Reconstruction of Orthogonal Polyhedra

by

Burkay Genc

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Computer Science

Waterloo, Ontario, Canada, 2008

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Burkay Genc

Abstract

In this thesis I study reconstruction of orthogonal polyhedral surfaces and orthogonal polyhedra from partial information about their boundaries. There are three main questions for which I provide novel results. The first question is "Given the dual graph, facial angles and edge lengths of an orthogonal polyhedral surface or polyhedron, is it possible to reconstruct the dihedral angles?" The second question is "Given the dual graph, dihedral angles and edge lengths of an orthogonal polyhedral surface or polyhedron, is it possible to reconstruct the facial angles?" The third question is "Given the vertex coordinates of an orthogonal polyhedral surface or polyhedron, is it possible to reconstruct the edges and faces, possibly after rotating?"

For the first two questions, I show that the answer is "yes" for genus-0 orthogonal polyhedra and polyhedral surfaces under some restrictions, and provide linear time algorithms. For the third question, I provide results and algorithms for orthogonally convex polyhedra. Many related problems are studied as well.

Acknowledgements

I thank my supervisor Prof. Therese Biedl for her guidance and friendship throughout this study. Her professional contribution to this thesis is invaluable.

I thank Professors Anna Lubiw and Craig Kaplan for their efforts as members of my committee. Their early corrections and advice shaped this thesis.

I thank Professors Jack Snoeyink and Todd Veldhuizen for their efforts as members of my committee. Their final corrections made this a true PhD thesis.

I thank my family for patiently waiting for me to return. I know that it was not easy, because it was not easy for me.

I thank all of my friends in Canada. Cursing bad weather has no point unless somebody hears it and asks "Wouldn't you like to be in Turkey now?"

And thank you God, for anything and everything.

Contents

List of Figures

Chapter 1

Introduction

In this thesis, I study the problem of reconstructing orthogonal polyhedral surfaces and orthogonal polyhedra from partial information.

Any scientist working on polyhedra knows that it is a difficult challenge to define precisely what polyhedra are. The challenge comes from the fact that there is no universally accepted set of objects that can be called polyhedra. Indeed, many important figures in mathematics and geometry tried to give a precise definition of what a polyhedron is. Some definitions were rejected for being too vague, and others for being too restrictive. Lakatos gives an excellent account of this debate [Lak76] by impersonating each scientist as a smart pupil in a classroom.

In the core of this historical debate lies *Euler's Polyhedral Formula*. This formula, stated by Euler in its initial form, relates the number of faces, edges and vertices of a polyhedron to each other. However, it is quite easy to show that this formula does not hold for some objects which can be considered as polyhedra by many scientists. Therefore, "improved" formulas have been proposed by other scientists that hold for more objects. These improved formulas led to improved definitions of polyhedra. As a result the definition of polyhedra evolved in time.

Besides defining polyhedra, we need methods for *describing* polyhedra, so that two different polyhedra can be distinguished from each other. Assume that you want a carpenter to manufacture a complex polyhedron for you. How would you describe it to the carpenter so that he can manufacture the exact polyhedron you want?

Orthogonal polyhedra are a type of polyhedra which admit only certain types of edges and faces (precise definitions will be given in Section 2.1). In this thesis, I will present two approaches for describing orthogonal polyhedra. I will show that some

elements in these descriptions carry redundant information and can be removed to be reconstructed from the remaining information when necessary. Therefore, my goal in this thesis is to study descriptions of orthogonal polyhedra and their reconstruction from these descriptions.

1.1 Motivation

Aside from being an interesting and attractive research topic for a computational geometer, reconstruction of orthogonal polyhedra has strong relations with other research topics in computational geometry. Here, I list the important related research topics and explain the motivation behind studying reconstruction of orthogonal polyhedra.

The first related research topic is Cauchy's Rigidity Theorem which states that if the faces of a convex polyhedron were replaced with rigid plates and the edges were replaced with hinges, the polyhedron would still be rigid (undeformable) (see e.g. [AZ99]). This is a very important result in computational geometry. Unfortunately, there is no known polynomial time algorithm to reconstruct the angles at the hinges. A natural question to ask is whether this result extends to non-convex polyhedra with other restrictions, such as orthogonal polyhedra. For orthogonal polyhedra, is it possible to provide a polynomial time algorithm that reconstructs the angles at the hinges? Furthermore, Cauchy shows that the faces determine the angles at the hinges, but is the opposite also true? If one is given the angles at the hinges, is it possible to reconstruct the faces?

The second research topic is the *net-folding* problem (see e.g. [DO07]), where the goal is to reconstruct a polyhedron from a net, i.e., what one obtains if a polyhedron is cut along some edges until its surface lies unfolded in a plane. In order to reconstruct the polyhedron, one needs to compute the folding angles at the edges of the net. These are the hinge angles in Cauchy's theorem. However, in this problem some of the hinges are not given. As in the previous example, there is no known polynomial time algorithm to reconstruct the folding angles for an arbitrary net.

The third related research topic is checking the validity of the description of a polyhedron used as input to an algorithm. Indeed, I started studying reconstruction of orthogonal polyhedra to validate input of an unfolding heuristic. This input was basically a graph based description of an orthogonal polyhedron. The input was created manually and was prone to errors. Validating such an input manually is

a tedious task. It can be done much faster by reconstructing the corresponding polyhedron.

In part of this thesis, I study reconstruction from vertex coordinates only, which has connections to numerous research topics. An interesting one is based on a childhood game called "connect-the-dots". In the original version of the game the player is given a point set in plane, as well as a fixed ordering of these points. Then, the player must draw lines between the points according to the given ordering and reveal a picture (or a polygon if the player is a geometer). If the ordering of the points is not given, is it still possible to obtain the same polygon in polynomial time? For general polygons the answer is negative. However, O'Rourke provides an algorithm for orthogonal polygons [O'R88]. He assumes that the given point set is the exact set of vertices of an orthogonal polygon. Is it possible to extend this result to three dimensions for orthogonal polyhedra? What if the points correspond to a "rotated" orthogonal polygon or polyhedron?

Pattern and object recognition from point sets [O'C74, Ahu80, ABCO+01] in two and three dimensions is another closely related research topic. Given a point set in three dimensions, it is possible to reconstruct a convex object that includes these points by finding the convex hull of the points in the set. However, reconstructing a non-convex object from a point set is quite difficult and the resulting objects are not necessarily unique. It is known that even reconstruction of orthogonal shapes from points in two dimensions is a difficult problem [Rap86]. Does it help to know that the given points are the vertices of the object to be reconstructed?

1.2 Overview of the Thesis

In Chapter 2, I provide a comprehensive and modern definition of polyhedra that describes the objects I consider in this thesis. This definition must be accompanied by clear definitions of the vertices, edges and faces of a polyhedron. Even when we have a definition of polyhedra, it is not an easy task to formally describe one. What makes two polyhedra different? How do you describe one so that anybody can draw a picture of it, construct a model of it, or use it as the input of an algorithm? There are many approaches to doing this in the literature. I will review the most relevant ones in Chapter 2.

In Chapter 3, I study an approach of describing an orthogonal polyhedron or polyhedral surface based on its dual graph, dihedral angles, facial angles and edge lengths. In Section 3.1, I show that it is easy to reconstruct the vertex coordinates of

an orthogonal polyhedron or an orthogonal polyhedral surface from its dual graph, dihedral angles, facial angles and edge lengths in linear time, as long as the graph is connected. In Section 3.2, I show that for orthogonal genus-0 polyhedra and polyhedral surfaces, the dihedral angles can be reconstructed from the dual graph, facial angles and edge lengths in linear time, as long as the graph is connected. In Section 3.3, I show that the facial angles of an orthogonal genus-0 polyhedron can be reconstructed from its dual graph, dihedral angles and edge lengths.

In Chapter 4, I study reconstructing orthogonally convex polygons and polyhedra from their vertex coordinates. In Section 4.1.2, I show that the edges and faces of an orthogonally convex polyhedron can be reconstructed in polynomial time from its vertex coordinates. In Section 4.2.1, I show that the edges of a rotated orthogonally convex polygon can be reconstructed from its vertex coordinates in polynomial time. In Section 4.2.2, I show that the edges and faces of a rotated orthogonally convex polyhedron can be reconstructed in polynomial time from its vertex coordinates. In Section 4.3, I study reconstructing an orthogonal polygon or polyhedron from a point set which includes points other than the vertices of the polygon/polyhedron.

I conclude with directions for future research in Chapter 5.

Chapter 2

Definitions and Background

In this chapter, definitions that are used throughout the thesis and background related to this study are given. Some definitions are standard in the literature and given here for reference. Detailed definitions for concepts which have multiple accepted definitions in the literature are given to clarify our understanding of the concept. Basic knowledge of algebraic, geometric and topological terms and notations and a background in algorithms will be required for the rest of this thesis.

2.1 Definitions

Polygons are among the building blocks of computational geometry and are referred to many times in this document. Usually, a polygon is defined using its boundary which is a *polygonal curve*:

Definition 2.1 (Polygonal Curve) *A polygonal curve is a simple closed curve embedded in two dimensional Euclidean space*[1] *that consists of a finite number of line segments given in a fixed cyclic order.*

Here, the term simple implies:

1. The intersection of each pair of segments consecutive in the cyclic order is their shared endpoint.
2. Non-consecutive segments do not intersect.

[1] An n-dimensional Euclidean space $\mathbb{E}^n$ is an n-dimensional real coordinate space $\mathbb{R}^n$ with a Euclidean structure [Mun00] that defines inner product, norm and distance functions.

Note that each polygonal curve is embedded in two dimensions and therefore planar. The concept of non-planar polygonal curves exists in the literature but will not be used in this thesis. A polygon can now be defined using a polygonal curve as follows:

Definition 2.2 (Polygon) *A polygon P is a closed and bounded region of a plane in $\mathbb{E}^3$, whose boundary ∂P is a polygonal curve.*

A *vertex* of a polygon P is a point on ∂P where ∂P changes slope. An *edge* of polygon P is a line segment on its boundary that connects two vertices. Put differently, while a polygonal curve may contain co-linear segments, these segments are merged and their shared endpoints are removed to obtain the edges of the polygon.

The Jordan Curve Theorem (see e.g. [O'R98]) states that a simple closed curve in a plane separates the plane into two disjoint regions. The region bounded by ∂P is called the *interior*, P_{int}, and the other region is called the *exterior*, P_{ext}. For a polygon P, $P_{int} = P - \partial P$ and $P_{ext} = \bar{P}$, where $\bar{P}$ is the complement of P.

Since this thesis is mostly on orthogonal polygons and polyhedra, it is necessary to define an *orthogonal polygon* (also known as rectilinear polygon in the literature). The following definition of an orthogonal polygon is similar to the one found in [O'R98]:

Definition 2.3 (Orthogonal Polygon) *An orthogonal polygon is a polygon whose edges are axis-parallel.*

Naturally, all angles at vertices of an orthogonal polygon are multiples of 90°. However, this does not imply that a polygon whose angles are all multiples of 90° is orthogonal. If all angles of a polygon are multiples of 90°, but its edges are not axis-parallel, then call it a *rotated orthogonal polygon*.

Convexity and *rectilinear convexity* are important concepts used in this thesis. The following are the definitions of convexity and convex hulls in general. Soon, a definition of rectilinear convexity will be given.

Definition 2.4 (Convexity) *A set $S \in \mathbb{E}^3$ is convex if $x \in S$ and $y \in S$ implies that the line segment $[xy]$ is entirely in S.*

The concept of a *convex hull* is also crucial:

Definition 2.5 (Convex Hull) *The convex hull of a set $S \in \mathbb{E}^3$ is the minimal convex set that includes S.*

This definition of general convexity is adequate for general polygons, however a different definition for rectilinear convexity is necessary, since the only convex orthogonal polygon is the rectangle. One way of defining rectilinear convexity is by restricting the points chosen to test convexity. This definition is similar to the definition found in [OSSW84]. First define a *rectilinear line segment* as follows:

Definition 2.6 (Rectilinear Line Segment) *A line is called rectilinear if it is parallel to one of the coordinate axis. A line segment is rectilinear if it belongs to a rectilinear line. Two points determine a rectilinear line segment in d dimensions if the points have $d-1$ coordinates in common.*

Then, define rectilinear convexity using only points that determine a rectilinear line segment:

Definition 2.7 (Rectilinear Convexity) *A set S is called rectilinearly convex if for any two of its points that determine a rectilinear line segment, the line segment lies entirely in S.*

It is now possible to define an *orthogonally convex polygon* which is frequently used in this thesis:

Definition 2.8 (Orthogonally Convex Polygon) *An orthogonally convex polygon is an orthogonal polygon that is rectilinearly convex.*

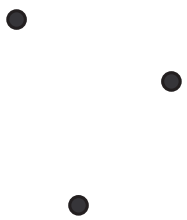

Figure 2.1: This set is disconnected but rectilinearly convex.

Note the difference between a rectilinearly convex polygon and an orthogonally convex polygon: a polygon may be rectilinearly convex even if it is not orthogonal.

Although the definition of rectilinear convexity allows disconnected sets to be rectilinearly convex (see Figure 2.1), the polygon definition prevents a disconnected region to be counted as a polygon. Therefore, an orthogonally convex polygon is always connected.

It is time to define a *polyhedron* which is an extension of polygons into three dimensions. However, giving a precise definition of a polyhedron is a challenging task. All geometers have an intuition of what a polyhedron is. Unfortunately, these intuitions do not always coincide. Imre Lakatos gives an excellent history of the discussions on polyhedral definitions in his book [Lak76]. Similarly, Cromwell outlines the concerns related to a definition of polyhedron [Cro97] inspired by the definition used by Coxeter [Cox73]. Let us first define a *polyhedral surface* and then use it to define a polyhedron, an approach similar to the one used to define a polygon using its boundary. Later, a discussion on these definitions is given inspired by the works of Lakatos and Cromwell [Lak76, Cro97].

Definition 2.9 (Polyhedral Surface) *A polyhedral surface is a finite set of interior disjoint polygons embedded in $\mathbb{E}^3$, such that the union of the polygons is a connected two-manifold.*

Here the term *manifold* represents an abstract space in which the neighborhood of every point is topologically homeomorphic to Euclidean space. In a *two-manifold*, the neighborhood of every point is topologically homeomorphic to a disk. I omit more precise definitions of the other topological terms, since their study is beyond the scope of this thesis. These definitions may be found in any introductory level topology book [Mun00].

I will shortly explain the implications of the terms in the polyhedral surface definition, but first define a polyhedron as follows:

Definition 2.10 (Polyhedron) *A polyhedron is a closed subset of $\mathbb{E}^3$ whose boundary can be expressed as a polyhedral surface.*

Now, I explain polyhedral surfaces, mostly by giving examples of sets that are not polyhedra. First consider two disjoint boxes (here, the term box represents the surface of a block). Is their union a polyhedral surface? Or consider a small box inside a large box, such that the boxes do not touch each other, so the small box is actually a *cavity* inside the large box. Do these two boxes define a polyhedral surface? See Figure 2.2 for a visualization of these two cases. In both cases,

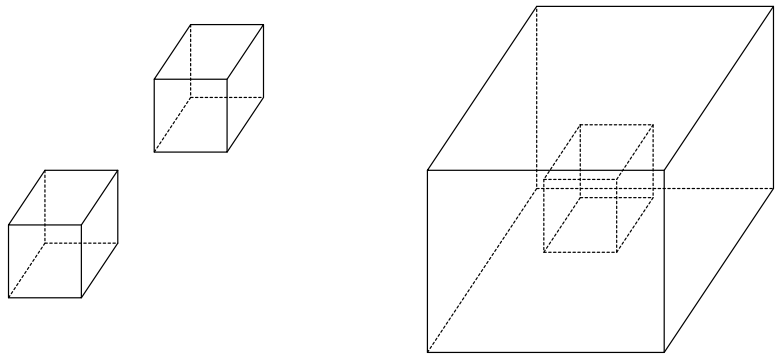

Figure 2.2: Two disconnected surfaces. Two boxes on the left, one box inside the other on the right.

the surface is not *connected* [2], i.e. it is *disconnected*. In this thesis, I require all polyhedral surfaces to be connected, since a disconnected surface can still be expressed as multiple connected surfaces whenever necessary.

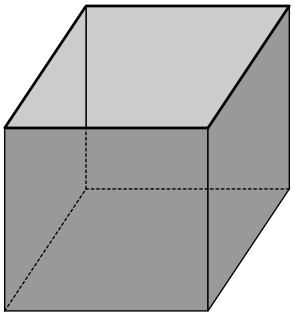

Figure 2.3: A box with its top face removed.

Now, consider a box with its top removed as in Figure 2.3. This surface has a boundary, shown as the bold rectangle in the figure. The surface "ends" at this boundary. Therefore, it cannot bound a three dimensional region of space and in particular cannot define a polyhedron. The polyhedral surface definition disallows such a boundary by requiring the surface to be a two-manifold. The neighborhood of the points on this boundary are not topologically homeomorphic to a disk.

What if the boundary of the surface is removed? Then the surface is called an *open* surface. For an open surface, the manifold requirement may hold, but such a surface cannot be used to define a polyhedron as it does not bound a three dimensional region of space. Again, the polyhedral surface definition disallows an open surface by requiring that a polyhedral surface is composed of a finite set of interior disjoint polygons. An open surface cannot be decomposed into a finite set

[2] A surface is connected if there exists a curve on the surface between any two points.

of polygons, because each polygon includes its boundary, i.e. it is *closed*. Infinitely many closed sets are required to cover an open set, but a polyhedral surface is required to be composed of a finite set of interior disjoint polygons.

Is a cylinder a polyhedral surface? A cylinder is a connected two-manifold and bounds a region of three dimensional space. A cylinder is curved, i.e., it has a continuously changing normal vector at parts of the surface. A curved surface cannot be composed of a finite set of interior disjoint polygons, hence it is not a polyhedral surface.

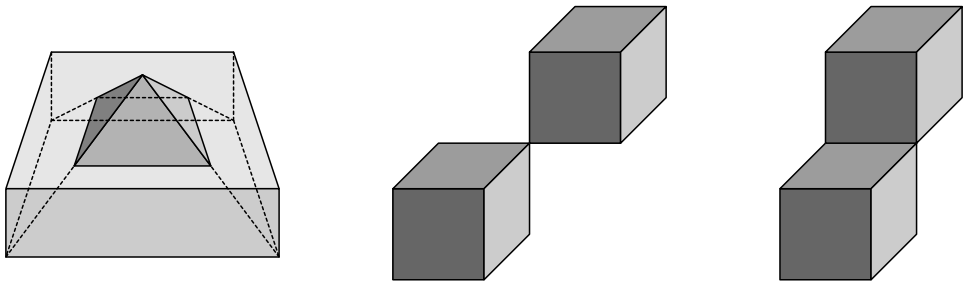

Figure 2.4: Three surfaces with non-manifold points.

Finally, consider the three surfaces in Figure 2.4. The first surface is self intersecting, the second surface has a *non-manifold vertex* and the third surface has a *non-manifold edge*. All three surfaces violate the manifold condition and are not accepted as polyhedral surfaces, both conventionally in the literature [Cro97, O'R98] and in this thesis.

The focus of this thesis is orthogonal polyhedral surfaces and orthogonal polyhedra. It is now a simple task to define them using the general definitions. But first, we need to define an orthogonal plane:

Definition 2.11 (Orthogonal Plane) *A plane in three dimensional Euclidean space is orthogonal if it is parallel to two coordinate axes.*

In other words, a plane is orthogonal if it is perpendicular to one of the coordinate axes.

Definition 2.12 (Orthogonal Polyhedral Surface) *An orthogonal polyhedral surface is a polyhedral surface composed of orthogonal polygons, each of which belongs to an orthogonal plane.*

Definition 2.13 (Orthogonal Polyhedron) *An orthogonal polyhedron is a polyhedron that can be bounded by an orthogonal polyhedral surface.*

Note that this definition demands only that an orthogonal polyhedron can be bounded by an orthogonal polyhedral surface. A polyhedron that is bounded by a non-orthogonal polyhedral surface could still be an orthogonal polyhedron.

Faces, *edges* and *vertices* are important elements of a polyhedral surface or polyhedron and their interrelations and properties may be used to describe a polyhedral surface. Therefore, their definitions are essential:

Definition 2.14 (Vertices of a Polyhedral Surface) *The set of vertices of a polyhedral surface is the set of points that are vertices of the polygons that the surface is composed of.*

Definition 2.15 (Edges of a Polyhedral Surface) *The set of edges of a polyhedral surface is the set of line segments on the boundary of the polygons that the surface is composed of, such that the ends of each edge are two vertices of the polyhedral surface, and no other vertex is contained in it.*

In particular, an edge of a polygon that includes three or more vertices of the polyhedral surface is not considered to be an edge of the polyhedral surface; it is the union of two or more edges of the polyhedral surface.

Definition 2.16 (Faces of a Polyhedral Surface) *The faces of a polyhedral surface are the interior disjoint polygons that the surface is composed of.*

The edges of a face are the edges of the polyhedral surface which are incident to the face. Note that as explained above these edges may be different from the edges of the polygon that defined the face. Similarly, the vertices of a face are the vertices of the polyhedral surface which are incident to the face. Therefore, each face of a polyhedral surface has a polygonal curve as its boundary.

The faces, edges and vertices of a polyhedral surface need not coincide with the faces, edges and vertices of the underlying polyhedron. Therefore, these are defined separately as follows:

Definition 2.17 (Faces of a Polyhedron) *The faces of a polyhedron are maximal, interior connected regions of the boundary of the polyhedron that lie within one plane and that can be expressed as the union of a finite set of polygons.*

The faces of a polyhedron are not necessarily polygons. This is discussed in detail after the definitions of edges and vertices.

Definition 2.18 (Vertices of a Polyhedron) *The vertices of a polyhedron are points on its boundary where three or more faces meet.*

Definition 2.19 (Edges of a Polyhedron) *The edges of a polyhedron are the line segments on the boundary of the polyhedron which are incident to exactly two vertices and two faces, such that the vertices are the endpoints of the line segment.*

The vertices and edges of a face of a polyhedron are the vertices and edges of the polyhedron that are incident to the face.

Two coplanar faces cannot share an edge, since the faces of a polyhedron are maximal. Therefore, the normals of the faces of a polyhedron change at each edge. This is not necessarily true for the edges of a polyhedral surface.

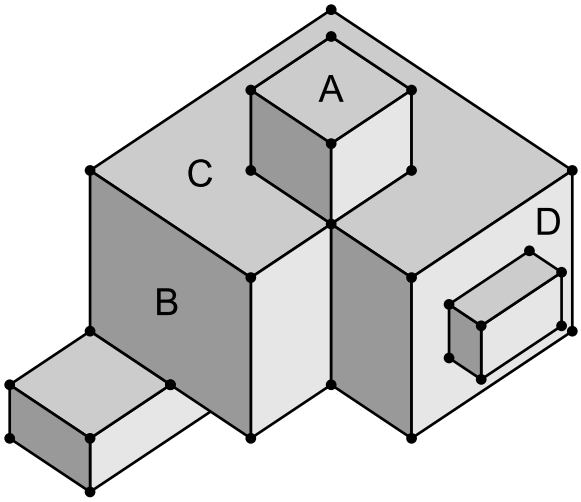

Figure 2.5: Faces of a polyhedron: A is a polygonal face, B is a face with collinear boundary edges, C is a self touching face and D is a face with a hole.

It was mentioned that the faces of a polyhedron are not necessarily polygons. This needs more clarification. Figure 2.5 shows four possible kinds of faces of a polyhedron. Face A is a polygonal face, i.e., a polygon. It can be described by the cyclic order of the vertices of its boundary. Face B has collinear edges on its boundary, and it can also be described by the cyclic order of the vertices of its boundary. Face C is a *self touching* face and is not a polygon: its boundary is not a polygonal curve. However, it can still be described by the cyclic order of the vertices of its boundary, such that the vertices where the face touches itself exist more than once in the cyclic order. Face D is a *face with holes* which means that its boundary is not connected. It is certainly not a polygon. A face with k

holes is a face with $k+1$ connected components on its boundary. One of these components enclose the others, call it the *outer boundary* of the face. The rest of the components are called the *hole boundaries* since they bound the holes of the face. Each component can be separately described by the cyclic order of its vertices.

One can show that given a polyhedral surface, the faces, edges and vertices of the underlying polyhedron can be obtained using the following approach:

1. Remove edges where two coplanar faces meet and merge the faces.
2. Remove vertices where less than three edges meet.

The *facial angles* and *dihedral angles* are important properties of a polyhedral surface or polyhedron:

Definition 2.20 (Facial Angle) *A facial angle is the interior angle between two consecutive edges of a face of a polyhedral surface or polyhedron.*

Definition 2.21 (Dihedral Angle) *A dihedral angle at an edge e is the interior angle between the two faces of a polyhedral surface or polyhedron adjacent to e.*

Call a dihedral angle a *flat dihedral angle* if its value is 180°, i.e. the corresponding faces are coplanar. A polyhedron does not have any flat dihedral angles, because no two of its adjacent faces are coplanar.

In an orthogonal polyhedral surface, all facial and dihedral angles are either $90°, 180°$ or 270°. This holds also for orthogonal polyhedra except that dihedral angles cannot be 180°. However, this property is not sufficient as a definition for orthogonal polyhedra, because it allows arbitrary rotations. If all facial and dihedral angles of a polyhedral surface are multiples of 90°, but the faces do not lie in orthogonal planes, then call this polyhedral surface a *rotated orthogonal polyhedral surface*. *Rotated orthogonal polyhedra* are defined similarly.

It is difficult to define convexity on orthogonal polyhedra. The general definition of convexity does not help with orthogonal polyhedra, because the only type of orthogonal polyhedron that is convex in the usual sense is the *block*, a solid object with six rectangular sides. Therefore, a special definition of convexity is necessary for orthogonal polyhedra.

One elegant and thorough approach is presented by Fink and Wood [FW96] for sets with restricted orientations. Next is a summary of their approach as a general guideline, followed by a simplified definition for orthogonal convexity.

Fink and Wood define a *restricted-orientation set* as "a set of points whose intersection with lines from some fixed set is empty or connected" [FW96]. In three dimensions, these lines are defined by pairwise intersections of some fixed set of planes. If this fixed set consists of the three orthogonal planes $P_x = \{x = 0\}, P_y = \{y = 0\}, P_z = \{z = 0\}$ then the convexity enforced by this set becomes a so-called *orthogonal restricted-orientation convexity*. However, as shown in Figure 2.6, a disconnected set as in 2.6(e) may be convex under this definition. Even more, the polyhedron in 2.6(d) is also convex, even though its intersection with some translations of the planes in the orientation set are not connected.

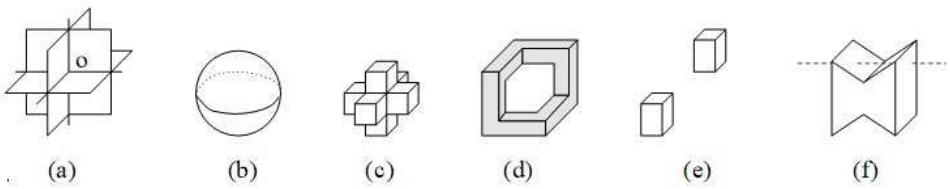

Figure 2.6: Restricted-orientation convexity in three dimensions. (a) is the orientation set; only (f) is not convex. Figure by Fink and Wood [FW96].

Similar to the two dimensional case of orthogonal polygons, the polyhedron definition does not admit disconnected surfaces as orthogonally convex. Fink and Wood define a *restricted-orientation connected set* as "a closed set whose intersection with every restricted-orientation flat is empty or path-connected; that is, every two points of the intersection can be connected by a path that is wholly contained in the intersection" to avoid objects like in Figure 2.6(d). For the orthogonal case, a restricted-orientation flat is parallel to one of the three orthogonal planes as shown in Figure 2.6(a).

Fink and Wood's definition can be restated more simply for the special case of orthogonal polyhedra as follows:

Definition 2.22 (Orthogonally Convex Polyhedron) *An orthogonally convex polyhedron is an orthogonal polyhedron whose intersection with every orthogonal plane is either empty or a single orthogonally convex polygon.*

Most of this thesis is concerned with polyhedra that are topologically homeomorphic to a ball, i.e., they have genus 0. A conventional way of defining genus of a polyhedral surface or a polyhedron in general is to use *non-separating curves*:

Definition 2.23 (Non-separating Curve) *A closed curve embedded in a polyhedral surface is non-separating if the surface remains in one piece when the surface is cut along the curve [Cro97].*

A sphere has no non-separating curve because any closed curve on the surface of a sphere separates it into two pieces as shown in Figure 2.7. However, a torus has non-separating curves.

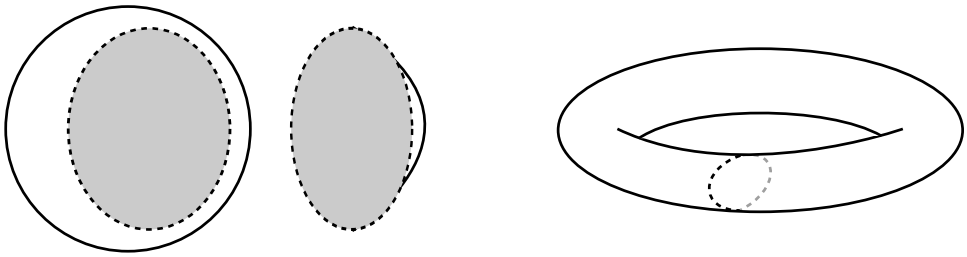

Figure 2.7: A sphere is separated into two pieces when cut along any closed curve on its surface, but not a torus.

The genus of a polyhedral surface is defined using non-separating curves as follows (our definition is similar to the definitions used by Hoppe [Hop79] and O'Rourke [O'R98]):

Definition 2.24 (Genus of a Polyhedral Surface) *The genus of a polyhedral surface is the maximum number of disjoint non-separating curves, such that the surface remains in one piece if cut along all curves.*

If the genus of a polyhedral surface is i, then it is called a *genus-i* polyhedral surface. Usually, if the genus of a polyhedral surface is not 0, then it is called a *higher-genus* polyhedral surface. The *genus of a polyhedron* is determined by the genus of a polyhedral surface that bounds it. This is well-defined because two polyhedral surfaces that bound the same polyhedron may have different faces, edges and vertices, but the same genus, because they describe the same surface.

In the following chapters graph terminology is used frequently. Therefore, it is necessary to provide some basic definitions related to graphs. Most of the following is standard terminology [Die05]:

Definition 2.25 (Graph, Node, Arc) *A graph $G = (N, A)$ consists of a set N of nodes and a set A of arcs; an arc is an unordered pair of elements of N.*

Normally, the elements of a graph are called the vertices and edges of the graph. However, the terms vertex and edge are already used as properties of the polyhedron and its surface. Using them once again for graphs would possibly cause confusion for the reader. Therefore, the terms node and arc, which are normally used in directed graphs, are used for all graphs in this thesis.

A *path* between two nodes v_0 to v_k of a graph G is a sequence of nodes $v_0, \ldots, v_k$ such that there exists an arc between v_{i-1} and v_i. A path is a *simple path* if its nodes are distinct. A *cycle* is a path such that there is an arc between the first and the last node. A cycle is simple if each node exists in it only once.

A graph G is connected if there is a path from any node to any other node in the graph; otherwise it is disconnected.

If a graph G is drawn on a surface S such that two arcs intersect only at a node which is an end node of both, then G is said to be topologically *embedded* in S. Here, drawing a graph on a surface means that the nodes of the graph are represented as points on the surface, and the arcs of the graph are represented as curves on the surface connecting the points representing the nodes of the arc. The embedding divides the surface into distinct regions, and each region is called a *face* of the embedding. A graph is *planar* if it has an embedding in a plane (or equivalently on the surface of a sphere).

The cycle separating a face of an embedded graph from the rest of the surface is called the *boundary* of that face. If the boundary of a face in one embedding always corresponds to the boundary of a face in the other then the two embeddings are said to be *equivalent* [HPR93]. If all embeddings of a graph on a surface are equivalent, then the embedding is called *unique*.

2.2 Graphs of Polyhedral Surfaces and Polyhedra

The faces, edges and vertices of a polyhedral surface or polyhedron are closely interrelated. Each face is described by a boundary and each boundary is described by its edges. The edges of the boundary are described by their end vertices. Two faces meet at each edge and at least two edges and faces meet at each vertex. When two faces share an edge, they are called *adjacent* faces. When two edges meet at a vertex, they are called *adjacent* edges. If two edges are next to each other in the cyclic order of edges of a face boundary, they are said to be *consecutive* on that boundary. If an edge or a vertex is on the boundary of a face, then the face is

incident to that edge or vertex. If a vertex is the end vertex of an edge, then the edge is *incident* to the vertex. Altogether, these relations describe the *topological structure* of a polyhedral surface or polyhedron [SSP00].

One way to represent this topological structure is to represent it as a graph. There are different graphs that one can obtain from the incidence and adjacency relations of the polyhedral surface or polyhedron. In this thesis, two of these graphs are mentioned: the edge-vertex graph [Han82, O'R98] and the edge-face graph [ADF85, HPR93]. The edge-vertex graph of a polyhedral surface or polyhedron is presented next. Then, the edge-face graph is discussed in detail.

2.2.1 Edge-Vertex Graph of a Polyhedral Surface

Definition 2.26 (Edge-Vertex Graph of a Polyhedral Surface) *A graph G is the edge-vertex graph of a polyhedral surface S, if for each vertex of S there is a corresponding node in G and for each edge e in S there is a corresponding arc r in G, such that the end nodes of r are the corresponding nodes of end vertices of e.*

Although the edge-vertex graph of a polyhedral surface reveals the incidence relations between the edges and vertices of a surface, it fails to capture any information about the faces. In order to extract this information from the edge-vertex graph, one needs an embedding of this graph [Han82] on a surface which itself is embedded in three dimensional Euclidean space. As long as the graph is connected, one can describe an embedding for it by fixing the cyclic order of incident arcs at each node, such that consecutive edges of a face are consecutive in the cyclic order of the node corresponding to their common end point. Then, one can easily compute the faces of the embedding from this order.

It is important that one establishes a fixed way of determining the order of incident edges at each vertex, because it is possible to embed a graph in a surface in multiple ways. In this thesis, this order is the clockwise cyclic order of the edges around a vertex as observed from outside the polyhedral surface.

A polyhedral surface does not have any faces with holes, and one can show that therefore the edge-vertex graph of a polyhedral surface is connected.

Throughout the thesis, the term *graph of the polyhedral surface* is used to refer to the edge-vertex graph of a polyhedral surface along with the fixed cyclic order of the arcs around the nodes as described above.

2.2.2 Edge-Vertex Graph of a Polyhedron

The edge-vertex graph of a polyhedron is defined in the same way to the edge-vertex graph of its surface. But, there is one major difference: The edge-vertex graph of a polyhedron can be disconnected. It is still possible to determine the cyclic order of the arcs around each node from the polyhedral surface and provide an embedding for each connected component. However, there is no information that describes how the connected components are related to each other in this embedding.

It is possible to assume that some extra information is known to relate each connected component in the embedding. A better solution is to use an alternative graph which is always connected, admits a single fixed embedding and implicitly stores the extra information. This graph is the edge-face graph and is discussed in Sections 2.2.3 and 2.2.4.

Throughout the thesis, the term *graph of the polyhedron* is used to refer to the edge-vertex graph of a polyhedron along with the fixed cyclic order of the arcs around the nodes.

2.2.3 Edge-Face Graph of a Polyhedral Surface

Definition 2.27 (Edge-Face Graph of a Polyhedral Surface) *A graph G is the edge-face graph of a polyhedral surface S, if for each face in S there is a corresponding node in G and for each edge e in S there is a corresponding arc r in G, such that the end nodes of r are the corresponding nodes of the two faces incident to e.*

The edge-vertex graph of a polyhedral surface reveals the incidence relations between the vertices and edges of a polyhedral surface. The edge-face graph of a polyhedral surface reveals the incidence relations between the edges and faces of a polyhedral surface. Therefore, it does not have any direct information related to the vertices of the surface. This information can be extracted from the graph if an embedding of the graph is known. One can describe an embedding for an edge-face graph by fixing a cyclic order of the incident arcs of each node. In this thesis, this order is the counterclockwise cyclic order of the corresponding edges incident to the corresponding face on the surface. The edge-face graph of a polyhedral surface is always connected.

Note that the edge-face graph of a polyhedral surface is the dual of the edge-vertex graph of the polyhedral surface with the fixed embedding [HPR93]. In

other words, one can obtain the edge-face graph of a polyhedral surface from the embedding of its edge-vertex graph by creating a node in the edge-face graph for each face of the edge-vertex graph and an arc between two nodes of the edge-face graph for every edge shared by the faces corresponding to these nodes in the edge-vertex graph. Then, the embedding of the edge-vertex graph naturally implies an embedding on the edge-face graph, which is exactly the one explained above.

Throughout the thesis, the term *dual graph of the polyhedral surface* is used to refer to the edge-face graph of a polyhedral surface along with a fixed cyclic order of the arcs around the nodes as described above.

2.2.4 Edge-Face Graph of a Polyhedron

The edge-face graph of a polyhedron is defined in the same way as the edge-face graph of its surface. However, when the graph of the polyhedron is not connected, a small modification in the representation of the embedding is necessary.

Consider a polyhedron that has a face with multiple holes. The graph of this polyhedron may be disconnected because of the multiple disconnected boundaries of this face. Now, consider the edge-face graph of this polyhedron. An embedding of the edge-face graph must determine a fixed cyclic order of the incident edges of this face. One can easily do this for the outer boundary of the face. However, the hole boundary edges have no consecutiveness relation with the outer boundary edges. Therefore, they cannot be included in the same cyclic order. A simple way to deal with this is to represent each boundary of a face with holes by a separate cyclic order of its edges.

For an example, consider the face with two holes in Figure 2.8. Usually, graphs are represented using adjacency lists, such that each node has a list of arcs that are incident to it. The cyclic order of arcs incident to a node can be represented trivially using this data structure. To modify an adjacency list to represent a face with holes, assume that each node has multiple disjoint lists of arcs that are incident to it. The face in Figure 2.8 can be represented by $[\{abcdefghij\}, \{ponmlk\}, \{xwvutsrq\}]$. The three separate adjacency lists tell that the face has three boundaries. The first list is always the outer boundary. Then the second and the third are the hole boundaries. The order of the hole boundaries is not important, since it does not imply anything. The cyclic order of both the outer boundary and the hole boundaries are counter-clockwise according to the face. This ensures that the face is always to the left of the edges when traversing the boundaries.

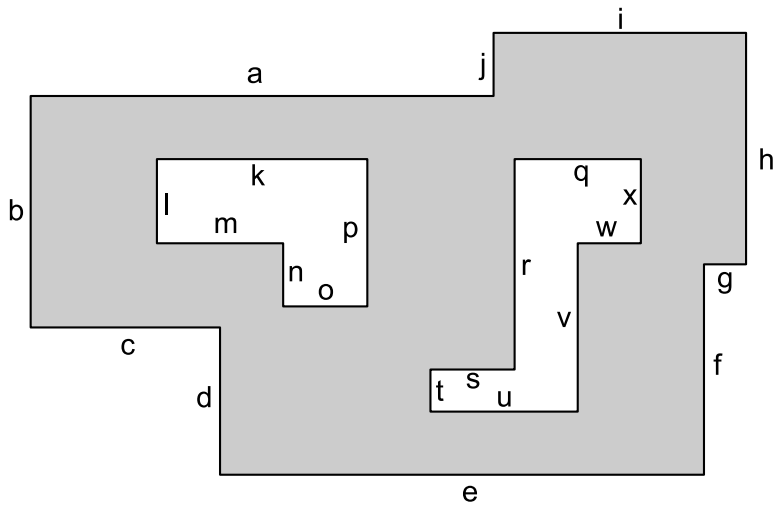

Figure 2.8: A face with two holes. The label of an edge represents the other face that shares the edge.

Using multiple adjacency lists, one can keep track of separate boundaries of faces and still know how connected components of the edge-vertex graph of the polyhedron are related to each other.

Throughout the thesis, the term *dual graph of the polyhedron* is used to refer to the edge-face graph of a polyhedron along with a fixed cyclic order of the edges of each boundary of each face of the polyhedron as described above.

2.3 Boundary Representations

The graph structures discussed in the previous section describe only the relations between vertices, edges and faces, but give no geometric information where they are located. In this section, I describe in detail a *boundary representation*, which adds geometric information. Boundary representations are frequently used in existing studies [BHS80, Req80, Sil81, MS82, AA97, SSP00] for describing polyhedra using different topological and geometric information. I will also refer to boundary representations frequently in this thesis.

A boundary representation *describes* a polyhedral surface or a polyhedron using topological and geometric information. In other words, given the boundary representation of a polyhedron P, one can distinguish P from any other polyhedra. The topological information consists of the incidence and adjacency relations of the vertices, edges and faces. This is essentially the edge-vertex graph and the edge-face graph, but a more common approach in the literature is to represent this topological information in a tree structure. The root of this tree is the object to be described. The children of the root are the faces of the object. The children of the faces are the edges of the faces and the children of the edges are their end vertices.

The most common geometric information to be included in a boundary representation is the coordinates of the vertices. When the coordinates of the vertices are known, one can uniquely describe each edge using its two end points. Here, uniquely means that in all possible realizations of this polyhedral surface or polyhedron in three dimensional Euclidean space, the edges coincide. Then, the edges uniquely describe the faces and the faces uniquely describe the polyhedral surface or the polyhedron.

Alternative ways of representing both the topological and geometric information exist. In this thesis, the dual graph of a polyhedral surface or polyhedron is used to represent the topological information. This information is more powerful than the usual tree structure, because the latter uses one directional incidence relations. For example, computing the incident faces of a vertex is quite time consuming with a tree structure, but straightforward with the dual graph.

Although vertex coordinates are a practical choice for describing geometric properties of a polyhedral surface or polyhedron, other geometric properties may also be used to describe a polyhedral surface or polyhedron. Some of these geometric properties are the dihedral angles, facial angles and edge lengths. A major part of this thesis (Chapter 3) deals exactly with the problem of which of these properties

are enough to reconstruct the vertex coordinates, especially for orthogonal polyhedral surfaces and orthogonal polyhedra. Other researchers studied alternative properties; these results are presented in Section 2.4.

Whether a given set of topological and geometric information actually corresponds to a valid polyhedral surface or polyhedron is an important question. For the tree structure and vertex coordinates, Sakkalis et al. [SSP00] studied necessary and sufficient conditions for validity, which result in a polynomial time algorithm to test validity. In this thesis, I present some necessary conditions, but they are not necessarily sufficient. One needs to apply tests as described by Sakkalis et al. [SSP00] to ensure that the obtained boundary representation is valid.

2.4 Related Studies

Apart from boundary representations, there are many other ways to describe a polyhedron. They can be grouped into two approaches: describing the surface or describing the volume. Researchers have been divided among these two approaches throughout history depending on their understanding of polyhedra. Dürer, Kepler, Cauchy and Euler described a polyhedron using its surface, while Plato, Archimedes, Descartes and L'Huilier described a polyhedron as a solid [Cro97]. In modern science, polyhedra are used in practical applications as much as they are used in theoretical studies [Req80, Whi94]. Practical applications of polyhedra, such as describing rigid objects in manufacturing environments, benefit from the solid based descriptions of polyhedra, whereas theoretical studies, such as Minkowski's and Steinitz' theorems [Gru67], focus on its surface descriptions.

Here, I review existing studies on describing a polyhedral surface or a polyhedron. Since the focus of this thesis is surface properties of a polyhedron rather than the solid properties, I start with surface descriptions in detail and only outline a few solid descriptions.

2.4.1 Surface Descriptions

Most surface descriptions are a modification of boundary representations. One intuitive approach to describe a polyhedron is via its *wireframe* [Han82], which is the edge-vertex graph of the polyhedron along with vertex coordinates. This is hence the boundary representation, but omits faces and their incidence relations with edges and vertices.

Wireframes are known to be *ambiguous* [Man88, Req80], i.e., a given wireframe could correspond to multiple polyhedra. Figure 2.9 shows two different polyhedra with the same wireframe. For orthogonal polyhedra, O'Rourke [O'R88] shows that the wireframe uniquely describes the faces of the polyhedron.

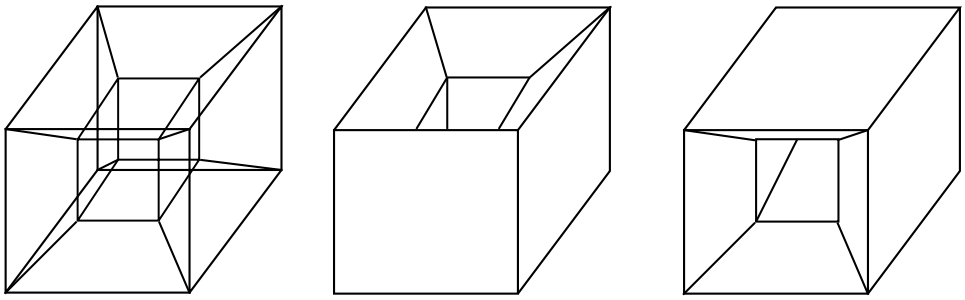

Figure 2.9: A wireframe, and two corresponding genus-1 polyhedra.

Hanrahan [Han82] studies wireframes in detail. For genus-0 polyhedra he uses Whitney's Unique Embeddability theorem [Whi32], that is "if a planar graph is triconnected, then it has a unique embedding", and shows that a wireframe describes a unique polyhedron if it has a triconnected [3] edge-vertex graph. He also provides a linear time algorithm for reconstructing the faces of a polyhedron from its edge-vertex graph.

Heath, Paripati and Roach [HPR93] studies an alternative where instead of using the edge-vertex graph of the polyhedron, they use the spherical dual of the polyhedron. The *spherical dual representation* (SDR for short) of a polyhedron consists of its edge-face graph and the plane equation for each face. For genus-0 polyhedra, they show that the SDR describes a unique polyhedron. For higher genus polyhedra, they show that the SDR describes a unique polyhedron if all faces of the polyhedron have four incident edges.

There are also approaches to describing a polyhedron which do not use a graph structure at all. Lucier [Luc06] studies the following question in his thesis: "Do the edge lengths and edge directions describe a unique polyhedron?" He separated this into two sub-problems: edges with direction and length information, and edges with length information only. Unfortunately, his results show that for most cases it is NP-hard to determine whether the input describes a valid polyhedral surface.

Another interesting and well-known approach to describing a polyhedron, which also has uses in computer vision, is based on silhouettes or perspective projections

[3] A graph is *triconnected* if it has no pair of nodes whose removal disconnects the graph.

of a polyhedron onto a plane instead of using the actual coordinates in three dimensions. A silhouette according to Laurentini [Lau97] is a two dimensional image of an object, which contains the perspective projections of the visible points of that object from a fixed viewpoint. He addresses the problem of finding the theoretical minimum number of silhouettes necessary for describing a unique object. He shows that only objects coincident with their visual hulls are describable using silhouettes [Lau97].

Unlike Laurentini's silhouettes, each of which is a polygon, Whiteley [Whi94] makes use of the orthogonal projections of the vertices and edges of the polyhedra, which he calls a *picture*. The question he studies is "Which pictures, with their given incidence structure, can be lifted back into space, so that distinct faces lie in distinct planes?" Here, the incidence structure is the adjacency relation between the vertices and their incident faces. The picture also includes the projected coordinates for each vertex. Lifting this picture into space means computing the z-coordinates of the vertices and the equations of the planes of the faces, such that the incidence structure holds true.

Whiteley's approach in its simple form does not necessarily describe a unique polyhedron, not even for convex polyhedra. Whiteley further explains how one can add other geometric properties to describe a unique polyhedron, such as dihedral angles, facial angles and edge lengths.

Some researchers [Dek95, BGRT99, MT04] considered describing a polyhedron using two distinct triangulations of a simple polygon, such that the polygon is the projection boundary and the triangulations correspond to two projections from the z-axis.

Among different types of projection information, line drawings are one of the more common ones to be studied by different researchers [MW80, WM81, NG88, Pen89, Sug84, Sug86, WG93, YCT94]. A line drawing is a two dimensional diagram composed of a finite number of straight line segments [Sug86]. For a good overview on describing a polyhedron using projections see also Hasan [Has05].

Franklin [Fra87] shows that given the vertex coordinates and the rays representing the incident edges at each vertex, one can compute certain properties of a polyhedron, such as total edge length and volume. It is also possible to test whether a given point lies inside a polyhedron using this description. However, Franklin does not mention whether the given information can be used to compute a boundary representation of the polyhedron.

Surface Descriptions of Orthogonal Polyhedra

All the approaches discussed above can be used for describing an orthogonal polyhedron. Moreover, there are also studies that only deal with orthogonal polyhedra. Here, I outline the most important ones.

O'Rourke [O'R88] studies wireframes for orthogonal polyhedra. He shows that the edges and vertex coordinates of an orthogonal polyhedron describes a unique set of faces, which is not always possible for general polyhedra. He also provides an algorithm to compute the faces in $O(n \log n)$ time.

The *Extreme Vertices Model* (or *EVM* for short) proposed by Aguilera and Ayala [AA97] uses the *extreme vertices* of a polyhedron, which are the degree-3 vertices, i.e. the vertices with three incident edges[4]. This approach can also be used to describe an orthogonal "pseudo-polyhedron", which is a connected union of cubes that is not necessarily a manifold (see Figure 2.10). Given the extreme vertices and their coordinates, they show how to reconstruct a unique orthogonal pseudo-polyhedron in quadratic worst case time. This is closely related to Chapter 4 of this thesis, where I study the related problem of reconstructing orthogonal polyhedra from all their vertices (with given coordinates), as well as extra vertices on the boundary and inside the polyhedron.

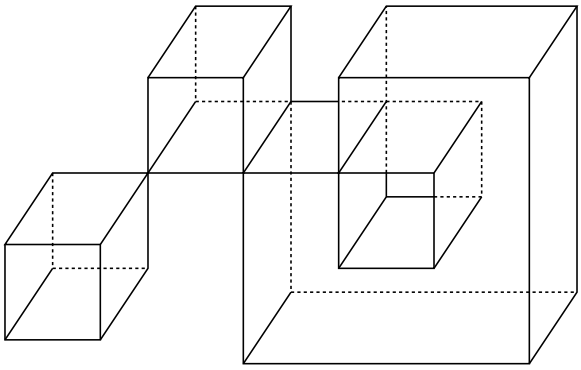

Figure 2.10: An orthogonal pseudo-polyhedron. Image from Aguilera [AA97].

Bournez, Maler and Pnueli [BMP99] also study extreme vertices of an orthogonal polyhedron. They show that using the coordinates of the extreme vertices of a polyhedron, one can efficiently implement membership, face-detection and Boolean operations on orthogonal polyhedra. However, they do not provide an algorithm for reconstructing the boundary representation from the input vertices.

[4]For a more formal definition of extreme vertices refer to [AA97].

2.4.2 Solid Descriptions

Most descriptions of polyhedra as solids are variants of *Constructive Solid Geometry* (CSG for short). Since my focus is on orthogonal polyhedra, I will define CSG and outline variants only for orthogonal polyhedra.

Constructive Solid Geometry [RV77] describes a polyhedron as a composition of simple solids. A CSG description of an orthogonal polyhedron uses only axis-parallel blocks. It combines blocks with boolean set operators and allows translation and scaling of each block.

Spatial Occupancy Enumeration is a CSG approach to describe an orthogonal polyhedron using disjoint fixed size cubes, such that two cubes either share an edge or vertex, or attach to each other at one face. This description is sometimes referred as *voxelization* and each cube is then referred as a *voxel*. Another name for this structure used in texture mapping is *polycubes* [THCM04].

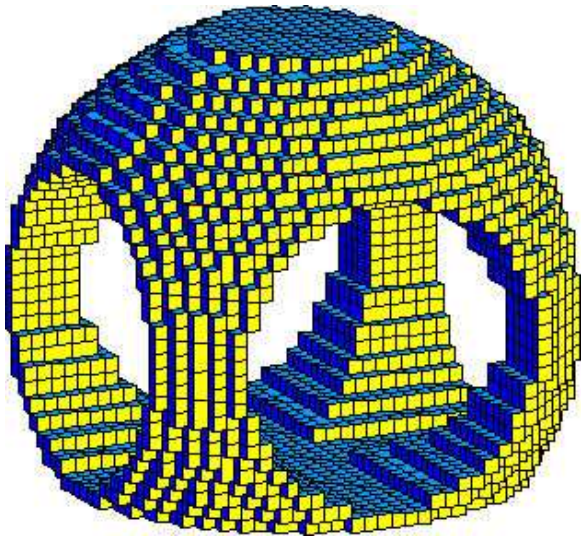

Figure 2.11: Spatial Occupancy Enumeration. Image from Aguilera [AA98]

This approach to describing an orthogonal polyhedron uses a lot of memory and cannot describe orthogonal polyhedra with non-integral vertex coordinates.

An *Octree* describes an orthogonal polyhedra by dividing the minimum axis-aligned bounding box of the polyhedron into 8 equal disjoint boxes that are called *octants*. Each octant then recursively describes the parts of the orthogonal polyhedron that it intersects. In other words, each octant is another Octree. If one of the octants is full or empty, then it is not divided any further. This recursive structure has a fixed depth, which determines the accuracy of the description. If

the depth is not large enough, then an Octree fails to describe an orthogonal polyhedron precisely. In order to overcome the accuracy issues related to Octrees, some alternative descriptions have been proposed [ABJN85, BN85, Nav89].

Chapter 3

Reconstruction from Dual Graphs

In this chapter, I study reconstruction of the boundary representation of an orthogonal polyhedron or a polyhedral surface from its dual graph accompanied by the dihedral angles, the facial angles and the edge lengths information.

In Section 2.3, the boundary representation was presented, which includes the incidence relations between the elements (faces, edges and vertices) of the polyhedral surface or the polyhedron, and the coordinates of the vertices. This information uniquely describes the vertices, edges and faces of the polyhedral surface or the polyhedron. Therefore, throughout this chapter the goal is to reconstruct the vertex coordinates using the given input.

It is worth mentioning that verification of the resulting boundary representation is not a concern in this thesis. Methods for verifying a given boundary representation are provided in detail by Shen et al. [SSP00]. In all algorithms presented in this thesis, if there is a possibility of checking for an invalid input, then this possibility is mentioned. These checks present necessary conditions for a valid input, but they are not necessarily sufficient to say that the output is a valid boundary representation of a polyhedral surface of polyhedron.

I start in Section 3.1 by showing that the coordinates of the vertices of an orthogonal polyhedron or polyhedral surface can be reconstructed from its dual graph, dihedral angles, facial angles and edge lengths in linear time. Then, the vertex coordinates along with the dual graph provide a boundary representation of an orthogonal polyhedron or polyhedral surface.

In Section 3.2, I show that the dihedral angles of an orthogonal polyhedron whose graph is connected or an orthogonal polyhedral surface can be reconstructed in linear time from its dual graph and facial angles. After computing the dihedral

angles one can reconstruct the vertex coordinates if the edge lengths are known using the algorithm of Section 3.1. Again, this results in a boundary representation of an orthogonal polyhedron or polyhedral surface.

In Section 3.3, I show that the facial angles of an orthogonal polyhedron can be reconstructed in linear time from its dual graph, dihedral angles and edge lengths. Computing the facial angles of an orthogonal polyhedral surface is also possible with some restrictions which I discuss in detail in Section 3.3.2. Finally, I show a simple linear time algorithm for computing the facial angles of a general polyhedron, if all vertices of the polyhedron have three incident edges. In all cases, once the facial angles are reconstructed, one can apply the algorithm of Section 3.1 to reconstruct the vertex coordinates and obtain the boundary representation of a polyhedron.

3.1 Reconstructing Vertex Coordinates

In this section, an $O(m)$ time algorithm for computing the vertex coordinates of an orthogonal polyhedron or polyhedral surface from its dual graph, facial angles, dihedral angles and edge lengths is given. The coordinates reconstructed from this information are unique up to translation and rotation, i.e., one can translate the polyhedron or the polyhedral surface to an arbitrary location or rotate it by a multiple of 90° around any coordinate axis and obtain another valid solution.

In Section 3.1.1, an algorithm is given for computing the vertex coordinates of an orthogonal polyhedral surface. In Section 3.1.2, I show that this algorithm also works for computing the vertex coordinates of an orthogonal polyhedron if some additional information for each connected component is known. In Section 3.1.3, a generalization of this algorithm to general polyhedra and polyhedral surfaces is given. In all cases, the presented algorithms run in $O(m)$ time. The orthogonal computations can be done with finite precision arithmetic. However, the computations for general polyhedra and polyhedral surfaces require an unlimited amount of precision due to involved multiplications with real numbers. Therefore, in Section 3.1.3 all operations are done using real RAM model which is theoretically capable of storing an unlimited amount of precision per number.

3.1.1 Reconstructing Vertex Coordinates From Dual Graph of the Polyhedral Surface

In this section, I outline a linear time algorithm for computing the vertex coordinates of an orthogonal polyhedral surface from its dual graph, dihedral angles, facial angles and edge lengths. Note that the coordinates are reconstructed up to translation and rotation.The algorithm is straightforward and I will give an outline and omit the trivial details.

The algorithm consists of the following steps:

1. Let F be an arbitrary face and set its face normal parallel to the x-axis. Let e be an edge of F. Set its edge direction to be parallel to the y-axis.
2. Set one of the endpoints of e as the origin.
3. Compute the coordinates of the other vertices of F using the facial angles and the edge lengths.
4. Let F' be an adjacent face with unknown coordinates. Use the dihedral angle between F and F' to compute the normal of F' from the normal of F.
5. Let $F = F'$. Go to the third step and repeat in a depth-first manner until all coordinates are reconstructed.

If the provided data corresponds to a valid polyhedral surface, then the dual graph is connected and all faces of the surface are reached exactly once at step 3. Since all edges of an orthogonal polyhedral surface are parallel to one of the coordinate axes, only a single coordinate value changes between two consecutive vertices of a face. The amount of change is equal to the length of the edge connecting the two vertices. Therefore, one can reconstruct the coordinates of a vertex with a single addition operation. Also, an orthogonal polyhedral surface has six different types of face normals, so step 4 requires constant time per edge and does not require any high precision calculations.

It is possible to detect some invalid input during the computation of the vertex coordinates. In step 3, one can compute the coordinates of the initial vertex using the last edge and check whether the computed coordinates coincide with the initial coordinates or not. Moreover, the coordinates of each vertex are computed for each of its incident faces. Unless all computed coordinates coincide for the same

vertex, either the dihedral angles, facial angles or the edge lengths are invalid. This algorithm processes each edge twice, once for each face sharing the edge. Therefore, the overall running time of the algorithm is $O(m)$.

Theorem 3.1 *The coordinates of the vertices of an orthogonal polyhedral surface can be reconstructed from its dual graph, dihedral angles, facial angles and edge lengths in $O(m)$ time up to translation and rotation.*

After reconstructing the coordinates of the vertices, the necessary information for a boundary representation of the polyhedral surface is obtained. This boundary representation is unique up to rotation and translation.

The translation and rotation can be fixed with extra information about the polyhedral surface. The following minimal set of information is an example that can be used to fix the translation and the rotation:

- The face normal of an arbitrary face F,
- The direction vector of an edge on the outer boundary of F, and
- The coordinates of one of the vertices.

Normally, the reconstructed polyhedral surface can be rotated around the coordinate axis by a multiple of 90°, and it can be translated to an arbitrary location. A known face normal disallows rotations around two coordinate axis, so the polyhedral surface can only be rotated around the axis parallel to the known face normal. The direction vector of an edge of the same face disallows this remaining rotational freedom. The three known coordinate values of one of the vertices fix the location of the polyhedral surface. The rotation and translation can be fixed in linear time by applying the same rotation and translation to all vertices after their coordinates are reconstructed as described in this section.

3.1.2 Reconstructing Vertex Coordinates From Dual Graph of the Polyhedron

Recall that Section 2.1 discussed the differences between the faces of a polyhedral surface and a polyhedron. Basically, all faces of a polyhedral surface are bounded by polygonal curves, but the faces of a polyhedron may be self touching and may

have holes. The existence of faces with holes depends on the connectedness of the edge-vertex graph (see Section 2.2) and is not an issue for polyhedral surfaces where the edge-vertex graph is always connected. But self touching faces may occur in both connected and disconnected graphs. The following is a discussion of changes required in the algorithm of Section 3.1.1 for connected and disconnected graphs of orthogonal polyhedra.

Graph of the Polyhedron is Connected

In Section 2.1, representations of self touching faces of an orthogonal polyhedron were presented. This representation treats the boundary of a self touching face as a closed and connected set of edges with a fixed cyclic order. Note that the algorithm presented in Section 3.1.1 does not require a face to be a polygon. As long as each face boundary is closed and connected and a cyclic order of the vertices along with the edge lengths between the consecutive vertices is present, one can use the same algorithm to reconstruct the coordinates of the vertices of an orthogonal polyhedron whose graph is connected.

Theorem 3.2 *The coordinates of the vertices of an orthogonal polyhedron whose graph is connected can be reconstructed from its dual graph, dihedral angles, facial angles and edge lengths in $O(m)$ time up to translation and rotation.*

Graph of the Polyhedron is Disconnected

If the graph of the polyhedron is disconnected, then there exist faces with multiple boundaries. One of these boundaries is the outer boundary and the rest are hole boundaries. Without further information, one can translate (without causing intersections with the outer boundary or other hole boundaries) and rotate (a multiple of 90°) the hole boundaries inside the outer boundary. To fix both the rotation and the translation of each hole boundary one needs additional information, for example the following per boundary:

- The relative coordinates of a vertex on the hole boundary.
- The relative direction of an edge incident to that vertex.

Here, relative coordinates mean the difference of coordinates of a vertex on the outer boundary of the face and a vertex on the hole boundary. For an orthogonal

polyhedron, this corresponds to a difference in two coordinate values as the third coordinate value is constant in each face (the constant coordinate value can be obtained from the other vertices of the outer boundary). Similarly, the relative edge direction is the angular difference between the direction vector of an edge of the outer boundary and an edge of the hole boundary, each incident to one of the vertices described above. The two edges are either parallel or perpendicular, and knowing which is sufficient to propagate the computation to the hole boundary (or to the outer boundary).

Using this information, one can compute the vertex coordinates of a hole boundary based on the vertex coordinates of the outer boundary, or vice versa. I omit implementation specific details here. Once the coordinate information is propagated from one boundary to another, the algorithm proceeds separately for each connected component of the graph of the polyhedron.

Theorem 3.3 *The coordinates of the vertices of an orthogonal polyhedron can be reconstructed from its dual graph, dihedral angles, facial angles, edge lengths and relative translation and rotation information per each connected component of the graph of the polyhedron in $O(m)$ time up to translation and rotation.*

3.1.3 General Polyhedra and Polyhedral Surfaces

It is worth mentioning that our results so far can be extended to general polyhedra and polyhedral surfaces, if operations under the real RAM model with access to infinite precision real number arithmetic is allowed. In this section, the necessary modifications to the algorithm of Section 3.1.2 are outlined to reconstruct the vertex coordinates of a polyhedron from its dual graph, dihedral angles, facial angles and edge lengths. I only present the algorithm for polyhedra with disconnected graphs which includes more challenges. Computation of vertex coordinates for polyhedral surfaces as well as polyhedra with connected graphs extends similarly.

The most closely related work in the literature is [BLS05], where Biedl, Lubiw and Sun show that reconstruction from the net of a polyhedron can be done in polynomial time. However, they assume polygonal faces.

There are three important challenges that need to considered. The first one is that simple addition operations cannot be used to reconstruct the coordinates of the vertices within a face, since two consecutive edges of a face of a general polyhedron or polyhedral surface may admit an arbitrary real value as their facial

angle. The second challenge is that the face normal of a face cannot be as easily determined from the face normal of an adjacent face and the dihedral angle in between, since two adjacent faces may admit an arbitrary real value as the dihedral angle. The third challenge is that the relative edge direction defined between the outer boundary and the hole boundary of a face with holes can have an arbitrary real value.

Recall that the first step is to fix a face normal N for an arbitrary face F. Then fix an edge direction D for an edge $e = (v, u)$ of F. This fixes the rotation of the polyhedron. Finally, fix the location of one endpoint v of e. This fixes the translation of the polyhedron. Let the length of e be l. Coordinates of u can be computed as $v+l \cdot D$. Now, let e' be the next edge on the boundary. Then direction vector D' can be computed using the facial angle θ between the edges e and e' by rotating D around axis N. Figure 3.1 shows the available information including the edge length l.

Once D' is computed, the coordinates of w can be computed from $u+l' \cdot D'$ where l' is the length of edge e'. Repeating this for all edges of the face, the coordinates of all vertices of this face can be reconstructed.

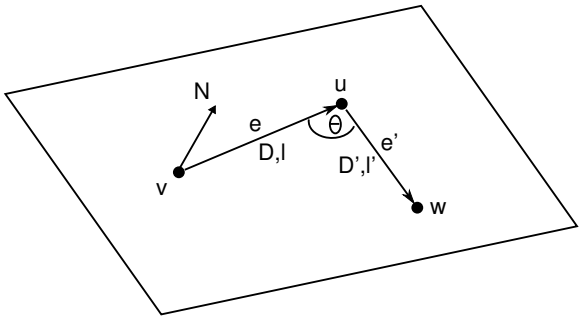

Figure 3.1: Compute the vertex coordinates and edge directions from known coordinates, edge directions and facial angle θ.

The face normal of an adjacent face can be computed similarly by applying a vector rotation to N. For this rotation, the rotation axis is the direction vector of the edge shared by the two faces and the rotation angle is the dihedral angle between these faces.

If the current face has holes in it, then the relative vertex coordinates and edge direction information provided for the hole boundary is used to propagate the information between boundaries. A similar vector rotation around the face normal is used to compute the direction vector of an edge on the hole boundary. The relative coordinate information provides the coordinates of a vertex on the hole

boundary. With these coordinates and the edge direction, the coordinates of the remaining vertices on the hole boundary can now be reconstructed.

Hence, to resolve all three challenges, a vector rotation around an arbitrary axis in three dimensional space is used. There are different methods for rotating a vector around an arbitrary axis, but the *quaternion* method is easier to compute and less error prone. I will outline this method here, but the reader may refer to Altmann's book [Alt86] on quaternions for further details.

A quaternion can be thought as a vector in $\mathbb{R}^4$ usually written as $q = (w, x, y, z)$ or $q = [w, v]$ where $v = (x, y, z)$ and corresponds to $q = w + xi + yj + zk$ where $i^2 = j^2 = k^2 = -1$ and $ij = k$. The *conjugate* of a quaternion is given by the vector $\sim q = (w, -x, -y, -z)$. A rotation of θ degrees around a unit axis e in three dimensions is represented in quaternion form as $q = [\cos(\theta/2), e \cdot \sin(\theta/2)]$. To apply this rotation to a vector t, one computes the rotated vector t' as $[0, t'] = q \cdot [0, t] \cdot \sim q$.

As in the orthogonal case every edge is processed a constant number of times. The overall complexity of the algorithm is $O(m)$, where m is the number of edges of the polyhedron. Note that the complexity of propagating relative information per each hole and face normals per each face are dominated by $O(m)$ as the number of holes and faces are always less than the number of edges.

Theorem 3.4 *The coordinates of the vertices of a polyhedron can be reconstructed from its dual graph, dihedral angles, facial angles, edge lengths and relative translation and rotation information per each connected component of the graph of the polyhedron in $O(m)$ time up to translation and rotation.*

3.2 Reconstructing Dihedral Angles

In Section 3.1, a linear time algorithm to reconstruct vertex coordinates of an orthogonal polyhedron or polyhedral surface from its dual graph, dihedral angles, facial angles and edge lengths is presented. In this section, I consider unknown dihedral angles: Is it possible to reconstruct the dihedral angles of an orthogonal polyhedron or polyhedral surface from its dual graph, facial angles and edge lengths? I show that they can be reconstructed in linear time. Moreover, the algorithm depends on the edge lengths in a weak manner. I conjecture that the dihedral angles of both orthogonal polyhedra and polyhedral surfaces can be reconstructed in linear time from the dual graph and the facial angles alone. Combined with the results of Section 3.1, this shows that the boundary representation of an orthogonal polyhedron or polyhedral surface can be reconstructed from its dual graph, facial angles and edge lengths.

Computing the set of dihedral angles of an orthogonal polyhedron from its dual graph, facial angles and edge lengths is similar in nature to the problem solved by Cauchy's Rigidity Theorem. Cauchy states that a unique set of dihedral angles of a convex polyhedron is determined by its dual graph, facial angles and edge lengths. However a polynomial time algorithm for computing the dihedral angles of a convex polyhedron has yet to be found. Sabitov [Sab96] proposed an exponential algorithm and recently Fedorchuk and Pak [FP05] showed that the unknown internal diagonal lengths between each pair of vertices are roots of a polynomial of degree at most 4^m, where m is the number of edges of the polyhedron. If we know the lengths of these internal diagonals then reconstruction is possible. Alexandrov [Ale05] extends Cauchy's theorem by showing that any convex surface can be triangulated by adding extra edges and remains rigid. The new edges introduced during triangulation must either end at the original vertices of the polyhedral surface or at new vertices on the original edges. However, he fails to provide an algorithm to compute the dihedral angles as well. Bobenko and Izmestiev [BI06] finally provided a constructive proof for Alexandrov's theorem which leads to an algorithm, which is not polynomial in the number of vertices of the polyhedral surface, but appears effective in practice.

Connelly [Con79] shows that some non-convex polyhedral surfaces can be realized with different sets of dihedral angles. Therefore, it is known that Cauchy's theorem does not hold for general non-convex polyhedra.

I present the following results for orthogonal polyhedra and polyhedral surfaces: In Section 3.2.1, a linear time algorithm for reconstructing the dihedral angles of an orthogonal polyhedron is given. This algorithm works for any orthogonal

polyhedron for which the graph of the polyhedron is connected. If the graph of the polyhedron is disconnected, then it is NP-hard to determine whether a set of dihedral angles can be computed. NP-hardness can be overcome by assuming extra information.

In Section 3.2.2, a linear time algorithm for obtaining the dual graph of an orthogonal polyhedron from the corresponding dual graph of the polyhedral surface is given. Together with Section 3.2.1, this implies reconstruction of dihedral angles even for orthogonal polyhedral surfaces, as long as the graph of the polyhedron is connected or extra information is given.

3.2.1 Reconstructing Dihedral Angles From Dual Graph of the Polyhedron

In this section, a linear time algorithm is given that reconstructs the dihedral angles of any orthogonal polyhedron with connected graph from the dual graph, facial angles and edge lengths. The crucial observation behind the algorithm is that there are a constant number of ways to realize a vertex of an orthogonal polyhedron and these realizations are distinguishable from each other if one of the incident dihedral angles is known.

All operations in this section can be done using finite precision arithmetic. Indeed, all facial angles that are used and all dihedral angles that are computed are either 90° or 270°. Therefore, two bits are sufficient to store each one. No multiplication or division operation is necessary to compute the dihedral angles.

Graph of the Polyhedron is Connected

A vertex of an orthogonal polyhedron can have three, four or six incident edges, and then it is called a *degree-3*, *degree-4* or *degree-6* vertex, respectively. Up to rotation, there are exactly four different ways of realizing a degree-3 vertex, exactly two different ways of realizing a degree-4 vertex and exactly one way of realizing a degree-6 vertex. All of these realizations are given in Figure 3.2. Each realization is accompanied by an illustration of the face corresponding to this vertex in the dual graph. The known facial angles are given inside the faces between the corresponding arcs. The dihedral angles are given as labels of arcs so that the reader can easily compare the realization of the vertex and the corresponding dual graph face. If two realizations have the same number of incident edges and the same facial angles, then they are grouped inside a dotted border.

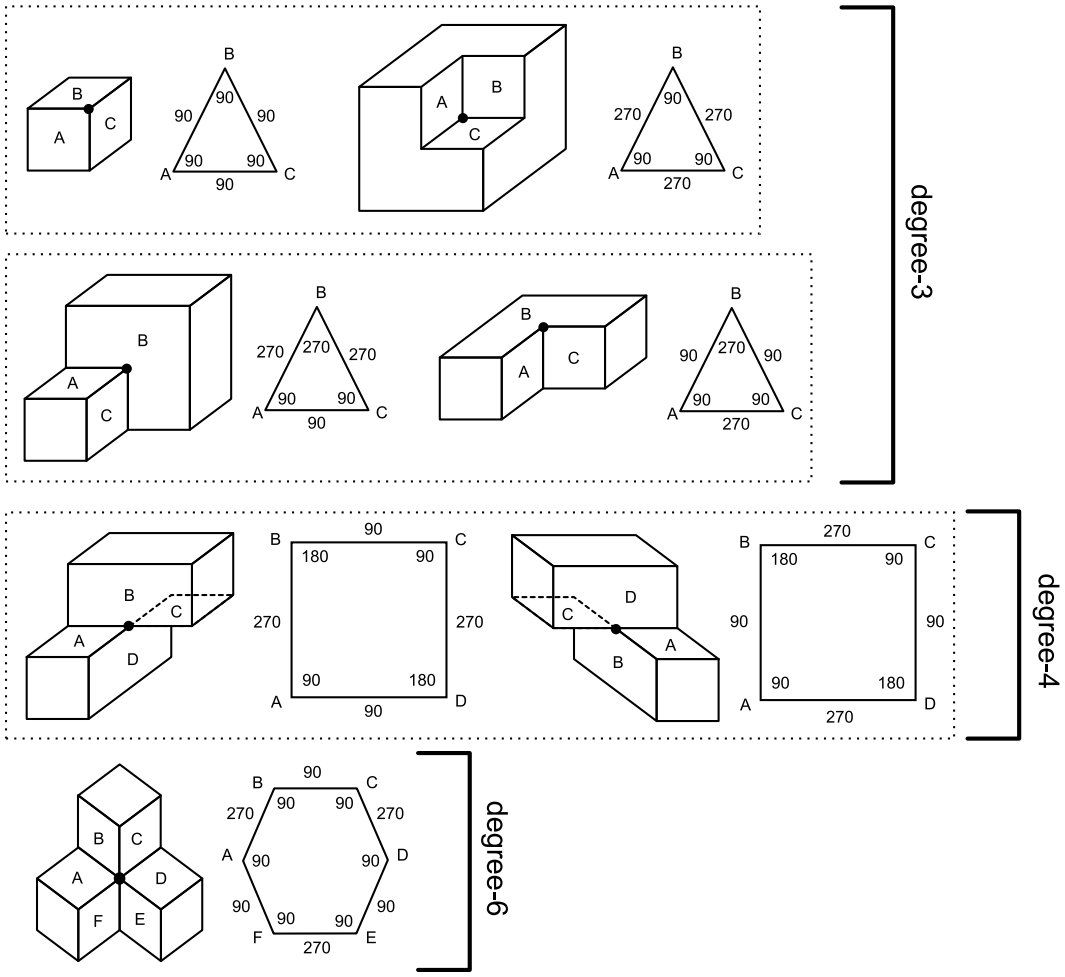

Figure 3.2: All vertex constructions of an orthogonal polyhedron.

It is possible to ask how one knows whether these are the only possible realizations of the vertices of an orthogonal polyhedron. In an orthogonal grid, each vertex is surrounded by 8 octants. By considering for each octant whether it is occupied by the polyhedron or not, one can obtain 2^8 potential realizations. Some of these cannot be a vertex of an orthogonal polyhedral surface as they do not conform to the definition of a polyhedral surface. All realizations including invalid ones are shown in Figure 3.3 (omitting symmetric realizations). The ten valid realizations that may occur on an orthogonal polyhedral surface are marked with solid borders. The top-right number gives the number of symmetric realizations, the bottom-left number gives the number of occupied octants. Two realizations, one with all quadrants empty and the second with all quadrants occupied are omitted. One can obtain the seven valid vertex realizations of an orthogonal polyhedron by merging the adjacent coplanar faces of the ten valid vertex realizations of an orthogonal

polyhedral surface.

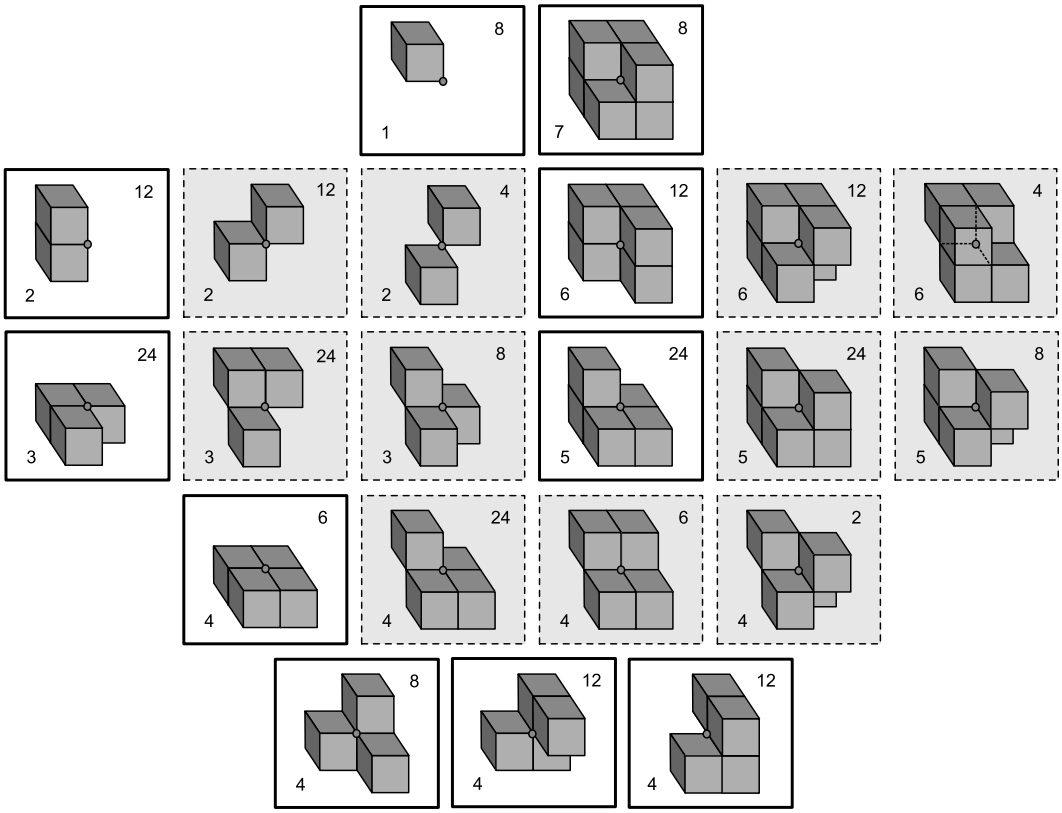

Figure 3.3: All vertex constructions using orthogonal blocks.

The following observation is crucial for understanding the next step of the algorithm: Given the dual graph, facial angles and one dihedral angle incident to a vertex, then there is only one realization that may correspond to that vertex. This realization determines the remaining incident dihedral angles immediately.

I demonstrate this observation with the following example: The top two drawings of Figure 3.2 give two realizations of degree-3 vertices. These two realizations are the only ones for which all incident facial angles are 90°. Therefore, the degree of the vertex along with the facial angle information reduces the number of possible realizations of this vertex down to two. For the first realization, all dihedral angles are 90°. For the second realization, they are all 270°. Therefore, if one of the incident dihedral angles are known, there is only one possible realization left and the remaining incident dihedral angles are immediately determined. By studying all cases, one can show that this argument holds for the other realizations as well.

Hence, starting with one known dihedral angle value, all dihedral angles of the orthogonal polyhedron can be computed as long as the graph of the polyhedron

is connected. Assume that an edge e exists with a known dihedral angle value. This dihedral angle value determines the other dihedral angle values of the incident edges of the endpoints of e. Note that it is possible that none of the seven realizations match the given facial angles and the known dihedral angle. In this case, report that the input is invalid and does not represent a valid orthogonal polyhedron. Otherwise, let e' be one of the edges which shares an endpoint v with e. Since the dihedral angle value of e' is now known, it can be used to compute the remaining dihedral angle values incident to its other endpoint. Since the graph of the polyhedron is connected, all edges are reachable by a simple path, and one can compute the dihedral angles on this path one by one using a depth-first propagation of the dihedral angle information. Therefore, all dihedral angle values are computable given one. The algorithm processes each edge only once, therefore the running time of the algorithm is $O(m)$, where m is the number of edges of the polyhedron.

At this point, all that is missing is a single dihedral angle value to initiate the algorithm. Fortunately, one can be computed with trial and error without affecting the running time of the algorithm: All edges of an orthogonal polyhedron have a dihedral angle value of either 90° or 270°. Pick an edge e arbitrarily and assign a dihedral angle value of 90°. The algorithm can now be initiated with this dihedral angle value and all dihedral angle values be computed.

However, it is not known yet whether the initial assignment of 90° to e was correct. What if the correct value should be 270°? In order to understand how the surface changes depending on these two values, consider the realizations in Figure 3.2. The first three rows have two possible realizations each. These two realizations in each row have the opposite dihedral angle values for each edge. Therefore, if one dihedral angle is swapped between 90° and 270°, then the other dihedral angles swap as well. This also holds for degree-6 vertices. So, depending on the two possible dihedral angle values of e, one can obtain two polyhedra, such that if the dihedral angle value of an edge is 90° on one, then it is 270° on the other. Indeed, it is easy to show that with same edge lengths the boundaries of these two polyhedra are congruent. The difference is in the face normals. For the correct polyhedron, all face normals point towards the outside of the polyhedron. For the incorrect polyhedron, all face normals point towards the inside of the polyhedron, call it the *inverted* polyhedron.

It is easy to detect whether the reconstructed dihedral angles belong to the correct polyhedron or the inverted one. First, compute the vertex coordinates in linear time as described in Section 3.1.2. Then, find the face with the greatest

x-coordinates (breaking ties arbitrarily) and whose face normal is parallel to the x-axis (recall that the face normals are computed as a byproduct during the computation of the vertex coordinates). This face is a "rightmost" face of the orthogonal polyhedron, and its face normal must point in the positive x direction. If so, then the dihedral angles belong to the correct polyhedron. If not, then the dihedral angles belong to the inverted polyhedron; swap every dihedral angle with its complement to obtain the correct dihedral angles. Note that, this is the only step of the algorithm where edge lengths are used. If edge lengths are unknown, then one of the two solutions is correct. I leave it as an open problem to determine efficiently which one.

Theorem 3.5 *The dihedral angles of an orthogonal polyhedron whose graph is connected can be computed from its dual graph, facial angles and edge lengths in $O(m)$ time.*

Graph of the Polyhedron is Disconnected

Theorem 3.5 holds only for orthogonal polyhedra whose graph is connected. In this section, I show that if the graph of the polyhedron is not connected, then it is NP-hard to determine whether a set of dihedral angles exist that realizes a given dual graph, facial angles and edge lengths as an orthogonal polyhedron.

NP-hardness can be shown via a reduction from the PARTITION problem which is known to be NP-hard [GJ79]. The input of the PARTITION problem is a set of integers $a_1, \ldots, a_k$, such that $\sum_{i=1}^{k} a_i = 2K$ and the expected output is a set $S \subset \{a_1, \ldots, a_k\}$ such that $\sum_{a_i \in S} a_i = K$.

The goal is to show that given an instance of the PARTITION problem, one can construct the dual graph of an orthogonal polyhedron along with the facial angles and edge lengths, so that any algorithm which reconstructs dihedral angles also computes set S. This reduction was inspired by a similar reduction by Biedl et al. [BLS05]. The problem they study is called the 2D ORTHOGONAL FOLDING problem and defined as follows: "Given a sequence of straight line segments with joints between the segments, determine if we can bend all joints at right angles such that we obtain a simple polygon."

Their reduction is as follows: Given an instance of the PARTITION problem, $A = \{a_1, \ldots, a_k\}$, create an instance of the 2D ORTHOGONAL FOLDING problem, $A' = \{1, a_1, 1, a_2, \ldots, 1, a_k, 1, L,\ v, L\}$, such that $L = K + 1$ and $v = k + 1$. A'

determines the edge lengths of an orthogonal polygon in counterclockwise order. A solution to the 2D Orthogonal Folding instance then corresponds to a solution for the Partition instance (see Figure 3.4). One can read this solution from the direction of the edges representing the a_i values of the Partition solution, such that the edges directed down belong to S and the edges directed up do not. Since the total length of edges directed down and the edges directed up must be equal, the sum of a_i in S becomes K.

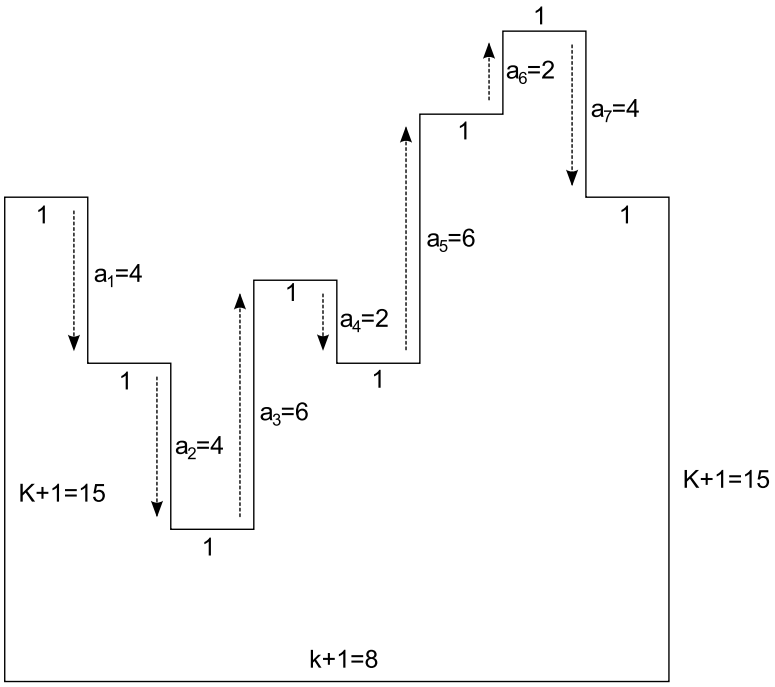

Figure 3.4: The solution of the 2D Orthogonal Folding problem for the Partition instance $\{4, 4, 6, 2, 6, 2, 4\}$.

It is possible to generalize this approach to three dimensions as illustrated in Figure 3.5. In this figure, a simpler instance of the Partition problem is considered. Each vertical edge (as enumerated from left to right in Figure 3.4) is replaced with a block of the same height. Here, a block really means that the graph of the polyhedron adds as subgraph the graph of a cube, with edge lengths as follows: For the i^{th} vertical edge, the corresponding block has a width and depth of $2(k-i)+7$ and the same height as the edge. Then, attach the bottom of the $(i+1)^{st}$ block to the top of the i^{th} block, i.e. the face corresponding to the bottom of the $(i+1)^{st}$ block is identified with the face corresponding to the top of the i^{th} block. The top face of the i^{th} block hence becomes a face with a hole, such that the distance between the face boundary and the hole boundary is 1 at all four sides. The face of the top of the last block is identified with the face of the bottom of the first block;

this effectively creates a "tunnel" that starts from the bottom face of the first block and ends at the top face of the last block. The width and depth of this tunnel is 3 and its length is $K+1$. The tunnel similarly creates faces with holes where it is attached. The bottom face of the first block then has a thickness of $2k+2$ and the top face of the last block has a thickness of 1.

Except for the first and last, each block in this construction corresponds to an a_i value. Notice that each of those blocks is free to go up or down, corresponding to the choice of bending at 90° or 270° in the 2D Orthogonal Folding Problem. Hence, a solution of the Partition instance can be read from the directions of the blocks in any realization of this graph by an orthogonal polyhedron. If a block goes down, then the corresponding a_i is in S, otherwise it is not. This decision corresponds to the two possible choices of dihedral angles at the edges of the hole boundaries.

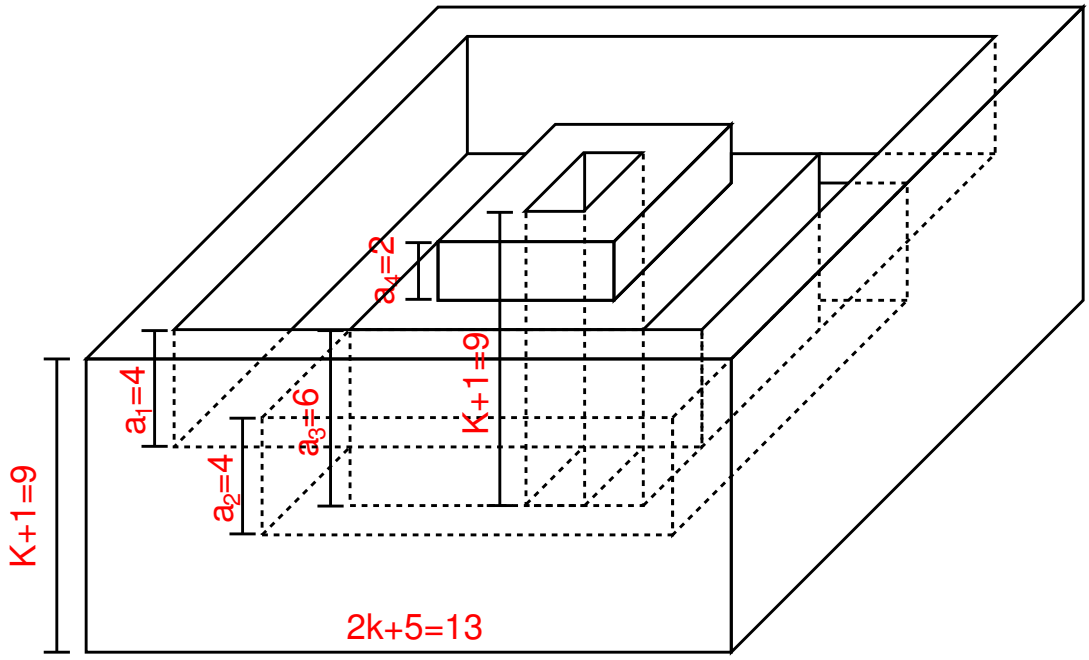

Figure 3.5: The polyhedron corresponding to the Partition instance $\{4, 4, 6, 2\}$.

The constructed polyhedron has genus 1. It is possible to modify this construction slightly to achieve the same result with a genus 0 orthogonal polyhedron. A sample cross section representing the Partition instance $\{3, 6, 9, 4, 2\}$ is given in Figure 3.6. This new construction is obtained by removing the tunnel and attaching a new block of height $K + 1/2$ to the top face of the a_k block. The top face of this new block is exactly $1/2$ units above the bottom face of the first block, therefore a "handle" on one side of the polygon is added to the construction which restricts

the top face of the new block to be less than 1 unit away from the bottom of the first block. The height of the handle is $2K + 2$ to avoid intersections with the other edges of the polygon.

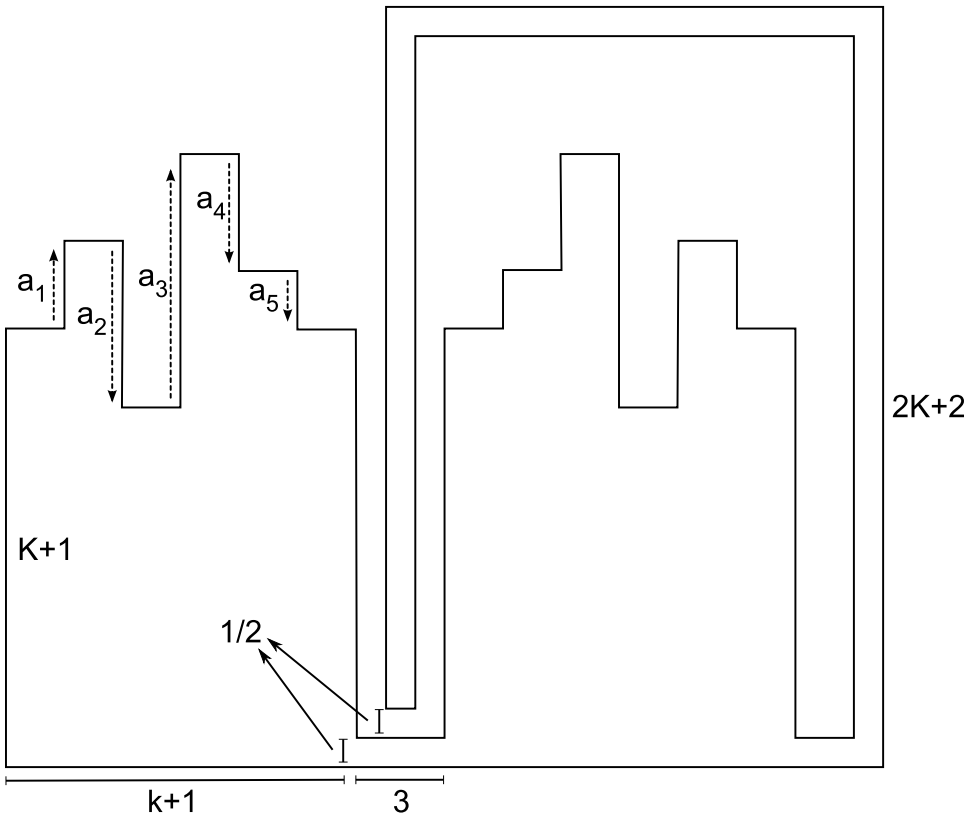

Figure 3.6: A sample 2D cross section of the genus-0 orthogonal polyhedron created using the PARTITION problem instance $\{3, 6, 9, 4, 2\}$.

Figure 3.7 shows the orthogonal polyhedron for a simpler instance of the PARTITION problem. The subgraphs of the handle and the first block are connected, and hence leaves no choice of dihedral angles. However, at each face with a hole, a decision must be made: "is the next block placed upwards or downwards?" The equivalent of this decision in the PARTITION problem is: "is this number included in S or not?" Therefore, any reconstruction of this orthogonal polyhedron provides a solution to the PARTITION problem. Clearly, the size of this construction is polynomial.

Theorem 3.6 *It is NP-hard to determine whether a set of dihedral angles exist that realizes a given dual graph, facial angles and edge lengths as an orthogonal polyhedron.*

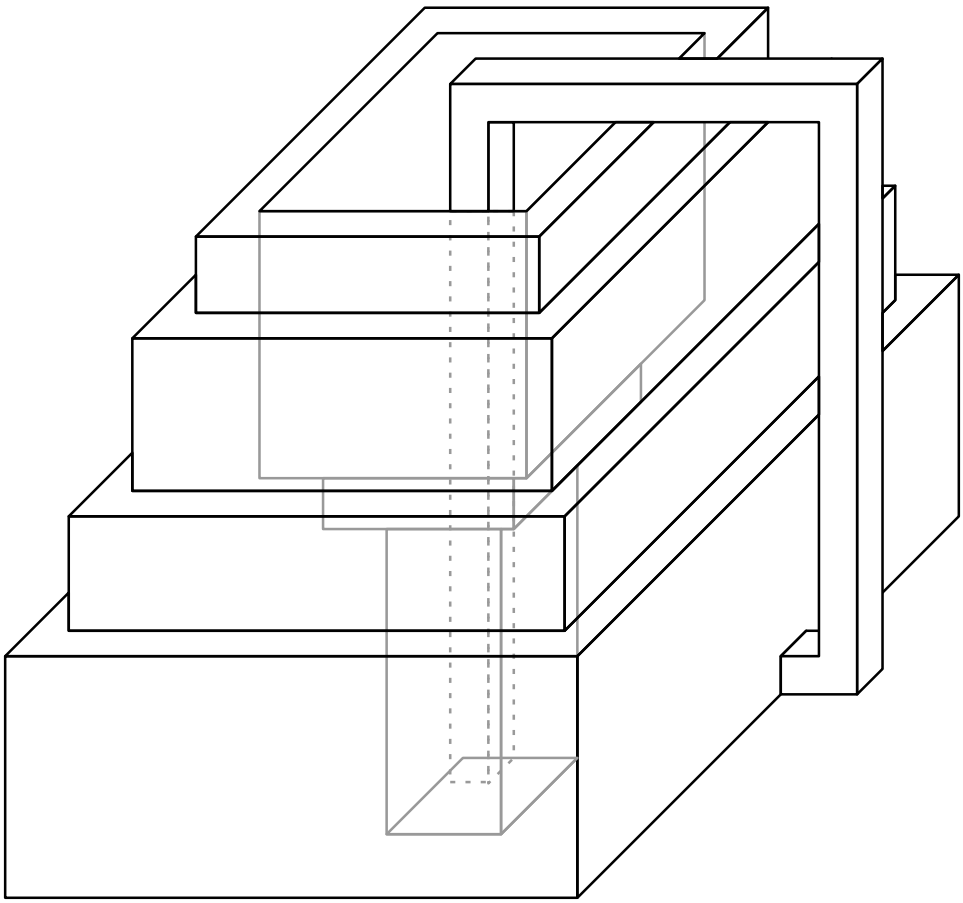

Figure 3.7: The orthogonal polyhedron corresponding to the PARTITION problem instance.

Reconstruction for Disconnected Polyhedral Graphs with Extra Information

The algorithm described in Section 3.2.1 can reconstruct the dihedral angles of an orthogonal polyhedron as long as the graph of the polyhedron is connected. Also, it is shown in the previous section that the source of the problem is faces with holes which separates the graph of the polyhedron. Each face with k holes separates the graph of polyhedron into $k + 1$ components: one component attached at each hole and one attached at the face boundary. The problem is that there is no information to decide which way to attach the components, which is what makes this problem NP-hard.

Clearly, if one knows how to put two components of the polyhedron together, then the NP-hardness proof holds no more. For example, consider an orthogonally convex polyhedron for which all components are attached in the same way to

maintain orthogonal convexity. So, knowledge of the orthogonal convexity of the polyhedron is sufficient to tell how to put the components together. However, for general orthogonal polyhedra more information is necessary.

One efficient way of solving this problem is to provide a relative dihedral angle information between the outer and hole boundaries of faces with holes. For one edge of the outer boundary and one edge of the hole boundary, this information tells whether the dihedral angles of the two edges are equal or not. If the dihedral angle of one edge is known, then the dihedral angle of the other can be reconstructed using this extra bit of information.

Note that this does not have any negative effect on the run time of the algorithm. If the graph is connected, all dihedral angle information is propagated via the edges of the polyhedron. Where the graph is not connected, the extra bit serves as an edge between the outer boundary and the hole boundaries of a face with holes.

Theorem 3.7 *Given relative dihedral angle information for each hole boundary, the dihedral angles of an orthogonal polyhedron can be reconstructed from its dual graph, facial angles and edge lengths in $O(m)$ time.*

3.2.2 Reconstructing Dihedral Angles From Dual Graph of the Polyhedral Surface

In Section 3.2.1, a linear time algorithm for computing the dihedral angles of an orthogonal polyhedron from its dual graph, facial angles and edge lengths is given, provided the graph of polyhedron is connected. In this section, I show that this can be done also for orthogonal genus-0 polyhedral surfaces in linear time. The core idea of the algorithm is to show that one can detect the edges with flat dihedral angles of an orthogonal genus-0 polyhedral surface in linear time and remove them from the dual graph of the polyhedral surface, so that the remaining graph is the dual graph of the orthogonal polyhedron. Once the dual graph of the orthogonal polyhedron is obtained, one can use the algorithm given in Section 3.2.1 to compute the missing dihedral angles of the orthogonal polyhedron.

First, some necessary observations related to dual graphs of orthogonal genus-0 polyhedral surfaces are made in the next section. Certain structures that exist in them, such as straight cycles and cycle compounds, as well as their properties are discussed in detail. Then, a linear time algorithm is given to compute the "types" of the cycle compounds of the dual graph of an orthogonal genus-0 polyhedral surface.

The types of the cycle compounds are then used to assign "types" to the nodes of the dual graph. The type of a node represents the axis that the face normal of the corresponding face on the surface is parallel to. After computing the node types, the edges of the surface which have a flat dihedral angle are immediately identified and eliminated from the dual graph of the surface in linear time, resulting in the dual graph of the polyhedron.

Straight Cycles, Cycle Compounds and Stripes

This section discusses certain structures of the dual graph D of an orthogonal genus-0 polyhedral surface S.

One can use the facial angles of each face to separate the edges of that face into two sets as follows: Let $e_0, e_1, e_2, ..., e_k$ be the cyclic ordering of the edges of face f_i. Let e_0 be in set A. If the facial angle between e_0 and e_1 is 90° or 270°, then put e_1 in set B, otherwise put it in set A. Similarly, if e_i is in set A and the facial angle between e_i and e_{i+1} is 90° or 270°, then put e_{i+1} in set B, otherwise put it in set A. Clearly, this approach separates the edges of each face into two sets, such that all edges in one set are parallel to each other, and perpendicular to all edges in the other set.

Remember that each face f_i of the polyhedral surface is represented by a node v_i in D and the edges of f_i are represented by incident arcs of v_i. Separating the edges of f_i into two sets hence splits the incident arcs of v_i into two sets (see Figure 3.8). For all m edges of the surface, this can be done in $O(m)$ time and requires $O(m)$ storage space.

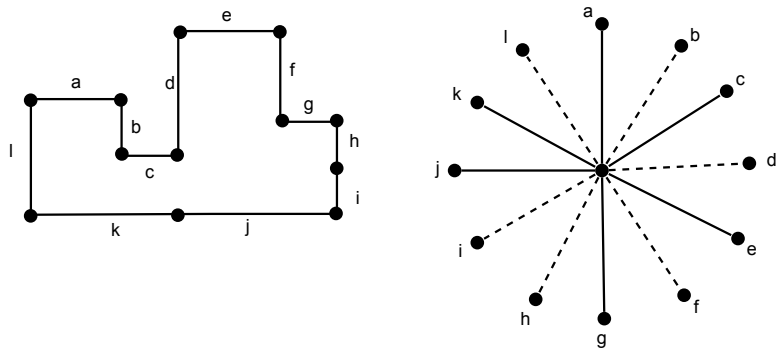

Figure 3.8: Face f_i of the surface on the left, corresponding dual graph node on the right. Solid arcs represent set A, dashed arcs represent set B.

Assume that S is quadrangulated, i.e., all faces of the surface are quadrangles and have exactly 4 vertices. Then each node in D has exactly two arcs in set A

and two arcs in set B. In this case, call the arcs in the same set *straight arcs* of each other. This is because in the planar embedding of this dual graph, walking through this node using the arcs in the same set corresponds to walking straight through the node. On the contrary, if one arrives at a node using a set A arc, and then follow a set B arc, then this corresponds to a left or right turn at this node.

If one starts at an arbitrary arc of D and follows the straight arc at one of its endpoints and repeats this for each new arc, one eventually comes back to the initial arc and compute a simple cycle. Such a cycle is called a *straight cycle*, because any two consecutive arcs on this cycle are straight arcs of each other. The simplicity of this cycle is due to the nature of the corresponding structure on the orthogonal polyhedral surface, which is discussed below.

A straight cycle corresponds to a sequence of faces on a quadrangulated orthogonal polyhedral surface, such that two consecutive faces in this sequence share an edge, and all such edges are parallel. This can be easily derived from the definition of a straight cycle. For a quadrangulated orthogonal polyhedral surface two such edges have the same length. Therefore, the faces in this sequence cover the same x, y or z-range, which is a connected part of the surface obtained by intersecting the surface with two parallel planes through vertices, without any vertices in between. Call this a *stripe* of the surface (see Figure 3.9). By definition of a closed polyhedral surface, a stripe cannot self intersect nor touch itself. Therefore, the corresponding straight cycle in the dual graph is simple (if the input is valid as assumed throughout the chapter).

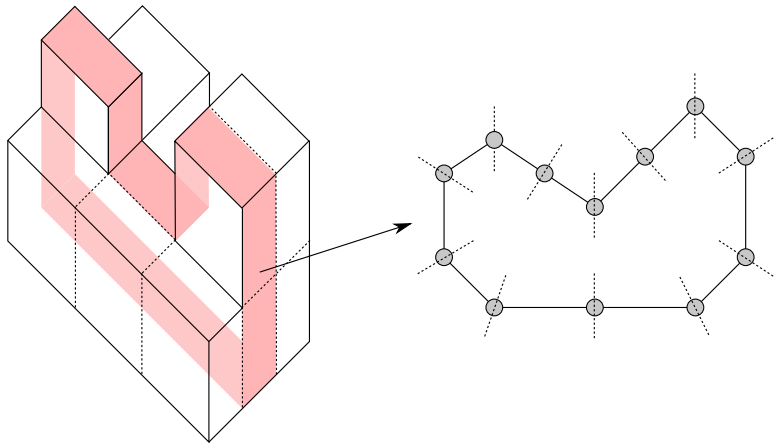

Figure 3.9: A stripe of a quadrangulated orthogonal polyhedral surface corresponds to a straight cycle in the dual graph of the surface.

If the orthogonal polyhedral surface is not quadrangulated, then one could make

it quadrangulated by subdividing the faces. However, this would have cubic worst case run time. A better approach is to generalize the concept of straight cycles as follows: Start at arc r. At one of its end nodes v, let r be in set A. Follow all set A arcs of node v to reach nodes $v_1, ..., v_k$ and mark them as visited. Repeat recursively for each v_i, following arcs of the same set as the one along which v_i was reached, until no more new nodes are marked as visited. It is possible to reach each node more than once, even though the arcs of the same set are followed at each node. The result is a set of nodes and arcs that is a connected union of cycles in the dual graph; call this a *cycle compound* of the dual graph.

There is a close relationship between the straight cycles and the cycle compounds. If an orthogonal polyhedral surface is quadrangulated by adding edges and vertices, each cycle compound transforms into a set of disjoint straight cycles. Similarly, the adjacent coplanar faces of a quadrangulated orthogonal polyhedral surface can be merged (removing unnecessary vertices during the process) and the corresponding disjoint straight cycles merge into one cycle compound in the dual graph. This relation is used later, but some other definitions are necessary at this point.

Call a face of an orthogonal polyhedral surface an *x-face* if it has a face normal parallel to the x-axis; *y-faces* and *z-faces* are similarly defined. If a face is not an x-face, then call it a *yz-face* to show that it is either a y-face or a z-face. Call an edge an *x-edge*, if it is parallel to the x-axis; *y-edges* and *z-edges* are defined similarly. These will be referred to as the *types* of the faces and the edges of the polyhedral surface.

The boundary of an x-face consists of y-edges and z-edges. An x-face shares a y-edge with another x-face or a z-face. Similarly, a y-face consists of x- and z-edges, and a z-face consists of x- and y-edges.

If a node of the dual graph of the orthogonal polyhedral surface corresponds to an x-face of the surface, then call it an *x-node*. An x-arc in the dual graph corresponds to an x-edge of the surface. Moreover, call a node of the dual graph an xy-node, if it corresponds to an xy-face of the surface. Similar definitions hold for other node and arc types. These will be referred to as the types of the nodes and the arcs of the dual graph.

The following example clarifies these concepts. Assume that u is an x-node in the dual graph D (hence corresponds to an x-face of the surface S). Node u has at least two incident y-arcs and at least two incident z-arcs (an x-node does not have incident x-arcs). Assume that one of its y-arcs is e, and the other endpoint

of e is node v. Then, v is not a y-node, however it is not known whether it is an x-node or a z-node, yet. Therefore, call it an xz-node until its true type is known. Identifying the node types in this manner is an important step in the algorithm.

The following lemma formalizes these observations (the terms in the parentheses are the equivalent terms for quadrangulated surfaces):

Lemma 3.1 *Let D be the dual graph of a (quadrangulated) orthogonal genus-0 polyhedral surface. Then, for $a \neq b \neq c \neq a$ and $a, b, c \in \{x, y, z\}$:*

1. *An a-node in D is incident to b-arcs and c-arcs.*
2. *A c-arc in D connects two ab-nodes.*
3. *All arcs of a straight cycle or cycle compound of D are of the same type.*
4. *A straight cycle or cycle compound of D consists only of two types of nodes.*
5. *Each node in D belongs to two straight cycles or cycle compounds.*

Proof. (1) holds due to the definitions of node and arc types. To show (2), note that a c-arc is never incident to a c-node due to (1). Therefore, it always connects a- or b-nodes. (3) is due to the definition of a straight cycle or cycle compound. (4) is an implication of (2) and (3). Since all arcs of a straight cycle or cycle compound are of the same type (say c-arcs), and this arc type connects only two types of nodes (a- or b-nodes), all nodes of this straight cycle or cycle compound must be of type a or b (ab-nodes). (5) holds, since the incident arcs of a node are of two different types and each type reveals a different cycle compound. ■

Call a straight cycle (cycle compound) an *ab-cycle* (*ab-compound*) if its nodes are ab-nodes. This will be referred to as the type of the straight cycle or the cycle compound. Two straight cycles (cycle compounds) *cross* at a node, if that node belongs to both of them. Recall that the dual graph of the polyhedral surface is given with a fixed planar embedding. If two straight cycles share a node, then the corresponding curves intersect in the embedding.

The following lemma provides some of the properties of straight cycles and cycle compounds which are used later in this section to show that if the types of two cycle compounds are known, then the types of all cycle compounds in the dual graph can be computed.

Lemma 3.2 *Let D be the dual graph of a (quadrangulated) orthogonal genus-0 polyhedral surface. Then,*

1. *No two cycle compounds (straight cycles) that cross can have the same type.*
2. *Two straight cycles cross an even number of times.*
3. *If three cycle compounds (straight cycles) pairwise cross, then the types of two of them determine the type of the third.*
4. *An ab-compound (ab-cycle) and an ac-compound (ac-cycle) can cross only at an a-node, where $a \neq b \neq c \neq a$ and $a, b, c \in \{x, y, z\}$.*

Proofs. The proofs for $1, 3$ and 4 are given only for cycle compounds; since a straight cycle is a cycle compound for a special polyhedral surface, the claims hold for straight cycles as well.

1 Assume two cycle compounds C_1 and C_2 cross at node u and have the same type, say xy. Then, u must be an x- or a y-node, say an x-node. Now, u is incident to y- and z-arcs by Lemma 3.1.(1). By definition of cycle compounds, all z-arcs of u belong to one of C_1 or C_2, say C_1. Then, the y-arcs must belong to C_2 by Lemma 3.1.(5). y-arcs connect xz-nodes by Lemma 3.1.(2), therefore C_2 must be an xz-compound. This is a contradiction, so C_1 and C_2 cannot be of the same type.

2 Let C_1 and C_2 be two straight cycles. It was shown that all straight cycles are simple. Since D is planar by genus 0, it can be embedded in the plane with two edges intersecting only at their end nodes. Then, C_1 and C_2 correspond to two simple closed curves in this embedding. These curves must intersect an even number of times by the Jordan Curve theorem (see e.g. [O'R98]). Since these intersections only occur at their nodes, two straight cycles share an even number of nodes.

3 For three cycle compounds C_1, C_2 and C_3, Lemma 3.2.(1) says that if C_1 crosses C_2 they will be of two different types. If C_3 crosses both of them, then it has neither type. Since there are three different types of cycle compounds, there is only one type left for C_3.

4 An ab-compound consists of c-arcs only, and an ac-compound consists of b-arcs only by Lemma 3.1.(3). When they cross at node u, they cover all incident arcs of that node by Lemma 3.1.(5). Therefore the incident arcs of u are b- and c-arcs. Only an a-node has incident arcs of type b and c by Lemma 3.1.(1).■

The next two lemmas establish properties of a quadrangulated orthogonal polyhedral surface. However, soon it will be shown that the results obtained from quadrangulated orthogonal polyhedral surfaces can be used for all orthogonal polyhedral surfaces.

The following observations and definitions are necessary for understanding the lemmas: Since D is given with a fixed planar embedding, it can be drawn in the plane for a genus-0 polyhedral surface. Within this drawing, two edges intersect only at their endpoints. Because the embedding is fixed, this drawing is unique up to the choice of the outer face. Then, every straight cycle C divides D into three disjoint subgraphs: C, C^i and C^o. It does not make a difference how one names C_i and C_o, but the reader may assume C^i refers to the "inner" subgraph and C^o refers to the "outer" subgraph after fixing some outer face. Given a straight cycle C' crossing C at nodes u and v, call (u, v) a *crosspair* of C' with respect to C^i, if one of the open segments $]u, v[$ of C' stays inside C^i at all times. Also call the open segment $]u, v[$ a *crosspath* of C' with respect to C^i. Figure 3.10 shows an example where (u, v) is a crosspair with respect to C^i, but neither (v, w) nor (u, w) is one. However, (v, w) is a crosspair with respect to C^o. A crosspath divides C^i into three disjoint subgraphs, one being the crosspath itself. Note that one or two of these subgraphs other than the crosspath may be empty. A crosspath always has at least one node and two arcs, otherwise the stripe of C is self touching and the surface is not an orthogonal polyhedral surface.

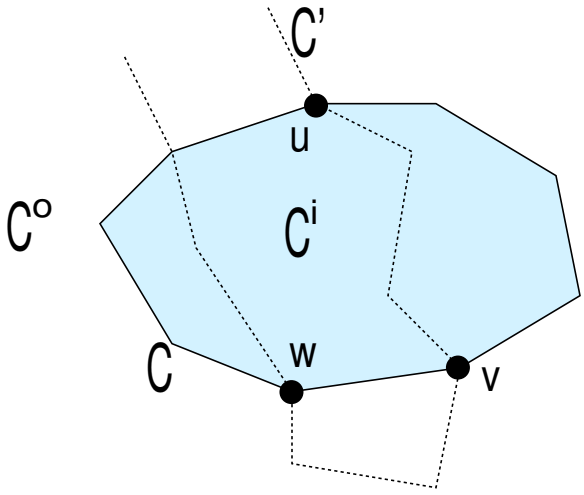

Figure 3.10: Two straight cycles crossing. (u, v) is a crosspair of C' with respect to C^i.

Lemma 3.3 *Let D be the dual graph of a quadrangulated orthogonal genus-0 poly-*

hedral surface. Let C, C_1, C_2 be straight cycles in D. For $k = 1, 2$, let (u_k, v_k) be a crosspair of C_k with respect to C^i. If (u_1, v_1) interleaves (u_2, v_2) on C, i.e. the order along C is either u_1, u_2, v_1, v_2 or u_1, v_2, v_1, u_2, then C_1 and C_2 cross each other.

Proof. Since (u_1, v_1) is a crosspair with respect to C^i, it divides C into two simple chains, call them C' and C''. The crosspath $]u_1, v_1[$ divides C^i into three disjoint subgraphs: $]u_1, v_1[$, C^{i1} and C^{i2}. Since (u_1, v_1) interleaves (u_2, v_2), u_2 is on C' and v_2 is on C'' (or vice versa). (u_2, v_2) is a crosspair, therefore the segment of C_2 between u_2 and v_2 stays inside C^i at all times. Therefore, this segment must start on C', possibly travel in C^{i1}, cross C_1, possibly travel in C^{i2} and end on C'' (or vice versa). Since D has a fixed plane embedding, C_1 must intersect C_2 at a node in the drawing. So C_1 and C_2 cross. ■

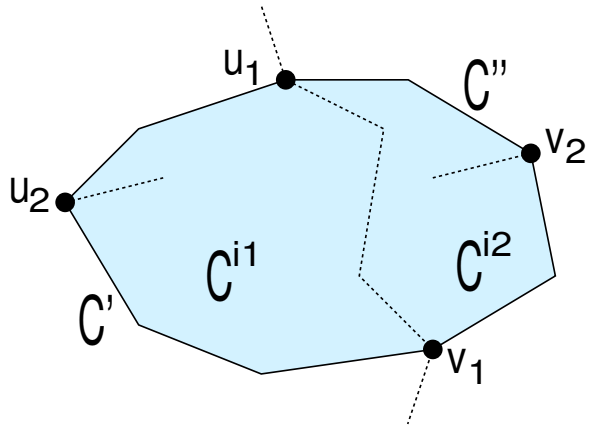

Figure 3.11: (u_1, v_1) interleaves (u_2, v_2). The crosspaths must intersect in C^i.

We now introduce a crucial lemma. We first prove it for quadrangulated orthogonal genus-0 polyhedral surfaces, but it will extend to all orthogonal genus-0 polyhedral surfaces.

Lemma 3.4 *Let C be a straight cycle in the dual graph D of a quadrangulated orthogonal genus-0 polyhedral surface. Define a graph H with a node u_j for each crosspair P_j with respect to C^i, and an arc (u_j, u_k) if P_j and P_k interleave each other on C. Then H is connected.*

Proof. Figure 3.12 shows a sample graph H to clarify the concept before proceeding into the details of the proof.

In order to prove this lemma, one can construct H and show that if H is disconnected then C cannot be a straight cycle of a polyhedral surface. Throughout this proof all crosspairs are defined with respect to C^i.

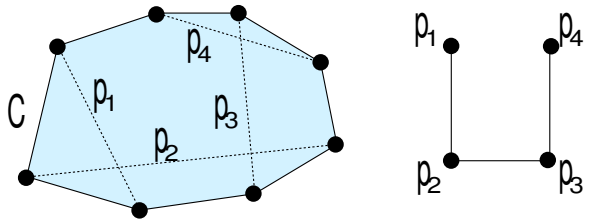

Figure 3.12: Crosspairs of a straight cycle and the corresponding graph H.

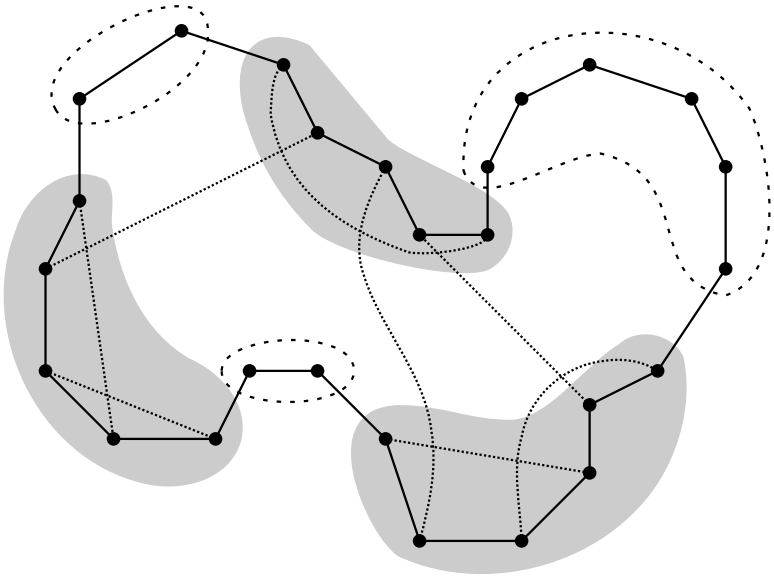

Figure 3.13: C (solid edge segments) is divided into many disconnected components by one connected component, if H is disconnected.

To construct H, start with one crosspair and create a node representing it in H. Then, for any P_i that interleaves a crosspair for which a corresponding node exists in H, create a new node in H. Create the arcs between the nodes in H as the nodes are created. Assume for contradiction that H is disconnected, which means some crosspairs are not reached when the algorithm terminates (see also Figure 3.13). Then C is separated into multiple disconnected components (shown with dashed lines) by the computed connected component (shown with gray shades). The crosspaths of the connected component are shown by dotted lines, the other crosspaths are omitted in Figure 3.13. Two nodes from two different disconnected components cannot define a crosspair on C as such a crosspair would interleave a crosspair from the connected component and thus require its nodes to be part of the connected component. Therefore, the nodes of each disconnected component define crosspairs only within that component. This immediately implies that there is an even number of nodes in each disconnected component (no node

can be in two crosspairs, because each node has degree 4 and hence only belongs to two straight cycles).

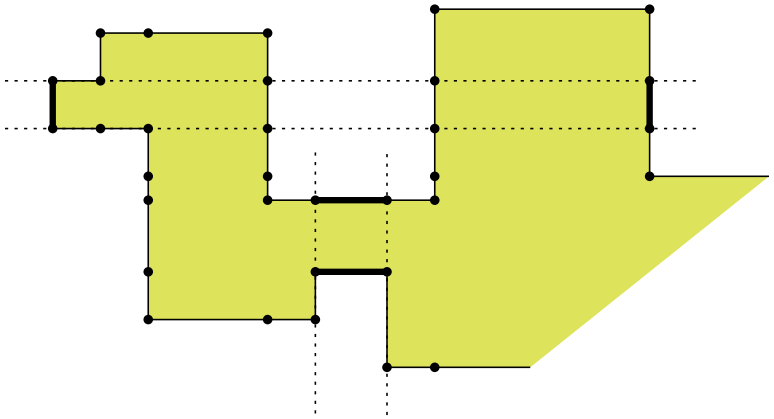

Figure 3.14: The arcs corresponding to two crosspairs on the projection of a segment of a stripe.

Remember that each straight cycle corresponds to a stripe of the surface (see also page 48). Projecting the stripe of C onto a cross-section gives a planar orthogonal polygon, see also Figure 3.14. Two nodes of a crosspair correspond to two edges of this orthogonal polygon that span the same x or y range on the projection. This holds because the edges corresponding to a crosspair are part of a straight cycle in the dual graph which corresponds to a stripe on the surface, and the edges represent the range of this stripe on the projection. Note that the edges do not necessarily "see" each other within the projection, such as the vertical edges in Figure 3.14.

Every node on C belongs to a crosspair, because two straight cycles pass through each node, one is C and the other one creates a crosspair on C. Therefore, for each edge of the projection of the stripe there must be another edge on the projection which spans the same x or y range, so the two edges correspond to a crosspair. Consider any component of H that is not reached in the algorithm. Such a component corresponds to a contiguous segment of the stripe of C, and hence a simple orthogonal chain in the projection of this stripe. Any node on this component can only be in a crosspair with another node of the same component, and the corresponding edges span the same x or y range. This implies that any horizontal or vertical line must intersect the chain an even number of times. However, by the Jordan Curve Theorem, this implies that the chain is closed while it is known to be open, since it does not include the component reached by the construction. This is a contradiction, so the construction cannot terminate in a state where there are unreachable crosspairs, and H must be connected. ■

This lemma immediately leads to the following corollary:

Corollary 3.1 *Let C be a straight cycle in the dual graph D of a quadrangulated orthogonal genus-0 polyhedral surface. Define a graph H' with a node u_i for each straight cycle C_i of D that crosses C, and an arc (u_i, u_j) if C_i and C_j cross each other. Then H' is connected.*

Proof. The nodes of graph H as defined in Lemma 3.4 correspond to the crosspairs created by the straight cycles that cross C. Therefore, each node in H' corresponds to one or more nodes in H. One can convert H into H' by contracting the nodes in H that represent the crosspairs created by the same straight cycle into one node in H'. Since H is connected, H' is connected. ■

This corollary actually holds even for cycle compounds in the dual graph of a general polyhedral surface:

Lemma 3.5 *Let C be a cycle compound in the dual graph D of an orthogonal genus-0 polyhedral surface. Define a graph H'' with a node u_i for each cycle compound C_i of D that crosses C, and an arc (u_i, u_j) if C_i and C_j cross each other. Then H'' is connected.*

Proof. The surface of an orthogonal polyhedron can be quadrangulated into polynomial number of faces. A simple but not optimal way to do this is to first quadrangulate each face separately, and then repeat it for all new faces until all faces are quadrangulated. It can be shown that the number of new faces created this way can be cubic in terms of the number of vertices of the surface.

So let D be given, and create a quadrangulated surface from D with dual graph D'. The cycle compound C then gets split into numerous straight cycles, say $C'_1, \ldots, C'_k$. For each C'_l, create the graph H'_l as in Corollary 3.1. Graph H'' then is obtained from the union of the H'_ls by contracting nodes of coplanar adjacent faces. Since each H'_l is connected, contracting them yields a connected graph, so H'' is connected.■

This end the list of properties of cycle compounds in the dual graph of an orthogonal genus-0 polyhedral surface.

Computing Face Types

In this section, a linear time algorithm for computing the types of the faces of an orthogonal genus-0 polyhedral surface from its dual graph and facial angles. The

first step is to construct the cycle compounds in the dual graph of the orthogonal genus-0 polyhedral surface D. Next, an algorithm to identify the types of the cycle compounds is given. Finally, the type of each node is computed from the cycle compound types.

The first step is constructing the cycle compounds in D. As explained in the previous section, one can construct a cycle compound by starting at an arc and following the arcs in the same set at each node in a recursive manner. Therefore, start with an unmarked arc and construct a new cycle compound until all arcs are marked (become an arc of a cycle compound). All cycle compounds of D can be constructed in $O(m)$ time, where m is the number of arcs in D, because each arc is visited only once during this step. After all cycle compounds are constructed, iterate through all nodes of D, and for the two cycle compounds that share each node, store a reference to each other. If a node has both sets of arcs in the same cycle compound, an error flag is raised to show that the input data is not valid. A reference is stored from each node to the cycle compounds it belongs to. As a result, each node has a pointer to its cycle compounds and each cycle compound has pointers to all other cycle compounds that it crosses. These pointers require $O(n)$ space to store (since the number of crossings is equal to the number of nodes) and can be computed in $O(n)$ time, where n is the number of nodes in D.

The next step is to identify the types of all cycle compounds. The following lemma shows that this is indeed possible for all cycle compounds in D.

Lemma 3.6 *Let C and C' be two cycle compounds of D with known types that cross each other. Then the types of all cycle compounds that cross C can be computed.*

Proof. Define graph H'' as in Lemma 3.5 with respect to C, so nodes of H'' represent cycle compounds that cross C. Since H'' is connected, there exists a simple path from any node to any other node in H''. This implies that there exists a path $v' = v_1, v_2, ..., v_j = w$ from node v' (representing cycle compound C') to any other node w in H''. By definition of H'', for any two consecutive nodes in this path, the corresponding cycle compounds in D cross each other and C. By Lemma 3.2.(3), the types of C and the cycle compound corresponding to v_k determine the type of the cycle compound corresponding to v_{k+1}, for $k = 1, ..., j-1$.

Since the type of the cycle compound corresponding to v' is already known, the type of the cycle compound corresponding to v_2 is immediately determined. One can then apply the same logic to v_2 and v_3, and to the rest of the nodes on the path,

until the type of the cycle compound corresponding to $v_j = w$ is computed. Using this approach, the types of all cycle compounds that cross C can be computed.■

Since D is connected, this lemma generalizes as follows:

Lemma 3.7 *Given two cycle compounds with known types that cross each other, the types of all cycle compounds in the dual graph D of an orthogonal genus-0 polyhedral surface can be computed.*

Proof. Let C_1 and C_2 be the two cycle compounds with known types. By Lemma 3.6, the types of all cycle compounds that cross C_1 and C_2 can be computed. Every new cycle compound whose type is computed can be used to compute the types of the cycle compounds that cross it. Since D is connected, all cycle compounds are connected, and by expanding the set of cycle compounds with known types by computing the type of one cycle compound at a time, one can compute the types of all cycle compounds in D. ■

This lemma shows that the types of all cycle compounds can be computed from two cycle compounds with known types. Note that it is possible to arbitrarily assign types to two cycle compounds that cross each other without affecting the values of the dihedral angles. The types of the cycle compounds affect only the rotation of the surface which is not our concern in this section. The following algorithm gives the technical details of how Lemma 3.7 can be implemented to compute all cycle types in linear time.

In the following algorithm, $\mathrm{T}(C_i)$ holds a list of possible types of a cycle compound C_i. If nothing is known about the type of C_i, then $\mathrm{T}(C_i)$ is $\{xy, xz, yz\}$. If C_i is known to cross an xy-compound, then $\mathrm{T}(C_i)$ is $\{xz, yz\}$. If C_i crosses both an xy-compound and an xz-compound, then $\mathrm{T}(C_i)$ is $\{yz\}$. When there is only one item in $\mathrm{T}(C_i)$, then this is the type of C_i, and C_i becomes "identified". B is an array of three buckets. Each bucket of B is a list of cycle compounds. A cycle compound can only be in one bucket. If a cycle compound C_i is in $\mathrm{B}[j]$ ($j \in \{1, 2, 3\}$), then it implies that $|\mathrm{T}(C_i)|$ (the cardinality of $\mathrm{T}(C_i)$) is j. Therefore, if a cycle compound is in B[1], then its type is known. The algorithm follows:

1. Let C_1 and C_2 be two cycle compounds that cross. Arbitrarily assign distinct types to C_1 and C_2, and set $\mathrm{T}(C_1)$ and $\mathrm{T}(C_2)$ accordingly.
2. For all $C_i \neq C_1, C_2$; set $\mathrm{T}(C_i) = \{xy, yz, xz\}$.
3. For $k = 1, \ldots, 3$, create the empty buckets $\mathrm{B}[k]$.

4. For each C_i, put C_i into B[|T(C_i)|].

5. While there are cycle compounds in B[2] or B[3]

   (a) If B[1] is empty, output an error message.

   (b) Else remove a cycle compound C from B[1].

      i. For each cycle compound C' that crosses C

         A. Remove the type of C from T(C').

         B. Move C' to B[|T(C')|].

Steps $1-4$ are initialization of the necessary data structures and require $O(n)$ time to complete.

Step 5 is where the types of the rest of the cycle compounds are computed. If B[2] and B[3] are not both empty, then there are cycle compounds with unknown types. If this happens when B[1] is empty (i.e., when all cycle compounds with known types have been used to compute the types of others), then this contradicts Lemma 3.7; so the input is not valid, as reported in 5.(a).

In 5.(b) a cycle compound C with known type is picked and removed from B[1]. Then, in 5.(b).i, for each cycle compound C' that crosses C (a list of all cycles that cross C was computed while constructing the cycle compounds), the type of C is removed from T(C'). Then this cycle is moved into its new bucket. Clearly, 5.(b).i is the bottleneck for the time complexity of this algorithm. Careful observation of this step reveals that every cycle compound is moved from B[3] to B[2] and then to B[1]. Each move is triggered by a crossing with another cycle compound. Since crossings occur at nodes of the dual graph, 5.(b).i cannot be executed more than the number of nodes in the dual graph. Therefore, Step 5 requires $O(n)$ time overall, where n is the number of nodes in the dual graph. This also means the overall time complexity of the algorithm is $O(n)$.

After the types of the cycle compounds are computed, the next step is to compute the types of the nodes in D. This is done according to Lemma 3.2.4. Every node belongs to two different cycle compounds whose types are known. Therefore, the type of the node is determined by the types of these cycle compounds. For example, if a cycle compound of type xy (which implies that it consists of x and y nodes) and a cycle compound of type yz (which implies that it consists of y and z nodes) cross at a node, then the type of the node is y which is the only node type included in both cycle compounds. If two cycle compounds of the same type cross at one node, then an error flag is raised to show that the input is not valid.

This step requires visiting each node once and thus requires a running time of $O(n)$ which is equivalent to $O(m)$ for a planar graph, where n is the number of nodes and m is the number of arcs in D. As a conclusion:

Theorem 3.8 *The face types of an orthogonal genus-0 polyhedral surface can be computed in $O(m)$ time from its dual graph and facial angles.*

Detecting and Removing the Edges with Flat Dihedral Angles

So far, the node types in the dual graph have been computed. The node types correspond to the face types of the polyhedral surface. Now, I use this information to detect and remove the edges of the surface with flat dihedral angles.

If two adjacent nodes of the dual graph D have the same type, the face normals of the corresponding faces of the surface must be parallel. Moreover, their direction must be the same, i.e. if one of them has a face normal in positive x direction, then the other must also have a face normal in positive x direction. Otherwise, the dihedral angle between them would have to be 360°, and this is not allowed in an orthogonal polyhedral surface. Therefore, two adjacent faces of the surface with the same face type always have a flat dihedral angle in between them. After computing the face types, one can tell which arcs in D have flat dihedral angles by looking at the types of their end nodes: If the types of the end nodes are the same, then the arc corresponds to an edge with a flat dihedral angle on the surface.

To remove all such arcs, follow a basic two step algorithm: In the first step, contract all arcs in D whose endpoints have the same type. This is equivalent to merging the corresponding coplanar faces of the surface. In the second step, for all faces in D that have only two arcs left, remove one of the arcs. These faces correspond to the vertices of the surface which are incident to only two collinear edges. After that, all edges of the surface with flat dihedral angles and all degree-2 vertices are removed. Thus, the dual graph of the polyhedron is obtained. It is now possible to proceed with the algorithm described in Section 3.2.1 to compute the dihedral angles of the remaining edges. Once these dihedral angles are computed, along with the flat dihedral angles that were computed earlier, all dihedral angles of the polyhedral surface is obtained.

It is worth mentioning that after contracting an arc, one must update the end nodes of all other arcs incident to one of the contracted nodes. In the worst case this means $O(n^2)$ updates. To decrease this runtime, it is possible to compute

all connected components of nodes which are connected with arcs of flat dihedral angles first, and then replace each component at once with a single node. At the end, the number of all updates is limited by the number of arcs in the resulting graph, which is $O(m)$.

I conclude that:

Theorem 3.9 *The dihedral angles of an orthogonal genus-0 polyhedral surface can be computed from its dual graph, facial angles and edge lengths in $O(m)$ time.*

Although the edge lengths are not used to find the edges with flat dihedral angles, they are necessary to find the remaining dihedral angles as described in Section 3.2.1. The reason for the restriction to a genus-0 surface is probably clear: For a higher genus surface, even the initial observations do not hold. For a quadrangulated orthogonal higher genus polyhedral surface, two straight cycles may cross an odd number of times. Therefore, proper crosspairs cannot be created and the graph H as defined in Lemma 3.4 is not necessarily connected. Once this lemma breaks, it may not be possible to identify the types of all cycle compounds. This means the types of the nodes cannot be computed and the edges of the surface with flat dihedral angles cannot be removed. Therefore, the dual graph of the polyhedron cannot be obtained and the remaining dihedral angles cannot be computed.

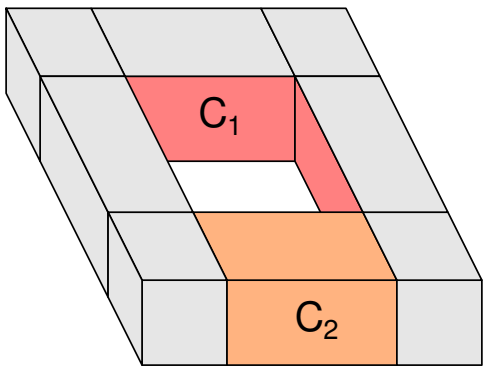

Figure 3.15: A genus-1 orthogonal polyhedral surface where the algorithm terminates with unknown cycle compound types.

In Figure 3.15, an example is given for which the algorithm fails to compute the types of all straight cycles (the surface is quadrangulated). In this example, C_1 and C_2 are two straight cycles whose stripes intersect on the surface. Therefore, C_1 and C_2 share a node. However, it is not possible to compute the type of a third straight cycle, if the types of C_1 and C_2 are known: There is no other stripe that intersects both C_1 and C_2.

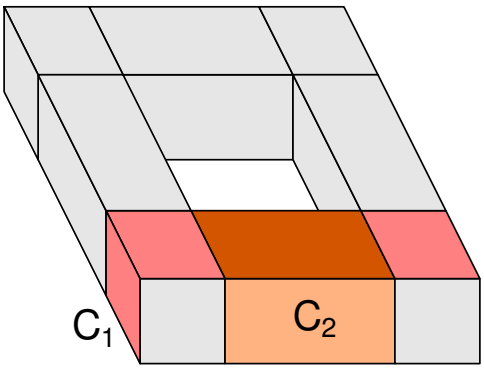

Figure 3.16: An alternative pair of straight cycles for which the algorithm computes all straight cycle types.

However, if the algorithm is initiated with a different pair of straight cycles (see Figure 3.16), it may terminate after successfully computing all straight cycle types. Whether it is always possible to find a "good" pair or not is an interesting topic for future studies.

3.3 Reconstructing Facial Angles

In this section, a linear time algorithm for reconstructing the facial angles of an orthogonal genus-0 polyhedron from its dual graph, dihedral angles and edge lengths is given. It is also shown that for orthogonal polyhedral surfaces with flat dihedral angles reconstructing the facial angles from the same information is not always possible. Finally, it is shown that the facial angles incident to degree-3 vertices can be reconstructed even for a general polyhedron. However, this result does not extend to higher degree vertices of a polyhedron, or to general polyhedral surfaces.

In Section 3.3.1, I give an algorithm for orthogonal genus-0 polyhedra. The core idea behind the algorithm is that there are a constant number of ways for realizing a vertex of an orthogonal polyhedron from its dual graph and dihedral angles. For degree-3 and degree-6 vertices of an orthogonal polyhedron, the dual graph and the dihedral angles locally determine the values of the facial angles. This is not true for degree-4 vertices. In Section 3.3.1, I show a propagation scheme that eventually reconstructs all facial angles when the polyhedron has genus 0. All computations in this section can be done with finite precision arithmetic. Indeed, they can be done using only integer numbers, since all angles are multiples of $90°$.

In Section 3.3.2, it is shown that reconstructing all facial angles of an orthogonal genus-0 polyhedral surface from its dual graph, dihedral angles and edge lengths is not always possible. However, by removing the edges of the surface with flat dihedral angles, one can still compute the facial angles of the corresponding orthogonal genus-0 polyhedron.

In Section 3.3.3, an algorithm for reconstructing the facial angles incident to degree-3 vertices of a general polyhedron from its dual graph and dihedral angles is given. It is also shown that for higher degree vertices the local information is not sufficient to determine the facial angles. The computations in this section require real numbers and infinite precision arithmetic. Therefore, the real RAM model is used.

3.3.1 Reconstructing Facial Angles From Dual Graph of the Polyhedron

In this section, I show that the facial angles of an orthogonal genus-0 polyhedron can be reconstructed from its dual graph, dihedral angles and facial angles. The reconstruction is quite similar in nature to the reconstruction of dihedral angles

of an orthogonal polyhedron from its dual graph and facial angles, as was shown in Section 3.2.1. Again, the crucial idea is that there are a constant number of ways for realizing a vertex given the order of its incident faces and dihedral angles. Degree-3 and degree-6 vertices are trivial to handle, but degree-4 vertices require a more delicate approach. Nonetheless, all facial angles can be computed in $O(n)$ time for an orthogonal genus-0 polyhedron with n vertices.

Figure 3.2 shows all possible vertex constructions of degree-3, degree-4 and degree-6 vertices of an orthogonal polyhedron. There is a unique way of realizing a degree-6 vertex, and all of its incident facial angles are 90°. Therefore, all facial angles incident to degree-6 vertices of an orthogonal polyhedron can be immediately determined (the dihedral angles are not even necessary). Now, consider the degree-3 constructions. There are four of them, and one can easily verify that the dihedral angles are different in each construction. In other words, given the dihedral angle values incident to a degree-3 vertex of an orthogonal polyhedron, one can easily choose the construction corresponding to these dihedral angles. Then, the facial angles are trivially determined from this construction. Clearly, degree-3 and degree-6 vertices can be processed in $O(n)$ time using this approach.

However, degree-4 vertices pose a difficulty. The two possible constructions have the same set of dihedral angles but a different set of facial angles at each incident face. Therefore, given the order of incident faces of this vertex along with the associated dihedral angles, one cannot choose between these two constructions. On the other hand, if one of the facial angles is given, it is trivial to choose the correct construction and determine the other facial angles. In the following, it is shown that if there are one, two or three degree-4 vertices in a face, then their facial angles can be reconstructed in linear time using the edge lengths of the face. Finally, it is shown that when the polyhedron is genus 0 and there are degree-4 vertices with unknown incident facial angles, there is always a face incident to at most three of these vertices. Therefore, all facial angles of degree-4 vertices can be reconstructed in linear time.

Reconstructing Facial Angles from turn-angles

Since the input dual graph is the dual graph of an orthogonal polyhedron, its faces can be self touching and/or have holes. For faces with holes, the component connected to the outer boundary and the components connected to the hole boundaries can be considered separately. In other words, treat the outer boundary as the

boundary of a polygonal or a self touching face and process the hole boundaries separately.

In a polygonal or a self-touching face F of an orthogonal polyhedron, there are three types of facial angles: $90°, 180°$ and $270°$. One can assign a *turn-angle* value ($-1, 0$ or $+1$) to each of these facial angles as follows: If one walks along the boundary of F in a clockwise fashion, a $90°$ facial angle corresponds to a right turn, assign it a turn-angle value of $+1$; a $180°$ facial angle means no turn, assign it a turn-angle value of 0; a $270°$ facial angle means a left turn, assign it a turn-angle value of -1.

Now, start on any edge of F and walk over the boundary in a clockwise fashion. At every vertex, add its turn-angle value to the variable *total-turn-angle*, which is initially 0. This variable counts the number of right turns minus the number of left turns. Once all vertices are visited, the total-turn-angle must be $+4$ corresponding to one full rotation. Therefore, for both polygonal faces and self touching faces of an orthogonal polyhedron the total-turn-angle is $+4$.

A second observation follows: A degree-4 vertex has incident facial angles only of $90°$ and $180°$. In other words, a degree 4 vertex contributes $+1$ or 0 to the total-turn-angle of a polygonal or self touching face. Now, assume that such a face of the orthogonal polyhedron has a single degree-4 vertex on its boundary. Calculate the total-turn-angle of this face, excluding the turn-angle value at the degree-4 vertex. If the total comes out to be $+4$, then the turn-angle value at the degree-4 vertex must be 0 (facial angle $180°$). Otherwise, the total-turn-angle must be $+3$ and the turn-angle value at the degree-4 vertex is $+1$ (facial angle $90°$). For any other values of the total-turn-angle, there is no facial angle value at the degree-4 vertex that can realize this face.

This observation leads to the following lemma:

Lemma 3.8 *Let α be the total-turn-angle of a face of an orthogonal polyhedron, excluding the unknown facial angles values. All facial angles of this face can be computed if α is $+4$ or there are at most $4 - \alpha$ unknown facial angles.*

Proof. All facial angles incident to degree-3 or degree-6 vertices of the face can be computed immediately as described earlier. Therefore, the only unknown facial angles are incident to degree-4 vertices of the face.

If α is greater than $+4$, then the face cannot be realized, because the unknown degree-4 vertices do not contribute negative values to the total-turn-angle value.

If it is +4, then all unknown facial angles must be 180° and contribute 0 to the total-turn-angle of the face. For all $\alpha < 4$, we need $4 - \alpha$ more 90° facial angles to complete the total-turn-angle value to +4. If there are exactly that many, then all are 90° facial angles. If there are fewer, then the face cannot be realized. ■

So, all facial angles incident to degree-4 vertices which fall under the condition given in Lemma 3.8 can be computed in $O(n)$ time: The total-turn-angle is computed for each face of the polyhedron, which takes $O(n)$ time. Then, each degree-4 vertex of the polyhedron with unknown incident facial angles is visited to see if one of its incident faces allows to identify its incident facial angles according to Lemma 3.8. This takes $O(n)$ time for n vertices, since there exactly four incident faces at each vertex. Therefore, this step takes $O(n)$ time overall.

Two or Three Unknown Facial Angles on a Face

In the previous step, all facial angles incident to degree-4 vertices of a face are computed, given that the face satisfies Lemma 3.8. This step treats faces where Lemma 3.8 does not hold, and there are two or three vertices with unknown incident facial angles. A linear time algorithm to compute the facial angles of such faces is given below. The algorithm uses the edge lengths to compute the unknown facial angles.

The following corollary is an immediate result of Lemma 3.8:

Corollary 3.2 *If a polygonal or self touching face of an orthogonal polyhedron has unknown facial angles after the previous step, then there are at least two such angles and at least one of them is* 90° *and at least one of them is* 180°*.*

Let F be a face as described in Corollary 3.2 with exactly two vertices with unknown incident facial angles. These vertices divide the boundary of the face into two chains, call them C_1 and C_2. All facial angles on these chains are known except the facial angles at their endpoints. Each chain can be reconstructed using the edge lengths. For each chain, assume a line segment between its endpoints. These two line segments coincide for a valid face. If one of the unknown facial angles at the degree-4 vertices is changed, then the angle between the two line segments change accordingly (normally it is 0°). Since there is a single angle between the line segments that causes them to coincide, there is a single facial angle at each vertex that realizes a valid face. It is possible to try the two possible facial angles, 90° or 180°, and see which one is the correct facial angle that causes the line segments to

coincide. After computing one of the facial angles, update the total-turn-angle of the face and compute the other facial angle according to Lemma 3.8.

Now, consider a face with three unknown facial angles incident to degree-4 vertices. Similarly, compute the three chains separated by these vertices, and call them C_1, C_2 and C_3 in clockwise order. For each chain construct the line segment between its endpoints. These three line segments create a triangle if the face is a valid face. In other words, for a correct assignment of facial angles to the degree-4 vertices, these three line segments become the edges of a triangle. Figure 3.17 shows an example face (assume that all facial angles are known to visualize the face), the three degree-4 vertices with unknown incident facial angles, and the corresponding chains and line segments.

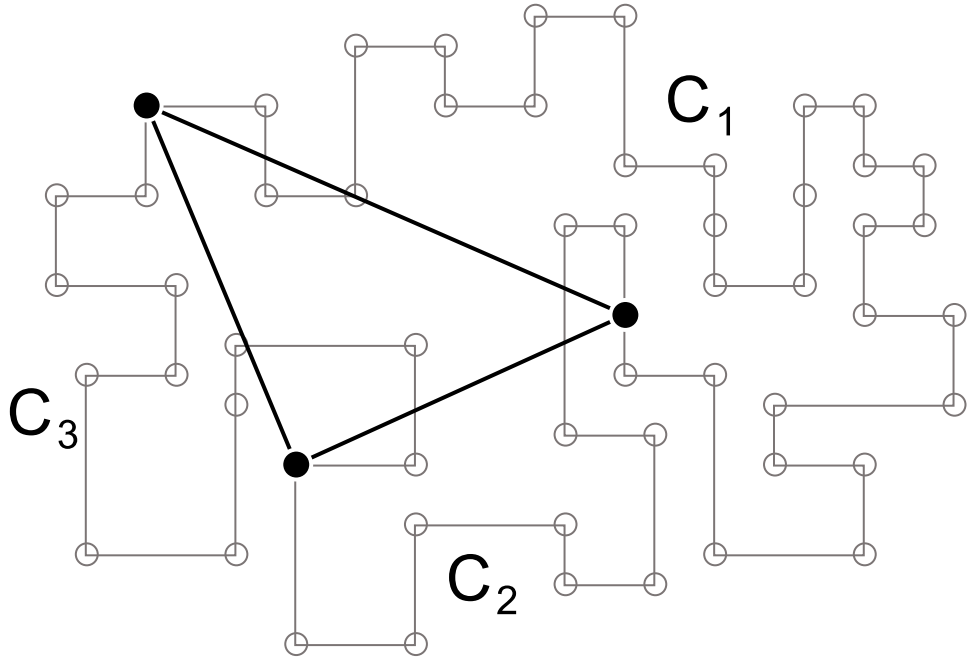

Figure 3.17: A face of an orthogonal polyhedron, its three degree-4 vertices with unknown facial angles and the corresponding chains and line segments.

No matter which facial angles are used at the degree-4 vertices, the lengths of these line segments do not change. It is a well known geometric property that the lengths of the edges of a triangle determine the interior angles of the triangle. Therefore, the lengths of the line segments determine a unique set of facial angles that completes these segments to a triangle. Try all combinations of facial angles at each degree-4 vertex to find the correct combination. There are two choices at each vertex (90° or 180°), but there must be at least one 90° and one 180°. Therefore, the number of cases to try to find the correct facial angles is $2^3 - 2 = 6$. Therefore:

Lemma 3.9 *For a face of an orthogonal polyhedron with* 2 *or* 3 *unknown facial angles, all facial angles can be computed from the order of the edges of its boundary and the lengths of its edges.*

Note that instead of trying every combination of facial angles, one can compute the facial angles directly from the triangle. However, this requires the use of infinite precision arithmetic. By trying all combinations, any operation that requires infinite precision is avoided.

In the worst case, each edge of the polyhedron is included in the computation of two chains. Therefore, the overall time complexity becomes $O(m)$. Note that once the facial angles incident to a degree-4 vertex are computed, its incident faces are checked to see whether the updated total-turn-angle of that face can be used to compute any more unknown facial angles. This does not affect the overall time complexity, since this is a simple update of total-turn-angles of the incident faces and at most $O(n)$ updates are necessary for n vertices.

As a conclusion, if any face has an unknown facial angle at this point, then the face has at least four degree-4 vertices with unknown incident facial angles. Then, all incident faces of these vertices must have at least four vertices with unknown facial angles. This holds for all faces and vertices with unknown incident facial angles. The next section shows that this is not possible for a valid genus-0 orthogonal polyhedron.

All Facial Angles are Reconstructible

Lemma 3.10 *All facial angles of degree-4 vertices of an orthogonal genus-0 polyhedron can be reconstructed from the dual graph, dihedral angles and edge lengths of the polyhedron.*

Proof. For contradiction, assume that after the previous steps of the algorithm there are still some unknown facial angles. As the facial angles incident to degree-3 and degree-6 vertices are immediately reconstructible, all unknown facial angles must be incident to degree-4 vertices of the polyhedron.

Consider the graph of the polyhedron and create a graph H as follows: There is a node in H for every vertex and face of the graph of the polyhedron that has unknown incident facial angles. There is an arc between two nodes u and v in H, if and only if u corresponds to a face and v corresponds to a vertex of that face in the

graph of the polyhedron. Clearly, H must be planar, because one can obtain it from the graph of the polyhedron by creating a new node in each face with unknown facial angles and connecting it to the vertices of that face with unknown incident facial angles. Then, it is possible to remove the other edges and obtain H without affecting planarity.

Let H have k nodes representing the vertices and l nodes representing the faces. Every vertex is a degree-4 vertex and hence adjacent to four faces. Each face must be adjacent to at least four vertices, so $k \geq l$. H must have exactly $4k$ arcs, because the vertices are all degree-4 and their representing nodes must have exactly four arcs each. However, a planar bipartite graph with $k + l$ nodes can have at most $2(k + l) - 4 < 4k$ arcs, a contradiction. ■

This lemma is the only part in this section where planarity of the graph of the polyhedron is necessary. In other words, the algorithm presented here may work for higher genus orthogonal polyhedra, but would require generalized version of Lemma 3.10 to show correctness for higher genus.

All computations used require $O(n)$ time overall for all degree-4 vertices of the polyhedron. It was already shown that facial angles incident to degree-3 and degree-6 vertices are computable in $O(n)$ time. As a conclusion:

Theorem 3.10 *All facial angles of an orthogonal genus-0 polyhedron can be reconstructed from its dual graph, dihedral angles and edge lengths in $O(n)$ time.*

This theorem involves edge lengths due to Lemma 3.9. A possible direction for future research is to improve this lemma and avoid using edge lengths. I conjecture that the facial angles are determined solely by the dual graph and the dihedral angles.

3.3.2 Reconstructing Facial Angles From Dual Graph of the Polyhedral Surface

In this section, I show that for an orthogonal polyhedral surface, it is NP-hard to determine whether a set of facial angles exist that realizes the dual graph, dihedral angles and edge lengths of the surface. The reduction is from the PARTITION problem which is known to be NP-hard [GJ79]. The input to the PARTITION problem is a set of integers $a_1, \ldots, a_k$, such that $\sum_{i=1}^{k} a_i = 2K$ and the expected output is a set $S \subset \{a_1, \ldots, a_k\}$ such that $\sum_{a_i \in S} a_i = K$.

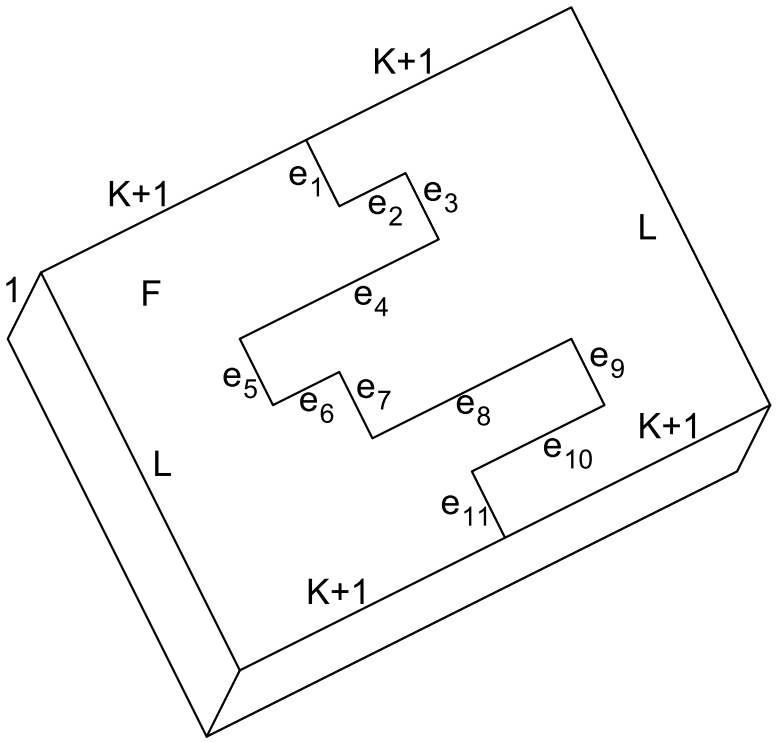

Figure 3.18: The orthogonal polyhedral surface corresponding to the PARTITION instance $\{2, 6, 2, 6, 4\}$. The figure is not to scale.

Given an instance $\{a_1, \ldots, a_k\}$ of the PARTITION problem, construct the dual graph of an orthogonal polyhedral surface as shown in Figure 3.18. Here $e_i = 2K+1$ for odd values of i, and $e_i = a_{i/2}$ for even values of i. Also, $L = (2K + 1)(k + 1)$. This ensures that e_i is "horizontal" for odd values of i and "vertical" for even values of i in any realization of this polyhedron.

It can be shown that any algorithm that can compute the facial angles from this dual graph accompanied by its dihedral angles and edge lengths, also computes a solution to the underlying PARTITION problem. The solution of the PARTITION problem can be read from the facial angles as follows: e_1 and e_{2k+1} are collinear. Consider face F and the facial angles between the edges e_{2i-1} and e_{2i} for $i = 1...k$. These facial angles must be assigned in a way that the "vertical" edges cancel each other and e_1 becomes collinear with e_{2k+1}. Put a_i in set S if and only if the facial angle in F between e_{2i-1} and e_{2i} is 90°. Then, the sum of the a_i values in S must be equal to the sum of the rest of the a_i values, so that the "vertical" edges cancel each other. S is then a solution to the PARTITION instance.

Theorem 3.11 *For an orthogonal polyhedral surface, it is NP-hard to determine whether a set of facial angles exist that realizes the dual graph, dihedral angles and*

edge lengths of the surface

It is still possible to compute the facial angles of the genus-0 orthogonal polyhedron bounded by a genus-0 orthogonal polyhedral surface. Since the dihedral angles are known, one can remove all arcs from the dual graph of the surface if they correspond to the edges with flat dihedral angles. This gives the dual graph of the polyhedron, and proceeding with the algorithm described in Section 3.3.1, it is possible to compute the facial angles of the corresponding genus-0 orthogonal polyhedron.

Moreover, consider two coplanar and adjacent faces F_1 and F_2 of the genus-0 orthogonal polyhedral surface which share only a single edge e. First remove e from the polyhedral surface and merge F_1 and F_2 into F'. Then reconstruct the facial angles as described in the previous paragraph. It is now possible to reinsert e into the surface, split F' into F_1 and F_2 and compute the incident facial angles of e using the total-turn-angle values of F_1 and F_2. It is possible to formalize this observation as follows:

Theorem 3.12 *For a genus-0 orthogonal polyhedral surface S where any two coplanar faces share at most one edge, all facial angles can be reconstructed from the dual graph, dihedral angles and edge lengths of S in $O(n)$ time.*

3.3.3 Extensions to General Polyhedra

For general polyhedra, it is possible to reconstruct the facial angles incident to a degree-3 vertex from the dual graph of the polyhedron and the dihedral angles between the incident faces. However, both storing the dihedral angles and the computations involved requires infinite precision arithmetic and can only be handled without rounding errors under the real RAM model of computation. The core idea behind the algorithm is that the incident faces of a vertex describe a *spherical triangle* on the surface of a unit sphere centered at this vertex. Without loss of generality, assume that the intersection of each face with this unit sphere creates a single spherical line segment, otherwise the sphere can be scaled down as much as necessary. Here, a spherical triangle refers to a figure formed on the surface of a sphere by three great circular arcs intersecting pairwise at three points on the sphere. The lengths of the edges of a spherical triangle constructed from the incident faces of a vertex correspond to the facial angles (in radians) of the faces. The interior angles of the spherical triangle correspond to the dihedral angles between the faces.

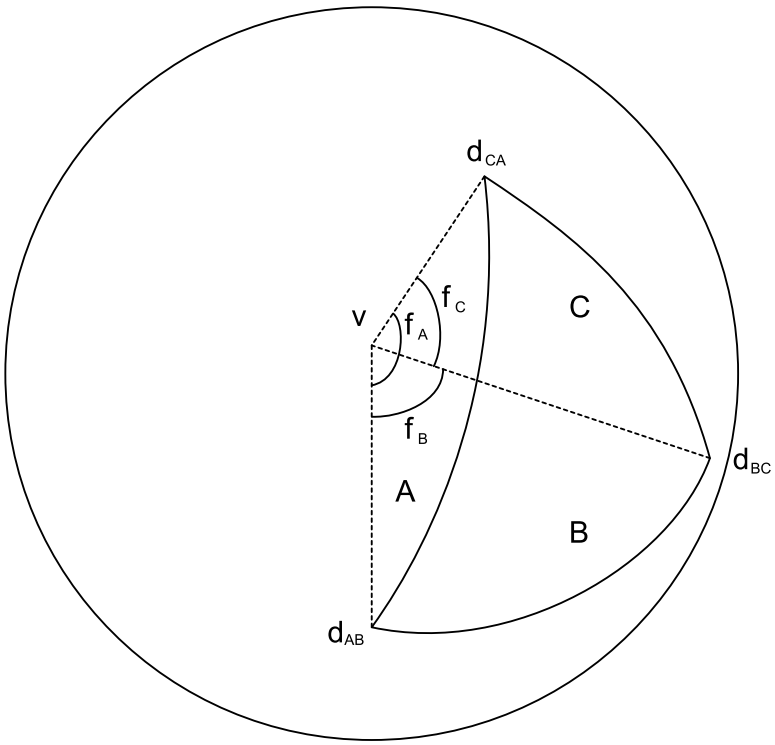

Figure 3.19: The spherical triangle cooresponding to a degree-3 vertex.

Figure 3.19 shows an example spherical triangle that corresponds to a vertex of a polyhedron and its incident faces. Here, A, B and C represent the faces incident to vertex v. f_A, f_B and f_C are the facial angles incident to v and d_{AB}, d_{BC}, and d_{CA} correspond to the dihedral angles. The following equations determine the relations between the angles of a spherical triangle [GGH+89]:

$$\cos d_{AB} = -\cos d_{BC} \cos d_{CA} + \sin d_{BC} \sin d_{CA} \cos f_C$$

$$\cos d_{BC} = -\cos d_{CA} \cos d_{AB} + \sin d_{CA} \sin d_{AB} \cos f_A$$

$$\cos d_{CA} = -\cos d_{AB} \cos d_{BC} + \sin d_{AB} \sin d_{BC} \cos f_B$$

Given the dihedral angles, one can compute the facial angles from these equations. However, one must be cautious that if the dihedral angles are not less than 180°, the corresponding facial angles must be replaced by the complements of their values. This can be easily handled by considering the constant number of cases created by the two possible values of each dihedral angle (less than 180° or greater than 180°). The cyclic order of the faces in the embedding of the dual graph is used here to determine which case corresponds to the given vertex. Also note that this approach does not work for general polyhedral surfaces, because if all incident

dihedral angles are flat, then the facial angles can be determined in an arbitrary way as long as their sum is 360°.

Theorem 3.13 *The incident facial angles of a degree-3 vertex of a polyhedron are determined by its dual graph and dihedral angles. All such facial angles of the polyhedron can be reconstructed in $O(n)$ time.*

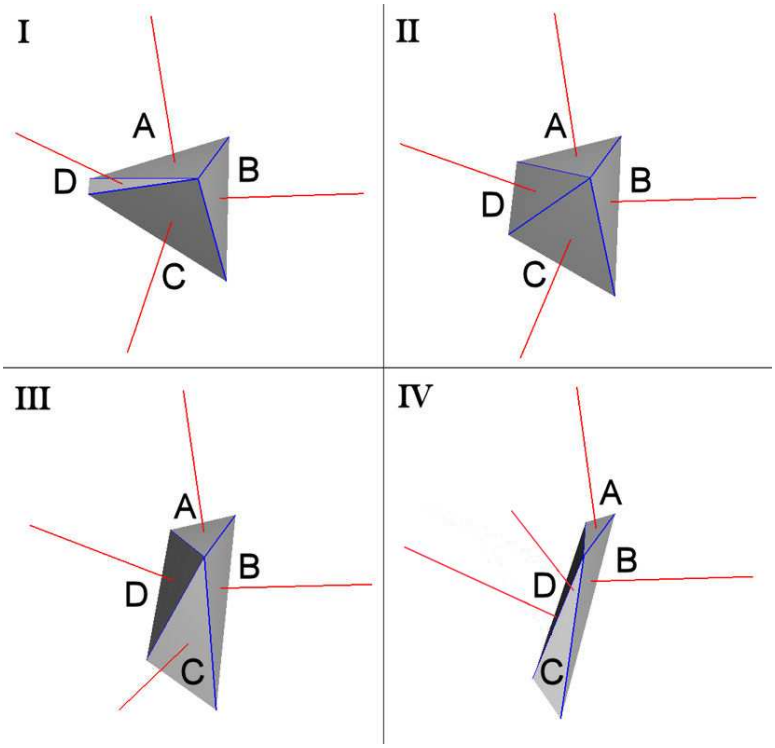

Figure 3.20: From I to IV: Known dihedral angles allow different facial angles to be realized locally. The line segments are the directions of the face normals.

This result does not extend to higher degree vertices of a polyhedron. Figure 3.20 demonstrates that for a vertex with four incident faces and known dihedral angles, the incident facial angles can be assigned in infinitely many ways. In the figure the face normals of faces A and B are fixed, and the face normal of face C is gradually changed without changing the dihedral angle between faces B and C (in other words, the angle between face normals of A and B are fixed). For each different value of the face normal of face C, a matching face normal for face D was computed according to the given dihedral angle values.

In case the reader wants to justify the drawings mathematically, the following is a list of the dihedral angles and face normal values used in each drawing:

- In all drawings the dihedral angles are as follows:
  - $d_{AB} = 90°$, $d_{BC} = 90°$, $d_{CD} \simeq 54.73561°$, $d_{DA} \simeq 54.73561°$
- Face normals in each drawing are as follows:
  - In all drawings: $n_A = \begin{bmatrix} 0.0 \\ 1.0 \\ 0.0 \end{bmatrix}$, $n_B = \begin{bmatrix} 1.0 \\ 0.0 \\ 0.0 \end{bmatrix}$
  - Drawing I: $n_C \simeq \begin{bmatrix} 0.0 \\ -0.3272 \\ 0.9449 \end{bmatrix}$, $n_D \simeq \begin{bmatrix} -0.0952 \\ 0.5773 \\ 0.8109 \end{bmatrix}$
  - Drawing II: $n_C \simeq \begin{bmatrix} 0.0 \\ -0.1872 \\ 0.9823 \end{bmatrix}$, $n_D \simeq \begin{bmatrix} -0.4239 \\ 0.5773 \\ 0.6978 \end{bmatrix}$
  - Drawing III: $n_C \simeq \begin{bmatrix} 0.0 \\ 0.3366 \\ 0.9416 \end{bmatrix}$, $n_D \simeq \begin{bmatrix} -0.7079 \\ 0.5773 \\ 0.4067 \end{bmatrix}$
  - Drawing IV: $n_C \simeq \begin{bmatrix} 0.0 \\ 0.8584 \\ 0.5129 \end{bmatrix}$, $n_D \simeq \begin{bmatrix} -0.8007 \\ 0.5773 \\ 0.1593 \end{bmatrix}$

3.4 Reconstructing Edge Lengths

In this section, I consider the problem of reconstructing the edge lengths of an orthogonal polyhedron or polyhedral surface from its dual graph, facial angles and dihedral angles. Clearly, it is not possible to compute a unique set of edge lengths, because any set that realizes a valid surface can be scaled arbitrarily to obtain a different surface with the same dual graph, facial angles and dihedral angles. A reasonable goal is to compute any working set of edge lengths. Even computing such a set turns out to be a very challenging task. In this section, a simple linear time algorithm for reconstructing the edge lengths of a quadrangulated orthogonally convex polyhedral surface is given.

Biedl, Lubiw and Spriggs [BLS07] show an LP formulation to find a feasible set of edge lengths for general convex polyhedra. Their formulation is based on the fact that the sum of the edge vectors of each face is equal to zero. For convex polyhedra this is enough to ensure that the resulting edge lengths yield a valid polyhedron.

This LP formulation is not capable of computing the edge lengths of a non-convex polyhedral surface. The problem here is that the resulting polyhedral surface may be self intersecting. Therefore, an algorithm that assigns lengths to the edges in a way that no self intersections occur is necessary.

The two dimensional variant of the problem is solvable for orthogonal polygons (see e.g. [EFK01]), and in fact any orthogonal graph drawing. However, this approach is not extensible to three dimensions. Orthogonal drawings of arbitrary graphs are not guaranteed to exist in three dimensions if extra bends in edges are not allowed. Moreover, testing whether a drawing exists is NP-hard [Pat05].

First let us consider a simple case where it is possible to assign lengths to the edges that realize the surface: a quadrangulated orthogonally convex polyhedral surface. A detailed discussion on quadrangulated orthogonal polyhedral surfaces was given in Section 3.2.2, and some important properties of the straight cycles of the dual graph were presented. Reviewing the definitions and Lemmas 3.1 and 3.2 of Section 3.2.2 may help the reader for the next discussion.

Recall that, a straight cycle is a simple cycle in the dual graph, such that the edges of the surface corresponding to the arcs of the straight cycle are parallel to the same axis. Let all arcs of a straight cycle be x-arcs, so the corresponding surface edges are parallel to the x-axis. The nodes between consecutive arcs of this straight cycle are either y- or z-nodes, and the corresponding faces of the surface have face normals parallel to y- or z-axis. The type of this cycle is yz, i.e. this is a yz-cycle.

Two straight cycles cannot share a node if they are of the same type by Lemma 3.2.

A stripe is a set of faces on the surface corresponding to a straight cycle of the dual graph. A stripe has type xy if it contains x- and y-faces, and similarly for the other types. On the quadrangulated polyhedral surface, the faces that belong to an xy-stripe have the same minimum and maximum y-coordinates and cover a part of the surface between these y-coordinates. Call this coordinate range the *range* of the stripe. Multiple stripes may have overlapping ranges and cover different parts of the surface, as in Figure 3.21. The surface can self-intersect only if two stripes have overlapping ranges. If there are no overlapping ranges, every stripe strictly separates the surface into two parts. For example, a yz-stripe separates the surface into two parts such that the minimum x-coordinate on one half is greater than the maximum x-coordinate on the other half.

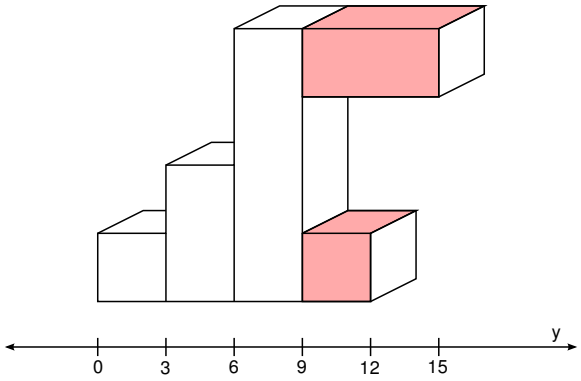

Figure 3.21: Two xz-stripes with overlapping ranges.

For a quadrangulated orthogonally convex polyhedral surface, no two stripes of the same type can have overlapping ranges. Therefore, each stripe separates the surface into two parts as described above. No matter what range is assigned to a stripe, the two parts of the polyhedral surface stay apart and cannot intersect each other. Assigning a range to a stripe is equivalent to assigning the same length to all edges on that stripe. Therefore, a trivial algorithm for assigning lengths to the edges of an orthogonally convex quadrangulated polyhedral surface is to assign the same length to all edges. This ensures that all resulting faces are valid, and the surface is not self intersecting.

Theorem 3.14 *A valid set of edge lengths of a quadrangulated orthogonally convex polyhedral surface can be obtained by assigning the same length to all edges.*

Unfortunately, a similar approach does not work for orthogonally non-convex polyhedral surfaces. Then, two stripes may have overlapping ranges, but overlap-

ping ranges do not always mean a self intersection. See Figure 3.22 for an example. In this case, the two xz-stripes have overlapping ranges. However, they are separated by a yz-stripe, such that the x-coordinates of the vertices of one stripe is always greater than the x-coordinates of the vertices of the other stripes.

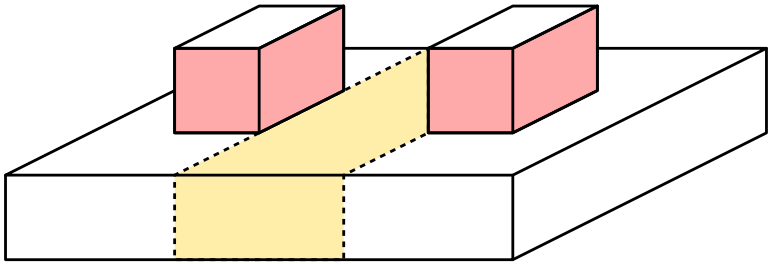

Figure 3.22: Two xz-stripes are strictly separated by a yz-stripe.

In some cases, an overlap that could normally cause a self intersection can be avoided by carefully assigning ranges to the other stripes. Figure 3.23 gives an example. One can first assign a range to stripe C, and then make B taller than C, so C and D can not overlap.

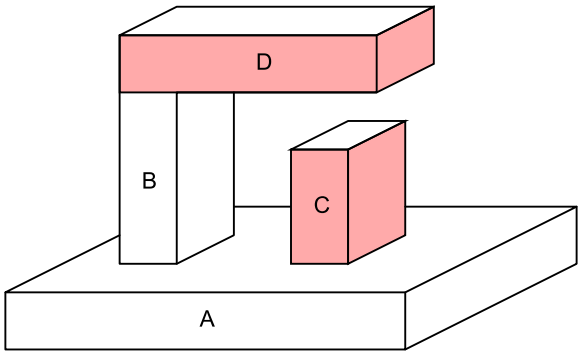

Figure 3.23: Two xz stripes that could possibly create an intersection can be strictly separated by a careful assignment of ranges to other stripes.

Therefore, assigning lengths to the edges of an orthogonally non-convex quadrangulated polyhedral surface, such that no self intersections occur, requires taking global information into account. I could not derive an efficient algorithm for this task and leave it as an interesting direction for further research.

3.5 Reconstructing Several Unknown Properties

In this section, reconstructing more than one element of an orthogonal polyhedron or polyhedral surface is considered. Most problems here remain open. Assume that

both the edge lengths and dihedral angles are unknown and must be computed from the dual graph and the facial angles. In Section 3.2, algorithms for computing the dihedral angles of an orthogonal polyhedron or orthogonal polyhedral surface from the dual graph, facial angles and edge lengths are given. It is not possible to apply these algorithms to compute the dihedral angles in the absence of edge lengths (though edge lengths are needed only to eliminate the "inverse" solution, see p. 40).

Consider the case where both the edge lengths and the facial angles are unknown. In Section 3.3 an algorithm is given to compute the facial angles of an orthogonal genus-0 polyhedron which used both the dihedral angles and the edge lengths. Again, this algorithm cannot be used for computing the facial angles in the absence of edge lengths, since they were critically needed to determine facial angles in faces with multiple degree-4 vertices.

Now consider the case where both the dihedral angles and the facial angles are unknown. With only the dual graph and the edge lengths known, computing a unique set of dihedral angles and facial angles is not possible (see Figure 3.24). This holds even for orthogonally convex polyhedra. Biedl et al. [BLS05] proved that determining whether a given polygonal chain with fixed edge lengths can be realized as an orthogonal polygon is NP-hard. This proof can easily be modified to show that determining whether a given dual graph and the edge lengths can be realized as an orthogonal polyhedron is NP-hard. Simply, replace each edge of the polygon with a rectangle as shown in Figure 3.25.

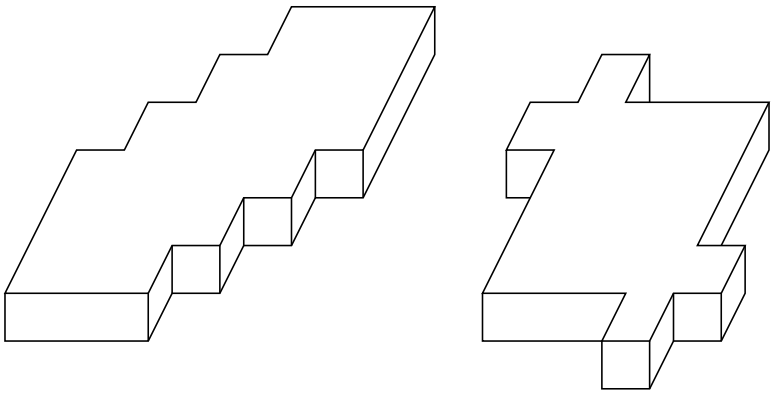

Figure 3.24: Two orthogonally convex polyhedra with the same dual graph and edge lengths.

If dihedral angles, facial angles and edge lengths are all unknown, computing a unique set of each of these is not possible due to the reasons discussed above. It may

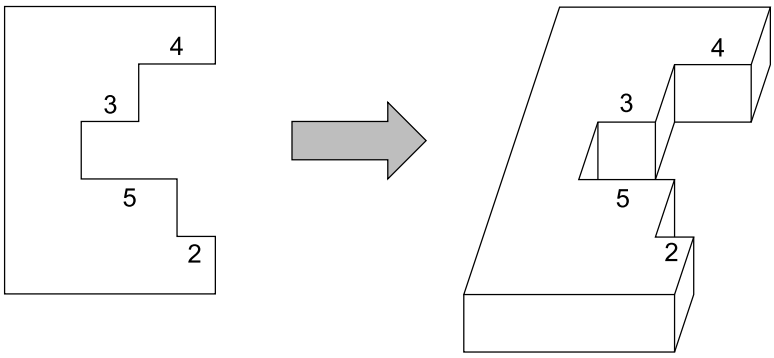

Figure 3.25: The orthogonal polyhedron constructed from the orthogonal polygon used by Biedl et al. [BLS05].

be possible to compute a valid set for each that realizes an orthogonal polyhedron. I leave this as a direction for further research.

3.6 Future Research Topics

An interesting direction for future research is reconstructing the dihedral angles of an orthogonal polyhedral surface even if the surface is not genus 0. As discussed on p. 61, the algorithms of Section 3.2 will not work here without careful modifications.

I conjecture that the algorithm presented in Section 3.3.1 can compute the facial angles of higher genus orthogonal polyhedra. However, I fail to provide a solid proof of this claim. Therefore, either proving this conjecture or finding a counter example is an important future research topic.

The same algorithm uses edge lengths to compute the facial angles. It would be more useful to have an algorithm that computes the facial angles without using the edge lengths.

Finally, most of the problems in Section 3.4 and 3.5 remain open.

Chapter 4

Reconstruction from Vertex Coordinates

In Chapter 3, it was shown that an orthogonal polyhedral surface or an orthogonal polyhedron can be described via its dual graph and some subsets of its dihedral angles, facial angles and edge lengths.

In this chapter, I consider reconstructing the boundary representation of an orthogonal polyhedron from its vertex coordinates. In order to reconstruct a boundary representation, one must compute the edges and faces of the orthogonal polyhedron. This computation is closely related to the pattern and object recognition problem studied by many researchers [O'C74, Ahu80, ABCO$^+$01]. O'Rourke studied a similar reconstruction problem in two dimensions for orthogonal polygons [O'R88].

I start with the related background in literature, and I outline O'Rourke's algorithm for computing the edges of an orthogonal polygon from the coordinates of its vertices. These edges are unique, i.e. there is only one polygon that can be reconstructed from the input vertex coordinates [O'R88]. Then I show that one can compute the boundary representation of an orthogonally convex polyhedron from its vertex coordinates in $O(n \log n)$ time. Next, I consider rotated orthogonal polygons and polyhedra. Recall that these are polygons (and polyhedra) whose all facial (and dihedral angles) are 90°, but their edges are not axis-parallel. I show that in two dimensions, the edges of a rotated orthogonally convex polygon can be reconstructed from its vertex coordinates in polynomial time. I show that, the reconstructed edges are unique, i.e. they are the only set of edges that realizes the given vertex coordinates as a rotated orthogonally convex polygon. Recently,

Mumford and Löffler reported that uniqueness holds for all rotated orthogonal polygons. In three dimensions, I show that the edges and faces of a rotated orthogonally convex polyhedron can be reconstructed from its vertex coordinates in polynomial time. Finally, I consider additional points on the boundary and inside of the polygons and polyhedra.

Computing the edges of general polygons from vertices in two dimensions was first studied by Steinhaus [Ste64]. Unless the vertices are all collinear, there always exists a polygon, and it can be found in many ways; Hurtado et al. [HTT03] give a good overview on this topic. Computing the edges and faces of a general polyhedron from a set of vertices in three dimensions was first studied by Grünbaum [Grü94]; extensions and generalizations can be found in [HTT03]. For convex polyhedra, this can be done by computing the three dimensional convex hull of the point set [Day90, Ede87, PH77, PS85].

There are studies also on non-convex polyhedra. A summary and some methods are presented in [HTT03]. Unfortunately, these methods fail to compute a unique set of edges and faces, since different non-convex polyhedra may have the same vertices.

For orthogonal polygons, Bournez et al. [BMP99] claim that a unique set of edges cannot be determined from the vertex coordinates alone (see Figure 4.1). However, in their definition of an orthogonal polygon they allow the polygon boundary to touch itself repeatedly, which is something not allowed in this thesis. If the boundary of the orthogonal polygon is not allowed to touch itself, then there is a simple $O(n \log n)$ algorithm to compute the boundary edges of the polygon ([O'R88], see also Section 4.1.1).

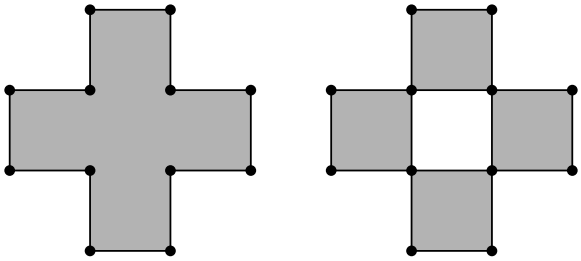

Figure 4.1: Two orthogonal "polygons" (one touching itself) that have the same vertex coordinates. Figure from Bournez et al. [BMP99].

For orthogonal polyhedra, the vertex coordinates fail to determine a unique set of edges and faces even for the definition of polyhedra used in this thesis, see Figure 4.2. But the polyhedron in this example is not orthogonally convex. I show

that, for orthogonally convex polyhedra, the edges and faces can be computed from the vertex coordinates in $O(n \log n)$ time.

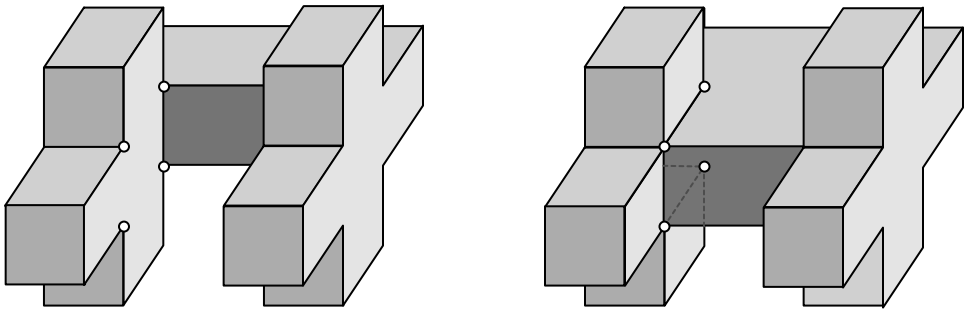

Figure 4.2: Two orthogonal polyhedra that have the same vertex set.

No one appears to have studied computing the boundary of an orthogonally convex polyhedron from its vertex coordinates. This is closely related to computing the orthogonally convex hull of a point set in three dimensions. A similar problem is computing the *maxima* of a point set in three dimensions. Although, one can obtain the orthogonally convex hull from the maxima of a point set, no studies exist that outline this computation; more details on this are given in Section 4.1.2.

4.1 Orthogonal Polygons and Polyhedra

In this section, I consider reconstructing the edges and/or faces of an orthogonal polygon or orthogonally convex polyhedron from its vertex coordinates. Polynomial time algorithms are presented to compute the edges of an orthogonal polygon or the edges and faces of an orthogonally convex polyhedron. The case of orthogonal polygons was studied by O'Rourke [O'R88], and I review his algorithm in Section 4.1.1. An $O(n \log n)$ time algorithm for reconstructing the edges and faces of an orthogonally convex polyhedron from its vertex coordinates is given in Section 4.1.2.

4.1.1 Orthogonal Polygons

O'Rourke shows that the edges of an orthogonal polygon can be reconstructed from its vertex coordinates in $O(n \log n)$ time [O'R88]. Thus, he solves the following problem:Given a set of points S in the plane, construct an orthogonal polygon with vertex set S if one exists. For his algorithm, it is clear that there can be only one such polygon.

Given a set S of n points in the plane, let $s_1, \ldots, s_t$ be a maximal set of points in S that have the same y-coordinate, sorted by increasing x-coordinate. Each vertex of an orthogonal polygon is incident to exactly one horizontal edge and one vertical edge of the boundary. Therefore, if S determines a unique orthogonal polygon, then (s_i, s_{i+1}) must be an edge for odd values of i and is not an edge for the even values (see Figure 4.3).

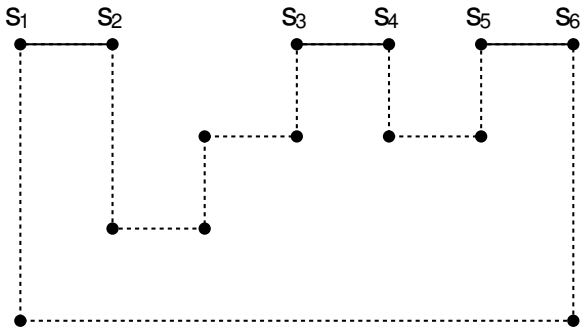

Figure 4.3: Computing the edges of an orthogonal polygon from its vertex coordinates.

Repeat this for every set of points with the same x-coordinate, and every set of points with the same y-coordinate to obtain all polygon edges. The overall complexity of the algorithm is $O(n \log n)$ dominated by the sorting of the points.

Theorem 4.1 *[O'R88] The edges of an orthogonal polygon can be reconstructed from its vertex coordinates in $O(n \log n)$ time.*

4.1.2 Orthogonally Convex Polyhedra

Next, I consider orthogonally convex polyhedra. Do the vertex coordinates determine the edges and faces of an orthogonally convex polyhedron? Thus I study the following problem: Given a set S of points in $\mathbb{E}^3$, construct an orthogonally convex polyhedron with vertex set S if one exists.

I present an algorithm that reconstructs the edges and faces of such a polyhedron from its vertex coordinates in $O(n \log n)$ time and uses $O(n)$ space. All computations can be done with finite precision.

First, let us consider two other possible approaches of reconstructing the edges and faces of an orthogonally convex polyhedron from its vertex coordinates. The first approach is an extension of O'Rourke's algorithm from two to three dimensions.

It is known that reconstructing the edges is enough to determine the faces [O'R88]. So, reconstruct the edges as follows: For each orthogonal plane, compute the set of points in it, and use O'Rourke's algorithm to compute the edges within that plane. Unfortunately, this does not reconstruct the edges correctly. For the original algorithm in two dimensions, the crucial ingredient is that every vertex is incident to exactly one horizontal and one vertical edge. This is not true for the boundary of an orthogonally convex polyhedron: A vertex may have one or two incident edges parallel to a coordinate axis. Figure 4.4 gives an example.

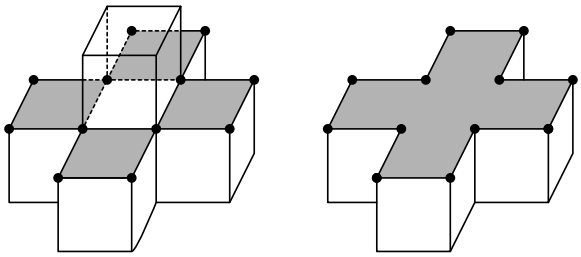

Figure 4.4: Two orthogonally convex polyhedra with the same set of vertices but different edges and faces within one orthogonal plane.

The second approach is to compute the so-called *orthogonally convex hull* of the vertices. The orthogonally convex hull of a point set may be defined in different ways, see the detailed study by Ottman et al. [OSSW84]. The following definition is used in the rest of this thesis which covers point sets in both two and three dimensions:

Definition 4.1 (Orthogonally Convex Hull) *The orthogonally convex hull of a point set S is the minimum orthogonally convex set that contains S.*

Note that this definition may lead to disconnected hulls which are still orthogonally convex. Rawlins and Wood [RW88] show that the orthogonally convex hull of the vertices of an orthogonally convex polygon is the polyhgon itself, and therefore always connected. This result can be easily extended to three dimensions for orthogonally convex polyhedra. However, there appears to be little work done on efficiently computing the orthogonally convex hull of a point set in three dimensions. Computing the *maxima* of a point set is the closest result in the literature. The maxima M of a point set S is a maximal subset of S such that no point in S *dominates* any point in M. Here, point p dominates point q if all coordinate values of p are greater than the coordinate values of q. It is possible to compute eight different maximas of a point set in three dimensions by reversing the dominance

definition separately for each axis. Then, it can be shown that the points of these eight sets are the vertices of the orthogonally convex hull. However, an algorithm to reconstruct the edges and faces of the orthogonally convex hull from the maxima points does not exist in the literature. Instead of computing the maximas and then the orthogonally convex hull of the input point set to compute the boundary of the polyhedron, I present an algorithm that computes the boundary of the polyhedron by computing the orthogonally convex hulls of layers of vertices in two dimensions. The algorithm requires $O(n \log n)$ time and $O(n)$ space, where n is the number of vertices of the polyhedron.

The algorithm can be outlined in two steps: In the first step, sweep through all six orthogonal directions and construct the *shadows* (to be defined precisely later) of parts of the polyhedron. In the second step, reconstruct the edges and faces of the polyhedron from these shadows. The edges and faces together with the vertex coordinates determine the boundary representation of the polyhedron.

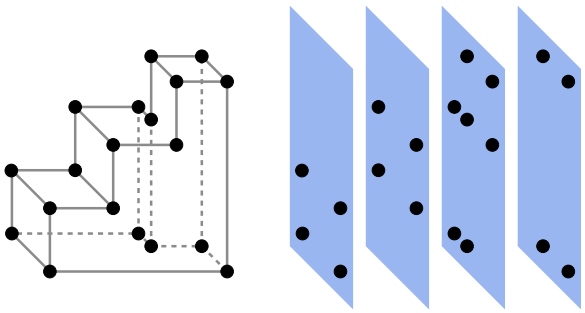

Figure 4.5: A simple orthogonal polyhedron and its x-layers.

To explain the first step in more detail, the following definitions are necessary. Let P be an orthogonally convex polyhedron. Let $x_1 < x_2 < \cdots < x_t$ be the values for which some vertex of P has x-coordinate x_i. Each of these x_i determines a vertex *layer* of the polyhedron (see Figure 4.5). For $i = 1, \ldots, t$, let P_i^- be the left half ($x \leq x_i + \varepsilon$) of the polyhedron obtained when slicing P with a plane $\{x = x_i + \varepsilon\}$ (for $0 < \varepsilon < x_{i+1} - x_i$), and let π_i be the projection of P_i^- onto an x-plane; call π_i the i^{th} *shadow in* x^+*-direction* (see Figure 4.6). Define π_0 to be the empty set. I also refer to the projection of each x-face onto an x-plane as its shadow.

The following lemmas are crucial:

Lemma 4.1 *Let P be an orthogonally convex polyhedron. Then each $\pi_i, i = 1, ..., t$ is an orthogonally convex polygon. Moreover, π_i is the orthogonally convex hull of the vertices of P_i^- projected onto an x-plane.*

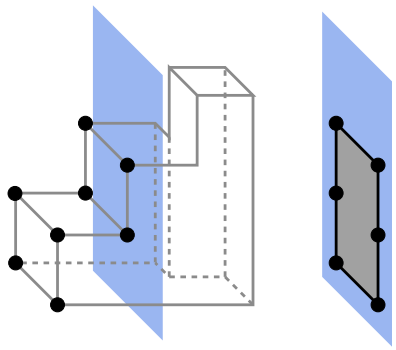

Figure 4.6: π_2: Shadow of the second x-layer in x^+-direction.

Proof. If the shadow of any layer is not an orthogonally convex polygon, then there exists an orthogonal line on the plane of projection whose intersection with the shadow creates at least two orthogonal line segments. Then, there exists an orthogonal plane perpendicular to the projection plane and passing through this line whose cross-section with P is not an orthogonally convex polygon. However, this implies that P is not orthogonally convex. Therefore, all shadows must be orthogonally convex polygons.

Let the orthogonally convex hull of a point set S or a polygon p be denoted by $OCH(S)$ or $OCH(p)$, respectively. Let S_i be the set of points obtained by an orthogonal projection of the vertices of P_i^- onto an x-plane. I prove the second part of the lemma by showing that both $OCH(S_i) \supseteq \pi_i$ and $\pi_i \supseteq OCH(S_i)$.

$OCH(S_i) \supseteq \pi_i$: Let u be an arbitrary point of π_i. Draw a line perpendicular to the projection plane at u, this line must intersect P_i^-. Indeed, this line must intersect one x-face F on P_i^-. Clearly, each point on F is inside the orthogonally convex hull of the vertices of F. Similarly, the projection of each point on F is inside the orthogonally convex hull of the projections of its vertices. Therefore, u belongs to the orthogonally convex hull of the projected vertices of F, which is part of $OCH(S_i)$. Therefore, $OCH(S_i) \supseteq \pi_i$.

$\pi_i \supseteq OCH(S_i)$: The orthogonally convex hull of a point set is the minimum orthogonally convex set that includes all the points. Therefore, $OCH(S_i)$ is the minimum orthogonally convex set which includes all points in S_i. Also by definition of the shadow and the first part of the lemma, π_i is an orthogonally convex polygon which includes all points in S_i. Since $OCH(S_i)$ is the minimum of all such sets, $\pi_i \supseteq OCH(S_i)$. ■

For the next lemma, let $int(p)$ be the interior of a polygon p. Also, let an x^-*-face* be an x-face whose face normal is in negative x-direction. Similarly, define

x^+-faces.

Lemma 4.2 *Let P be an orthogonally convex polyhedron. For $i = 1, \ldots, t$, let F_i be the set of all x^--faces whose x-coordinate is x_i. Let U be the union of the shadows of the faces in F_i. Then $\pi_i - \pi_{i-1} \subseteq U \subseteq \pi_i - int(\pi_{i-1})$.*

Proof. $U \supseteq \pi_i - \pi_{i-1}$: During a sweep in positive x-direction, the only type of faces that cast a two dimensional shadow are the x^-- and x^+-faces. So π_i is the union of the projection of all x^--faces and x^+-faces with x-coordinate at most x_i. Any such face with x-coordinate less than x_i is already included in π_{i-1}. Also, any x^+-face with x-coordinate x_i is covered by the shadow of the cross-section immediately before it, and hence also included in π_{i-1}. Therefore, $\pi_i \subseteq U \cup \pi_{i-1}$ as desired.

$\pi_i - int(\pi_{i-1}) \supseteq U$: Let f' in U be the shadow of an x^--face f in F_i. Clearly $f' \subseteq \pi_i$. Assume for contradiction that f' intersects $int(\pi_{i-1})$. Then an x^+-face g exists, such that its x-coordinate is less than x_i and the interior of its shadow overlaps f'. Let l be the line parallel to the x-axis and passing through a point in the overlapped region of f' and the shadow of $int(g)$. l intersects both f and $int(g)$. Since g is an x^+ face and f has a larger x-coordinate than g, there exists a segment of l (adjacent to its intersection point with $int(g)$) which is outside the polyhedron. This contradicts the orthogonal convexity of the polyhedron, so g cannot exist. Therefore, $f' \subseteq \pi_i - int(\pi_{i-1})$, and applying the union over all faces shows $U \subseteq \pi_i - int(\pi_{i-1})$. ■

This lemma implies the following corollary:

Corollary 4.1 *Let P be an orthogonally convex polyhedron. For $i = 1, \ldots, t$, let F_i be the set of all x^--faces whose x-coordinate is x_i. Let U be the union of the shadows of the faces in F_i. Then U is the closure*[1] *of $\pi_i - \pi_{i-1}$.*

As an example to Corollary 4.1, Figure 4.7 shows an orthogonally convex polyhedron and its positive x-layer shadows. The shadow at each layer corresponds to an orthogonal projection of the left half of the polyhedron at that layer. The darker shadings at each shadow corresponds to the shadow of the previous layer. The lightly shaded areas of each shadow correspond to the actual x^--faces of the polyhedron. As shown earlier, the difference of the shadow of the current layer and

[1]The closure of an open planar region is the union of the region and its boundary.

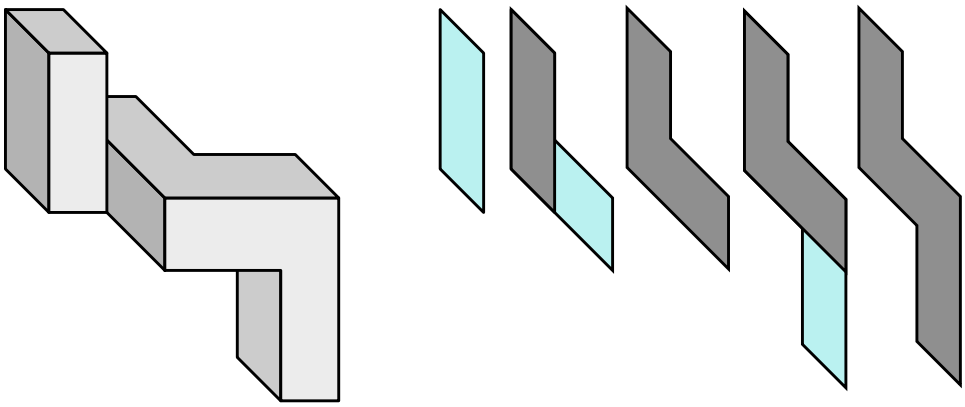

Figure 4.7: An orthogonally convex polyhedron and its positive x-layer shadows.

the shadow of the previous layer gives a projection of the actual polyhedron faces (after taking the closure).

These lemmas naturally imply an algorithm to compute the x^--faces: Sort all points with respect to their x-coordinates. For each layer, reconstruct the shadow by computing the orthogonally convex hull of the projections of the vertices in this and previous layers. Then, compute the closure of $\pi_i - \pi_{i-1}$ and extract the faces. Repeating this for all six directions, obtain all faces as well as the edges and incidence relations. This computes the only possible set of faces that could belong to an orthogonally convex polyhedron P with vertex set S.

This algorithm can easily be implemented in $O(n^2 \log n)$ time, if each π_i is reconstructed using the two dimensional orthogonally convex hull algorithm by Ottman et al. [OSSW84], which takes $O(n \log n)$ time and is applied $O(n)$ times (for each layer in each of six directions.)

The time complexity can be improved by dynamically updating the orthogonal convex hull as new points are added at each layer, and immediately reconstructing the faces. Overmars and Leeuwen [OvL81] showed that the orthogonally convex hull can be updated in $O(\log^2 n)$ time. But one can do better if only additions are allowed, namely $O(\log n)$ time per point is achievable. This is well-known for general convex hulls, see e.g. [PS85], and can easily be modified to handle orthogonally convex hulls by maintaining four staircases that describe the orthogonally convex hull.

To reconstruct the actual x^--faces with x-coordinate x_i, observe that all connected components of $\pi_i - \pi_{i-1}$ are incident to at least one point in layer i. Hence, exploring the boundary of π_i from vertices in layer i, one can walk around each connected piece of $\pi_i - \pi_{i-1}$, hence find each face F and explicitly list all vertices

and edges. The time to do this is proportional to the number of vertices in the faces found; over all faces and all layers this is $O(n)$ time. Once the ith layer is swept, π_{i-1} is no longer necessary and can be discarded, hence the space complexity is $O(n)$.

Theorem 4.2 *The edges and faces of an orthogonally convex polyhedron P can be reconstructed from its vertex coordinates in $O(n \log n)$ time.*

Note that this algorithm assumes that the input set of vertices is the exact set of vertices of the orthogonally convex polyhedron. Therefore, it cannot be used as a means of computing the orthogonally convex hull of an arbitrary point set, though whether it could be modified to do so is an interesting open problem.

This section considered only orthogonally convex polyhedra. Reconstructing arbitrary orthogonal polyhedra from their vertices appears to be considerably harder (partly because the answer is not unique, see Figure 4.2 on p. 82), and is left for future studies.

4.2 Rotated Orthogonal Polygons and Polyhedra

In Section 4.1, it is shown that the edges (and faces) of an orthogonal polygon or an orthogonally convex polyhedron can be reconstructed from its vertex coordinates. In this section, I consider rotated orthogonal polygons and polyhedra. In Section 4.2.1, I show that the edges of a rotated orthogonal polygon can be reconstructed from its vertex coordinates in $O(n \log n)$ time if the polygon is orthogonally convex, and in $O(n^2 \log n)$ time otherwise. In Section 4.2.2, I show that the edges and faces of a rotated orthogonally convex polyhedron can be reconstructed from its vertex coordinates in $O(n^2 \log n)$ time. I conjecture that this rotated orthogonal polyhedron is unique for the given vertex coordinates.

The following notation is used in the rest of this chapter: An *α-orthogonal polygon* (for $0 \leq \alpha < \pi/2$) is a polygon for which all edges have slope $\tan(\alpha)$ or $-\cot(\alpha)$. A *rotated orthogonal polygon* is an α-orthogonal polygon for some $0 \leq \alpha < \pi/2$. A *rotated orthogonal polyhedron* is a polyhedron obtained by applying some rotation matrix to an orthogonal polyhedron. Similarly define *α-orthogonally convex polygon* and *rotated orthogonally convex polygon/polyhedron*.

The operations involved in the following sections require multiplications of real numbers. Therefore, all operations are assumed to be done under the real RAM model of computing.

4.2.1 Rotated Orthogonal Polygons

In this section, it is shown that the boundary of a rotated orthogonal polygon can be reconstructed from its vertex coordinates, i.e., a solution is given to the following problem: Given a set S of points in $\mathbb{E}^2$, construct a rotated orthogonal polygon with vertex set S if one exists. First, I present a straightforward algorithm for all rotated orthogonal polygons that computes the rotation and reconstructs the boundary in $O(n^2 \log n)$ time. The computed rotation is unique according to a recent result by Löffler and Mumford [LM07]. Next, I show that for rotated orthogonally convex polygons, one can compute the edges in $O(n \log n)$ time.

The straightforward approach consists of trying all possible angles α for rotation, and for each of them, running the algorithm of Section 4.1.1. To find the possible angles, compute the (conventional) convex hull $CH(S)$ of S. For each of the edges on the convex hull, try the rotation α that makes this edge horizontal or vertical. Only one of these rotations can be the rotation of the rotated orthogonal polygon because of the following lemma:

Lemma 4.3 *Let P be an orthogonal polygon, and let $CH(P)$ be the convex hull of its vertices. Then there are at least four edges of P that are on the boundary of $CH(P)$. If P is orthogonally convex, then there are exactly four such edges, and they are edges of $CH(P)$.*

Proof. Let e_t be the topmost edge of P, i.e., the horizontal edge whose endpoints have the largest y coordinates. Note that there may be more than one such edge. The following observation holds for all of them. Since the endpoints of e_t have the largest y coordinates among all vertices, they must exist on the convex hull. Moreover, the convex hull does not have any other points which have greater y coordinates, therefore a convex hull edge must pass through these two points. Thus, e_t is on the boundary of $CH(P)$. The same holds for at least three more edges, namely the bottommost, rightmost and leftmost edges of P. Moreover, for an orthogonally convex polygon, there is only one topmost edge (and also only one bottommost, rightmost and leftmost edge). Therefore, for an orthogonally convex polygon, there are exactly four such edges. Each of these edges is a convex hull edge, otherwise one of the endpoints of the topmost convex hull edge is outside the polygon, which implies the convex hull is not minimal. ■

This lemma also holds for rotated orthogonal polygons, since the rotation does not change the convex hull. Hence the algorithm ("For each rotation suggested

by the convex hull, rotate the point set, then run the algorithm of Section 4.1.1") works correctly.

The time complexity of this algorithm is $O(n^2 \log n)$ for an input set of n points, since the convex hull has at most n edges and can be computed in $O(n \log n)$ time (see e.g. [PS85]). An interesting question is whether two α-orthogonal polygons with different rotations can exist for the same point set. Recently, Löffler and Mumford [LM07] reported that this is not the case, i.e. the orthogonal polygon obtained using this approach is unique.

Theorem 4.3 *The boundary of a rotated orthogonal polygon P can be reconstructed from its vertex coordinates in $O(n^2 \log n)$ time.*

The time complexity can be improved for rotated orthogonally convex polygons, hence the rest of this subsection deals with the following problem: Given a set S of points in $\mathbb{E}^2$, construct a rotated orthogonally convex polygon with vertex set S if one exists. For orthogonally convex polygons, with some pre-computation one can eliminate all but one possible rotation, and hence improve the time complexity to $O(n \log n)$. First, let us show that only one rotation is possible. This is known (from the result of Löffler and Mumford [LM07]), but the proof given here allows to also find the rotation quickly, as opposed to the non-constructive proof by Löffler and Mumford.

Given an edge e of a convex polygon C, let $H(e)$ be the closed half-disk of e (one half of the disk whose diameter is e) that intersects C. See also Figure 4.8(a).

Lemma 4.4 *Let P be an α-orthogonally convex polygon for some $0 \leq \alpha < \pi/2$, and let $e = (v, w)$ be an edge of $CH(P)$. If e is not an edge of P, then $H(e)$ contains at least one vertex, $u \neq v, w$ of P.*

Proof. Assume e is not an edge of P. Vertex v has two incident edges in P, one of slope $\tan(\alpha)$ and the other of slope $-\cot(\alpha)$. Furthermore, these edges must be inside $CH(P)$, which (since the convex hull has angles less than π and the two edges are at angle $\pi/2$) means that one of them must enter $H(e)$. Call the other endpoint of this edge v'. Similarly one can argue that one incident edge of w must enter $H(e)$; call its other endpoint w'. See also Figure 4.8(a).

Clearly, $v' \neq w$ and $w' \neq v$ since e is not an edge of P. If either one of v' and w' is inside $H(e)$, then we are done. But if both are outside $H(e)$, then the edges (v, v')

and (w, w') cross, because the two corresponding rays have perpendicular slopes and hence meet exactly on $H(e)$. But the boundary of a polygon is a polygonal curve, which is simple and must not cross itself, a contradiction. ■

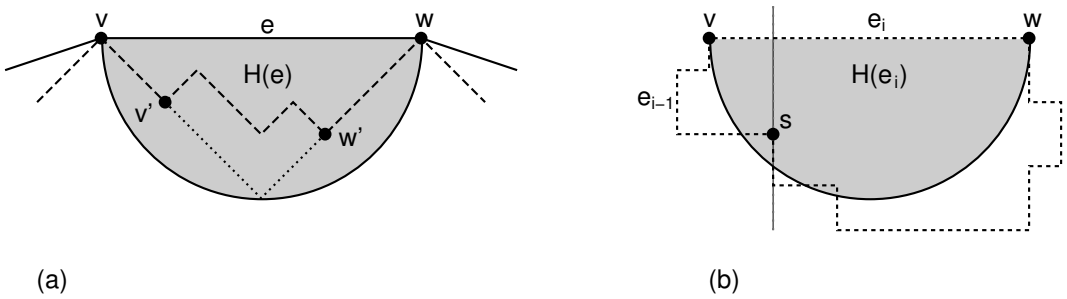

Figure 4.8: (a) $H(e)$ must contain another vertex. In this picture, $\alpha \approx \pi/6$. Polygon P is dashed. (b) e_{i-1} can at most be half as long as e_i. Polygon P is dashed.

Another important observation is the following: Let P be a rotated orthogonally convex polygon and let $e_0, \ldots, e_3$ be the four edges of P that are also edges of $CH(P)$, in order as encountered when walking along P (cf. Lemma 4.3). Call these the *extreme edges* of P. In what follows, the indices of $e_0, \ldots, e_3$ will be taken modulo 4, and $||e_i||$ denotes the length of e_i.

Lemma 4.5 *Let P be a rotated orthogonally convex polygon with extreme edges $e_0, \ldots, e_3$. For any i, if $H(e_i)$ contains some vertex of P other than the endpoints of e_i, then either $||e_i|| \geq 2||e_{i-1}||$ or $||e_i|| \geq 2||e_{i+1}||$.*

Proof. Apply a rotation to P, such that after the rotation e_i is horizontal and the rest of P has smaller y-coordinates than e_i. Let v and w be the endpoints of e_i with v left of w. Let $s \neq v, w$ be a vertex of P inside $H(e_i)$. See also Figure 4.8(b). The intuition behind the lemma is that s is on the "other side" of P, which imposes restrictions on how long the neighboring extreme edges can be.

To prove this formally, consider the vertical line through s. This line intersects P at s and somewhere along edge (v, w). It cannot intersect the boundary of P anywhere else by orthogonal convexity. So it splits the boundary of P into two chains, one of which is monotone in y-direction since P is orthogonally convex. Assume that the monotone chain is the one that contains s and v but not w; this chain contains e_{i-1} (the other case is similar, yielding a bound on e_{i+1}). Since the chain is monotone in y-direction, edge e_{i-1} (which is vertical) cannot be longer than the distance from s to (v, w), which is at most $||e_i||/2$ since s is inside $H(e_i)$. Therefore,

depending on which chain is monotone, either $||e_i|| \geq 2||e_{i-1}||$ or $||e_i|| \geq 2||e_{i+1}||$. ■

The contrapositive of this lemma implies that if e_i is the shortest extreme edge, then $H(e_i)$ must not contain any other vertex of the polygon, which yields the following crucial corollary:

Corollary 4.2 *Let P be a rotated orthogonally convex polygon. Then for at least one extreme edge e of P, $H(e)$ contains no other vertex of P.*

This corollary and Lemma 4.4 now imply that there exists only one rotated orthogonally convex polygon that corresponds to a set of vertex coordinates:

Theorem 4.4 *For a set S of points in plane, there exists at most one rotated orthogonally convex polygon whose set of vertices is exactly S.*

Proof. Assume for contradiction that S is the set of vertices of both an α-orthogonally convex polygon P and an α'-orthogonally convex polygon P' where $0 \leq \alpha \neq \alpha' < \pi/2$ (see Figure 4.9). By Corollary 4.2, there exists an extreme edge e of P for which $H(e)$ contains no other point of S. Edge e is on $CH(S) = CH(P) = CH(P')$, but it is not an edge of P', since P' has different slopes than P. So by Lemma 4.4, $H(e)$ contains some vertex of P', which is a point of S, a contradiction. ■

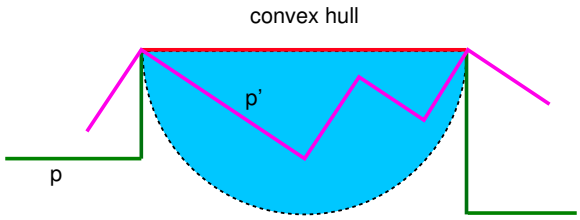

Figure 4.9: If two rotated orthogonally convex polygons have the same vertices, then one of them violates Lemma 4.4 or Corollary 4.2.

This result also helps to compute the edges of the polygon efficiently:

Theorem 4.5 *The edges of a rotated orthogonally convex polygon can be reconstructed from its vertex coordinates in $O(n \log n)$ time.*

Proof. First, compute the convex hull $CH(S)$ of S; this takes $O(n \log n)$ time (see e.g. [PS85]). Then, for each edge e of S, test whether $H(e)$ is empty. This can be done by pre-computing the Voronoi-diagram of S in $O(n \log n)$ time (see e.g. [PS85]). A Voronoi diagram of n points of set S is a partition of the plane into n regions associated to the points, such that the points in each region are closer to the associated point than to any other point in S. For each edge, $H(e)$ is empty if and only if the nearest points of the midpoint of edge e are the endpoints of edge e and no other point of S. One query in the Voronoi diagram is necessary per edge, and each such query takes $O(\log n)$ time [PS85]. Therefore, for all edges this can be read from the Voronoi diagram in $O(n \log n)$ total time.

If $H(e)$ is empty for some edge e, then by Lemma 4.4, e *must* be an edge of the rotated orthogonally convex polygon P. So, rotate S such that e becomes horizontal or vertical, and then apply the algorithm from Section 4.1.1 to compute the edges of P. One such edge must exist by Corollary 4.2. ■

This approach does not work for general orthogonal polygons, since Corollary 4.2 holds only for orthogonally convex polygons.

4.2.2 Rotated Orthogonally Convex Polyhedra

In this section, I show that one can compute the rotation, as well as the boundary edges and faces of a rotated orthogonally convex polyhedron in $O(n^2 \log n)$ time from its vertex coordinates. I conjecture that this rotation is unique. The formal problem statement is as follows: Given a set S of points in $\mathbb{E}^3$, construct a rotated orthogonally convex polyhedron with vertex set S if one exists.

In order to reconstruct the edges and faces, a trivial algorithm is to compute the convex hull of the vertices of the polyhedron, and then try all possible rotations that make two non-parallel convex hull faces lie in orthogonal planes. For each rotation, one can apply the algorithm presented in Section 4.1.2 to reconstruct the edges and faces of the orthogonally convex polyhedron. This approach takes $O(n^3 \log n)$ time in the worst case, because there may be as many as $O(n^2)$ rotations that must be tried for $O(n)$ faces of the convex hull. Clearly, one cannot reconstruct the edges and faces of a general orthogonal polyhedron with this approach as they are not uniquely determined by the vertex coordinates (recall Figure 4.2), and no algorithm is known for finding one possible answer.

Unfortunately, it is not known whether there are multiple rotations that can

allow an orthogonally convex polyhedron to be realized by the given vertex coordinates (Löeffler and Mumford study only the two-dimensional case.)

On the other hand, one can eliminate many of the rotations by applying the algorithm of Section 4.2.1 to each face of the convex hull. More precisely, assume that F is a face of the convex hull, and S_F is the set of all input points that lie in F. Apply the algorithm of Section 4.2.1 to S_F which takes $O(|S_F| \log |S_F|)$ time. If it succeeds, then the resulting rotated orthogonally convex polygon P_F can be a face of a rotated orthogonally convex polyhedron P, but only if the rotation is such that the face normal of F and edges of P_F become parallel to coordinate axes. After rotating S with this rotation, apply the algorithm of Section 4.1.2 to compute the edges and faces of the orthogonally convex polyhedron; this will reconstruct P if it exists. This takes $O(n \log n)$ time per face and $O(n^2 \log n)$ overall for $O(n)$ faces.

Theorem 4.6 *The edges and faces of a rotated orthogonally convex polyhedron can be reconstructed from its vertex coordinates in $O(n^2 \log n)$ time.*

There are some heuristics to decrease the number of rotations to be tried, but none of them leads to an improvement in the worst case. In particular, the three dimensional equivalent of Lemma 4.4 holds for an edge e of $CH(S)$, if $H(e)$ is replaced by a ball $B(e)$ spanned by e. Thus if $B(e)$ is empty for some edge e (which can be tested in $O(n \log n)$ time), then there is only one possible rotation. Unfortunately, there are orthogonally convex polyhedra for which the 3D equivalent of Corollary 4.2 does *not* hold, so there need not always be an edge for which $B(e)$ is empty (see Figure 4.10). Also, since Corollary 4.2 need not hold, it is not known whether the rotated orthogonally convex polyhedron in Theorem 4.6 is unique.

4.2.3 Open Problems

The main remaining open problem concerns describing a rotated orthogonally non-convex polyhedra using its vertex coordinates. It is shown that there are multiple orthogonal polyhedra that correspond to a set of vertex coordinates, but is it possible to reconstruct at least some polyhedron for orthogonally non-convex rotated/non-rotated point sets?

Another interesting open problem concerns time complexity depending on input. If the vertex coordinates are known to be integers within a limited range, is it possible to improve the run time similar as for other problems in computational geometry [Cha06]?

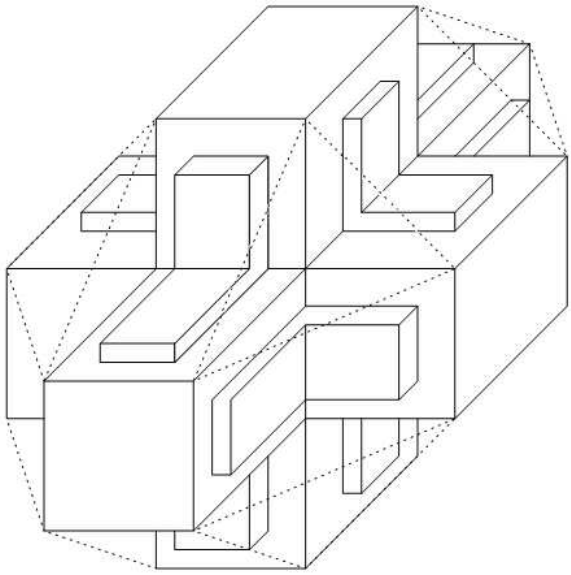

Figure 4.10: An orthogonally convex polyhedron for which there exists no edges on the convex hull with empty balls.

4.3 Additional Points

In this section, additional points (points other than the vertices of the polygon or polyhedron) are assumed to exist in the input. There are two possible types of additional points: points on the boundary of the orthogonal polygon/polyhedron, or points inside the polygon/polyhedron. The first type corresponds to the vertices of a polyhedral surface or a polygonal curve. The second type occurs when one is provided arbitrary points inside an orthogonal polygon or polyhedron. The formal problem statements are as follows:

Given a set S of points in $\mathbb{E}^2$ (or $\mathbb{E}^3$),

- Construct a (rotated) orthogonally convex polygon/polyhedron P, such that all vertices of P are in S, and all points in S are on the boundary of P.
- Construct a (rotated) orthogonally convex polygon/polyhedron P, such that all vertices of P are in S, and all points in S are in P.

4.3.1 Non-rotated Orthogonally Convex Polygons

Given the vertex coordinates of an orthogonal polygonal curve (i.e., additional points on the polygon boundary), the algorithm of O'Rourke presented in Section 4.1.1 does not work anymore, because vertices do not necessarily have exactly

one horizontal and one vertical edge. In fact, two different orthogonal polygonal curves that correspond to two different orthogonal polygons may have the same vertices (see Figure 4.11).

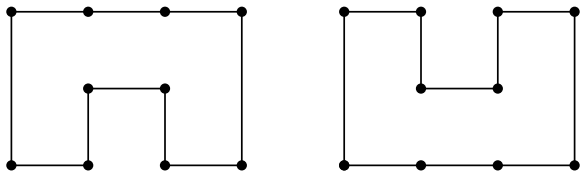

Figure 4.11: Same set of vertices, two different orthogonal polygons.

If the polygon is orthogonally convex, one can reconstruct it by computing the orthogonally convex hull of the point set in $O(n \log n)$ time [OSSW84]. However, for arbitrary orthogonal polygons determining whether an orthogonal polygonal curve exists for the given vertex coordinates is NP-hard [Rap86].

If additional points inside the polygon exist in the input, then for orthogonally convex polygons, one can again compute the orthogonally convex hull of the point set and the convex hull edges are the edges of the orthogonally convex polygon.

4.3.2 Non-rotated Orthogonally Convex Polyhedra

For orthogonally convex polyhedra, the algorithm presented in Section 4.1.2 still works even if there are additional points inside or on the boundary of the polyhedron. This is because the additional points do not affect the shadows created at each layer. The only change through the course of the algorithm is that now every vertex of the polyhedral surface which is not a vertex of the polyhedron may be in a separate layer. However, the overall runtime is still $O(n \log n)$: each additional point contributes $O(\log n)$ to shadow updates. Doing so the shadow is not changed and the point is identified as an additional point.

4.3.3 Rotated Orthogonally Convex Polygons

For additional points on the boundary of a rotated orthogonally convex polygon, the algorithm in Section 4.2.1 still works. The only small modification needed is that after the computation of the convex hull, one must merge all collinear convex hull edges. Then, the correct rotation can be computed as discussed in Section 4.2.1. After the rotation, one can compute the orthogonally convex hull of the point

set. The orthogonally convex hull edges then describe the boundary of the rotated orthogonally convex polygon.

For additional points inside a rotated orthogonally convex polygon, Corollary 4.2 does not hold anymore. However, one can still compute an orthogonally convex polygon using the input coordinates. First compute the convex hull of the input point set and then for each edge of the convex hull rotate the point set to make this edge horizontal. Then, compute the orthogonally convex hull of this rotated point set and check if all vertices of the hull are in the input set. If so, then the orthogonally convex hull edges are the edges of an orthogonally convex polygon. However, this approach does not produce a unique rotated orthogonal polygon as shown in Figure 4.12. The running time of this approach is $O(n^2 \log n)$, since in the worst case the orthogonally convex hull of the point set is computed $O(n)$ times.

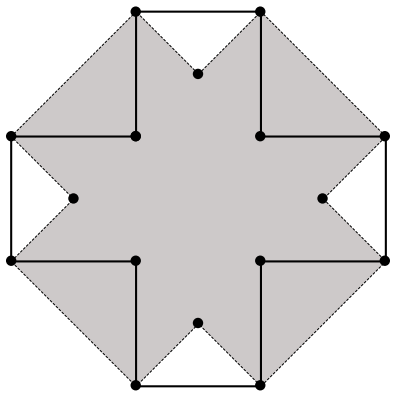

Figure 4.12: Two rotated orthogonally convex polygons constructed from same input coordinates: additional points inside the polygons are allowed.

For additional points on the boundary of a rotated orthogonal polygon, the NP-hardness result of Rappaport [Rap86] still holds after a correct rotation is computed. For additional points inside a rotated orthogonal polygon the problem is open.

4.3.4 Rotated Orthogonally Convex Polyhedra

For additional points on the boundary of a rotated orthogonally convex polyhedron, the algorithm presented in Section 4.2.2 does not work anymore, because the additional points on each face makes it impossible to apply the algorithm of Section 4.2.1. Therefore, the best known runtime is $O(n^3 \log n)$.

For additional points inside a rotated orthogonally convex polyhedron once again the algorithm of Section 4.2.1 for rotated orthogonally convex polygons does

not work for the faces of the polyhedron, and the best available algorithm is the first algorithm presented in Section 4.2.2 for rotated orthogonally convex polyhedra.

Non-uniqueness of general orthogonal polyhedra reconstructed from the vertex coordinates was shown in Figure 4.2. This holds also for rotated orthogonal polyhedra. The problem of reconstructing some rotated orthogonal polyhedron from the given point coordinates is left open.

Figure 4.13 summarizes the existing results mentioned in this chapter as well as new results presented in this thesis.

		non–rotated		rotated	
		2D	3D	2D	3D
orthogonal	vertices of polygon/ polyhedron	O(nlog n) [O'R88] unique	unknown ––––– not unique	$O(n^2 \log n)$ Sec. 4.2.1 unique, [LM07]	unknown ––––– not unique
	vertices of p. curve/ p. surface	NP–hard [Rap86] not unique	unknown ––––– not unique	NP–hard [Rap86] not unique	unknown ––––– not unique
	additional points inside	O(nlog n) [OSSW84] not unique	unknown ––––– not unique	unknown ––––– not unique	unknown ––––– not unique
orthogonally convex	vertices of polygon/ polyhedron	O(nlog n) [O'R88] unique	O(nlog n) Sec. 4.1.2 unique	O(nlog n) Sec. 4.2.1 unique	$O(n^2 \log n)$ Sec. 4.2.2 –––––
	vertices of p. curve/ p. surface	O(nlog n) [OSSW84] unique	O(nlog n) Sec. 4.3.2 unique	O(nlog n) Sec. 4.3.3 unique	$O(n^3 \log n)$ Sec. 4.3.4 not unique
	additional points inside	O(nlog n) [OSSW84] unique	O(nlog n) Sec. 4.3.2 unique	$O(n^2 \log n)$ Sec. 4.3.3 not unique	$O(n^3 \log n)$ Sec. 4.3.4 not unique

Figure 4.13: A summary of results cited or presented in Chapter 4. The first row in each cell shows the runtime of the fastest known algorithm that computes the edges (and faces), if an algorithm exists. The second row is either a citation of an existing work that presented this algorithm, or a reference to a section in this thesis. The third row shows whether the computed edges (and faces) using this algorithm is unique or not. If the third row is empty, then I neither have a uniqueness proof nor a counterexample that shows non-uniqueness.

Chapter 5

Conclusion

In this thesis, I studied reconstructing orthogonal polyhedra and polyhedral surfaces from their dual graphs, dihedral angles, facial angles, edge lengths or vertex coordinates. I provided fast polynomial time algorithms for reconstructing missing boundary elements to obtain a boundary representation of the orthogonal polyhedron or orthogonal polyhedral surface.

In Chapter 3, I studied reconstructing genus-0 orthogonal polyhedra and polyhedral surfaces from their dual graphs, dihedral angles, facial angles and edge lengths up to translation and rotation.

In Section 3.1, I showed that the vertex coordinates of an orthogonal polyhedral surface can be reconstructed from its dual graph, dihedral angles, facial angles and edge lengths in linear time. The same also holds for an orthogonal polyhedron if the graph of the polyhedron is connected. Finally, if the graph of the polyhedron is disconnected, then I show that the vertices of an orthogonal polyhedron can be reconstructed from its dual graph, dihedral angles, facial angles, edge lengths and relative translation and rotation information per each connected component of the graph of the polyhedron in linear time. These results extend to general polyhedra and polyhedral surfaces.

In Section 3.2, I show that the dihedral angles of a genus-0 orthogonal polyhedra can be reconstructed from its dual graph, facial angles and edge lengths in linear time if the graph of the polyhedron is connected. If the graph of the polyhedron is not connected, then reconstructing a working set of dihedral angles becomes NP-hard. Finally in Section 3.2.2, I provide a linear time algorithm for identifying the flat dihedral angles of a genus-0 orthogonal polyhedral surface from its dual graph and facial angles.

In Section 3.3, I show that the facial angles of a genus-0 orthogonal polyhedron can be reconstructed from its dual graph, dihedral angles and edge lengths in linear time. Then, I show that reconstructing the facial angles of a polyhedral surface from its dual graph, dihedral angles and edge lengths is NP-hard. I partially extend the results to general polyhedra and show that if all vertices have three incident edges, then the facial angles can be reconstructed from the dual graph, dihedral angles and edge lengths in linear time.

In Section 3.4, I show that a working set of edge lengths for a quadrangulated orthogonally convex polyhedral surface can be computed easily. However, this edge set is not the only one that realizes the given dual graph, facial angles and dihedral angles as an orthogonal polyhedral surface. The general cases of orthogonal polyhedra and polyhedral surfaces are left open.

In Section 3.5, I study reconstructing multiple unknown properties, such as reconstructing both the dihedral angles and the facial angles from the dual graph and the edge lengths. The problems I study in this section are still open.

In Chapter 4, I study the problem of reconstructing rotated and non-rotated orthogonal polygons and polyhedra from their vertex coordinates. In Section 4.1.2, I show that the edges and faces of an orthogonally convex polyhedron can be reconstructed from its vertex coordinates in $O(n \log n)$ time. In Section 4.2.1, I show that the edges of a rotated orthogonally convex polygon can be reconstructed from its vertex coordinates in $O(n \log n)$ time. In Section 4.2.2, I show that the edges and faces of a rotated orthogonally convex polyhedron can be reconstructed from its vertex coordinates in $O(n^2 \log n)$ time. In Section 4.3, I study the reconstruction of rotated and non-rotated orthogonal polygons and polyhedra from points sets that include all of their vertices as well as some additional points.

5.1 Open Problems

Many open problems have been mentioned throughout the thesis already. The following is a list of open problems that are not directly related to any results in this thesis.

Open Problem 5.1 *What happens if the orthogonality restriction is relaxed in a controlled fashion? For example, if diagonal edges and faces are allowed to exist?*

An important direction for future research is relaxing the orthogonality restriction by allowing other edge directions/face normals. This allows higher precision for

polyhedra whose shape is not truly orthogonal. Most of the algorithms presented in this thesis strongly depend on orthogonality. However, especially the algorithms presented in Chapter 3 are good candidates for being extended in this direction.

Open Problem 5.2 *What other geometric properties of an orthogonal polyhedral surface or polyhedron may be used to reconstruct it?*

In this thesis, I focused on the dual graph and four geometric properties of a polyhedron: the dihedral angles, the facial angles, the edge lengths and the vertex coordinates. However, there are many more properties such as the face normals, face areas, edge directions, etc. Is it possible to reconstruct the dihedral angles from edge directions? Is it possible to reconstruct the edge lengths from face areas?

Open Problem 5.3 *Are all facial angles, dihedral angles, edge lengths and the whole graph necessary to reconstruct the vertex coordinates? Is it possible to reconstruct the dihedral angles if only some facial angles are known? Is it possible to reconstruct the facial angles if only some dihedral angles are known?*

It is possible to remove one facial angle from each face and compute it later from the total turn-angle-value of the face. Similarly, if the facial angles of a face are known, the lengths of two edges can be computed from the lengths of the other edges provided that they are not parallel. Clearly, the sets of information I used as input in this thesis are not minimal. Then, what information is minimal and sufficient to reconstruct an orthogonal polyhedron?

List of References

[AA97] A. Aguilera and D. Ayala. Orthogonal polyhedra as geometric bounds in constructive solid geometry. In *Proceedings of the Fourth ACM Symposium on Solid Modeling and Applications*, pages 56–67. ACM Press, 1997.

[AA98] A. Aguilera and D. Ayala. *Orthogonal Polyhedra: Study and Application*. PhD thesis, Universitat Politecnica de Catalunya, 1998.

[ABCO+01] Marc Alexa, Johannes Behr, Daniel Cohen-Or, Shachar Fleishman, David Levin, and Claudio T. Silva. Point set surfaces. In *VIS '01: Proceedings of the conference on Visualization '01*, pages 21–28, Washington, DC, USA, 2001. IEEE Computer Society.

[ABJN85] D. Ayala, P. Brunet, R. Juan, and I. Navazo. Object representation by means of nonminimal division quadtrees and octrees. *ACM Trans. Graph*, 4(1):41–59, 1985.

[ADF85] S. Ansaldi, K. DeFloriani, and B. Falcidieno. An edge-face relational scheme for boundary representations. *Computer Graphics Forum*, 4(4):319–332, December 1985.

[Ahu80] N. Ahuja. Dot pattern processing using Voronoi polygons as neighborhoods. In *International Conference on Pattern Recognition*, pages 1122–1127, 1980.

[Ale05] Alexander Danilovich Alexandrov. *Convex Polyhedra*. Springer-Verlag, Berlin, 2005. Translated from the 1950 Russian edition by N. S. Dairbekov, S. S. Kutateladze and A. B. Sossinsky, With comments and bibliography by V. A. Zalgaller and appendices by L. A. Shor and Yu. A. Volkov.

[Alt86] Simon L. Altmann. *Rotations, Quaternions, and Double Groups*. Oxford University Press, 1986.

[AZ99] M. Aigner and G. Ziegler. *Proofs from THE BOOK*. Springer Verlag, 1999.

[BGRT99] P. Bose, F. Gomez, P. Ramos, and G. Toussaint. Drawing nice projections of objects in space. *Journal of Visual Communication and Image Representation*, 10(2):155–172, 1999.

[BHS80] I. C. Braid, R. C. Hillyard, and I. A. Stroud. Stepwise construction of polyhedra in geometric modeling. In K. W. Brodlie, editor, *Mathematical Methods in Computer Graphics and Design*, pages 123–141. Academic Press, Boston, MA, 1980.

[BI06] A. I. Bobenko and I. Izmestiev. Alexandrov's theorem, weighted Delaunay triangulations, and mixed volumes. http://arxiv.org/abs/math/0609447, 2006. arXiv:math/0609447v1 [math.DG].

[BLS05] T. Biedl, A. Lubiw, and J. Sun. When can a net fold to a polyhedron? *Computational Geometry: Theory and Applications*, 31(3):207–218, 2005.

[BLS07] Therese Biedl, Anna Lubiw, and Michael J. Spriggs. Cauchy's theorem and edge lengths of convex polyhedra. In Frank K. H. A. Dehne, Jörg-Rüdiger Sack, and Norbert Zeh, editors, *Algorithms and Data Structures, 10th International Workshop, (WADS 2007)*, volume 4619 of *Lecture Notes in Computer Science*, pages 398–409. Springer, 2007.

[BMP99] O. Bournez, O. Maler, and A. Pnueli. Orthogonal polyhedra: Representation and computation. In *Hybrid Systems: Computation and Control*, volume 1569 of *Lecture Notes in Computer Science*, pages 46–60, Nijmegen, The Netherlands, 29–31 March 1999.

[BN85] P. Brunet and I. Navazo. Geometric modelling using exact octree representation of polyhedral objects. In C. E. Vandoni, editor, *Eurographics '85*, pages 159–169, Diagonal 647, 08080 Barcelona, Spain, 1985. Department de Methods Informatics Esc. Tec. Sup. Ing. Industrials, Elsevier Science Publishers B. V. (North–Holland).

[Cha06] Timothy M. Chan. Point location in $o(\log n)$ time, Voronoi diagrams in $o(n \log n)$ time, and other transdichotomous results in computational geometry. In *FOCS*, pages 333–344. IEEE Computer Society, 2006.

[Con79] Robert Connelly. The rigidity of polyhedral surfaces. *Mathematics Magazine*, 52(5):275–283, 1979.

[Cox73] Harold Scott Macdonald Coxeter. *Regular polytopes*. Dover Publications, New York, 1973.

[Cro97] Peter R. Cromwell. *Polyhedra*. Cambridge University Press, Cambridge, England, June 1997.

[Day90] A. M. Day. The implementation of an algorithm to find the convex hull of a set of 3D points. *ACM Transactions on Graphics*, 9(1):105–132, January 1990.

[Dek95] Boris V. Dekster. Convex hulls of spatial polygons with a fixed convex projection. *Contributions To Algebra and Geometry*, 36(1), 1995.

[Die05] Reinhard Diestel. *Graph Theory (Graduate Texts in Mathematics)*. Springer, August 2005.

[DO07] Erik D. Demaine and Joseph O'Rourke. *Geometric Folding Algorithms: Linkages, Origami, Polyhedra*. Cambridge University Press, July 2007.

[Ede87] H. Edelsbrunner. *Algorithms in combinatorial geometry*. Springer, Berlin, 1987.

[EFK01] Markus Eiglsperger, Sándor P. Fekete, and Gunnar W. Klau. *Orthogonal Graph Drawing*, volume 2025 of *Lecture Notes in Computer Science*, chapter 6, pages 121–171. Springer Berlin / Heidelberg, 2001.

[FP05] Maksym Fedorchuk and Igor Pak. Rigidity and polynomial invariants of convex polytopes. *Duke Mathematics Journal*, 129(2):371–404, 2005.

[Fra87] W. R. Franklin. Polygon properties calculated from the vertex neighborhoods. In *Proceedings of the* 3^{rd} *Annual Symposium on Computational Geometry (SCG '87)*, pages 110–118, 1987.

[FW96] E. Fink and D. Wood. Fundamentals of restricted-orientation convexity. *Information Sciences*, 92(1-4):175–196, 1996.

[GGH+89] W. Gellert, S. Gottwald, M. Hellwich, H. Kastner, and H. Kunstner, editors. *VNR Concise Encyclopedia of Mathematics*, chapter 12, pages 261–282. Van Nostrand Reinhold, New York, 2^{nd} edition, 1989.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, 1979.

[Gru67] B. Grunbaum. *Convex Polytopes*. Interscience Publishers, 1967.

[Grü94] B. Grünbaum. Hamiltonian polygons and polyhedra. *Geombinatorics*, 3:83–89, January 1994.

[Han82] Patrick M. Hanrahan. Creating volume models from edge-vertex graphs. *SIGGRAPH Computer Graphics*, 16(3):77–84, 1982.

[Has05] Masud Hasan. *Reconstruction and visualization of polyhedra using projections*. PhD thesis, School of Computer Science, University of Waterloo, 2005.

[Hop79] Reinhold Hoppe. Ergänzung des Eulerschen Satzes von den Polyedern. *Archiv der Mathematik und Physik*, 63:100–103, 1879.

[HPR93] Lenwood S. Heath, Praveen K. Paripati, and John W. Roach. Representing polyhedra: faces are better than vertices. *Computational Geometry: Theory and Applications*, 3(6):327–351, 1993.

[HTT03] F. Hurtado, G. T. Toussaint, and J. Trias. On polyhedra induced by point sets in space. In *Canadian Conference on Computational Geometry*, pages 107–110, 2003.

[Lak76] Imre Lakatos. *Proofs and Refutations*. Cambridge University Press, New York, NY, 1976.

[Lau97] Aldo Laurentini. How many 2D silhouettes does it take to reconstruct a 3D object? *Computer Vision and Image Understanding*, 67(1):81–87, 1997.

[LM07] Maarten Löffler and Elena Mumford. Rotating rectilinear graphs. In 17^{th} *Fall Workshop on Computational and Combinatorial Geometry*, Hawthorne, New York, November 2007.

[Luc06] Brendan Lucier. Unfolding and reconstructing polyhedra. Master's thesis, School of Computer Science, University of Waterloo, 2006.

[Man88] M. Mantyla. *An Introduction to Solid Modeling*. Computer Science Press, Rockville, Md, 1988.

[MS82] M. J. Mantyla and R. Sulonen. GWB: A solid modeler with euler operators. *IEEE Computer Graphics and Applications*, 2(7):17–31, 1982.

[MT04] B. Marlin and G. T. Toussaint. Constructing convex 3-polytopes from two triangulations of a polygon. *Computational Geometry: Theory and Applications*, 28(1):41–47, 2004.

[Mun00] James Munkres. *Topology*. Prentice Hall, Upper Saddle River, New Jersey, second edition, 2000.

[MW80] G. Markowsky and M. A. Wesley. Fleshing out wire frames. *IBM Journal of Research and Development*, 24:582–597, 1980.

[Nav89] I. Navazo. Extended octree representation of general solids with plane faces: Model structure and algorithms. *Computers & Graphics*, 13:5–16, 1989.

[NG88] I. V. Nagendra and Uday G. Gujar. 3-D objects from 2-D orthographic views–A survey. *Computers & Graphics*, 12(1):111–114, 1988.

[O'C74] J. F. O'Callaghan. Computing the perceptual boundaries of dot patterns. *Computer Graphics Image Processing*, 3:141–162, 1974.

[O'R88] Joseph O'Rourke. Uniqueness of orthogonal connect-the-dots. In *Machine Intelligence and Pattern Recognition: Computational Morphology*, pages 97–104. Elsevier Science Publishers B.V., Amsterdam, The Netherlands, 1988.

[O'R98] J. O'Rourke. *Computational geometry in C (2nd ed.)*. Cambridge University Press, New York, NY, USA, 1998.

[OSSW84] T. Ottmann, E. Soisalon-Soininen, and D. Wood. On the definition and computation of rectilinear convex hulls. *Information Sciences*, 33:157–171, 1984.

[OvL81] M. H. Overmars and J. van Leeuwen. Maintenance of configurations in the plane. *Journal of Computer and System Sciences*, 23(2):166–204, 1981.

[Pat05] M. Patrignani. Complexity results for three-dimensional orthogonal graph drawing. In P. Healy and N. S. Nikolov, editors, *Graph Drawing*, volume 3843 of *Lecture Notes in Computer Science*, pages 368–379. Springer, 2005.

[Pen89] Michael A. Penna. A shape from shading analysis for a single perspective image of a polyhedron. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-11(6):545–554, 1989.

[PH77] F. 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.

[PS85] Franco P. Preparata and Michael I. Shamos. *Computational geometry: an introduction*. Springer-Verlag New York, Inc., New York, NY, USA, 1985.

[Rap86] David Rappaport. On the complexity of computing orthogonal polygons from a set of points. Technical Report TR-SOCS-86.9, McGill University, 1986.

[Req80] A. G. Requicha. Representations for rigid solids: Theory, methods, and systems. *ACM Comput. Surv.*, 12(4):437–464, 1980.

[RV77] A. A. G Requicha and H. B. Voelcker. Constructive solid geometry. Technical Report TM-25, Production Automation Project, University of Rochester, New York, 1977.

[RW88] Gregory J. E. Rawlins and Derick Wood. Ortho-convexity and its generalizations. In *Machine Intelligence and Pattern Recognition: Computational Morphology*, pages 137–152. Elsevier Science Publishers B.V., Amsterdam, The Netherlands, 1988.

[Sab96] Idzhad Sabitov. The volume of a polyhedron as a function of its metric and algorithmical solution of the main problems in the metric theory of polyhedra. *International School-Seminar Devoted to the N. V. Efimov's Memory*, pages 64–65, 1996.

[Sil81] C. Silva. Alternative definitions of faces in boundary representations of solid objects. Technical Memo 36, Production Automation Project, Univ. of Rochester, Rochester, NY, 1981.

[SSP00] Guoling Shen, Takis Sakkalis, and Nicholas M. Patrikalakis. Representational validity of boundary representation models. *Computer-Aided Design*, 32(12):719–726, 2000.

[Ste64] Hugo Steinhaus. *One Hundred Problems in Elementary Mathematics*. Dover Publications, New York, 1964.

[Sug84] K. Sugihara. A necessary and sufficient condition for a picture to represent a polyhedral scene. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 6(5):578–586, 1984.

[Sug86] K. Sugihara. *Machine interpretation of line drawings*. MIT Press, Cambridge, Massachusetts, 1986.

[THCM04] Marco Tarini, Kai Hormann, Paolo Cignoni, and Claudio Montani. Polycube-maps. *ACM Transactions on Graphics*, 23(3):853–860, 2004.

[WG93] W. Wang and Georges G. Grinstein. A survey of 3D solid reconstruction from 2D projection line drawings. *Computer Graphics Forum*, 12(2):137–158, 1993.

[Whi32] H. Whitney. Non-separable and planar graphs. *Transactions of the American Mathematical Society*, 34:339–363, 1932.

[Whi94] W. Whiteley. How to describe or design a polyhedron. *Journal of Intelligent and Robotic Systems*, 11(1 - 2):135–160, March 1994.

[WM81] M. A. Wesley and G. Markowsky. Fleshing out projections. *IBM Journal of Research and Development*, 25(6):934–954, 1981.

[YCT94] Qing-Wen Yan, C. L. Philip Chen, and Zesheng Tang. Efficient algorithm for the reconstruction of 3D objects from orthographic projections. *Computer-Aided Design*, 26(9):699–717, 1994.