by replacing k with $d-k$.

There are two ways one might think of representing, and operating, in the dual. The first is to just represent the primal subdivision as we have already. Then whenever it is desired to work in the dual, a flag might be changed, indicating that whenever a $switch_k$ is asked for, $switch_{d-k}$ should be accessed instead. Here the dual is represented

implicitly.

The second approach is to explicitly represent the dual by a separate cell-tuple structure. Define $T_M^{dual} = \{(dual\ c_{\alpha_d}, \ldots, dual\ c_{\alpha_0}) \mid dual\ c_{\alpha_d} \prec \cdots \prec dual\ c_{\alpha_0} \in C^{dual}\}$. There is a one-to-one correspondence between T_M and T_M^{dual}, where $(c_{\alpha_0}, \ldots, c_{\alpha_d})$ corresponds to $(dual\ c_{\alpha_d}, \ldots, dual\ c_{\alpha_0})$. Define $switch_R$ in light of this correspondence, i.e. define $switch_R : T_M \to T_M^{dual}$ by $switch_R((c_{\alpha_0}, \ldots, c_{\alpha_d})) = (dual\ c_{\alpha_d}, \ldots, dual\ c_{\alpha_0})$. A basic property is immediate from the definition:

$$switch_{RkR} = switch_{d-k}.$$

One way of thinking of the use of $switch_R$ is to think of applying $switch_R$ as saying 'work on the dual now'. Then a sequence of operations may be applied to the dual as if operating on the original complex. When finished, a second application of $switch_R$ 'brings back the original'.

Consider the set of tuples $T_M \cup T_M^{dual}$, acted on by $switch_R$ and $switch_k, 0 \leq k \leq d$. We summarize the 'algebraic properties' of this new structure in the next lemma.

Lemma 4.16 *If $t \in T_M \cup T_M^{dual}$ and $i \neq j \in \{0, \ldots, d\}$, then:*

(ct1) $switch_i(t) \neq t$,

(ct2) $switch_{ij}(t) \neq t$,

(ct3) $switch_{i^2}(t) = t$,

(ct4) *if* $j = i \pm 1$, $\exists m \geq 2$ *such that* $switch_{(ij)^m}(t) = t$,

(ct5) *if* $j \neq i, i \pm 1$, *then* $switch_{(ij)^2}(t) = t$,

(ct6) $switch_{RiR}(t) = switch_{d-i}(t)$

Either of the approaches discussed above will work in practice. The second method offers the advantage of having explicit pointers to dual cells, giving a place for information about the dual cells to be stored, if necessary. The first method uses half as much space

as the second. In this case, if explicit representation of dual cells is required, then each primal cell descriptor could include extra information about its dual, or point to a dual cell, or the implementation of a cell-tuple could include an extra array of pointers to dual cells.

The second method will make the connection between the cell-tuple structure and the quad-edge and facet-edge data structures straightforward.

4.8 Extension to Manifolds-with-Boundary

In many applications, the objects of study have boundaries, so it is important that a data structure which is going to actually be used be able to handle objects with boundaries in a consistent fashion. Applications in fields such as computer graphics, computer vision, physical simulations, and computer-aided design often model physical objects which have boundaries. Algorithms in computational geometry often use objects with boundaries. For example, an incremental convex hull algorithm may have to represent an intermediate result which is part of the final convex hull, which has a boundary.

We consider two approaches to representing boundaries. The first treats the boundary as an impenetrable skin, and only uses cell-tuples which are 'in the interior' of the manifold. In this case, some of the orderings may be simple paths rather than simple circuits. The second approach can be thought of as treating the space 'outside of the manifold' as another d-cell (though it is not a cell in general), and represents the order 'across the boundary', so that all orderings remain simple cycles. This has the advantage that the structure of the boundary is directly represented in the data structure.

If (M, C) is a subdivided d-manifold-with-boundary, $switch_k$ is well-defined on all cell-tuples except when $k = d$ and $t_{k-1} \subseteq \partial M$. In terms of cells, if $c_{\alpha_{k-1}} \prec c_{\alpha_k} \prec c_{\alpha_{k+1}}$, where $\dim(c_{\alpha_i}) = i, 0 \leq k \leq d$, then: if $k = d$ and $c_{\alpha_{k-1}} \subseteq \partial M$, then there is no $c_{\alpha'_k} \neq c_{\alpha_k}$ such that $c_{\alpha_{k-1}} \prec c_{\alpha'_k} \prec c_{\alpha_{k+1}}$.

4.8.1 First Approach

The first approach is to let the undefined case remain undefined, or define it to return some special value. Then one knows when one has come to the boundary, and can act accordingly. We give a lemma which corresponds to Corollary 4.3:

Lemma 4.17 *If (M, C) is a subdivided d-manifold-with-boundary, $c_{\alpha_{k-1}} \prec c_{\alpha_k} \prec c_{\alpha_{k+1}}$, where $0 \leq k \leq d$, $c_{\alpha_i} \in C$ and $\dim(c_{\alpha_i}) = i$, then:*

1. *If $k = d$ and $c_{\alpha_{k-1}} \subseteq \partial M$, then there is no $c_{\alpha'_k} \neq c_{\alpha_k}$ such that $c_{\alpha_{k-1}} \prec c_{\alpha'_k} \prec c_{\alpha_{k+1}}$,*
2. *otherwise, there is a unique $c_{\alpha'_k} \neq c_{\alpha_k}$ such that $c_{\alpha_{k-1}} \prec c_{\alpha'_k} \prec c_{\alpha_{k+1}}$.*

We will not prove Lemma 4.17 here. The proof is similar to that given for Lemma 4.23 in the next section.

Under the respective conditions of this claim, define *switch* for triples of cells:

1. $switch(c_{\alpha_{k-1}}, c_{\alpha_k}, c_{\alpha_{k+1}}) = \lambda$,
2. $switch(c_{\alpha_{k-1}}, c_{\alpha_k}, c_{\alpha_{k+1}}) = c_{\alpha'_k}$.

We do not need to prove the lemma analogous to Lemma 4.2, because the definition of *switch* for triples can be used to define $switch_k$, where $0 \leq k \leq d$, for cell-tuples in manifolds-with-boundary. If $t = (c_{\alpha_0}, \ldots, c_{\alpha_d}) \in T_M$, let $c_{\alpha'_k} = switch(c_{\alpha_{k-1}}, c_{\alpha_k}, c_{\alpha_{k+1}})$. If $c_{\alpha'_k} \neq \lambda$ define $switch_k(t) = (c_{\alpha_0}, \ldots, c_{\alpha_{k-1}}, c_{\alpha'_k}, c_{\alpha_{k+1}}, \ldots, c_{\alpha_d})$, and if $c_{\alpha'_k} = \lambda$ define $switch_k(t) = \lambda$ For the sake of the next definition, define $switch_k(\lambda) = \lambda$.

If $w = w_1 \ldots w_\ell \in \{0, \ldots, d\}^*$, define

$$switch_w(t) = \begin{cases} switch_{w_\ell}(switch_{w_{\ell-1}}(\cdots(switch_{w_2}(switch_{w_1}(t)))\cdots)) & \text{if } w \neq \lambda, \\ t & \text{if } w = \lambda, \end{cases}$$

Let $S = S(c_{\alpha_{k-2}}, c_{\alpha_{k+1}})$ as before, and let $m = \#(S)$. **A circular ordering of S** is an ordering $c_{\eta_0}, \ldots, c_{\eta_{m-1}}$ such that:

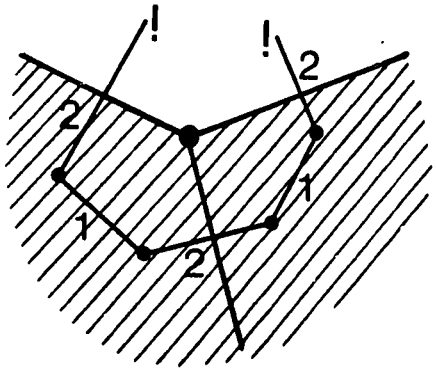


Figure 4.9: Circular ordering at the boundary, first approach

(1) c_{η_i} is a (k–1)-cell if i is even, and is a k-cell if i is odd,

(2) $c_{\eta_{i-1}}$ and $c_{\eta_{i+1}}$ share c_{η_i}, for $0 < i < m-1$,

(3) (a) If $k = d$ and $c_{\alpha_{k-2}} \subseteq \partial M$, then m is odd, and $c_{\eta_0}, c_{\eta_{m-1}} \subseteq \partial M$,

(b) Otherwise, m is even, and c_{η_1} and $c_{\eta_{m-1}}$ share c_{η_0}.

(See Figure 4.9.)

The statements of Theorem 4.4 and Theorem 4.5 may be modified accordingly. The necessary lemmas, and the proofs of the two theorems, require only modifications for the special cases involving the boundary.

4.8.2 Second Approach

It would be nice if we could simply let the space 'outside' the manifold-with-boundary be another cell. First, this would make the extension to manifolds-with-boundary immediate. Also, it would allow us to access incidence and ordering information on the boundary of the manifold as easily as can be done for the boundary of a cell. For example, given a face in the boundary of the manifold, it may be useful to access incident

faces which are also in the boundary. This might occur in the implementation of an incremental convex hull or Delaunay triangulation algorithm.

In general, however, the space 'outside' a manifold-with-boundary is not a cell. A simple example is a solid torus embedded in $\mathbb{R}^3$. The complement of this is a solid torus with an interior point removed, which is not a 3-cell.

It is possible to modify the cell-tuple structure so that it behaves almost as if the boundary of the manifold were simply the boundary of a single cell. We can't say that the complement of M is a cell, but we can show that the boundary of M is a $(d-1)$-manifold (without boundary). Thus we can use all of our previous results to represent the boundary by cell-tuples of one lower dimension. In fact, what we will do is define an abstract special cell, c_{α_∞}, whose 'boundary' is defined to be ∂M, and then define cell-tuples as before. When it comes to proving the necessary properties, however, we will have to treat this cell as a special case.

Lemma 4.18 *∂M is a $(d-1)$-manifold (without boundary).*

Proof If $x \in \partial M$, then there exists an open neighborhood U of x which is homeomorphic to $\mathbb{B}^d_{1/2}$. Let $h : U \to \mathbb{B}^d_{1/2}$ be such a homeomorphism. Let $\mathbb{R}^d_1 = \{(x_1, \ldots, x_d) \in \mathbb{R}^d \mid x_1 = 0\}$.

If $y \in U \cap \partial M$, then $h(y) \in \mathbb{B}^d_{1/2} \cap \mathbb{R}^d_1$ (see [Mun84] pp. 198: essentially, h gives a coordinate patch about x. It also a coordinate patch about y, hence maps y into $\mathbb{B}^d_{1/2} \cap \mathbb{R}^d_1$).

If $y \in U - \partial M$ then $h(y) \notin \mathbb{B}^d_{1/2} \cap \mathbb{R}^d_1$ (y has an open neighborhood homeomorphic to $\mathbb{B}^d$).

Thus, $U \cap \partial M$ is homeomorphic to $\mathbb{B}^d_{1/2} \cap \mathbb{R}^d_1$, which is homeomorphic to $\mathbb{B}^{d-1}$. By definition of subspace, $U \cap \partial M$ is open in ∂M. So every point in ∂M has an open neighborhood homeomorphic to $\mathbb{B}^{d-1}$.

If $x \neq y \in \partial M$, then there exist disjoint open (in M) neighborhoods U and V of x and y, respectively. Then $U \cap \partial M$ and $V \cap \partial M$ are disjoint open (in ∂M) neighborhoods of x and y, respectively. So ∂M is Hausdorff. ∎

By Fact 4.9, $\psi_M^{-1}(\partial M)$ is the polytope of a subcomplex of K_M. Since ψ_M is a homeomorphism, $\partial|K_M| = \psi_M^{-1}(\partial M)$. Let $\psi_M^\partial = \psi_M|_{\partial|K_M|}$ be the restriction of ψ_M to $\partial|K_M|$. Then the following is immediate:

Lemma 4.19 *$\psi_M^\partial : \partial|K_M| \to \partial M$ is a triangulation.*

Lemma 4.20 *If $c_{\alpha_{d-2}} \subseteq \partial M$ is a $(d-2)$-cell, then there are exactly two $(d-1)$-cells in ∂M which contain $c_{\alpha_{d-2}}$.*

This follows from the fact that $\partial|K_M|$ is a $(d-1)$-manifold and Fact 4.8.

The next result is that a cell is either completely in the boundary of the manifold, or is completely in its interior.

Lemma 4.21 *$c_\alpha \cap \partial M \neq \emptyset$ if and only if $c_\alpha \subseteq \partial M$.*

Proof ($\Leftarrow$) Trivial.

($\Rightarrow$) We use the fact that if the interior of a simplex in K_M intersects $\partial|K_M|$, then that simplex is contained in $\partial|K_M|$.

Let $c_\alpha \in C$ be such that $c_\alpha \cap \partial M \neq \emptyset$. If c_α was a d-cell, then $c_\alpha \subseteq \text{Int } M$ (because every point of c_α has an open neighborhood homeomorphic to $\mathbb{B}^d$). Thus c_α is not a d-cell.

Let $x \in c_\alpha \cap \partial M$. Then $\psi_M^{-1}(x) \in \text{Int } \sigma$ for some $\sigma \in K_M$, and $\psi_M^{-1}(x) \in \partial|K_M|$, so $\sigma \subseteq \partial|K_M|$. Since $c_\alpha = \bigcup_{\alpha_0 < \cdots < \alpha_\ell < \alpha} c_{sd}(\alpha_0, \ldots, \alpha_\ell, \alpha)$ (from Lemma 4.6), $\sigma = \sigma(\alpha_0, \ldots, \alpha_\ell, \alpha)$ for some (possibly empty) sequence $\alpha_0 < \cdots < \alpha_\ell$. By the observation at the beginning of the proof, this implies that $v_\alpha \in \partial|K_M|$.

Consider the case where c_α is a $(d-1)$-cell. By Fact 4.7, there is a $(d-1)$-simplex σ' such that $v_\alpha \in \sigma' \subseteq \partial|K_M|$. By Fact 4.8, there is a unique d-simplex σ'' in K_M which contains σ'. Since $\sigma' \subseteq \partial|K_M|$, none of its vertices are labeled d. Therefore, the vertex

in $\sigma'' - \sigma'$ is v_{α_d}, where $\dim(\alpha_d) = d$. This α_d is unique, since σ'' is unique. Thus there is exactly one d-cell c_{α_d} incident to c_α.

Now consider any $(d-1)$-simplex $\sigma = \sigma(\alpha_{i_0}, \ldots, \alpha_{i_{d-2}}, \alpha)$ such that $\alpha_{i_0} < \cdots < \alpha_{i_{d-2}} < \alpha$. Since α_d is unique, $\sigma(\alpha_{i_0}, \ldots, \alpha_{i_{d-2}}, \alpha, \alpha_d)$ is the unique d-simplex in K_M containing σ. Thus σ is in $\partial|K_M|$ (again by Fact 4.8). Also, all of σ's subsimplices are in $\partial|K_M|$, by the observation at the beginning of this proof. So every simplex of the form $\sigma(\alpha_{i_0}, \ldots, \alpha_{i_\ell}, \alpha)$ is contained in $\partial|K_M|$, and $\psi_M(\sigma(\alpha_{i_0}, \ldots, \alpha_{i_\ell}, \alpha)) \subseteq \partial M$.

Thus $c_\alpha = \bigcup_{\alpha_0 < \cdots < \alpha_\ell < \alpha} c_{sd}(\alpha_0, \ldots, \alpha_\ell, \alpha)$ is contained in ∂M.

Now consider the case where c_α is a k-cell, $0 \leq k < d-1$. As before, there is a $(d-1)$-simplex σ' such that $v_\alpha \in \sigma' \subseteq \partial|K_M|$, and also as before, one of its vertices is labeled by $d-1$, say this vertex is $v_{\alpha_{d-1}}$. Then $c_\alpha < c_{\alpha_{d-1}} \Rightarrow c_\alpha \subseteq \dot{c}_{\alpha_{d-1}}$, so $c_\alpha \subseteq \partial M$ by continuity of ψ_M. ∎

The next corollary follows from the definition of subdivided manifold and from Lemma 4.18 and Lemma 4.21.

Corollary 4.22 *Let* $C^\partial = \{c \in C \mid c \subseteq \partial M\}$. *Then* $(\partial M, C^\partial)$ *is a* $(d-1)$*-subdivided manifold.*

Define the abstract cell c_{α_∞} so that $\dim(c_{\alpha_\infty}) = d$ and $c < c_{\alpha_\infty}$ for all $c \in C^\partial$. Let $C^+ = C \cup \{c_{\alpha_\infty}\}$.

$\mathcal{A}_M$ and K_M are defined as before, using only cells from C. Define T_M, using the set C^+:

$$T_M = \{(c_{\alpha_0}, \ldots, c_{\alpha_d}) \mid c_{\alpha_0} \prec \cdots \prec c_{\alpha_d},\ c_{\alpha_i} \in C^+\}.$$

Define *assoc* as before, using all ascending chains whose cells are in C^+. Since Lemma 4.10 is only for chains from C, its proof goes through as before. We need to prove Corollary 4.3 for C^+.

Lemma 4.23 *If* (M, C) *is a subdivided* d*-manifold-with-boundary,* $\dot{c}_{\alpha_{k-1}} \prec c_{\alpha_k} \prec c_{\alpha_{k+1}}$,

where $0 \leq k \leq d$, $c_{\alpha_k} \in C^+$ *and* $\dim(c_{\alpha_k}) = k$, *then there is a unique* $c_{\alpha'_k} \neq c_{\alpha_k}$ *such that* $c_{\alpha_{k-1}} \prec c_{\alpha'_k} \prec c_{\alpha_{k+1}}$.

Proof If neither c_{α_k} nor $c_{\alpha_{k+1}}$ are equal to c_{α_∞}, then the proof goes through exactly as before.

If $c_{\alpha_{k+1}} = c_{\alpha_\infty}$, then $c_{\alpha_{k-1}}, c_{\alpha_k} \subseteq \partial M$, so by Lemma 4.20, there is a unique $c_{\alpha'_k}$ such that $c_{\alpha_{k-1}} \prec c_{\alpha'_k} \subseteq \partial M$, i.e. $c_{\alpha_{k-1}} \prec c_{\alpha'_k} \prec c_{\alpha_\infty}$.

If $c_{\alpha_k} = c_{\alpha_\infty}$, then $k = d$ and by Lemma 4.10, there is a tuple $t \in T_M$ such that $t_{d-1} = c_{\alpha_{k-1}}$ and $t_i \in C$, for $0 \leq i \leq d$. Now $c_{\alpha_{k-1}} \prec t_d$, and $t_d \in C \Rightarrow t_d \neq c_{\alpha_\infty}$. Letting $c_{\alpha'_d} = t_d$ shows existence. We must show uniqueness. Let $c_{\alpha'_i} = t_i$, for $0 \leq i \leq d$. Now, $\sigma(\alpha'_0, \ldots, \alpha'_{d-1})$ is a $(d\text{-}1)$-cell in $\partial|K_M|$. By Fact 4.8 there exists a unique α_d such that $\sigma(\alpha'_0, \ldots, \alpha'_{d-1}, \alpha_d)$ is a d-cell in K_M, i.e. $c_{\alpha'_d}$ is unique (in C). ∎

As in the first approach, we can use this to define the *switch*$_k$ for cell-tuples: If $t = (c_{\alpha_0}, \ldots, c_{\alpha_d}) \in T_M$, let $c_{\alpha'_k} = switch(c_{\alpha_{k-1}}, c_{\alpha_k}, c_{\alpha_{k+1}})$, and define $switch_k(t) = (c_{\alpha_0}, \ldots, c_{\alpha_{k-1}}, c_{\alpha'_k}, c_{\alpha_{k+1}}, \ldots, c_{\alpha_d})$.

The next corollary shows that the cell-tuple structure for the boundary of (M,C) is essentially contained in the cell-tuple structure for (M,C) itself. It follows from the definition of *switch*$_k$ and Corollary 4.22.

Lemma 4.24 *If* t *and* t' *are in the cell-tuple structures of* (M,C) *and* $(\partial M, C^\partial)$, *respectively, such that* $t_d = c_{\alpha_\infty}$ *and* t *and* t' *agree on components* $0, \ldots, d-1$, *then* $switch_k(t)$ *and* $switch_k(t')$, *for* $0 \leq k \leq d-1$, *also agree on components* $0, \ldots, d-1$.

Finally, we need to show Lemma 4.11 in the with-boundary case. Note that it will not work for $assoc(c_{\alpha_\infty})$, since ∂M isn't even necessarily connected.

Lemma 4.25 *If* $t \in T_M$ *and* $I = \{i_0, \ldots, i_\ell\} \subseteq \{0, \ldots, d\}$, $I \neq \emptyset$, *then, with the exception of the case where of* $I = \{d\}$ *and* $t_d = c_{\alpha_\infty}$, $assoc(t_{i_0}, \ldots, t_{i_\ell}) = switch_{I^*}(t)$.

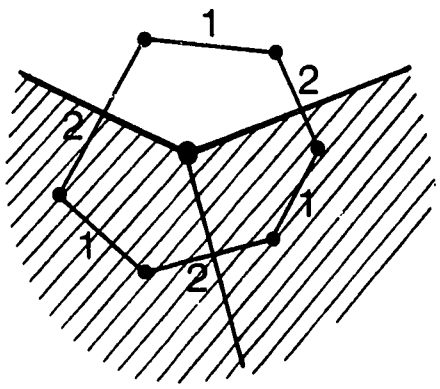


Figure 4.10: Circular ordering at the boundary, second approach

Proof That $switch_{\hat{I}^*}(t) \subseteq assoc(t_{i_\ell}, \ldots, t_{i_0})$ goes through exactly as before.

We now show $assoc(t_{i_0}, \ldots, t_{i_\ell}) \subseteq switch_{\hat{I}^*}(t)$.

Case 1: $t_d \neq c_{\alpha_\infty}$. If $d \in I$ then the proof goes through as before.

If $d \notin I$ the proof goes through as before for the set of all $t' \in assoc(t_{i_0}, \ldots, t_{i_\ell})$ such that $t'_d \neq c_{\alpha_\infty}$ i.e. $assoc(t_{i_0}, \ldots, t_{i_\ell}) \cap \{t' \in T_M \mid t'_d \neq c_{\alpha_\infty}\} \subseteq switch_{\hat{I}^*}(t)$. If $t' \in assoc(t_{i_0}, \ldots, t_{i_\ell})$ and $t'_d = c_{\alpha_\infty}$, then $switch_d(t') \in assoc(t_{i_0}, \ldots, t_{i_\ell}) \cap \{t' \in T_M \mid t'_d \neq c_{\alpha_\infty}\} \Rightarrow t' \in switch_{\hat{I}^*}(t)$.

Case 2: $t_d = c_{\alpha_\infty}$. If $d \notin I$, then $[switch_d(t)]_d \neq c_{\alpha_\infty}$, and by case 1 $assoc(t_{i_0}, \ldots, t_{i_\ell}) \subseteq switch_{\hat{I}^*}(switch_d(t)) = switch_{\hat{I}^*}(t)$. If $d \in I$, then the result follows from Lemma 4.11 applied to $(\partial M, C^\partial)$, and Lemma 4.24. ∎

These lemmas are all we need to re-prove Theorem 4.4 and Theorem 4.5.

Now all of the ordering 'internal' to the manifold is as before, and the ordering of the cells in the boundary is consistent with that inside.

It is always possible to detect if a particular tuple is 'on the outside' of the boundary, by testing if $t_d = c_{\alpha_\infty}$. In the case that $t_d = c_{\alpha_\infty}$, applying $switch_k$, with $k \neq d$, moves around on the boundary as a (d–1)-subdivided manifold, with all of the boundary ordering available. (See Figure 4.10.)

The extended cell-tuple structure given in this approach has the simplicity and uniformity that the cell-tuple structure for manifolds (without boundary) had. In addition, the boundary is represented in a manner that is consistent with the interior of the manifold, and allows access to all incidence and ordering information for the boundary as a subdivided manifold of one lower dimension.

4.9 Relation to Other Implicit-Cell Representations

4.9.1 Introduction

In this section we will examine the relationship of the cell-tuple structure with the quad-edge data structure, the facet-edge data structure, n-G-maps, and chamber systems. In particular, we will discuss how the basic elements and the query operators correspond. Note that we may do the comparison only on the objects allowed by both the cell-tuple structure and the representation it is being compared with. The comparison between constructors (operators which modify structure) will appear in the next chapter.

4.9.2 Relation to the Quad-Edge Data Structure

The set of objects to which the cell-tuple structure and the quad-edge data structure both apply are subdivided 2-manifolds (under our definition, and without boundaries). There is a one-to-one correspondence between primal directed, oriented edges in the edge algebra and the cell-tuples in the cell-tuple structure. To see this correspondence visually, first consider placing the directed, oriented edges as in Figure 4.11. The direction of the shaft of the arrow shows the direction, and which side the half-head is on gives the 'current direction of clockwise rotation' around the base of the arrow. Figure 4.12 shows a slightly larger piece of a subdivision, showing primal quad-edges on the left and cell-tuples on the right. The location gives the correspondence.

Given a tuple, the corresponding directed, oriented edge is given by: t_1 tells which edge is involved, t_0 gives the direction (as the origin of travel, i.e. the base of the arrow),

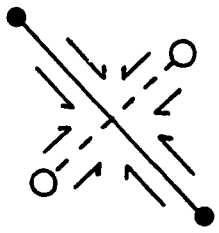


Figure 4.11: Another depiction of directed, oriented edges

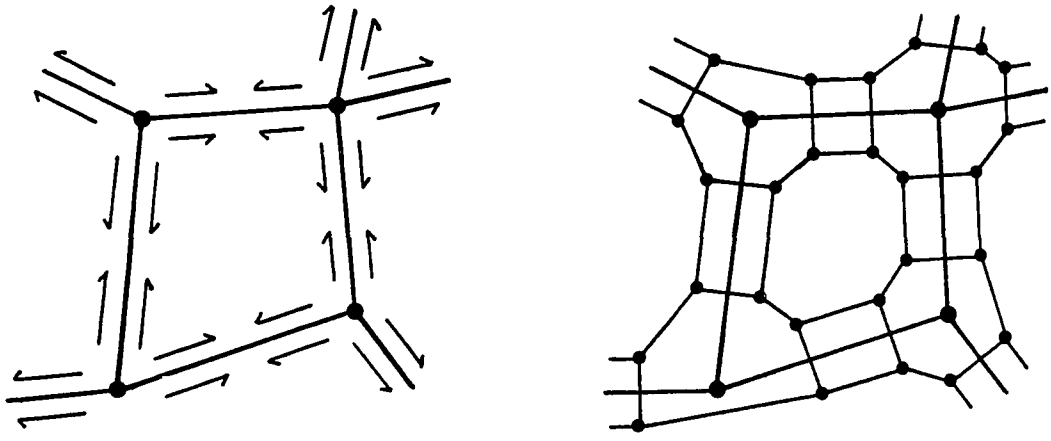


Figure 4.12: Pictorial comparison between directed, oriented edges and cell-tuples

and t_2 gives the orientation (as the direction of rotation, i.e. which face the half-head points into). We will use the notation $\wp$ for this correspondence, $\wp : T_M \rightarrow ES$.

To extend this correspondence to the complete edge-algebra, we must include the dual cell-tuple structure explicitly, as described in Section 4.7. A similar correspondence to that described in the preceding paragraph then holds.

We can indicate the correspondence between operators in the two structures informally (where e is a directed, oriented edge in the edge-algebra, which corresponds to the cell-tuple t in the cell-tuple structure), by showing how the operations on the edge-algebra may be 'written in terms of' the *switch* operators:

$$e\ Onext \leftrightarrow switch_{12}(t),$$

$$e\ Flip \leftrightarrow switch_2(t),$$

$$e\ Rot \leftrightarrow switch_{2R}(t).$$

The correspondence is given more formally by stating that for all t in the set of cell-tuples:

$$\wp(t)Onext = \wp(switch_{12}(t)),$$

$$\wp(t)Flip = \wp(switch_2(t)),$$

$$\wp(t)Rot = \wp(switch_{2R}(t)).$$

The operations on the cell-tuple structure may be 'written in terms of' the edge-algebra operators as:

$$switch_0(t) \leftrightarrow e\ Rot^2\ Flip,$$

$$switch_1(t) \leftrightarrow e\ Onext\ Flip$$

$$switch_2(t) \leftrightarrow e\ Flip.$$

The ten properties (E1) – (E5) and (F1) – (F5) for the edge algebra were listed in Section 3.3.3. To show that the the cell-tuple structure and the edge algebra are equivalent means showing that the properties (ct1) – (ct5) from Lemma 4.15 hold if

and only if (E1) – (E5) and (F1) – (F5) hold. For example, to show that (ct1) – (ct5) imply F2: if $e = \wp(t)$, then $t = switch_{22}(t) = switch_{2112}(t) = switch_{212212}(t)$ implies $e = \wp(switch_{212212}(t)) = \wp(switch_{2122}(t))$ *Onext* $= \wp(switch_{212}(t))$ *Flip Onext* $= \wp(switch_2(t))$ *Onext Flip Onext* $= \wp(t)$*Onext Flip Onext Flip* $= e$ *Flip Onext Flip Onext*.

Guibas and Stolfi introduce a refinement of the original subdivision, called the 'completion', which is exactly the generalized barycentric subdivision.

We have already noted the difference between the classes of objects represented by the cell-tuple structure and the quad-edge data structure. We have now shown how the basic elements and the query operations in each may be related. The basic elements in both are essentially the same, although they are described somewhat differently. The power of the query operators is the same in both cases; the main difference is that the *switch* operators correspond naturally to operations on the cell-tuples themselves, and generalize uniformly to higher dimensions.

4.9.3 Relation to the Facet-Edge Data Structure

As in the case of the edge algebra, there is a one-to-one correspondence between the cell-tuples in the cell-tuple structure for a subdivided 3-manifold and the primal facet-edges in the facet-edge data structure: t_1 gives the edge, t_2 gives the facet, t_0 gives the direction within the face, and t_3 gives the direction around the edge. This same correspondence holds for the dual, and to show how the cell-tuple structure and the facet-edge data structure agree, we must explicitly include the dual cell-tuple structure and the $switch_R$ operator. Then we can 'rewrite' the facet-edge operators in terms of the $switch_k$ operators (below, the facet-edge a corresponds to the cell-tuple t):

$a\,Clock \leftrightarrow switch_{03}(t)$,

$a\,Enext \leftrightarrow switch_{01}(t)$,

$a\,Fnext \leftrightarrow switch_{23}(t)$,

$a\,Rev \leftrightarrow switch_{3}(t)$,

$a\,Sdual \leftrightarrow switch_{R}(t)$.

The class of 3-complexes allowed in [DL 87] are subdivisions of spaces homeomorphic to $\overline{\mathbb{B}}^3$ or $\mathbb{B}^3$, where the cell boundaries are allowed to self-intersect (though, as noted in Section 3.4.2, others may be generated by using the constructive operators). So in the case of three dimensions, our definition of subdivided manifolds allows a larger class of underlying spaces than the facet-edge data structure, but restricts the class of cells allowed to those without self-intersecting boundaries.

4.9.4 Relation to n-G-maps

As darts are abstract entities, without much explicitly developed about their nature as topological objects, it is hard to give a pictorial, intuitive correspondence between darts and cell-tuples. However, taking the cell-tuple structure as an abstract system obeying Lemma 4.15, the rules followed by each are roughly the same: α_i acts on B in n-G-maps in the same way that $switch_i$ acts on T_M. The only difference is the way that boundaries are handled.

In an n-G-map, $b\alpha_n = b$ if b is a boundary dart, whereas in the cell-tuple structure, either $switch_n(t)$ is equal to a special value, or to another 'outside' cell-tuple. If we take the first method in the cell-tuple structure for boundaries, setting $switch_n(t)$ to be equal to t would be just as easy to detect as setting it to an arbitrary value, i.e. this would make the cell-tuple structure agree completely with n-G-maps in this abstract framework. Recall that in this approach, the boundary is represented only implicitly. The definition of n-G-map does not include any explicit representation of the boundary

as an (n–1)-G-map, as we gave in the second approach for handling the boundary in the cell-tuple structure, so direct access to the boundary as a subdivided (d–1)-manifold is not possible.

The objects represented are a very general class of objects, larger than the class of subdivided d-manifolds (for instance, recall the examples given in Section 3.5.1). Thus for the class of objects represented by n-G-maps, we cannot give any results about ordering.

4.9.5 Relation to Chamber Systems

The concept of chamber system described in Section 3.5.2, when appropriately applied to a subdivided manifold (M, C), is basically a labeled version of the 1-skeleton of the dual of the barycentric subdivision of (M, C).

If a subdivided d-manifold (M, C) is considered as a geometry over the set $0, \ldots, d$, so that the elements of the geometry are the cells of C and the labeling of the elements is the dimension of the cells, then the complex (in the context of Tits' work) associated with this geometry is equivalent to the abstract simplicial complex $\mathcal{A}_M$ defined in Section 4.3. The chambers are then the d-simplices of $\mathcal{A}_M$. The partition $\mathcal{P}_i$ in the associated chamber system is a essentially a matching between chambers, which corresponds to the pairs of d-simplices associated by $switch_i$. ($switch$ is defined on the d-simplices of K_M in Section 4.3. By the correspondence between the simplices of $\mathcal{A}_M$ and K_M, this gives a definition of $switch$ on $\mathcal{A}_M$.) By the one-to-one correspondence between the d-simplices of $\mathcal{A}_M$ and the cell-tuples of T_M, there is a one-to-one correspondence between chambers and cell-tuples, so that a pair of chambers are in the partition $\mathcal{P}_i$ if and only if the corresponding pair of cell-tuples are connected by $switch_i$.

From the argument just given, it follows that the chamber system associated with a subdivided manifold is essentially the same as the edge-labeled graph characterization of the cell-tuple structure.

Tits' does not consider the representation of subdivided manifolds or ordering, and does not consider computational issues. The fact that the same basic idea has occurred

in two independent investigations of rather different phenomena perhaps lends weight to its being a useful one.

Chapter 5

Constructors

5.1 Introduction

So far in this dissertation we have focused entirely on representing objects, but in practice we want to create, manipulate, change, merge, separate and destroy objects as well. We will use the term 'constructor' to refer to operators which effect such changes.

When defining constructors there are a number of considerations which come into play. It would be desirable to have a rich set of powerful constructors which correspond to natural operations on the objects in an application and are independent of the implementation. On the other hand, it would be desirable to have a small set of constructors which are easy to implement, and which can be proven to maintain required topological properties. One might refer to these as high and low level constructors, respectively, and it may make sense to implement high level constructors 'on top of' low level ones.

We will break the discussion of constructors into two distinct parts, which correspond roughly to the distinction made in the previous paragraph. First we will describe the constructors that have been defined for the implicit-cell representations considered in this dissertation, which are given in terms of the combinatorial structure of the representation rather than the topology of the objects. This includes the constructors for the quad-edge data structure, the facet-edge data structure, and n-G-maps, as well as a discussion of

the possible constructors for the cell-tuple structure. These may be thought of as low level, because they are phrased in terms of the representations, and their effects on the cells of the subdivision may not seem natural.

Then we will discuss constructors defined directly on cells within subdivided manifolds, without reference to any particular representation. This will include Euler operators, used in constructive solid geometry, and a new set of constructors for working with higher dimensional objects. These may be thought of as high level, as they are independent of representation, and since they are phrased in terms of cells, may correspond to the way users think about working with real objects.

No matter which approach is taken, there is a basic distinction between creative (and destructive) constructors, which produce and remove objects, and manipulative constructors, which modify objects.

Since we are concerned with topological structure rather than shape, when we talk about a cell, we will actually be talking about a 'cell descriptor' (an abstract entity or placeholder), without worrying about the details of representing geometry.

5.2 Constructors for Implicit-Cell Representations

5.2.1 Introduction

In this section we will consider the constructors given for the quad-edge data structure, the facet-edge data structure, and for n-G-maps. We will describe the constructors in terms of the (combinatorial) representations with which each of these works is concerned, and then describe what effect they have on the topology. The main criteria for these constructors is that they preserve the rules which define the representations, while performing a reasonable action on the topology.

We will apply the same idea to the cell-tuple structure, by asking what changes we can make to it that preserve the properties in Lemma 4.15, while making sense topologically.

While we have loosely categorized these constructors as low level, as compared to the

constructors acting on cells, the really low-level actions are those that operate on 'data structures' and 'pointers'. For instance, in the quad-edge data structure, constructors must be given to create basic elements (quad-edges), to set the basic operators (*Onext*, *Flip* and *Rot*), and to connect geometric information with the topological structure. In the cell-tuple structure, such low level operations might be given as: $make\_ct(k)$ makes a new k-tuple; $kill\_ct(t)$ destroys a cell-tuple; $set\_switch(t^1, k, t^2)$ sets $switch_k(t^1)$ to t^2, and $set\_cell(t, k, c)$ sets the cell descriptor for t_k to c.

5.2.2 Constructors in the Quad-Edge Data Structure

Guibas and Stolfi have cut the number of operators down to what is probably the bare minimum, with one constructor for producing new objects and one for combining existing objects. The constructor *MakeEdge* produces a subdivision of a 2-sphere, consisting of one face, one edge and two vertices (endpoints of the edge), and the constructor *Splice* merges or separates edge-rings. We will present one way of thinking of the geometric effect of *Splice* now.

The *Splice* constructor takes two directed, oriented edges a_1 and a_2 as input, where a_1 and a_2 are both primal or both dual, and $a_1 \neq a_2\, Onext\, Flip$. Let v_1, v_2 be the origins of a_1, a_2, respectively; let e_1, e_2 be the underlying edges of a_1, a_2, respectively; and let f_1, f_2 be the faces to the left of a_1, a_2, respectively. Imagine cutting a slit in the interior of f_1, from some prescribed interior point of f_1 to v_1, and cutting a similar slit in the interior of f_2, from an interior point to v_2. Now the edge of the slit in f_1 which is 'near' a_1 is joined with the slit in f_2 which is 'away from' a_2, and the edge of the slit in f_2 which is 'near' a_2 is joined with the slit in f_1 which is 'away from' a_1. If $v_1 = v_2$ and $f_1 = f_2$, then *Splice* causes no change to the subdivision. Otherwise, if $f_1 = f_2$, the prescribed interior points must coincide, the slits cannot overlap, and the face will be split; and if $v_1 = v_2$, then the vertex will be 'split' into two new vertices, and f_1 and f_2 will be merged into one face. (See Figure 5.1 for an example where $v_1 \neq v_2$, $f_1 \neq f_2$.)

This can be described neatly in terms of the generalized barycentric subdivision. Let

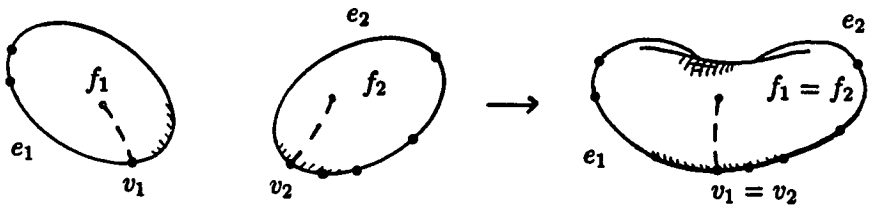


Figure 5.1: Geometric interpretation of *Splice*

v_{v_1}, v_{e_1}, and v_{f_1} be the vertices in the generalized barycentric subdivision which are the 'central' points of v_1, e_1 and f_1 respectively, and let σ_1 be the 2-simplex $\sigma(v_{v_1}, v_{e_1}, v_{f_1})$. Similarly, let v_{v_2}, v_{e_2}, and v_{f_2} be the vertices in the generalized barycentric subdivision which are the central points of v_2, e_2 and f_2 respectively, and let σ_2 be the 2-simplex $\sigma(v_{v_2}, v_{e_2}, v_{f_2})$. The basic idea is that the slit in f_1 is the edge $\sigma(v_{v_1}, v_{f_1})$, and the slit in f_2 is the edge $\sigma(v_{v_2}, v_{f_2})$. Let σ_1' be the 2-simplex which shares edge $\sigma(v_{v_1}, v_{f_1})$ with σ_1, and let σ_2' be the 2-simplex which shares edge $\sigma(v_{v_2}, v_{f_2})$ with σ_2. The result of *Splice*(e_1, e_2) will be to make σ_1 and σ_2' adjacent and σ_1' and σ_2 adjacent.

In terms of the edge algebra, this is phrased as defining a new edge algebra with the same directed, oriented edges and new operations *Onext'*, *Flip'* and *Rot'*. Thus *Splice*$(a, b) : (E, E', Onext, Rot, Flip) \rightarrow (E, E', Onext', Rot', Flip')$ such that

$$a\ Onext' = b\ Onext,$$
$$b\ Onext' = a\ Onext,$$
$$\alpha\ Onext' = \beta\ Onext,$$
$$\beta\ Onext' = \alpha\ Onext,$$
$$(b\ Onext\ Flip)Onext' = a\ Flip,$$
$$(a\ Onext\ Flip)Onext' = b\ Flip,$$
$$(\beta\ Onext\ Flip)Onext' = \alpha\ Flip,$$
$$(\alpha\ Onext\ Flip)Onext' = \beta\ Flip,$$
$$c\ Onext' = c \text{ for all other } c,$$

where $\alpha = a\ Onext\ Rot$ and $\beta = b\ Onext\ Rot$, (with the aforementioned restrictions that a and b are either both primal or both dual, and $b \neq a\ Onext\ Flip$). In this formulation, one can see that the two quad-edge rings about the origin vertices are being merged or broken, and similarly for the two quad-edge rings around the incident faces.

Guibas and Stolfi prove that the *Splice* constructor, when applied in accordance with the stated restrictions, always produces a valid edge algebra, and hence corresponds to a realizable subdivision of a 2-manifold.

It is possible to define a *Splice* constructor on the cell-tuple structure which agrees with the action of the quad-edge *Splice* constructor. Define $Splice(t^1, t^2) : (T, switch) \rightarrow (T, switch')$ so that

$$switch'_1(t^1) = switch_1(t^2),$$
$$switch'_1(t^2) = switch_1(t^1),$$
$$switch'_1(switch_1(t^1)) = t^2,$$
$$switch'_1(switch_1(t^2)) = t^1,$$
$$switch'_1(t) = t \text{ for all other } t,$$
$$switch'_0(t) = t \text{ for all } t,$$
$$switch'_2(t) = t \text{ for all } t,$$

with the restriction that $t^2 \neq switch_1(t^1)$. Note that in the cell-tuple structure there are only half as many modifications to make, because the dual is not represented explicitly.

5.2.3 Constructors in the Facet-Edge Data Structure

Dobkin and Laszlo define four constructors for creating and modifying facet-edge data structures. The constructor *make_facet_edge* produces a new basic object (a facet-edge pair). The constructor *splice_facets* merges or separates facet-rings (the rings of facet-edges associated with a face), and *splice_edges* merges or separates edge-rings (the rings of facet-edges associated with an edge). These are similar to *Splice*, except that the slits are now 2-dimensional separations – the geometric effect can again be understood in terms of the generalized barycentric subdivision. The last constructor, *transfer*, modifies incidence relations directly. There is, however, no guarantee that what is produced by these operations has desirable effects, i.e. the underlying space may not be a manifold (in fact, the basic element produced by *make_facet_edge* does not represent a manifold). The authors state that "little imagination is needed in using *splice_facets* or *splice_edges* to create the most exquisite garbage."

The constructor *splice_facets* phrased in terms of the cell-tuple structure is given by:

$$
\begin{aligned}
&switch_2'(t^1) = switch_2(t^2),\\
&switch_2'(t^2) = switch_2(t^1),\\
&switch_2'(switch_2(t^1)) = t^2,\\
&switch_2'(switch_2(t^2)) = t^1,\\
&switch_2'(switch_0(t^1)) = switch_{02}(t^2),\\
&switch_2'(switch_0(t^2)) = switch_{02}(t^1),\\
&switch_2'(switch_{02}(t^1)) = switch_0(t^2),\\
&switch_2'(switch_{02}(t^2)) = switch_0(t^1),\\
&switch_2'(t) = t \text{ for all other } t,\\
&switch_0'(t) = t \text{ for all } t,\\
&switch_1'(t) = t \text{ for all } t,\\
&switch_3'(t) = t \text{ for all } t.
\end{aligned}
$$

and the constructor *splice_edges* phrased in terms of the cell-tuple structure is given by:

$$
\begin{aligned}
&switch_1'(t^1) = switch_1(t^2),\\
&switch_1'(t^2) = switch_1(t^1),\\
&switch_1'(switch_3(t^1)) = t^2,\\
&switch_1'(switch_3(t^2)) = t^1,\\
&switch_1'(switch_1(t^1)) = switch_{13}(t^2),\\
&switch_1'(switch_1(t^2)) = switch_{13}(t^1),\\
&switch_1'(switch_{13}(t^1)) = switch_1(t^2),\\
&switch_1'(switch_{13}(t^2)) = switch_1(t^1),\\
&switch_2'(t) = t \text{ for all other } t,\\
&switch_0'(t) = t \text{ for all } t,\\
&switch_2'(t) = t \text{ for all } t,\\
&switch_3'(t) = t \text{ for all } t.
\end{aligned}
$$

5.2.4 Constructors for n-G-maps

Four constructors are given for n-G-maps, called Operation 1, Operation 2, Operation 3, and Operation 4. Operation 1 takes a closed $(n-1)$-G-map G and produces an n-G-map which has G as its boundary. Operation 2 identifies two 'isomorphic $(n-1)$-faces' which lie in distinct n-G-maps. Operation 3 identifies two non-adjacent isomorphic $(n-1)$-faces which lie in the same n-G-map. Operation 4 identifies two adjacent isomorphic $(n-1)$-faces which lie in the same n-G-map.

These constructors fit nicely into the framework to be given in Section 5.3.4, so their direct representation in terms of *switch* will not be given here, but will be described in terms of the constructors to be defined in that section. Operation 1 is the constructor that we will call *lift* (though the n-G-map being operated on is not required to be homeomorphic to a sphere). Operations 2 – 4 are specific cases of what we will call the *join* constructor. There are no creative constructors given for n-G-maps to 'get started', though it is easy enough to define the analogue of what we will call *make_vertex*. There

are also no constructors corresponding to *unjoin*, *split*, or *unsplit*.

5.2.5 Possibilities for Constructors in the Cell-Tuple Structure

Since we know that we cannot in general identify the topology of an object from any combinatorial representation of it for $d \geq 3$, we cannot hope to define constructors which are general enough to build arbitrary subdivided manifolds which also ensure that we do not violate the rules which define subdivided manifolds. The best that we can do is to maintain properties (ct1) – (ct5) of Lemma 4.15 (which is enough to maintain the property of being a subdivided manifold when $d = 2$). In applications we may be able to make use of knowledge about the specific objects being manipulated to apply constructors so as to maintain desired topological properties.

We will investigate in this section constructors which maintain properties (ct1) – (ct5) of Lemma 4.15. Before proceeding, we introduce the function $trav(t, I)$, where $I \subset \{0, \ldots, d\}$, which produces a list of all the cell-tuples in $switch_{I^*}(t)$, so that the cell-tuples appear in an order which depends only on the some prescribed ordering of all words in I^*. For instance, a possible implementation would be to do a depth-first search of the subgraph of G_M containing all edges labeled by elements of I, so that the edges leaving a node were always searched in increasing order of their labels. For definiteness, the reader may assume that *trav* is implemented in this way.

Let (M, C) be a subdivided d-manifold, and $(T_M, switch)$ its cell-tuple structure. We are looking for a constructor which modifies *switch*, so that the resulting *switch'* operator satisfies (ct1) – (ct5).

The smallest change that we might make is to change $switch_k$ for a single cell-tuple. Then we must make additional changes to *switch* to satisfy (ct1) – (ct5). Let $t^1, t^2 \in T_M$, $0 \leq k \leq d$, and suppose we want to change $switch_k(t^1)$ so that the result includes $switch'_k(t^1) = t^2$. Let $t^3 = switch_k(t^1)$, and note that we must require that $t^2 \neq t^3$, so as not to violate (ct1). We observe that $switch'_k(t^2) = t^1$ must hold, to satisfy (ct3). Let $t^4 = switch_k(t^2)$. We must also define $switch'_k(t^3)$ and $switch'_k(t^4)$.

The simplest possibility would be to let $switch'_k(t^3) = switch_k(t^4)$ and $switch'_k(t^4) = t^3$. Though this is not enough, it is useful to give this simple operation a name. Define $attach_k(t^1, t^2) : (T_M, switch) \to (T_M, switch')$, where $t^2 \neq switch_k(t^1)$, by

$$\begin{aligned}
&switch'_k(t^1) = t^2,\\
&switch'_k(t^2) = t^1,\\
&switch'_k(switch_k(t^1)) = switch_k(t^2),\\
&switch'_k(switch_k(t^2)) = switch_k(t^1),\\
&switch'_k(t') = switch_k(t') \text{ for all other } t' \in T_M.
\end{aligned}$$

We can write $Splice(t^1, t^2)$ quite succinctly using this operator: $Splice(t^1, t^2) = attach_1(t^1,\ switch_1(t^2))$. Similarly, the two analogous constructors in the facet-edge data structure may be written in terms of $attach_k$: $splice\_facets(t^1, t^2)$ is given by applying $attach_2(t^1, switch_2(t^2))$ and $attach_2(switch_0(t^1), switch_{02}(t^2))$ (in either order), and $splice\_edges(t^1, t^2)$ is given by applying $attach_1(t^1, switch_1(t^2))$ and $attach_1(switch_3(t^1),\ switch_{31}(t^2))$ (in either order).

We must make a few more modifications: $attach_k$ does not necessarily give a new structure which satisfies Lemma 4.15, as (ct4) or (ct5) may be violated. If we impose the constraint that $switch'_\ell = switch_\ell$ for all $\ell \neq k$, then we can determine the minimum set of changes required to satisfy (ct1) – (ct5). (This constraint is justified by simplicity, and it turns out to match up with the constructors we will propose in Section 5.3.4.) It follows from (ct5), in any cell-tuple structure, that if $\ell > k+1$ or $\ell < k-1$, we must have $switch_\ell(switch_k(t)) = switch_k(switch_\ell(t))$. Since we are requiring that $switch'_\ell = switch_\ell$, it follows that $switch'_k(switch_\ell(t)) = switch_\ell(switch'_k(t))$ for all $t \in T_M$. By applying this rule repeatedly, we have the requirement that $switch'_k(switch_w(t)) = switch_w(switch'_k(t))$ for all $w \in I$, where $I = \{0, \ldots, d\} - \{k-1, k, k+1\}$. A natural way to satisfy the requirement is to apply $attach_k(switch_w(t^1), switch_w(t^2))$ for all $w \in I^*$, i.e. to apply $attach_k$ to all of the corresponding pairs in $trav(t^1, I)$ and $trav(t^2, I)$. For this to make sense, the two subgraphs traversed must be isomorphic.

We can define a new constructor $genljoin_k(t^1, t^2) : (T_M, switch) \to (T_M, switch')$,

where $t^1 \neq t^2$. $genljoin_k(t^1, t^2)$ is effected by applying $attach_k(switch_w(t^1), switch_w(t^2))$ for all $w \in I^*$. This maintains rules (ct1), (ct2), (ct3), and (ct5) of Lemma 4.15 by the preceding arguments. Thinking in terms of the graph of the cell-tuple structure, $genljoin_k$ maintains the property that the edges labeled k form a complete matching; taken with (ct1), this ensures that (ct4) is maintained.

Using $genljoin_k$ we can write the quad-edge operation $Splice(t^1, t^2)$ as $genljoin_1(t^1, switch_1(t^2))$, and the facet-edge operations $splice\_facets(t^1, t^2)$ and $splice\_edges(t^1, t^2)$ as $genljoin_2(t^1, switch_2(t^2))$, and as $genljoin_1(t^1, switch_1(t^2))$, respectively.

It is easy to show that if $I = \{0, \ldots, d\} - \{k, k+1\}$, or $I = \{0, \ldots, d\} - \{k-1, k\}$, or $I = \{0, \ldots, d\} - \{k\}$, then applying $attach_k$ to the corresponding pairs in $trav(t^1, I)$ and $trav(t^2, I)$ gives new constructors which satisfy Lemma 4.15.

We will see in the next sections how these constructors can be used to implement a set of operators which correspond to the kinds of changes we would like to make to the geometry of the objects under consideration.

5.3 Constructors for Explicit-Cell Representations

5.3.1 Introduction

The discussion in the following sections will give constructors which act on the basic elements of subdivided manifolds, namely on cells.

The most direct implementation of a subdivision in terms of cells is via an implementation of an incidence graph. In this and the following chapter, we will refer to the node representing a cell c as $node(c)$; if c is a k-cell, we will say that $node(c)$ is a k-node. We will mention here the low level operations needed to maintain an incidence graph. However, the rest of the section will not refer to any specifics of a particular implementation. For the incidence graph, it is necessary to be able to create and destroy nodes, manipulate an incidence relation between nodes, and maintain geometric information attached to nodes. A simple form of this approach might be to provide four constructors:

make_node(k) produces a node representing a k-dimensional cell, for any $0 \leq k \leq d$; *kill_node*(n) destroys a node; *make_incident*(n_1, n_2) makes n_1 and n_2 incident, when the dimensions of n_1 and n_2 differ by one; *kill_incident*(n_1, n_2) makes n_1 and n_2 not incident.

5.3.2 Euler Operators

In the field of constructive solid geometry, which is concerned with building models of 3-dimensional objects, one approach represents objects by maintaining a representation of their boundaries. Several papers ([Bau75], [BEH79], [BHS80], [ELS75], [MS82]) propose or discuss sets of constructors, which they call 'Euler operators', for creating and modifying such 2-dimensional surfaces.

The Euler polyhedron formula ($V - E + F = 2$ for the surfaces of polyhedra, where V is the number of vertices, E is the number of edges, and F is the number of faces) may be proved ([Sti84]) by showing that the following operations do not change the Euler characteristic $V - E + F$:

- splitting an edge into two by a new vertex,
- splitting a face into two by a new edge joining two of the face's vertices,
- the inverses of the operations above.

The papers cited above each give a basic constructor to produce a simple object, and sets of constructors which maintain some invariant similar to the Euler characteristic. As an example, [BEH79] maintains #(faces) − #(edges) + #(vertices) − 2*#(shells) − 2*#(holes) + #(rings). (Rings are annuli, shells are boundaries of 3-cells, holes are holes in 3-cells.) To get started, the operator *make_face_vertex_shell* produces a 2-sphere with a single vertex on it. Then the following are a few of the operators which may be used to combine such basic objects: *make_edge_vertex* produces a new vertex and a new edge which joins the new vertex with an existing vertex; *split_edge* splits an existing edge into two edges by creating a new vertex on the existing edge; and

make_edge_face splits an existing face into two new faces by adding an edge between two of the existing face's vertices. *kill_edge_vertex*, *kill_split_edge*, and *kill_edge_face* are their inverses, respectively. (Note that the operators from Euler's proof are *split_edge*, *kill_split_edge*, *make_edge_face*, and *kill_edge_face*.) There are also more complicated operators for producing and removing rings and holes, etc.

5.3.3 Changing the Dimension of Objects

Before defining a set of constructors, we must decide whether or not to include a means of changing the dimension of an object. Much of the existing work deals always with objects of a fixed dimension, but there are advantages to including constructors which change the dimension.

In the case that only subdivisions of a fixed dimension are dealt with, creative constructors give a simple subdivision of that dimension, and other constructors produce new subdivisions of the same dimension. This makes for a simple, unified framework, which works particularly well in the most-used and best-understood case of two dimensions (e.g. boundaries of solids). A question that must be answered in this case is "What is the right creative constructor to supply?"

Guibas and Stolfi's *MakeEdge* constructor supplies a 2-sphere subdivided into two vertices, one edge, and one face, and the creative constructors in several of the CSG systems provide similar basic objects. Dobkin and Laszlo's *make_facet_edge* supplies a facet-edge, which doesn't correspond to a specific cell or combination of cells. What should we provide in higher dimensions? This might simply be a single cell-tuple, which would agree with *make_facet_edge* in the 3-dimensional case, or a small prescribed cell-tuple structure, as does *MakeEdge* in the 2-dimensional case. In the case where we create a single cell-tuple, the geometric intuition would be to think of a simplex in the generalized barycentric subdivision. In the case where we give some small fixed structure, it's not clear what that structure should be.

The second alternative is to allow subdivisions of various dimensions to exist simul-

taneously. This could still be supported by the above approach, as long as there existed constructors for all dimensions, and the dimension of objects never changed. However, it is useful to be able to change the dimension of objects, and this requires a different set of constructors. In particular, a very natural way to describe an object is by giving its boundary – this is the approach of boundary representation in constructive solid geometry. This suggests that a general method for building higher dimensional objects is to build the boundaries of objects, then 'fill in their interiors'. This may be applied 'recursively', to start with low dimensional objects, and build objects of higher and higher dimension.

We will take this second alternative, allowing the changing of dimensions of objects (when appropriate), because it allows the natural construction technique using boundaries, because it is more general than the first (the first is almost a subset of the second), and because it is not clear what creative constructors should be supplied if the first approach is taken.

5.3.4 Proposed Constructors for d-Dimensional Objects

In this section we will give a small set of constructors which allow the building of any subdivided manifold, and which also fit our intuition of building objects. The constructors we will give are generalizations of basic constructors which appear in low dimensions. This is the main basis for claiming that they are natural.

We will generalize the splitting of edges and faces, which comes from the proof of the Euler polyhedron formula and appear as Euler operators in many constructive solid geometry systems. This gives a means of changing the cellular structure of a subdivision without changing the underlying manifold.

We will give constructors which join and separate objects, allowing the creation of simple objects and combining them into more complicated ones. This can be thought of as the opposite of that in the previous paragraph; loosely speaking, this gives a means of changing the underlying manifold of a subdivision without changing the cellular

structure.

The idea of representing an object by its boundary, one of the main methods in graphics and constructive solid geometry, when generalized to higher dimensions, can be used to build higher dimensional objects by first building their boundaries, and then raising the dimension to 'fill in the interior'.

The above ideas will be translated into constructors, and a basic creative constructor will be given, defined independently of the representation of subdivided manifolds. By the way in which this is done, the basic actions – creating and destroying subdivisions, changing their dimension, changing their cellular structure, and merging and separating them – will be handled more or less orthogonally. The constructors we will introduce are *make_vertex*, *kill_vertex*, *lift*, *unlift*, *join*, *unjoin*, *split*, and *unsplit*.

The basic creative constructor, *make_vertex*, simply produces a vertex. Starting with vertices, we can apply a sequence of the other constructors to produce objects of increasing dimension. *kill_vertex* simply removes an isolated vertex.

The constructor *lift* takes a subdivided (k–1)-manifold (M, C), such that M is homeomorphic to S^{k-1}, and produces a new subdivided k-manifold $(M', C \cup \{c_{\alpha_k}\})$, in which M' is homeomorphic to $\overline{\mathbb{B}}^k$, and the boundary of c_{α_k} is equal to $|C|$. Intuitively, *lift* 'fills in the interior' of C with c_{α_k}. (See Figure 5.2.) As we are keeping track only of the topology, we don't need to have a geometric description of c_{α_k}. For instance, given two distinct vertices, lift will to produce an edge having those vertices as endpoints. The inverse constructor of *lift* will be denoted by *unlift*.

The constructor *join* identifies pairs of cells. (See Figure 5.3.) For this to make sense, if c_{α_k} and $c_{\alpha'_k}$ are the two cells, then c_{α_k} and $c_{\alpha'_k}$, when taken with their boundaries, must be equivalent as subdivided manifolds. Then *join* merges c_{α_k} and $c_{\alpha'_k}$ together, as well as all of the their corresponding subcells. To do this merging, one must decide how to specify the desired 'match-up' between the subcells in the boundaries of the two cells. For example, given two square 2-cells, taken with their boundaries, there are eight ways that the vertices on the boundary might be identified. Each of these allow valid

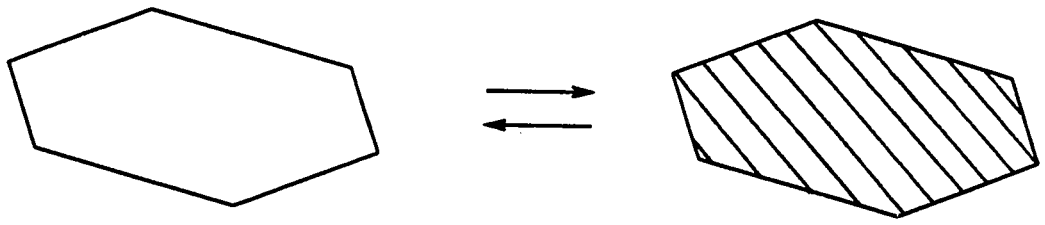


Figure 5.2: *lift* and *unlift*

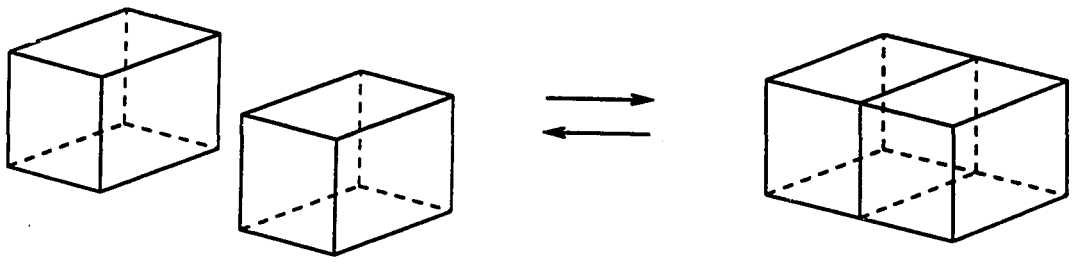


Figure 5.3: *join* and *unjoin*

extensions to homeomorphisms over the whole boundary and the interior of the square, so some scheme must be devised to specify which of these is desired.

One possibility for such a scheme is to give a matching between all of the cells in the boundaries of the two cells being joined. This isn't necessary, however; it is enough to give a matching between the vertices of the two cells. (Note that it is not enough to give a matching between the (k–1)-cells in the boundary of the cells being joined.) Another possibility is to specify for each cell a chain of boundary cells of decreasing dimension i.e. for c_{α_k}, give a chain $c_{\alpha_k} \succ \cdots \succ c_{\alpha_0}$, and for $c_{\alpha'_k}$, give a chain $c_{\alpha'_k} \succ \cdots \succ c_{\alpha'_0}$. Identifying the corresponding cells in these chains completely specifies how all the subcells of c_{α_k} and $c_{\alpha'_k}$ will be matched.

If the *join* constructor is applied to cells in a subdivided d-manifold, these cells must be (d–1)-cells for a valid subdivided manifold to be produced (otherwise Corollary 4.3 will be violated). Another restriction is that the cells being identified must be boundary cells.

The inverse of the *join* constructor is *unjoin*, which takes a non-boundary (d–1)-cell, and separates it into two boundary cells. When this separation is performed, the cells on its boundary may or may not be separated into two cells, as well. If they are boundary cells (before the given (d–1)-cell is separated), then they must be separated; otherwise, they are not.

The constructor *split* is simply a generalization of the Euler operators which split edges and faces. Given a subset of the cells on the boundary of a k-cell c_{α_k}, such that if taken as a subdivided (k–2)-manifold their union is homeomorphic to S^{k-2}, *split* performs a $lift$ on the subset, so as to split c_{α_k} into two new k-cells. (See Figure 5.4.) *split* may be applied for cells of any dimension $1 \leq k \leq d$, if the above criteria are met.

The inverse constructor of *split* will be denoted by *unsplit*. Given a k-cell which is incident to exactly two (k+1)-cells, the k-cell will be eliminated, and the two (k+1)-cells will be merged into one.

Figure 5.5 gives an example of a construction using these operators.

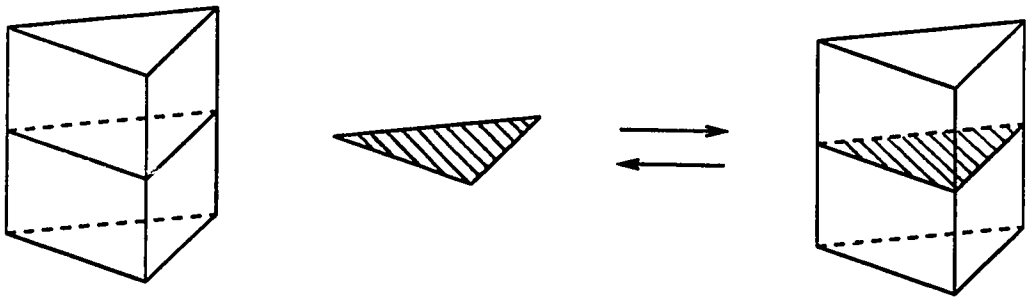


Figure 5.4: *split* and *unsplit*

Note that *join*, *unjoin*, *split*, and *unsplit* are defined independently of *make_vertex*, *kill_vertex*, *lift*, and *unlift*, so if a different scheme for creating basic elements and changing dimensions is desired, or no changing of dimension is allowed, they still remain valid constructors.

It's not hard to show any subdivided d-manifold can be built from this set of constructors. We will give a simple recursive construction using only *make_vertex*, *lift*, and *join*. To build a subdivided d-manifold where $d \geq 1$, first build the boundary of every d-cell in it separately. If $d = 1$, building the boundary consists of two applications of *make_vertex*. If $d \geq 2$, the boundaries are built recursively. Perform a *lift* on every one of these (d–1)-dimensional boundaries, to get the d-cells together with their boundaries. Now for every pair of d-cells which need to be adjacent, do (d–1)-dimensional *joins* on the common pairs of (d–1)-cells in the boundaries.

To give a feeling for the cost of this construction, we will calculate the number of *make_vertex*, *lift*, and *join* operations it requires. To achieve this, we will count the number of k-cells produced during the construction, for $0 \leq k \leq d$. If (M,C) is a subdivided manifold or a subdivided manifold-with-boundary, let $C_k = \{c \in C \mid \dim(c) = k\}$. If $c \in C$, let $\iota_\ell(c)$ be the number of ℓ-cells incident to c. The number of d-cells

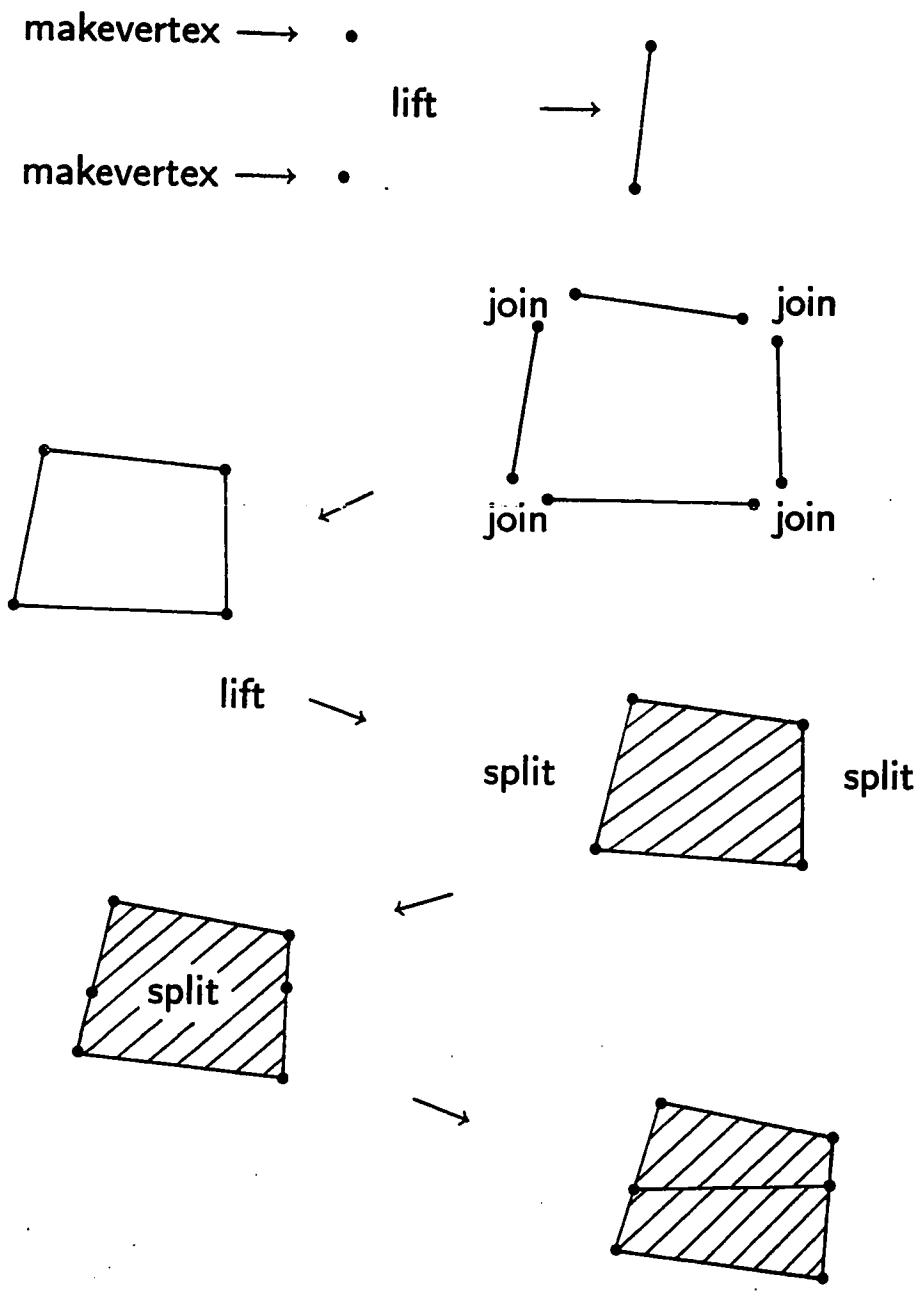


Figure 5.5: Example of use of the proposed constructors

produced in the construction is equal to $\#(C_d)$, the number of d-cells in C. This may be written as $\sum_{c_{\alpha_d} \in C_d} 1$. A copy of every ($d$–1)-cell in C is produced in the construction for every d-cell in whose boundary it appears. The number of (d–1)-cells produced is thus $\sum_{c_{\alpha_d} \in C_d} \imath_{d-1}(c_{\alpha_d}) = \sum_{c_{\alpha_{d-1}} \prec c_{\alpha_d}, c_{\alpha_i} \in C_i} 1$. For $0 \leq k < d$, a copy of every k-cell in C is produced for every (k+1)-cell produced in the construction, in whose boundary the k-cell appears. The number of k-cells produced is thus $\sum_{c_{\alpha_{k+1}} \prec \cdots \prec c_{\alpha_d}, c_{\alpha_i} \in C_i} \imath_k(c_{\alpha_{k+1}}) = \sum_{c_{\alpha_k} \prec \cdots \prec c_{\alpha_d}, c_{\alpha_i} \in C_i} 1$.

If $k = 0$, this gives that the number of vertices produced, and hence the number of applications of *make_vertex*, is $\sum_{c_{\alpha_0} \prec \cdots \prec c_{\alpha_d}, c_{\alpha_i} \in C_i} 1 = \#(T_M)$. The number of applications of *lift* is equal to the number of cells of dimension greater than zero produced in the construction, which is $\sum_{k=1}^{d} \#(k\text{-cells produced}) \leq d \cdot \#(T_M)$. Since each application of *join* reduces the number of cells by at least one, the number of *join*'s performed must be less than or equal to the totals number of cells produced, so is bounded by $\sum_{k=0}^{d} \#(k\text{-cells produced}) \leq d \cdot \#(T_M)$. Thus the total number of constructors required is $O(\#(T_M))$. When implemented using the cell-tuple structure, *make_vertex*, *lift*, and *join* do not destroy cell-tuples, so the number of cell-tuples produced during the construction is equal to $\#(T_M)$. Furthermore, for each cell-tuple $switch_k$ is changed at most once for each $0 \leq k \leq d$. So the number of primitive operations in the cell-tuple structure is also bounded by $O(\#(T_M))$.

This particular construction may not be the most efficient. It may be possible to use the *split* operator to eliminate some of the redundant creation and merging of cells. The above construction is given to show that any subdivided manifold may be built using the proposed constructors.

5.4 Implementing the Proposed Constructors

We will briefly discuss how the proposed constructors in the preceding section may be implemented using the incidence graph and the cell-tuple structure. This will show that these constructors may indeed be implemented in two fairly different contexts, and

will give a means of comparing the merits of the two representations for implementing constructors.

When we say that we are 'implementing' them here, we do not mean that we are giving a program, but rather saying how we might express the proposed constructors in the context of these two representations. An actual implementation, for the cell-tuple structure, will be given in the next chapter.

5.4.1 Using the Incidence Graph

The creative constructor *make_vertex* creates a new node for a vertex: $n = node(c_{\alpha_0})$. This simply requires creating a new 0-node in the incidence graph. $kill\_vertex(n)$ deletes node n.

To say how to implement $lift$, we need to decide how to describe its input. A natural way is to represent the input as a list of (k–1)-nodes. Then to perform $lift$, we produce a new k-node, attach down-pointing arcs from it to the input set, and set the up-pointing arcs of the input set to the new k-node. $unlift(n)$ removes node n and all arcs pointing down from, and up to, n.

The *join* constructor has as input two (d–1)-nodes $node(c_\alpha)$ and $node(c_{\alpha'})$, along with some specification of how they should be brought together. This specification will require supplying two lists of cells, in addition to the cells being joined. Then the nodes for all of the corresponding subcells of the two input cells must be merged together. This requires an algorithm for matching corresponding subcells, i.e. traversing the two incidence graphs in the same manner. This may be complicated.

To implement *unjoin* we need only specify the (d–1)-node $node(c_\alpha)$ to be separated. Using the down-pointing arcs for $node(c_\alpha)$, a list of all incident (d–2)-nodes is created, and those which are boundary nodes are also separated. This continues for decreasing dimension, down to zero. Separation of this set of cells means making two new nodes and discarding the existing node, and to each cell being separated, copying the incidence pointers to each of the new sets.

To implement the *split* constructor, it is necessary to specify a k-cell c_{α_k} and a list ℓ of (k–2)-cells. The node for c_{α_k} is replaced by two new k-nodes $node(c_{\alpha'_k})$ and $node(c_{\alpha''_k})$, so that they each get a copy of $node(c_{\alpha_k})$'s up-pointing list, and each of the nodes which pointed down at $node(c_{\alpha_k})$ now point down at $node(c_{\alpha'_k})$ and $node(c_{\alpha''_k})$. A $lift$ is done on the set of nodes for cells in ℓ, and the node for the new (k–1)-cell which that $lift$ produces is made incident to $node(c_{\alpha'_k})$ and $node(c_{\alpha''_k})$. Now comes the hard part: the down-pointing lists for $c_{\alpha'_k}$ and $c_{\alpha''_k}$ must be produced. These two lists form a partition of c_{α_k}'s down-pointing list, so it is straightforward to obtain the set of (k–2)-nodes which make up the two new down-pointing lists. The problem is deciding which of the two down-pointing lists each of these nodes belongs to. This requires a search of the boundary of the cell being split, which is rather complicated when expressed in terms of the incidence graph. The up-pointing lists for these (d–2)-cells must also be set properly, which can be done easily after the down-pointing arcs are set.

unsplit only requires specifying a single (k–1)-cell. This node is removed, and the nodes for the two k-cells which were incident to it are merged. This merge includes merging their down-pointing lists and their up-pointing lists, and updating the lists of the nodes that these pointed to.

There are two parts of this implementation which are awkward. The first is the specification of the way to 'match up' two cells when doing a join, and the second is setting of the down-pointing arcs for the two new cells created by a split. We will see in the next section that the cell-tuple structure handles these simply.

5.4.2 Using the Cell-Tuple Structure

We will describe how to implement the proposed constructors using the cell-tuple structure. We will make use of the routine $trav(t, I)$ introduced in Section 5.2.5. We will also have to make use of a subcomplex to define *split*. A subcomplex will be specified by giving a 'child' cell-tuple structure in which each cell-tuple has a pointer to the corresponding cell-tuple in its 'parent' cell-tuple structure.

We will use the notation $\dim(t)$ to mean that cell-tuple t is 'related to' a manifold of that dimension, i.e. has $\dim(t)+1$ components.

The basic creative constructor *make_vertex* takes a vertex as its input argument. $make\_vertex(c_{\alpha_0})$ allocates a 0-dimensional cell-tuple t, with $t_0 = c_{\alpha_0}$ and $switch_0(t)$ set to null. $kill\_vertex(t)$ deletes t.

The dimension-increasing constructor $lift$ takes as inputs a cell-tuple t and a cell c (supplied by the particular application), where $\dim(t) = k-1$ and $\dim(c) = k$. To enact $lift(t,c)$, do the following for all $t' \in trav(t, \{0,\ldots,k-1\})$: increase the dimension of t' by one (to k), set $switch_k(t')$ to *bd*, (the special value indicating the boundary), and set t_k to c. The dimension-decreasing constructor $unlift$ takes as input a cell-tuple t, where $\dim(t) = k$. To enact $unlift(t)$, decrease the dimension of t' by one (to $k-1$) for all $t' \in trav(t, \{0,\ldots,k-1\})$, i. e. eliminate the d-dimensional component of each t'.

The constructor *join* takes as input two cell-tuples t^1 and t^2, whose dimensions are equal to k. Its effect is to identify t^1_{k-1} and t^2_{k-1}. To execute $join(t^1, t^2)$, let list $\ell_1 = trav(t^1, \{0,\ldots,k-2\})$, and list $\ell_2 = trav(t^2, \{0,\ldots,k-2\})$, and suppose that the lengths of these lists is n. Then join is accomplished by doing $attach_k(\ell_1[i], \ell_2[i])$ for $1 \le i \le n$.

The constructor *unjoin* takes as input a cell-tuple t whose dimension is equal to k. Its effect is to separate t_{k-1} into two new cells. To execute $unjoin(t)$, let list $\ell_1 = trav(t, \{0,\ldots,k-2\})$, and list $\ell_2 = trav(switch_k(t), \{0,\ldots,k-2\})$. Then *unjoin* is accomplished by setting $switch_k(t')$ to *bd* for all $t' \in \ell_1 \cup \ell_2$.

The constructor *split* takes as input two cell-tuples t^1 and t^2, where $\dim(t^1) = k$ and $\dim(t^2) = k-2$. The cell to be split is t^1_k, and the boundary of the cell which will do the splitting is a (k–2)-dimensional subcomplex of which t^2 is a representative. It must be the case that t^1 and t^2 agree on their components from 0 to $k-2$, inclusive, and furthermore, auxiliary pointers must exist from each cell-tuple of the subcomplex to the corresponding cell-tuple in the parent complex. The only exception is when $k = 1$. In this case, $\dim(t^2) = 0$ and t^2_0 gives the vertex to do the splitting.

To execute $split(t^1, t^2)$ for $k > 1$, first do a lift of t^2 to create the splitting cell. Before attaching the splitting cell, since the final result is a k-dimensional object, we must make a copy of the cell-tuple structure associated with this $(k-1)$-cell, in order to have a cell-tuple structure for each 'side' of the cell. This is done by making a copy of all of the cell-tuples associated with the cell created by *lift*, and attaching the appropriate *switch*'s in the copy to match the original. The two copies are then attached to each other by $switch_k$ (this operation will be called *double*). This is what the splitting cell must look like in the resulting structure. Now this doubled cell must be attached to the cell being split. Let list $\ell = trav(t^2, \{0, \ldots, k-2\})$. For all $t \in \ell$, let t' be the associated cell-tuple in the parent cell-tuple structure. Connect t and t' by $switch_{k-1}$, and connect $switch_k(t)$ and $switch_k(t')$ by $switch_{k-1}$. This completes the *split* constructor for $k > 1$.

To execute *split* when $k = 1$, there is only one possible way to *split* the edge t_1^1, so no subcomplex need be given. If $e = t_1^1$ is the edge to be split, and v and v' are e's endpoints, then copies of $assoc(v, e)$ and $assoc(v', e)$ are made and connected to each other via $switch_1$. This will be the associated set of the splitting vertex. Each of the copied cell-tuples are now also connected to their progenitors by $switch_0$, to complete the split.

The input to *unsplit* is a single cell, given by specifying a cell-tuple t and a dimension k, meaning that the cell t_k is to be removed. To remove this cell means traversing its boundary and disconnecting all *switch*'s attaching it to the rest of the structure. Let list $\ell = trav(t, \{0, \ldots, k-1, k+1, \ldots, d\})$, and for every $t' \in \ell$, connect $switch_{k-1}(t')$ and $switch_{k,k-1}(t)$ by $switch_{k-1}$. This will eliminate the desired cell. The operation can be made an actual inverse of *split* without much more work.

When it comes to actually coding the proposed constructors using the cell-tuple structure, three special operators will be very useful. *copy* makes a copy of a subcomplex, *subcx* makes a copy of a subcomplex in which each new cell-tuple has an auxiliary pointer to the cell-tuple it was copied from (its parent), and *double* makes a new copy of a k-dimensional cell-tuple structure, and connects the corresponding cell-tuples in the

original and in the copy by $switch_{k+1}$. These operations don't, in general, preserve all topological properties, but are useful as intermediate operations for implementing the proposed constructors, as well as other operations.

5.5 Maintaining the Connection Between Topology and Geometry

In two dimensions, the topological constructors may be done in constant time. In higher dimensions, the topological constructors may be done in time proportional to the complexity of the cells acted upon. However, such constructors may affect the geometry of higher dimensional cells, which are of greater complexity. For example, when doing a *split* of a 3-cell by a 2-cell, the work to update the topology is proportional to the number of edges of the 2-cell, but the geometry of the 3-cell is completely changed. To update the geometry means forming new cell descriptors for the new 3-cells created by the *split*, and changing every cell-tuple associated with the original 3-cell so that their 3-dimensional components point at the correct new cell descriptors.

In the basic method, in which every cell-tuple points to all of its components, access to a cell descriptor takes constant time, but to change the geometry of a cell means traversing all of its associated cell-tuples. An alternative is to think of the cell-tuples as abstract entities, not having every cell-tuple point to all of its components, but for each cell to have a single cell-tuple which points to it. When a cell-tuple wishes to get hold of one of its components, it may have to traverse the entire associated set of that cell to find the cell-tuple pointing to that component. To update the geometry means only changing the one pointer, however. This method requires less space than the first method, but in general takes time proportional to the complexity of the cell to answer queries about cells.

It is possible that a hybrid between the two extremes just described would provide a good compromise between space conservation, quick queries about cells, and reasonably

quick updates of the required pointers to cells. This would mean having a subset of the cell-tuples associated with a cell actually pointing to it.

The reverse part of this problem is making topological queries, given a particular cell. A natural solution is to have a pointer from a cell descriptor back into the cell-tuple structure. This is enough to answer any topological query, but more information could be kept in the cell descriptor to improve efficiency or convenience.

Maintaining the connection between topology and geometry, allowing access and easy updates of topology and geometry simultaneously, while also allowing changes to be made to either independently, is one of the most challenging problems for an implementation.

Finding an elegant method of simultaneously representing topological and geometric data is a possible area for future research.

Chapter 6

Implementation

6.1 Introduction

For any representation or model to be useful in computer science, it must serve as a bridge between our notion of the entities being represented, and what can be reasonably implemented on a machine.

Up to this point we have discussed the cell-tuple structure only as an abstract representation. As with any theoretical idea, it is important to consider implementation issues, and if possible build an implementation, not only to evaluate the usefulness of the idea, but because unexpected problems and new insights are often discovered during the implementation process.

In this chapter we will consider various ways that the abstract representation may be fit into standard paradigms used in computer science by discussing several possible methods of implementation. This will also include a comparison of size requirements and access speed.

We will also describe a program which implements the cell-tuple structure and the set of constructors proposed in the preceding chapter. Related issues such as the handling of geometric information and the display of output will be discussed, and the implementation of a few simple algorithms will be described.

6.2 Methods of Implementation

For an implementation to be useful, it must allow quick access to structural information. This means being able to answer queries about incidence and ordering. Incidence queries are typically of the form "is c_1 incident to c_2" or "return all cells of dimension ℓ which are incident to c." An ordering query may be of the form "return a circular ordering of all the cells between c_1 and c_2" (under appropriate conditions), or "return $switch(c_{\alpha_{k-1}}, c_{\alpha_k}, c_{\alpha_{k+1}})$. It is also important that the implementation be able to support constructors; this will be discussed later in this chapter.

The important issues of size and speed will be considered for each of the implementation methods discussed.

For the purposes of this section, we will not be concerned with the geometric structure of cells; we will assume that each cell has some sort of representation in memory, which we will refer to as a 'cell descriptor'.

6.2.1 The Database Approach

Given the set of cell-tuples for a subdivided d-manifold, the $switch_k$ operators may be deduced from this set, by Lemma 4.2. Thus the cell-tuple structure may be represented just by the set of cell-tuples. A natural paradigm for representing a set of tuples is a database.

Suppose the database management program can answer the following type of query: $Q(x_1, \ldots, x_d)$, where x_i is either equal to '$*$' or is a k-cell. The result returned is the set of all tuples which agree with the non-$*$ components. Thus, if $c_1 < \cdots < c_\ell$, we can obtain $assoc(c_1, \ldots, c_\ell)$ by asking for $Q(*, \ldots, *, c_1, *, \ldots, *, c_\ell, *, \ldots, *)$. For example, $assoc(c_{\alpha_k})$ may be obtained by asking for $Q(*, \ldots, *, c_{\alpha_k}, *, \ldots, *)$. Furthermore, if $t = (c_{\alpha_0}, \ldots, c_{\alpha_d})$, this allows us to obtain $switch_k(t)$ by computing $Q(c_{\alpha_0}, \ldots, c_{\alpha_{k-1}}, *, c_{\alpha_{k+1}}, \ldots, c_{\alpha_d}) - \{t\}$.

If the set of tuples is kept as a list with no additional structure, such queries would generally take time linear in the number of cell-tuples. It is beyond the scope of this

dissertation to analyze the time requirements of more complicated database models.

6.2.2 The Pointer Approach

The most direct way to implement the cell-tuple structure is to allocate a record for every cell-tuple, which contains an array of pointers to cell descriptors (representing the components of the cell-tuple) and an array of pointers to other cell-tuple records (representing the *switch* operators). Sequences of *switch* operations can be obtained by pointer chasing. As cells are represented implicitly by the cell-tuple structure, it is not absolutely necessary to keep pointers to cell descriptors in every cell-tuple record (this was discussed in Section 5.5).

6.2.3 The Graph Approach

Since the graph for the cell-tuple structure contains all of the *switch* information, it would be possible to use any data structure which is capable of representing edge-labeled graphs. This might be particularly useful if the user has access to a set of general graph routines. Applying the $switch_k$ operator corresponds to following an edge labeled k. If implemented as a directed graph, the result is equivalent to the pointer based approach. In fact, the graph approach is so similar to the pointer based approach that we won't consider them separately after this. The edges of a graph may be followed as if they were pointers, and the pointer-based method can be thought of as a specific implementation of the graph method.

6.2.4 Alternatives

As we will see in the next section, the number of cell-tuples goes up quickly with dimension. Since the $switch_k(t)$ depends only on t_{k-1}, t_k, t_{k+1}, there is a lot of redundancy in storing the *switch* operators – $switch(t_{k-1}, t_k, t_{k+1})$ is stored separately in all cell-tuples in $assoc(t_{k-1}, t_k, t_{k+1})$.

An alternative to the database and pointer methods would be to make triples of

the form $(c_{\alpha_{k-1}}, c_{\alpha_k}, c_{\alpha_{k+1}})$ the basic data elements in an implementation. To each such triple $(c_{\alpha_{k-1}}, c_{\alpha_k}, c_{\alpha_{k+1}})$ would be attached a single pointer, pointing to the triple $(c_{\alpha_{k-1}}, c_{\alpha'_k}, c_{\alpha_{k+1}})$, where $c_{\alpha'_k} = switch(c_{\alpha_{k-1}}, c_{\alpha_k}, c_{\alpha_{k+1}})$. Thus all of the *switch* operations are represented. Two cells whose dimension differs by one are incident if and only if they appear as consecutive components of some triple, i.e. the arcs of the incidence graph are contained in the triples, so incidence information is represented directly. The work described in this thesis has been implemented, using a similar variant of the database method, as part of the most recent version of Bolt, Baranak and Newman's distributed tank simulator ([Smy89]).

Another, somewhat similar approach uses the idea of attaching the *switch* operators to triples, but uses the incidence graph as the framework around which the implementation is built. Here the standard incidence graph is maintained, with *switch* operators attached to either the nodes or the arcs of the incidence graph. In the case of attaching to nodes, each node has a list of pointers, one for each triple in which the node's cell is the middle component. Thus if the node represents cell c_{α_k}, then for every $c_{\alpha_{k-1}} \prec c_{\alpha_k}$ and every $c_{\alpha_k} \prec c_{\alpha_{k+1}}$, there is a pointer for the pair $(c_{\alpha_{k-1}}, c_{\alpha_{k+1}})$ which points to the node representing $switch(c_{\alpha_{k-1}}, c_{\alpha_k}, c_{\alpha_{k+1}})$. In the case of attaching to edges, consider the edge connecting nodes corresponding to $(c_{\alpha_{k-1}}, c_{\alpha_k})$. Then that edge will have a list of pointers, one for every cell $c_{\alpha_{k+1}}$ such that $c_{\alpha_k} \prec c_{\alpha_{k+1}}$, pointing to the node representing $switch(c_{\alpha_{k-1}}, c_{\alpha_k}, c_{\alpha_{k+1}})$. This approach (without specifying whether attaching to nodes or edges) will be referred to as the **augmented incidence graph.**

6.2.5 Three Useful Examples

In this section we will describe the generalization of three familiar objects to the general case of d-dimensions. These will be used throughout the remainder of this chapter.

The unit d-sphere is the set of all points in $\mathbb{R}^{d+1}$ whose distance from the origin is one. We will describe the minimal subdivision of the d-sphere. For each $0 \leq k \leq d$, there are two k-cells. The two 0-cells form a 0-sphere. The two 1-cells have as their

common endpoints the two 0-cells, forming a 1-sphere. The two 2-cells have this 1-sphere as their common boundary. And so on. (To be explicit define the two k-cells, $0 \leq k \leq d$, by $c_{k,1} = \{(x_1, \ldots, x_d) \mid x_1 < 0, x_{k+1} \neq 0, x_{k+2} = x_{k+3} = \cdots = x_{d+1} = 0\}$ and $c_{k,2} = \{(x_1, \ldots, x_d) \mid x_1 > 0, x_{k+1} \neq 0, x_{k+2} = x_{k+3} = \cdots = x_{d+1} = 0\}$.) We call this minimal, because if there were any fewer cells in the subdivision, there would have to be a cell whose boundary self-intersected. This is a subdivided d-manifold (without boundary).

The second object is the d-simplex defined in Section 2.5. If the interior of each face of a d-simplex is taken as a cell, and that d-simplex is the underlying manifold, then this is a subdivided d-manifold-with-boundary.

The third object is the d-dimensional hypercube. This may be defined as the set of all points $(x_1, \ldots, x_d)$ in $\mathbb{R}^d$ such that $0 \leq x_i \leq 1$ for $i = 1, \ldots, d$. To make this into a subdivided d-manifold, we must partition it into cells. If $0 \leq k \leq d$, every k-cell may be constructed as follows. Choose any set $I = \{i_1, \ldots, i_k\} \subset \{0, \ldots, d\}$ (if $k = 0$, I is empty). For $i \notin I$, fix x_i to be equal to 0 or 1, and for $i \in I$, let x_i take on all values in the interval $(0, 1)$. Then the set of all such $(x_1, \ldots, x_d)$ forms the k-cell. This is a subdivided d-manifold-with-boundary.

When the three examples above are used in discussions of the cell-tuple structure, only cell-tuples corresponding to 'interior' cells will be considered (i.e. for the last two examples, which have boundaries, the first approach to handling the boundary will be used). This is done only to make the discussions simpler.

6.2.6 Size

In this section we will investigate the size of some of the implementations of the cell-tuple structure. By size we mean the number of basic data objects involved. We will give rough bounds on the number of cell-tuples in the cell-tuple structure. We will also calculate the number of edges in the incidence graph, the number of triples in the augmented incidence graph, and the number of cell-tuples in the cell-tuple structure for the three

Table 6.1: Values of ι_{k-1}, ι_{k+1}, $\#(C_k)$ and $\#(C)$ for the three examples

	$\iota_{k-1}(k)$	$\iota_{k+1}(k)$	$\#(C_k)$	$\#(C)$
d-sphere	2	2	2	$2d+2$
d-simplex	$k+1$	$d-k$	$\binom{d+1}{k+1}$	$2^{d+1}-1$
d-hypercube	$2k$	$d-k$	$2^{d-k}\binom{d}{k}$	3^d

objects described in the preceding section.

In the case of incidence graphs, we will include in our count edges incident to the non-existent cells c_{-1} and c_{d+1}. Similarly, in the case of triples, we will include in our count triples whose middle components have dimensions 0 and d. The count of the number of cell-tuples is unchanged whether these extra cells are included or not.

If (M, C) is a subdivided d-manifold, let $\iota_\ell(c)$ be the number of ℓ-cells incident to a cell c. A rough upper bound on the number of cell-tuples in T_M is given by $\#(C_d) \cdot \prod_{0 \le k < d} \max_{c \in C_{k+1}} \#(\iota_k(c))$. Since every k-cell is incident to at least two $(k-1)$-cells, a strict lower bound is given by $2^d \cdot \#(C_d)$. This lower bound is be achieved by the minimal subdivision of the d-sphere.

Suppose that for each $0 \le k \le d$, every k-cell is incident to the same number $\iota_{k-1}(k)$ of $(k-1)$-cells and to the same number $\iota_{k+1}(k)$ of $(k+1)$-cells (define $\iota_{-1}(0) = 1$, $\iota_0(-1) = \#(C_0)$, $\iota_{d+1}(d) = 1$, $\iota_d(d+1) = \#(C_d)$).

Then the number of edges in the incidence graph will be $\sum_{k=0}^{d+1} \#(C_k)\iota_{k-1}(k) = \sum_{k=-1}^{d} \#(C_k)\iota_{k+1}(k)$, the number of triples in the augmented incidence graph will be $\sum_{k=0}^{d} \#(C_k)\iota_{k-1}(k) \cdot \iota_{k+1}(k)$, and the number of cell-tuples will be $\#(C_d) \cdot \prod_{0<k\le d} \iota_{k-1}(k)$.

We will give the formulas for these numbers for the three objects described in the previous section. Table 6.1 gives values of $\iota_{k-1}(k)$, $\iota_{k+1}(k)$, $\#(C_k)$, and $\#(C)$. Table 6.2

Table 6.2: Number of incidence graph edges, triples and cell-tuples for the three examples

	incidence graph arcs	triples	cell-tuples
d-sphere	$4d+4$	$8d$	2^{d+1}
d-simplex	$(d+1)2^d+1$	$(d+1)(d2^{d-1}+1)$	$(d+1)!$
d-hypercube	$2\cdot 3^{d-1}d+2^d+1$	$4d(d-1)3^{d-2}+2^d d+2d$	$2^d d!$

gives the number of edges in the incidence graph, the number of triples in the augmented incident graph, and the number of cell-tuples.

Just for clarity, a few of the calculations for values in the Tables 6.1 and 6.2 will be given below.

For the d-simplex, the number of edges in the incidence graph is

$$\sum_{k=0}^{d+1} \#(C_k)\imath_{k-1}(k) = 1 + \sum_{k=0}^{d} \binom{d+1}{k+1}(k+1) = (d+1)2^d + 1;$$

and the number of triples in the augmented incidence graph is

$$(d+1)+\sum_{k=0}^{d-1}\binom{d+1}{k+1}(k+1)(d-k) = (d+1)+(d+1)\cdot d\cdot\sum_{k=0}^{d-1}\binom{d-1}{k} = (d+1)(d2^{d-1}+1).$$

For the d-hypercube, the number of edges in the incidence graph is

$$\sum_{k=-1}^{d} \#(C_k)\imath_{k+1}(k) = 1 + 2^d + \sum_{k=0}^{d-1} 2^{d-k}\binom{d}{k}(d-k)$$

$$= 2^d + 2d\sum_{k=0}^{d-1}\binom{d-1}{k}2^{d-k-1} = 1 + 2^d + 2d3^{d-1};$$

and for $d \geq 2$, the number of triples is

$$2^d d + \sum_{k=1}^{d-1} 2^{d-k}\binom{d}{k}2k(d-k) \; + 2d = 2^d d + d(d-1)\sum_{k=1}^{d-1} 2^{d-k+1}\binom{d-2}{k-1} \; + 2d$$

Table 6.3: Minimal d-sphere: some values for small d

d	cells	edges	#triples	ct's	edges/cell	triples/cell	ct's/cell
0	2	0	2	2	0.00	1.00	1.00
1	4	4	8	4	1.00	2.00	1.00
2	6	8	16	8	1.33	2.67	1.33
3	8	12	24	16	1.50	3.00	2.00
4	10	16	32	32	1.60	3.20	3.20
5	12	20	40	64	1.67	3.33	5.33
6	14	24	48	128	1.71	3.43	9.14
7	16	28	56	256	1.75	3.50	16.00
8	18	32	64	512	1.78	3.56	28.44
9	20	36	72	1024	1.80	3.60	51.20
10	22	40	80	2048	1.82	3.64	93.09

$$= 2^d d + d(d-1)\sum_{k=0}^{d-2}\binom{d-2}{k}2^{d-k} + 2d = 2^d d + 4d(d-1)\sum_{k=0}^{d-2}\binom{d-2}{k}2^{d-2-k} + 2d$$

$$= 2^d d + 4d(d-1)3^{d-2} + 2d.$$

Tables 6.3, 6.4, and 6.5 give some examples of how the sizes of these objects grow with d. It is obvious from the formulas and numbers that the number of cell-tuples grows very quickly with dimension. Thus in practice, if even a modest number of dimensions is to be represented, one of the alternative implementations will have to be used.

Another observation is that for two and three dimensions, the size of the cell-tuple structure is competitive with that of the other two methods. This suggests that the cell-tuple structure is more than just a specialized structure to be used for problems where higher dimensional objects are required, but is in fact a reasonable structure (in terms of space and power) to use in the more common case of low dimensions as well.

6.2.7 Speed

For an application to be effective, its implementation must meet certain performance demands. In this section we will discuss the time required to perform some basic operations on subdivided manifolds.

Table 6.4: d-dimensional simplex: some values for small d

d	cells	edges	#triples	ct's	edges/cell	triples/cell	ct's/cell
0	1	0	1	1	0.00	1.00	1.00
1	3	2	4	2	0.67	1.33	0.67
2	7	9	15	6	1.29	2.14	0.86
3	15	28	52	24	1.87	3.47	1.60
4	31	75	165	120	2.42	5.32	3.87
5	63	186	486	720	2.95	7.71	11.43
6	127	441	1351	5040	3.47	10.64	39.69
7	255	1016	3592	40320	3.98	14.09	158.12
8	511	2295	9225	362880	4.49	18.05	710.14
9	1023	5110	23050	3628800	5.00	22.53	3547.21
10	2047	11253	56331	39916800	5.50	27.52	19500.15

Table 6.5: d-dimensional hypercube: some values for small d

d	cells	edges	#triples	ct's	edges/cell	triples/cell	ct's/cell
0	1	1	1	1	1.00	1.00	1.00
1	3	4	4	2	1.33	1.33	0.67
2	9	16	20	8	1.78	2.22	0.89
3	27	62	102	48	2.30	3.78	1.78
4	81	232	504	384	2.86	6.22	4.74
5	243	842	2330	3840	3.47	9.59	15.80
6	729	2980	10116	46080	4.09	13.88	63.21
7	2187	10334	41734	645120	4.73	19.08	294.98
8	6561	35248	165360	10321920	5.37	25.20	1573.22
9	19683	118610	634482	185794560	6.03	32.24	9439.34

The operations of interest are queries and constructors. All of the queries, and the required operations underlying the proposed constructors, may be reduced to those in the following list.

- determine whether or not two cells are incident,
- obtain the boundary of a cell (as a set of cells or cell-tuples),
- obtain the set of cells of a given dimension which are incident to a cell (as a set of cells or cell-tuples),
- obtain *switch* (*switch*$_k$ of a cell-tuple, or *switch* of a tripie),
- obtain a circular ordering of the cells between $c_{\alpha_{k-2}} \prec c_{\alpha_{k+1}}$.

We will consider the question of how long each of these take for the incidence graph, the cell-tuple structure, and the augmented incidence graph. We will discuss each of the items above in turn for the incidence graph and the cell-tuple structure. The augmented incidence graph, being a hybrid of the two methods just considered, may be more easily considered after discussing the other two. (We will not consider the database model in any detail. If a database is simply a list of cell-tuples, each of the tasks above take time at least proportional to the number of cell-tuples. Any optimization of the database model will probably involve an adaptation of one of the other methods.)

Before beginning the discussion, we will mention that in the cell-tuple structure, implemented by pointers, if $I = \{i_0, \ldots, i_\ell\}$ and t is a cell-tuple such that $t_{i_j} = c_{\alpha_{i_j}}$ for $i_j \in I$, then $assoc(c_{\alpha_{i_0}}, \ldots, c_{\alpha_{i_\ell}}) = switch_{I^*}(t)$ may be produced in linear time by doing a depth-first search of the subgraph of G_M which contains t and whose edges' labels are in I.

Consider the question "is c_{α_k} incident to c_{α_ℓ}?" for the incidence graph. In the case that $k = \ell$, one need only determine if $c_{\alpha_k} = c_{\alpha_\ell}$ or not. If the dimensions differ by one, say $\ell = k + 1$, a search through the up-pointing list of $node(c_{\alpha_k})$ or the down-pointing list of $node(c_{\alpha_\ell})$ is required. In the worst case the entire list will be traversed. Thus the

worst case time is proportional to the up- or down-degree of the incidence graph. If the dimensions differ by more than one, say $\ell > k+1$, then a cascading sequence of searches down (or up) is required. In the worst case, all downward paths of length $\ell - k$ starting at $node(c_{\alpha_\ell})$ may have to be explored.

To ask the same question in the cell-tuple structure, we can assume that we have cell-tuples t and t' such that $t_k = c_{\alpha_k}$ and $t'_\ell = c_{\alpha_\ell}$. The question is equivalent to asking if there exists a cell-tuple t'' such that $t''_k = c_{\alpha_k}$ and $t''_\ell = c_{\alpha_\ell}$. This requires traversing $assoc(c_{\alpha_k})$ (or $assoc(c_{\alpha_\ell})$); that is, all cell-tuples associated with c_{α_k} are checked. The number of cell-tuples checked is at least as large as the number of paths explored in the incidence graph search, but producing $assoc(c_{\alpha_k})$ is much simpler.

Obtaining the boundary of a cell in the case of the incidence graph is a vague question – does this mean all cells in the boundary, or only cells of some dimension? We will assume that finding the set of incident cells of one lower dimension is a satisfactory answer, as this turned out to be enough in the applications that were implemented, and if all cells are required, then the same query may be applied recursively. Thus obtaining the boundary reduces to the next question on the list, which is treated in the next paragraph. Obtaining the boundary of a cell c_α in the case of the cell-tuple structure means finding all of the cell-tuples associated with the cell, which is a traversal of $assoc(c_\alpha)$.

To obtain the set of cells of a given dimension which are incident to a cell in the context of the incidence graph, the same search is required as that described for determining whether or not two cells are incident. The time required is equal to the worst case time for that search. In the case of the cell-tuple structure, the question requires returning a set of cell-tuples which are distinct representatives of the set of desired cells. This requires doing a traversal of the associated set of the cell, hence takes time proportional to the size of that set.

To obtain $switch_k$ in the cell-tuple structure takes constant time. In the case of the incidence graph, the question must be phrased as finding $switch(c_{\alpha_{k-1}}, c_{\alpha_k}, c_{\alpha_{k+1}})$, where $c_{\alpha_{k-1}} \prec c_{\alpha_k} \prec c_{\alpha_{k+1}}$. This requires finding the common elements of the down-pointing list

of $c_{\alpha_{k+1}}$ and the up-pointing list of $c_{\alpha_{k-1}}$, so is proportional to the larger of the up- and down-degrees of the incidence graph.

To obtain a circular ordering in the cell-tuple structure requires applying an alternating sequence of $switch_{k-1}$ and $switch_k$ operations. This takes time proportional to the number of cells in the ordering. To obtain the ordering in the incidence graph requires finding the set of cells involved and then doing something similar to sorting, hence takes an extra log factor.

In the augmented incidence graph, incidence queries can of course be answered in the same time as for the basic incidence graph. How quickly *switch* may be obtained depends on how quickly the arrays associated with each triple may be accessed. If some sort of hashing scheme is used, this could be constant expected time. In this case, a circular ordering could be obtained in time proportional to the number of cells involved in the ordering.

6.3 A Program

6.3.1 Introduction

It was decided to build a program using the cell-tuple structure, to observe how well it would work in practice, and to (hopefully) gain new insights into the theory and discover problems. Thus the main objectives were to make the implementation correspond to the abstract model, be flexible and easy to use, and provide tools for observing algorithms. By the same token, performance was not a major consideration.

There are a number of choices to be made in any implementation. We list a few that occurred in the context of this program:

- which method of implementation of the cell-tuple structure,
- what set of constructors,
- what geometric objects and operations on them,

- what interface layer, if any,
- how to display and output results,
- software issues, e.g. what machine, what language,
- which algorithms to implement.

The cell-tuple structure was implemented using the pointer-based method, with each cell-tuple containing an array of pointers to cell descriptors, as this corresponds most closely with its definition. The set of proposed constructors described in Section 5.3.4 was implemented using some basic operations on the cell-tuple structure.

The possibilities for geometric objects and operations are many and varied. We chose to implement 'flat' geometric objects in the general higher dimensional case, for their simplicity and because they are the entities used by such classic objects in computational geometry as convex hulls, Voronoi diagrams, and arrangements.

Because it is natural to think in terms of cells, it was decided to provide an entity corresponding to a cell. This is a pair consisting of a cell-tuple and an integer – this represents a cell (the component of the cell-tuple specified by the integer), and also represents 'location' within the cell. These pairs turn out to be very convenient when writing algorithms. The proposed constructors were implemented to operate on these pairs. To simplify the discussion here, we will not describe the implementation of these pairs, but will describe everything directly in terms of the cell-tuple structure. When reference is made to these pairs, they will be called 'ct-pairs'.

One of the main ideas of the cell-tuple approach (as in the earlier work that it builds on), is keeping a sharp line between the topology and the geometry. The language C++ was chosen mainly to make use of this language's class feature, to implement important entities as abstract data types. An overview of the major classes is given in Figure 6.1.

Visual display of higher dimensional objects is a difficult task. It was desired to show geometry and the cell-tuple structure simultaneously. We simply projected all cells into

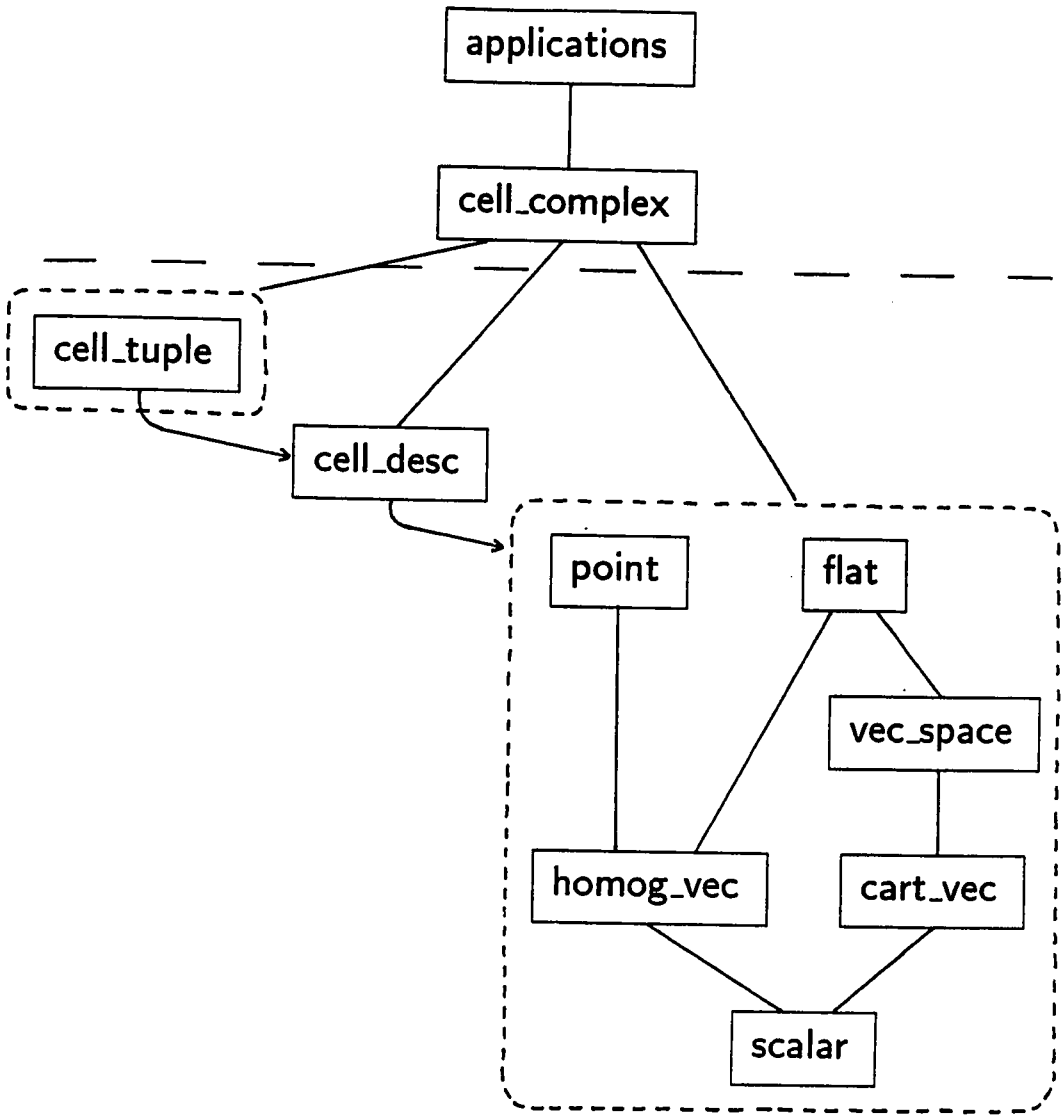


Figure 6.1: Overview of the major classes in the C++ implementation

two dimensions, and used color to distinguish between cells and the differently labeled cell-tuple graph edges.

We will give further detail on these topics in the following sections.

6.3.2 A Note on the Code Description

For the purpose of illustration, it is useful to give examples of programs, routines, and fragments of these. When giving examples, it is desirable to convey the important points without bogging the reader down with unnecessary details. In our examples, we have eliminated all error checking and debugging aids. We will only give examples which involve the structures under consideration.

We will use a pseudo-code when giving examples. This should be readable by anyone familiar with a block structured language which includes pointers. A structure, declared by a `structure` statement, is a named set of memory locations. A structure's members are named entities within the structure; they are accessed as `<structure>.<member>`. Members may be subroutines or functions, as well as built-in and user-defined types and structures. A pointer to an entity is a memory location containing the entities address. Given a pointer `<pstructure>` to a `structure`, its members are accessed as `<pstructure>^<member>`.

The structure `array_of_<type>` is an array of elements of type `<type>`. The number of elements of the array is given by the member `.nelem`. When such an array is created, the number of elements is given as an argument; if that argument is omitted, it will have zero elements initially. The i^{th} element of the array a is accessed as `a[i]`. An element `e` may be added at the end of the array by invoking the member routine `.append(e)`. (What we are calling structures here were actually implemented as C++ classes, to make use of information hiding, operator overloading, constructors and destructors, etc.)

There is a special value `NULL` which may be assigned to pointers, which means that the pointer does not point to anything. This is the default value taken on by pointers upon creation.

Please bear in mind that the presentation made here is for illustrative value, rather than efficiency.

6.3.3 Implementation of the Cell-Tuple Structure

The pointer method of implementing the cell-tuple structure was chosen because it most closely reflected the original definition of the cell-tuple structure. The structure `celltuple` (see Figure 6.2) contains two arrays; one implements the cell-tuple itself as an array of pointers to cell descriptors, and the other implements the *switch* operator as an array of pointers to other `celltuples`.

Because it is desired in this work to keep a clear line between the topology and geometry, and because the description of a cell must contain some geometric information, we keep all geometric information about cells within cell descriptors, which are implemented as the structure `celldesc` (see Figure 6.2). The cell-tuples may make no use of the geometric information they contain. `celltuples` may only point to `celldescs`, so the geometry of cells may be changed without any effect on the cell-tuple structure.

The types `pcelltuple` and `pcelldesc` are pointers to `celltuples` and `celldescs`, respectively.

It will be necessary to do searches on the cell-tuple structure, which requires marking cell-tuples which have been visited. This is implemented by the boolean `.marked` member of `celltuple`. A similar marker is needed to do searches of the cell structure, e.g. when looking for the set of cells incident to a given cell, so `celldesc` also has a `.marked` member.

The dimension of a `celltuple` is accessed by the member `.dim`, and `celltuple` also has an auxiliary member `.aux`, which is a `celltuple` pointer, needed by several operations.

A traversal within the cell-tuple structure is accomplished by the function `ctTraverse`, which returns an array of pointers to cell-tuples, in the order given by the traversal (i.e. this implements $trav(t, I)$, which was first mentioned in Section 5.2.5). An array

```
structure celltuple {
    boolean marked;
    int dim;
    pcelltuple aux;
    array_of_pcelltuple sw;
    array_of_pcelldesc cd;
}

pcelltuple new celltuple {
    marked = false;
    dim = 0;
    aux = NULL;
    sw[0] = BD;
    cell[0] = NULL;
}

structure celldesc {
    boolean marked;
    ...
}

pcelldesc new celldesc {
    marked = false;
    ...
}
```

Figure 6.2: Basic structures **celltuple** and **celldesc**, and constructors

```
array_of_pcelltuple ctTraverse(pcelltuple t, array_of_int search)
{
    ...
}


array_of_pcelltuple ctIncident(pcellTuple t, int k, int l)
{
    int i, d;
    array_of_int search;
    array_of_pcelltuple trav, result;

    search = {0,...,k-1,k+1,...,d};
    trav = ctTraverse(t, search);
    for (i=0; i<trav.nelem; i++) {
        if (trav[i]^cd[l]^marked == false) {
            result.append(trav[i]);
            trav[i]^cd[l]^marked = true;
        }
    }
    for (i=0; i<trav.nelem; i++)
        trav[i]^cd[l]^marked = false;

    return result;
}
```

Figure 6.3: `ctTraverse` and `ctIncident`

of integers specifies the *switch* operators to be used in such a traversal, i.e. given a `pcelltuple t` and the array $I = \{i_0, \ldots, i_\ell\}$ as input, `ctTraverse` returns $switch_I$ applied to `t`. `ctTraverse` is implemented as a depth-first search on the subgraph induced by edges whose labels are in I. A standard stack algorithm is used, so we show only the declaration in Figure 6.3.

When writing routines, it is useful to think in terms of cells. For instance, it is often necessary to work with a set of cells incident to some given cell. The routine `ctIncident` (see Figure 6.3), takes as input `pcelltuple t` and an integer `k` = k (thus specifying the cell t_k), as well as an integer `l` = ℓ, which specifies the dimension of the incident cells

```
void ctSetcell(pcelltuple t, int d, pcelldesc c)
{
    int i;
    array_of_int search;
    array_of_pcelltuple trav;

    search = {0,...,d-1}
    trav = ctTraverse(t, search);
    for (i=0; i<trav.nelem; i++)
       trav[i]^cd[d] = c;
}
```

Figure 6.4: `ctSetcell`

being sought; `ctIncident` returns a list of cell-tuples, which are distinct representatives of the ℓ-cells incident to t_k. This is accomplished by traversing all of the cell-tuples associated with t_k, and appending ℓ-cells to the `result` array (and marking them) when they are first encountered.

The operations that we perform on the topology have no effect on the geometry, yet in practice the application will usually involve geometry. A simple way of providing a connection between the topology and the geometry is to give a single, simple routine for updating geometric information in the topology. This is implemented by `ctSetcell`, which takes as input a cell-tuple, an integer k, and a cell descriptor. The cell-tuple and integer specify the k-cell to be is traversed; the k^{th} component of each cell-tuple in the traversal is set to point to the cell descriptor. (See Figure 6.4.)

6.3.4 Constructors

In the section on constructors, we described *make_vertex*, *kill_vertex*, *lift*, *unlift*, *join*, *unjoin*, *split*, *unsplit*, *copy*, *double*, and *subcx*. In the C++ program, *lift*, *unlift*, *join*, *unjoin*, *split*, *unsplit*, *copy*, and *double* are implemented on cell-tuples, and ct-pair versions of these operations using the cell-tuple implementations serve as a convenient

```
void ctLift(pcelltuple t)
{
    int i, d;
    array_of_int search;
    array_of_pcelltuple trav;

    d = t.dim;
    search = {0,...,d-1};
    trav = ctTraverse(t, search);
    for (i=0; i<trav.nelem; i++) {
        t.dim = d+1;
        t.sw[d+1] = BD;
        t.cd[d+1] = NULL;
    }
}
```

Figure 6.5: `ctLift`

interface. There are no cell-tuple versions of *make_vertex* and *kill_vertex*; the ct-pair versions use `new celltuple`, `ctSetcell` and built-in destructors. Similarly, there is no cell-tuple version of *subcx*; the ct-pair version essentially uses `ctCopy`. We will only give the cell-tuple versions here.

The constructor `ctLift` simply does a traversal of cell-tuples, as specified by a `search` array, and increases the dimension of each by one. (See Figure 6.5.) `ctUnlift` is the same, except that within the `for` loop, we have only the statement `t.dim = d-1`.

The implementation of *attach* is a direct transfer of the description given in Section 5.3.4. (See Figure 6.6.) To implement *join* and *unjoin*, it was convenient to implement the more general *genljoin* constructor, as this is useful in several other routines. Then `ctJoin` and `ctUnjoin` could be implemented as simple calls to `ctGenljoin`. (See Figure 6.7.)

To make a new copy of a subcomplex, the function `ctCopy` takes as input a cell-tuple and a search array. It makes a new cell-tuple for every cell-tuple encountered in the search (starting from the given cell-tuple). As these copies are made, each of the cell-

```
void ctAttach(pcelltuple t1, pcelltuple t2, int k)
{
    if (t1^sw[k] != BD)
        (t1^sw[k])^sw[k] = t2^sw[k];
    if (t2^sw[k] != BD)
        (t2^sw[k])^sw[k] = t1^sw[k];
    t1^sw[k] = t2;
    t2^sw[k] = t1;
}
```

Figure 6.6: `ctAttach`

tuples being copied has its `.aux` member set to point at its copy. This allows the copying of the `.switch` pointers; if two original cell-tuples are connected by $switch_k$, and one of them has been copied, then as soon as the second is copied, $switch_k$ is set for both of the copies. (Figure 6.8 gives the implementation.)

The implementation of *double* is straightforward (see Figure 6.8).

There are a few ways that one might implement *split*. (We will refer to the k-cell being split as the splittee, and the (k–1)-cell which does the splitting as the splittor.) The variations come in the way that the boundary of the splittor is specified. It could, for instance be given as a list of subcells of the splittee. This is simple to specify, but when one tries to implement it, there is some difficulty with 'matching up' the boundaries of the cells in this list with those in the splittee. It is more convenient if the specification of the splittor includes as much information as possible. The way that this was implemented was to require that the boundary of the splittor be constructed, and that each cell-tuple in this boundary have its `.aux` pointer point to the corresponding cell-tuple in the splittee. Note that the case where the splittor is a vertex presents a special case – to construct this boundary is essentially doing the split itself. So in the case where the splitter is a vertex, we simply require cell-tuples identifying the splitter (vertex) and the splittee (edge). The implementation of *unsplit* is somewhat simpler than that of *split*. (See Figures 6.9 and 6.10.)

```
void ctGenljoin(pcelltuple t1, pcelltuple t2, array_of_int search, int k)
{
    int i;
    array_of_pcelltuple trav1, trav2;

    trav1 = ctTraverse(t1, search);
    trav2 = ctTraverse(t2, search);
    for (i=0; i<trav.nelem; i++)
        ctAttach(trav[1], trav[2], k);
}


void ctJoin(pcelltuple t1, pcelltuple t2)
{
    array_of_int search;

    search = {0,...,t1.dim-2};
    ctGenljoin(t1, t2, search, t1.dim);
}


void ctUnjoin(pcelltuple t)
{
    int i, d;
    array_of_pcelltuple trav;
    array_of_int search;

    d = t^dim;
    search = {0,...,d-2};
    trav = ctTraverse(t, search);
    for (i=0; i<trav.nelem; i++) {
        (trav[i]^sw[d])^sw[d] = BD;
        trav[i]^sw[d] = BD;
    }
}
```

Figure 6.7: ctGenljoin, ctJoin and ctUnjoin

```
pcelltuple ctCopy(pcelltuple torig, array_of_int search);
{
    int i;
    pcelltuple t, tnew;
    array_of_pcelltuple trav;

    trav = ctTraverse(torig, search);
    for (i=0; i<trav.nelem; i++) {
        t = trav[i];
        tnew = new celltuple;
        t^aux = tnew;
        for (j=0; j<d; j++) {
            tnew^cd[j] = t^cd[j];
            if (j in search && t^sw[j]^aux != NULL) {
                tnew.sw[j] = t.sw[j]^aux;
                t.sw[j]^aux^sw[j] = tnew;
            }
        }
    }
    for (i=0; i<trav.nelem; i++)
        trav[i]^aux = NULL;
}


void ctDouble(pcelltuple t)
{
    pcelltuple tcopy;
    array_of_int search;

    search = {0,...,t^dim};
    tcopy = ctCopy(t, search);
    ctGenljoin(t, tcopy, search, t^dim);
}
```

Figure 6.8: ctCopy and ctDouble

```
void ctSplit(pcelltuple t, int dc)
{
    int i, d;
    pcelltuple end1, end2, mid1, mid2;
    array_of_int search;
    array_of_pcelltuple trav;

    if (dc == 0) {
        end1 = t;
        end2 = end1^sw[0];
        d = end1^dim;
        search = {2,...,d};
        mid1 = ctCopy(end1, search);
        mid2 = ctCopy(end2, search);
        ctGenljoin(end1, mid1, search, 0);
        ctGenljoin(end2, mid2, search, 0);
        ctGenljoin(mid1, mid2, search, 1);
    }
    else {
        ctLift(t);
        d = t^dim;
        ctDouble(t);
        search = {0,...,d-1};
        trav = ctTraverse(t, search);
        for (i=0; i<trav.nelem; i++) {
            ctAttach(trav[i], trav[i]^sw[d], d+1);
            ctAttach(trav[i], trav[i]^aux, d);
            trav[i]^aux = NULL;
            trav[i]^sw[d+1]^aux = NULL;
        }
    }
}
```

Figure 6.9: ctSplit

```
void ctUnsplit(pcelltuple t, int k)
{
    int i, j;
    pcelltuple t1;
    array_of_int search;
    array_of_pcelltuple trav;

    search = {0,...,k-1,k+2,...,d};
    trav = ctTraverse(t, search);
    for (i=0; i<trav.nelem; i++) {
        t1 = trav[i];
        ctAttach(t1, t1^sw[k+1], k);
        t1.dim = k;
        t1.sw[k] = BD;
    }
}
```

Figure 6.10: `ctUnsplit`

6.3.5 Homogeneous Vectors and Two-Sided Spaces

A d-vector in $\mathbb{R}^d$ may be represented in homogeneous coordinates by a $(d+1)$-vector of real numbers. We will refer to such a representation as a **homogeneous vector**, and use square brackets to distinguish them from Cartesian vectors. The homogeneous vector $[x_0,\ldots,x_d]$, where the x_i are scalars, and $x_0 \neq 0$, represents the Cartesian vector $(x_1/x_0,\ldots,x_d/x_0)$. Thus there is a 1-to-1 correspondence between Cartesian vectors $(y_1,\ldots,y_d)$ and sets of homogeneous vectors $\{[x_0,\ldots,x_d] \mid x_0 \neq 0, x_i/x_0 = y_i, i = 1,\ldots,d\}$. If we now consider the set of all homogeneous vectors $\{[x_0,\ldots,x_d] \mid x_i \in \mathbb{R}\}$, then what do those homogeneous vectors with $x_0 = 0$ correspond to? A natural way to interpret them is as vectors at infinity: $[0,x_1,\ldots,x_d]$ corresponds to a point at infinity 'in the direction of' $(x_1,\ldots,x_d)$. The space of homogeneous vectors can then be given a topology, so as to represent projective d-space, which we will not discuss here.

If the set of homogeneous vectors for $\mathbb{R}^d$ is considered itself as a $(d+1)$-dimensional Euclidean space, then a very nice feature of using homogeneous coordinates is that a

flat in $\mathbb{R}^d$ becomes a subspace in $\mathbb{R}^{d+1}$. Also, any affine transformation on $\mathbb{R}^d$ may be accomplished by multiplication by one $d+1 \times d+1$ matrix, rather than a multiplication by a $d \times d$ matrix and addition of a d-vector, as required in the Cartesian representation. This makes composition of affine transformations implementable by matrix multiplications, which is useful in applications e.g. graphics.

If there is a distinction made between homogeneous vectors whose first coordinate is positive and those whose first coordinate is negative, then we have a representation of a 'two-sided' or 'directed, oriented' d-space, as described by Stolfi in his dissertation [Sto88]. Consider the set of all homogeneous vectors with $d+1$ real coordinates. Two such vectors $u = [u_0, \ldots, u_d]$ and $v = [v_0, \ldots, v_d]$ will be equivalent if $\text{sign}(u_0) = sign(v_0)$ and either (1) u_0 and v_0 are both non-zero and $u_i/u_0 = v_i/v_0$ for $1 \leq i \leq d$ or (2) $u_0 = v_0 = 0$ and $u_i = v_i$ for $1 \leq i \leq d$. This can be interpreted as follows: the equivalence classes corresponding to positive and negative first coordinates are each equivalent to a Euclidean d-dimensional space, and the equivalence classes with zero first coordinate correspond to the (common) points of infinity of both. The whole set of equivalence classes can be given the topology of the d-sphere. The set of points in the Euclidean space corresponding to equivalence classes with positive first coordinates will be called the 'front side' or 'positively oriented' space, while those corresponding to equivalence classes with negative first coordinates will be called the 'back side' or 'negatively oriented' space.

Stolfi defines addition (and subtraction) on vectors in a two-sided space as follows: $[x_0, \ldots, x_d] + [y_0, \ldots, y_d] = [x_0y_0, x_0y_1 + y_0x_1, x_0y_2 + y_0x_2, \ldots, x_0y_d + y_0x_d]$, as long as either x_0 or y_0 is non-zero. This allows not only addition of finite points, but addition of a finite and an infinite point, to give an infinite point. Addition of two infinite points ($x_0 = y_0 = 0$) is undefined. Multiplication by a scalar is: $s * [x_0, x_1, \ldots, x_d] = [x_0, sx_1, \ldots, sx_d]$. If $s \neq 0$, this can be simplified to $s * [x_0, x_1, \ldots, x_d] = [x_0/s, x_1, \ldots, x_d]$.

Homogeneous coordinates were implemented in our program. The two prime motivations were the ability to represent points at infinity in a consistent manner, and the

representation of flats as subspaces in one dimension higher. The latter made the writing of flat intersection routines easier. A particular reason for representing points at infinity was for an arrangement algorithm. If cells either wholly or partly at infinity can be represented in exactly the same way as those not at infinity, the algorithm can be written simply and uniformly.

To treat vectors at infinity uniformly, Stolfi's definition of addition and subtraction was extended to almost all pairs of vectors. The following is the result: $[x_0, \ldots, x_d] + [y_0, \ldots, y_d] = [0, 0, \ldots, 0, x_j y_j, x_j y_{j+1} + y_j x_{j+1}, x_j y_{j+2} + y_j x_{j+2}, \ldots, x_j y_d + y_j x_d]$, where j is the smallest non-negative integer such that either x_j or y_j is non-zero, and $j < d$. It must remain undefined for $j = d$, as that case may turn out to be equivalent to adding $+\infty$ and $-\infty$. This agrees with Stolfi's definition when $j = 0$. The vector $[0, \ldots, 0]$ is considered meaningless – it is used to denote an uninitialized vector, or the results of an illegal operation, in the implementation.

Essentially, what this does is make the space of points at infinity a two-sided space in its own right, of dimension d–1. Of course, this two-sided space has a space at infinity, which is a (d–2)-dimensional two-sided space, and so on. Thus this set of homogeneous vectors can be thought of as the union of 'positive' and 'negative' real spaces of decreasing dimension: $(pos\mathbb{R}^d \cup neg\mathbb{R}^d) \cup ((pos\mathbb{R}^{d-1} \cup neg\mathbb{R}^{d-1}) \cup \cdots \cup ((pos\mathbb{R}^1 \cup neg\mathbb{R}^1) \cup (pos\{0\} \cup neg\{0\}))) \cdots)))$. Thinking of the two sided d-space as a d-sphere, with k-cells for each $pos\mathbb{R}^k$ and $neg\mathbb{R}^k$, we have a subdivision which is the minimal subdivision of the d-sphere.

6.3.6 Geometry

The geometric objects implemented by C++ classes were scalars, Cartesian vectors, homogeneous vectors, points, Cartesian vector spaces, and flats.

The field of scalars which most theoretical algorithms use is the set of real numbers, which cannot be implemented exactly on a computer. When floating point numbers, of whatever precision, are used, round-off error eventually comes into play. Exact imple-

mentations of other fields, such as rational numbers, may work on many problems, but probably not all, and are usually inefficient. There may be occasions where the user wants to treat certain cases as if they were special scalar values, e.g. infinity, not-a-number (the result of an illegal operation). Thus it is particularly important to implement scalars as abstract data types, allowing the possibility of changing the scalar's internals without rewriting higher-level algorithms.

In the current version of the program, `scalars` include a single precision floating point value, plus several other members which indicate if the scalar contains a valid value, and whether it is infinite. The arithmetic and logical operators are overloaded, to make these operations on scalars look the same as on built-in types, and allowing these operators to work on infinite scalars.

Cartesian vectors, `cart_vecs`, are a class implementation of an array of scalars, which allow addition and subtraction, multiplication by a scalar, dot and cross products, and projection of one vector onto another. A real d-vector is represented by a Cartesian vector containing d scalars. Homogeneous vectors, `homog_vecs`, are also arrays of scalars. Cartesian vectors may be represented by `homog_vecs`, and the same operations are supported as for `cart_vecs`.

A point in $\mathbb{R}^d$ is represented by homogeneous coordinates as if it was a Cartesian vector. If p_1 and p_2 are points and v is a vector, then the following operations are possible: $p + v$ gives a point, $v + p$ gives a point, and $p_1 - p_2$ gives a vector. There is no multiplication by a scalar, or addition of points.

A subspace of $\mathbb{R}^d$ is implemented as a class `vec_space`. The subspace is represented by a set of orthogonal vectors in Cartesian coordinates, which span the subspace. Upon creation, an instance of this class is empty. Arbitrary vectors (of like dimension) can be added – when a vector v is added, if the current subspace spanned is S, and if $v \notin S$, then $v -$ (proj of v into S) is added to the existing orthogonal basis. The join and meet of two subspaces can be formed, the perpendicular subspace can be formed, the perpendicular vector from the subspace to a vector may be formed, the inclusion of a vector in a space

may be decided, and so on.

Flats are implemented by the class `flat`. This class represents a flat by its homogeneous coordinates, that is as a subspace in the Euclidean space of one higher dimension. This allows the join and meet of two flats, and so on, to be produced by using those operations on the `vec_space`. By using homogeneous coordinates, flats at infinity are represented in the same way as those which are not.

6.3.7 Cell Descriptors

Cell descriptors provide the means to separate the representation of topological structure from the representation of geometry. Cell descriptors may contain whatever geometric information the user wishes to put in them, and this may change at any time. Cell-tuples only point to cell descriptors, so remain unaffected by changes to their internals. The only exception is that a flag is provided to the cell-tuple operations, allowing cell descriptors to be marked for search operations.

All of the problems we will consider concern 'flat' cells, that is cells which can be formed by the intersection of a flat and some number of half-spaces. In particular, every cell is the convex hull of a finite set of points. This gives one simple way of representing geometry – if every cell descriptor for a vertex contains a point, and every cell descriptor for higher dimensional cells has a pointer back into the cell-tuple structure, this is enough: to obtain the set of points defining a higher dimensional cell, the set of all vertices can be obtained by a call to `ctIncident`.

In practice, one often wants to do tests involving the spanning flat of a cell, so it is useful for cell descriptors to contain a flat or a pointer to a flat (we actually include a flat). Since our implementation was to be used for experimentation, it was useful to include other information, in particular information for drawing cells. This will be discussed in the next section.

6.3.8 Display

One of the big problems with working with objects in higher dimensional spaces is visualizing them. This is an area of current research in and of itself, and is out of the domain of this dissertation.

In our implementation, visual output was designed to be sent to any standard 2-dimensional display. Images were displayed on both monochrome and color workstations using X, on a stand-alone frame-buffer machine connected to a camera, and printed in a color-separated format in PostScript. Using X allowed the use of animation, and the other two allowed the production of hard-copy output of static images.

As the main goal of having a visual display was for debugging and illustrative purposes, it seemed desirable to be able to display the geometry of the cells and the cell-tuple structure simultaneously. To see how the constructors, and algorithms using them, worked it seemed useful to include an animated display which reflected the changes performed by the constructors.

For the display of cells, it was decided to draw only vertices and edges, for two reasons. The first was that drawing higher dimensional cells would increase the time to develop the software, and the second was that it was thought that it would increase the running time enough to interfere with the effectiveness of the animation. A simple scheme was used for projecting points into the image plane. For each dimension i in $\mathbb{R}^d$, a 2-vector u_i is specified in the image plane, representing that axis' projection. Then the point $(x_1, \ldots, x_d)$ is projected to $x_1 u_1 + \cdots + x_d u_d$.

Because it was desired to be able to work with infinite cells, it was decided to include an option which mapped the extended $\mathbb{R}^d$ onto S^d, and then project the result into the image plane. Thus, extended $\mathbb{R}^2$ is represented as a disk in the image plane and extended $\mathbb{R}^3$ is represented as a closed ball. To represent two-sided space requires showing two separate disks or balls, and the line or plane at infinity is depicted twice. Straight line segments in extended $\mathbb{R}^2$ or $\mathbb{R}^3$ are then depicted as sections of ellipses.

To display the cell-tuple structure, cell-tuples were displayed as points, and the graph

G_M was drawn with appropriate edges between the cell-tuples. The dimension of the cell-tuple was given by the color of the dot depicting it, and the label of a graph edge was depicted by its color.

It was necessary to devise a method for placing cell-tuples. They should, as intuitively described earlier, be located within their d-component and near their other components. Using the definition of addition for homogeneous vectors given in Section 6.3.5, we can define the barycenters of both finite and infinite cells. Since the cells under consideration are convex, their barycenters lie inside them. A natural way to compute a location for cell-tuples is to take a weighted average of their components' barycenters. This is what was done. Various weightings were tried until one was found which gave visually pleasing results.

There is one problem with the location scheme just described, which has to do with points at infinity: if a cell has some finite and some infinite vertices, the barycenter ends up at infinity, while we would like it to fall in the finite portion of the cell for display. This method also sends all cell-tuples of such a cell to infinity. This problem was handled by using a slightly more complicated solution, for display purposes only, which isn't interesting enough to describe here.

There were various options offered for controlling animation. At the finest level the image could be updated after every change to a `celldesc`, and/or after every change to a `celltuple`. This was good for low-level debugging. At a somewhat higher level, the image could be updated after every application of a constructor. This was good for debugging and demonstrating algorithms. The decision as to which of the above options to used is made when the program is started. During the running of the program, the user can have the program pause after any specified number of updates. Thus, by telling it to run continuously for, say, 100 updates, then pause, then advance one update at a time between pauses, the user can get quickly to the portion of the algorithm of interest, then proceed slowly.

The display, with the various projection and animation options, was extremely valu-

able in developing the program.

6.3.9 Algorithms Implemented

We will briefly describe here implementations of algorithms to build 2-dimensional arrangements, two-sided spaces of arbitrary dimension, and simplices of arbitrary dimension.

The optimal algorithm for constructing 2-dimensional arrangements given in [CGL83] was a natural choice, as it uses ordering, and the arrangement has infinite regions (allowing us to take advantage of the generality of our representation of cells). This is an incremental, or on-line, algorithm. The arrangement is built by adding the input lines, one at a time in their input order, to the arrangement built thus far. To add a line, the left-most edge which it intersects, in the current arrangement, is found (in our implementation, this edge is at infinity). Then the boundary of the region to the right of this edge is checked, by following the incident edges in order around the region, until the edge is found which intersects the line being inserted. The two intersections found so far define an edge of the arrangement, which is used to split the region. Now the region to the right of the most recent intersection found is searched, and so on. An example of such an insertion is given in Figure 6.11. Our implementation is given in Figure 6.12.

Before adding any lines, the routine creates the cell-tuple structure for a 2-dimensional two-sided space. To insert any line, the cell-tuples associated with the back space (of the two-sided space) are traversed (in either order), essentially 'walking around on the other side of the line at infinity'. Each $switch_{01}$ brings a cell-tuple with a new edge as its 1-component. The line being inserted is checked to see if it intersects this edge. If so, that edge is split by the intersection point. Doing a $switch_{21}$ from the explorer cell-tuple (`tcheck`) gives a cell-tuple on the first edge of the next region to be explored. The cell-tuples on the boundary of this new region are traversed by a sequence of $switch_{01}$'s. When an edge on this region is found which intersects the line, this edge is split, and then the region is split by an edge connecting the two new points. And so on. When

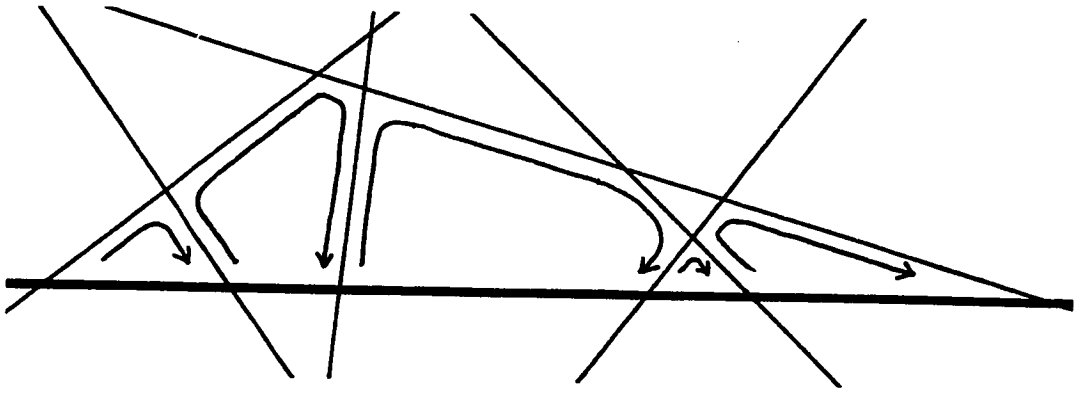


Figure 6.11: Inserting a line into a 2-dimensional arrangement

the line has been completely inserted, then the explorer cell-tuple winds up on the back space, ready to insert a new line.

The arrangement routine calls the routine `twoSidedSpace`, which is interesting in and of itself. By iterating the `ctDouble` routine d times, the cell-tuple structure for a minimal subdivision of the d-sphere is constructed. The rest of the routine is devoted to constructing the flats at infinity with the appropriate orientations. (See Figure 6.13.)

As an example of the use of *join* in higher dimensions, we give here the algorithm to build a d-dimensional simplex (Figure 6.14). It consists of d calls to the more general routine `cone`, which makes the cone of a cell (see Figure 6.15).

6.3.10 Conclusions

The C++ code for the system was about 6700 uncommented lines in length, which were divided into major components with the sizes given in the following list.

- cell descriptors: 456 lines,
- cell-tuples: 1121 lines,

```
pcelltuple arrangement(array_of_flat lines)
{
    boolean first;
    int i;
    pcelltuple tcheck, tlast, tnew, tend1, tend2, space;
    pcelldesc back_space;
    flat cur_line;

    space = twoSidedSpace(2);
    tcheck = space^sw[2];
    back_space = tcheck^cd[2];
    for (i=0; i<lines.nelem; i++) {
        first = true;
        repeat {
            while (intersect(lines[i],tcheck^cd[1]) == NULL)
                tcheck = (tcheck^sw[0])^sw[1];
            ctSplit(tcheck, 0);
            tnew = tcheck^sw[0];
            tnew^cd[0] = intersect(lines[i], tcheck^cd[1];
            if (!first) {
                tend1 = new celltuple;
                tend2 = new celltuple;
                tend1^aux = tlast;
                tend2^aux = tnew;
                ctJoin(tend1, tend2);
                ctSplit(tend1, 1);
            }
            tlast = tnew^sw[2];
            tcheck = (tcheck^sw[2])^sw[1];
            first = false;
        } until (tcheck^cd[2] == back_space);
    }
    return tcheck;
}
```

Figure 6.12: arrangement (construct 2-dimensional arrangement)

```
pcelltuple twoSidedSpace(int d)
{
    int i, j;
    pcelltuple tpos;
    point ppos, pneg;
    flat fpos, fneg;

    tpos = new celltuple;
    fneg.reverseOrient;
    for (i=0; i<=d; i++) {
        ctDouble(tpos);
        for (j=0; j<=d; j++) {
            ppos[j] = scalar(0);
            pneg[j] = scalar(0);
        }
        ppos[i] = scalar(i);
        pneg[i] = scalar(-1);
        fpos.addPoint(ppos);
        fneg.addPoint(pneg);
        ctSetcell(tpos, i, new celldesc(fpos));
        ctSetcell(tpos^sw[i], i, new celldesc(fneg));
    }
    return tpos;
}
```

Figure 6.13: `twoSidedSpace` (make a d-dimensional two-sided space)

```
pcelltuple simplex(array_of_point points)
{
    int i;
    pcelltuple s;

    s = new celltuple;
    s^cd[0] = new celldesc(points[0]);
    for (i=1; i<points.nelem; i++)
        s = cone(s, new celldesc(points[i]));
    return s;
}
```

Figure 6.14: `simplex` (make an n-dimensional simplex)

```
pcelltuple cone(pcelltuple tbase, pcelldesc v_apex)
{
    int i, j, d;
    pcelltuple t, tsub_facet, tadj_sub_facet;
    array_of_int search;
    array_of_pcelltuple trav_base, trav_facet;

    d = tbase^dim;
    if (d == 0) {
        t = new celltuple;
        t^cd[0] = v_apex;
        ctJoin(tbase, t);
    }
    else if (d == 1) {
        search = {};
        t = cone(ctCopy(tbase, search), v_apex);
        ctJoin(tbase, t);
        t = cone(ctCopy(tbase^sw[0], search), v_apex);
        ctJoin(tbase^sw[0], t);
        ctJoin((tbase^sw[1])^sw[0], t^sw[0]);
    }
    else {
        trav_base = ctIncident(tbase, d, d-1);
        search = {0,...,d-2};
        for (i=0; i<trav_base^nelem; i++) {
            ctJoin(trav_base[i],
                   cone(ctCopy(trav_base[i], search), v_apex);
            trav_facet = ctIncident(trav_base[i], d-1, d-2);
            for (j=0; j<trav_facet^nelem; j++) {
                tsub_facet = trav_facet[j];
                tadj_sub_facet = tsub_facet^sw(d-1);
                if (tadj_sub_facet^sw(d) != NULL)
                    facetJoin((tsub_facet^sw(d))^sw(d-1),
                              (adj_sub_facet^sw(d))^sw(d-1);
            }
        }
    }
    lift(base);
    return base;
}
```

Figure 6.15: cone (construct cone of d-cell)

- ct-pairs: 810 lines,
- display: 1322 lines,
- geometry: 3028 lines.

Implementing the cell-tuple structure was fun and relatively straight-forward. Implementing the geometry, on the other hand, was a never-ending task. The one exciting part of implementing the geometry was the work described with two-sided spaces. When undertaking the implementation, the author had high hopes of producing a general, robust, and efficient d-dimensional geometric package. These goals had to be scaled back considerably as the complexity of such an undertaking became apparent. Implementing the display took a reasonable amount of work, and turned out reasonably well. There were many decisions to be made about when to compute needed values for the display, and where to keep them. Having the animation feature was very valuable, if not indispensable, for debugging and understanding.

Working with C++, even with all of its shortcomings, was worthwhile. The class structure provided, as much as anything, a good organizational framework, and the type-checking was very helpful.

It is likely that application-specific implementations, in particular for low dimensions, could be built relatively easily and could be competitive with other existing methods. When moving to higher dimensions, the difficulty of the geometry increases, and the efficiency issues become important very quickly.

Chapter 7

Conclusion

7.1 Synopsis

The primary contribution of this dissertation is the introduction of the cell-tuple structure, which gives a simple, easily implemented data structure for representing the topological structure of geometric objects, the ordering within such objects, their topological duals, and their boundaries. This work generalizes and unifies existing work in this area. In the cases where it overlaps other work, it provides a simpler, more consistent structure.

Part of this work, a contribution in itself, is the generalization of the ordering of cells familiar in two and three dimensions to the general higher dimensional case, whose existence was proved by using the cell-tuple structure. It is hoped that this ordering will lead to new results in computational geometry.

The class of objects studied, the subdivided manifolds, is felt to be general enough to encompass most applications, yet restricted enough to admit representation by the incidence graph and by the cell-tuple structure, and also allows the proof of the ordering results.

In addition to describing the cell-tuple structure as a static representation, we have investigated possible constructors for creating and modifying subdivided manifolds.

In an attempt to make this work useful, we have discussed implementation issues and an actual implementation. The work described in this thesis has been implemented, using a variant of the database method described in Section 6.2.4, as part of the most recent version of Bolt, Baranak and Newman's distributed tank simulator ([Smy89]).

7.2 Directions for Future Work

We will briefly discuss three possible directions in which further work might be pursued: generalizing and extending the cell-tuple structure and similar representation schemes as an end in itself; using the cell-tuple structure to obtain new results in computational geometry; and the use of the cell-tuple structure in practical applications.

In this work, the basic building blocks were cells whose boundaries were not allowed to self-intersect, and the objects represented by their unions (the underlying spaces) were topological manifolds. A natural generalization of the problem is to consider larger classes of cells, or underlying spaces, or both. If a larger class of cells is allowed, the cell-tuple description does not work out as simply, but many of the basic ideas may go through. Similarly, if a larger class of underlying spaces is allowed, i.e. non-manifolds, the ordering result may be lost, but additional representational power may be gained. Weiler ([Wei86]) offers an approach for handling non-manifolds in two dimensions. Lienhardt's work ([Lie89]) on n-G-maps deals with a very general class, a class which is larger than the class of all spaces which follow the rules (ct1) – (ct5). There may be a valuable class of objects between those represented by the cell-tuple structure and by n-G-maps which offers increased representational power at a small loss of simplicity and ordering properties.

Another direction in which the cell-tuple structure might be extended is to modify it so as to simplify the connection with geometric data, as discussed in Section 5.5. By modifying the representation of topology, and possibly restricting the geometry of the objects represented, the problems discussed in that section might be eliminated. This would greatly enhance the performance and ease of use of representations of geometric

objects in applications.

One of the most exciting possibilities for future work is in using the cell-tuple structure to obtain new algorithms for open computational geometry problems. We will discuss possibilities for future work on the problems described in Section 1.2. The asymptotically best existing algorithms for computing the convex hull of n points in the general case of d dimensions run in time $O(n^{\lfloor d/2 \rfloor})$ for even $d \geq 4$, and time $O(n^{\lfloor d/2 \rfloor} \log n)$ for odd $d > 4$. The existing lower bound for $d \geq 4$ is $\Omega(n^{\lfloor d/2 \rfloor})$, given by the size of the output, so closing the $\log n$ gap for odd $d > 4$ is an open problem. Another open problem for higher dimensions is to generalize the divide-and-conquer algorithm devised for the 3-dimensional convex hull ([PH77]) to $d \geq 4$. (Buckley [Buc88] gives a divide-and-conquer algorithm for $d = 4$, but gives no analysis of the running time. It does not appear to be optimal.) Edelsbrunner comments on this problem ([Ede87], pp. 173): "A crucial step in this generalization will be the formulation of order relations among the faces of the convex hull which can be exploited in 'merging' two separated polytopes." Perhaps the ordering developed in this dissertation may help to produce such a generalization.

For the Voronoi diagram, there is a similar $\log n$ gap for even $d \geq 4$. Since the lifting method reduces the Voronoi diagram problem to the convex hull problem in one higher dimension, an improved convex hull algorithm will also give an improved Voronoi diagram algorithm.

As there exists an optimal algorithm for the general case of $d \geq 2$, the d-dimensional arrangement problem is not open. However, the problem of finding the d-dimensional arrangement efficiently in parallel was an open problem, for which the first solutions are given by Hagerup, Jung, and Welzl [HJW90] and Anderson, Beame, and Brisson [ABB90]. The first of these ([HJW90]) gives an optimal randomized CRCW PRAM algorithm for constructing the d-dimensional arrangement. The second ([ABB90]) gives an optimal randomized on-line EREW PRAM algorithm for 2-dimensional arrangements, and a near-optimal (off by a factor of $\log^* n$) deterministic CREW PRAM algorithm for d-dimensional arrangements. This deterministic d-dimensional algorithm originally used

the cell-tuple structure to develop the d-dimensional algorithm; it was later found to be possible to eliminate the explicit use of the cell-tuple structure if desired, though the cell-tuple structure works quite naturally.

We have described in this dissertation an implementation, which may be considered as a prototype. No attempt was made to conserve memory and no great effort was made to improve speed or tune the implementation to a particular application. To be used in any real system, it would be necessary to optimize memory usage and performance for the particular application. This means choosing from among the implementation methods presented in Section 6.2, as well deciding how to maintain the connection between geometry and topology. In any real application one will have to implement a geometry package with some degree of robustness, and some means of displaying the output.

It is hoped that the cell-tuple structure will provide a means of producing new theoretical results, as well as being used in real applications, where its uniformity can simplify the necessary query and construction operations.

Bibliography

[AB83] David Avis and Binay K. Bhattacharya. Algorithms for Computing d-Dimensional Voronoi Diagrams and Their Duals. In Franco P. Preparata, editor, *Advances in Computing Research, Volume 1: Computational Geometry*, pages 159–180. JAI Press, 1983.

[ABB90] Richard Anderson, Paul Beame, and Erik Brisson. Parallel Algorithms for Arrangements. In *Second Annual Symposium on Parallel Algorithms and Architectures*, 1990.

[Bau75] B. G. Baumgart. A Polyhedron Representation for Computer Vision. *AFIPS National Computer Conference Proceedings*, 44:589–596, 1975.

[BEH79] A. Baer, C. Eastman, and M. Henrion. Geometric Modelling: a Survey. *Computer-Aided Design*, 11(5):253–272, 1979.

[BHS80] I. C. Braid, R. C. Hillyard, and I. A. Stroud. Stepwise Construction of Polyhedra in Geometric Modelling. In K. W. Brodlie, editor, *Mathematical Methods in Computer Graphics and Design*, pages 123–141. Academic Press, 1980.

[Bow81] A. Bowyer. Computing Dirichlet Tesselations. *The Computer Journal*, 24(2):162–166, 1981.

[Bro79] Kevin Q. Brown. Voronoi Diagrams from Convex Hulls. *Information Processing Letters*, 9(5):223–228, 1979.

[Buc88] C. E. Buckley. A Divide-and-Conquer Algorithm for Computing 4-Dimensional Convex Hulls. In H. Noltemeier, editor, *Computational Geometry and its Applications: Proceedings of the International Workshop on Computational Geometry CG '88*, pages 113–135. Springer-Verlag, 1988.

[CGL83] B. Chazelle, L. J. Guibas, and D. T. Lee. The Power of Geometric Duality. In *24th Symposium on Foundations of Computer Science*, pages 217–225, 1983.

[CK70] Donald R. Chand and Sham S. Kapur. An Algorithm for Convex Polytopes. *Journal of the ACM*, 17(1):78–86, 1970.

[DH07] M. Dehn and P. Heegaard. Analysis Situs. *Encyklopadie der Mathematischen Wissenschaften mit Einschluss ihrer Anwendungen*, III AB3 Leipzig:153–220, 1907.

[DH87] Andreas W. M. Dress and Daniel Huson. On Tilings in the Plane. *Geometriae Dedicata*, 24:295–310, 1987.

[DK78] Gopal Danaraj and Victor Klee. A Representation of 2-Dimensional Pseudomanifolds and its Use in the Design of a Linear-Time Shelling Algorithm. *Annals of Discrete Mathematics*, 2:53–63, 1978.

[DL87] David P. Dobkin and Michael J. Laszlo. Primitives for the Manipulation of Three-Dimensional Subdivisions. *Proceedings of the 3rd ACM Symposium on Computational Geometry*, pages 86–99, 1987.

[Dwy86] Rex A. Dwyer. A Simple Divide-and-Conquer Algorithm for Constructing Delaunay Triangulations in $O(n \log n \log n)$ Expected Time. In *Proceedings of the 2nd ACM Symposium on Computational Geometry*, pages 276–284, 86.

[Dwy87] Rex A. Dwyer. On the Convex Hull of Random Points in a Polytope. *Journal of Applied Probability*, pages 688–699, 87.

[Dwy89] Rex A. Dwyer. Higher-Dimensional Voronoi Diagrams in Linear Expected Time. In *Proceedings of the 5th ACM Symposium on Computational Geometry*, pages 326–333, 89.

[Edd77] William F. Eddy. A New Convex Hull Algorithm for Planar Sets. *ACM Transactions on Mathematical Software*, 3(4):398–403, 1977.

[Ede87] H. Edelsbrunner. *Algorithms in Combinatorial Geometry.* Springer-Verlag, 1987.

[Efr65] Bradley Efron. The Convex Hull of a Random Set of Points. *Biometrika*, 52(3 and 4):331–343, 1965.

[ELS75] C. Eastman, J. Lividini, and D. Stoker. A Database for Designing Large Physical Systems. *AFIPS National Computer Conference Proceedings*, 44:603–611, 1975.

[EOS86] H. Edelsbrunner, J. O'Rourke, and R. Seidel. Constructing Arrangements of Lines and Hyperplanes with Applications. *SIAM Journal on Computing*, 15(2):341–363, 1986.

[FH89] Reinhard Franz and Daniel Huson. The Classification of Quasi-Regular Polyhedra of Genus 2. Unpublished manuscript, 1989.

[For86] Steven Fortune. A Sweepline Algorithm for Voronoi Diagrams. *Proceedings of the 2nd ACM Symposium on Computational Geometry*, pages 313–322, 1986.

[Für86] Z. Füredi. Random Polytopes in the d-Dimensional Cube. *Discrete and Computational Geometry*, pages 314–319, 1986.

[Gra72] R. L. Graham. An Efficient Algorithm for Determining the Convex Hull of a Finite Planar Set. *Information Processing Letters*, 1:132–133, 1972.

[Grü67] Branko Grünbaum. *Convex Polytopes.* Interscience Publishers, 1967.

[GS78] P. J. Green and R. Sibson. Computing Dirichlet Tesselations in the Plane. *The Computer Journal*, 21(2):168–173, 1978.

[GS85] Leonidas 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.

[HJW90] Torben Hagerup, Hermann Jung, and Emo Welzl. Efficient Parallel Computation of Arrangements of Hyperplanes in d Dimensions. In *Second Annual Symposium on Parallel Algorithms and Architectures*, 1990.

[Jar73] R. A. Jarvis. On the Identification of the Convex Hull of a Finite Set of Points in the Plane. *Information Processing Letters*, 2:18–21, 1973.

[Kle64] Victor Klee. Convex Polytopes and Linear Programming. In *Proceedings IBM Scientific Computing Symposium on Combinatorial Problems*, pages 123–158, 1964.

[Kle80] Victor Klee. On the Complexity of d-Dimensional Voronoi Diagrams. *Archiv Der Mathematik*, 34:75–80, 1980.

[KS86] David G. Kirkpatrick and Raimund Seidel. The Ultimate Planar Convex Hull Algorithm? *SIAM Journal on Computing*, 15(1):287–299, 1986.

[Lee80] D. T. Lee. Two Dimensional Voronoi Diagram in the l_p Metric. *Journal of the ACM*, 27:604–618, 1980.

[Lie88] Pascal Lienhardt. Subdivisions of Surfaces and Generalized Maps. Technical report, Départment D'Informatique, Universite Louis Pasteur, 1988.

[Lie89] Pascal Lienhardt. Subdivisions of n-Dimensional Spaces and n-Dimensional Generalized Maps. *Proceedings of the 5th ACM Symposium on Computational Geometry*, pages 228–236, 1989.

[LW69] Albert T. Lundell and Stephen Weingram. *The Topology of CW Complexes.* Van Nostrand Reinhold, 1969.

[Mar58] A. A. Markov. The Problem of Homeomorphy. In *Proceedings of the International Congress of Mathematicians*, 1958.

[MP78] D. E. Muller and Franco P. Preparata. Finding the Intersection of Two Convex Polyhedra. *Theoretical Computer Science*, 7(2):217–236, 1978.

[MS82] Martti Mantyla and Reijo Sulonen. GWB: A Solid Modeler with Euler Operators. *IEEE Computer Graphics and Applications*, 2(7):17–31, 1982.

[Mun75] James R. Munkres. *Topology: A First Course.* Prentice-Hall, 1975.

[Mun84] James R. Munkres. *Elements of Algebraic Topology.* Addison-Wesley, 1984.

[PH77] Franco P. Preparata and S. J. Hong. Convex Hulls of Finite Sets of Points in Two and Three Dimensions. *Communications of the ACM*, 20(2):87–93, 1977.

[Poi04] H. Poincaré. Cinquième Complément à l'Analysis Situs. *Rendiconti del Circolo Matemtico di Palermo*, 18:45–110, 1904.

[Pre79] Franco P. Preparata. An Optimal Real-Time Algorithm for Planar Convex Hulls. *Communications of the ACM*, 22(7):402–405, 1979.

[Sal66] George Thomas Sallee. *Incidence Graphs of Convex Polytopes.* PhD thesis, Department of Mathematics, University of Washington, 1966.

[Sei86] Raimund Seidel. Constructing Higher-Dimensional Convex Hulls at Logarithmic Cost per Face. *Proceedings of the 18th ACM Symposium on Theory of Computation*, pages 404–413, 1986.

[SH75] Michael Ian Shamos and Dan Hoey. Closest-Point Problems. *17th Symposium on Foundations of Computer Science*, pages 151–162, 1975.

[Sib78] R. Sibson. Locally Equiangular Triangulations. *The Computer Journal*, 21(3):243–245, 1978.

[Smy89] S. Smyth. UDIS Paper – Data Model. Unpublished manuscript, 1989.

[Spe88] Jean-Claude Spehner. Merging in Maps and in Pavings. Technical report, Laboratoire de Mathématiques et Informatique, Universite Haute Alsace, 1988. Technical Report.

[Sti84] John Stillwell. *Classical Topology and Combinatorial Group Theory.* Number 72 in Graduate Texts in Mathematics. Springer-Verlag, 1984.

[Sto88] Jorge Stolfi. *Primitives for Computational Geometry.* PhD thesis, Department of Computer Science, Stanford University, 1988. Published as SRC Report number 36, by Digital Equipment Corporation's Systems Research Center.

[Swa85] Garret Swart. Finding the Convex Hull Facet by Facet. *Journal of Algorithms*, 6:17–48, 1985.

[Tit81] Jacques Tits. A Local Approach to Buildings. In Chandler Davis, Branko Grünbaum, and F. A. Sherk, editors, *The Geometric Vein*, pages 519–547. Springer-Verlag, 1981.

[VKF74] I. A. Volodin, V. E. Kuznetsov, and A. T. Fomenko. The Problem of Discriminating the Standard Three-Dimensional Sphere. *Russian Mathematical Surveys*, 29(5):71–172, 1974.

[Wat81] D. F. Watson. Computing the n-Dimensional Delaunay Tessellation with Applications to Voronoi Polytopes. *The Computer Journal*, 24(2):167–172, 1981.

[Wei85] K. Weiler. Edge-Based Data Structures for Solid Modeling in Curved Surface Environments. *IEEE Computer Graphics and Applications*, 5(1):21–40, 1985.

[Wei86] Kevin J. Weiler. *Topological Structures for Geometric Modeling.* PhD thesis, Department of Computer and Systems Engineering, Rensselaer Polytechnic Institute, 1986.

[Whi49] J. H. C. Whitehead. Combinatorial Homotopy I. *Bulletin of the American Mathematical Society*, 55:213–245, 1949.

Biographical Note

Erik Brisson was born on April 12, 1957 in Kansas City, Missouri, graduated from Dighton-Rehoboth Regional High School in 1974, earned the degree of Bachelor of Science in Mathematics from the Massachusetts Institute of Technology in 1978, the degree of Master of Arts in Mathematics from the University of California, San Diego in 1983, and the degree of Doctor of Philosophy in Computer Science from the University of Washington in 1990.