Universitat Politècnica de Catalunya

PhD. Thesis

Doctoral Program in Biomedical Engineering

Skeletal representations of orthogonal shapes

Jonàs Martínez Bayona

Advisors:

Dr. Marc Vigo Anglada
Dra. Núria Pla Garcia

Barcelona
November 5, 2013

Resum

L'esquelet d'un objecte representa propietats topològiques i geomètriques de l'objecte i alhora redueix la seva dimensió. Els esquelets s'utilitzen en diversos àmbits de la ciència i atreuen l'atenció de nombrosos investigadors. En l'àmbit del BioCAD, l'anàlisi de propietats estructurals com ara la porositat dels biomaterials requereix el càlcul previ de l'esquelet. A mesura que les imatges 3D μCT es fan més grans, es requereixen algorismes eficients i robusts que calculin esquelets més simples. L'esquelet més popular és el "medial axis", definit com el conjunt de punts de l'objecte que tenen almenys dos punts més propers en la frontera de l'objecte. Malauradament el medial axis és molt sensible a petites pertorbacions de la frontera de l'objecte. És a dir, un petit canvi en la frontera de l'objecte pot implicar un canvi considerable del seu medial axis. A més, el càlcul exacte del medial axis només és possible per determinats tipus d'objectes. Per exemple, el medial axis dels poliedres té una estructura no lineal i és difícil de calcular de manera robusta. Aquests problemes van motivar l'aparició d'aproximacions del medial axis. Existeixen dos mètodes d'aproximació principals: aquells en què l'objecte s'aproxima amb un altre classe d'objecte i aquells en què la mètrica Euclidiana es substitueix per una altra mètrica aproximada.

La principal aportació d'aquesta tesi és la simplificació conjunta, a l'hora de calcular l'esquelet, de l'objecte d'entrada i de la mètrica considerada. L'objecte d'entrada s'aproxima amb un objecte ortogonal, que són polígons o poliedres delimitats per segments o cares, respectivament, paral·lels als eixos de coordenades o plans coordenats. En el mateix sentit, la mètrica Euclidiana es substitueix per la mètrica L_∞ o mètrica de Chebyshev. Tot i la seva estructura simple, existeixen pocs treballs en la literatura sobre esquelets d'objectes ortogonals. Gran part dels esforços s'han dedicat a imatges i volums binaris, que són un subconjunt dels objectes ortogonals. En aquesta tesi s'introdueixen dos nous tipus d'esquelet basats en aquest paradigma: el "*cube skeleton*" i el "*scale cube skeleton*". El cube skeleton es compon de segments rectes o cares planes i és homotopicament equivalent a l'objecte d'entrada. El scale cube skeleton es basa en el cube skeleton i presenta una família d'esquelets que són menys sensibles a pertorbacions de la frontera de l'objecte. A més, es presenten els algorismes necessaris per calcular el cube skeleton de polígons i poliedres així com el scale cube skeleton de polígons. Diversos resultats experimentals confirmen l'eficiència, la robustesa i l'aplicació pràctica de tots els mètodes presentats.

Abstract

Skeletal representations are important shape descriptors which encode topological and geometrical properties of shapes and reduce their dimension. Skeletons are used in several fields of science and attract the attention of many researchers. In the BioCAD field, the analysis of structural properties such as porosity of biomaterials requires the previous computation of a skeleton. As the size of three-dimensional μCT images become larger, efficient and robust algorithms that extract simple skeletal structures are required. The most popular and prominent skeletal representation is the medial axis, defined as the shape points which have at least two closest points on the shape boundary. Unfortunately, the medial axis is highly sensitive to noise and perturbations of the shape boundary. That is, a small change of the shape boundary may involve a considerable change of its medial axis. Moreover, the exact computation of the medial axis is only possible for a few classes of shapes. For example, the medial axis of polyhedra is composed of non planar surfaces, and its accurate and robust computation is difficult. These problems led to the emergence of approximate medial axis representations. There exists two main approximation methods: the shape is approximated with another shape class or the Euclidean metric is approximated with another metric.

The main contribution of this thesis is the combination of a specific shape and metric simplification. The input shape is approximated with an orthogonal shape, which are polygons or polyhedra enclosed by axis-aligned edges or faces, respectively. In the same vein, the Euclidean metric is replaced by the L_∞ or Chebyshev metric. Despite the simpler structure of orthogonal shapes, there are few works on skeletal representations applied to orthogonal shapes. Much of the efforts have been devoted to binary images and volumes, which are a subset of orthogonal shapes. Two new skeletal representations based on this paradigm are introduced: the *cube skeleton* and the *scale cube skeleton*. The cube skeleton is shown to be composed of straight line segments or planar faces and to be homotopical equivalent to the input shape. The scale cube skeleton is based upon the cube skeleton, and introduces a family of skeletons that are more stable to shape noise and perturbations. In addition, the necessary algorithms to compute the cube skeleton of polygons and polyhedra and the scale cube skeleton of polygons are presented. Several experimental results confirm the efficiency, robustness and practical use of all the presented methods.

Acknowledgments

Foremost, I would like to express my gratitude to my advisors, Núria Pla and Marc Vigo, for his patience and guidance throughout my research.

I am also grateful to my thesis committee members, Tao Ju and Evanthia Papadopoulou, for their suggestions and insightful comments.

My sincere thanks also goes to Oswin Aichholzer and Guido Brunnett for welcoming me into their research groups, where I gained valuable insights.

I want to thank my fellow labmates Pascual Abellán, Irving Cruz, Eloy Félix, Marc Freixas, Marta Hidalgo, Sergio Moya, and Eduard Vergés who have accompanied me during these four years. I would like to extend my gratitude to the faculty and staff in the research group GIE.

Finally, I would like to thank my family and friends for their support and love. This thesis is dedicated to them.

This thesis has been partially supported by the project TIN2008-02903 and TIN2011-24220 of the Spanish government and by the IBEC (Bioengineering Institute of Catalonia).

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Skeletal representations are a powerful tool used in a broad number of scientific fields. A recent survey of skeleton applications [198] presents its practical use in shape matching and retrieval, geographical and urban models, archeology, visual perception, generation of artistic content, motion analysis and animation, medical imaging, biology, chemistry, and molecular design among others. Thus, skeletons still attract the attention of many researchers and numerous scientific articles are devoted to them.

Skeletons reduce the dimension of shapes, try to capture their geometric and topological properties, and seek to understand how these structures encode local and global features. Intuitively, the skeleton of a shape is a thin version of that shape (see Figure 1.1). However, there is not a unique way to define a skeleton. The skeleton usually emphasizes shape properties such as its connectivity, topology, length or direction.

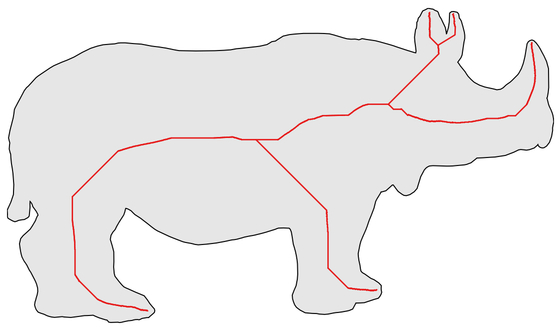


Figure 1.1: A skeletal representation of a planar shape is shown in red.

In recent years, the spectacular development of bioengineering has yield to the apparition of a new application field of computer aided design, the BioCAD, aimed at the design of new models for biological structures and simulation methods for biological processes. One of the challenges of the BioCAD field is to understand the morphology of the pore space of bone, biomaterials, rocks, etc. There exist several approaches that obtain a network with pores and connections between them or that characterize plate and rod elements, which rely on a previous computation of a skeleton. Vergés et al. [216] presented a framework to analyse structural properties such as porosity of bioimplants for bone reconstruction, which rely on the previous computation of the porous space skeleton. Their method mimics the operation of the mercury porosimeter device that intrudes this liquid into the sample by applying increasing pressures forcing it to traverse smaller narrows or throats. The skeleton is used to detect the propagation paths (see Figure 1.2). As real datasets obtained from μCT tend to be very large, robust and fast algorithms to extract their skeletal representation are required.

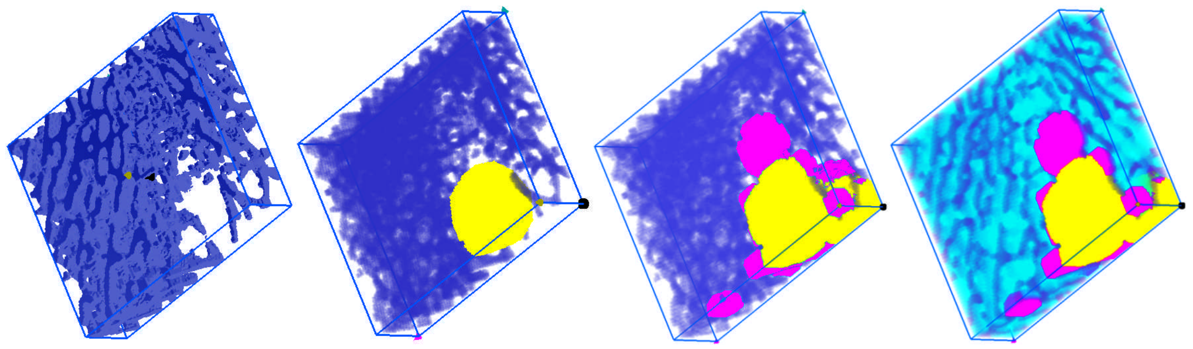


Figure 1.2: Porous material shown in blue. Filling process of a porous material guided by its skeleton (courtesy of Vergés et al. [216]).

The most prominent skeletal structure is the medial axis [46]. The medial axis of a shape can alternatively be defined as the locus of centers of maximally inscribed balls or as the set of points which have at least two closest points on the shape boundary. A significant disadvantage of medial axis is related to its high instability under small perturbations of a shape: two similar shapes may have very different medial axes. An approach to deal with instability consists in removing irrelevant parts of the medial axis. However, it is known that removing these parts can lead to undesirable results with shapes that have features at different scales [21]. A reliable computation of stable skeletal representations is still considered to be an open problem, despite the intense efforts of prior years.

In addition, the medial axis of polyhedra is composed of quadric surfaces and its combinatorial complexity is between $O(n^2)$ and $O(n^{3+\epsilon})$ for any $\epsilon > 0$. Thus, fewer algorithms are able to compute the exact medial axis of polyhedra [75]. The high instability of the medial axis lead to the development of approximate approaches. There exist two main approximation methods:

- The shape is approximated with another shape class for which the computation of a skeleton is easier. For example, the medial axis of a planar shape can be approximated by a subset of the Voronoi diagram of a finite set of shape boundary points [52].
- The Euclidean metric is approximated with another metric. For example, the medial axis can be approximated by considering a convex distance function [8].

The main contribution of this thesis is the intelligent combination of simplifying both the shape and the distance function. More specifically, the input shape is approximated by an orthogonal polygon or polyhedron and the L_∞ metric is considered to compute the skeleton (see Figure 1.3b). It will be shown that this combination gives skeletons with bounded combinatorial complexity, composed of straight line segments or planar faces with restricted orientation, and that can be computed robustly and efficiently.

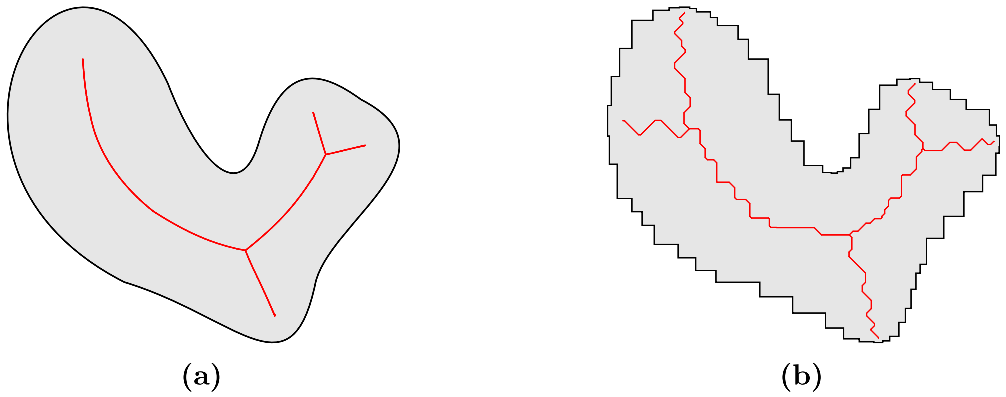


Figure 1.3: **(a)** Medial axis of a simple planar shape shown in red. **(b)** Shape approximate by an orthogonal polygon and L_∞ metric considered. Resulting skeletal representation shown in red.

Orthogonal polygons and polyhedra have their edges and faces axis-aligned, respectively. Binary images and volumes are a discrete subset of orthogonal polygons and polyhedra respectively. Orthogonal representations are often used in computational geometry, where their constrained structure has enabled advances on complex problems for arbitrary shapes. Sudhalkar et al. [208] observed that the L_∞ metric behaves well with discrete shapes. Despite the simpler structure of orthogonal shapes, only few works on skeletal representations applied to orthogonal shapes have been found in the literature. Much of the efforts have been focused on binary images and volumes.

Classical methods to compute the skeleton of discretized datasets are based on thinning methods. These methods iteratively erode the shape boundary in order to obtain a skeleton. The time complexity of thinning methods is constrained by the size of the discretized space. Thus, computing the skeleton of huge datasets with thinning approaches is very time-consuming. On the contrary, the time complexity of all the methods presented in this thesis are constrained by the number of boundary vertices of the orthogonal shape.

Most existing geometric algorithms are not designed for numerical robustness; they are based on a machine model with exact real arithmetic. Thus, floating point arithmetic may cause implementations to fail. For a practical use, it is required to implement correct and robust algorithms for carrying out geometric computations. However, geometric computations have an underlying difficulty that sets the problem apart from other application areas in computer science and engineering [124].

Robustness issues are a major problem when dealing with orthogonal shapes. There is a high probability of suffering from geometric degeneracies. Furthermore, discretized datasets usually contain non-manifold features that must be considered. This thesis will also cover how to robustly obtain skeletons by treating geometric degeneracies and non-manifold features.

This thesis addresses the aforementioned stability problems of skeletal representations. A new skeletal representation named *cube skeleton* is introduced and proven to be homotopically equivalent to the shape. Moreover, the cube skeleton reduces the dimension of the input shape and it is composed of line segments or planar polygons with restricted orientation. In addition, the *scale cube skeleton* is a new skeletal representation based on the cube skeleton that defines a simplification scheme where shape features are ignored first, if they are small relative to their neighborhood. A family of more stable skeletons can be computed with the scale cube skeleton.

The input of all the algorithms presented along this thesis requires that the orthogonal shape is represented as a boundary representation (B-Rep). Unfortunately, most of the available datasets are discrete and need to be converted to a boundary representation. An efficient method that allows the conversion of a wide range of different representations is presented. In addition, the presented algorithms require the orthogonal polyhedron to only be composed of trihedral vertices, which are vertices with three incident faces. An algorithm to correctly modify the topology of a polyhedron, such that is only composed of trihedral vertices, is also presented.

1.1 Objectives

Summarizing, the objectives of this thesis are the following:

1. Design and implement an algorithm to obtain the boundary representation of orthogonal polygons and polyhedra, from different types of solid representation schemes.
2. Design and implement an algorithm to trihedralize an orthogonal polyhedron.
3. Implement an existing algorithm to compute the L_∞ Voronoi diagram of orthogonal polygons, with a specific implementation aimed to the three-dimensional case.
4. Design and implement a novel robust algorithm to compute the L_∞ Voronoi diagram of orthogonal polyhedra.
5. Introduce a more stable skeletal representation applied to orthogonal shapes.

1.2 Document outline

This document is structured as follows. Chapter 2 introduces the notation used along this thesis and the previous definitions. Chapter 3 presents the state of the art in skeletal representations: the different types and its properties, the algorithms to compute them and their applications. Chapter 4 describes an algorithm to extract the boundary representation of orthogonal polygons and polyhedra. Chapter 5 describes a procedure to trihedralize an orthogonal polyhedron, required to compute its L_∞ Voronoi diagram. Chapter 6 specific implementation to compute the L_∞ Voronoi diagram of orthogonal polygons efficiently and robustly. Chapter 7 presents an algorithm to compute the L_∞ Voronoi diagram of orthogonal polyhedra efficiently and robustly. Chapter 8 introduces the cube skeleton and its properties. Chapter 9 introduces the scale cube skeleton and its properties. Finally, there is an annex with a glossary explaining terms and acronyms used along the thesis and the list of referenced bibliography.

1.3 Machine specifications

All the results presented in this thesis were carried on a 64-bit PC with a 3.9GHz core and 16 GB of RAM memory.

Chapter 2

Notation and previous definitions

In this chapter, the notation and previous definitions applied throughout this thesis are introduced. The dimension of the Euclidean space is denoted by n. For a point $x \in \mathbb{R}^n$, x_i denotes its coordinate $i \in \{1 \ldots n\}$. Table 2.1 shows basic notation employed along this thesis.

Notation	Description
$\lvert A\rvert$	Cardinality of the set A.
∂A	Boundary of the set A.
A^c	Complement of the set A.
$\mathrm{cl}\,(A)$	Closure of the set A.
$\mathrm{int}\,(A)$	Interior of the set A.
$A \setminus B$	Relative complement of the set B in A.
$\inf\,(A)$	Infimum of the set $A \subset \mathbb{R}$.
$\mathrm{abs}\,(x)$	Absolute value of $x \in \mathbb{R}$.
$\lvert\lvert x\rvert\rvert$	Euclidean norm of $x \in \mathbb{R}^n$.
$d\,(x, y)$	Euclidean distance between two points $x, y \in \mathbb{R}^n$.
$\hat{v}$	Unit vector of a non-zero vector $v \in \mathbb{R}^n$.
$[x, y]$	Closed segment between points $x, y \in \mathbb{R}^n$.

Table 2.1: Basic notation.

Shape representations

In the following, some shape representations are introduced.

Definition 1. *A* planar straight line graph *is a set of non-intersecting line segments.*

Definition 2. *A* polygon *is a bounded subset of* $\mathbb{R}^2$ *enclosed by a finite set of line segments, called* edges. *The edges meet only at their endpoints, called* vertices. *A* polygon face *is a single connected component of polygon edges.*

An *axis-aligned* object is an object in n-dimensional space whose shape is aligned with the coordinate axes of the space.

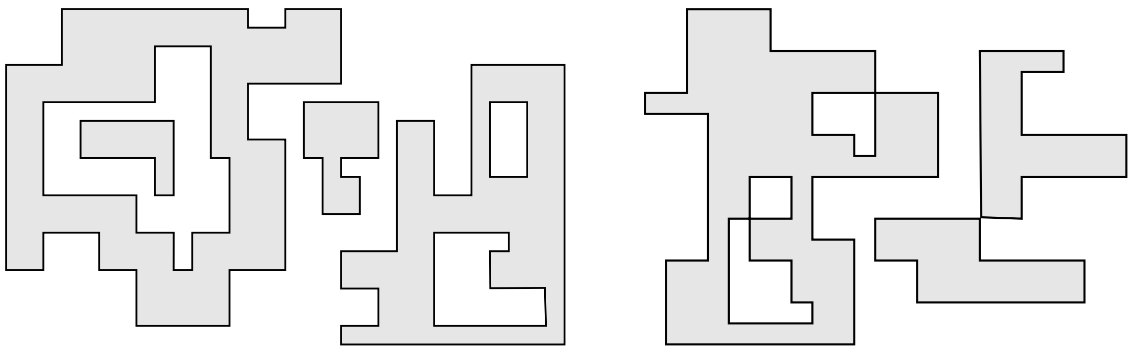
Definition 3. *An* orthogonal polygon *is a polygon enclosed by axis-aligned edges. An edge is called* vertical *if it is parallel to the ordinate and* horizontal *if it is parallel to the abscissa.*

Definition 4. *An* orthogonal manifold polygon *is an orthogonal polygon such that every vertex is shared by a vertical and a horizontal edge (see Figure 2.1a).*

Definition 5. *An* orthogonal pseudomanifold polygon *is a generalization of an orthogonal manifold polygon, which allows non-manifold vertices. A* non-manifold vertex *is a polygon vertex with four incident edges (see Figure 2.1b).*

Property 1. *The* interior angle *between two neighboring edges of an orthogonal polygon is either $\frac{\pi}{2}$ or $\frac{3\pi}{2}$.*

Definition 6. *A manifold vertex of an orthogonal polygon is called* convex *if their incident edges form an interior angle of $\frac{\pi}{2}$, and* reflex *if the angle is $\frac{3\pi}{2}$.*



(a) **(b)**

Figure 2.1: **(a)** An orthogonal manifold polygon. **(b)** An orthogonal pseudomanifold polygon.

Definition 7. *A* polyhedron *is a bounded subset of $\mathbb{R}^3$ enclosed by a finite set of non-intersecting planar polygons, called* faces.

Definition 8. *A* trihedral vertex *is a vertex of a polyhedron that has exactly three incident faces.*

Definition 9. *An* orthogonal polyhedron *is a polyhedron enclosed by axis-aligned faces.*

Definition 10. *An* orthogonal manifold polyhedron *is an orthogonal polyhedron such that every edge of a face is shared by exactly one other face.*

Definition 11. *An* orthogonal pseudomanifold polyhedron *is a generalization of an orthogonal manifold polyhedron, which allows non-manifold edges or vertices. A* non-manifold edge *is an edge adjacent to four faces and a* non-manifold vertex *is a non-trihedral vertex.*

Definition 12. *An* orthogonal shape *is an orthogonal polygon or polyhedron.*

Definition 13. *A* boundary representation model (B-Rep) *represents a shape by a collection of its boundary elements. Boundary representation models are composed of two parts: geometry (surfaces, curves and points) and topology. The main topological items are: faces, edges and vertices. A face is a bounded portion of a surface; an edge is a bounded piece of a curve and a vertex lies at a point. Other elements are a* shell*, a set of connected faces, and a* loop*, a circuit of edges bounding a face.*

The vertices of an orthogonal shape are denoted as v_j, edges as e_j, and faces as f_j, where the subscript $j \in \mathbb{N}$ is their identifier. In addition, a face of an orthogonal polyhedron can also be denoted as f_{dj}, where the subscript $d \in \{x, y, z\}$ indicates that the face f is parallel to the axis-aligned plane $\{x, y, z\} \setminus d$. For example, f_{1x} is parallel to the axis-aligned plane yz, and as a consequence all its points have a constant x coordinate.

The L_∞ distance

The L_∞ distance or Chebyshev distance is a metric defined on a vector space where the distance between two vectors is the greatest of their differences along any dimension.

Definition 14. *The L_∞ distance between two points $x, y \in \mathbb{R}^n$ is:*

$$d_\infty(x, y) = \max_i (\text{abs}(x_i - y_i))$$

Definition 15. *Let x be a point of $\mathbb{R}^n$ and let g be a bounded subset of $\mathbb{R}^n$. The L_∞ distance from x to g is:*

$$d_\infty(x, g) = \inf \{d_\infty(x, y) : \forall y \in g\}$$

Observe that as g is a bounded set, it follows that the distance $d_\infty(x, g)$ is also bounded.

Definition 16. *Let x be a point of $\mathbb{R}^n$ and let G be a disjoint collection of bounded subsets of $\mathbb{R}^n$. The L_∞ distance from x to G is:*

$$d_\infty(x, G) = \inf \{d_\infty(x, g) : \forall g \in G\}$$

The L_∞ Voronoi diagram

The *Voronoi diagram* is a space decomposition determined by a metric and a family of objects in the space. Voronoi diagrams play an important role in computational geometry and other related topics [27]. The classical definition of the Voronoi diagram of polygons [201, 175] or polyhedra [96, 206] assumes that the boundary is decomposed disjointly into vertices, open edges, and open faces, in the case of polyhedra, which are referred to as *sites*. Then, the Voronoi region associated to each site s is the set of points inside the polygon or polyhedron that are closer to s, under a given metric, than to any other site. However, in the case of orthogonal shapes (see Definition 12) and the L_∞ metric, this classical definition gives two or three-dimensional regions, that are L_∞ equidistant to several sites laying on a common axis-aligned line or plane, and the resulting Voronoi diagram does not have a meaningful application because it does not reduce the original dimension (see Figure 2.2).

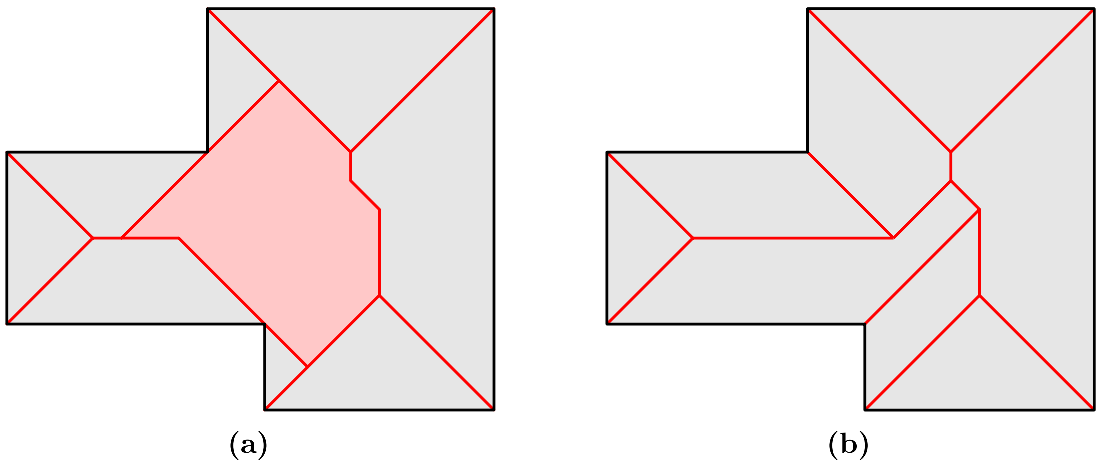


Figure 2.2: A simple orthogonal polygon and its Voronoi diagram. The points of the orthogonal polygon which do not belong to any Voronoi region are shown in red. **(a)** L_∞ Voronoi diagram that considers vertices and open edges as sites. **(b)** L_∞ Voronoi diagram that only considers open edges as sites.

In order to overcome this problem, two complementary aspects of the Voronoi diagram definition are slightly modified with respect to the classical definition. First, the sites are restricted to be open edges and faces of polygons and polyhedra, respectively.

Definition 17. *Let $\mathcal{P}$ be an orthogonal polygon. Then,* $\text{sites}(\mathcal{P})$ *is the set of open edges of $\mathcal{P}$.*

Definition 18. *Let $\mathcal{P}$ be an orthogonal polyhedron. Then,* $\text{sites}(\mathcal{P})$ *is the set of open faces of $\mathcal{P}$.*

Second, the set of closest sites from a point x, which is referred to as $\epsilon(x, \text{sites}(\mathcal{P}))$ (see Definition 19), only considers the L_∞ distance to a point in a site, which are now open, and not as an infimum. Thus, as it can be seen in the following definition, the boundary of sites is effectively ignored.

Definition 19. *Let x be a point of $\mathbb{R}^n$ and let G be a disjoint collection of bounded subsets of $\mathbb{R}^n$. $\epsilon(x, G)$ is the subset of G defined as:*

$$\epsilon(x, G) = \{g \in G : \exists y \in g, d_\infty(x, y) = d_\infty(x, G)\}$$

Now, the L_∞ Voronoi regions and Voronoi diagram can be properly defined.

Definition 20. *Let $\mathcal{P}$ be an orthogonal shape and s a site of $\mathcal{P}$. The* Voronoi region *of s is the set:*

$$Vor_\infty(s, \mathcal{P}) = \{x \in \mathcal{P} : \epsilon(x, \text{sites}(\mathcal{P})) = \{s\}\}$$

Definition 21. *The L_∞* Voronoi diagram *of an orthogonal shape $\mathcal{P}$ is the space partition induced by its Voronoi regions. That is, the tuple:*

$$V_\infty(\mathcal{P}) = (Vor_\infty(s, \mathcal{P}))_{s \in \text{sites}(\mathcal{P})}$$

Definition 22. *The set $E_\infty(\mathcal{P})$ of an orthogonal shape $\mathcal{P}$ is the closure of the set of points in $\mathcal{P}$ that do not belong to any Voronoi region:*

$$E_\infty(\mathcal{P}) = \text{cl}\left(\mathcal{P} \setminus \bigcup_{s \in \text{sites}(\mathcal{P})} Vor_\infty(s, \mathcal{P})\right)$$

The classical definition of Voronoi diagrams considers that the points that do not belong to any Voronoi region have two or more closest sites. Similarly, the set of points from $E_\infty(\mathcal{P})$ can be classified in one of the following cases:

- $|\epsilon(x, \text{sites}(\mathcal{P}))| \geq 2$. This point follows the classical definition of Voronoi diagrams.
- $\epsilon(x, \text{sites}(\mathcal{P})) = \emptyset$. In this case, the minimum L_∞ distance from x to the sites of $\mathcal{P}$ is only attained at the boundary of a site s. As the boundary of s is shared with at least one another site, x is at minimum L_∞ distance to the closure of two or more sites (considering the usual definition, obtained as an infimum). Thus, it can be considered that x also follows the classical definition of Voronoi diagrams.

Observe that under Definition 20 the L_∞ Voronoi regions are not necessarily open sets. There may exist some point x, such that the set $\epsilon(x, \text{sites}(\mathcal{P}))$ contains a single site s, and x is contained in the boundary of $Vor_\infty(s, \mathcal{P})$ (see Figure 2.3). The closure of Definition 22 ensures that these points also belong to $E_\infty(\mathcal{P})$. In fact, these points are at the same infimum L_∞ distance to two or more sites and also follow the classical definition of Voronoi diagrams.

It will be shown in Chapters 6 and 7 that if an orthogonal polygon or polyhedron has collinear edges or coplanar faces, respectively, then the set $E_\infty(\mathcal{P})$ may not reduce the dimension of the orthogonal shape. Moreover, the three dimensional $E_\infty(\mathcal{P})$ set may be not homotopical equivalent to the input shape $\mathcal{P}$ (see Chapter 8).

Property 2. *The L_∞ Voronoi regions of a polygon are bounded by straight line segments, which can be axis-aligned or rotated $\frac{\pi}{4}$ radians [175].*

Definition 23. *Each Voronoi region of an orthogonal polygon is bounded by* Voronoi edges*. The endpoint of a Voronoi edge, that is not a polygon vertex, is called* Voronoi vertex*.*

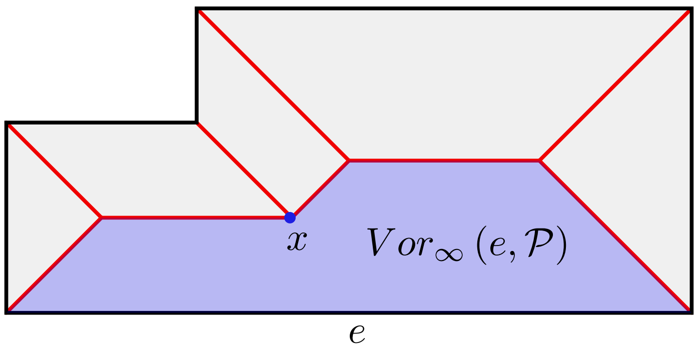


Figure 2.3: A simple orthogonal polygon $\mathcal{P}$ and its Voronoi diagram. The L_∞ Voronoi region associated to the open edge e, is shown in blue. The point x also drawn in blue, only contains e in the set $\epsilon(x, \text{sites}(\mathcal{P}))$ and belongs to the boundary of $Vor_\infty(e, \mathcal{P})$. The point x is also contained in $E_\infty(\mathcal{P})$.

Analogously to the two-dimensional case, it will be seen that the L_∞ Voronoi regions of an orthogonal polyhedron are bounded by planar faces with restricted orientation.

Definition 24. *Each Voronoi region of an orthogonal polyhedron is bounded by* Voronoi faces*. The segments bounding a Voronoi face, that are not polyhedron edges, are called* Voronoi edges*. The endpoint of a Voronoi edge, that is not a polyhedron vertex, is called* Voronoi vertex.

Sweep line algorithms

A relevant type of algorithm used extensively in computational geometry is the *sweep line algorithm* [80]. Different algorithms presented in this thesis are based on the sweep line technique. The strategy in a sweep line algorithm is to sweep the plane with a line. While the sweep line moves, information is maintained by the *status data structure* regarding the intersection between the sweep line and the processed object. This information does not change, except at certain special points called *event points*. An event has an associated priority that corresponds to the time when it occurs. The events are ordered in a priority queue, usually implemented as a self-balancing binary search tree. The sweep line approach can be generalized to three dimensions by considering a sweeping plane instead of a line. See Algorithm 1 for an overview of the sweep line algorithm.

Homotopy equivalence

Speaking colloquially, a topological space is homotopy equivalent to another if it can be "continuously deformed", by bending, shrinking and expanding operations, into the another. For example, a solid sphere is homotopy equivalent to a point.

Definition 25. *Let X and Y be two topological spaces, and $f, g : X \to Y$ continuous maps. A* homotopy *from f to g is a continuous function $F : X \times [0,1] \to Y$ satisfying:*

$$F(x,0) = f(x), F(x,1) = g(x), \forall x \in X$$

Algorithm 1 Sweep line algorithm.

```
Let $\mathbb{E}$ be an empty priority queue of events
Let $\mathbb{W}$ be the status data structure
Insert in $\mathbb{E}$ the initial events
**while** $|\mathbb{E}| > 0$ **do**
    $e \leftarrow \textit{pop\_element}\,(\mathbb{E})$
    Process event $e$
    Insert new events, if they exist, in $\mathbb{E}$
    Update $\mathbb{W}$
**end while**
```

If such a homotopy exists, then f is homotopic *to g.*

Definition 26. *Two topological X spaces and Y are* homotopy equivalent *if there exist continuous maps $f : X \to Y$ and $g : Y \to X$, such that the composition $g \circ f$ is homotopic to the identity map of X and $f \circ g$ is homotopic to the identity map of Y [100].*

Homotopy equivalence is important because many properties are *homotopy invariant*, that is, they respect the relation of homotopy equivalence. As an example, the *Euler characteristic* is a homotopy invariant between homotopy equivalent polygons or polyhedra, and as a consequence they must have the same number of connected components, cycles, holes and tunnels.

Chapter 3

Related work

Contents

3.1 Skeletal representations

In this section some properties of skeletal representations are characterized. Thereafter the medial axis, the straight skeleton and other skeletal representations are introduced. The reader is referred to the book by Siddiqi and Pizer [198] and to surveys [179, 38, 185, 21, 210] for a detailed review of skeletal representations.

3.1.1 Properties

A skeletal representation must not necessarily meet all the following properties and some of them may be mutually exclusive.

Dimension reduction and homotopy equivalence

Skeletal representations normally reduce the dimensionality of the input shape [207, 142, 73, 185]. That is, if the shape has dimension $n > 1$, its skeleton must have a dimension lower than n. Homotopy equivalence (see Definition 26) is also a commonly desired property [225, 195, 150, 61]. However, it should be noted that homotopy equivalence does not necessarily imply dimension reduction.

Hierarchical representation

The skeleton parts can be organized in a hierarchical structure [171, 133, 72, 127]. A hierarchical representation could concisely describe the relation between the different regions of the skeleton and the shape. Moreover, a multiresolution hierarchy could define multiple skeletons of the same shape with different resolutions. The hierarchical representation may be intrinsic to the formulation of the skeleton or may be directly derived from it.

Reconstructability

A skeletal representation can exactly or approximately reconstruct the original shape [44, 10, 14]. Thus, the skeleton becomes a transformation of the original shape. Usually, this transformation associates to each point of the skeleton with the nearest distance, under some metric, to the shape boundary.

Stability and smoothness

A skeletal representation may be desired to be stable under perturbations of the shape boundary [61, 63, 21]. Commonly, boundary perturbations are defined either by parametric continuity C^n or by the Hausdorff distance. The smoothness of a skeleton is strongly connected to its stability under perturbations.

Invariance under isometric transformations

A skeletal representation may be required to be invariable under isometric transformations [171, 121, 73, 60]. An isometric transformation preserves the distances between points of a shape. Any reflection, translation and rotation is an isometry on Euclidean spaces.

Shape abstraction

A skeletal representation may be desired to represent the shape concisely, capturing only those shape aspects that are relevant for a specific application [40, 54, 107]. For example, it may be required that the skeleton intuitively resembles the shape and that the relevant features for the application are captured.

3.1.2 Medial axis

In the late 1960s, Blum [46] proposed the *medial axis* as a tool for biological shape recognition. In the technical literature, the definition of medial axis and skeleton might change from one author to another. The definitions given in [150] are adopted here. Let $\mathcal{O}$ be an open subset of $\mathbb{R}^n$. The strictly positive, real valued function $\mathcal{R}$ defined on $\mathcal{O}$ is the Euclidean distance to the boundary:

$$\mathcal{R}(x) = \inf \{d(x, y) : \forall y \in \partial\mathcal{O}\}$$

For any point $x \in \mathcal{O}$ the set of closest boundary points $\Gamma(x)$ is defined as:

$$\Gamma(x) = \{y \in \partial\mathcal{O} : d(x, y) = \mathcal{R}(x)\}$$

Then, the medial axis $\mathcal{M}(\mathcal{O})$ of $\mathcal{O}$ is the set of points with at least two closest points on the boundary of the shape (see Figure 3.1):

$$\mathcal{M}(\mathcal{O}) = \{x \in \mathcal{O} : |\Gamma(x)| \geq 2\}$$

The medial axis together with the associated radius function $\mathcal{R}$ is called the medial axis transform (MAT). The MAT can be used to reconstruct the shape of the original domain. The medial axis transform of $\mathcal{O}$ is denoted by $MAT(\mathcal{O})$.

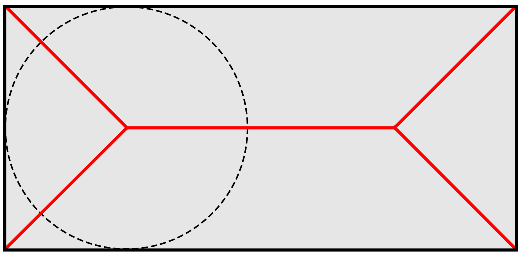


Figure 3.1: Medial axis of a simple polygon is shown in red. A ball of $MAT(\mathcal{O})$ is shown with dotted lines.

The *skeleton* is defined as the set of centres of maximal balls. An open ball $\mathbb{B} \subset \mathcal{O}$ is maximal if every ball that contains $\mathbb{B}$ and is contained in $\mathcal{O}$ equals $\mathbb{B}$. The concepts of skeleton and medial axis are used interchangeably by some authors, while some other authors regard them as related, but not the same. In fact, the medial axis is a subset of the skeleton which is, in turn, a subset of the closure of the medial axis [162].

Homotopy equivalence

It has been proved by Lieutier [150] that $\mathcal{O}$ is homotopically equivalent (see Definition 26) to its medial axis. The vector function $\nabla(x)$, introduced in [106], has an important role in the proof of homotopical equivalence. Let x be a point of $\mathbb{R}^n \setminus \mathcal{M}(\mathcal{O})$ and let $\Theta(x)$ be the center of the smallest ball enclosing the set $\Gamma(x)$. Then, the function $\nabla(x)$ is:

$$\nabla(x) = \frac{x - \Theta(x)}{\mathcal{R}(x)}$$

This vector function is an extension of the gradient of $\mathcal{R}$ outside $\mathcal{O} \setminus \mathcal{M}(\mathcal{O})$ and is used to construct a continuous deformation in order to prove the homotopy equivalence.

Wolter [225] previously proved the homotopy equivalence in the particular case of open sets with a C^2 boundary in $\mathbb{R}^n$ and open sets with a piecewise C^2 boundary in the plane. The proof relies on a deformation retract. If $x \in \mathcal{O} \setminus \mathcal{M}(\mathcal{O})$, there is a unique point $y \in \partial\mathcal{O}$ that minimizes the Euclidean distance to x. Then, the point $m(x)$ is the intersection between the half line that passes through x and y, and the medial axis. The map m is well defined and continuous and therefore it is possible to construct a deformation retract. In addition, Sherbrook et al.[195] proved that a path connected polyhedral solid without cavities has a path connected medial axis.

Finiteness properties

The medial axis can be composed of infinitely many branches, even if the shape boundary is C^∞ smooth. Choi et al. [67] proved that in the smoother case of a piecewise analytic boundary in the plane, the medial axis is made of a finite number of curves. Chazal and Soufflet [63] extended the finiteness property to semianalytic bounded open sets which are bounded and open subsets for which every point has a neighborhood defined by a finite system of analytic equations and inequalities. Giblin and Kimia [105] proposed a classification of each medial axis point in $\mathbb{R}^3$ based upon the order of contact between the shape boundary and the maximal ball associated with the skeletal point. The points are then organized into sheets, curves and points (see Figure 3.2).

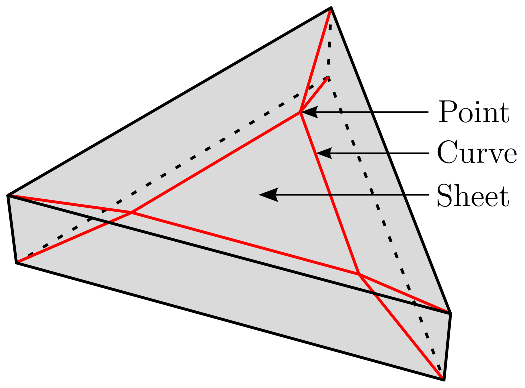


Figure 3.2: Types of points of the medial axis of a polyhedron.

Stability properties

The medial axis has high sensitivity to small changes in the shape boundary [21] (see Figure 3.3). However, small modifications of a shape do not usually affect the entire medial axis. This is so because the medial axis is *semicontinuous* with respect to the

Hausdorff distance [162]. More formally, let X and Y be two subsets of $\mathbb{R}^n$. The one-sided Hausdorff distance $d_H(X|Y)$ of X from Y is:

$$d_H(X|Y) = \sup_{x\in X}\inf_{y\in Y} d(x,y)$$

The Hausdorff distance between X and Y is defined as the maximum between both one-sided Hausdorff distances:

$$d_H(X,Y) = \max\{d_H(X|Y), d_H(Y|X)\}$$

As the function $\mathcal{M}$ is semicontinuous, for every bounded open subset $X \subset \mathbb{R}^n$ and for every $\lambda > 0$, there exists $\epsilon > 0$ such that for every subset $Y \subset \mathbb{R}^n$:

$$d_H(X^c, Y^c) < \epsilon \rightarrow d_H(\mathcal{M}(X)|\mathcal{M}(Y)) < \lambda$$

In other words, a small Hausdorff distance between the complements of X and Y implies that the one-sided Hausdorff distance of the medial axis of X from the medial axis to Y is bounded [21]. Note that this bound is only in one direction.

The medial axis is continuous when C^2 perturbations are applied to some classes of shape in $\mathbb{R}^3$[63]. However, this result does not provide a solution to practical problems because usually data describing shapes are given with noise which is not C^2 smooth.

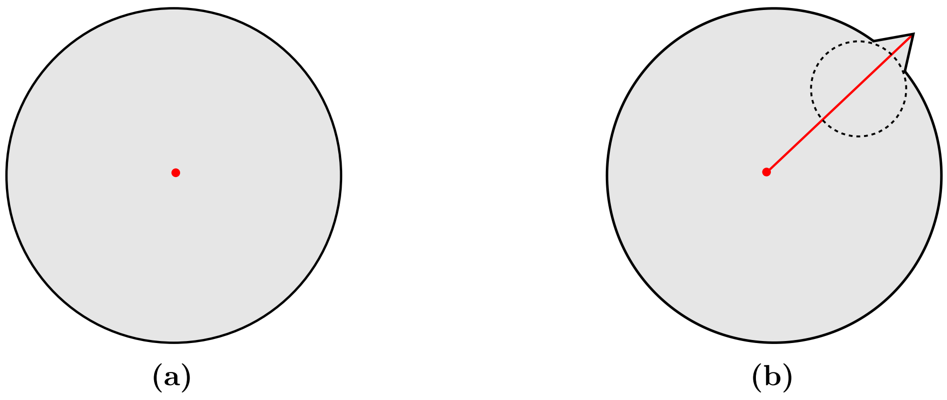


Figure 3.3: (a) A small perturbation in the boundary of a circle **(b)** causes a large change of its medial axis, shown in red.

Giblin and Kimia [104] classified the transitions of the medial axis in three dimensions. Transitions are changes of the medial axis encountered under deformations of shape. An enumeration of all possible transitions is done by examining the type of contact between the $MAT(\mathcal{O})$ balls and the shape boundary.

The medial axis of polygons and polyhedra

In the case of polygons and polyhedra, the medial axis is a subset of the generalized Voronoi diagram of the vertices, edges, and faces of the shape. The combinatorial complexity of the Voronoi diagram of a polyhedron is high: the upper bound on its combinatorial complexity is between $O(n^2)$ (see Figure 3.4) and $O(n^{3+\epsilon})$ for any positive ϵ, where n is its number of faces, edges and vertices [192]. Koltun and Sharir [140] have shown that under a convex distance function, as an example the L_1 and L_∞ metric, the combinatorial complexity reduces to $O(n^{2+\epsilon})$ for any $\epsilon > 0$. The medial axis of a polygon is composed of linear and quadratic curve segments. The medial axis of a polyhedron is a piecewise algebraic of degree two: each quadric surface is the bisector of two boundary features (vertices, edge, or faces). The complex structure of the medial axis of polyhedra is an important drawback.

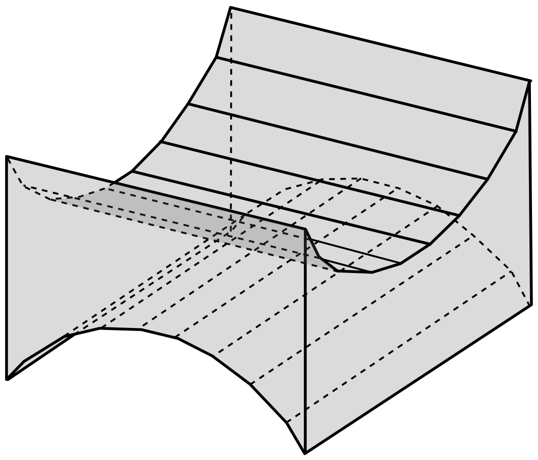


Figure 3.4: Lower bound example of a polyhedron which have a medial axis with $O(n^2)$ faces. Every upper face interacts with the lower faces, giving the quadratic bound.

3.1.3 The λ medial axis

Chazal and Lieutier [62] presented the λ *medial axis* of $\mathcal{O}$, denoted by $\mathcal{M}_\lambda(\mathcal{O})$. They proved its stability under Hausdorff distance perturbations of $\mathcal{O}^c$. $\mathcal{M}_\lambda(\mathcal{O})$ is a subset of $\mathcal{M}(\mathcal{O})$ and it is defined as the set of points for which any ball containing the set of closest points on the boundary has a radius of at least λ. That is:

$$\mathcal{F}(x) = \inf\{r, \exists y \in \mathbb{R}^n, \mathbb{B}_{y,r} \supset \Gamma(x)\}$$

And the λ medial axis is defined as:

$$\mathcal{M}_\lambda = \{x \in \mathcal{O} : \mathcal{F}(x) \geq \lambda\}$$

The *weak feature size* $wfs(\mathcal{O})$ is the minimum distance between $\mathcal{O}^c$ and the critical points of $\mathcal{O}$. A point x is critical if $\nabla(x) = 0$. Then, $\mathcal{M}_\lambda(\mathcal{O})$ is homotopically equivalent to $\mathcal{O}$ when $\lambda < wfs(\mathcal{O})$.

3.1.4 The scale axis transform

Giesen et al. [107] presented the *scale axis transform*, that is based upon the medial axis transform. The scale axis transform induces a family of skeletons that yield a family of successively simplified skeletons, where shape features are ignored first if they are small relative to their neighborhood. Its definition is based on the scaling of the MAT balls. For $s > 0$ the s-scaled shape $\mathcal{O}_s$ is defined as:

$$\mathcal{O}_s = \bigcup_{\mathbb{B}_{c,r} \in MAT(\mathcal{O})} \mathbb{B}_{c,sr}$$

Then, the s-scale axis transform $SAT(\mathcal{O})$ is defined as the set balls associated with the medial axis of the grown shape $\mathcal{O}_s$ that are scaled back by a factor of $1/s$, that is:

$$SAT(\mathcal{O}) = \{\mathbb{B}(c, r/s) : \mathbb{B}_{c,r} \in MAT(\mathcal{O}_s)\}$$

The study of topological properties of the scale axis is done by analysing the properties of the gradient field induced by the multiplicative distance to the medial balls of $\mathcal{O}$. As the λ medial axis, the scale axis transform is homotopical equivalent to the shape when the weak feature size, associated to the gradient of the multiplicative distance, is greater than the scaling factor s. Unfortunately, the scale axis transform is not necessarily a subset of the $\mathcal{O}$ and it may lie outside.

3.1.5 Straight skeleton

Although the straight skeleton was first introduced to computational geometry in 1995 by Aichholzer et al. [11, 10] their roots actually go back to the 19th century (see e.g. [177, p. 215-122]), motivated by the construction of building roofs. The straight skeleton will be explained in detail because it has an important relation with the skeletal representations presented in this thesis. The straight skeleton of a polygon relies on a shrinking process in which the edges of the polygon are moved inward parallel to themselves at a constant speed. Each vertex moves along the angular bisector of its incident edges. The shrinking continues as long as one event of the following types are found (see Figure 3.5):

1. *Edge event*: An edge collapses to length zero. If its neighboring edges still have non zero length, they become adjacent.
2. *Split event*: An edge is split by a reflex vertex. The edges adjacent to the reflex vertex are now adjacent to the two parts of the split edge.

The straight skeleton of a polygon is only composed of straight line segments, while the medial axis of a polygon may involve parabolic curves. Thus, the straight skeleton effectively characterizes the shape of a polygon and preserves its linear structure. In the case of orthogonal polygons, it is known that the straight skeleton of a polygon in general position is equivalent to the L_∞ Voronoi diagram [175]. Vyatkina [218] also proved that the straight skeleton contains a set of pruned medial axis for some of its parts. Aurenhammer [28] introduced the concept of weighted straight skeleton of a polygon, that allows the edges to move inward at different speeds.

The straight skeleton can be extended to polyhedra and it is defined by the shrinking of the polyhedron faces [81]. Assume that the polyhedron is in general position and all of

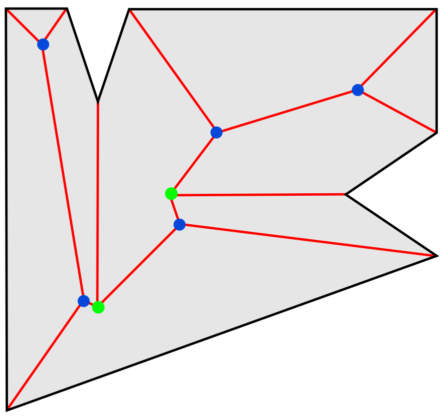


Figure 3.5: Straight skeleton shown in red of a simple polygon. Split events correspond to the green vertices of the straight skeleton and edge events to the blue ones.

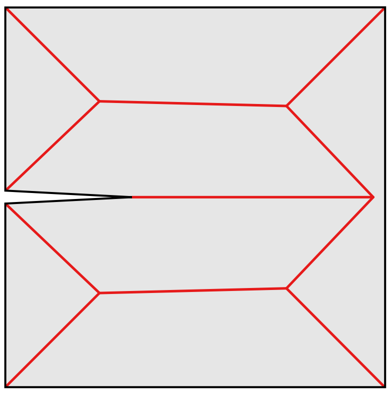
its vertices are trihedral. Then, the three-dimensional straight skeleton is defined by the next events found during the shrinking [33]:

1. *Edge event*: The meeting point of the four faces around an edge that vanishes.
2. *Split event*: A vertex, with one or two incident reflex edges, that runs into an opposite edge.
3. *Hole event*: A vertex, with all its three incident edges being reflex, that runs into a face.
4. *Edge-Split event*: Two reflex edges that cross each between them.

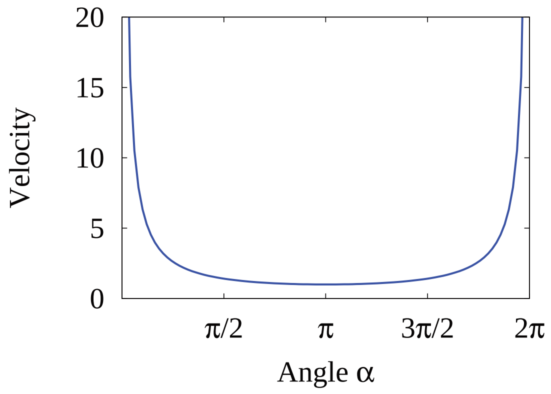
In three dimensions, the straight skeleton is only composed by vertices, edges and planar faces. The combinatorial complexity of the straight skeleton of a simple polyhedron is $O\left(n^2\alpha^2\left(n\right)\right)$ in the worst-case, where $\alpha\left(n\right)$ is the inverse of the Ackermann function and n is the number of vertices [33]. In the case of orthogonal polyhedra, the complexity is reduced to $O\left(n^2\right)$ [33]. Thus, the straight skeleton has a combinatorial complexity tighter than the medial axis.

Each vertex of a polygon, that moves inward during the construction of the straight skeleton, has a certain *velocity*. The *velocity* of a polygon vertex only depends on the dihedral angle between its two incident edges. Let α be the angle of a vertex. Consider that its incident edges are shrunk by a unit. Then, $\sin\left(\frac{\alpha}{2}\right)^{-1}$ gives the distance from the origin vertex and the shrunk vertex.

As shown in Figure 3.6 the velocity of a vertex dramatically increases for reflex angles that approach 2π and 0 radians. A major drawback of straight skeletons arises from the non-locality effects of reflex vertices: a reflex vertex may collide with a far element. It is expected that the same problem happens in three dimensions. This non-locality makes it difficult to use straight skeletons in applications where locality defined by a distance function is needed. Aichcholzer et al. [11] observed that if a polygon contains no acute angles, the resulting skeleton closely resembles its Voronoi diagram.



(a)



(b)

Figure 3.6: (a) Non-locality effects in the straight skeleton induced by reflex vertices of a simple polygon. The reflex vertex moves much faster than the other vertices. **(b)** Plot of the velocity of a vertex according to its angle. Note that the velocity tends to increase rapidly for higher and lower angle values.

3.1.6 Other skeletal representations

There exist alternative skeletal representations apart from the medial axis and the straight skeleton. Some of which are analyzed below.

The linear axis

The linear axis [215] is a skeletal representation derived from the straight skeleton that approximates the medial axis of a polygon while trying to reduce the aforementioned non-locality effects of reflex vertices. The initial set of shrinking edges is an altered version of the original polygon: zero-length edges are added at reflex vertices. A special class of equivalence between the linear axis and the medial axis is introduced. An algorithm to add the appropriate number of zero length edges is proposed. Vyatkina [219] generalized the linear axis to the case of planar straight line graphs and showed that in some cases the medial axis equivalence may not exist. The computation of the linear axis requires the previous computations of the medial axis. Thus, it seems more reasonable to directly approximate the parabolic curves of the medial axis with straight line segments rather than computing the linear axis.

Reeb graph and level set

Shinagawa et al. [197, 196] introduced a diagram encoding the topology of a shape. They called this diagram the reeb graph, in reference to a classical topological graph associated with a Morse function defined on a smooth surface [166]. The vertices of the reeb graph are critical points of the Morse function: points where the gradient of the Morse function are zero. The vertices of the reeb graph can be thought of as representing different components of the shape and the edges can be regarded as connections between these different components. The reeb graph is an accurate representation of the shape topology.

However, the geometric features of the shape, important in many skeleton applications, may not be captured

The reeb graph is not defined in the same space as the original shape. Level set diagrams [144] try to embed into the three-dimensional space each edge of the reeb graph using level sets of a real-valued function defined over the vertices of a shape (see Figure 3.7). The properties of reeb graphs strongly depend on the election of the function defined over the shape and the induced gradient. Biasotti et al. [41] presented a detailed overview on possible choices of that function and an extended reeb graph which can handle degenerate critical points [39].

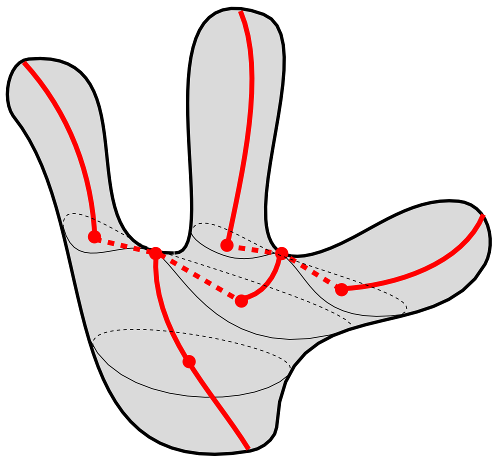


Figure 3.7: The level set diagram of a shape, defined by the centres of the connected components of the level sets (shown with thin lines) of a given function (courtesy of Lazarus and Verroust [144])

Curve skeleton

In a variety of applications, it is desirable that a three-dimensional shape skeletal representation is a linear or curvilinear one-dimensional structure. In the literature, these skeletal representations are commonly referred to as curve skeletons or center-line skeletons. However, there is not a unique way to define curve skeletons. Cornea et al. [73] presented a set of desired properties and algorithms to compute curve skeletons. While the curve skeleton captures the essential topology of a shape more compactly than the medial axis, it cannot be used to reconstruct the shape.

3.2 Skeleton algorithms

In this section, the most relevant properties to evaluate skeleton algorithms are presented. Thereafter an overview of the different types of algorithms to compute different skeletal representations is also provided.

3.2.1 Properties

Input shape

Generally, the input shape of a skeleton algorithm can be classified into the following categories:

- *Continuous representation*: the shape is represented as a collection of connected surface elements.
- *Discrete representation*: the shape is represented by a set of points sampled along its boundary or by a cell decomposition of the shape.

Robustness

The input of a geometric algorithm may be a set of geometric elements that are not in general position. A set of at least $n+1$ points in n-dimensional Euclidean space is said to be in general position if no hyperplane contains more than n points. A set containing k points for $k < d+1$ is in general linear position if and only if no $(k-2)$-dimensional flat contains all k points. If a set of points is not in general linear position, it is called a degenerate case. When it comes to implementing geometric algorithms degenerate cases are difficult to treat, in particular if there are several cases that have to be distinguished. As exact arithmetic computations are slow, algorithms using them are usually inefficient. The robustness of an algorithm is measured by its ability to handle degenerate cases.

Accuracy

An algorithm may compute either an exact or an approximate skeleton. An exact representation of the skeleton has to explicitly include all the skeleton parts. However, there are applications where an approximate representation is sufficient. The algorithm accuracy is strongly connected with some skeleton properties such as the stability under perturbations.

Efficiency

Algorithm efficiency is mainly related to the time complexity. In the case of algorithms that compute skeletons it is desirable to have a fast algorithm. As the size of real datasets grows, it becomes more important to develop efficient algorithms.

3.2.2 Methods to compute the medial axis

In the following, different methods to compute the medial axis are presented.

Exact methods

The medial axis can be computed exactly only for few types of shape representations. In practice, few algorithms compute exactly the medial axis because of its complexity. One class for which it is possible in principle are semi-algebraic sets [21], defined by a finite sequence of polynomial equations and inequalities. The medial axis of such sets is also semi-algebraic and it may be computed with tools from computer algebra, although no current implementations exist.

There exist approaches that try to compute the exact medial axis of free form shapes. In [90] the bisector between rational boundary curves is analyzed. However, few methods are able to compute the exact medial axis of free form shapes [9]. In fact, most approaches approximate the geometry of the shape using straight line segments and planar faces.

The medial axis of a union of balls in $\mathbb{R}^n$ can be computed from the Apollonius diagram of the balls or from convex hulls of finitely many balls in $\mathbb{R}^n$ [29, 48]. In addition, the medial axis of the union of balls is piecewise linear and can be constructed from the Voronoi diagram of a finite set of points [20].

In 1979 Kirkpatrick [136] presented a Voronoi based algorithm to compute the medial axis of polygons. The algorithm is based on the previous computation of a generalized Voronoi diagram. A much simpler algorithm based on a divide and conquer approach was introduced by Lee [145]. In this case, the medial axis is constructed by subdividing the polygon into small parts, recursively computing its Voronoi diagram and merging them. A linear time algorithm for computing the medial axis of convex polygons [3] and simple polygons [66] finally established the time complexity of medial axis computation of polygons. In the case of planar straight line graphs, the best algorithm runs in $O(n \log n)$ time [230]. Held [119] presented an efficient and robust implementation to compute the Voronoi diagram of points and line segments. In the case of polyhedra, Held [118] proposed a simple algorithm only suitable for convex polyhedra. Culver et al. [75] presented an exact approach that employs exact arithmetic and works with algebraic curves and surfaces (see Figure 3.8). However, the exact computation of the medial axis of polyhedra is usually not feasible for practical applications.

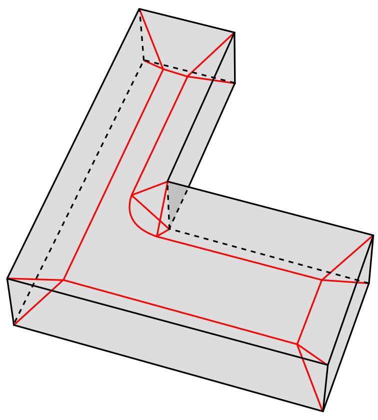


Figure 3.8: Medial axis of a simple polyhedron shown in red.

Approximated methods

The high instability of the medial axis has led to the development of approximate medial axis representations. A common way to approximate the medial axis is to consider the *approximation paradigm* described in [21]. First, the input shape is approximated with another class of shape for which the medial axis can be computed exactly and easier. Second, the medial axis of the approximated shape is computed. Finally, the unstable parts of the medial axis are pruned (see Figure 3.9). Most of the approximate methods require a pruning step which, in some cases, cannot effectively remove unstable parts of the medial axis (see Section 3.1.2).

$$\begin{tikzcd}
\mathcal{O} \arrow[r, "\text{Approximate shape}"] \arrow[d, dashed, "\text{Approximate } \mathcal{M}"'] & \overline{\mathcal{O}} \arrow[d, "\text{Compute } \mathcal{M}"] \\
\mathrm{P}\left(\mathcal{M}\left(\overline{\mathcal{O}}\right)\right) & \mathcal{M}\left(\overline{\mathcal{O}}\right) \arrow[l, "\text{Prune}"']
\end{tikzcd}$$

Figure 3.9: Approximation paradigm.

Voronoi diagram and medial axis

The medial axis can also be approximated directly from the Voronoi diagram [86] by computing a subset of the Voronoi diagram that converges to the medial axis as the sampling density increases. Brandt [51] showed that given a shape in $\mathbb{R}^2$, the Voronoi diagram defined by a sample of points of the boundary approximates its medial axis (see Figure 3.10). A sample of the boundary is a finite set of points on that boundary. In [14], the required sampling density of the boundary varies with the local feature size on the curve so that areas of less detail can be sampled less densely. Unfortunately, Amenta and Bern [13] showed that this property does not directly hold in $\mathbb{R}^3$. The problem is that some vertices of the Voronoi Diagram may lie too close to the shape boundary creating undesired artifacts. To handle this problem they introduced the concept of a pole. The poles of a sample point are the two farthest vertices of its Voronoi region, one on each side of the boundary. For a shape whose boundary is a smooth C^1-manifold the poles approximate the medial axis [15].

Attali and Montanvert [25] showed that a three-dimensional shape can be approximated by a finite union of balls. Amenta and Kolluri [16] extended this result and pointed that given a sample of points on the boundary shape, the union of a subset of the Voronoi balls approximates the original shape. Giesen et al. [108] proposed an algorithm that, given a sample from the surface of a three-dimensional shape with smooth boundary, computes a medial axis approximation that captures the topology of the shape. Balint et al. [164] presented an approach to compute an approximate the scale axis transform (see

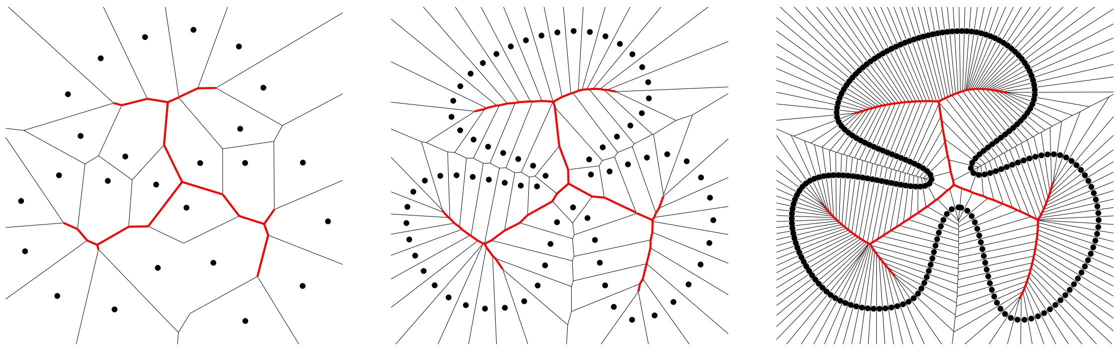


Figure 3.10: Progressive approximation of the medial axis using the Voronoi diagram of a sample of points (courtesy of Attali and Montanvert [24]). Medial axis approximation shown in red.

Section 3.1.2) of a three-dimensional shape by considering the union of balls approximating the shape. Yang et al. [229] also employed the union of balls in order to extract a simplified medial axis by sampling points on the surface of the balls and determining their closest features on the shape.

There has also been attempts to linearize and reduce the complexity of the medial axis of polygons by simplifying the underlying Voronoi diagram. These strategies try to obtain an approximated skeletal representation only composed by flat surfaces and straight segments. Canny and Donald [55] define a Voronoi diagram based on a measure of distance which is not a true metric. This leads to a formulation with lower combinatorial complexity of the Voronoi diagram. MacAllister et al. [163] proposed a linear representation of the Voronoi diagram with a space complexity linear in the number of polygons rather than in the number of edges. The medial axis can also be organized in a hierarchical structure. Ogniewick and Kübler [171] introduced a hierarchical organization of the medial axis branches of a polygon using the Voronoi diagram leading to a multiresolution representation of the skeleton.

Approximate methods for polyhedra

The approximate methods for polyhedra are discussed in more detail in a separate section because of its relevence in this thesis. Sherbrooke et al. [194] used a tracing classification scheme that computes an approximated medial axis of a polyhedron. Seam curves are represented with a piecewise-linear approximation. The vertices of the medial axis are connected by tracing along adjacent edges, and finally the faces of the medial axis are found by traversing closed loops of vertices and edges. Milenkovic [165] approximated the Voronoi diagram of polyhedra by computing all the points equidistant to four features (vertex, edge or face).

Sud et al. [206, 205] proposed a homotopy preserving algorithm to compute a simplified medial axis that removes unstable features of the original medial axis and retains its topological structure. They first compute an approximated medial axis and then prune by considering the separation angle formed by connecting a point on the medial axis to the closest points on the shape boundary.

The use of the dual representation of the Voronoi diagram (the Delaunay triangulation) to obtain the skeleton was also explored [193, 181, 23]. Etzion and Rappoport [96] constructed the Voronoi diagram of a polyhedron by separating the computation of the symbolic and geometric parts of the diagram. The symbolic part of the diagram, the Voronoi graph, is computed by a space subdivision algorithm and the geometric part is constructed by approximation. A divide and conquer approach to compute the medial axis was described in [204].

Aichholzer et al. [8] proposed an approach to compute an approximated medial axis of triangulated solids, with respect to a piecewise linear metric instead of the Euclidean one, that induces a piecewise linear skeletal representation. Martin et al. [157] introduced a skeletal representation called generalized swept mid-structure. The input shape is decomposed into a set of slices and the two-dimensional skeletons of adjacent slices are matched in order to obtain the swept mid-structure. Hisada et al. [122] presented a Voronoi based skeleton that tries to capture features of the medial axis that are more relevant to the human perception.

Other approximate methods

Hoffman [125] presented an algorithm to compute the medial axis of CSG. For each pair of boundary elements the set of points equidistant to both elements and with minimum distance is computed and sorted according to its distance to the boundary. Finally, the skeleton is constructed by tracing the arising edges and faces.

There also exists approaches to compute and approximate the medial axis of three-dimensional point clouds. Goswami et al. [112] introduced a Voronoi based algorithm that identifies flat and tubular regions of a three-dimensional point cloud. Ma et al. [154] introduced an algorithm to approximate the medial axis given a point cloud and the normal vectors to the surface at every point.

Musuvathy et al. [170] presented an approach to compute the medial axis of three-dimensional shapes bounded by C^4-smooth parametric B-spline surfaces by determining the self-intersection of the boundary moving along the inward normal direction. Cao et al. [56] presented a method to compute the medial axis of planar shapes based on the saddle point programming approach. Ba et al [31] extended the saddle point approach to the three-dimensional case.

Pruning methods

Pruning methods remove peripheral branches of the medial axis by trying to capture its stable part [189]. Pruning can either be performed implicitly as a post processing step or be implicitly integrated in the skeleton computation. Commonly, branches on the border of the medial axis are removed until some stopping condition is satisfied. The condition usually estimates the stability of portions of the medial axis [22, 24] or is defined by the difference between the shape and the shape reconstructed from the pruned medial axis [52]. Ward and Ghassan [223] presented the groupwise medial axis transform: the information derived from a similar group of shapes determines the significance of every medial axis branch that may be pruned. Ming-Ching and Kimia [58] proposed an approach to regularize the medial axis of three-dimensional shapes, so that similar shapes yield similar medial axis after the regularization.

Discrete methods

Discrete methods are applied to discrete representations such as images and volumes. Discrete methods are very robust since they work with integer arithmetic and geometric degeneracies are easier to treat. One class of discrete methods are *thinning algorithms* [79], that propagate the boundary of a discrete shape until a thin skeletal representation is found. In [143] a survey of thinning algorithms in $\mathbb{R}^2$ is presented. Thinning algorithms operate in discrete space and commonly rely on the concept of simple point [168]. A simple point of a discrete shape can be removed without changing the shape topology and can be characterized by only checking its local neighborhood. This property makes thinning algorithms well-suited for parallelization [231, 110, 153]. Recently, new characterizations of simple points which hold up to four dimensions have been presented [74]. Thinning algorithms may not be able to obtain a thin skeleton since there may exist simple points which cannot be eroded unless the shape topology is altered (see Figure 3.11). In addition, the time complexity of discrete algorithms is bounded by the number of pixels or voxels, which grows rapidly for large datasets.

Directional thinning algorithms alternate the direction to remove simple points [17]. However, the resulting skeleton is sensitive to the remove order and it is not unique. Subfield sequential thinning algorithms divide the discrete space into several subsets named subfields and at each iteration, only voxels belonging to one of the subfields are considered for deletion [37]. Thinning algorithms can benefit from spatial decompositions such as quadtrees [183] and octrees [226]. The computation of the approximate discrete medial axis can be greatly accelerated by computing the approximate Voronoi diagram by using a GPU [123].

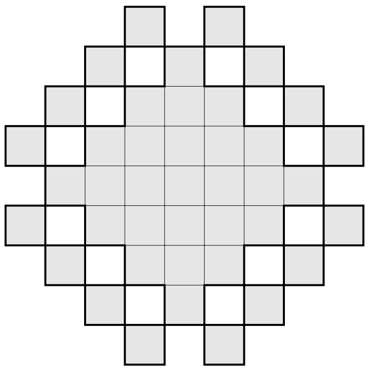


Figure 3.11: A two-dimensional discrete shape that cannot be reduced to a thin skeleton by using a thinning algorithm, since there are not any simple point to remove.

Ayala et al. [30] presented a thinning algorithm to compute the skeleton of an encoded volume [4] by using Boolean operations. Sudhalkar et al. [208] introduced the box skeleton of discrete shapes that relies on the L_∞ metric. The box skeleton is the set of centers of all the maximal squares in 2D or cubes in 3D included in the shape. However, if only these maximal squares or cubes are considered the homotopy equivalence may be lost (see Figure 3.12a). The discrete space induces a dual graph that is iteratively pruned until a thin skeleton is found. The set of simplicial complexes induced by the discrete space and the graph is also considered. At every thinning step, a part of the graph is removed if the associated simplicial complex collapse preserves the homotopy equivalence (see Figure 3.12b). All possible configurations of the collapse of simplicial complexes are analyzed. Thus, it is proved by induction that the resulting skeleton is homotopically equivalent to the discrete shape.

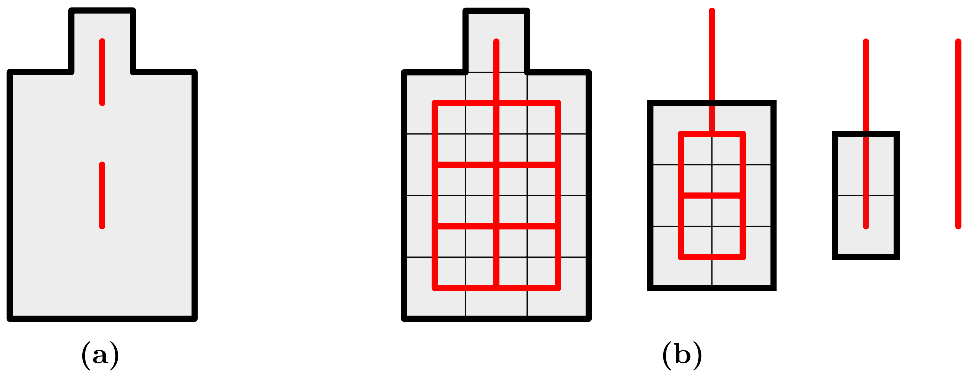


Figure 3.12: Illustration of the thinning algorithm to compute the box skeleton. **(a)** Set of centers, shown in red, of the maximal squares included into an orthogonal polygon. **(b)** The iterative pruning of the skeleton graph induced by the 2D discrete shape preserves the homotopy equivalence.

The generalized potential field model [7] is derived from the MAT and it is applied to discretized shapes. The shape boundary is assumed to be charged and the valleys of the resulting potential field are used to estimate the axes for the medial axis transform. The potential valleys are found by following a force field.

Ju et al. [131] introduced a thinning algorithm, combined with a pruning step that is able to create a family of simplified skeletons. Liu et al. [151] presented a thinning algorithm that operates on shapes represented as cell complexes and introduced a significance measure of the skeleton parts called medial persistence. Postolski et al. [180] proposed a filtered medial axis of two-dimensional discrete shapes inspired by the scale axis simplification scheme.

Another class of discrete methods rely on the *distance transform* or distance field of a shape, that defines for each interior point the minimum distance to the boundary under a given metric. Common metrics are the Euclidean distance L_2, the Manhattan distance L_1 and the Chessboard distance L_∞ (see Figure 3.13). A Euclidean distance transform was first proposed by Danielsson [77]. See [97] and [129] for a survey about two and three-dimensional distance fields, respectively. Lee and Horng [146] proved that the L_∞ distance transform and the L_1 distance transform of a discretized image are interchangeable.

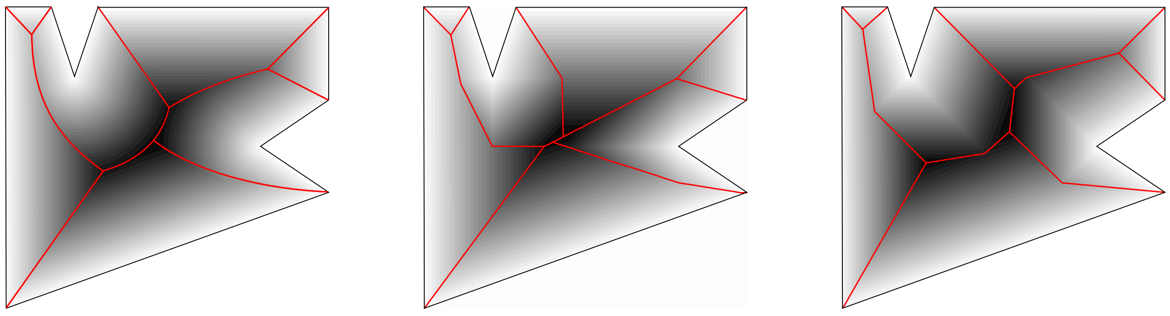

(a) Euclidean metric (b) L_1 metric (c) L_∞ metric

Figure 3.13: Distance transform of the interior of a simple polygon according to different metrics. The darker regions are the farther regions from the shape boundary. Medial axis, under each metric, shown in red.

The *ridges* of the distance transform are points that are locally centered according to a metric. By detecting the ridges of the distance transform a skeleton can be extracted [135]. The set of candidate ridges are pruned and connected to obtain a skeletal representation. The main advantage of distance transform algorithms is that computing the distance transform is fast, although, the ridge extraction slows the overall process.

Some algorithms use an approximate distance transform in order to reduce the computation complexity [232] or use local features to guide the skeleton extraction [155]. In [148], a Euclidean distance mapping combined with an active contour model is used for a minimization process. An approximate medial axis computation based on the distance transform [99, 202] is also possible. Hesselink et al [120] presented the integer medial axis, that uses a modification of an algorithm for Euclidean transforms. Solís and Lang [200] compute the skeleton of a 2D discrete shape by approximating it with a contour and computing its integer medial axis. Chaussard et al. [59] proposed a discrete version of the λ-medial axis (see Section 3.1.2) that relies on the integer medial axis.

General field functions [7] rely on a general transform guided by a repulsive force. The skeleton computation is done by detecting the local extremes of the field and connecting them [69]. The advantage of general field functions over the distance transform is that they are less sensitive to noise on the boundary of a shape although its computation is less efficient.

3.2.3 Methods to compute the straight skeleton

Aichholzer et al. [11] first proposed a trivial method to compute the straight skeleton of polygons that considers each pair of edges of the polygon, in order to detect the next possible events. An algorithm of the same authors for planar straight graphs [10] maintains a triangulation of the part of the plane that has not been reached yet by the propagating set of edges. Felkel and Obdrzalek [98] provided details on the implementation of a straight skeleton algorithm based on [11].

In general, the time complexity to compute straight skeletons of polygons and polyhedra is greater than the medial axis. This is because the medial axis is defined with a distance function, that is a local property, while the straight skeleton is defined by the sequence of collision events during the shrinking process, that is a global property. For example, it is possible to apply a divide and conquer scheme to compute the medial axis, while it is not possible in general for straight skeletons.

Eppstein and Erickson [92] simulate the sequence of collisions between edges and vertices during the straight skeleton shrinking process using a complex technique for maintaining extrema of binary functions [91]. In [64] a connection between the straight skeleton and the motorcycle graph is presented. Consider a set of points in the plane, called *motorcycles*, that drive along straight line rays according to its speed and start time, and a set of straight line segments, called *walls*. Every motorcycle leaves a trace behind it and stops driving when reaching the trace of another motorcycle or a wall. The arrangement of these traces is called motorcycle graph (see Figure 3.14). Huber and Held [128] presented an approach that also relies on the previous computation of the motorcycle graph and that is focused for a practical use. Das et al. [78] introduced a deterministic algorithm for monotone polygons. Palfrader et al. [174] proposed a fast algorithm based on a kinetic triangulation data structure.

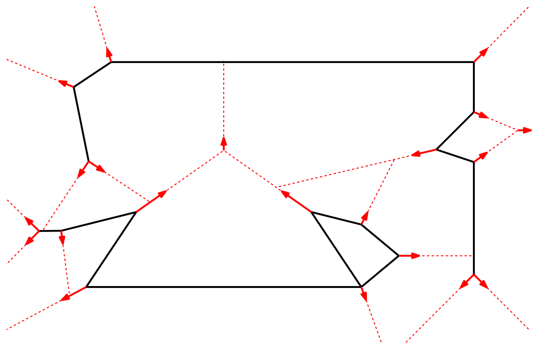


Figure 3.14: The motorcycle graph, in dashed line, is defined by the motorcycles and walls. Every arrow depicts the speed vector of a motorcycle. Walls are shown bold (courtesy of Huber and Held [128])

The first algorithm found in the literature that addresses the construction of straight skeletons of three-dimensional polyhedra was presented by Barequet et al. [33]. In the case of orthogonal polyhedra, the planar straight skeleton of every polyhedron face is computed. Thus, the collision between each pair of features of the polyhedron or of the straight skeleton of each polyhedron face is computed. There are in the worst-case n^2 collisions, where n is the number of vertices of the polyhedron, and the validity of a collision can be checked in logarithmic time. This leads to an algorithm with $O\left(n^2 \log n\right)$ time complexity. A more complex output sensitive algorithm that uses orthogonal range searching techniques is proposed. Both algorithms are not implemented. An advantage of the orthogonal algorithm is that its complexity is bound by a quadratic factor with respect to the number of vertices of the polyhedron. In the case of simple polyhedra, all the initial collisions between pairs of features are computed. Then, every time a valid collision is processed new future collisions are found on neighboring features around the collision. The complexity of the output-sensitive algorithm is $O\left(kn\right)$, where k is the number of vertices of the straight skeleton.

3.2.4 Methods to compute the curve skeleton

Most of the algorithms to compute the curve skeleton are designed for the discrete case. Zhou and Toga [233] presented a technique to extract the curve skeleton by considering a three-dimensional distance transform. Bitter et al. [45] also considered the distance transform and implicitly mapped the voxels to graph vertices and the voxel neighbor relations to graph edges in order to apply the Dijkstra algorithm [88] as a substep of their skeleton algorithm. While most of the existing thinning algorithms compute the medial axis of a shape, some of them also apply to curve skeleton [228, 222]. Cornea et al. [72] presented a hierarchical curve skeleton, to define a whole family of skeletons, that relies on the computation of a repulsive force field over a discrete shape. A common technique is to obtain a curve skeleton by pruning a higher dimensional skeletal representation [18]. Some authors [191, 212, 126] also proposed different approaches to compute the curve skeleton of point clouds.

There also exists approaches to compute the curve skeleton by considering a continuous representation of the input shape. The curve skeleton can be formulated as a subset of the medial axis with the help of a medial geodesic function [84]. Au et al. [26] presented an algorithm that works directly on the shape boundary by contracting it into a curve skeleton. Tagliasacchi et al. [211] presented a curve skeleton that relies on the mean curvature flow [83]: the shape geometry is collapsed, driven by the curvature flow, until the curve skeleton is obtained. Sobiecki et al. [199] compared some contraction based methods to extract the curve skeleton and analyzed their limitations.

3.3 Skeleton applications

A high number of scientific applications employ skeletal representations. This can be verified according to the huge amount of available literature related to skeleton applications. In this section, some of the most relevant skeleton application fields are reviewed. A more extensive survey of skeleton applications has been presented by Leymarie et al. [149].

The first application of skeletal representations were in the areas of medicine and biology as Blum [47] conceived them as a natural descriptor for shapes such as cells and body tissues. Skeletons are used for segmentation of medical images and volumes [178] (see Figure 3.15), analysis of vascular structures [221], path planning in surgical navigation [101], morphometry and recognition of body tissues. Skeletons are also useful to analyse molecular and atomic structures.

The problem of shape recognition relates to the process of searching a database of shapes in order to efficiently retrieve instances that are similar to a particular shape. Skeletal representations provide a particularly attractive choice because they encode the shape topology and geometry [121] (see Figure 3.16).

Skeletons also have been extensively used to analyze the motion and animate articulated figures such as humans. The most common skeletal representation used for this

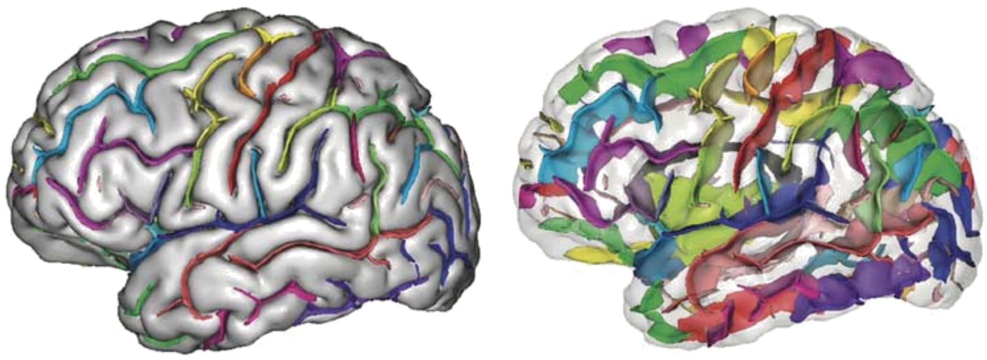


Figure 3.15: The cortical folds of a brain are extracted by computing a three-dimensional medial axis using a thinning procedure (courtesy of Mangin et al. [156])

purpose is the curve skeleton. In motion planning, the skeletons are used as a constraint to find optimal paths leading from a goal to a target locus [220], as an example to devise a routing scheme for networks [54]. In addition, it has been used for surface meshing and FEM [203], solid representation [190], shape morphing, reconstruction [13], simplification [213], deformation [57], decomposition [133], and generation [184]. The skeleton is also useful tool in collision detection [50], self-intersection detection, and visibility problems. Skeletal representations are also relevant in the fields of engineering analysis and CAD [85], and applications in fluid simulation [2].

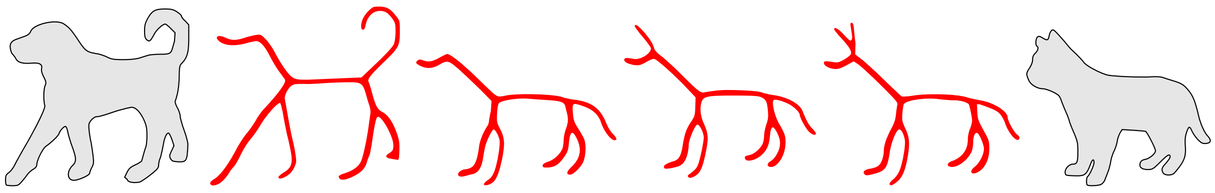


Figure 3.16: Shape matching of two shapes by using their skeletons (courtesy of Sebastian et al. [188]).

Perception studies confirm that the human visual system relies on skeletal structures to understand and identify shapes. A computational scheme proposed by Kimia [134] supports the speculation that a skeleton representation of a shape is generated early in visual processing. Lescroart and Biederman [147] studied how the visual system represent shapes by specifying the relations between the medial axes of its parts: a magnetic resonance study verified that the medial axis structure is represented earlier in the human visual system. Thus, skeletons are relevant shape representations in fields where human perception must be considered.

The ability of straight skeletons to define the roofs [6] (see Figure 3.17) of a building has been used in procedural modeling of cities [169]. It has been also used in computational origami [82], interpolation between polygons [34] and crowd simulation [116]. Barequet et al. [35] computed the straight skeleton of polyhedra as a step to reconstruct a subdivision of the three-dimensional space, given by arbitrarily oriented slices, into labeled domains.

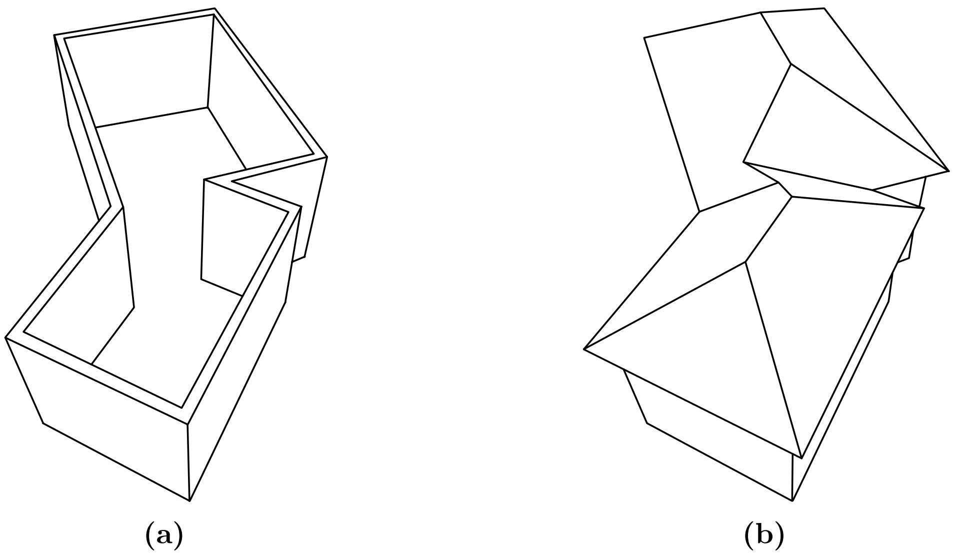


Figure 3.17: Building roof (b) extracted from the straight skeleton of the roof base (a).

3.4 Conclusions

In this chapter, the state of the art in skeletal representations has been reviewed. It can be summarized in the following main observations:

- The medial axis is unstable and difficult to compute exactly. Most of the existing approaches to approximate the medial axis involve approximating the shape or the Euclidean metric and removing unstable parts of the skeleton.
- The straight skeleton of polygons and polyhedra has tighter combinatorial complexity upper bounds compared with the medial axis. Moreover, it is only composed of linear segments and planar faces. In the literature, the straight skeleton of polyhedra has been little studied and skeletal representations derived or having a certain connection with the straight skeleton are rare.
- Reeb graphs and curve skeletons, accurately capture the topological information of a shape but may fail to encode some of its geometric features.
- Most of the approaches to compute skeletons operate in the discrete space. This is mainly due to these reasons: discrete methods avoid geometric degenerate cases, use integer arithmetic, and their algorithmic complexity is, in general, lower than geometric methods.
- Although the constrained structure or orthogonal shapes, few skeletal representations specifically designed for them are found in the literature.

Chapter 4

Preprocess: B-Rep extraction of orthogonal shapes

Contents

4.1 Introduction

Extracting the boundary of two and three-dimensional orthogonal shapes is a fundamental operation in fields such as image and volume processing or CAD applications. Moreover, the algorithms presented in Chapter 6 and 7 require as input a B-Rep model describing an orthogonal polygon or polyhedron, respectively. Unfortunately, the boundary extraction techniques reviewed in the literature were unable to appropriately treat pseudomanifold shapes. In this chapter, two sweep line algorithms to compute a boundary representation

(B-Rep) of orthogonal polygons and polyhedra are presented. This chapter is an adapted version of [217].

The input of the presented methods is the set of vertices of the orthogonal shape and, in the case of orthogonal polyhedra, some neighborhood information. The output of the algorithms is a B-Rep model composed by the set of vertices, the set of edge loops, the set of faces and the relations *face:loop* and *loop:vertices*. The relation *face:loop* distinguishes the external loop of the face from the rest of the loops, the holes of the face. The relation *loop:vertices* is stored as a connected list of vertices such that the normal of the face points toward the exterior of the orthogonal shape. This implies that while the external loop of a face is oriented clockwise, hole loops are oriented counterclockwise. Note that edges are implicit and can easily be recovered. Besides, it is also straightforward to obtain all other topological relations from the given ones. Although the presented method can deal with orthogonal polyhedra with any number of shells with cavities, the problem of classifying three-dimensional shells and its cavities is not addressed.

In addition, methods to obtain the input vertices of the presented methods from several other models are presented. In the following, the literature of boundary extraction techniques is reviewed.

4.2 Related work

In this section, a *binary image* is either a two-dimensional binary image or a three-dimensional regular grid composed of voxels.

Boundary extraction of binary images is approached by two main methods: *polygonal* and *digital*. Polygonal or bevelled-form methods represent the boundary as a set of edges (2D) or triangles (3D) [70]. Digital or block-form methods represent the boundary as a set of axis-aligned edges (2D) or faces (3D) that can be voxel size [182] or not restricted to any size [5].

In general, digital models exhibit formal properties such as closure, orientedness and connectedness, whereas polygonal models and related techniques are mainly devoted to visualization purposes [114]. Polygonal models tend to produce poorly shaped and degenerate polygons that can be partially corrected with smoothing [152] or by modifying the regular grid [87].

A common drawback of the existing boundary extraction techniques is that the resulting boundary elements consist of several little edges (2D) or little quadrangular or triangular faces (3D). Several adaptive attempts have been developed to reduce this redundancy. A k-d tree can be used to extract a manifold quadrilateral mesh [113] while the dual contouring method uses an octree and produces a manifold mesh [186].

Alternatively, digital methods interpret the boundary of a binary image as orthogonal polygons or polyhedra that can be manifold or pseudomanifold. An adjacency pair (m, n) is associated to a binary image, meaning that foreground voxels are m-adjacent and background voxels are n-adjacent [141]. Using the same adjacency relations for the foreground and background voxels means that paradoxes can be avoided by restricting these pairs to $(4, 8)$ and $(8, 4)$ for 2D, and $(6, 26)$ and $(26, 6)$ for 3D [141]. However, images with this adjacency pair present pseudomanifold configurations.

The constrained structure of orthogonal polygons and polyhedra has been attractive in several research fields. Orthogonal shapes are directly studied to solve several problems, but they are also used as bounding or approximated shapes of arbitrary polygons and polyhedra. Examples of the first case are found in manufacturing applications as unfolding the boundary of a polyhedron [173], mesh reconstruction from scanner data [42], two and three-dimensional visibility problems [103, 68] and urban models [209]. Moreover, orthogonal shapes are used as bounding structures for spatial databases [94], and as approximate shapes in motion planning [12], grasping in manufacturing processes [65] and for reachable states of dynamical systems [19].

In the following, some papers that deal with the extraction of orthogonal boundaries and seek for suitable representations for orthogonal objects are reviewed in more detail. Some approaches are based on spatial enumeration models, while others use vertex lists to represent orthogonal shapes.

Regarding spatial enumeration models, Montani and Scopigno [167] proposed algorithms to convert quadtrees to polygons and octrees to polyhedra. The authors do not mention how to treat non-manifolds, neither in two nor in three dimensions, and they avoid to compute inclusion relationships as they only consider the case with only one loop of edges bounding a face. Karunakaran and Shringi [132] also developed a program that converts an octree to a B-Rep. The main drawback of their proposal is that it is very specific for numerical control (NC) machines, and as a consequence it is difficult to generalize.

Heijmans [117] computed the zonal graph for two-dimensional images and used it to perform image processing. The zonal graph is a graph representing the inclusion relationships between loops. Every vertex of the graph represents a loop and it is specified whether it belongs to the foreground (external loop) or to the background (hole loop). Park and Choi [176] presented a different method to extract the boundary of a two-dimensional image, with all its loops and inclusion relationships, in order to extract cutting areas from a sculptured surface in the NC field. The sculptured surface is first discretized and cutting areas are obtained. They use a run-length codification of the 2D image and devise an algorithm which is $O(nr)$, nr being the number of runs. The boundary consists of axis-aligned polygons with maximal horizontal edges and vertical edges of pixel size, as a consequence of the run length encoding being used. This method does not deal with pseudomanifold images. Concerning 3D images, Damiand [76] presented a topological map describing the boundary between voxels and an incremental algorithm to extract the boundary. The model is a combinatorial map, and the method obtains first the B-Rep of every voxel and then, it applies removal operators for faces, edges and vertices.

The problem of inclusion relationship has also been studied for three-dimensional shells. Gargantini et al. [102] presented a method that computes inclusion relationships between shells and represents them in a tree structure.

Vertex list representations are a compact way to represent orthogonal shapes that can perform geometric transformations, general interrogations and even Boolean operations. It has been shown that a list of all vertices of an orthogonal shape without any additional information is ambiguous, even in two dimensions [49]. Figure 4.1a shows two different

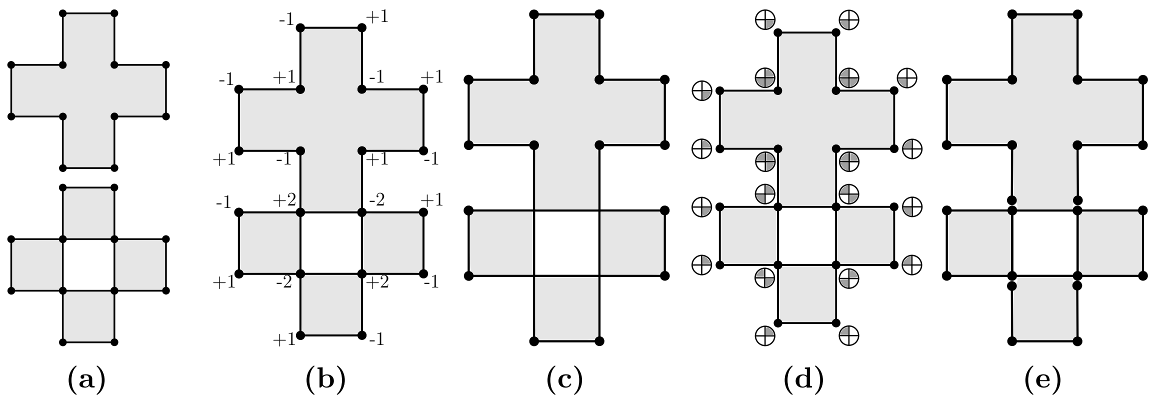


Figure 4.1: **(a)** Two different orthogonal polygons that have the same vertices. **(b)-(d)** A pseudomanifold shape represented using the weighted vertex set, the EVM and the neighborhood models, respectively. **(e)** For the same shape, the set of vertices that the presented approach needs as input.

orthogonal polygons with the same vertices. However, vertex lists, with some additional information associated to the vertices, constitute an implicit and complete representation model. Vertex lists represent the same kind of shapes that can be represented by spatial enumeration models with the advantage that they require less storage and are not sensitive to translation [95]. With respect to run length encoding, the number of runs depends on the precision of the underlying pixel or voxel model as well as on the run-length direction chosen. Although run-length encoding is a compressed model, it represents the interior of the shape while vertex lists represent the boundary.

In two dimensions, the problem of reconstructing an orthogonal polygon from its vertices has been solved by O'Rourke [172]. The authors presented an $O(n \log n)$ algorithm based on sorting points on any axis-aligned line and then adding an edge between every other pair of points. This proposal is only valid for orthogonal manifold polygons and does not recover the hole-face containment relationship.

Esperança and Samet presented and formalized a vertex list based model [93, 95], suitable for polygons and polyhedra as well as for multi-valued regular grids. To avoid confusion with other vertex lists models, this model is referred to the *weighted vertex list* representation. In this model, orthogonal polyhedra are represented by a subset of lexicographically sorted vertices with an associated weight. The model as, well as the processes applied to it, is recursive in the dimension. For example, vertex list representations in n-dimensional space can be represented and processed by vertex list representations in $n-1$ dimensional space using the project and unproject operators. These authors presented methods to apply geometric transformations, set operations and display solids represented as vertex lists. However, they did not describe any conversion algorithm from vertex lists to a more explicit B-Rep model.

Two other models use lexicographically sorted vertex lists and are recursive in the dimension: the *neighborhood representation* and the *Extreme Vertices Model* (EVM). The neighborhood representation [49] is a model for orthogonal polyhedra used as an approximate model for reachable states in dynamical systems. The authors of this work present algorithms for point membership and Boolean operations. They also describe a method to detect faces of the polyhedron. The method has linear time complexity on the number of vertices. However, it only finds the set of vertices associated to an axis-aligned plane and it does not obtain an explicit B-Rep model with hole-face inclusion relationships.

The EVM [4, 5] is another vertex list model that stores a subset of vertices, called *extreme vertices*, without any associated information, and represents indistinctly manifolds and pseudomanifolds. General set membership including Boolean operations have been developed with this model as well as an algorithm [5] that extracts from the EVM an explicit B-Rep. The method obtains all vertex coordinates and oriented faces applying a sweep-based process. However, the inclusion relationship between loops is computed with a quadratic complexity brute-force method. Moreover, the method presents a flaw when dealing with vertices with four incident faces, since some kind of non-manifold vertices are not properly managed [217].

There exists another model that uses a vertex list [43], but it is restricted to *orthogonally convex polyhedra* which are the subset of orthogonal polyhedra that satisfies the following condition: any axis-aligned line intersects the polyhedron in at most one line segment. This restricted class of polyhedra can be represented with all its vertices and without any associated information. The authors present a $O(n \log n)$ algorithm to extract faces from the vertex list, but they do not study the hole-face inclusion problem.

4.3 B-Rep extraction of orthogonal polygons

A sweep line algorithm (see Section 2) to compute the B-Rep of an orthogonal polygon is presented. The events are the vertical polygon edges encountered by the sweep line. The status data structure maintains the intersection between the sweep line and the polygon. The input of the algorithm is the set of vertices of an orthogonal manifold polygon. The output is its B-Rep model including the *hole:face* relation information: the set of holes corresponding to each polygon face.

4.3.1 Algorithm design

The two-dimensional algorithm is based on the following lemma, which is the same observed by O'Rourke [172] but stated in terms of a sorted list:

Lemma 1. *Let L_{xy} be the lexicographically xy-sorted list of vertices of an orthogonal manifold polygon. Then, the pairs of consecutive vertices sorted in L_{xy} are the vertical polygon edges. Conversely, the pairs of consecutive vertices of the lexicographically yx-sorted list of vertices L_{yx} are the horizontal polygon edges.*

Note that given a vertex v of a vertical edge, it is possible to find the other vertex of the edge searching for v in L_{xy} and taking either the next element or the previous element, depending on whether v is in an odd or even position in L_{xy}.

Therefore, a loop of edges can be formed by starting from one vertical edge e, search for it in L_{xy}, take the endpoint of e that is in an even position, search for it in L_{yx}, take the corresponding endpoint, and so forth until e is found again. If during this *alternate search* the visited edges are marked, it is possible to form all the loops of edges by traversing the vertices of L_{xy} and forming a loop each time a non-marked vertex is visited. This approach is very similar to the O'Rourke algorithm used to reconstruct polygons [172]. The alternate search always forms loops of edges clockwise oriented, regardless of whether they are external or not and does not recover the information on whether the loops are external or holes.

Having all of this in mind, a sweep line algorithm to extract the B-Rep of an orthogonal manifold polygon has been designed. Let $\mathcal{L}$ be the vertical sweep line, sweeping from left to right. The events correspond to the vertical edges of the polygon and their priority is defined by the x coordinate. In addition, the intersection between $\mathcal{L}$ and the polygon is maintained by the status data structure $\mathcal{S}$ and used to recover the hole-face containment relationships during the sweep. More precisely, $\mathcal{S}$ is the intersection of $\mathcal{L}$ with the set of polygon faces. It is stored as a y-sorted set of vertical segments, each one labeled with the identifier of the face it belongs to. The orientation of the hole loops is changed once the sweep is finished. Figures 4.2 and 4.3 illustrate the sweeping process.

Property 3. *At any moment of the sweeping process, the edge loops with some vertices located left to $\mathcal{L}$ have already been formed and classified, either as external or as hole loops of a polygon face.*

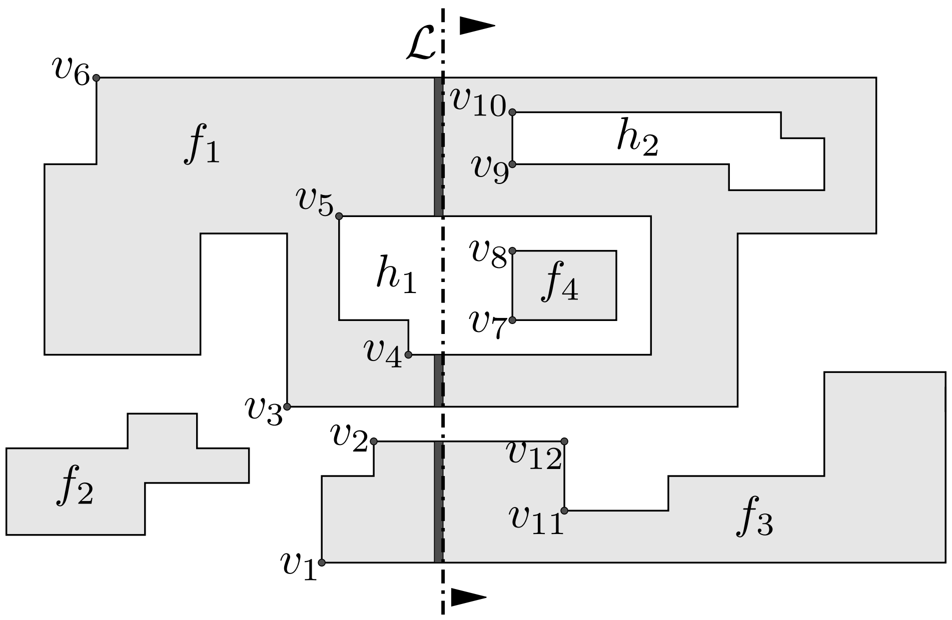


Figure 4.2: Scheme of the sweep line algorithm to compute the B-Rep of an orthogonal polygon. The status of $\mathcal{S}$ corresponds to the step j of Figure4.3

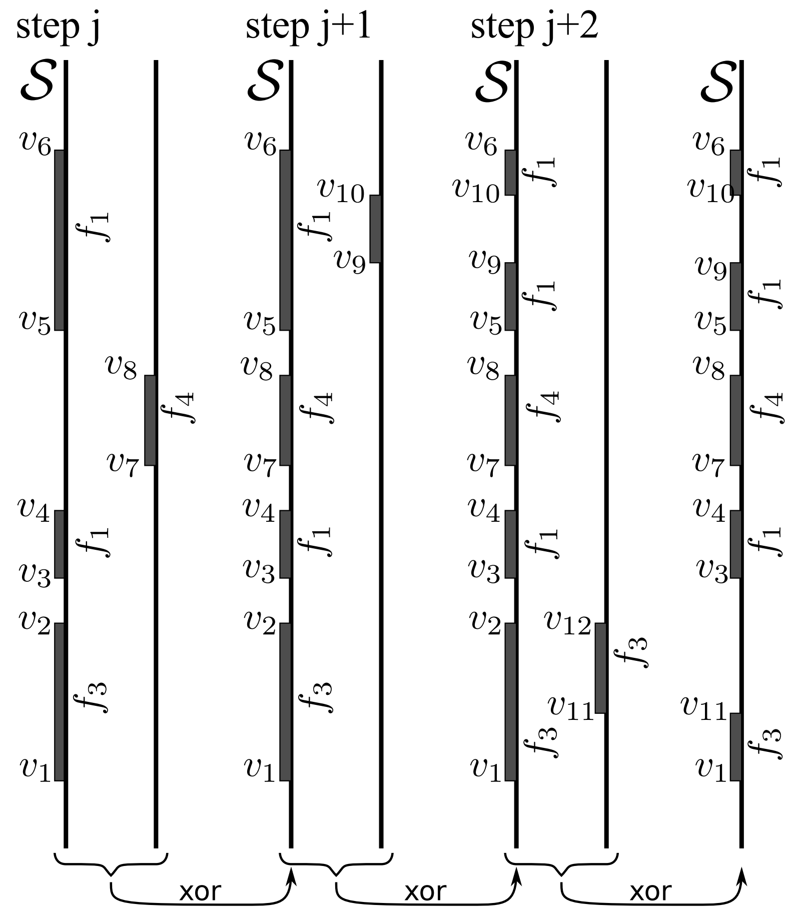


Figure 4.3: Three steps showing how the status data structure $\mathcal{S}$ of Figure 4.2 is updated using a XOR operation with the new segments.

In addition, the algorithm needs to know which face belongs to each visited vertical edge: the *edge:face* relationship. This information is stored as edge marks: edges are marked with the identifier of the face they belong to.

An event is processed in the following way: if the edge is already marked, $\mathcal{S}$ needs to be updated, since it is an edge of an already formed loop. Otherwise, if the edge is not marked, it must be the leftmost vertical edge of an unformed loop. $\mathcal{S}$ is updated and the edge is classified as belonging to the external loop of a new face or as a hole loop of an already formed face. Then, the loop is formed by performing the alternate search method, and finally all the edges of the recently formed loop are marked as related to the face.

Observe that the update of $\mathcal{S}$ with a new vertical edge e can be done as a XOR operation between $\mathcal{S}$ and e. This denotes searching for the y coordinate of the lower endpoint of e in $\mathcal{S}$ and then merging the segment endpoints using its y values. Note that at most three segments, the new one and two in $\mathcal{S}$, are implied in this merge.

A non-marked edge is classified as belonging to an external loop if it falls outside all segments in $\mathcal{S}$. Otherwise, the edge must be part of a hole loop and must fall strictly inside one of the segments of $\mathcal{S}$. In other words, both y values of the edge endpoints fall inside a segment of $\mathcal{S}$. In fact, the labels of segments in $\mathcal{S}$ are stored in order to know the face that a hole belongs to. See Algorithm 2 for an overview of the two-dimensional B-Rep extraction.

Algorithm 2 Two-dimensional B-Rep extraction algorithm.

```
$L_{xy} \leftarrow$ Lexicographically $xy$-sorted list of vertices
$L_{yx} \leftarrow$ Lexicographically $yx$-sorted list of vertices
$i \leftarrow 0$
while $i < |L_{xy}|$ do
    edge $\leftarrow \{L_{xy}[i], L_{xy}[i+1]\}$
    if edge not marked then
        Classify the loop as a face or a hole
        Form the loop by performing the alternate search on $L_{xy}$ and $L_{yz}$
    end if
    Update the intersection $\mathcal{S}$
    $i \leftarrow i + 2$
end while
```

As an example, Figure 4.2 shows an intermediate step j of the algorithm. At the current state, $\mathcal{S}$ is composed of three segments (shaded intervals in the figure): one for face f_3, and two for face f_1. The external loops f_1, f_2, f_3 and the hole loop h_1 that have some vertices on the left of $\mathcal{L}$ have already been formed and classified. The next three events are the vertical edges v_7v_8, v_9v_{10}, and $v_{11}v_{12}$. Figure 4.3 shows the update of $\mathcal{S}$ according to these three steps. Notice that each update requires a XOR operation between a set of sorted segments and a new segment. Processing edge v_7v_8 will activate the reconstruction of loop f_4, since it is a non-marked edge. This loop will be classified as the external loop of a face, since v_7v_8 falls outside segments in $\mathcal{S}$ and a new face f_4 is added to the set of faces. Then, $\mathcal{S}$ will be updated, adding a new face segment f_4 to it between the two segments of face f_1. When edge v_9v_{10} is processed, loop h_2 will be formed, and it will be classified as a hole loop of face f_1 since v_9v_{10} falls inside the upper segment of $\mathcal{S}$. $\mathcal{S}$ will be updated by splitting the last segment into two new ones belonging to f_1. When edge $v_{11}v_{12}$ is processed no loop will be formed, as it is a marked edge belonging to f_3, and only $\mathcal{S}$ will be updated.

4.3.2 Managing orthogonal pseudomanifold polygons

Orthogonal pseudomanifold polygons include non-manifold vertices with four incident edges. Dealing with these vertices implies that to properly form loops following the pseudomanifold definition: edge cycles have to separate the four edges properly. By taking into account that, later, when dealing with polyhedra, the output polygon faces will correspond to the faces of a polyhedron, the separation is done in such a way that the interior set of each face must be a single connected component (see Figure 4.4). Note that a face of an orthogonal manifold polyhedron can be a pseudomanifold polygon.

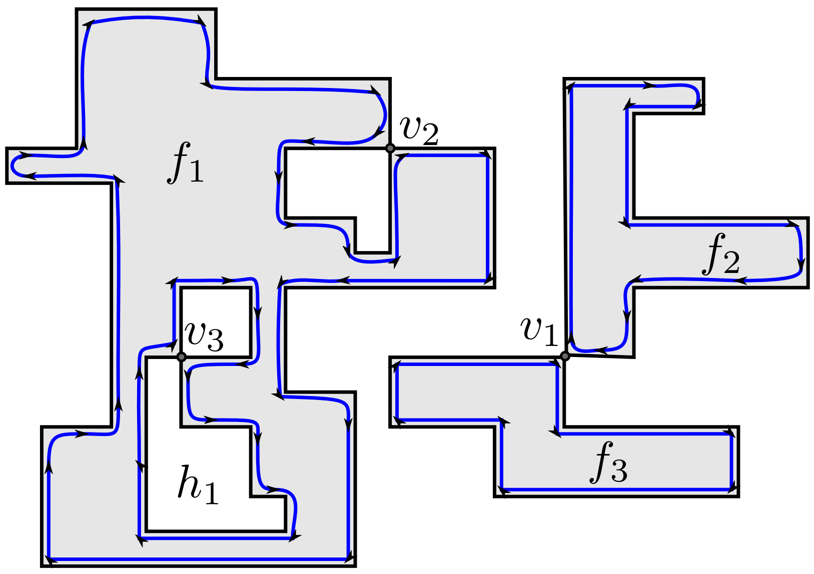


Figure 4.4: Edge loop reconstruction of an orthogonal pseudomanifold polygon. Three faces will be formed, one of them with a hole. The blue path shows the way loops are formed.

In general, a pseudomanifold polygon cannot be reconstructed only from the set of its vertices [49]. However, the method proposed in the previous section, may be modified to handle pseudomanifold polygons by slightly varying the input and some parts of the algorithm.

First of all, non-manifold vertices are inserted twice in the input list. As a consequence, when the lexicographically sorted lists are computed, Lemma 1 still holds true. Deciding whether a vertex is manifold or non-manifold must be done with the help of the original representation scheme. For example, if the original model is a binary image, all pixel vertices with a non-manifold configuration around them must be inserted twice in the input list.

Second, the procedure to form loops must be modified in order to handle non-manifold vertices. During the alternate search, when a duplicated vertex is found, only one of the two vertical or horizontal edges that emanate from it belong to a loop. Consider that all loops are formed clockwise. If the loop that is being formed is an external one, the edge that forms a convex vertex with the last edge belongs to the loop. Otherwise, two non-connected components could be merged (vertex v_1 in Figure 4.4) or form a loop that leaves a complete cycle to its right, that would be incorrectly identified as one of its holes (vertex v_2 in Figure 4.4). Conversely, when a hole loop is being formed, the opposite rule is considered: the edge that forms a concave vertex in the hole loop is chosen (vertex v_3 in Figure 4.4).

And thirdly, the update of $\mathcal{S}$ must also be slightly modified. Since non-manifold vertices have two vertical edges, an endpoint of a segment in $\mathcal{S}$ may coincide with the beginning point of the next segment, although the two segments may belong to different faces. As a consequence, the XOR operation will merge two segments only if they share an end and belong to the same face.

4.3.3 Algorithm complexity

Lemma 2. *The boundary representation of an orthogonal polygon can be computed by the sweep line algorithm in $O(n \log n)$ time and $O(n)$ space, where n is the number of vertices of the input orthogonal polygon.*

Proof. Lexicographical sorting of the set of input vertices requires $O(n \log n)$ time. Once sorted, looking for a vertex when forming the polygons can be done with a binary search, and since each vertex is searched only once in L_{xy} or L_{yx}, forming all polygons requires $O(n \log n)$ time. Segments in $\mathcal{S}$ are sorted by their vertical position, therefore it may be implemented as a balanced binary search tree. Updating $\mathcal{S}$ with a new segment requires first a search for one of the edge endpoints in $\mathcal{S}$, and then a modification where at most three vertical segments are implied, so, it requires a maximum of two insertions and/or two deletions in the binary tree. Search, insert and delete operations in a balanced binary search tree require $O(\log\ n)$ time. Overall, n updates in $\mathcal{S}$ require $O(n \log n)$ time. Therefore, the overall time complexity is $O(n \log n)$.

In addition, the algorithm requires few lists of vertices and a balanced binary search tree whose worst-case size is $O(n)$. The edge marks may be stored as inverse pointers inside L_{xy}. Thus, the space complexity is $O(n)$. □

4.4 B-Rep extraction of orthogonal polyhedra

4.4.1 Algorithm design

The boundary representation of orthogonal polyhedra is composed of the boundary representations describing the polyhedron faces parallel to the three coordinate planes. Following an approach similar to [167], three traversals in the three axis directions can be done. Each traversal visits all planes parallel to an axis-aligned plane that contain faces of the polyhedron, forms the lists of vertices of the polyhedron that lie on the plane, and calls the 2D algorithm that given this list extracts the faces lying onto the plane. Before dealing with the different traversals of the polyhedron, it has to be decided which vertices are duplicated.

Orthogonal manifold polyhedra have three kinds of vertices, depending on the number of faces incident to them: V3, V4 and V6 (see Figure 4.5). V3 vertices are inserted in the vertex list once. V6 vertices have to be inserted twice in each list, because each pair of incident coplanar faces of a V6 vertex correspond to a 2D pseudomanifold vertex.

V4 vertices are the most particular case. The faces incident to a V4 vertex parallel to one of the coordinate directions also have a pseudomanifold configuration. But in the other two coordinate directions, there is only one face incident to the vertex, and these two faces contain two consecutive collinear edges (see Figure 4.5). However, observe that if the V4 vertices are always inserted twice in the lists, regardless of the coordinate axis being traversed, the 2D algorithm forms edge loops even if they contain collinear edges.

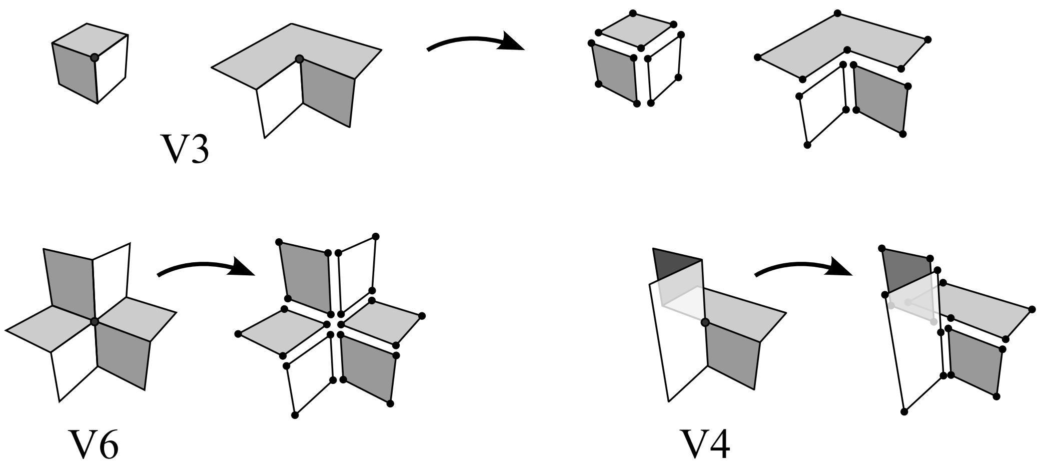


Figure 4.5: The three types of orthogonal manifold vertices and the faces they give place to.

Indeed, when the input list of vertices is sorted into L_{xy} and L_{yx}, two consecutive collinear edges will induce a zero length edge in one of these lists. As a post-process, the duplicate vertices are removed from the edge loops in a row, so that zero length edges are deleted.

Instead of performing these three traversals, a more simple approach which only needs three calls to the 2D algorithm is possible. Note that if the lexicographically zxy-sorted list L_{zxy} of all the vertices of the orthogonal polyhedron is created, a single call to the 2D algorithm outputs the entire set of faces parallel to the Z plane. Being the first sorting criterion the z value, vertices with the same z are grouped in L_{zxy}. Then, the 2D sweep line will be performed successively with vertices on each plane parallel to z, and the alternate search will look for vertices in lists L_{zxy} and L_{zyx} as required. Needless to say, the 2D algorithm must be adapted to handle 3D coordinates. In summary, the polyhedron vertices are sorted using the six possible lexicographical orders into six lists $L_{zxy}, L_{zyx}, L_{yxz}, L_{yzx}, L_{xyz}, L_{xzy}$, and then the 2D algorithm is applied three times.

Notice that this approach may be slower than the original 3D idea that traverses all orthogonal plane layers, while in one case the alternate search uses a single list with all the vertices, the other a collection of smaller lists are used. However, the worst-case algorithmic complexity of both approaches is the same, as all vertices of a polyhedron may be located in only two planes.

There is one thing that remains unexplained concerning the 3D boundary extraction: output faces have to be oriented according to face normals. This is left to the next section, where the processing of orthogonal pseudomanifold polyhedra is explained.

4.4.2 Managing orthogonal pseudomanifold polyhedra

The 3D boundary extraction algorithm must handle 3D non-manifold vertices in order to extract the boundary of orthogonal pseudomanifold polyhedra. There are some 3D non-manifold vertex configurations such that more than two faces meeting at a vertex lie on the same axis-aligned plane (see Figure 4.6). Therefore, the algorithm presented in the previous section is not able to completely handle pseudomanifolds.

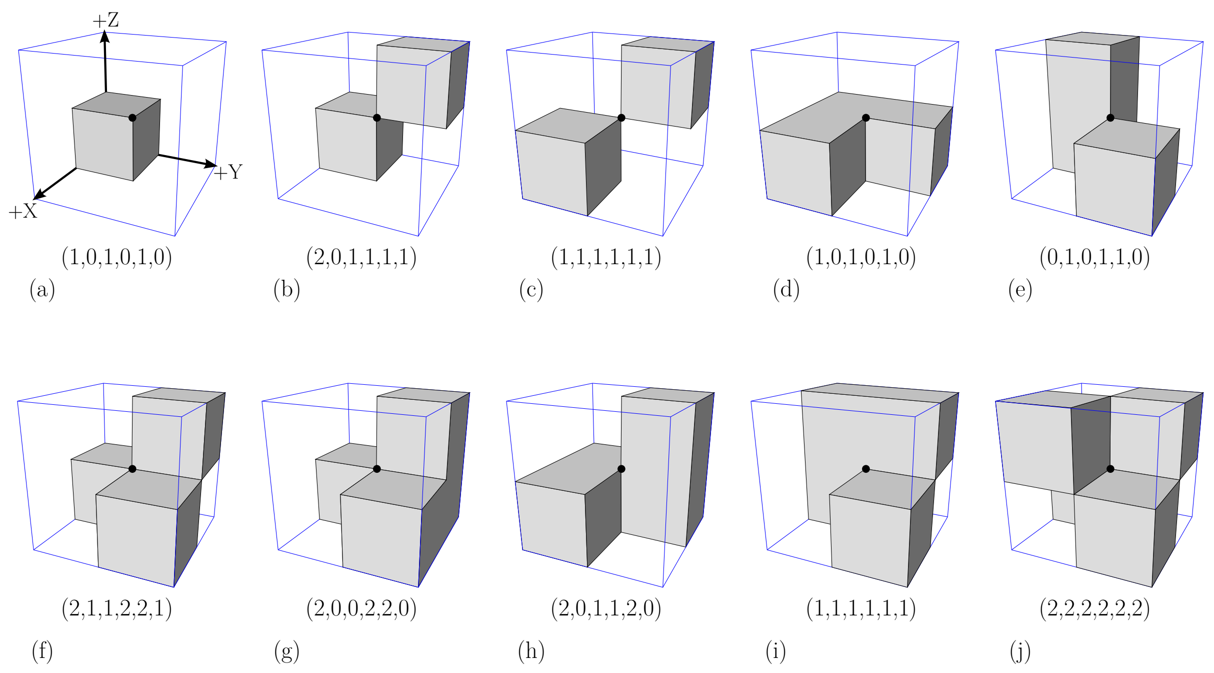


Figure 4.6: Vertices of an orthogonal pseudomanifold polyhedron and the number of times they must be inserted in the six sorted lists. Printed values correspond to orientations $(+X, -X, +Y, -Y, +Z, -Z)$.

Observe that any orthogonal non-manifold 3D vertex has at most two incident faces having the same *oriented* plane. Therefore, a list of vertices for each oriented axis direction $(+X, -X, +Y, -Y, +Z$ and $-Z)$ is computed and then it is proceeded as in the manifold case, applying six times the 2D algorithm. The 3D algorithm needs an input list of vertices with six values associated to each vertex that indicate the number of times (0, 1 or 2) the vertex must be repeated for each call of the 2D algorithm. Except for the V4 vertices, these values coincide with the number of incident faces to the vertex that are oriented with each different coordinate axis. Figure 4.6 shows manifold and non-manifold vertices and their corresponding six values. The ten cases represented in the figure cover all possible vertex configurations; the 256 total cases can be deduced by either rotations and/or dual configurations of these ten cases [4].

As each call to the 2D algorithm is performed for each oriented axis, it is straightforward to orient the output set of edge loops, by reversing either the hole loops or the external loops depending on the axis orientation.

4.4.3 Algorithm complexity

Lemma 3. *The boundary representation of an orthogonal polyhedron can be computed in $O(n \log n)$ time and $O(n)$ space, where n is the number of vertices of the input orthogonal polyhedron.*

Proof. The algorithm requires six lexicographical sortings of the input list of vertices followed by six calls to the 2D boundary extraction algorithm. Thus, the overall time and space complexity is the same as in the two-dimensional case. □

4.5 B-Rep extraction from other representation schemes

In this section, it is explained how the input vertex list is obtained from different existing representation schemes. In particular, the set of vertices of a non-manifold polyhedron and the classification to obtain their corresponding six values have to be computed from different enumeration models: the voxel model and the octree model; and from models based on vertex lists: the neighborhood representation, the EVM model and the weighted vertex list representation.

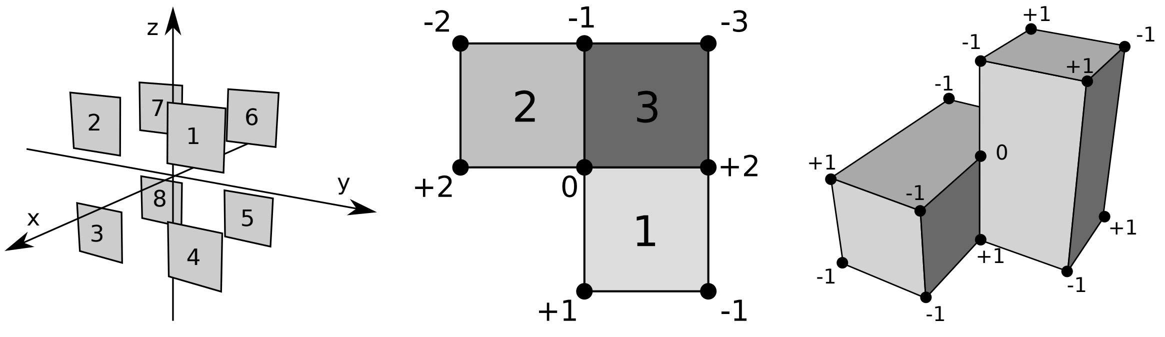
In the case of a voxel model, a traversal of all the voxel vertices is performed. Then, for each voxel, it is decided whether it contributes any polyhedron vertex and, in this case, the number of incident faces to the vertex. Moreover, as the voxels are usually stored in lexicographical order, the consistency between consecutive voxels can be used to compute the corresponding six values. Therefore, the time complexity depends linearly on the number of voxels. In most of the cases the number of polyhedron vertices is smaller than the number of voxels and, it is expected that the time spent to compute the input set of vertices will be greater than to compute the B-rep.

In case of an octree representation, the terminal nodes of the octree have to be visited, and the same analysis of that in the voxel model is performed. Now, the time complexity depends on the size of the shape boundary that gives the number of octree nodes to analyze.

The neighborhood representation [49] represents an orthogonal polyhedron by its set of vertices, and any vertex has attached its neighboring information. In the 3D case, the neighboring information of a vertex v gives information about the inclusion in the polyhedron of the eight vertices of a box centered in v. Thus, it is straightforward to compute the number of oriented faces incident in each vertex, necessary to compute the corresponding six values. The time complexity to compute the input vertex list is $O(n)$, because it requires the traverse of all the vertices. Thus, the B-Rep model is obtained in $O(n \log n)$ time, where n is the number of vertices.

In case of an EVM [5], the boundary extraction method can be applied directly to a two-dimensional EVM because the extreme vertices and manifold vertices coincide. For 2D pseudomanifolds a preprocess that extracts non-manifold vertices is needed. Three-dimensional EVM can be converted into its complete set of oriented faces with pseudomanifold polygons, using EVM appropriate methods [217]. Then, the 2D boundary extraction algorithm can be applied to each oriented face.

The weighted vertex list model represents a d-dimensional scalar field as a set of weighted vertices [95]. Each vertex of the set v placed at $p(v)$ with weight $w(v)$ and coordinates p_i, $i = 1 \ldots d$ modifies the scalar field by adding $w(v)$ to all points q such that $p(v) \leq q$, that is to all points q with coordinates $q_i, i = 1 \ldots d$ such that $p_i \leq q_i, \forall i = 1 \ldots d$. A set of vertices $V = \{v_1, \ldots v_n\}$, where the position of vertex v_i is denoted by $p(v_i)$ and its weight by $w(v_i)$, assigns a scalar field Q_v to any point q, being $Q_v(q) = \sum_{p(v_i)\leq q} w(v_i)$. For convenience, the authors propose to store the set of vertices as a lexicographically sorted list. An orthogonal solid is represented by a vertex list that maps the interior of the solid to 1 and the outside to 0. Figure 4.1b shows an example of an orthogonal manifold represented as a weighted vertex list. The weight can be computed by examining the scalar field values in a neighborhood of the vertex: assuming classical numbering of quadrants, in 2D, the weight of any vertex is the sum of the field values of the two even quadrants around the vertex minus the odd quadrant field values; similarly, in 3D, if octants are numbered as in Figure 4.7a, the weight of a vertex can be computed by adding the four even octant values and subtracting the odd octant values. As a consequence, for example, non-manifold 2D vertices have weight +2 or -2.



(a) **(b)** **(c)**

Figure 4.7: **(a)** Octant numbering. **(b)** and **(c)** are two examples of models represented as a weighted vertex set that have a vertex with zero weight. **(b)** A 2D multivalued scalar field. **(c)** An orthogonal polyhedron with a $V4$ vertex that has zero weight.

As the weighted vertex list represents a polyhedron with lexicographically sorted lists of weighted vertices, it seems very suitable to use the presented approach to transform from this model to a B-Rep. However, the authors state that this vertex representation is unique only if two restrictions are fulfilled: no two vertices may lie at the same point in the space, and no vertices may have zero weight [95]. Notice that in 3D, some kind of vertices of a pseudomanifold have zero weight (cases c, h and the dual configuration of c in Figure 4.6). Incidentally, multivalued 2D fields can also have zero weight vertices (see Figure 4.7b). This can easily be checked by summing and subtracting field values around the vertex. This implies that these zero weight vertices will not be in the vertex list. Therefore, some kind of vertices that are not explicit in weighted vertex list model are needed by the B-Rep extraction algorithm. Recovering these missing zero weight vertices and also the six corresponding values of all vertices can be done in a very similar way than in the EVM case explained in [217], using a traversal of vertex list that temporarily stores sections. Notice that the zero weighted missing vertices in this representation scheme are also missing in the EVM model.

4.6 Experimental results

The presented B-Rep extraction algorithms have been tested extensively with several datasets. The experiments were conducted with a database of around three thousand randomly generated voxel models and on several voxel models from scanned objects. Figure 4.8 shows the B-Rep extraction times. The available memory was limited to 16GB and the time computation was at most twenty minutes for each dataset. The presented approach revealed an average runtime of $59.3\, n \log n\ \mu s$, where n is the number of vertices of the orthogonal polyhedron. These tests have also been useful to ensure the robustness of the presented approach, since large random models include several non-manifold vertex and hole-face containment configurations. Typically, in voxel models from scanned objects, the number of voxels is hundreds of times higher than the number of vertices and only about 5% of the vertices of the shape are non-manifold.

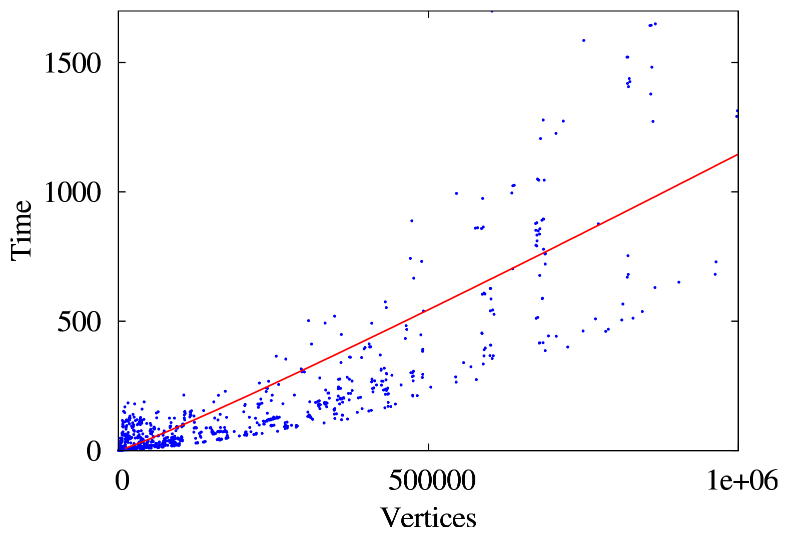


Figure 4.8: Running times of the boundary extraction algorithm. The abscissa denotes the number of input vertices. The ordinate denotes the running time in seconds. Every blue point depicts the runtime for a single dataset. The red line corresponds to the curve fitted to the expected linearithmic time complexity of the algorithm.

4.7 Conclusions

The problem of reconstructing an orthogonal pseudomanifold polygon or polyhedron given its set of vertices and, in 3D, the number of incident faces to a vertex has been studied. Algorithms to compute the B-Rep representation of any orthogonal pseudomanifold polygon and polyhedron in $O(n \log n)$ time, where n is the number of vertices, have been presented. Although the introduced approach can be used to convert spatial enumeration models as well as vertex lists based models, it is best suited for the second category of models, because in the spatial enumeration models the number of elements rules the global time complexity of the whole conversion process. However, it is also recommended for the first category, since the presented approach correctly handles all kinds of non-manifold vertices and outputs, a complete hierarchical B-Rep model which includes all hole-face relationships.

Chapter 5

Preprocess: trihedralization of orthogonal polyhedra

Contents

5.1 Introduction

An orthogonal polyhedron can have vertices with more than three incident faces, that is, non-trihedral vertices. The approach presented in Chapter 7, to compute the L_∞ Voronoi diagram of orthogonal polyhedra, requires that the vertices of the input polyhedron are trihedral.

Barequet et al. [33] noticed that the vertices of a polyhedron with more than three incident faces need to be split into trihedral vertices in order to compute the straight skeleton. During the straight skeleton shrinking process, the topology of non-trihedral vertices may change, and the splitting into trihedral vertices must capture these changes.

Definition 27. *(BRÉDIF [53]) Given an input polyhedron, the* trihedralization *process consists in changing only its topology such that the modified polyhedron is only composed of trihedral vertices and it is self-intersection free.*

The local trihedralization of a non-trihedral vertex can be seen as the dual problem of planar polygon triangulation, where polygon vertices correspond to polyhedron faces and polygon faces correspond to polyhedron vertices [53] (see Figure 5.1). Two dual vertices are connected if they have an incident edge to the non-trihedral vertex. It is known that the trihedralization problem does not always have a solution [53]. For example, Figure 5.2

shows a polyhedron vertex that cannot be trihedralized without perturbing the position of the incident faces.

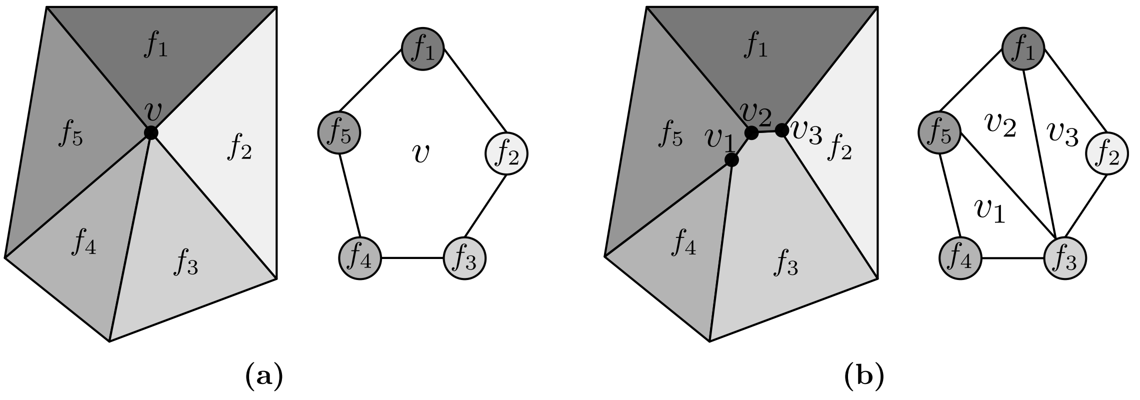


Figure 5.1: Local trihedralization of a vertex. The dual representation is shown at the right of both figures. **(a)** A vertex with five incident faces. **(b)** Trihedralization of the vertex into three trihedral vertices **(b)**.

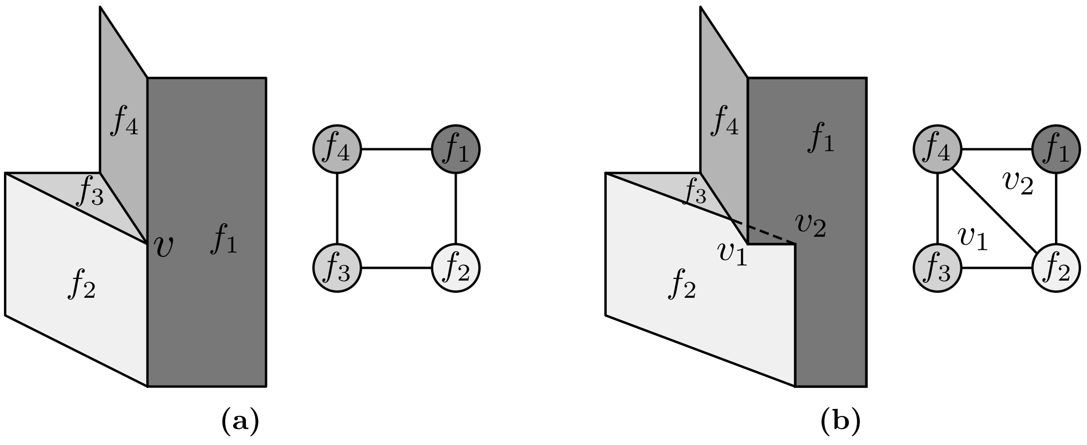


Figure 5.2: **(a)** A vertex v with four incident faces that is impossible to trihedralize. Observe that there are only two ways to triangulate the dual polygon. For more details on all possible invalid combinations see [53, p. 150]. **(b)** An invalid self-intersecting trihedralization of v, where two new trihedral vertices v_1 and v_2 are introduced.

Barequet et al. [33] trihedralize by using two complementary weighted straight skeletons [28] and consider that the adjacent faces of a non-trihedral vertex undergo a translation inwards along their normal of the same infinitesimal distance. Unfortunately, it is unproven whether this method forms a trihedralization that is not self intersecting in the general case [53]. In addition, the specific trihedralization of orthogonal polyhedra is not considered, when actually it may be needed.

A kinetic framework [115] to compute the trihedralization of a polyhedron is presented in [53]. The polyhedron faces are continuously moved during a kinetic evolution, while guaranteeing locally self-intersection free faces. The proposed approach only guarantees that the polyhedron faces are not locally self-intersecting but does not ensure that the polyhedron as a whole is non-self-intersecting. Thus, trihedralizing a polyhedron in the general case is still an open problem.

However, the trihedralization of orthogonal polyhedra is simpler, as the number of vertex configurations is finite. The main idea of the presented approach is to consider that some polyhedron faces are translated along their normal vector and inwards an infinitesimal amount, such that non-trihedral vertices are split into trihedral vertices and the polyhedron remains self-intersection free. Each polyhedron face could be moved a different infinitesimal amount. Observe that an alternative would have been to consider that the polyhedron faces are moved outwards instead of inwards.

Throughout this chapter $\mathcal{P}$ denotes an orthogonal polyhedron, either manifold or pseudomanifold. Next, the trihedralization of each vertex configuration is analyzed.

The vertices of $\mathcal{P}$ can be classified into different configurations according to the number and type of incident edges and faces [4]. Figure 5.3 shows all possible configurations that a vertex can have, without taking into account symmetries and rotations. The following parameters characterize the trihedralization of each vertex configuration:

- *Number of trihedral vertices.* A non-trihedral vertex may be split into more than one trihedral vertex.
- *Number of new zero-length edges.* Zero-length edges between new trihedral vertices may be introduced.
- *Number of possible trihedralizations.* A non-trihedral vertex may admit more than one trihedralization.
- *Number of introduced constraints.* The trihedralization of a vertex may introduce constraints between its incident coplanar faces.

The trihedralization may introduce the next types of constraints between faces of $\mathcal{P}$ that are coplanar and have the same orientation:

- Two faces of $f_1, f_2 \in \mathcal{P}$ are merged into a single one. This constraint is denoted as $f_1 = f_2$. It is equivalent to consider that the infinitesimal inwards displacement of both faces is the same.
- Two faces of $f_1, f_2 \in \mathcal{P}$ cannot be merged and the infinitesimal inwards displacement of f_1 must be greater than f_2. This constraint is denoted as $f_1 > f_2$.

Table 5.1 contains a classification of all possible trihedralizations according to the previously mentioned vertex and trihedralization properties, and a reference to a figure analyzing each trihedralization in detail. Each referenced figure at the left shows an illustration of the vertex trihedralization. The planes containing the faces of $\mathcal{P}$ are not perturbed and only the topology of $\mathcal{P}$ is modified in the process.

After a case analysis of all vertex configurations, it is observed that the only non-trihedral vertices that admit more than one possible trihedralization are the configurations

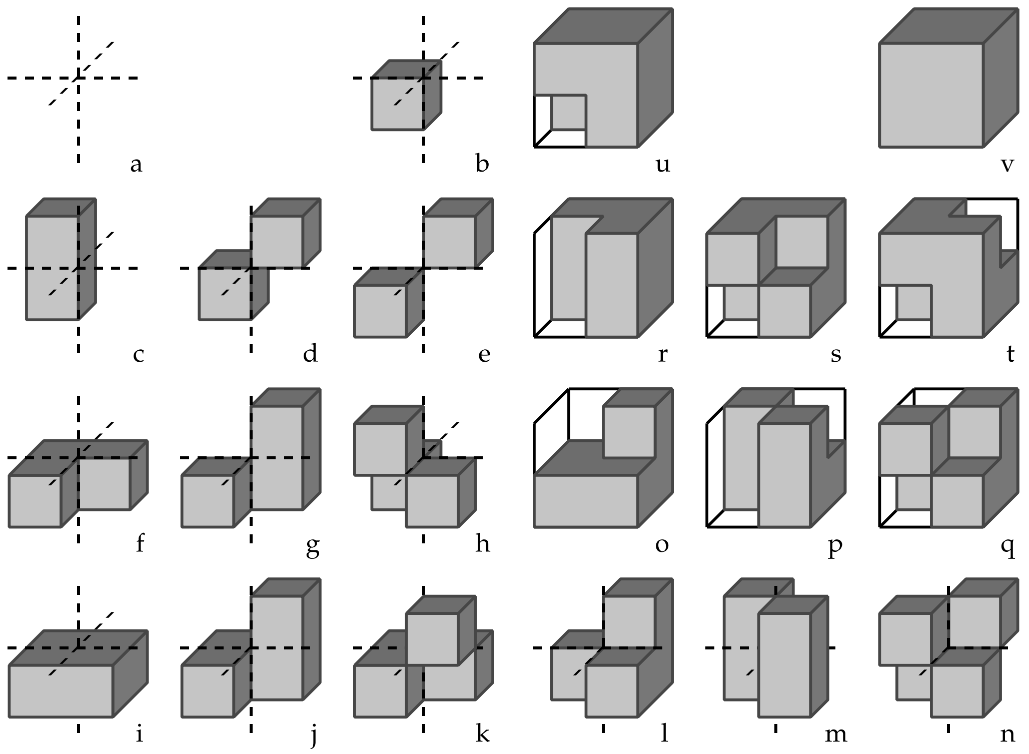


Figure 5.3: The vertices of an orthogonal polyhedron are determined according to the presence or absence of each of eight surrounding boxes. There are 2^8 combinations which, by applying rotational symmetries, may be grouped into twenty-two equivalence classes (courtesy of Aguilera [4]). Observe that some cases are the same vertex configuration.

l and *q* of Table 5.1. In fact, *q* can be seen as a vertex *l* split apart from a trihedral vertex and its analysis can be reduced to the configuration *l*. In addition, *l*, *q* and *s* are the only vertex configurations that introduce constraints between faces. Thus, most of the complexity of trihedralizing an orthogonal polyhedron is related to these three configurations.

Next, a set of trihedralizing constraints that avoid global self-intersections is presented.

Property 4. *Consider the vertex configuration* l *(see Figure 5.10a) with incident faces* f_{x1}, f_{x2}, f_{y1}, f_{y2}, f_{z1} *and* f_{z2}. *The trihedralizing constraints can be described by the next Boolean function:*

$$((f_{x1} = f_{x2}) \wedge (f_{y1} > f_{y2}) \wedge (f_{z1} > f_{z2})) \vee$$
$$((f_{x1} < f_{x2}) \wedge (f_{y1} = f_{y2}) \wedge (f_{z1} < f_{z2})) \vee$$
$$((f_{x1} > f_{x2}) \wedge (f_{y1} < f_{y2}) \wedge (f_{z1} = f_{z2}))$$

The constraints introduced by the vertex configuration *q* are analogous to the vertex configuration *l*.

Property 5. *Consider the vertex configuration* s *of Figure 5.15a with incident coplanar faces* f_{x1} *and* f_{x2}. *The trihedralizing constraint can be described by the Boolean function* $f_{x1} = f_{x2}$.

	Manifold		Pseudomanifold									
Configuration	j	l	d	g	p	k	s	e	t	h	q	n
Incident faces	4	6	4	5	5	6	6	6	6	9	9	12
Trihedral vertices	2	2	2	1	3	2	2	2	6	3	3	4
Trihedralizations	1	3	1	1	1	1	1	1	1	1	3	1
New edges	1	0	0	0	2	0	0	0	6	0	2	0
Constraints	0	3	0	0	0	0	1	0	0	0	3	0
Figure	5.9	5.10	5.11	5.12	5.13	5.14	5.15	5.16	5.17	5.18	5.19	5.20

Table 5.1: Topological and trihedralization properties of non-trihedral vertices.

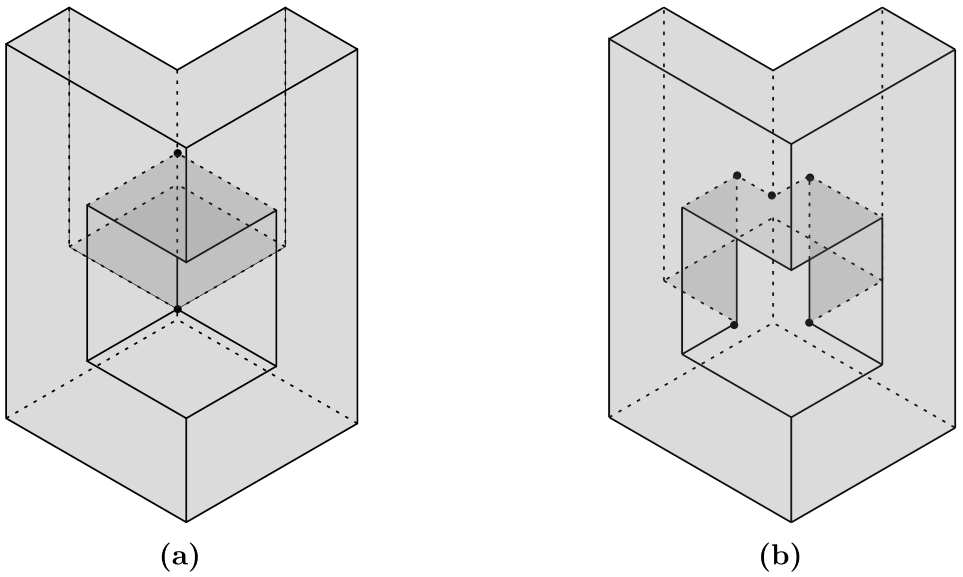


Figure 5.4: The local trihedralization of two vertices of type p and s connected by a common edge yields a global trihedralization that is self-intersection free.

As previously mentioned, in the general case of polyhedra a trihedralization that is locally self-intersection free in the neighborhood of a vertex does not necessarily guarantee global self-intersection [53]. Fortunately, the simple structure of the orthogonal polyhedra allows to obtain a trihedralization by simply using a local criterion at each non-trihedral vertex (see Figure 5.4 as an example) and satisfying all the trihedralization constraints.

Definition 28. *The* constraint satisfaction problem (CSP) *is the process of finding a solution to a set of constraints that impose conditions that a set of variables must satisfy.*

Lemma 4. *Finding a trihedralization of $\mathcal{P}$ is equivalent to solving the constraint satisfaction problem over the set of constraints induced by the vertex configurations* l, q *and* s.

Proof. The set of constraints is restricted to equality and strict inequalities induced by vertex configurations l, q and s (see Property 4 and 5). All the other vertex configurations can be directly trihedralized while ensuring non-self-intersections. Note that constraints can only be defined between coplanar faces of $\mathcal{P}$ with the same orientation. Every face of $\mathcal{P}$ is associated to an integer, that represents the infinitesimal amount of displacement, as follows. Consider a maximal subset $\mathcal{F}$ of coplanar faces. Every face $f \in \mathcal{F}$ can be mapped to an integer of the finite domain $\{1 \dots |\mathcal{F}|\}$. The problem of trihedralizing is reduced to mapping these integers to every face of $\mathcal{P}$ such that the all the constraints introduced by the trihedralization are satisfied. □

5.2 Existence and uniqueness of trihedralization

It is important to determine if a trihedralization always exist. It is known that a trihedralization of general polyhedra without self-intersections may not exist (see Brédif [p. 149][53]). In the restricted setting of orthogonal polyhedra, and under the introduced constraint set, it also may not exist.

Lemma 5. *Every orthogonal polyhedron without vertices of configuration* l *or* q *can be trihedralized with the presented method. However, there are cases of orthogonal polyhedra, with vertices of configuration* l *or* q*, such that the trihedralizing constraints cannot be satisfied.*

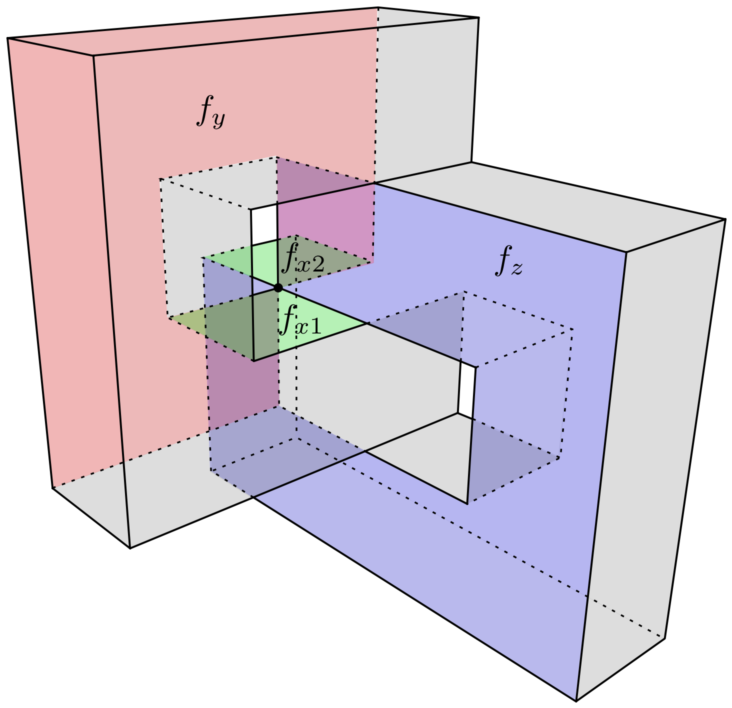
Proof. The vertex configurations q and l introduce the same restrictions, as the case q is reduced to the case l. If a vertex l has only five different incident faces it is still possible to trihedralize it. But if only four different incident faces (see Figure 5.5) converge to the vertex l, the constraint of Property 4 cannot be satisfied in any way. Thus, in some cases and with the presence of vertices l or q, a self-intersection free trihedralization may not exist.

On the other hand, if there are not vertices l and q, all the remaining trihedralizations do not introduce any constraint apart from the case s. However, the equality constraint of case s (see Property 5) can always be satisfied, as only equality constraints are present. Thus, without the presence of vertices l or q a trihedralization always exists. □

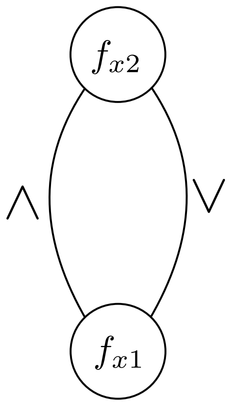
Lemma 6. *There can be at most* 3^n *different trihedralizations of* $\mathcal{P}$*, where* n *is the number of vertices of configuration* l *and* q.

Proof. For each vertex of type configuration l and q there are three different ways to trihedralize it. Thus, 3^n different trihedralizations exist. □

As there can exist several trihedralizations, the straight skeleton of $\mathcal{P}$ (see Chapter 7) may have a different topology depending on the trihedralization. In fact, the topology changes only affect zero-length edges or zero-area faces of the straight skeleton that are contained in other edges or faces of the skeleton. Thus, any trihedralization yields a geometrically equivalent straight skeleton.



(a)



(b)

Figure 5.5: (a) A vertex configuration l with four different incident faces f_{x1}, f_{x2}, f_y and f_z. The faces axis-aligned with the x coordinate are drawn in green, the axis-aligned with the y coordinate in red, and the axis-aligned with the z coordinate in blue. Two pairs of incident faces are forced to be equally displaced inwards in order to trihedralize the vertex. As a consequence, it is not possible to satisfy Property 4. **(b)** Graph drawing of the constraints, impossible to satisfy, imposed by the faces f_y and f_z.

5.3 Computing the trihedralization

To solve a CSP on a finite domain is a NP-complete problem in general. Specific classes of constraint satisfaction problems are shown to be tractable. A problem is tractable if there exists a polynomial time algorithm to solve it. Tractability can be obtained by restricting the domain of variables and the constraints. In particular, two kinds of restrictions can be considered:

- *Relational restrictions.* These restrictions determine the domain and the values satisfying the constraints.
- *Structural restrictions.* These restrictions determine the way constraints are distributed over the variables.

The objective is to design a specific algorithm that takes into account these restrictions. Concerning the relational restrictions, it has been shown that the domain of the values satisfying the constraints is bound by each finite set of coplanar faces of $\mathcal{P}$ (see Lemma 4). Some structural restrictions of the trihedralization problem are described next.

Definition 29. *Let $\mathcal{P}$ be an orthogonal polyhedron. Let V be set of vertices of configuration* l, q *or* s *of $\mathcal{P}$. Let E be the set of tuples (v_1, v_2), where $v_1, v_2 \in V$ are two vertices of a common face of $\mathcal{P}$. Then, $\mathcal{G}_v = (V, E)$ is an undirected graph (see Figure 5.6b).*

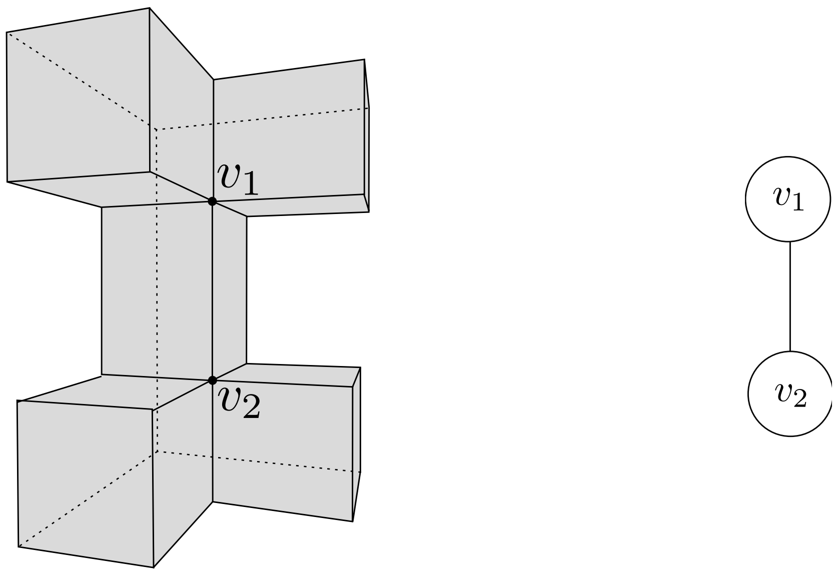


Figure 5.6: (a) An orthogonal polyhedron with two vertices v_1 and v_2 of configuration l. **(b)** Graph drawing of G_v.

Definition 30. *A* connected component *of an undirected graph is a maximal subgraph where any two vertices are connected to each other by edges.*

Lemma 7. *The complete trihedralization of $\mathcal{P}$ can be divided into independent trihedralizations associated to each connected component of $\mathcal{G}_v$.*

Proof. Consider each connected component of $\mathcal{G}_v$. It is possible to trihedralize independently each one, as the constraints that are introduced in one connected component cannot affect the ones of another. □

Definition 31. Backtracking *[139] is a general algorithm for finding all or some solutions to a computational problem, that incrementally builds candidates to the solutions, and abandons each partial candidate, that is backtracks, as soon as it determines that the partial candidate cannot be a valid solution.*

It is possible to design a simple backtracking algorithm that solves the CSP trihedralization problem. Initially, the presented algorithm takes into account the constraints induced by vertices s. Then, for each vertex l and q, a pair of coplanar faces is merged while ensuring that the current constraints are satisfied. The algorithm backtracks if the current constraints cannot be satisfied. Next, some definitions and properties of vertices of $\mathcal{G}_v$ and its associated constraints are given.

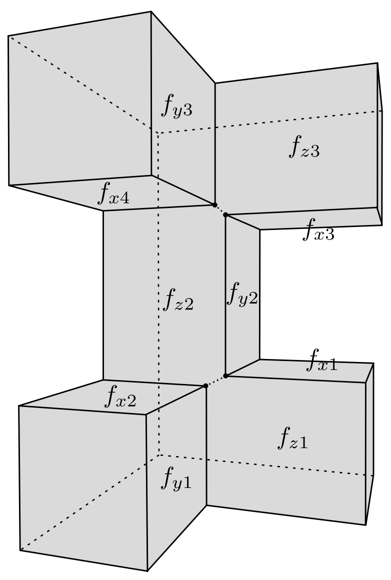
Definition 32. *Let v be a vertex of $\mathcal{G}_v$. At any moment of the backtracking process, the set $constraints\,(v)$ contains all the equality and strict inequality constraints associated to the incident faces of the vertex v.*

The next property can be easily seen as there can only be, for each three orthogonal directions, one constraint associated to vertex of $\mathcal{G}_v$.

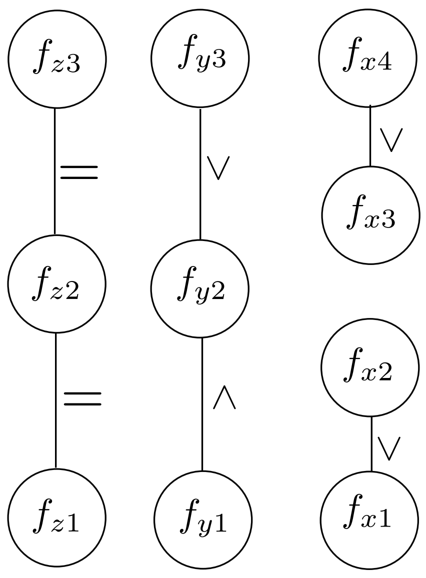
Property 6. *Let v be a vertex of $\mathcal{G}_v$. At any moment of the backtracking process, $|constraints\,(v)| \leq 3$.*

Property 7. *Let v be a vertex of $\mathcal{G}_v$ that has not yet been trihedralized. At any moment of the backtracking process, if v is a vertex configuration* l *or* q *and $|constraints\,(v)| = 1$, then it is always possible to merge a pair of coplanar faces of v (see Property 4) such that all the current constraints are satisfied.*

Definition 33. *$\mathcal{G}_c$ is an undirected graph where a vertex of $\mathcal{G}_c$ corresponds to a face of an orthogonal polyhedron $\mathcal{P}$ and an edge between two graph vertices of $\mathcal{G}_c$ indicates a constraint between the two faces, either equality or strict inequality (see Figure 5.7b).*



(a)



(b)

Figure 5.7: (a) An illustrative visualization of a trihedralization of the polyhedron shown in Figure 5.6a. **(b)** A drawing of the graph $\mathcal{G}_c$ that contains the constraints between faces.

The constraints between pairs of coplanar faces are stored in the data structure $\mathcal{G}_c$ that must support insertion and deletion of constraints. Moreover, $\mathcal{G}_c$ must support the search of the constraint associated to a pair of faces, if it exists.

The algorithm proceeds as follows. First of all, the connected components of the graph $\mathcal{G}_v$ are computed (see Lemma 7). Then, for every vertex of configuration s, its associated equality constraints are inserted into $\mathcal{G}_c$. Then a backtracking process (see

Algorithm 4) is performed over the remaining vertices of configuration l and q in order to find a trihedralization of $\mathcal{P}$. For every vertex, a different pair of coplanar faces to be merged is selected until its associated constraints are satisfied. This is always possible if less than two constraints are associated to the vertex (see Property 7). Otherwise, the algorithm may backtrack to the previously processed vertex of $\mathcal{G}_v$.

Algorithm 3 Algorithm to trihedralize $\mathcal{P}$.

```
$\mathcal{G}_v \leftarrow$ non-trihedral $\mathcal{P}$ vertices of configuration $s$, $l$ and $q$
for $v \in \mathcal{G}_v$ do
    if $v$ is a vertex configuration $s$ then
        Insert constraint of $v$ in $\mathcal{G}_c$
    end if
end for
$CC \leftarrow$ connected components of $\mathcal{G}_v$ (only consider vertices $l$ and $q$)
for $\mathcal{G}'_v \in CC$ do
    if $\neg$ selectMergeFace($\mathcal{G}'_v$, first vertex of $\mathcal{G}'_v$) then (see Algorithm 4)
        It is impossible to trihedralize $\mathcal{P}$
    end if
end for
```

$\mathcal{G}_v$ and $\mathcal{G}_c$ can be stored as an *adjacency list* [71], such that insertions of vertices and edges are performed in constant time and deletions in linear time with respect to the number of graph edges. In the case of $\mathcal{G}_c$, searching the constraint between two graph vertices can be done in linear time, with respect to the number of graph vertices. The connected components of $\mathcal{G}_v$ can be computed straightforwardly in linear time, using either breadth-first search or depth-first search. Besides, the backtracking process may require an exponential number of steps (see Lemma 6) and the algorithm remains NP-complete. However, experimental results over real and randomly generated datasets confirm that in practice few backtracking steps are required.

5.4 Experimental results

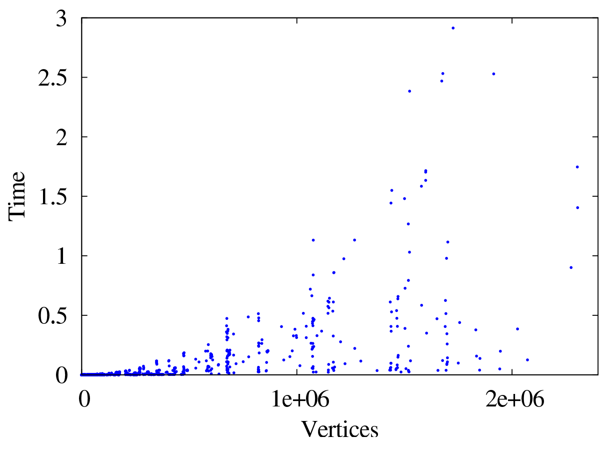
The constraint satisfaction problem defined by the trihedralization problem has been implemented by the presented approach and a publicly available constraint solver [111]. The experiments were conducted with a database of around one thousand real datasets and orthogonal polyhedrons generated randomly. The available memory was limited to 16GB and the time computation was at most twenty minutes for each dataset. The presented approach was on average sixty times faster than the generic constraint solver (see Figure 5.8). Only 0.0001% of the datasets had some vertex impossible to trihedralize, which is almost negligible (see Figure 5.5). As expected, the datasets impossible to trihedralize were the most time consuming of all. However, all the real datasets could be trihedralized.

Algorithm 4 Backtracking algorithm that selects a pair of incident coplanar faces of v to merge. v is a vertex configuration l or q.

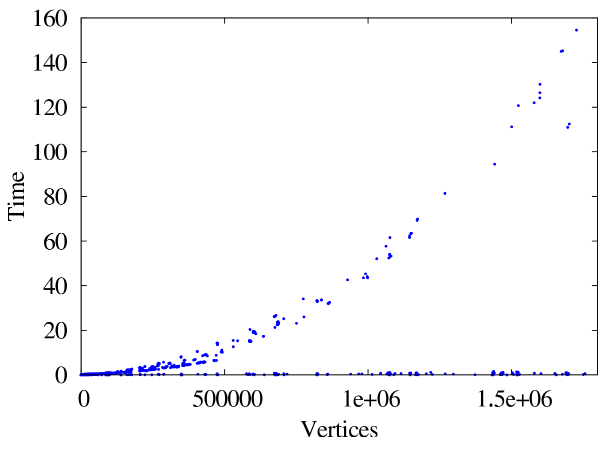
```
function selectMergeFace($\mathcal{G}_v$,$v$)
    if all vertices of $\mathcal{G}_v$ processed then
        return true
    end if
    for $dir \in$ the three orthogonal directions do
        Let $cdir$ be the three constraints of merging the faces of $v$ parallel to $dir$
        Find current constraints of $v$ in $\mathcal{G}_c$
        if $cdir$ satisfy the current constraints of $v$ then
            Insert $cdir$ in $\mathcal{G}_c$
            if $\neg$ selectMergeFace($\mathcal{G}_v$, next vertex of $\mathcal{G}_v$ to process) then
                Delete $cdir$ from $\mathcal{G}_c$
            else
                return true
            end if
        end if
    end for
    return false
end function
```

5.5 Conclusions

The problem of modifying the topology of an orthogonal polyhedron in order to only have trihedral vertices and self-intersection faces has been analyzed. This problem is known as the trihedralization problem. It has been shown that in the case of orthogonal polyhedra the trihedralization problem is equivalent to a constraint satisfaction problem. In addition, it has been shown that in some cases a trihedralization may not be possible (see Section 5.2). Although, a specific algorithm to compute the trihedralization has been introduced. Experimental results confirm the efficiency of the presented algorithm compared with a state of the art CSP solver. Moreover, the number of vertices from the tested datasets that do not admit a trihedralization has been very small and has only appeared in randomly generated datasets.

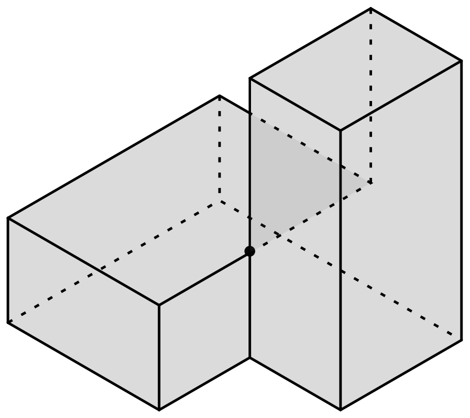


(a) Presented approach.

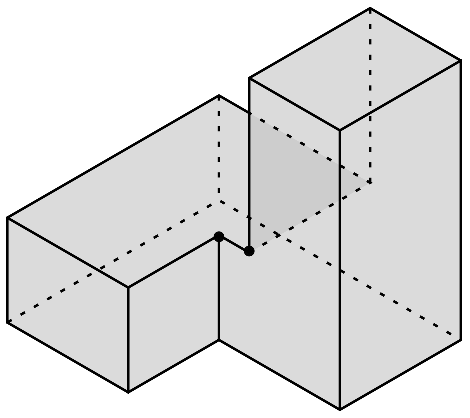


(b) Approach of [111].

Figure 5.8: Running times of different methods to compute the trihedralization of $\mathcal{P}$. The abscissa denotes the number of vertices of $\mathcal{G}_v$. The ordinate denotes the running time in seconds. Every blue point depicts the runtime for a single dataset.

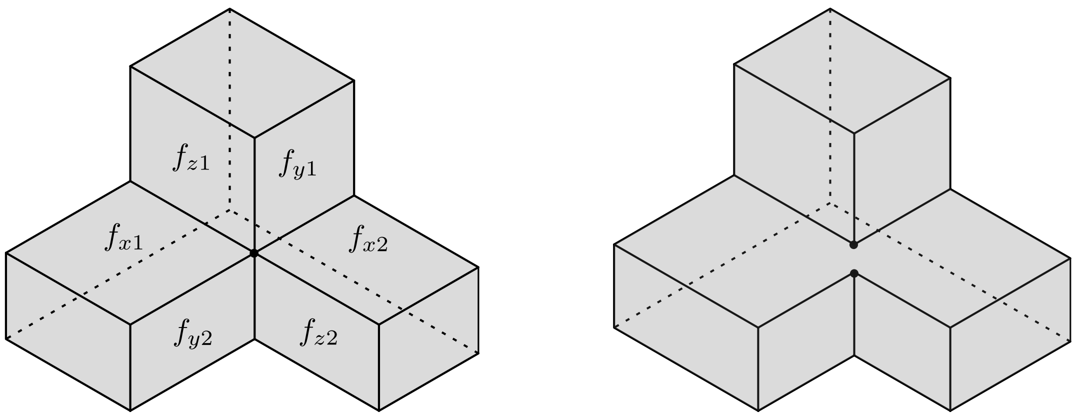


(a) Original vertex



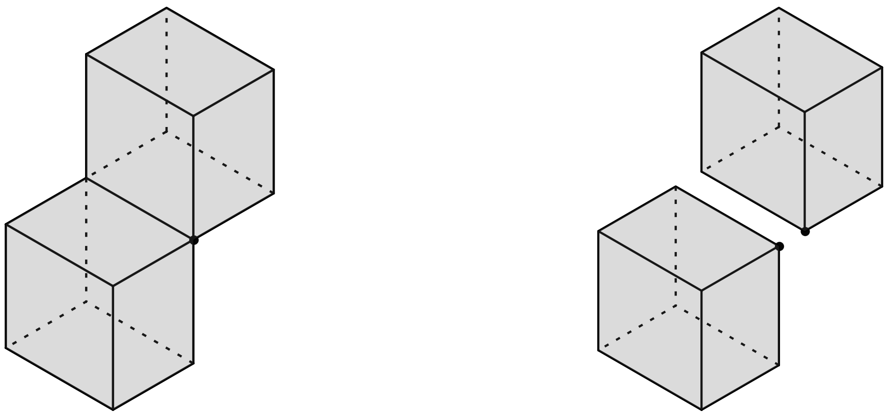
(b) Trihedralized vertex

Figure 5.9: The manifold vertex configuration j has four incident faces, two of them being coplanar with opposed orientation.



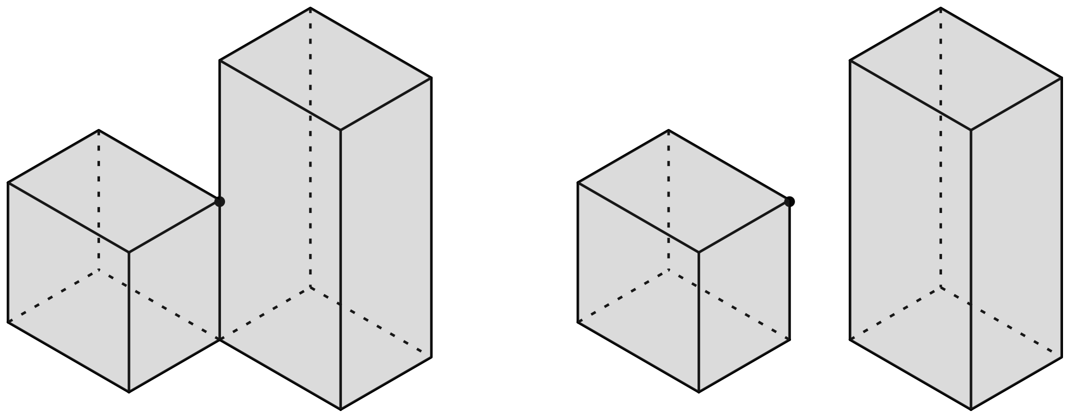
(a) Original vertex **(b)** Trihedralized vertex

Figure 5.10: The manifold vertex configuration l introduces three constraints between pairs of coplanar faces and admits three different trihedralizations depending on the merged pair. In (b), the pair of faces f_{x1} and f_{x2} are merged. Thus, in this particular case the constraints $f_{x1} = f_{x2}$, $f_{y1} > f_{y2}$ and $f_{z1} > f_{z2}$ must be taken into account.



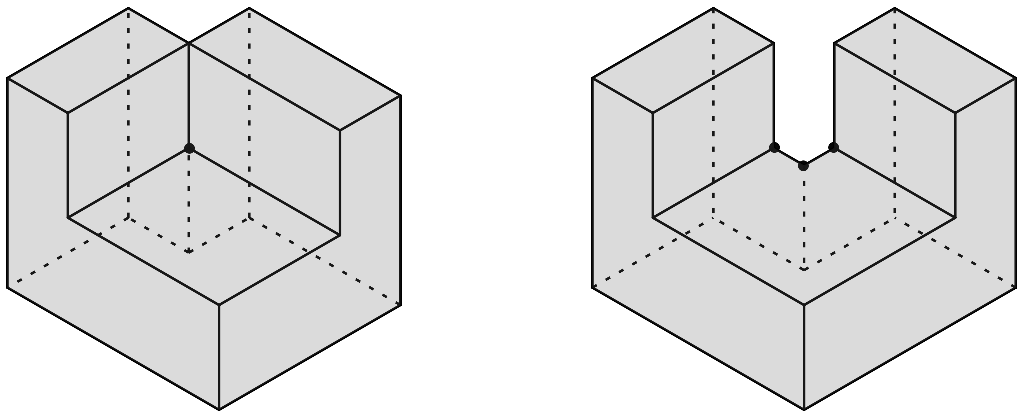
(a) Original vertex **(b)** Trihedralized vertex

Figure 5.11: The pseudomanifold vertex configuration d has four incident faces, where two pairs of faces are coplanar with opposed orientation. Two new trihedral vertices are introduced.



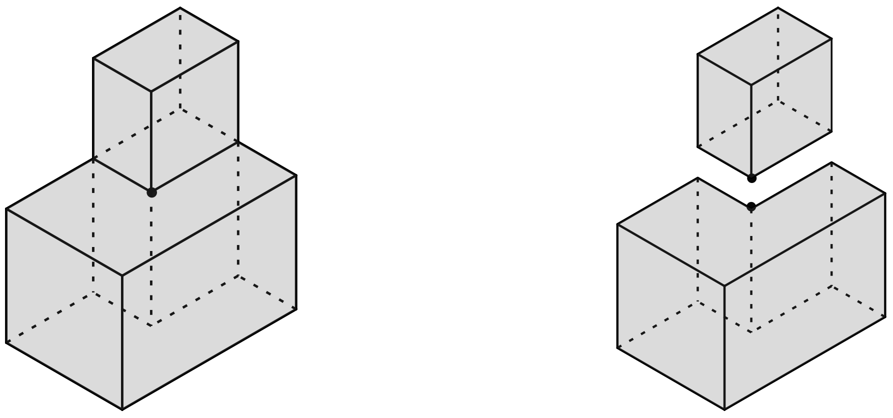
(a) Original vertex **(b)** Trihedralized vertex

Figure 5.12: The pseudomanifold vertex configuration g has five incident faces, where two pairs of them are coplanar. Only one new trihedral vertex is introduced.



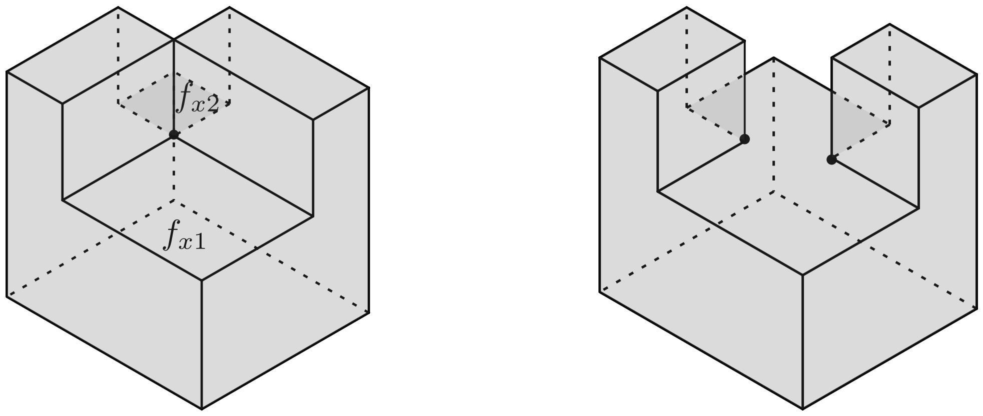
(a) Original vertex **(b)** Trihedralized vertex

Figure 5.13: The pseudomanifold vertex configuration p has five incident faces, where two pairs of them are coplanar. Three new trihedral vertices are introduced, lying in the remaining non-coplanar incident face. Although two new trihedral vertices may suffice to trihedralize it, a complementary third trihedral vertex is needed in order to ensure that the trihedralized polyhedron remains orthogonal.



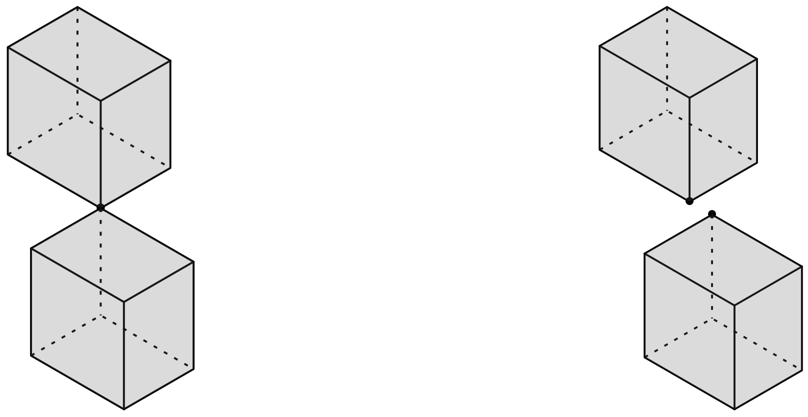
(a) Original vertex **(b)** Trihedralized vertex

Figure 5.14: The pseudomanifold vertex configuration k has six incident faces, where three pairs are coplanar faces with opposed orientation. Two new trihedral vertices are introduced.



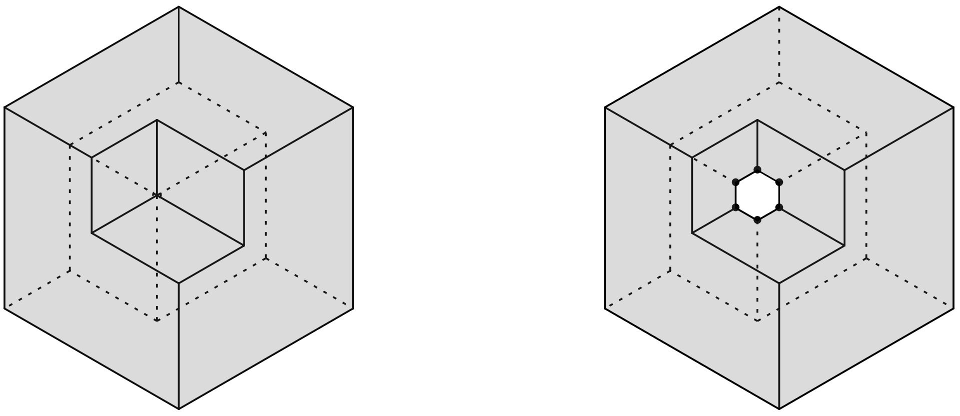
(a) Original vertex **(b)** Trihedralized vertex

Figure 5.15: The pseudomanifold vertex configuration s has six incident faces, where two pairs of faces are coplanar with opposed orientation and the other one has the same orientation. The pair of coplanar faces with the same orientation are merged. In (b), the constraint $f_{x1} = f_{x2}$ must be considered.



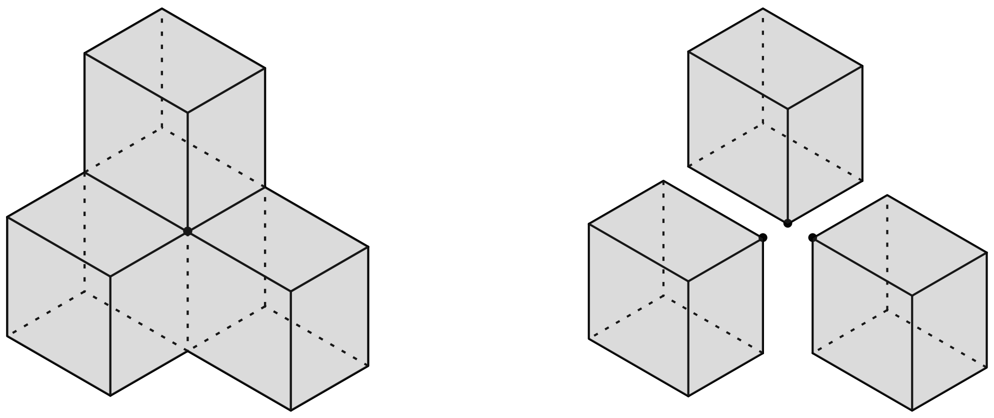
(a) Original vertex **(b)** Trihedralized vertex

Figure 5.16: The pseudomanifold vertex configuration e has six incident faces, where three pairs are coplanar faces with opposed orientation. Two new trihedral vertices are introduced.



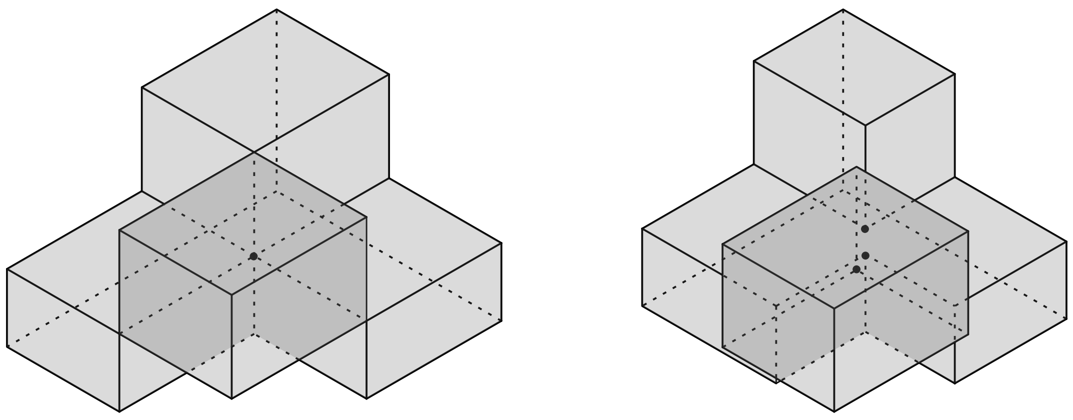
(a) Original vertex **(b)** Trihedralized vertex

Figure 5.17: The pseudomanifold vertex configuration t has six incident faces, where three pairs are coplanar faces with opposed orientation. Six new trihedral vertices are introduced. Although three new trihedral vertices may suffice to trihedralize it, three more complementary trihedral vertices are needed in order to ensure that the trihedralized polyhedron remains orthogonal.



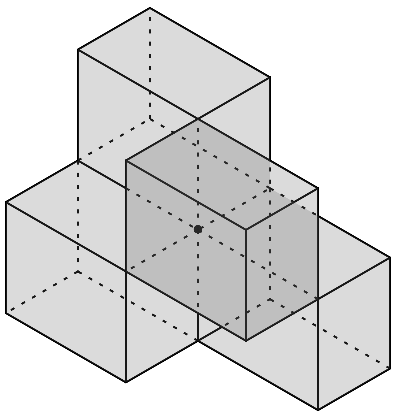
(a) Original vertex **(b)** Trihedralized vertex

Figure 5.18: The pseudomanifold vertex configuration h has nine incident faces, where three pairs are coplanar faces with the same orientation. Three new trihedral vertices are introduced.



(a) Original vertex **(b)** Trihedralized vertex

Figure 5.19: The pseudomanifold vertex configuration q has nine incident faces. The trihedralization introduces a vertex configuration of type l and a simple trihedral vertex.


(a) Original vertex

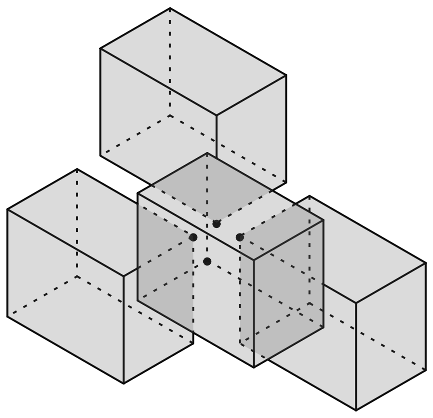

(b) Trihedralized vertex

Figure 5.20: The pseudomanifold vertex configuration n has twelve incident faces, where three pairs of coplanar faces have the same orientation. Four new trihedral vertices are introduced.

Chapter 6

L_∞ Voronoi diagram of orthogonal polygons

Contents

6.1 Introduction

It has been observed that the L_∞ Voronoi diagram and the straight skeleton of an orthogonal polygon in general position coincide (see Section 3.1.5). Thus, the Voronoi edges may serve as a skeletal representation of the polygon. Apart from the algorithm of Barequet et al. [33], which is mostly theoretical, there has not been found in the literature other algorithms to compute the L_∞ Voronoi diagram of polyhedra. Thus, it is desirable to have a two-dimensional algorithm which can be easily extended to the three-dimensional case.

In this chapter two specific implementations of a sweep line algorithm to compute the L_∞ Voronoi diagram of an orthogonal polygon are presented. One of the algorithms processes the edges of the orthogonal polygon, while the other one processes its vertices. The algorithm that processes the vertices has been developed in order to extend it to three dimensions. In addition, a detailed proof of the equivalence between the L_∞ Voronoi diagram and the straight skeleton of an orthogonal polygon is given. This chapter is an extended and adapted version of the article [160].

6.2 Previous definitions and properties

In the following, some basic previous definitions and properties are introduced. In this chapter, $\mathcal{P}$ denotes an orthogonal polygon. The Voronoi diagram of $\mathcal{P}$ under the L_∞ metric is denoted as $V_\infty(\mathcal{P})$ (see Definition 21).

Property 8. *(PAPADOPOULOU et al. [175]) Let $\mathcal{P}$ be in general position. The Voronoi diagram $V_\infty(\mathcal{P})$ has exactly $2n-2$ vertices and $2n-3$ edges, where n is the number of vertices of $\mathcal{P}$.*

Lemma 8. *Three axis-aligned open half-planes in $\mathbb{R}^2$ define a square if and only if two of them are parallel, their intersection is not empty and the remaining half-plane is perpendicular.*

Proof. It is clear that if three orthogonal half-planes are parallel they cannot define a square. Consider, without loss of generality, the next three open half-planes, two of them parallel and the other one perpendicular:

$$h_1 : x > c_1 \ , \ h_2 : x < c_2 \ , \ h_3 : y > c_3 \qquad c_1 \neq c_2$$

The intersection of half-planes h_1 and h_2 is non empty if $c_1 > c_2$. In this case, the square is bound by the abscissas c_1 and c_2, and the ordinates c_3 and $c_3 + c_2 - c_1$. The length of the square edges is $c_1 - c_2$ and the coordinate of its center is $\left(\frac{c_1+c_2}{2}, c_3 + \frac{c_1-c_2}{2}\right)$. □

Property 9. *Three edges of $\mathcal{P}$ define a square if and only if their associated half-planes oriented inside $\mathcal{P}$ define a square (see Lemma 8) and the intersection between the square boundary and every edge is not empty.*

Definition 34. *A Voronoi vertex of $V_\infty(\mathcal{P})$ is* degenerate *if it is L_∞ equidistant to more than three edges of $\mathcal{P}$.*

Definition 35. *An orthogonal polygon $\mathcal{P}$ is in* general position *if it is an orthogonal manifold polygon, without collinear edges and $V_\infty(\mathcal{P})$ has no degenerate Voronoi vertices.*

In the following Sections 6.3 and 6.4, $\mathcal{P}$ is considered to be in general position.

6.3 Edge-based algorithm

The edge-based algorithm to compute $V_\infty(\mathcal{P})$ is based on the more general sweep line algorithm (see Section 2) of Papadopoulou et al. [175] which calculates the L_∞ Voronoi diagram of planar straight line graphs. Let $\mathcal{L}$ be the vertical sweep line, sweeping from left to right. The priority queue $\mathbb{E}$ stores the events encountered during the sweep. The events are either vertical edges of $\mathcal{P}$ or candidate Voronoi vertices of $V_\infty(\mathcal{P})$.

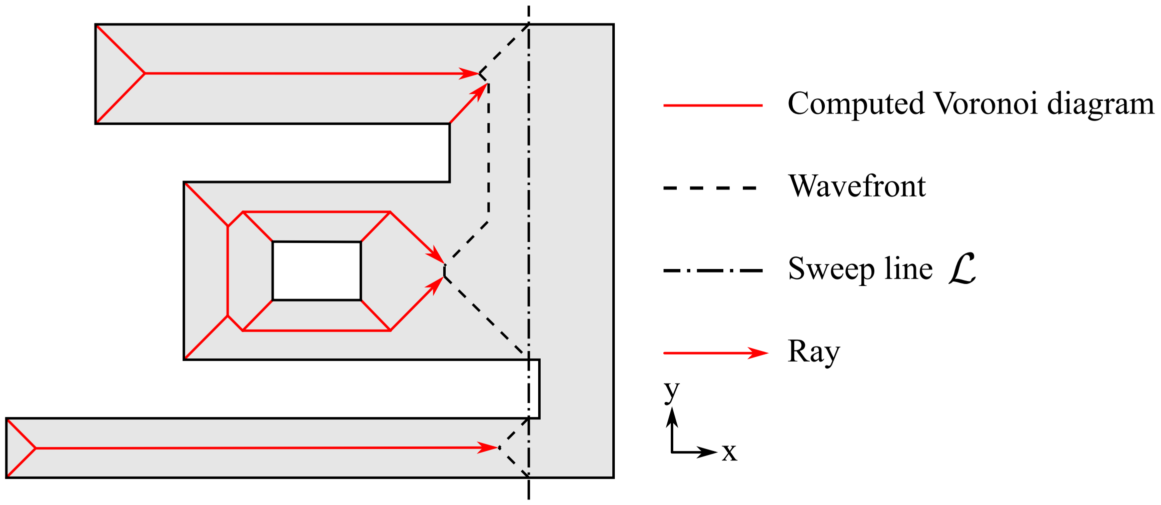


Figure 6.1: Scheme of the sweep line algorithm to compute the L_∞ Voronoi diagram of an orthogonal polygon. The orthogonal polygon, drawn in gray, is in general position.

Definition 36. *At any moment t of the sweeping process, $\mathcal{L}$ partitions the edges of $\mathcal{P}$ into three subsets: E_t are the edges completely to the left of $\mathcal{L}$, I_t are the edges that intersect $\mathcal{L}$, and R_t are the edges completely to the right of $\mathcal{L}$. C_t is the portion of I_t that lies at the left of $\mathcal{L}$. $\mathcal{L}_t$ are the edges of $\mathcal{P}$ contained in the line $\mathcal{L}$ and with the interior of $\mathcal{P}$ at the left, plus the portion of $\mathcal{L}$ that belongs to the interior of $\mathcal{P}$.*

An invariant of the algorithm is that, at any moment t of the sweeping process, the L_∞ Voronoi diagram of $E_t \cup C_t \cup \mathcal{L}_t$ is already computed.

Definition 37. *At any moment t of the sweeping process, the* wavefront *is the boundary of the Voronoi regions (see Definition 20) induced by the edges $\mathcal{L}_t$. The Voronoi edges induced by $\mathcal{L}_t$ are the* wavefront edges *and their Voronoi vertices are the* wavefront vertices.

Lemma 9. *The wavefront edges are monotone with respect to the sweep line.*

Proof. Let $p = (x, y)$ be a point in $\mathcal{L}_t$, and let e be the edge of $\mathcal{L}_t$ where p lies. Let $p' = (x', y)$ be the point with ordinate y and smallest abscissa x' in the Voronoi region of e. The maximal square centered at p' is empty, and contains every square centered at a point on the segment $[p, p']$, hence all of these are empty as well, and must belong to the Voronoi region of e. So there cannot exist any point of the wavefront between p and p' and it follows that the wavefront edges are monotone with respect to the sweep line (see Figure 6.2). □

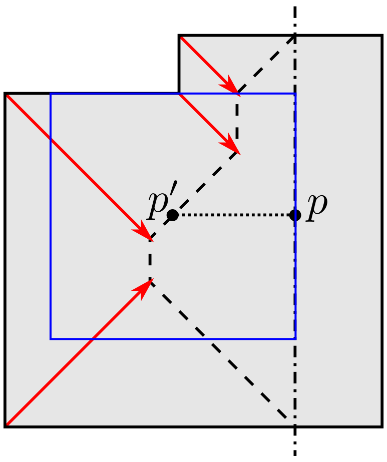


Figure 6.2: Illustration of Lemma 9. There cannot exist any point of the wavefront between p and p'. The maximal square associated to p' is drawn in blue.

Definition 38. *At any moment of the sweeping process, a* ray *is a half-line that supports a Voronoi edge, with its origin in a vertex of $\mathcal{P}$ or a Voronoi vertex of $V_\infty(\mathcal{P})$. The end of the Voronoi edge remains to be calculated. Once calculated, the ray becomes a Voronoi edge of $V_\infty(\mathcal{P})$ (see Figure 6.1).*

In general position, a ray is contained in the L_∞ bisector of two edges e_1 and e_2 of $\mathcal{P}$ and is denoted by $\{e_1, e_2\}$.

Definition 39. *The ray emanating from a vertex of $\mathcal{P}$ is defined by its two incident edges and it is called* vertex ray*.*

Definition 40. *The intersection point between two rays $\{e_1, e_2\}$ and $\{e_1, e_3\}$ is the center of the square defined by the edges e_1, e_2 and e_3 (see Property 9).*

Note that as $\mathcal{P}$ is in general position, the two intersecting rays $\{e_1, e_2\}$ and $\{e_1, e_3\}$ must share an edge e_1.

Definition 41. *The intersection point between a ray $\{e_1, e_2\}$ and an edge e_3 of $\mathcal{P}$ is the center of the square defined by the edges e_1, e_2 and e_3 (see Property 9).*

Definition 42. *At any moment t of the sweeping process, the set of rays that define a square with $\mathcal{L}$ are called* wavefront rays *(see Figure 6.1).*

Property 10. *At any moment t of the sweeping process, the vertices of the wavefront are:*

- *The center of the square defined by a wavefront ray and $\mathcal{L}$.*
- *The intersection between a horizontal edge of I_t and $\mathcal{L}$.*

The wavefront edges can be obtained by connecting consecutive wavefront vertices because they are monotone with respect to the sweep line (see Lemma 9). The wavefront rays are stored in a height-balanced binary tree $\mathbb{W}$, which is the status data structure (see Section 2), and sorted according to the ordinate of their associated wavefront vertices. Note that $\mathbb{W}$ describes the combinatorial structure of the wavefront. The wavefront data structure only needs to be updated each time an event occurs. Initially, the priority queue $\mathbb{E}$ is the sorted set of edge events induced by the vertical edges of $\mathcal{P}$. The priority of an edge event is determined by the edge constant x coordinate. Two different types of events may occur during the sweep:

- **Edge event**: it occurs when $\mathcal{L}$ meets a vertical edge of $\mathcal{P}$. The priority of an edge event is defined by the abscissa of the vertical edge.
- **Ray-ray junction event**: the intersection of two rays that could induce a Voronoi vertex. The priority of a ray-ray junction event is defined by the rightmost abscissa of the square defined by both rays. That is, as soon as the intersection is reached by the wavefront.

The correctness of the next algorithms follows using the same arguments as in [175]. In the following, it is described how to process each event.

6.3.1 Edge event

Two types of edge events are distinguished depending on the orientation of the vertical edges that induce them: vertical edges that have the interior of $\mathcal{P}$ at the right, and vertical edges with the interior of $\mathcal{P}$ at the left.

Property 11. *If the incident vertical edge of a vertex of $\mathcal{P}$ has the interior of $\mathcal{P}$ at the right then, after processing the edge, its vertex ray must be a wavefront ray.*

Consider first that a vertical edge has the interior of $\mathcal{P}$ at the right (see Figure 6.3). In this case, two new wavefront rays, that correspond to the two vertex rays of the edge, are inserted in $\mathbb{W}$.

Second, consider that a vertical edge has the interior of $\mathcal{P}$ at the left (see Figure 6.4). In this case, the algorithm must compute the portion of the wavefront that belongs to the Voronoi region of the vertical edge. First, the intersection between both vertex rays of the edge and the wavefront edges is computed. Then, the portion of the wavefront between the two intersected points becomes part of the final Voronoi diagram and the wavefront rays between the two intersected points are deleted from $\mathbb{W}$. In addition, consider a reflex vertex of the vertical edge, with an incident horizontal edge e_h, whose ray intersects the wavefront edge e_i. In this case, a new wavefront ray $\{e_h, e_i\}$ is inserted in $\mathbb{W}$.

The computation of the intersection between a vertex ray, whose interior of $\mathcal{P}$ is on its left, and the wavefront edges can be done efficiently with a binary search in $\mathbb{W}$ (see Figure 6.5). Two cases are distinguished depending on the vertex. Consider first that the incident horizontal edge of the vertex has the interior of $\mathcal{P}$ above (see Figure 6.5a). Let e_b and e_a be two consecutive wavefront edges, such that e_b is below e_a, and let $\{e_b, e_a\}$ be its associated wavefront ray. Let l_y denote the lower ordinate of the square defined by $\{e_b, e_a\}$ and $\mathcal{L}$. If the ordinate of the vertex lies below l_y then all the wavefront edges

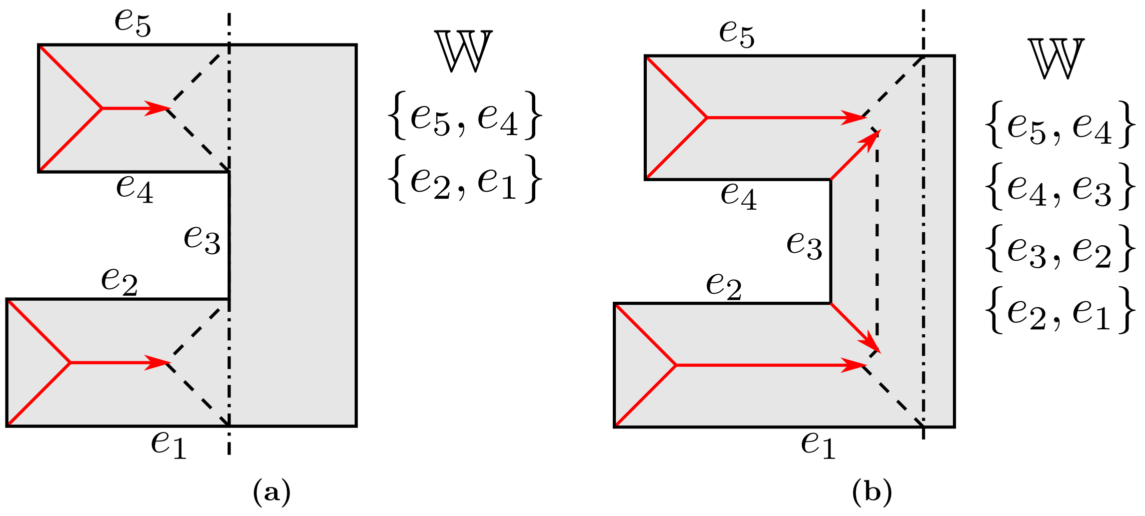


Figure 6.3: **(a)** Edge event processing of the vertical edge e_3, with the interior of $\mathcal{P}$ at the left. Two new wavefront rays $\{e_4, e_3\}$ and $\{e_3, e_2\}$ are inserted in $\mathbb{W}$. In addition, a new ray-ray junction event induced by the wavefront rays $\{e_5, e_4\}$ and $\{e_4, e_3\}$, and another one induced by the wavefront rays $\{e_3, e_2\}$ and $\{e_2, e_1\}$ are inserted in $\mathbb{E}$. **(b)** Status of $\mathbb{W}$ after processing the edge event.

above and including e_a can be excluded from the search. If the ordinate of the vertex lies above l_y then all the wavefront edges below e_b are excluded from the search. Finally, if the ordinate of the vertex is equal to l_y then the intersection is a degenerate Voronoi vertex. In the same way, if the incident horizontal edge of the vertex has the interior of $\mathcal{P}$ below the reverse search is applied (see Figure 6.5b).

Finally, the intersection of the wavefront rays inserted in $\mathbb{W}$ with other neighboring wavefront rays in $\mathbb{W}$ may induce new ray-ray junction events that are inserted in $\mathbb{E}$. See Algorithm 5 for an overview of the edge event processing.

6.3.2 Ray-ray junction event

A ray-ray junction event corresponds to the intersection between two rays (see Definition 40). The priority of a ray-ray junction is defined by the rightmost abscissa of the square defined by both junction event rays: as soon as the intersection is reached by the wavefront (see Figure 6.6). A ray-ray junction event defines a Voronoi vertex if the two rays still belong to $\mathbb{W}$ and are neighbors in $\mathbb{W}$. Observe that the junction event rays could have been removed from $\mathbb{W}$ by a previous event with higher priority, or $\mathbb{W}$ could have been updated such that both rays are no longer neighbors in $\mathbb{W}$. If the ray-ray junction defines a Voronoi vertex, the two rays are deleted from $\mathbb{W}$ and the two incident Voronoi edges to the Voronoi vertex are inserted in $V_\infty(\mathcal{P})$. In addition, a wavefront ray defined by the two different edges between the two rays is inserted in $\mathbb{W}$, and new ray-ray junction events induced by its neighboring wavefront rays in $\mathbb{W}$ may be found and inserted in $\mathbb{E}$. See Algorithm 6 for an overview of the ray-ray junction event processing.

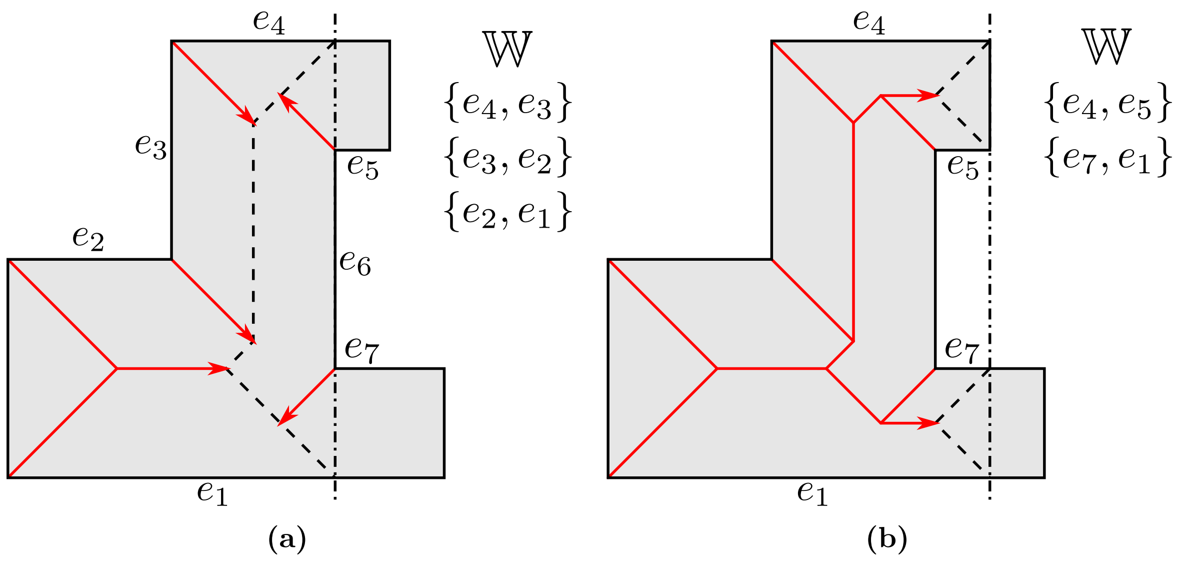


Figure 6.4: (a) Edge event processing of the vertical edge e_6, with the interior of $\mathcal{P}$ at the left. The vertex ray $\{e_5, e_6\}$ intersects the wavefront edge e_4. $\{e_6, e_7\}$ intersects the wavefront edge e_1. The portion of $\mathbb{W}$ between both intersected points is added to the completed $V_\infty(\mathcal{P})$ and the rays $\{e_4, e_3\}$, $\{e_3, e_2\}$ and $\{e_2, e_1\}$ are deleted from $\mathbb{W}$. Finally, as both vertices of the edge e_6 are reflex, the rays $\{e_4, e_5\}$ and $\{e_7, e_1\}$ are inserted in $\mathbb{W}$. **(b)** Status of $\mathbb{W}$ after processing the edge event.

6.3.3 Algorithm complexity

Lemma 10. *The $V_\infty(\mathcal{P})$ can be computed by the edge-based sweep line algorithm in $O(n \log n)$ time and $O(n)$ space, where n is the number of vertices of $\mathcal{P}$.*

Proof. As a consequence of Property 8 there are $O(n)$ events to process. In all the events it is required to insert and delete a constant number of rays in $\mathbb{W}$, each of which can be done in $O(\log n)$ time using a binary search in $\mathbb{W}$. Processing an edge event or a ray-ray junction event, that inserts new wavefront rays in $\mathbb{W}$, requires to check new ray-ray junction events induced by neighboring wavefront rays, also in $O(\log n)$ time. Processing an edge event associated to a vertical edge that has the interior of $\mathcal{P}$ on its left requires computation of two intersection points of the wavefront and can be done in $O(\log n)$ time with a binary search in $\mathbb{W}$, as explained in Section 6.3.1. Thus, $O(\log n)$ time is required to process an event and the overall time complexity is $O(n \log n)$. As $\mathbb{W}$ and $\mathbb{E}$ are linear data structures and $O(n)$ events are processed, the space complexity is $O(n)$. □

6.4 Vertex-based algorithm

The vertex-based algorithm is a variation of the edge-based algorithm, such that the events are associated to vertices of $\mathcal{P}$ or $V_\infty(\mathcal{P})$. In addition, the vertex-based algorithm allows the insertion in $\mathbb{W}$ of rays that does not intersect $\mathcal{L}$. These rays are sorted in $\mathbb{W}$ according to the ordinate of its origin. For example, the wavefront ray $\{e_2, e_5\}$ of

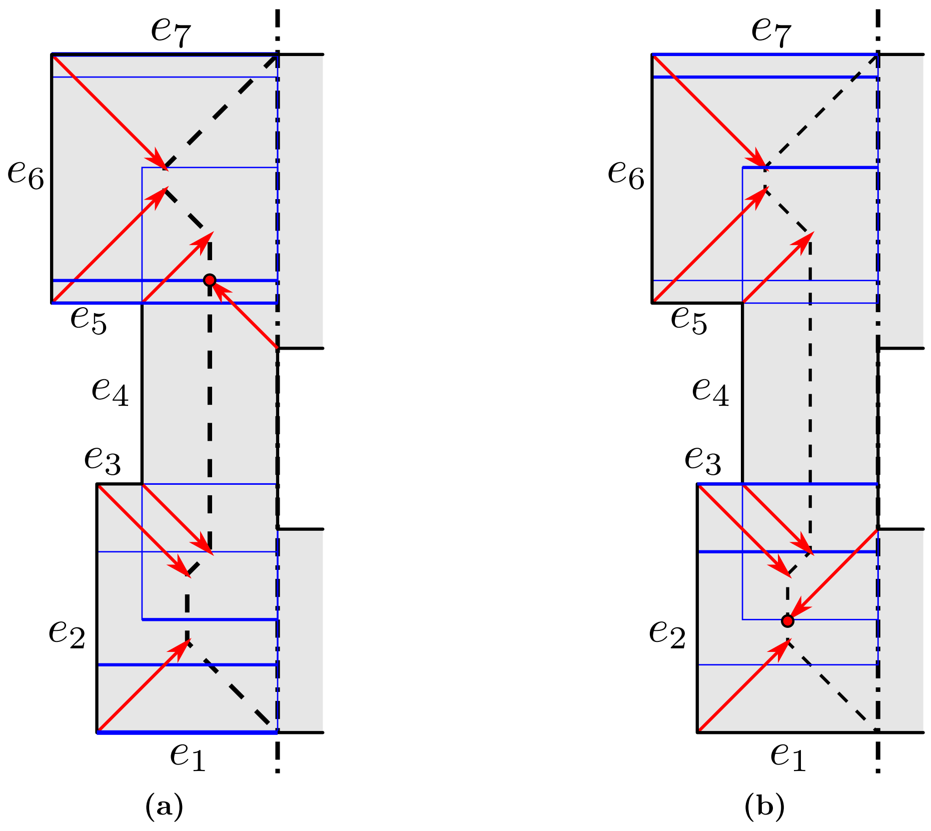


Figure 6.5: Computation of the wavefront edge intersected by a vertex ray. The blue squares are defined by the wavefront rays and $\mathcal{L}$. **(a)** The thick blue lines correspond to the lower ordinate of the squares. The binary search is finally bounded by the rays $\{e_5, e_4\}$ and $\{e_4, e_3\}$. Thus, the wavefront edge e_3 is intersected. **(b)** The thick blue lines correspond to the lower ordinate of the squares. The binary search is finally bounded by the rays $\{e_3, e_2\}$ and $\{e_2, e_1\}$. Thus, the wavefront edge e_2 is intersected.

Figure 6.7b does not intersect $\mathcal{L}$. One of the edges defining a ray that does not intersect $\mathcal{L}$ must be a vertical edge with the interior of $\mathcal{P}$ at the left. It is important for the vertex-based algorithm to determine the direction of these rays.

Property 12. *Let $\{e_1, e_2\}$ be a ray, with its origin in the Voronoi vertex equidistant to the edges e_1, e_2 and e_3. If any vertex bounded by the edges e_1e_3 and e_2e_3 form a reflex angle, then the ray is oriented towards the interior of $\mathcal{P}$ defined by e_3. Otherwise, the ray is oriented in the opposite direction.*

At any moment of the sweeping process, if two rays of $\mathbb{W}$ are defined by the same two edges, a Voronoi edge connecting their origins is added to $V_\infty(\mathcal{P})$ and both rays are deleted from $\mathbb{W}$. For example, the ray $\{e_3, e_4\}$ of Figure 6.7c is deleted from $\mathbb{W}$ after processing the vertex ray defined by the same edges.

Three different types of events may occur during the sweep:

- **Vertex event**: the occurrence of a vertex of $\mathcal{P}$. The priority of a vertex event is defined first by the vertex abscissa and then by its ordinate.

Algorithm 5 Edge event processing algorithm of the edge-based approach.

```
if edge has the interior of $\mathcal{P}$ on its right then
    Insert the two wavefront rays of the edge vertices in $\mathbb{W}$
else if edge has the interior of $\mathcal{P}$ on its left then
    Compute the intersection of the two rays of the edge vertices and $\mathbb{W}$
    Insert the portion of $\mathbb{W}$ between the two intersected points in $V_\infty(\mathcal{P})$
    Delete the wavefront rays between the two intersected points from $\mathbb{W}$
    if any vertex of the edge is reflex then
        Let $e_h$ be the incident horizontal edge to the reflex vertex
        Let $e_i$ be the intersected edge by the reflex vertex ray
        Insert the ray $\{e_h, e_i\}$ in $\mathbb{W}$
    end if
end if
Insert ray-ray junction events, induced by new rays of $\mathbb{W}$ and its neighbors, in $\mathbb{E}$
```

Algorithm 6 Ray-ray junction event processing algorithm of the edge-based approach.

```
Let the ray-ray junction event be defined by two rays $\{e_3, e_2\}$ and $\{e_2, e_1\}$
if both rays are neighbors in $\mathbb{W}$ then
    Delete both rays from $\mathbb{W}$
    Insert the two incident Voronoi edges to the Voronoi vertex in $V_\infty(\mathcal{P})$
    Insert the new wavefront ray $\{e_3, e_1\}$ in $\mathbb{W}$
    Insert ray-ray junction events, induced by ray $\{e_3, e_1\}$ and its neighbors, in $\mathbb{E}$
end if
```

- **Junction event**: the intersection between two rays or a ray and an edge that could induce a Voronoi vertex. The priority of a junction event is defined first by the left-most abscissa and then by the lowest ordinate of the square induced by the intersection. There exist two types of junction events:
  - **Ray-ray junction event**: the intersection of two rays.
  - **Ray-edge junction event**: the intersection of a ray and an edge of $\mathcal{P}$.

The events of $\mathbb{E}$ are sorted lexicographically according to its priority. If the priority of a vertex event and a junction event coincide, the vertex event is processed first. If two junction events have the same priority, the junction event whose square is bigger is processed first. Initially, $\mathbb{E}$ is the sorted set of vertex events induced by the vertices of $\mathcal{P}$. New events may be inserted in $\mathbb{E}$ each time an event is processed. The events are processed until $\mathbb{E}$ is empty. In the following, it is presented how to process each event. The ray-ray junction event processing is not detailed because it coincides with the edge-based algorithm one (see Section 6.3.2).

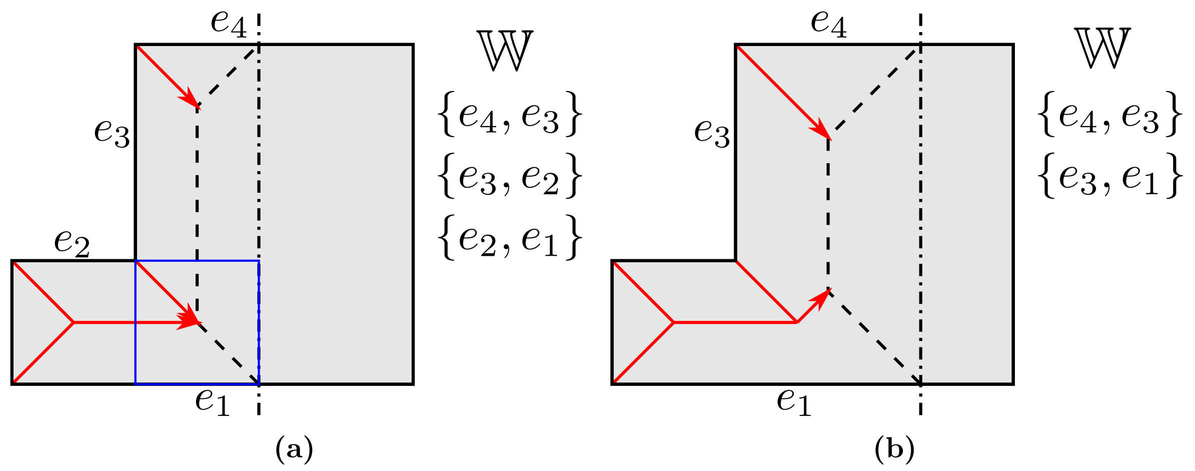


Figure 6.6: **(a)** Ray-ray junction event processing induced by the wavefront rays $\{e_3, e_2\}$ and $\{e_2, e_1\}$. Square defined by both rays is shown in blue. Both rays are deleted from $\mathbb{W}$ and their associated Voronoi edges are inserted in $V_\infty(\mathcal{P})$. A new wavefront ray (e_3, e_1) is inserted in $\mathbb{W}$ and a ray-ray junction event induced by the wavefront rays (e_4, e_3) and (e_3, e_1) is inserted in $\mathbb{E}$. **(b)** Status of $\mathbb{W}$ after processing the ray-ray junction event.

6.4.1 Vertex event

The vertex event processing is done as follows. First, the vertex ray is inserted in $\mathbb{W}$. If the vertex has the interior of $\mathcal{P}$ at the left and it is reflex (see Figure 6.7a), the wavefront edge intersected by the vertex ray is computed, as described in Section 6.3.1, and a ray-edge junction event induced by the vertex ray and the intersected wavefront edge is inserted in $\mathbb{E}$. Otherwise, neighboring wavefront rays in $\mathbb{W}$ may induce new ray-ray junction events that are inserted in $\mathbb{E}$. See Algorithm 7 for an overview of the vertex event processing.

Algorithm 7 Vertex event processing algorithm of the vertex-based approach.

```
Insert the vertex ray in $\mathbb{W}$
if the interior of $\mathcal{P}$ is on the left of the vertex and the vertex is reflex then
    Compute the intersected wavefront edge $e$ by the vertex
    Insert the ray-edge junction event induced by the vertex ray and $e$ in $\mathbb{E}$
else
    Insert ray-ray junction events, induced by the vertex ray and its neighbors, in $\mathbb{E}$
end if
```

6.4.2 Ray-edge junction event

A ray-edge junction event corresponds to the intersection between a ray and an edge of $\mathcal{P}$ (see Definition 41). Consider that a ray-edge junction event is defined by the ray $\{e_4, e_5\}$ and the intersected edge e_2 (see Figure 6.7b). A Voronoi edge between the origin of the ray and the induced Voronoi vertex is inserted in $V_\infty(\mathcal{P})$. The ray $\{e_4, e_5\}$ is deleted from

$\mathbb{W}$ and two new rays $\{e_4, e_2\}$ and $\{e_5, e_2\}$ are inserted in $\mathbb{W}$. New ray-ray junction events induced by the neighboring wavefront rays in $\mathbb{W}$ of the two new rays may be found and inserted in $\mathbb{E}$. See Algorithm 8 for an overview of the ray-edge junction event processing.

Algorithm 8 Ray-edge junction event processing algorithm.

```
Let $\{e_4, e_5\}$ be the ray and $e_2$ the intersected edge
Insert the Voronoi edge connecting the Voronoi vertex and the ray origin in $V_\infty(\mathcal{P})$
Delete the ray $\{e_4, e_5\}$ from $\mathbb{W}$
Insert the ray $\{e_4, e_2\}$ and $\{e_5, e_2\}$ in $\mathbb{W}$
Insert ray-ray junction events induced by the two new wavefront rays in $\mathbb{E}$
```

6.4.3 Algorithm complexity

Lemma 11. *The $V_\infty(\mathcal{P})$ can be computed by the vertex-based algorithm in $O(n \log n)$ time and $O(n)$ space.*

Proof. The operations performed to $\mathbb{E}$ and $\mathbb{W}$ are the same as with the edge-based algorithm. Thus, the algorithm takes the same time and space complexity (see Lemma 10). $\square$

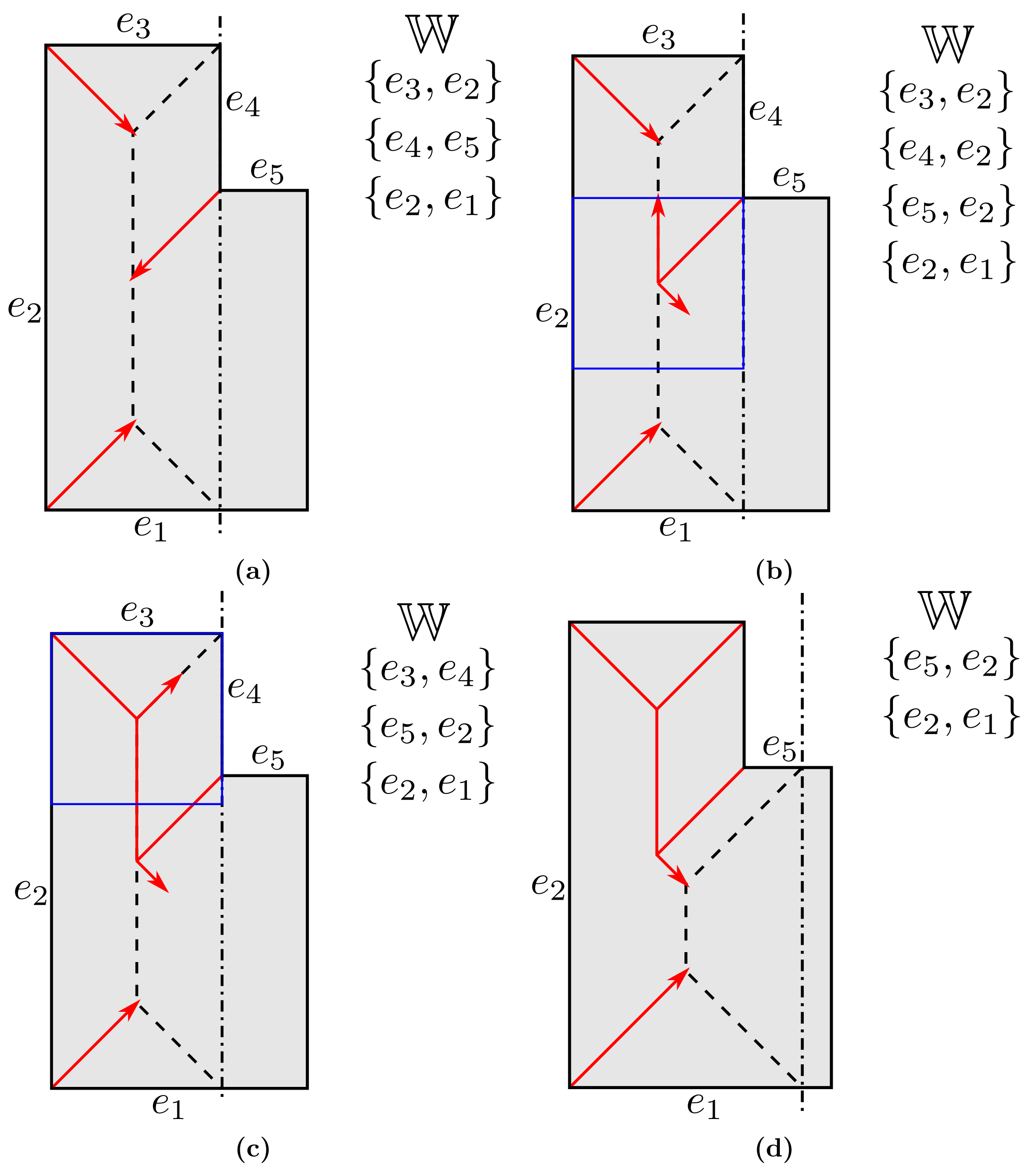


Figure 6.7: **(a)** The vertex event associated to the ray $\{e_4, e_5\}$ inserts the vertex ray in $\mathbb{W}$ and the ray-edge junction event, induced by the vertex ray and the edge e_2, in $\mathbb{E}$. **(b)** The ray-edge junction event induced by the ray $\{e_4, e_5\}$ and the edge e_2 inserts two new rays $\{e_4, e_2\}$ and $\{e_5, e_2\}$ in $\mathbb{W}$ and removes the ray $\{e_4, e_5\}$ from $\mathbb{W}$. A ray-ray junction event induced by the rays $\{e_4, e_2\}$ and $\{e_3, e_2\}$ is inserted in $\mathbb{E}$. Note that the ray $\{e_4, e_2\}$ is a ray that does not intersect $\mathcal{L}$. **(c)** The ray-ray junction event induced by rays $\{e_4, e_2\}$ and $\{e_3, e_2\}$ inserts the ray $\{e_3, e_2\}$ in $\mathbb{W}$. Later, when it is processed the vertex event associated to the ray $\{e_3, e_2\}$, both equal rays are deleted and a Voronoi edge between its origin is added to $V_\infty(\mathcal{P})$. **(d)** Status of $\mathbb{W}$ after processing all the junction and vertex events induced by the edge e_4.

6.5 Treatment of degenerate cases

In this section, $\mathcal{P}$ is considered to be an orthogonal polygon, either manifold or pseudomanifold, that could not be in general position.

Lemma 12. *Let $\mathcal{P}$ be an orthogonal pseudomanifold polygon, and let p be a point in the interior of $\mathcal{P}$. At most eight Voronoi edges of $V_\infty(\mathcal{P})$ converge to p.*

Proof. The Voronoi edges of $V_\infty(\mathcal{P})$ are axis-aligned or rotated $\frac{\pi}{4}$ radians (see Property 2). Thus, there are at most eight Voronoi edges converging to p, one for each possible direction (see Figure 6.8). □

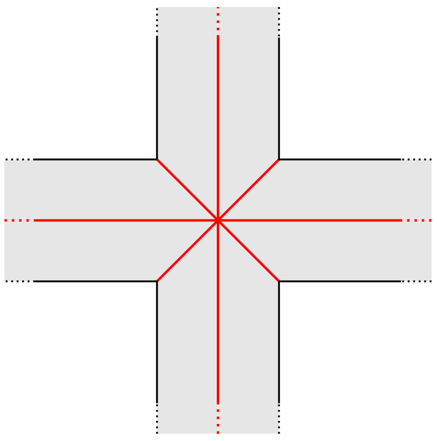


Figure 6.8: Worst-case degenerate Voronoi vertex of $V_\infty(\mathcal{P})$ induced by eight edges of $\mathcal{P}$.

Observe that under the Euclidean metric, an arbitrary amount of Voronoi edges may converge to a Voronoi vertex of a polygon. Thus, a more simple and robust approach can be designed for the case of orthogonal polygons and L_∞ metric. Three types of geometrical degeneracies are distinguished:

- **Non-manifold vertices of $\mathcal{P}$**: a vertex with four incident edges.
- **Degeneracy by collinearity**: a polygonal area whose points are L_∞ equidistant to two or more collinear edges.
- **Degeneracy by coincidence**: a Voronoi vertex L_∞ equidistant to more than three non-collinear edges of $\mathcal{P}$.

A technique to treat degeneracies based on a simulated perturbation is presented. This approach has certain connections to the simulation of simplicity proposed by Edelsbrunner et al. [89]. First, every non-manifold vertex is split into two manifold vertices that induce two vertex rays, similarly to the B-Rep extraction method of Chapter 4. Observe that this step is analogous to the three-dimensional trihedralization of orthogonal polyhedra presented in Chapter 5.

Finally, if a degeneracy is detected, the degeneracy by collinearity is treated before the degeneracy by coincidence. A degeneracy by collinearity is treated by simulating a perturbation of the collinear edges, such that only non-collinear edges are L_∞ equidistant to the Voronoi vertex. After that, a degeneracy by coincidence is treated by simulating a perturbation of the remaining set of non-collinear edges, such that the degeneracy is completely removed. In the following, it is presented how to treat each degeneracy.

6.5.1 Non-manifold polygon vertices

A non-manifold vertex of $\mathcal{P}$ is defined by two pairs of incident collinear edges. In addition, two pairs of incident edges define a convex interior angle and are considered to induce two different vertex rays.

Consider first the edge-based algorithm and a set of collinear vertical edges. It is enough to consider that the edge events associated to vertical edges with the interior of $\mathcal{P}$ at the left have higher priority than the ones with the interior at the right, in order to correctly process non-manifold vertices. Similarly, in the case of the vertex-based algorithm the vertex events associated to vertices with the interior of $\mathcal{P}$ at the left have higher priority than the ones with the interior at the right. Figure 6.9 illustrates how the non-manifold vertex v is managed.

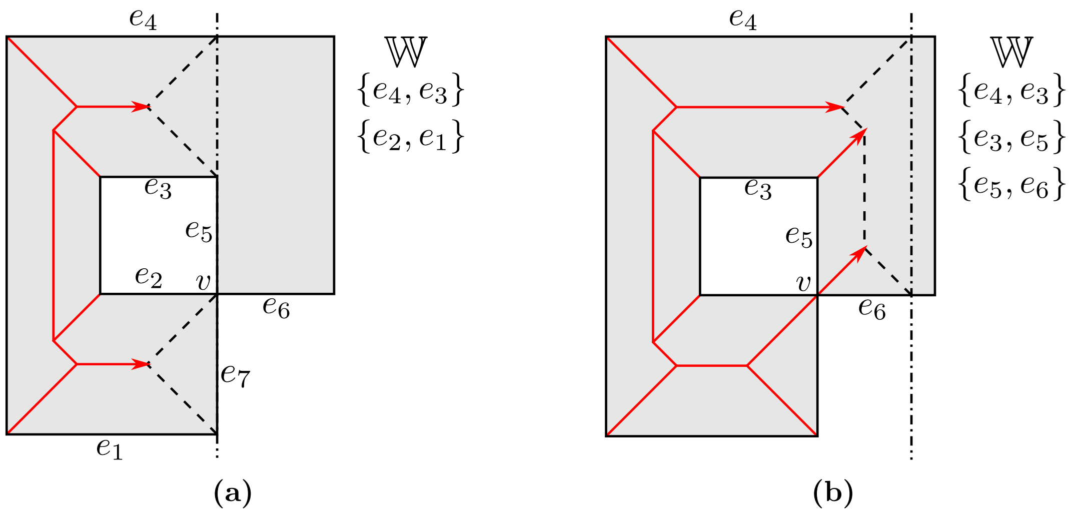


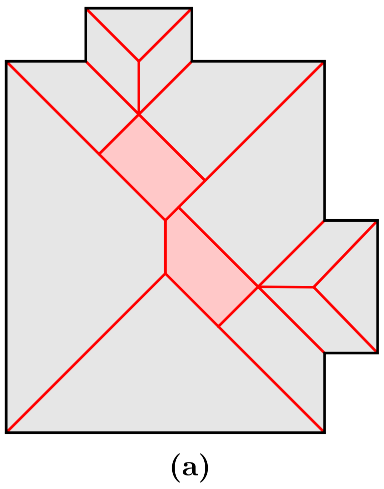
Figure 6.9: Non-manifold vertex v with four incident edges e_2, e_5, e_6 and e_7. Two rays $\{e_2, e_7\}$ and $\{e_5, e_6\}$ are induced by the vertex. **(a)** The vertical edge e_7 or the ray $\{e_2, e_7\}$ is processed first, depending on the sweep line algorithm considered. **(b)** Status of the wavefront after processing the edge e_7.

6.5.2 Degeneracy by collinearity

A degeneracy by collinearity occurs when a set points that are L_∞ equidistant to two or more collinear edges, define an area in $\mathcal{P}$ that separates different Voronoi regions.

Property 13. *The set of points of $\mathcal{P}$ that are L_∞ equidistant to two or more collinear edges of $\mathcal{P}$ can define a polygonal area, called* bisector area*, that separates Voronoi regions and it is bound by Voronoi edges (see Figure 6.10a).*

A classical method found in the Voronoi diagram literature to avoid degeneracies by collinearity is to only select some Voronoi edges bounding the bisector areas. This is done by defining a lexicographical order between the set of objects that induce the Voronoi diagram [138]. Following this idea, the next specific priority, called *collinear priority*, is additionally defined between collinear edges of $\mathcal{P}$: the rightmost horizontal edges have higher collinear priority and the above vertical edges also have higher collinear priority. As a consequence, when a degeneracy by coincidence is detected, the Voronoi edge bounding a bisector area that emanates from an edge of $\mathcal{P}$ with higher collinear priority is selected (see Figure 6.10b).


(a)

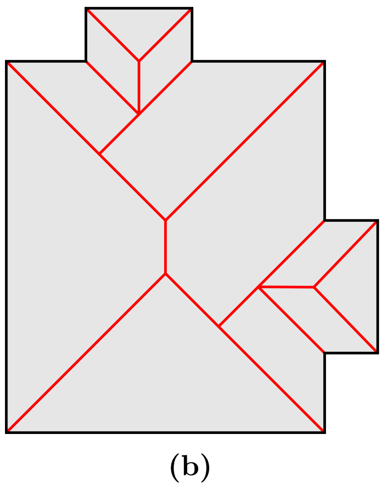

(b)

Figure 6.10: (a) The bisector areas induced by the two collinear horizontal edges and two vertical edges are shown in red. **(b)** Selection of the Voronoi edges bounding bisector areas by considering the collinear priority.

The collinear priority can also be seen as a simulated perturbation of the set of collinear edges. Suppose that the horizontal collinear edges are displaced by an infinitesimal amount towards the interior of $\mathcal{P}$, such that the rightmost edges are more displaced (see Figure 6.11). Similarly, the vertical edges above are displaced a little more.

Lemma 13. *Consider a set of collinear edges inducing a degeneracy by collinearity. It is possible to perturb them such that the degeneracy is removed.*

Proof. Consider that each edge of the collinear edges is moved inwards $\mathcal{P}$ by a different infinitesimal amount. Then, the distance that considers only the coordinate aligned with the collinear edges is different for each perturbed collinear edge. Thus, after the perturbation, the points of the bisector area must belong to a Voronoi region or to a Voronoi edge. □

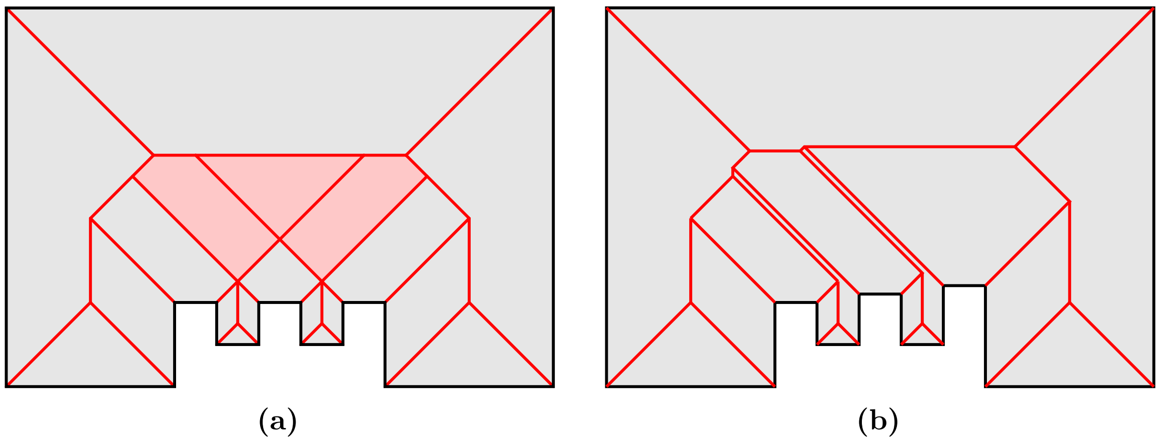


Figure 6.11: (a) An orthogonal polygon with three collinear horizontal edges inducing three bisector areas: two of them are L_∞ equidistant to two collinear edges and the other one is equidistant to three collinear edges.**(b)** Illustration of the infinitesimal perturbation of the collinear edges that simulates the collinear priority.

6.5.3 Degeneracy by coincidence

A degeneracy by coincidence occurs when a Voronoi vertex is equidistant to more than three non-collinear edges. Observe that after treating a degeneracy by collinearity, a degeneracy by coincidence still may arise.

Lemma 14. *A* coincident degenerate vertex *of $V_\infty(\mathcal{P})$ is L_∞ equidistant to four non-collinear edges of $\mathcal{P}$.*

Proof. At most eight Voronoi edges converge to a degenerate vertex is (see Lemma 12). These eight Voronoi edges must be associated to four reflex vertices with eight incident edges. However, four sets of two edges are collinear. After the collinear perturbation, only one edge of each set of four collinear edges can be L_∞ equidistant to a Voronoi vertex. Thus, at most four non-collinear edges can be L_∞ equidistant to a Voronoi vertex. □

Lemma 15. *A coincident degenerate Voronoi vertex of $V_\infty(\mathcal{P})$ can be separated into two non degenerate Voronoi vertices with the same coordinate and a zero length Voronoi edge connecting them.*

Proof. Consider, without loss of generality, that the four edges defining a coincident degenerate Voronoi vertex (see Lemma 14) are associated to the next half-planes oriented inside $\mathcal{P}$:

$$h_1 : x > c_1 \ , \ h_2 : x < c_2 \ , \ h_3 : y > c_3 \ , \ h_4 : y < c_4 \qquad c_1 \neq c_2 \ \wedge \ c_3 \neq c_4$$

Thus, as the Voronoi vertex is degenerate, it follows that $c_2 - c_1 = c_4 - c_3 > 0$. Consider that a perturbation of one infinitesimal unit ϵ is done for the horizontal edges, such that $h_1 : x > c_1 + \epsilon$ and $h_2 : x < c_2 - \epsilon$. Then, the four edges cannot define a single square as the horizontal length is now $c_2 - c_1 + 2\epsilon \neq c_4 - c_3$. Thus, two different squares can be defined by the four edges and their centers are two Voronoi vertices with the same coordinates connected by a zero length Voronoi edge. □

Thus, every time a coincident degeneracy is detected, a simulated perturbation is applied to the horizontal edges of the degeneracy (see Lemma 15). Figure 6.12 shows all the possible combinations of coincident degenerate Voronoi vertices, when two vertex rays are implied, and the result of applying the simulated perturbation. Note that the computation of the Voronoi vertex coordinates are not affected by the perturbation.

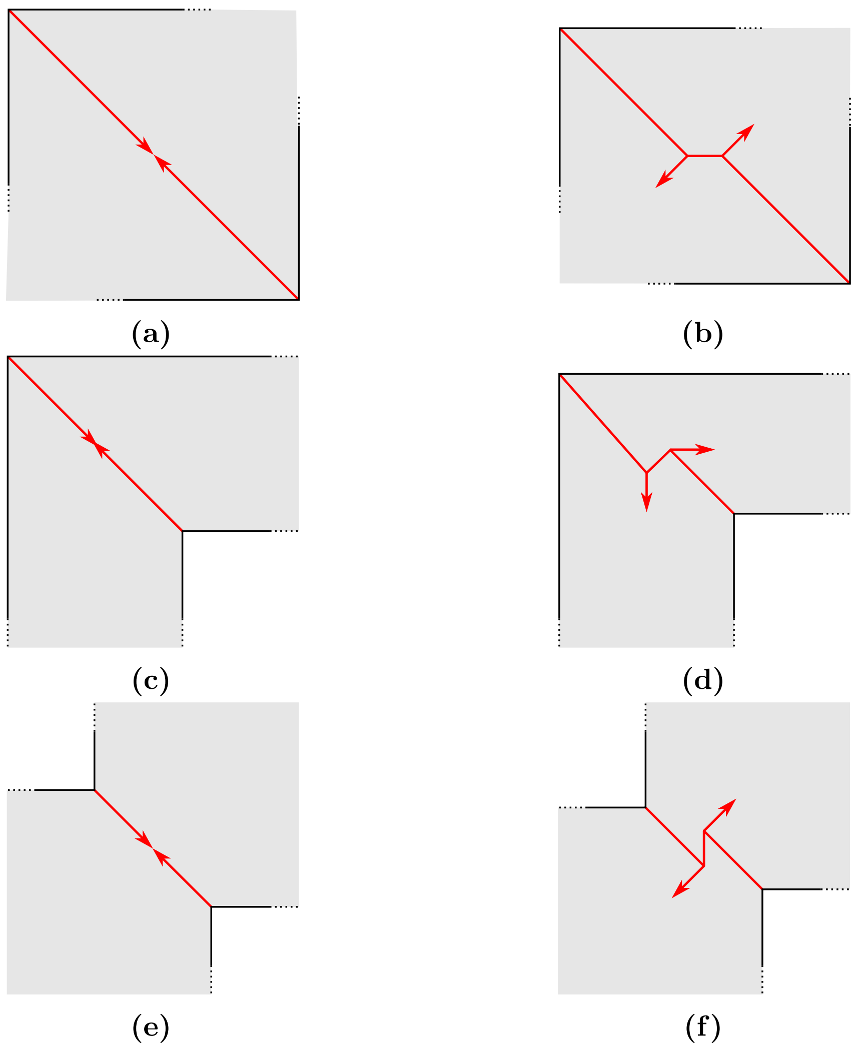


Figure 6.12: In the left column, there are coincident degenerate Voronoi vertices induced by four edges associated to two vertex rays. In the right column, there is an illustration of the horizontal edges perturbation, which result in two Voronoi vertices connected by a Voronoi edge.

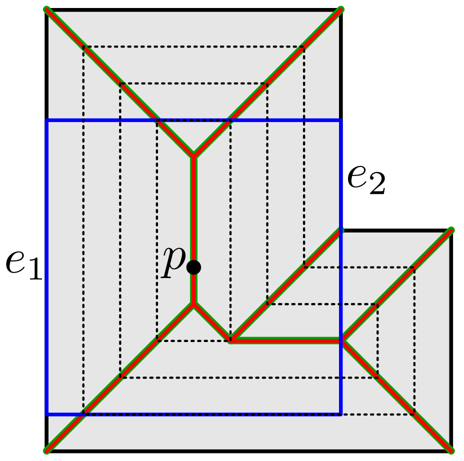
6.6 Straight skeleton equivalence

In this section, a detailed proof of the known equivalence between the Voronoi edges of $V_\infty(\mathcal{P})$ and the straight skeleton (see Section 3.1.5) of $\mathcal{P}$ is given.

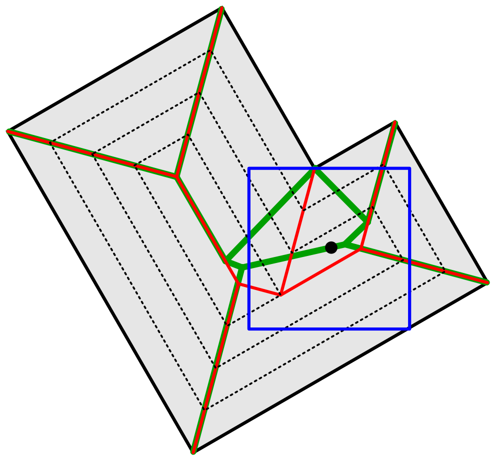
Lemma 16. *Let $\mathcal{P}$ be an orthogonal polygon without collinear edges. The Voronoi edges of $V_\infty(\mathcal{P})$ coincide with the straight skeleton of $\mathcal{P}$.*

Proof. Let p be a point of the Voronoi edges of $V_\infty(\mathcal{P})$ that is L_∞ equidistant to a set E of two or more non-collinear edges of $\mathcal{P}$ (see for example Figure 6.13a). First, as all the vertices of $\mathcal{P}$ have an angle of $\frac{\pi}{2}$ or $\frac{3\pi}{2}$, they move at equal speed during the straight skeleton wavefront propagation. Thus, the edges E are moved inwards $\mathcal{P}$, at equal speed, and parallel to themselves during the straight skeleton shrinking process. At any time of the propagation, p is L_∞ equidistant to the moving edges E, as they are axis-aligned. Observe that if the edges move at equal speed but are not axis-aligned, then $V_\infty(\mathcal{P})$ may not coincide with the straight skeleton (see for example Figure 6.13b). Suppose that at some time of the straight skeleton propagation, some edges of $\mathcal{P}$ and not from E are L_∞ equidistant to p. However, this would imply that these edges moved inwards at a higher velocity than the edges of E. Otherwise, they would belong to E. Then, the straight skeleton propagation of the edges E reach p at the same time and as a consequence p must belong to the straight skeleton of $\mathcal{P}$.

Observe that if $\mathcal{P}$ contains collinear edges, p could be L_∞ equidistant to an arbitrary amount of collinear edges, and as a consequence the straight skeleton of $\mathcal{P}$ could not coincide with $V_\infty(\mathcal{P})$. □



(a)



(b)

Figure 6.13: Illustration of Lemma 16. The straight skeleton is shown in red and the L_∞ Voronoi diagram is shown in green. The dotted lines illustrate the straight skeleton wavefront propagation. **(a)** The point p is L_∞ equidistant to e_1 and e_2 during the straight skeleton wavefront propagation. **(b)** The straight skeleton and the L_∞ Voronoi diagram may not coincide, although all the vertices of the straight skeleton wavefront propagate at an equal speed.

In the literature, some authors consider that the collinear edges of $\mathcal{P}$ may be considered to be equal or different, during the straight skeleton wavefront propagation. If they are considered to be equal, the Voronoi edges of $V_\infty(\mathcal{P})$ that do not bound a bisector area (see Section 6.5.2) coincide with the straight skeleton. The following lemma assumes the latter case.

Lemma 17. *The straight skeleton of an orthogonal polygon withouth collinear edges can be computed in $O(n \log n)$ time and $O(n)$ space, where n is the number of vertices of $\mathcal{P}$.*

Proof. The straight skeleton coincide with the edges of $V_\infty(\mathcal{P})$ (see Lemma 16). The edges of $V_\infty(\mathcal{P})$ can be computed in $O(n \log n)$ time and $O(n)$ space (see Lemmas 10 and 11). □

6.7 Experimental results

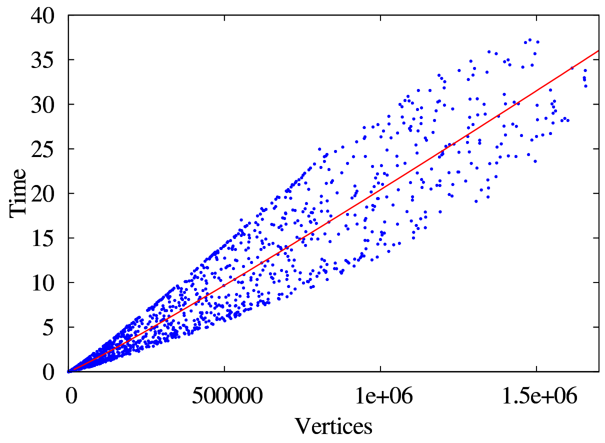
The presented specific implementations to compute the L_∞ Voronoi diagram and the straight skeleton of orthogonal polygons have been compared with an optimized thinning approach [30] that computes the straight skeleton of a binary image and the CGAL implementation of the straight skeleton of a polygon with holes [1]. Both algorithms have been implemented in C++ and the source code consists of about two thousand lines of code. Unfortunately, there is no publicly available software to compute the L_∞ Voronoi diagram of a polygon or line segments. The experiments were conducted with a database of around one thousand binary shapes and three thousand samples of randomly generated orthogonal manifold polygons with varying densities (see Figure 6.14). In addition, the presented approaches were tested with thousand datasets containing pseudomanifold vertices and floating point coordinates, in order to verify their robustness. The available memory was limited to 16GB and the time computation was at most twenty minutes for each dataset. The polygons have at most one and a half million vertices. The running times of the edge-based approach are almost equal to those of the vertex-based approach and are included in the same category. In order to be faithful with the comparison of the presented approach and the discrete approach of [30] the times of the presented Voronoi based approach also include the discretization of the Voronoi edges, which is almost negligible. Figure 6.15 shows the $V_\infty(\mathcal{P})$ of two simple datasets.

The CGAL straight skeleton computation has a theoretically quadratic complexity with respect to the number of vertices of the polygon (see Figure 6.14c). The average runtime was $0.037\, n^2\ \mu s$. Unfortunately, for polygons with more than seventeen thousand vertices the CGAL approach consumed all the avaliable RAM memory, thus slowing the computation process. For this reason these polygons have not been tested.

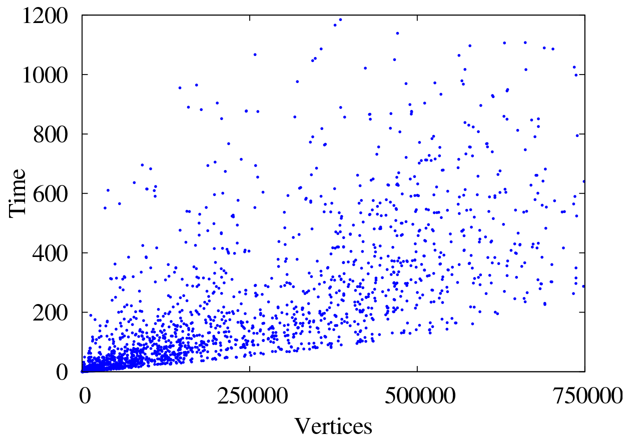
The CGAL straight skeleton computation using exact arithmetic also consumed all the available RAM memory for polygons with more than ten thousand vertices (see Figure 6.14d). Its average runtime was $11.5\, n^2\ \mu s$. The exact arithmetic dramatically dropped the performance by an average factor of three hundred compared with the inexact arithmetic approach.

The thinning method to compute the skeleton exceeded the twenty minutes limit of computation time for datasets with more than eight hundred thousand vertices. The time complexity of the thinning approach is bounded by the number of pixels of the binary image representing $\mathcal{P}$ and not by the number of vertices of $\mathcal{P}$. Thus, the time data shown in Figure 6.14b is not correlated with the number of vertices.

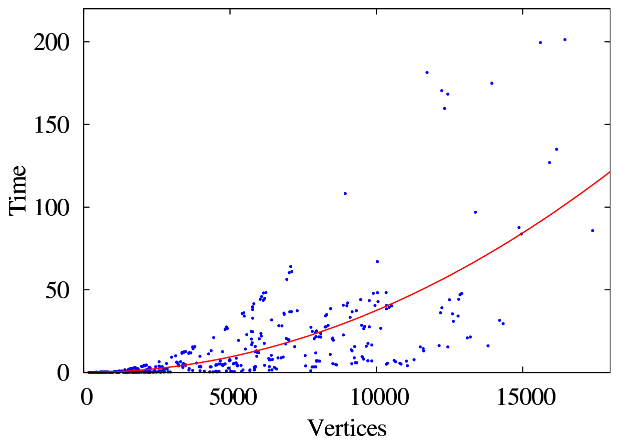
The presented implementations revealed an average runtime of $1.03\, n \log n\ \mu s$ and are an order of magnitude faster than the CGAL approaches. Surprisingly, the curve fitting of the runtimes of the presented approaches had the best asymptotic standard error when assuming that the algorithm has lineal time complexity compared with assuming linearithmic time complexity (see Figure 6.14a). However, it is expected that for datasets bigger than the tested ones the runtimes may exhibit a linearithmic complexity. As expected, the thinning approach was much slower than the presented approach because its time complexity depends on the size of the binary image, which grows fast.



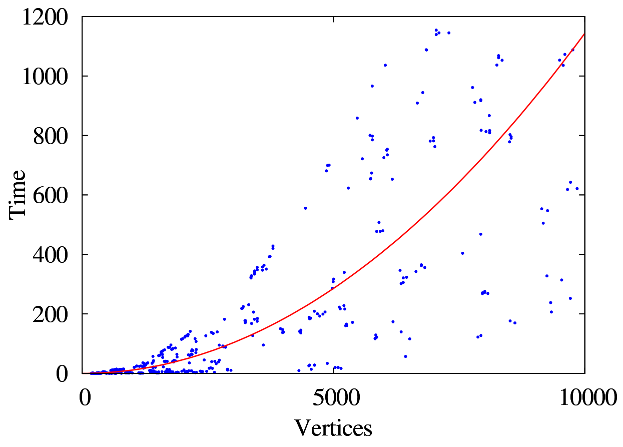
(a) Presented approach



(b) Approach of [30]

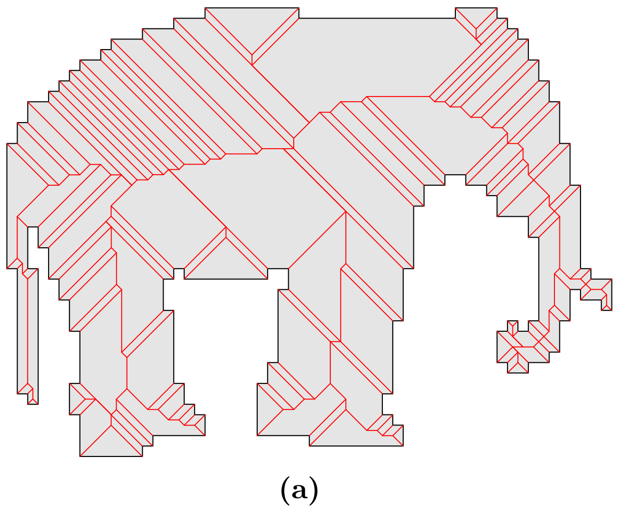


(c) CGAL



(d) CGAL, exact arithmetic

Figure 6.14: Running times of different methods to compute $V_\infty(\mathcal{P})$. The abscissa denotes the number of vertices of $\mathcal{P}$. The ordinate denotes the running time in seconds. Every blue point depicts the runtime for a single dataset. The red line corresponds to the curve fitted to the theoretical time complexity of each algorithm, with respect to the number of vertices of the polygon.


(a)

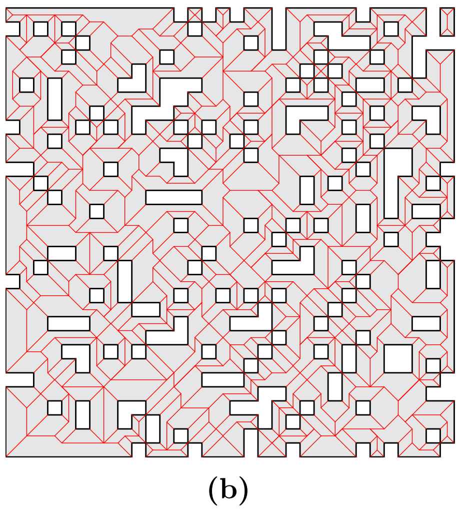

(b)

Figure 6.15: The edges of $V_\infty(\mathcal{P})$ are shown in red. Dataset **(b)** has been randomly generated.

6.8 Conclusions

Two practical specific implementations to compute the L_∞ Voronoi diagram and the straight skeleton of an orthogonal polygon have been presented. All the algorithms run in $O(n \log n)$ time and $O(n)$ space, where n is the number of vertices of the orthogonal polygon. Geometrical degeneracies and pseudomanifold orthogonal polygons are correctly treated. The equivalence of the straight skeleton and the L_∞ Voronoi diagram has been studied. In addition, experimental results over several datasets demonstrate the efficiency of the presented methods.

Chapter 7

L_∞ Voronoi diagram of orthogonal polyhedra

Contents

7.1 Introduction

Although the combinatorial complexity of the L_∞ Voronoi diagram of polyhedra ($O\left(n^{2+\epsilon}\right)$, for any $\epsilon > 0$) has been studied [140] , little has been done in relation to its computation. As with the two-dimensional case (see Chapter 6), the three-dimensional L_∞ Voronoi diagram may serve as a skeletal representation of orthogonal polyhedra. Thus, it is desirable to have a practical and robust algorithm to compute it.

In this chapter a sweep line algorithm to compute the L_∞ Voronoi diagram of an orthogonal polyhedron is presented. The two-dimensional vertex-based algorithm (see Chapter 6) is extended to three dimensions. The introduced method is proven theoretically and experimentally to be efficient and robust. In addition, a detailed proof of the equivalence between the L_∞ Voronoi diagram and the straight skeleton of an orthogonal polyhedron is given. This chapter is an extended and adapted version of the article [159].

7.2 Previous definitions and properties

In the following, some needed previous definitions and properties are introduced. Most of them are analogous to the ones provided for the two-dimensional case. In this chapter, $\mathcal{P}$ denotes an orthogonal polyhedron with only trihedral vertices (see Chapter 5). The Voronoi diagram of $\mathcal{P}$ under the L_∞ metric is denoted as $V_\infty(\mathcal{P})$ (see Definition 21). It is known that the straight skeleton of orthogonal polyhedra is a superset of the L_∞ Voronoi diagram [33]. Thus, it follows that the combinatorial complexity of $V_\infty(\mathcal{P})$ is $O(n^2)$ (see Section 3.1.5).

Lemma 18. *Four axis-aligned open half-spaces in $\mathbb{R}^3$ define a cube if and only if two of them are parallel, their intersection is not empty and the other two are aligned with the two remaining axis.*

Proof. It is clear that if three or more half-spaces are parallel they cannot define a cube. Consider, without loss of generality, the next open half-spaces, two of them parallel and the other two aligned with the two remaining axis:

$$h_1 : x > c_1 \ , \ h_2 : x < c_2 \ , \ h_3 : y > c_3 \ , \ h_4 : z > c_4 \qquad c_1 \neq c_2$$

The intersection of half-spaces h_1 and h_2 is non empty if $c_1 > c_2$. In this case, the length of the cube edges is $c_1 - c_2$. The coordinates of the cube vertices are directly obtained from the cube vertex(c_1, c_3, c_4) and the cube edges length. The center of the cube is $\left(\frac{c_1+c_2}{2}, c_3 + \frac{c_1-c_2}{2}, c_4 + \frac{c_1-c_2}{2}\right)$. □

Property 14. *Four faces of $\mathcal{P}$ define a cube if and only if their associated half-spaces oriented inside $\mathcal{P}$ define a cube (see Lemma 18) and the intersection between the cube boundary and all four faces is not empty.*

Definition 43. *A Voronoi vertex of $V_\infty(\mathcal{P})$ is* degenerate *if it is L_∞ equidistant to more than four faces of $\mathcal{P}$.*

Definition 44. *An orthogonal polyhedron $\mathcal{P}$ is in* general position *if it is an orthogonal manifold polyhedron, without coplanar faces and $V_\infty(\mathcal{P})$ has no degenerate Voronoi vertices.*

The orthogonal polyhedron $\mathcal{P}$ is considered to be in general position from Section 7.3 to 7.6. Section 7.8 deals with geometrical degeneracies.

7.3 Algorithm overview

The presented algorithm is based on the vertex-based sweep line algorithm (see Section 2) to compute the L_∞ Voronoi diagram of a polygon (see Section 6.4) and considers a sweep plane instead of a sweep line. The main idea of the method is to trace rays, corresponding to Voronoi edges, that progressively construct the Voronoi diagram. Let $\mathcal{L}$ be the sweep plane perpendicular to the z-axis ($z = t$). The sweep process is done by decreasing the value of t: by sweeping from top to bottom. The priority queue $\mathbb{E}$ stores the events found during the sweep. Like the two-dimensional case, the events are either vertices of $\mathcal{P}$ or candidate Voronoi vertices of $V_\infty(\mathcal{P})$. The status data structure $\mathbb{W}$, that maintains the wavefront, is an undirected graph.

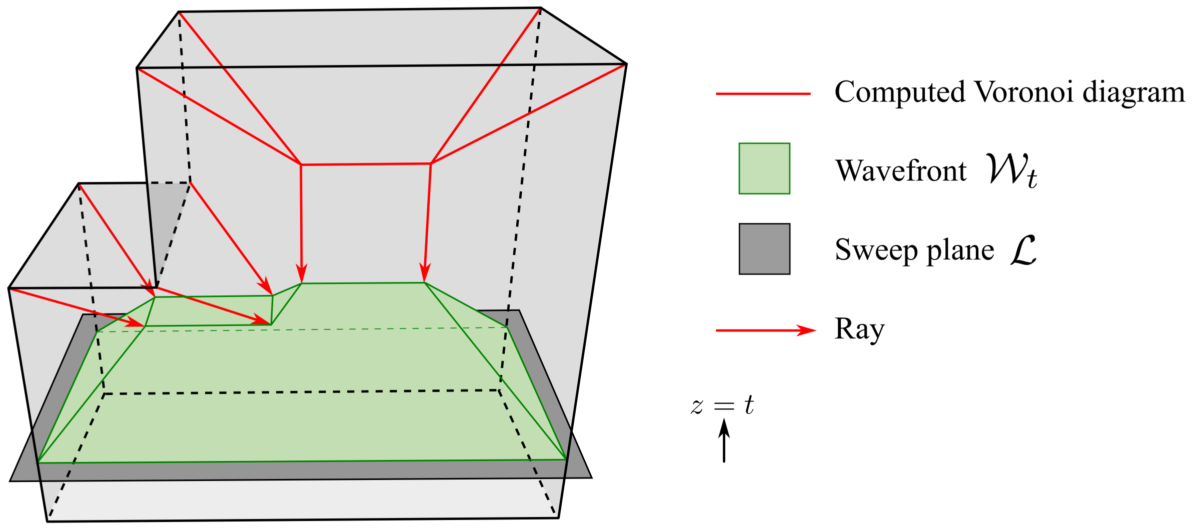


Figure 7.1: Scheme of the sweep plane algorithm to compute the L_∞ Voronoi diagram of an orthogonal polyhedron. The orthogonal polyhedron, drawn in gray, is in general position. The sweep plane $\mathcal{L}$ is considered to move down the z axis.

Definition 45. *At any moment t of the sweeping process, $\mathcal{L}$ partitions the faces of $\mathcal{P}$ into three subsets: A_t are the faces that lie above of $\mathcal{L}$, I_t are the faces that intersect $\mathcal{L}$, and B_t are the faces that lie below of $\mathcal{L}$. C_t is composed of the portions of the faces I_t that lie above $\mathcal{L}$. $\mathcal{L}_t$ is the set of faces of $\mathcal{P}$ contained in $\mathcal{L}$ and with the interior of $\mathcal{P}$ above, plus the portion of $\mathcal{L}$ that belongs to the interior of $\mathcal{P}$.*

Definition 46. *At any moment t of the sweeping process, the faces of $\mathcal{P}$ that lie above $\mathcal{L}$, that is $A_t \cup C_t$, are called $\mathcal{P}_t$.*

An invariant of the algorithm is that, at any moment t of the sweeping process, the L_∞ Voronoi diagram of $A_t \cup C_t \cup \mathcal{L}_t$, which is called $V_t(\mathcal{P})$, is already computed.

Definition 47. *At any moment t of the sweeping process, the* wavefront *$\mathcal{W}_t$ is the boundary of the Voronoi regions (see Definition 20) of $V_t(\mathcal{P})$ associated to the faces $\mathcal{L}_t$, and that are not $\mathcal{L}_t$ itself. The Voronoi faces induced by $\mathcal{L}_t$ are the* wavefront faces *and their Voronoi edges and vertices are the* wavefront edges *and* wavefront vertices*, respectively.*

The following lemma, related to the wavefront monotonicity, is analogous to the two-dimensional Lemma 9.

Lemma 19. *The wavefront faces are monotone with respect to the sweep plane.*

Proof. Let $p = (x, y, z)$ be a point in $\mathcal{L}_t$, in wavefront face f of $\mathcal{L}_t$, and let $p' = (x, y, z')$ be the point with maximal z' value in the Voronoi region of f. Thus, p' belongs to the wavefront. The interior of the cube centered at p' and touching p, do not intersect the boundary of $A_t \cup C_t \cup \mathcal{L}_t$. For every $x \in [p, p']$, the cube centered at x that touches p cannot intersect any face from $A_t \cup C_t \cup \mathcal{L}_t$, apart from the one that contains p. Thus, the maximal cube centered at p' is empty, and contains every cube centred at a point on the segment $[p, p']$, hence all of those are empty as well, and must belong to the Voronoi region of f. So, there cannot exist any point of the wavefront between p and p' and it follows that the wavefront faces are monotone with respect to the sweep plane. $\square$

Definition 48. *At any moment of the sweeping process, a* ray *is a half-line that supports a Voronoi edge, with its origin in a vertex of $\mathcal{P}$ or in a Voronoi vertex of $V_\infty(\mathcal{P})$. The end of the Voronoi edge remains to be calculated. Once calculated, the ray becomes a Voronoi edge of $V_\infty(\mathcal{P})$ (see Figure 7.1).*

In general position, a ray is contained in the L_∞ bisector of three faces f_1, f_2 and f_3 of $\mathcal{P}$ and is denoted by $\{f_1, f_2, f_3\}$.

Definition 49. *The ray emanating from a vertex of $\mathcal{P}$ is defined by its three incident faces and it is called* vertex ray.

Definition 50. *The intersection point between two rays $\{f_1, f_2, f_3\}$ and $\{f_1, f_2, f_4\}$ is the center of the cube defined by the faces f_1, f_2, f_3 and f_4 (see Property 14).*

Note that as $\mathcal{P}$ is in general position, the two intersecting rays $\{f_1, f_2, f_3\}$ and $\{f_1, f_2, f_4\}$ must share two faces, f_1 and f_2.

Definition 51. *The intersection point between a ray $\{f_1, f_2, f_3\}$ and a face f_4 of $\mathcal{P}$ is the center of the cube defined by the faces f_1, f_2, f_3 and f_4 (see Property 14).*

The next property determines the direction of a ray that emanates from a Voronoi vertex.

Property 15. *Let $\{f_1, f_2, f_3\}$ be a ray, with origin in the Voronoi vertex equidistant to the faces f_1, f_2, f_3 and f_4. If any edge bounded by the faces $f_1 f_4$ and $f_2 f_4$ and $f_3 f_4$ form a reflex angle, then the ray is oriented towards the interior of $\mathcal{P}$ defined by f_4. Otherwise, the ray is oriented in the opposite direction.*

For example, consider the ray $\{f_{y1}, f_{y2}, f_z\}$ of Figure 7.5, that emanates from the Voronoi vertex defined by the faces f_x, f_{y1}, f_{y2}, and f_z. As the edge bounded by the faces $f_x f_{y1}$ is reflex, the ray is inversely oriented towards the interior of $\mathcal{P}$ defined by f_x.

Definition 52. *At any moment t of the sweeping process, the set of rays that define a cube with $\mathcal{L}$ are called* wavefront rays *(see Figure 7.1).*

Property 16. *At any moment t of the sweeping process, the vertices of the wavefront are:*

- *The center of the cube defined by a wavefront ray and $\mathcal{L}$.*
- *The intersection point between an edge from a face of I_t, parallel to the z-axis, and $\mathcal{L}$.*

Observe that during the sweep process some rays that do not define a cube with $\mathcal{L}$ may appear (for example, the ray $\{f_{y1}, f_{y2}, f_{z1}\}$ of Figure 7.6b).

Definition 53. *The status data structure $\mathbb{W}$ is an undirected graph, whose vertices are the rays. An edge of this graph indicates that the corresponding rays share two faces.*

Note that the edges of the graph $\mathbb{W}$ are wavefront edges. The wavefront vertices defined by the intersection between an edge e of I_t and $\mathcal{L}$ (see Property 16) are implicitly defined by the wavefront ray induced by the faces bounding e and another face. For example, the ray $\{f_{x2}, f_{y1}, f_{y2}\}$ of Figure 7.6d explicitly defines a wavefront vertex that corresponds to the center of the cube defined with $\mathcal{L}$, and implicitly defines two wavefront vertices that correspond to the intersection between the edges, parallel to the z-axis and bounded by the faces $f_{x2}f_{y1}$ and $f_{x2}f_{y2}$, and $\mathcal{L}$.

Lemma 20. *$\mathbb{W}$ is a planar graph.*

Proof. As a consequence of Lemma 19, the wavefront vertices can be projected in the sweep plane, and it follows that $\mathbb{W}$ is a planar graph. □

Lemma 21. *A vertex of $\mathbb{W}$ has at most three incident edges.*

Proof. Two rays are connected by an edge through two common faces. As a ray is defined by three faces, there can be at most three different pairs of faces that define three edges of $\mathbb{W}$. □

In fact, $\mathbb{W}$ is a *bounded degree graph*, a graph where no vertex has more than k neighbors. There exist algorithms to test efficiently some properties of bounded degree graphs [109].

Lemma 22. *$\mathbb{W}$ has at most $O(n)$ vertices, where n is the number of vertices of $\mathcal{P}$.*

Proof. At any moment t of the sweeping process, consider a single face f_t of $\mathcal{L}_t$, which induces part or all the wavefront. The Voronoi region of f_t is bounded by at most $O(n)$ planes, which correspond to the L_∞ bisectors between f_t and the rest of faces of $\mathcal{P}_t$ (see Definition 46). Thus, the Voronoi region of f_t has at most $O(n)$ vertices and it follows that $\mathbb{W}$ also has at most $O(n)$ vertices. □

$\mathbb{W}$ is stored in a dynamic data structure for planar graphs [214], which uses balanced search trees. The insertion and deletion of vertices and edges on that dynamic data structure can be done in $O(\log n)$ time, where n is the number of graph vertices. Thus, the next property follows from Lemma 22.

Property 17. *The insertion and deletion of vertices and edges of* $\mathbb{W}$ *can be done in* $O(\log n)$ *time, where* n *is the number of vertices of* $\mathbb{W}$.

At any moment of the sweeping process, if two rays are defined by the same three faces, a Voronoi edge connecting their origins is added to $V_\infty(\mathcal{P})$ and, if necessary, both rays are deleted from $\mathbb{W}$. For example, the ray $\{f_{y1}, f_{y2}, f_{z1}\}$ of Figure 7.6b will be deleted from $\mathbb{W}$ when another ray, defined by the same three faces and origin in the opposite endpoint of the Voronoi edge containing the ray, will be created.

The wavefront faces can be obtained by traversing edge loops of wavefront rays from $\mathbb{W}$ induced by vertices with a common face. Note that $\mathbb{W}$ describes the combinatorial structure of the wavefront. The wavefront data structure only needs to be updated each time an event occurs. Two different types of events may occur during the sweep:

- **Vertex event**: the occurrence of a trihedral vertex of $\mathcal{P}$. The priority of a vertex event is defined by the vertex coordinate.
- **Junction event**: the intersection between two rays or a ray and a face that could induce a new Voronoi vertex. The priority of a junction event is given by the vertex of the junction event cube that has the lowest z, y and x coordinate, in this order. That is, as soon as the intersection is reached by the wavefront and not by the sweeping plane. There exist two types of junction events:
  - **Ray-ray junction event**: the intersection of two rays.
  - **Ray-face junction event**: the intersection of a ray and a face of $\mathcal{P}$.

The events of $\mathbb{E}$ are sorted according to its priority in zyx-lexicographical order. If two junction events have the same priority, the junction event whose cube has the bigger side length is processed first. In addition, if the z coordinate priority of a vertex event and a junction event coincide, the vertex event is processed first. Initially, $\mathbb{E}$ is the sorted set of vertex events induced by the vertices of $\mathcal{P}$. New events may be inserted in $\mathbb{E}$ each time an event is processed. The events are processed until $\mathbb{E}$ is empty. Every time a junction event is processed it must be checked if it is a Voronoi vertex. If not, the event is rejected. Observe that the event processing is analogous to the two-dimensional vertex based algorithm (see Section 6.4).

As the sweep plane moves below, the L_∞ distance from $\mathcal{L}$ and the set of faces P_t above $\mathcal{L}$ cannot decrease because the L_∞ distance is convex. Therefore, the following property holds.

Property 18. *For any* $t > t'$, *and for any face* $f \in P_t$, $Vor_\infty(f, P_t) \subseteq Vor_\infty(f, P_{t'})$.

The correctness of the presented algorithm is due to the following lemma.

Lemma 23. *The wavefront can only change its structure when an event occurs.*

Proof. First, suppose that no event occurs within a time interval and a wavefront face is removed from the wavefront. The wavefront faces move along their rays. Since the rays do not cross or do not intersect other faces, as junction events do not occur, it follows that no wavefront face can be removed from the wavefront.

Second, suppose that at time t in the interval $t' > t > t''$, a new wavefront face appears in the wavefront. Observe that this new wavefront face must be associated to a face of $P_{t'}$, otherwise a vertex event should have occured during the time interval. Due to Property 18, f has already contributed one or more wavefront faces to $W_{t'}$ and none of them has yet been removed, as their respective Voronoi regions cannot decrease in size. For each wavefront face of W_t associated to f, consider an arc that connects it with f. Since the arc is contained in the Voronoi region of f, it must pass through a wavefront face of $W_{t'}$. As $W_{t'}$ contains fewer wavefront faces associated to f than to W_t, there must be two arcs leading through the same wavefront face of $W_{t'}$. Thus, a wavefront face of W_t that is neighboring any of the wavefront faces associated to f, may be separated from its respective Voronoi region. This can only happens if new polyhedron faces that do not belong to $P_{t'}$ appear, which is a contradiction. It follows that the topological structure of the wavefront cannot change before time t''. □

The Voronoi edges and vertices of $V_\infty(\mathcal{P})$ are stored in a static undirected graph [71]. Every Voronoi vertex is L_∞ equidistant to four faces of $\mathcal{P}$ or, in the case of the vertices of $\mathcal{P}$, to three faces. A Voronoi edge is defined by the three faces shared by its two endpoint vertices. A Voronoi face can be obtained by traversing the edge loops of the graph that share a common Voronoi face.

In the following two sections, it is described how to process each event. Next, the computation of new junction events is discussed in Section 7.6. Section 7.8 presents a technique to cope with geometric degeneracies.

7.4 Vertex event

Thanks to the trihedralization step presented in Chapter 5, the vertex event processing becomes much simpler because only one vertex ray emanates from any vertex of $\mathcal{P}$.

Foremost, the vertex ray is inserted in $\mathbb{W}$. Then, consider that the incident face of the vertex, perpendicular to the z-axis, has the interior of $\mathcal{P}$ above. In this case, the vertex ray must intersect a wavefront face. Otherwise, the vertex ray will become a wavefront ray once the sweep plane is moved down.

The vertex ray is connected with its two adjacent vertex rays with the same z coordinate, if they are present, in $\mathbb{W}$. For example, Figure 7.6a shows the status of $\mathbb{W}$ after processing the four vertex events contained in the upper face, perpendicular to the z-axis.

In addition, consider the edge e_z of $\mathcal{P}$, incident to the vertex and parallel to the z-axis. The vertex ray may be connected with the wavefront ray of $\mathbb{W}$ defined by the two faces of $\mathcal{P}$ that bound the edge e_z and another face. For example, the vertex event induced by the ray $\{f_{x2}, f_{y2}, f_{z2}\}$ of Figure 7.6d is connected to the wavefront ray $\{f_{x2}, f_{y1}, f_{y2}\}$, because it is defined by the two faces bounding the incident edge of the vertex, parallel to the z-axis, and an additional third face.

Lemma 24. *The vertex event update of* $\mathbb{W}$ *can be done in* $O(\log n)$ *time, where* n *is the number of vertices of* $\mathbb{W}$.

Proof. The connection with the two adjacent vertex rays that have the same z coordinate can be done in $O(\log n)$ time (see Property 17). In addition, consider that for each edge e_z of $\mathcal{P}$, parallel to the z-axis, it is stored a pointer to the last wavefront ray, if it exists, that contains the faces bounding e_z. Observe that, at any moment of the sweeping process only one wavefront ray that can be associated to e_z is stored. Thus, the vertex ray can be also connected in $O(\log n)$ time. □

Finally, the nearest junction event induced by the vertex ray is computed (see Section 7.6) and inserted in $\mathbb{E}$. See Algorithm 9 for an overview of the vertex event processing.

Algorithm 9 Vertex event processing algorithm.

```
*Update* $\mathbb{W}$ by inserting the vertex ray
*Insert* the nearest junction event induced by the vertex ray in $\mathbb{E}$
```

7.5 Junction event

A junction event corresponds to the center of the cube defined by four faces of $\mathcal{P}$ (see Property 14) and may define a Voronoi vertex of $V_\infty(\mathcal{P})$. The priority of a junction event is given by the vertex of the defined cube that has the lowest priority according to the zyx-lexicographical order. A junction event may be invalidated before its priority point is reached, because new faces of $\mathcal{L}$ may induce new junction events that are closer. Next, it is described how to process each type of junction event.

7.5.1 Ray-ray junction event

A ray-ray junction event corresponds to the intersection between two rays (see Definition 50). A ray-ray junction event defines a Voronoi vertex if both rays belong to $\mathbb{W}$ and are connected in $\mathbb{W}$. If the ray-ray junction defines a Voronoi vertex, the two rays are deleted from $\mathbb{W}$ and the two Voronoi edges incident to the Voronoi vertex are inserted in $V_\infty(\mathcal{P})$. In addition, two new rays defined by the two different faces between the intersecting rays and each common face between them are inserted in $\mathbb{W}$ (see Figure 7.2 for an example).

The connectivity of every ray deleted or inserted in $\mathbb{W}$, during the processing of the ray-ray junction event, must be updated. The neighboring rays in $\mathbb{W}$ of the deleted rays are retrieved and connected, if necessary, with the new rays. Two rays are connected if they share two faces. For example, in Figure 7.6b the ray-ray junction event induced by the rays $\{f_{x2}, f_{y1}, f_{z1}\}$ and $\{f_{x2}, f_{y2}, f_{z1}\}$ creates two new rays, one that does not intersect $\mathcal{L}$ and the other a wavefront ray, that are connected with the neighboring rays in $\mathbb{W}$ with two common faces. In Figure 7.6c, three rays are deleted from $\mathbb{W}$ after processing the ray-ray junction event induced by the rays $\{f_{x1}, f_{y1}, f_{z1}\}$ and $\{f_{x1}, f_{y2}, f_{z1}\}$. The connectivity of the two remaining wavefront rays in $\mathbb{W}$ is updated.

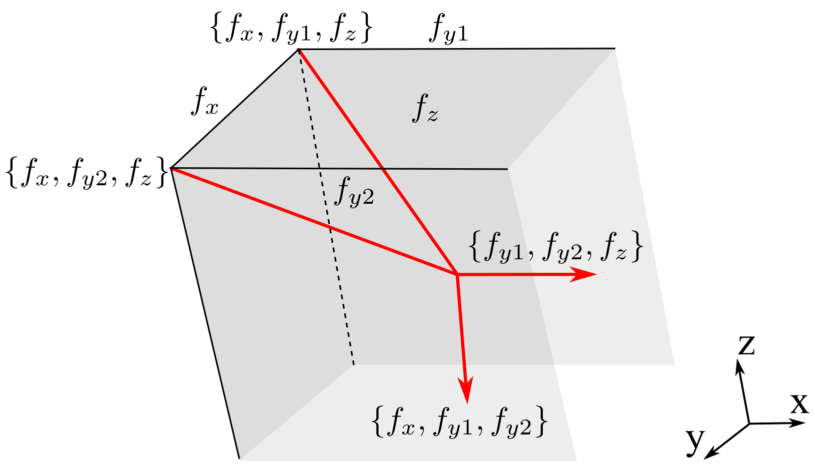


Figure 7.2: Ray-ray junction event. The rays $\{f_x, f_{y1}, f_z\}$ and $\{f_x, f_{y2}, f_z\}$ intersect and define a cube whose center is a Voronoi vertex. As a result, two new rays $\{f_x, f_{y1}, f_{y2}\}$ and $\{f_{y1}, f_{y2}, f_z\}$ are created.

Lemma 25. *The ray-ray junction event update of* $\mathbb{W}$ *can be done in* $O(\log n)$ *time, where* n *is the number of vertices of* $\mathbb{W}$.

Proof. The graph vertices and edges associated to the set of deleted rays are removed from $\mathbb{W}$ in $O(\log n)$ time (see Property 17). The wavefront graph connectivity of the new rays inserted must be updated. A ray-ray event may create at most two new rays that have at most six neighboring rays (see Lemma 21). Thus, only a constant number of possible connections between graph vertices must be checked. □

Finally, the nearest junction events induced by both new rays are computed (see Section 7.6) and inserted in $\mathbb{E}$. See Algorithm 10 for an overview of the vertex event processing.

Algorithm 10 Ray-ray junction event processing algorithm.

```
Let $\{f_1, f_2, f_3\}$ and $\{f_1, f_2, f_4\}$ be two intersecting rays
if both rays are in $\mathbb{W}$ and connected then
    Delete both intersecting rays from $\mathbb{W}$
    Insert the edges connecting the Voronoi vertex and both rays origin in $V_\infty(\mathcal{P})$
    Insert the rays $\{f_1, f_3, f_4\}$ and $\{f_2, f_3, f_4\}$ in $\mathbb{W}$
    Update connectivity of inserted and deleted rays in $\mathbb{W}$
    Insert the nearest junction event induced by both new rays in $\mathbb{E}$
end if
```

Unalterable ray-ray junction event

In some cases, a ray-ray junction event does not involve any change of $\mathbb{W}$. These junction events are called *unalterable ray-ray junction events.* The *well-formed* property of rays is introduced in order to characterize them.

Definition 54. *A point contained in a ray defines a cube if there exists a cube centered in the point whose boundary touches the three faces defining the ray.*

Definition 55. *A ray is* well-formed *if there exists a non empty segment s, with origin in the ray origin and contained in the ray, such that all the points of s define a cube with the ray.*

Definition 56. *An* unalterable ray-ray junction event *is a ray-ray junction event that will create at least one new ray that is not well-formed.*

Observe that unalterable ray-ray junction events may even appear in general position (see Figure 7.3 and 7.4) and can be detected by locally analyzing if each of the two new rays are well-formed. An unalterable ray-ray junction is processed by placing the origin of both rays inducing the junction event on the new Voronoi vertex. In addition, the nearest junction of both junction event rays needs to be recalculated (see Section 7.6) and inserted in $\mathbb{E}$. See Algorithm 11 for an overview of the unalterable ray-ray junction event processing.

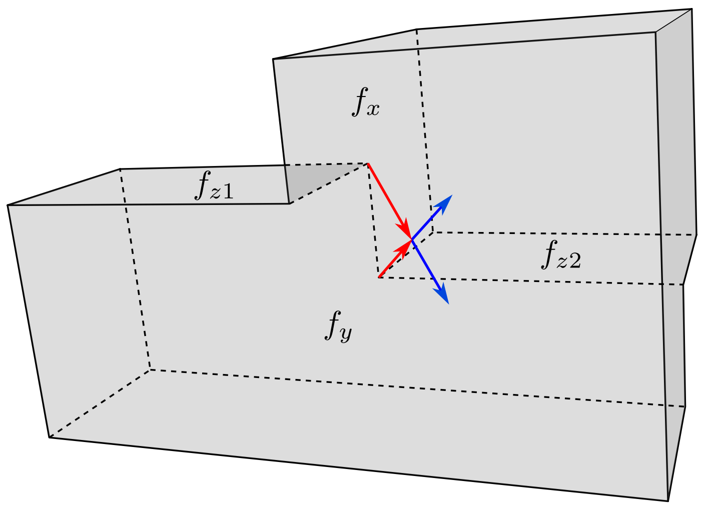


Figure 7.3: The ray-ray junction event induced by the red rays $\{f_x, f_y, f_{z1}\}$ and $\{f_x, f_y, f_{z2}\}$ is unalterable. The rays $\{f_y, f_{z1}, f_{z2}\}$ and $\{f_x, f_{z1}, f_{z2}\}$ are not well-formed since either the face f_{x1} or the face f_{x2} do not intersect any cube contained in both rays, except for the rays origin. Thus, the outgoing blue rays, after processing the junction event, are the original junction event rays with its origin changed to the new Voronoi vertex.

Algorithm 11 Unalterable ray-ray junction event processing algorithm.

```
Insert the edges connecting the Voronoi vertex and the origin of each ray in $V_\infty(\mathcal{P})$
Change the origin of both rays to the Voronoi vertex
Insert the nearest junction event of both rays in $\mathbb{E}$
```

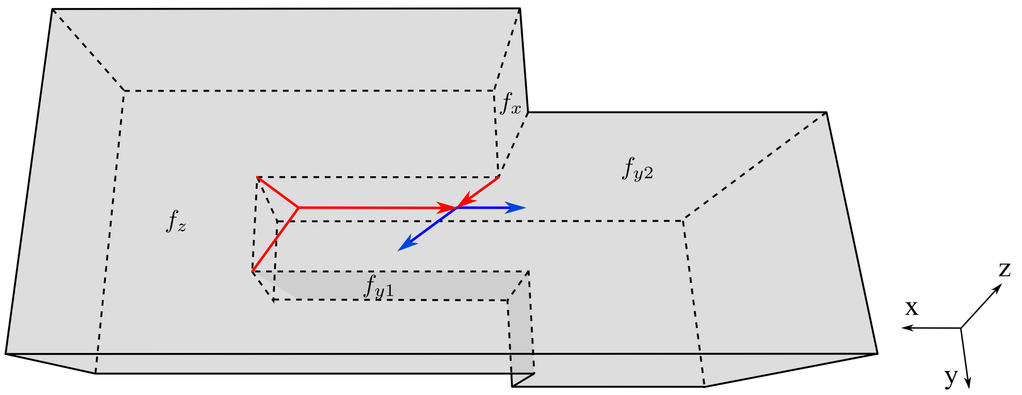


Figure 7.4: The ray-ray junction event induced by the red rays $\{f_{y1}, f_{y2}, f_z\}$ and $\{f_x, f_{y1}, f_z\}$ is unalterable. The ray $\{f_x, f_{y1}, f_{y2}\}$ is not well-formed since either the face f_x or the face f_{y2} do not intersect any cube contained in the ray. The outgoing rays are shown in blue.

7.5.2 Ray-face junction event

A ray-face junction event corresponds to the intersection between a ray and a face (see Definition 51). A ray-face junction event defines a Voronoi vertex if the ray belongs to $\mathbb{W}$. If the ray-face junction defines a Voronoi vertex, the ray is deleted from $\mathbb{W}$ and the incident Voronoi edge to the Voronoi vertex is inserted in $V_\infty(\mathcal{P})$. In addition, three new rays defined by each different pair faces from the intersecting ray and the intersected face are inserted in $\mathbb{W}$ (see Figure 7.5).

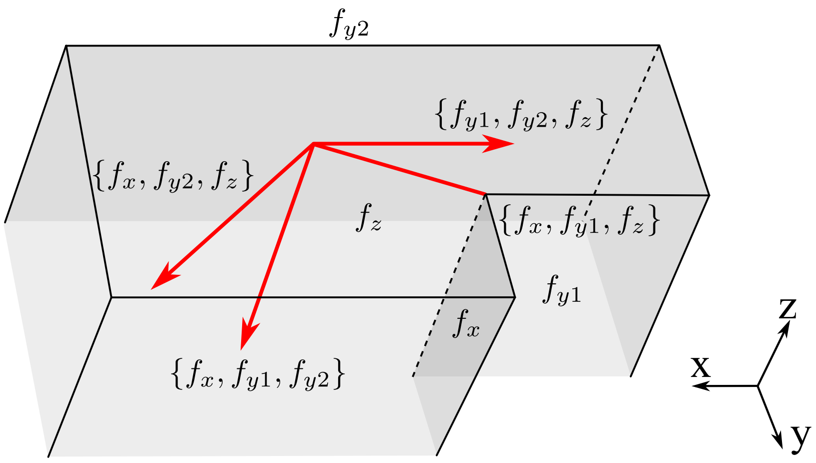


Figure 7.5: Ray-face junction event. The ray $\{f_x, f_{y1}, f_z\}$ intersects the face f_{y2} and define a cube whose center is a Voronoi vertex. Three new rays $\{f_x, f_{y1}, f_{y2}\}$, $\{f_x, f_{y2}, f_z\}$ and $\{f_{y1}, f_{y2}, f_z\}$ are created.

Similarly to the ray-ray junction event, the connectivity of every deleted and inserted ray in $\mathbb{W}$, during the processing of the ray-face junction event, must be updated. The neighboring rays in $\mathbb{W}$ of the deleted rays are retrieved and they are connected, if necessary, with the new junction event rays.

In addition, some wavefront edges of $\mathbb{W}$ may be split by the new rays. Let $\{f_1, f_2, f_3\}$ be a new ray, with origin in the junction event defined by the faces $F = \{f_1, f_2, f_3, f_4\}$. If the pair of faces (f_i, f_4), with $i \in \{1 \dots 3\}$, bounds a reflex edge of $\mathcal{P}$ and the pair of faces $F \setminus (f_i, f_4)$, corresponds to an existing wavefront edge containing the ray origin, then the new ray may split the wavefront edge into two wavefront edges. For example, consider the ray-face event of Figure 7.7b induced by the ray $\{f_{x2}, f_{y2}, f_z\}$ and the face f_{y3}. The new rays $\{f_{y2}, f_{y3}, f_z\}$ and $\{f_{x2}, f_{y3}, f_z\}$ split the wavefront edge, defined by the faces f_{y2} and f_{y3}, into two separate wavefront edges.

Lemma 26. *The ray-face junction event update of* $\mathbb{W}$ *can be done in* $O(\log n)$ *time, where* n *is the number of vertices of* $\mathbb{W}$.

Proof. As seen in Lemma 25, the connectivity of new rays inserted in $\mathbb{W}$ can be updated in $O(\log n)$ time. In addition, consider that every ray-face junction stores a pointer to each wavefront edge, if they exist, that may be split. At most, three pointers are required. Thus, the wavefront edge split can also be done in $O(\log n)$ time. □

Finally, the nearest junction events induced by the three new rays are computed (see Section 7.6) and inserted in $\mathbb{E}$. See Algorithm 12 for an overview of the vertex event processing.

Algorithm 12 Ray-face junction event processing algorithm.

```
Let $\{f_1, f_2, f_3\}$ be the ray and $f_4$ the intersected face
if the intersecting ray is in $\mathbb{W}$ then
    Delete the intersecting ray from $\mathbb{W}$
    Insert the edges connecting the Voronoi vertex and the ray origin in $V_\infty(\mathcal{P})$
    Insert the rays $\{f_1, f_2, f_4\}$, $\{f_2, f_3, f_4\}$ and $\{f_1, f_3, f_4\}$ in $\mathbb{W}$
    Update connectivity of inserted and deleted rays in $\mathbb{W}$
    Update split wavefront edges of $\mathbb{W}$
    Insert Compute the nearest junction event induced by three new rays in $\mathbb{E}$
end if
```

7.6 Nearest junction event computation

A fundamental step of the presented algorithms is the computation of the nearest junction event of a ray. Each time a new ray is created or after processing an unalterable ray-ray junction event, the nearest junction event of a ray must be inserted in $\mathbb{E}$.

Lemma 27. *The computation of the nearest ray-ray junction event can be done in* $O(1)$ *time.*

Proof. The total number of feasible ray-ray junctions of a ray is bound by the number of neighboring rays in $\mathbb{W}$. There are at most three neighbors (see Lemma 21), so, at most, three possible ray-ray junctions must be checked. □

For example, consider the ray $\{f_{x2}, f_{y3}, f_z\}$ of Figure 7.7b. The ray-ray junction events induced by its neighbors rays, $\{f_{x2}, f_{y1}, f_z\}$ and $\{f_{x1}, f_{y3}, f_z\}$, in $\mathbb{W}$ may be the nearest junction events.

Clearly, the most time consuming step of the nearest ray junction computation is the detection of ray-face junctions.

Property 19. *A ray may induce a ray-face junction event only if any pair of faces defining the ray form a reflex edge of $\mathcal{P}$.*

Lemma 28. *The computation of the nearest ray-face junction event can be done in $O(n)$ time, where n is the number of vertices of $\mathcal{P}$.*

Proof. Observe that $\mathbb{W}$ has linear size (see Lemma 22). First, consider that the incident face of the vertex ray, perpendicular to the z-axis, has the interior of $\mathcal{P}$ above. In this case, the wavefront face of $\mathbb{W}$ that is being intersected by the ray is found. This wavefront face can be found straightforwardly in linear time by checking all the wavefront faces. Observe that this ray-face junction event is always a Voronoi vertex.

Otherwise, the ray may intersect some wavefront edge. More precisely, let f be a face defining the ray. If f corresponds to a wavefront face, the ray may intersect a wavefront edge of f bound by the faces $\{f, f_i\}$. Observe, that this wavefront edge intersection is equivalent to the intersection between the ray and the face f_i. Thus, at most a linear number of wavefront edges may be checked. $\square$

An example of the computation of the nearest ray-face junction event induced by a vertex ray, such that its incident face is perpendicular to the z-axis, and has the interior of $\mathcal{P}$ above, can be seen in Figure 7.8. As $\mathcal{P}$ is in general position, the vertex ray induced by the vertex v must intersect a single wavefront face. This intersection is found by checking all the wavefront faces.

An example of the computation of the ray-face junction event induced by a ray, that is not a vertex ray with an incident face, perpendicular to the z-axis, and having the interior of $\mathcal{P}$ above, can be seen in Figure 7.7a. The vertex ray $\{f_{x2}, f_{y2}, f_z\}$ intersects the wavefront edge defined by the faces f_z and f_{y3}. In this case, the intersection is found, if it exists, by checking all the wavefront edges of the wavefront face of f_z.

7.7 Algorithm complexity

Lemma 29. *The $V_\infty(\mathcal{P})$ can be computed in $O(kn)$ time, where k is the number of events processed by the algorithm and n the number of vertices of $\mathcal{P}$.*

Proof. Each time an event is processed, the wavefront update is done in logarithmic time (see Lemmas 24, 25 and 26). The nearest ray-ray junction computation is done in constant time (see Lemma 27) and the nearest ray-face junction computation in linear time (see Lemma 28) with respect to n. Thus, the overall time complexity is $O(kn)$. $\square$

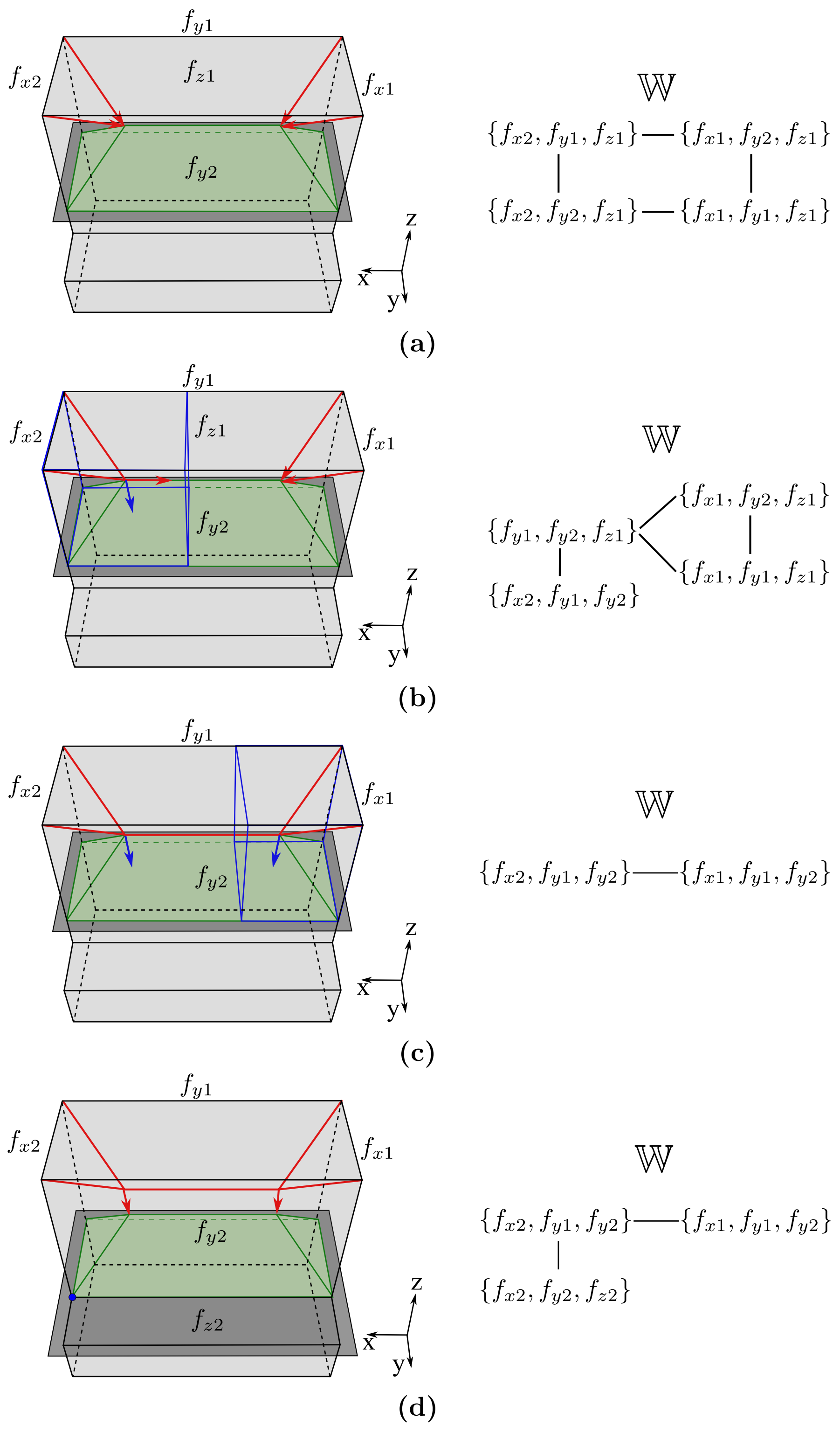
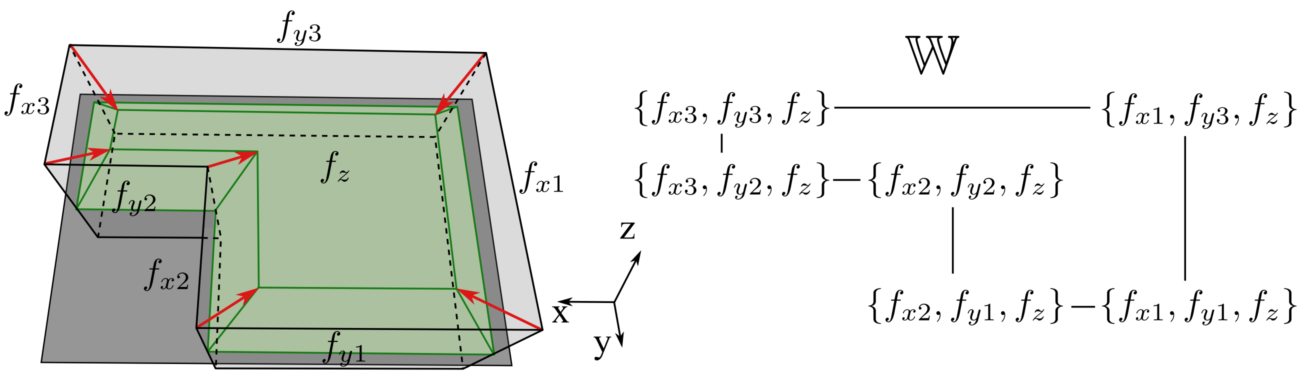
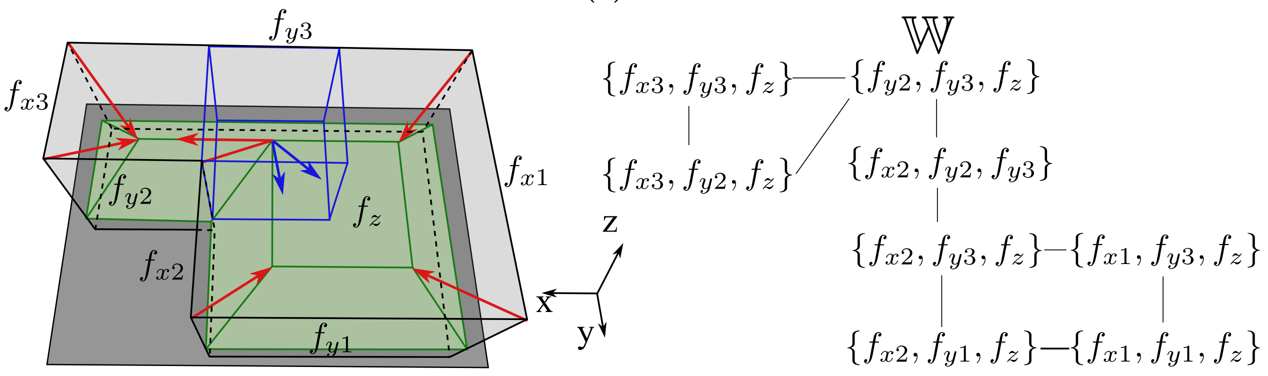

f_{y1}
f_{x2}
f_{z1}
f_{x1}
f_{y2}
$\mathbb{W}$
$\{f_{x2}, f_{y1}, f_{z1}\}$ — $\{f_{x1}, f_{y2}, f_{z1}\}$
$\{f_{x2}, f_{y2}, f_{z1}\}$ — $\{f_{x1}, f_{y1}, f_{z1}\}$
z
x
y
(a)
$\{f_{y1}, f_{y2}, f_{z1}\}$
$\{f_{x1}, f_{y2}, f_{z1}\}$
$\{f_{x1}, f_{y1}, f_{z1}\}$
$\{f_{x2}, f_{y1}, f_{y2}\}$
(b)
$\{f_{x2}, f_{y1}, f_{y2}\}$ — $\{f_{x1}, f_{y1}, f_{y2}\}$
(c)
f_{z2}
$\{f_{x2}, f_{y1}, f_{y2}\}$ — $\{f_{x1}, f_{y1}, f_{y2}\}$
$\{f_{x2}, f_{y2}, f_{z2}\}$
(d)

Figure 7.6: Illustration of the processing of vertex and ray-ray junction events. Status of $\mathbb{W}$ shown to the right. The blue arrows are wavefront rays, that currently define a zero length Voronoi edge, shown for illustrative purposes.



f_{y3}
f_{x3}
f_z
f_{x1}
f_{y2}
f_{x2}
f_{y1}
z
x
y
$\mathbb{W}$
$\{f_{x3}, f_{y3}, f_z\}$
$\{f_{x1}, f_{y3}, f_z\}$
$\{f_{x3}, f_{y2}, f_z\}$
$\{f_{x2}, f_{y2}, f_z\}$
$\{f_{x2}, f_{y1}, f_z\}$
$\{f_{x1}, f_{y1}, f_z\}$

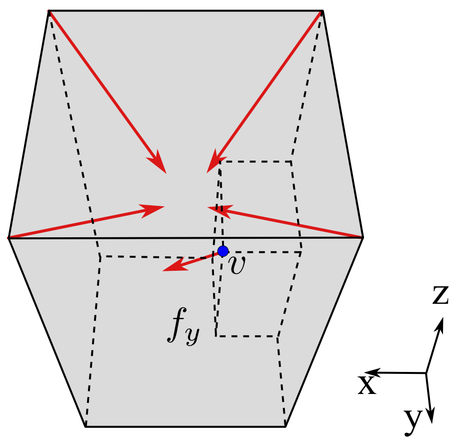
(a)



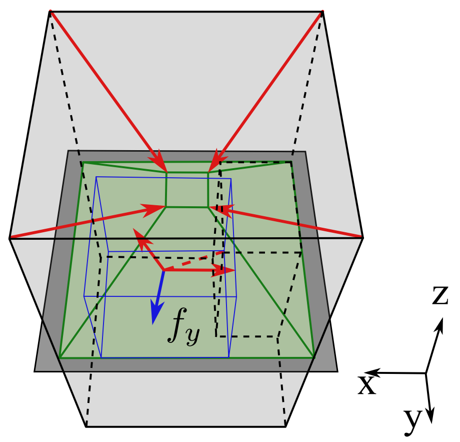
f_{y3}
f_{x3}
f_{x1}
f_{y2}
f_z
f_{x2}
f_{y1}
z
x
y
$\mathbb{W}$
$\{f_{x3}, f_{y3}, f_z\}$
$\{f_{y2}, f_{y3}, f_z\}$
$\{f_{x3}, f_{y2}, f_z\}$
$\{f_{x2}, f_{y2}, f_{y3}\}$
$\{f_{x2}, f_{y3}, f_z\}$
$\{f_{x1}, f_{y3}, f_z\}$
$\{f_{x2}, f_{y1}, f_z\}$
$\{f_{x1}, f_{y1}, f_z\}$

(b)

Figure 7.7: Illustration of the processing of a ray-face junction event, where a ray intersects a wavefront edge. Status of $\mathbb{W}$ shown at the right.



(a)



(b)

Figure 7.8: Illustration of the processing of a ray-face junction event, where a ray intersects a wavefront face. **(a)** The vertex ray induced by the vertex v must intersect a wavefront face. **(b)** After processing the ray-face junction event induced by the vertex ray of v, that intersects the wavefront face f_y, three new rays are created.

7.8 Treatment of degenerate cases

In this section, $\mathcal{P}$ could not be in general position. Geometrical degeneracies commonly arise in the case of orthogonal polyhedra, especially in the case of orthogonal polyhedra obtained from voxel representations. A robust algorithm must consider all of them.

Lemma 30. *Let $\mathcal{P}$ be an orthogonal polyhedron, ant let p be a point in the interior of $\mathcal{P}$. At most twenty-four Voronoi faces of $V_\infty(\mathcal{P})$ converge to p.*

Proof. The Voronoi edges of $V_\infty(\mathcal{P})$ are axis-aligned or rotated $\frac{\pi}{4}$ radians along one or two axes. Each Voronoi edge may be L_∞ equidistant to three different faces of $\mathcal{P}$. Thus, there are at most twenty-four Voronoi faces converging to p (see Figure 7.9). $\square$

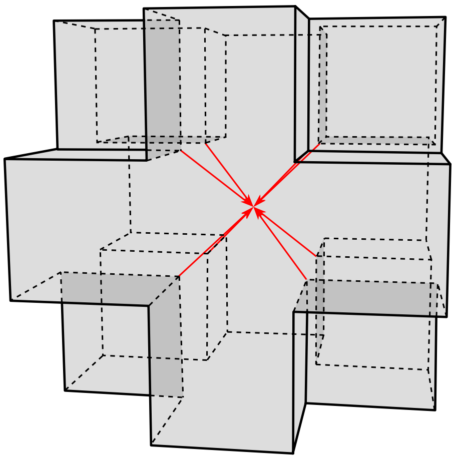


Figure 7.9: Worst-case degenerate Voronoi vertex of $V_\infty(\mathcal{P})$ induced by twenty-four faces of $\mathcal{P}$.

Observe that under the Euclidean metric, an arbitrary amount of Voronoi faces may converge to a Voronoi vertex of a polyhedron. Thus, a more simple and robust approach can be designed for the case of orthogonal polyhedra and L_∞ metric. Two types of geometrical degeneracies are distinguished:

- **Degeneracy by coplanarity**: a three-dimensional volume containing points that are L_∞ equidistant to two or more coplanar faces.
- **Degeneracy by coincidence**: there exists a Voronoi vertex L_∞ equidistant to more than four non-collinear faces of $\mathcal{P}$.

Note that it is not required to treat pseudomanifold degeneracies, as the trihedralization step previously solves these kinds of degeneracies.

A technique to treat degeneracies based on a simulated perturbation, that relies on simple geometric comparisons, is presented. This approach has a certain connection with the simulation of simplicity proposed by Edelsbrunner et al. [89]. When a degeneracy is detected, it is considered that the involved faces of $\mathcal{P}$ are displaced inwards by an

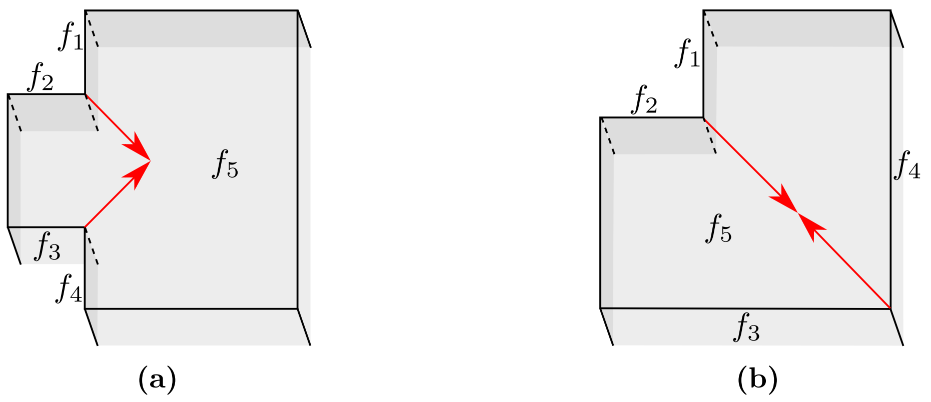


Figure 7.10: **(a)** Degeneracy by coplanarity. The junction between the ray $\{f_1, f_2, f_5\}$ and $\{f_3, f_4, f_5\}$ is degenerate due to coplanar faces f_1 and f_4. **(b)** Degeneracy by coincidence. The junction between the ray $\{f_1, f_2, f_5\}$ and $\{f_3, f_4, f_5\}$ is degenerate because the Voronoi vertex is equidistant to five faces.

infinitesimal amount. However, the computation of the coordinate of Voronoi vertices is always based on the original coordinates of $\mathcal{P}$. If a degeneracy is detected, the degeneracy by coplanarity is treated before the degeneracy by coincidence. In the following, it is presented how to treat each kind of degeneracy.

7.8.1 Degeneracy by coplanarity

A degeneracy by coplanarity occurs when a set of points that are L_∞ equidistant to two or more collinear faces define a three-dimensional volume in $\mathcal{P}$ which separates different Voronoi regions.

Property 20. *The set of points of $\mathcal{P}$ that are L_∞ equidistant to two or more coplanar faces of $\mathcal{P}$ can define a three-dimensional volume, called* bisector volume, *that separates Voronoi regions and it is bound by Voronoi faces (see Figure 7.11).*

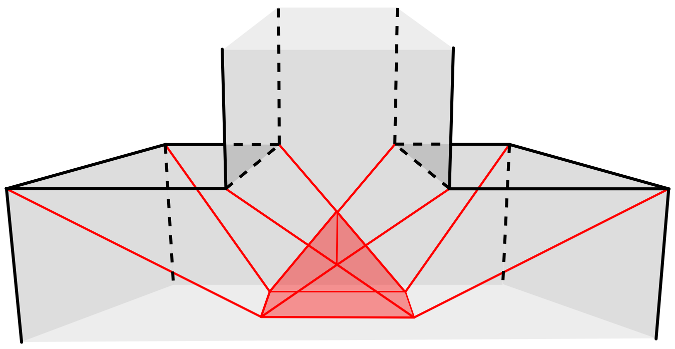


Figure 7.11: A portion of an orthogonal polyhedron. Degeneracy by coplanarity induced by two coplanar faces. The bisector volume is shown in red.

In two dimensions, two or more collinear edges may induce a bisector area (see Section 6.5.2). By choosing any part of the boundary of that area the degeneracy is solved. Analogously, in three dimensions the boundary of the bisector volume is selected. Every time a degeneracy by coplanarity is detected, a perturbation of the involved coplanar faces is simulated. The perturbation arbitrarily selects a portion of the bisector volume boundary that is L_∞ equidistant to the coplanar faces.

Lemma 31. *Consider a set of coplanar faces inducing a degeneracy by coplanarity. It is possible to perturb them such that the degeneracy is removed.*

Proof. Consider that each face of the coplanar faces is moved inwards $\mathcal{P}$ by a different infinitesimal amount. Then, the distance along the coordinate aligned with the coplanar faces is different for each perturbed coplanar face. As a consequence, the L_∞ distance must differ between each coplanar face. Thus, after the perturbation the points of the bisector volume must belong to a Voronoi region or to a Voronoi face. □

7.8.2 Degeneracy by coincidence

A degeneracy by coincidence occurs when a Voronoi vertex is equidistant to more than four non-coplanar faces. Observe that after treating first a degeneracy by coplanarity, a degeneracy by coincidence still may arise.

Lemma 32. *After the coplanar perturbation, a coincident degenerate vertex of $V_\infty(\mathcal{P})$ is L_∞ equidistant to at most six non-coplanar faces of $\mathcal{P}$.*

Proof. A degenerate vertex is L_∞ equidistant to at most twenty-four faces (see Lemma 30). However, six sets of four of these faces must be coplanar. After the coplanar perturbation, only one face of each set of four coplanar faces can be L_∞ equidistant to a Voronoi vertex. Thus, at most six non-coplanar faces can be L_∞ equidistant to a Voronoi vertex. □

Lemma 33. *A coincident degenerate vertex of $V_\infty(\mathcal{P})$ can be separated into two or four non degenerate Voronoi vertices with the same coordinates and a zero length edge connecting them or a zero area Voronoi face bounding them, respectively.*

Proof. First, consider without loss of generality that six faces defining a coincident degenerate vertex of $V_\infty(\mathcal{P})$ (see Lemma 32) are associated to the next half-spaces oriented inside $\mathcal{P}$:

$$h_1 : x > c_1\ ,\ h_2 : x < c_2\ ,\ h_3 : y > c_3\ ,\ h_4 : y < c_4\ ,\ h_5 : z > c_5\ ,\ h_6 : z < c_6$$
$$c_1 \neq c_2\ \wedge\ c_3 \neq c_4\ \wedge\ c_5 \neq c_6$$

Thus, as the Voronoi vertex is L_∞ equidistant to six faces it follows that $c_2 - c_1 = c_4 - c_3 = c_6 - c_5 > 0$. Assume that a perturbation of one infinitesimal unit ϵ is done for the faces aligned with the y coordinate, such that $h_3 : y > c_3 + \epsilon$ and $h_4 : y < c_4 - \epsilon$ and a perturbation of two infinitesimal ϵ units is done for the faces aligned with the z coordinate, such that $h_5 : z > c_5 + 2\epsilon$ and $h_6 : z < c_6 - 2\epsilon$. Then, the six faces cannot define a single cube as the y length of the cube is now $c_4 - c_3 + 2\epsilon$ and the z length is $c_6 - c_5 + 4\epsilon$. Thus, four different cubes must be defined by the six faces and their centers are four Voronoi vertices with the same coordinates bounded by a zero area Voronoi face.

Finally, the case of five faces defining a coincident degenerate vertex can be reasoned similarly to the case of six faces, and induce two Voronoi vertices with the same coordinates connected by a zero area Voronoi edge. □

Thus, every time a coincident degeneracy is detected a simulated perturbation is applied to the faces aligned with the y and the z coordinate that are involved in the degeneracy (see Lemma 33). Figure 7.12 shows two degeneracies by coincidence and the result of applying the simulated perturbation.

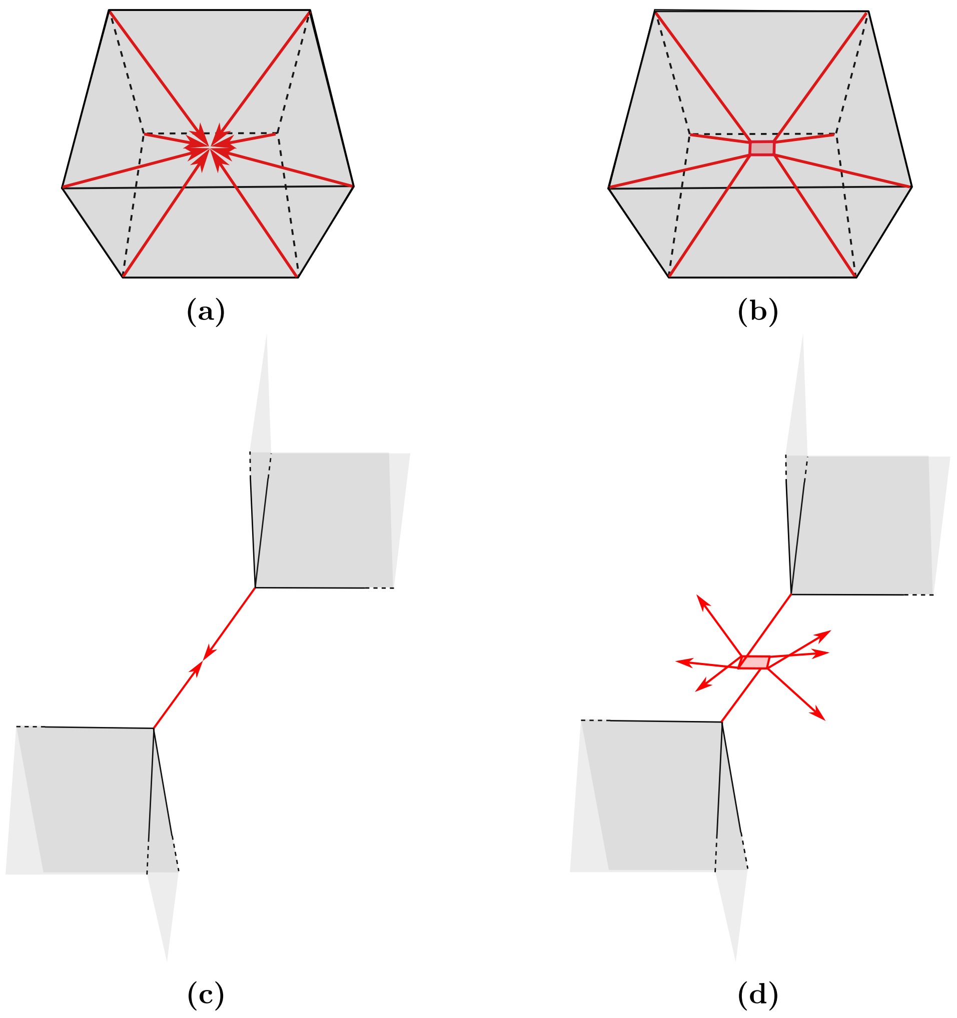


Figure 7.12: In the left column, a degeneracy by coincidence induced by six non-coplanar faces. In the right column, illustration of the faces perturbation, that results in four Voronoi vertices bounded by a Voronoi face. Observe that the two vertex rays of **(c)** have three incident reflex edges.

7.9 Straight skeleton equivalence

In this section, a detailed proof of the known equivalence between the Voronoi faces of $V_\infty(\mathcal{P})$ and the straight skeleton (see Section 3.1.5) of $\mathcal{P}$ is given.

Lemma 34. *The edges and vertices of $\mathcal{P}$ move at constant velocity during the straight skeleton propagation.*

Proof. All the edges of $\mathcal{P}$ have an angle of $\frac{\pi}{2}$ or $\frac{3\pi}{2}$. Thus, they move at a constant velocity. Moreover, all the vertices of $\mathcal{P}$ have three incident edges, each one aligned with a different axis, that move at constant speeds. As a consequence, all the vertices of $\mathcal{P}$ also move at constant speeds. □

Figure 7.13 shows the straight skeleton wavefront propagation of a simple orthogonal polyhedron. The following lemma is analogous to the two-dimensional straight skeleton equivalence Lemma 16.

Lemma 35. *Let $\mathcal{P}$ be an orthogonal polyhedron without coplanar faces. The Voronoi faces of $V_\infty(\mathcal{P})$ coincide with the straight skeleton of $\mathcal{P}$.*

Proof. Let p be a point of the Voronoi faces of $V_\infty(\mathcal{P})$, which is L_∞ equidistant to a set F of two or more non-coplanar faces of $\mathcal{P}$. Lemma 34 states that the edges and vertices of $\mathcal{P}$ move at a constant velocity during the straight skeleton wavefront propagation. Thus, at any time of the propagation, p is L_∞ equidistant to all the faces F. As with the two-dimensional case, in order to maintain this latter property it is crucial that the faces of $\mathcal{P}$ are axis-aligned. Furthermore, it is not possible that at any time of the straight skeleton wavefront propagation any face f of $\mathcal{P}$ not included in F becomes L_∞ equidistant to p, as this would imply that f has been moving inwards at a greater speed than the faces of F. Then, it follows that the straight skeleton propagation of F reaches p at the same time and as a consequence p must belong to the straight skeleton of $\mathcal{P}$.

Observe that, analogously to the two-dimensional case, if $\mathcal{P}$ contains coplanar edges, p could be L_∞ equidistant to an arbitrary amount of coplanar faces and the equivalence do not holds. □

For the sake of simplicity, it is considered that coplanar faces become a single face when they collapse, during the wavefront propagation, to a straight skeleton edge or vertex. In this case, the Voronoi faces of $V_\infty(\mathcal{P})$ that do not bound a bisector volume (see Section 7.8.1) coincide with the straight skeleton. Observe that the straight skeleton of orthogonal polyhedra can be computed by the previously presented sweep plane algorithm.

7.10 Experimental results

The presented sweep line algorithm to compute the L_∞ Voronoi diagram and the straight skeleton of orthogonal polyhedra have been implemented and compared with an optimized thinning approach [30] that computes the straight skeleton of a binary volume. The introduced algorithm has been implemented in C++ and the source code consists of about

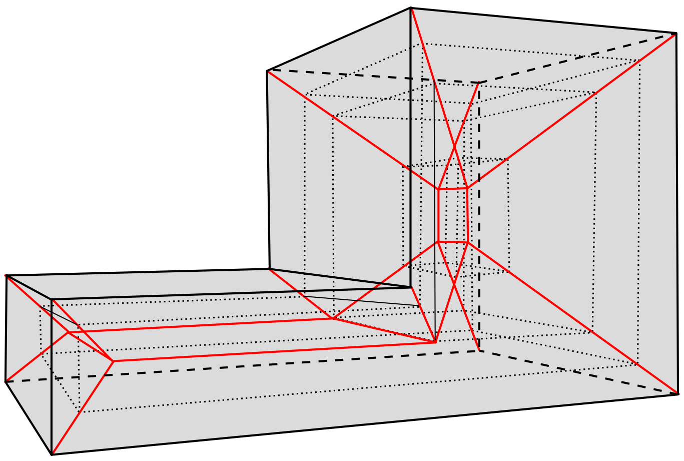


Figure 7.13: The dotted lines illustrate the straight skeleton wavefront propagation. The straight skeleton and L_∞ Voronoi diagram faces coincide. The L_∞ Voronoi diagram edges are shown in red.

three thousand lines of code. The experiments were conducted with a database of around one hundred binary shapes and three thousand samples of randomly generated orthogonal polyhedra with varying densities (see Figure 7.14). The available memory was limited to 16GB and the time computation was at most thirty minutes for each dataset. Figures 7.16 and 7.17 shows the $V_\infty(\mathcal{P})$ of some datasets.

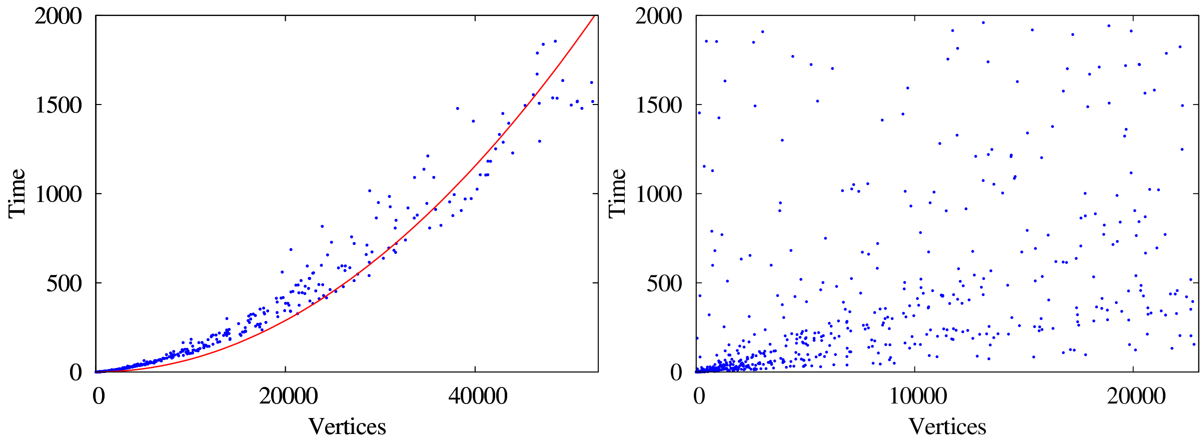
It has been verified experimentally that the combinatorial complexity of $V_\infty(\mathcal{P})$ is in fact linear (see Figure 7.15), although the theoretical combinatorial complexity of the straight skeleton of $\mathcal{P}$ is quadratic [33]. The average number of vertices of $V_\infty(\mathcal{P})$ was $5.62\,n$, where n is the number of vertices of $\mathcal{P}$.

The thinning method to compute the skeleton exceeded the thirty minutes limit of computation time datasets with more than ten thousand vertices. The time complexity of the thinning approach is bounded by the number of voxels of the binary volume representing $\mathcal{P}$ and not by the number of vertices of $\mathcal{P}$. Thus, the time data shown in Figure 7.14b is not correlated with the number of vertices.

As expected, the thinning approach was much slower than the presented approach because its time complexity depends on the size of the binary volume, which grows fast. The presented method to compute the skeleton exceeded the thirty minutes limit of computation time for datasets with more than sixty thousand vertices (see Figure 7.14a). As it has been verified that the combinatorial complexity of $V_\infty(\mathcal{P})$ is linear in the average case, the expected runtime of the presented algorithm may be quadratic. Thus, the expected average runtime was $0.72\,n^2\,\mu s$, where n is the number of vertices of $\mathcal{P}$.

7.11 Conclusions

A method to compute the L_∞ Voronoi diagram and the straight skeleton of an orthogonal polyhedron has been presented. Although the medial axis of polyhedra has been extensively



(a) Presented approach. **(b)** Approach of [30].

Figure 7.14: Running times of different methods to compute $V_\infty(P)$. The abscissa denotes the number of vertices of $\mathcal{P}$. The ordinate denotes the running time in seconds. Every blue point depicts the runtime for a single dataset. The red line corresponds to the curve fitted to the expected time complexity of the algorithm, with respect to the number of vertices of the polyhedron.

explored, the straight skeleton of polyhedra has devised less interest. However, the straight skeleton it is just composed of planar faces. A sweep line algorithm that works directly with the boundary of the shape is introduced. As a consequence, the time complexity of the presented algorithm is constrained by the size of its boundary. On the contrary, the time complexity of thinning and distance transform approaches are bound by the volume of the discretized object. Unlike the existing straight skeleton approaches that simulate a shrink of the object boundary, a plane-sweep algorithm has been considered. Furthermore, the presented algorithm is output sensitive with respect to the number of events generated by the algorithm. As the presented algorithm is designed for practical purposes, it is important that it is also robust. A technique to treat geometric degeneracies based on a simulated perturbation of the input polyhedron has been presented.

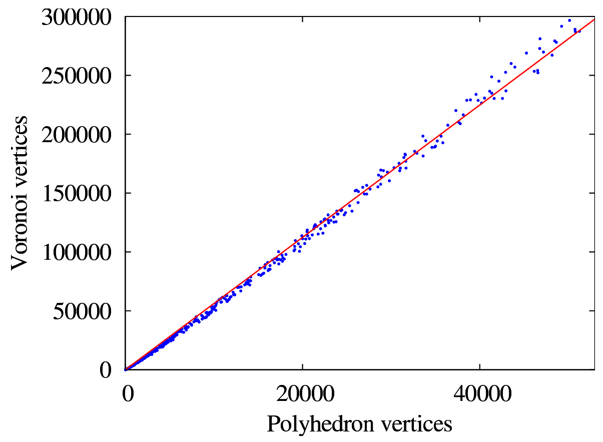


Figure 7.15: Vertices of $V_\infty(\mathcal{P})$ of several random and real datasets. The abscissa denotes the number of vertices of $\mathcal{P}$. The ordinate denotes the number of vertices of the straight skeleton of $\mathcal{P}$. The red line corresponds to the curve fitted to the expected number of vertices of $V_\infty(\mathcal{P})$, with respect to the number of vertices of $\mathcal{P}$.

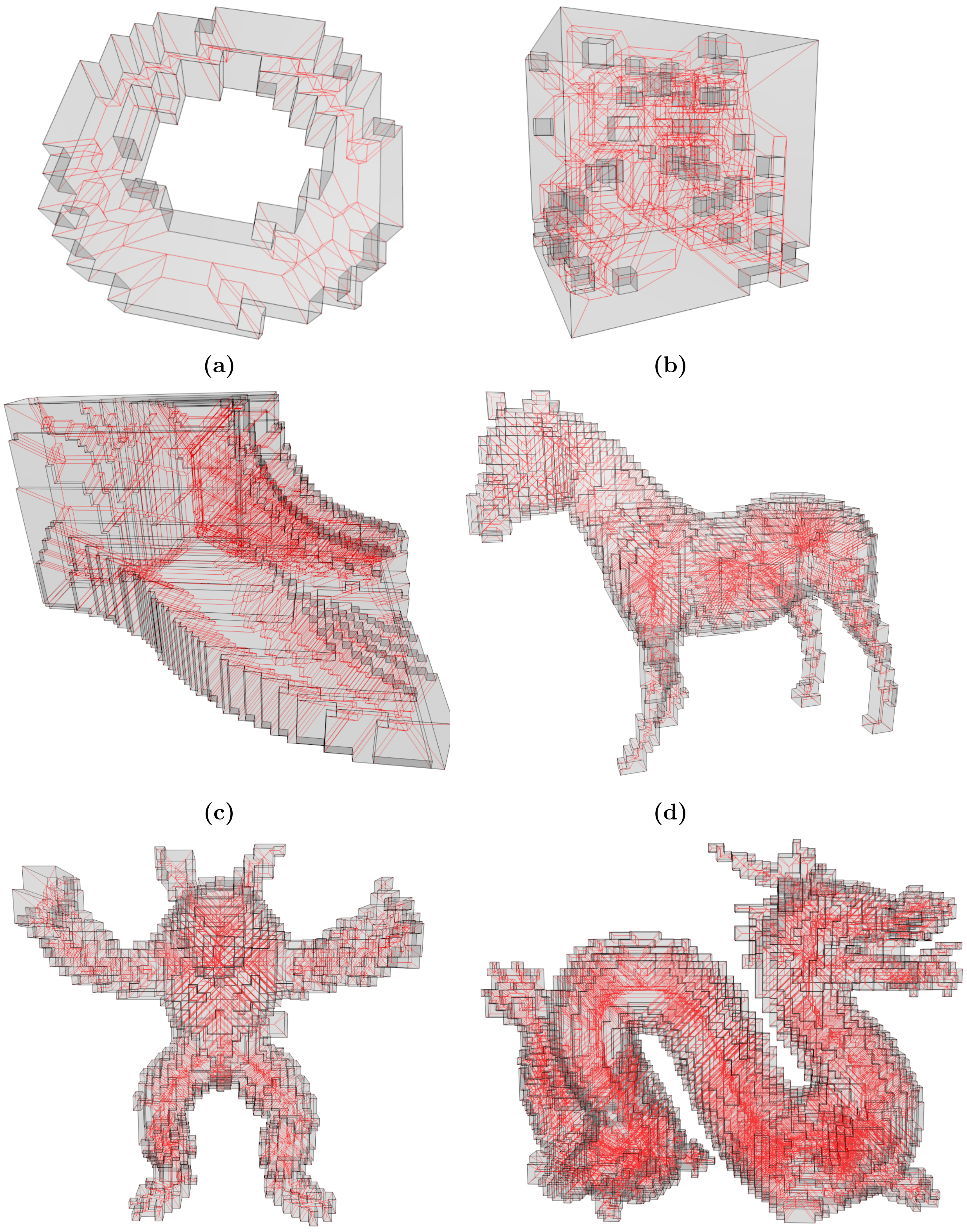
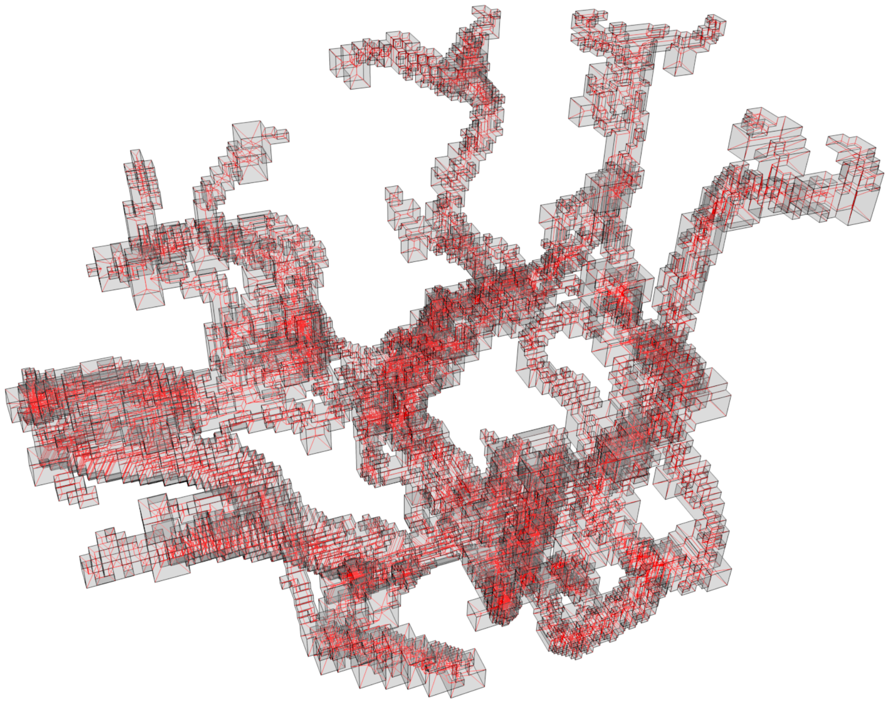
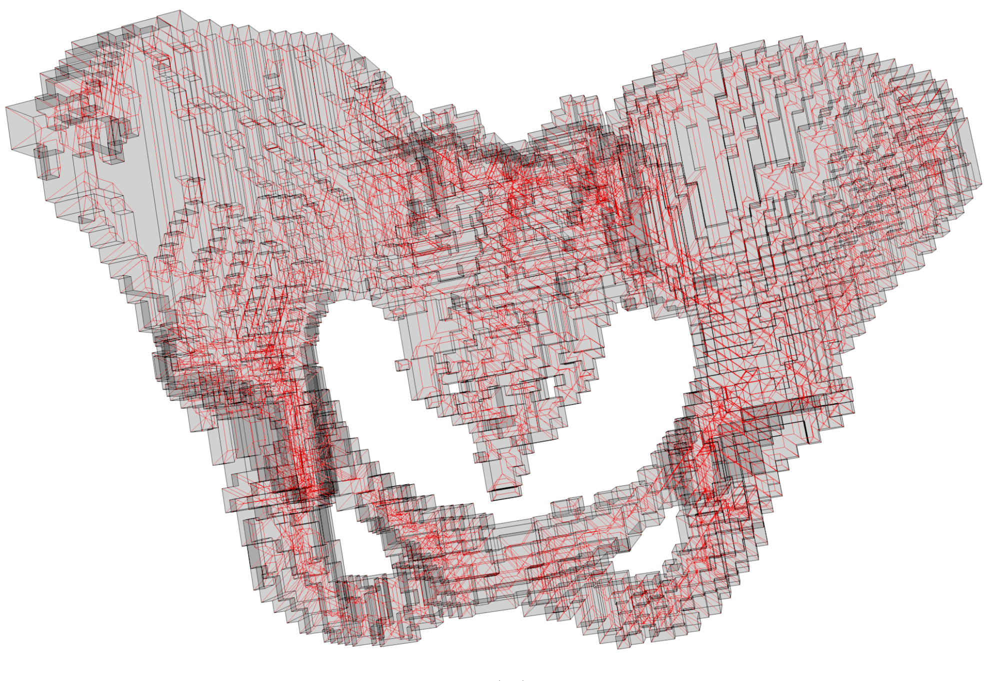


Figure 7.16: L_∞ Voronoi diagram of some three-dimensional datasets. The edges of $V_\infty(\mathcal{P})$ are shown in red. Dataset 7.16b has been randomly generated.



(a)



(b)

Figure 7.17: L_∞ Voronoi diagram of some three-dimensional datasets. **(a)** An aneurysm in the wall of a blood vessel. **(b)** A human pelvis.

Chapter 8

The cube skeleton

Contents

8.1 Introduction

This chapter introduces the cube skeleton of orthogonal shapes, a skeletal representation that extends and refines the L_∞ Voronoi diagram of orthogonal shapes (see Chapter 2).

First, it is discussed the problems with the L_∞ Voronoi diagram and the straight skeleton of orthogonal shapes, motivating the introduction of the cube skeleton. Let $\mathcal{S}$ be an orthogonal shape and let $E_\infty(\mathcal{S})$ be the set of points of $\mathcal{S}$ which do not belong to a single Voronoi region of the L_∞ Voronoi diagram (see Definition 22). Note that the set E_∞ separates Voronoi regions. The set E_∞ may be regarded as a skeletal representation of orthogonal shapes. However, E_∞ may not be homotopically equivalent to the orthogonal shape or may not effectively reduce its dimension.

Consider to the orthogonal polygon of Figure 8.1a. There exists a set of points in E_∞ that are equidistant to the two horizontal collinear edges of the polygon and are contained in the triangular area shown in red, as seen in Property 13. In this case, E_∞ does not reduce the dimension of the orthogonal shape. Similarly, a set of points in E_∞ of Figure 7.11 are equidistant to two coplanar faces and contained in a volume, as seen in Property 20. Papadopoulou et al. [175] consider only the outermost boundary of the region of E_∞ induced by two or more collinear edges of an orthogonal polygon, according to a lexicographical order between collinear edges [27]. Similarly, a perturbation technique of the collinear edges is considered in Chapter 6. However, this simple criterion defines a subset of E_∞ that may not be unique because it depends on the lexicographical order. In addition, the set E_∞ of Figure 8.1b is not homotopically equivalent to the orthogonal polyhedron because the Voronoi region associated to the face with a hole, disconnects the upper and lower connected sets of E_∞ points. Assigning a lexicographical order between coplanar faces, in analogy to the two-dimensional classical technique (see Section 6.5), does not ensure homotopy equivalence. Thus, the main motivation of the cube skeleton resides on the problems that exclusively arise in three dimensions.

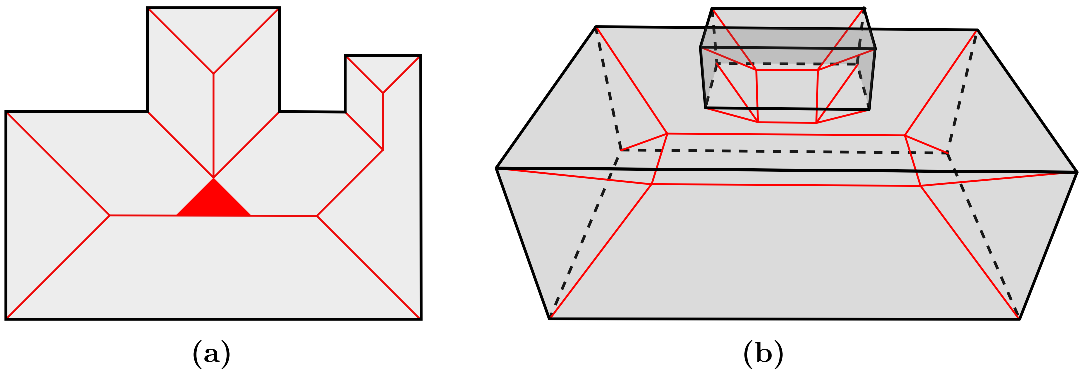


Figure 8.1: Using the L_∞ Voronoi diagram as a skeletal representation. Set of points E_∞ shown in red. **(a)** Orthogonal polygon. **(b)** Orthogonal polyhedron.

Concerning the straight skeleton of orthogonal shapes, in the literature it is not analyzed in detail how to deal with collinear edges and coplanar faces. In theory, these edges and faces move inwards during the straight skeleton propagation until they vanish into one or more points or segments. In two dimensions, some authors consider that after both collinear edges vanish, a new ray with its origin on the vanishing point and moving perpendicular to both edges is created. However, the three-dimensional case has not yet been analyzed and requires a more careful solution. As an example, for the polyhedron of Figure 8.1b the straight skeleton may not be homotopically equivalent to the orthogonal shape: the face with a hole is moved inwards until the hole vanishes into a single edge and the lower part of the straight skeleton remains disconnected to the upper part.

In this chapter a novel skeletal representation of orthogonal shapes, called the *cube skeleton*, is presented. It is proved that the cube skeleton preserves the homotopy of the original shape and reduces its dimension. The cube skeleton extends and refines the set E_∞ of an orthogonal shape in order to ensure homotopical equivalence and dimension reduction. As it will be shown, orthogonal shapes are well suited with the L_∞ metric in order to define skeletal representations. This chapter is an extended and adapted version of the first half of the article [158].

8.2 Cube skeleton definition

In this section the basic definitions and properties that will be used to define the cube skeleton are introduced.

Definition 57. *A* boundary element *of an orthogonal shape is:*

- *A vertex of an orthogonal shape.*
- *The set of collinear edges of an orthogonal shape, without their boundary vertices.*
- *The set of coplanar faces of an orthogonal polyhedron, without their boundary edges.*

Along this chapter, $\mathcal{S}$ is the interior of an orthogonal shape and $\partial\mathcal{S}$ is the set of all the boundary elements of $\mathcal{S}$. Observe that the intersection between every two different boundary elements of $\partial\mathcal{S}$ is empty and their union is exactly the boundary set of $\mathcal{S}$.

Definition 58. *Let b be a boundary element. The set $\tau(b)$ defines the coordinates of b that have a constant value:*

$$\tau(b) = \{i \in \{1 \ldots n\} : \forall x, y \in b, x_i = y_i\}$$

If $i \in \tau(b)$, b_i denotes the constant value of the coordinate i for all points in the boundary element b.

Definition 59. *Let b be a boundary element and let n be the dimension of the orthogonal shape. The dimension of b is denoted by* $\dim(b)$ *and is directly related to n and $\tau(b)$:*

$$\dim(b) = n - |\tau(b)|$$

If the boundary element is a vertex or composed of open edges or open faces its dimension is zero, one and two, respectively. The boundary elements of arbitrary dimension are denoted by the letter b. The boundary elements of dimension strictly zero, one or two are denoted by the letters v, e and f, respectively. Let $e, f \in \partial\mathcal{S}$, then ∂e and ∂f contains the vertices and edges of $\partial\mathcal{S}$ enclosing e and f, respectively.

Under the Euclidean metric, each point in the interior of an orthogonal shape has a non empty set of closest points on the shape boundary [150]. The number of closest points is used to define the medial axis. However, under the L_∞ metric it does not make sense to consider the number of closest points on the boundary because this number is usually infinite. For this reason, the cube skeleton will be defined by the set of closest boundary elements. The following definitions are given in order to properly define the set of closest boundary elements of a point in $\mathcal{S}$.

Note that the set $\epsilon(x, \partial\mathcal{S})$ (see Definition 19) defines a set of boundary elements strictly at a distance $d_\infty(x, \partial\mathcal{S})$ from x and ignores those edges or faces that exclusively contain the nearest points to x in its vertices or edges, respectively. As an example, consider the point x_2 of Figure 8.3. The set $\epsilon(x_2, \partial\mathcal{S}) = \{v_3\}$ only contain the vertex v_3, because the distance to its incident edges e_1 and e_2 do not consider the vertex v_3 shared by both edges.

Definition 60. *Let x be a point of $\mathcal{S}$. The set $\sigma_p(x, \mathcal{S})$ defines a subset of boundary elements from $\epsilon(x, \partial\mathcal{S})$ such that:*

$$\sigma_p(x, \mathcal{S}) = \{b_1 \in \epsilon(x, \partial\mathcal{S}) : \nexists b_2 \in \epsilon(x, \partial\mathcal{S}), b_1 \in \partial b_2\}$$

Several boundary elements of $\epsilon(x, \partial\mathcal{S})$ may lie on a common plane or line supporting them. For example, a set of edges enclosing a face lie on a common plane. Only the boundary elements of $\epsilon(x, \partial\mathcal{S})$ that are not contained in the boundary of another boundary element of $\epsilon(x, \partial\mathcal{S})$ belong to $\sigma_p(x, \mathcal{S})$. As an example, consider the point x_1 of Figure 8.3. The set $\epsilon(x_1, \partial\mathcal{S}) = \{e_1, e_3, v_1, v_2\}$ is properly refined by its subset $\sigma_p(x_1, \mathcal{S}) = \{e_1, e_3\}$.

Definition 61. *A* flat *is a point, line or plane in the n-dimensional space.*

Definition 62. *Let b be a boundary element.* flat(b) *is the flat that supports b and has the same dimension than b:*

$$\text{flat}(b) = \{x \in \mathbb{R}^n : \forall i \in \tau(b), x_i = b_i\}$$

For example, an edge is contained in a line and a face is contained in a plane, that are one and two-dimensional flats, respectively.

Definition 63. *Let b be a boundary element of $\partial\mathcal{S}$.* inter$(b, \mathcal{S})$ *is the intersection between the flat supporting b and $\mathcal{S}$:*

$$\text{inter}(b, \mathcal{S}) = \text{flat}(b) \cap \mathcal{S}$$

Observe that inter$(b, \mathcal{S})$ could be empty because $\mathcal{S}$ is an open set.

Lemma 36. *Let b be a boundary element of $\partial\mathcal{S}$. Then,* inter$(b, \mathcal{S})$ *has the following properties:*

- *If* $\dim(b) = 0$ *the set* inter$(b, \mathcal{S})$ *is empty.*
- *If* $\dim(b) = 1$ *the set* inter$(b, \mathcal{S})$ *is either empty or a set of collinear open segments.*
- *If* $\dim(b) = 2$, *the set* inter$(b, \mathcal{S})$ *is either empty or enclosed in an orthogonal polygon.*

Proof. If $\dim(e) = 0$, the intersection between a single point on $\partial\mathcal{S}$ and $\mathcal{S}$ is empty, because $\mathcal{S}$ is an open set. If $\dim(b) = 1$, inter$(b, \mathcal{S})$ is the intersection between a line and $\mathcal{S}$ which, if not empty, is composed of a set of collinear open segments contained in the line (see Figure 8.2b). Note that if flat(b) contains any reflex vertex of $\partial\mathcal{S}$ then inter$(b, \mathcal{S})$ is not empty. Finally, if $\dim(b) = 2$ the set inter$(b, \mathcal{S})$ is the intersection between a plane and $\mathcal{S}$, bounded by a set of axis-aligned segments such that every endpoint of a segment must be shared by exactly another endpoint (see Figure 8.2a). As $\mathcal{S}$ is open, inter$(b, \mathcal{S})$ is open and a segment enclosing inter$(b, \mathcal{S})$ cannot be shared by more than one other segment. Thus, the intersection is enclosed by an orthogonal polygon. In this case, if flat(b) contains any reflex edge of $\partial\mathcal{S}$ then inter$(b, \mathcal{S})$ is not empty. $\square$

Definition 64. *Let b be a boundary element. The set $\overline{\tau}(b)$ defines the coordinates of b that do not have a constant value:*

$$\overline{\tau}(b) = \{1 \dots n\} \setminus \tau(b)$$

Definition 65. *Let b be a boundary element of $\partial\mathcal{S}$ and x a point of $\mathcal{S}$.* $\operatorname{proj}(x, b)$ *is the* orthogonal projection *of x onto the flat supporting b:*

$$\operatorname{proj}(x, b) = \{y \in \mathbb{R}^n : \forall i \in \tau(b), \forall j \in \overline{\tau}(b), y_i = b_i, y_j = x_j\}$$

Figure 8.2 illustrates the orthogonal projection of a point onto a flat and the intersection between a flat and $\mathcal{S}$.

Lemma 37. *Let x be a point of $\mathcal{S}$ and $\sigma_p(x, \mathcal{S}) = \{b_1\}$. Suppose that the projection of x onto the flat of b_1 is contained in the intersection between the flat of b_1 and $\mathcal{S}$. Then, any boundary element in σ_p of the projected point belongs to the boundary of b_1. More formally:*

$$\operatorname{proj}(x, b_1) \in \operatorname{inter}(b_1, \mathcal{S}) \Rightarrow \forall b_2 \in \sigma_p(\operatorname{proj}(x, b_1), \operatorname{inter}(b_1, \mathcal{S})), b_2 \in \partial b_1$$

Proof. Consider in order to arrive to a contradiction that $b_2 \notin \partial b_1$. As $b_2 \in \operatorname{flat}(b_1)$ then $b_2 \notin \partial\mathcal{S}$. Otherwise, there would be another boundary element in $\sigma_p(x, \mathcal{S})$, with at least the same dimension of b_1 and containing in its boundary b_2. It follows that the distance from x to b_1 must be strictly lower than to b_2:

$$d_\infty(x, b_1) < d_\infty(x, b_2) \tag{8.1}$$

In addition, observe that the distance $d_\infty(x, b_1)$ is defined by the coordinates $\tau(b_1)$:

$$\exists i \in \tau(b_1), d_\infty(x, b_1) = |x_i - b_{1i}|$$

Thus, the distance from x to b_1 must be greater or equal than the distance from $\operatorname{proj}(x, b_1)$ to b_1:

$$d_\infty(x, b_1) \geq d_\infty(\operatorname{proj}(x, b_1), b_1) \tag{8.2}$$

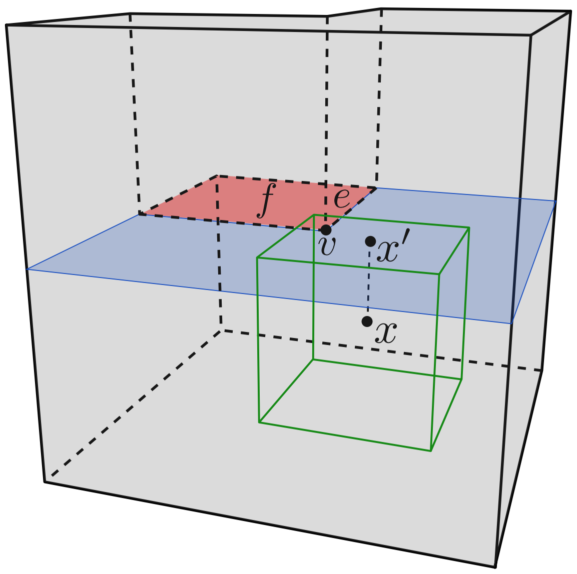
On the contrary, the distance $d_\infty(x, b_2)$ must be defined by the coordinates $\overline{\tau}(b_1)$. Otherwise, b_2 would have belonged to $\epsilon(x, \partial\mathcal{S})$ and $b_2 \in \partial b_1$, by definition of σ_p. Then, it follows that:

$$d_\infty(x, b_2) = d_\infty(\operatorname{proj}(x, b_1), b_2) \tag{8.3}$$

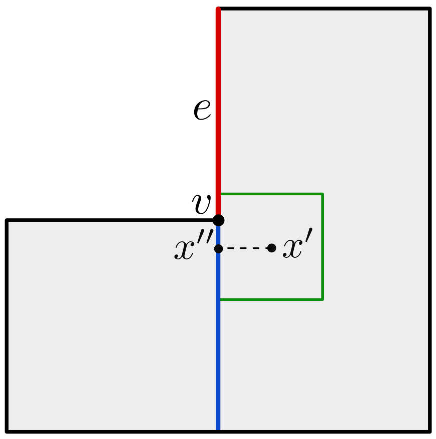
Finally, it follows from the three previous Equations 8.1, 8.2 and 8.3 that:

$$d_\infty(\operatorname{proj}(x, b_1), b_1) < d_\infty(\operatorname{proj}(x, b_1), b_2)$$

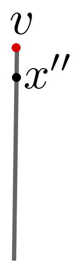
Which leads to a contradiction: the set $\sigma_p(\operatorname{proj}(x, b_1), \operatorname{inter}(b_1, \mathcal{S}))$ cannot contain b_2 because the distance from $\operatorname{proj}(x, b_1)$ to b_2 is greater than to b_1. This concludes the proof of Lemma 37. □



(a) $\mathcal{S}$.



(b) inter $(f, \mathcal{S})$.



(c) inter $(e, \text{inter}\,(f, \mathcal{S}))$.

Figure 8.2: Illustration of the orthogonal projection of a point onto a flat and the intersection between a flat and an orthogonal shape. **(a)** The set $\sigma_p\,(x, \mathcal{S})$ contains the face f, drawn in red. The green cube is a cube centered at x whose boundary are the points at distance $d_\infty\,(x, \partial\mathcal{S})$ from x. The orthogonal polygon shown in blue is inter $(f, \mathcal{S})$ and $x' = \text{proj}\,(x, f)$. **(b)** The set $\sigma_p\,(x', \text{inter}\,(f, \mathcal{S}))$ contains the edge e, drawn in red. The green square is a square centered at x whose boundary are the points at distance $d_\infty\,(x, \partial\mathcal{S})$ from x. The blue segment is inter $(e, \text{inter}\,(f, \mathcal{S}))$ and $x'' = \text{proj}\,(x', e)$. **(c)** The set $\sigma_p\,(x'', \text{inter}\,(e, \text{inter}\,(f, \mathcal{S})))$ contains the vertex v, drawn in red.

An illustration for Lemma 37 can be seen in Figure 8.2. The set $\sigma_p\,(x, \mathcal{S})$ contains the face f (see Figure 8.2a). Thus, the nearest boundary elements of the point x projected onto the orthogonal polygon inter $(f, \mathcal{S})$ (see Figure 8.2b) or the segment inter $(e, \text{inter}\,(f, \mathcal{S}))$ (see Figure 8.2c) must be either an edge or a vertex on ∂f.

Suppose that a skeletal representation is defined by the set of points in $\mathcal{S}$ whose set

$\sigma_p(x,\mathcal{S})$ contains two or more boundary elements. Then, this skeletal representation could not be homotopically equivalent to $\mathcal{S}$ (see Figure 8.3 as an example). Thus, the definition of the set σ_p must be extended.

So far the definitions and properties that will be necessary to define a proper set of closest boundary elements have been presented. For this purpose the recursively defined set $\sigma(x,\mathcal{S})$ is introduced.

Definition 66. *Let x be a point of $\mathcal{S}$. The set $\sigma(x,\mathcal{S})$ contains the* closest boundary elements *from x to $\partial\mathcal{S}$:*

$$\sigma(x,\mathcal{S}) = \begin{cases} \sigma(\text{proj}(x,b), \text{inter}(b,\mathcal{S})) & \text{If } \sigma_p(x,\mathcal{S}) = \{b\} \text{ and } \text{proj}(x,b) \in \text{inter}(b,\mathcal{S}) \\ \sigma_p(x,\mathcal{S}) & \text{Otherwise} \end{cases}$$

The above definition distinguishes two cases: if $\sigma_p(x,\mathcal{S})$ defines a single boundary element b and $\text{proj}(x,b) \in \text{inter}(b,\mathcal{S})$, the closest boundary elements are found in the set $\text{inter}(b,\mathcal{S})$. Otherwise, the set $\sigma(x,\mathcal{S})$ is equal to the set $\sigma_p(x,\mathcal{S})$. Notice that the recursion must finish when the set $\sigma_p(x,\mathcal{S})$ contains two or more boundary elements or not later than when $\dim(b) = 0$, because the set $\text{inter}(b,\mathcal{S})$ will be empty (see Lemma 36).

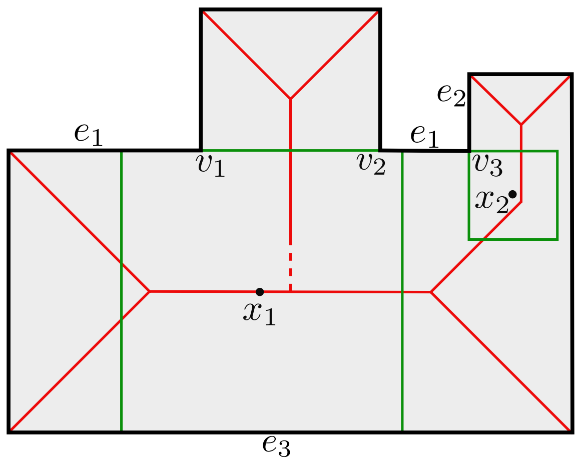


Figure 8.3: An orthogonal polygon. Illustration of Definitions 19, 60 and 66. The set of points such that $|\sigma_p(x,\mathcal{S})| \geq 2$ are drawn with the continuous red line. The dashed red line, contains the set of points such that $|\sigma_p(x,\mathcal{S})| = 1$ and $|\sigma(x,\mathcal{S})| \geq 2$. Note that the boundary element e_1 is composed of two collinear edges. In green, the squares centered at the points $x_1, x_2 \in \mathcal{S}$ with radius equal to $d_\infty(x_1,\partial\mathcal{S})$ and $d_\infty(x_2,\partial\mathcal{S})$, respectively.

In the rest of the chapter, the set $\sigma(x,\mathcal{S})$ is denoted by $\sigma(x)$ when $\mathcal{S}$ is the input orthogonal shape.

Lemma 38. *Let x be a point of $\mathcal{S}$. Then $\sigma(x) \subset \epsilon(x,\partial\mathcal{S}) \subset \partial\mathcal{S}$ and $\sigma(x) \neq \emptyset$.*

Proof. As a consequence of Lemma 37, the boundary elements found when x is projected onto a flat must be boundary elements of $\partial\mathcal{S}$ contained in $\epsilon(x,\partial\mathcal{S})$. In addition, as the closure of $\mathcal{S}$ is compact the set $\epsilon(x, \partial\mathcal{S})$ cannot be empty and its subset $\sigma_p(x, \mathcal{S})$ is also not empty because at least one boundary element must contain in its boundary other boundary elements of $\epsilon(x, \partial\mathcal{S})$. Thus, it follows that $\sigma(x) \neq \emptyset$. □

Definition 67. *(**Cube skeleton**) The cube skeleton CS of $\mathcal{S}$ is the set of points in $\mathcal{S}$ that have two or more closest boundary elements:*

$$CS = \{x \in \mathcal{S} : |\sigma(x)| \geq 2\}$$

Lemma 39. *Let x be a point of $\mathcal{S} \setminus CS$. Then $|\sigma(x)| = 1$.*

Proof. By Lemma 38 the set $\sigma(x)$ is not empty and if $x \in \mathcal{S} \setminus CS$ by Definition 67 must contain a single boundary element. □

Definition 68. *The open n-dimensional cube $\mathbb{C}_{c,r}$ with center c and radius r is the set:*

$$\mathbb{C}_{c,r} = \{x \in \mathbb{R}^n : d_\infty(c, x) < r\}$$

Definition 69. *(**Cube transform**) The cube transform CT of $\mathcal{S}$ is the set of n-dimensional cubes associated to each point of the cube skeleton with radius equal to the L_∞ distance to the boundary:*

$$CT = \left\{\mathbb{C}_{x, d_\infty(x, \partial\mathcal{S})} : x \in CS\right\}$$

8.3 Cube skeleton properties

In this section, it is shown in Theorem 1 that the cube skeleton is homotopically equivalent to $\mathcal{S}$ and in Theorem 2 that the cube skeleton reduces the dimension of $\mathcal{S}$ and has a constrained structure. In general, the proof of the important properties of the cube skeleton takes advantage of a piecewise partition of $\mathcal{S}$ induced by the boundary elements of $\partial\mathcal{S}$.

8.3.1 Homotopy equivalence

Definition 70. *Let b be a boundary element of $\partial\mathcal{S}$. The set $\lambda(b)$ are the points of $\mathcal{S} \setminus CS$ whose closest boundary element is b:*

$$\lambda(b) = \{x \in \mathcal{S} \setminus CS : \sigma(x) = \{b\}\}$$

Note that the set $\lambda(b)$ may be empty, for example, if the boundary element b is a convex vertex or edge of $\partial\mathcal{S}$. Next, some properties of the set $\lambda(b)$ are analyzed.

Lemma 40. *Let x be a point of $\mathcal{S} \setminus CS$ and let b_1 and b_2 be two boundary elements of $\partial\mathcal{S}$. If $x \in \lambda(b_1)$ and $\sigma_p(x, \mathcal{S}) = \{b_2\}$ then $x \notin \partial\lambda(b_2)$ and $b_1 \in \partial b_2$.*

Proof. As $\sigma(x, \mathcal{S}) = \{b_1\}$ and $\sigma_p(x, \mathcal{S}) = \{b_2\}$ then:

$$\dim(b_2) > \dim(b_1)$$

And by Lemma 37:

$$b_1 \in \sigma(\operatorname{proj}(x, b_2), \operatorname{inter}(b_2, \mathcal{S}))$$

Thus, as $b_1 \in \operatorname{flat}(b_2)$ it follows that:

$$b_1 \in \partial b_2$$

As $\operatorname{inter}(b_2, \mathcal{S})$ is a non empty open set there exists a neighborhood N_x of x such that:

$$\forall y \in N_x, \operatorname{proj}(y, b_2) \in \operatorname{inter}(b_2, \mathcal{S})$$

There cannot exist a point $y \in N_x$ such that $\sigma(y, \mathcal{S}) = \{b_2\}$. Thus, $\forall y \in N_x$ it holds that $\sigma(y, \mathcal{S}) \neq \{b_2\}$ and as a consequence:

$$x \notin \partial\lambda(b_2)$$

This concludes the proof of Lemma 40. □

Corollary 1. *Let x be a point of $\mathcal{S} \setminus CS$ and let b be a boundary element of $\partial\mathcal{S}$. If $x \in \lambda(b)$ and $\operatorname{proj}(x, b) \in b$ then $x \in \operatorname{int}(\lambda(b))$.*

Proof. Two different cases depending on the set $\sigma_p(x, \mathcal{S})$ are distinguished:

- $\sigma_p(x, \mathcal{S}) \neq \{b\} = \{b'\}$. In this case the set $\sigma(x, \mathcal{S})$ has been determined by recursion. Thus, $\operatorname{inter}(b', \mathcal{S})$ must be a non empty open set and there exists a neighborhood N_x of x such that:

  $$\forall y \in N_x, \operatorname{proj}(y, b') \in \operatorname{inter}(b', \mathcal{S})$$

  In addition, the projection of the points of N_x onto a boundary element that contains b' on its boundary is also in $\operatorname{inter}(b', \mathcal{S})$. As a consequence:

  $$\forall y \in N_x, \sigma_p(y, \mathcal{S}) \neq \sigma(y, \mathcal{S})$$

  As $\sigma(x, \mathcal{S})$ is obtained by recursion, the dimension of $\mathcal{S}$ can be reduced until having that $\sigma_p(x, \mathcal{S}) = \sigma(x, \mathcal{S}) = \{b\}$. This situation corresponds to the next base case.

- $\sigma_p(x, \mathcal{S}) = \{b\}$. If $\dim(b) > 0$ then b is open and there exists a neighborhood N_x of x such that:

  $$\forall y \in N_x, \operatorname{proj}(y, b) \in b \tag{8.4}$$

  Otherwise, if $\dim(b) = 0$ the projection $\operatorname{proj}(x, b)$ is always contained in b and there also exist a neighborhood N_x fulfilling Equation 8.4. Thus, it follows that $x \in \operatorname{int}(\lambda(b))$. This concludes the Corollary 1.

□

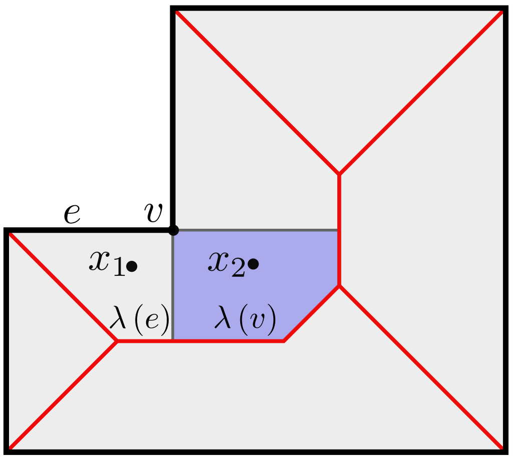


Figure 8.4: Illustration of Lemma 40 and its Corollary 1. Orthogonal polygon and its cube skeleton are drawn in red. The region $\lambda(v)$ is drawn in blue.

An illustration of Lemma 40 and its Corollary 1 can be seen in Figure 8.4. As proved in Lemma 40, the point $x_2 \in \lambda(v)$ such that $\sigma_p(x_2, \mathcal{S}) = \{e\}$ cannot be on the boundary of $\lambda(v)$ and the vertex v belongs to the boundary of the edge e. As shown by Corollary 1 the points $x_1 \in \lambda(e)$ and $x_2 \in \lambda(v)$ such that $\operatorname{proj}(x_1, e) \in e$ and $\operatorname{proj}(x_2, v) \in v$, respectively, must belong to the interior of their λ regions.

Definition 71. *Let x be a point and let b be a boundary element. The function $v(x,b)$ is:*

$$v(x,b) = x - \operatorname{proj}(x,b)$$

Definition 72. *Let x be a point of $\mathcal{S} \setminus CS$ and let b be a boundary element such that $x \in \lambda(b)$. The function $v(x)$ is:*

$$v(x) = v(x,b)$$

The function $v(x)$ will be used to prove the homotopy equivalence. Figures 8.5 and 8.6 illustrate the direction of $v(x)$ of the set of points $\mathcal{S} \setminus CS$ of a two and three-dimensional orthogonal shape, respectively.

Lemma 41. *For $x \in \mathcal{S} \setminus CS$ the function $v(x)$ is continuous.*

Proof. Let x be a point of $\mathcal{S} \setminus CS$ and b a boundary element of $\partial\mathcal{S}$. Let us first analyze the continuity of $v(x)$ in the interior of each non empty set $\lambda(b)$. In this case, it is clear that the function $v(x)$ is continuous because the coordinates $\tau(b)$ of $\operatorname{proj}(x,b)$ are constant and the coordinates $i \in \tau(b)$ of $\operatorname{proj}(x,b)$ are just the i-th coordinate of x.

Let b_1 and b_2 be two boundary element of $\partial\mathcal{S}$. As a consequence of Corollary 1 the continuity problems of $v(x)$ may arise when $x \in \lambda(b_1)$ and $\operatorname{proj}(x, b_1) \in \partial b_1$. Observe that this is equivalent to $x \in \lambda(b_1)$ and $\operatorname{proj}(x, b_1) \in b_2$, such that $b_2 \in \partial b_1$. Thus, the continuity of $v(x)$ is preserved if and only if:

$$v(x, b_1) = v(x, b_2) \tag{8.5}$$

That is equivalent to:

$$\text{proj}(x, b_1) = \text{proj}(x, b_2)$$

The possible critical points are studied by cases:

- $\dim(b_2) = \dim(b_1) - 1$. In this case there are two different subcases. If b_1 is a one-dimensional boundary element and b_2 is a vertex of its boundary, then $x \in \partial\lambda(b_2)$. This may occur if $\mathcal{S}$ is either an orthogonal polygon or polyhedron. Otherwise, b_1 is a two-dimensional boundary element and b_2 a one-dimensional boundary element on its boundary. In this case, $\mathcal{S}$ must be an orthogonal polyhedron and $x \in \partial\lambda(b_2)$.

- $\dim(b_2) = \dim(b_1) - 2$. In this case b_1 is a two-dimensional boundary element and the projection of x onto b_1 lies on a vertex b_2 of ∂b_1. The point x belongs to $x \in \partial\lambda(b_2)$ plus the boundary set of the λ regions of the incident edges of b_2 that belong to ∂b_1.

In both previous cases of critical points the next equation holds:

$$\text{proj}(x, b_1) \in b_2$$

Thus, the equivalence of Equation 8.5 is always true and the proof of Lemma 41 is now complete. □

An illustration for Lemma 41 can be seen in Figures 8.5 and 8.6. The point $x \in \mathcal{S} \setminus CS$ of Figure 8.5 belongs to $\lambda(e)$ and $\partial\lambda(v)$ and corresponds to the first case of possible critical points. The point $x \in \mathcal{S}$ of Figure 8.6 belongs to $\lambda(f)$, $\partial\lambda(e_1)$, $\partial\lambda(e_2)$ and $\partial\lambda(v)$ and corresponds to the second case of possible critical points.

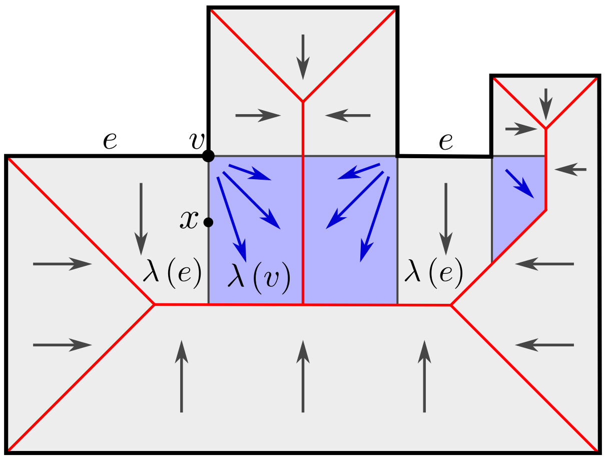
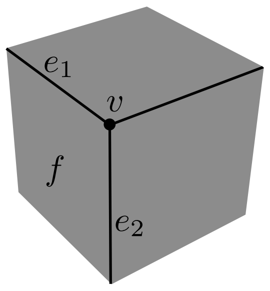
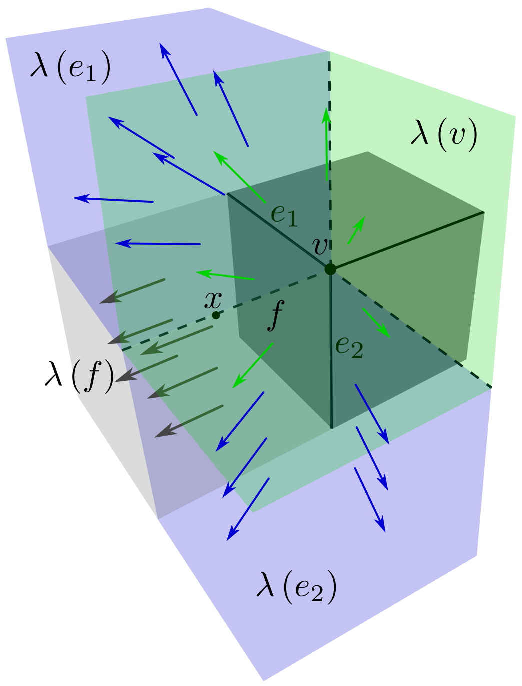


Figure 8.5: An orthogonal polygon and its cube skeleton are shown in red. Directions of $v(x)$ are indicated with arrows. The gray and blue arrows and portions of $\mathcal{S} \setminus CS$ are induced by boundary elements of dimension one and zero, respectively. The gray lines separating these regions are the points where the vector $v(x)$ needs to be smooth (see Lemma 41).





(a) **(b)**

Figure 8.6: **(a)** A vertex v of an orthogonal polyhedron with three incident reflex edges. **(b)** Directions of $v(x)$ are indicated with arrows. The gray, blue and green arrows and portions of $\mathcal{S}$ are induced by boundary elements of dimension two, one and zero, respectively. The planes separating the colored portions are regions where the vector $v(x)$ needs to be smooth (see Lemma 41).

Lemma 42. *Let b be a boundary element of $\partial\mathcal{S}$ and let $x \in \mathcal{S} \setminus CS$ be a point of $\lambda(b)$. Consider the open segment $s = int([x, proj(x, b)])$. Then, $\forall y \in s$ it holds that:*

$$y \in \lambda(b), \hat{v}(y) = \hat{v}(x), d_\infty(y, \partial\mathcal{S}) < d_\infty(x, \partial\mathcal{S})$$

Proof. Observe that if $y \in s$ then $y = \mu x + (1-\mu)\,\mathrm{proj}(x, b)$, with $\mu \in [0,1]$. Thus, $\forall i \in \tau(b)$ it holds that $y_i = \mu x_i + (1-\mu) b_i$ and $\forall j \in \overline{\tau}(b)$ it holds that $y_j = x_j$.

On the other hand, notice that as $x \in \lambda(b)$, then there exists a point $z \in b$ such that $d_\infty(x, z) = d_\infty(x, \partial\mathcal{S})$. This is due to the fact that if $\{b_2\} = \sigma_p(x, S)$ then there exists a point $z \in b_2$ such that $d_\infty(x, z) = d_\infty(x, \partial\mathcal{S})$ and, as b_2 is taken as an open set, the distance is achieved in a coordinate $i \in \tau(b_2)$. Thus, if $b \in \partial b_2$, which occurs when $\sigma(x, \mathcal{S})$ is obtained by recursion, the points in b also give the minimum distance. Moreover, z can be taken as $\mathrm{proj}(x, b)$, if $\mathrm{proj}(x, b) \in b$, or as close as required from $\mathrm{proj}(x, b)$, if $\mathrm{proj}(x, b) \in \partial b$. Thus, consider that $z \in b$ such that $d_\infty(\mathrm{proj}(x, b), z) < d_\infty(\mathrm{proj}(x, b), y)$. Then, if $i \in \tau(b)$ it follows that:

$$d_\infty(x, \partial S) = d_\infty(x, z) =\mid x_i - z_i \mid > \mid y_i - z_i \mid = d_\infty(y, z) \geq d_\infty(y, \partial S)$$

In addition, y must belong to $\lambda(b)$, otherwise there would be a boundary element $b_3 \in \partial S$ such that $b_3 \in \partial b$ and $d_\infty(y, b_3) < d_\infty(y, z)$, where z is the point of the previous argument. In this case, if $d_\infty(y, b_3)$ is achieved in a coordinate $i \notin \tau(b)$, as $x_i = y_i$, leads to a contradiction with the fact that $x \in \lambda(b)$. Let a be a point of $\mathcal{S}$ at a minimum distance from b_3. On the contrary, if $d_\infty(y, b_3)$ is achieved in a coordinate $i \in \tau(b)$ it follows that:

$$| a_i - x_i | \leq | a_i - y_i | + | y_i - x_i | < | z_i - y_i | + | y_i - x_i | = | z_i - x_i |$$

Which also gives a contradiction. Finally, as $y \in \lambda(b)$ then $\operatorname{proj}(x, b) = \operatorname{proj}(y, b)$ and $y - \operatorname{proj}(y, b) = \mu(x - \operatorname{proj}(x, b))$. As a consequence, the normalized vectors are equal, that is $\hat{v}(y) = \hat{v}(x)$. Observe that it is possible to move in direction $\hat{v}(x)$ on a neighborhood of x, and still remain in $\lambda(b)$. Thus, the intersection of s and $\lambda(b)$ is an open set. This concludes the proof of Lemma 42. □

Definition 73. *Let x be a point of $\mathcal{S} \setminus CS$. The distance from x to CS along the direction $v(x)$ is:*

$$d_{CS}(x) = \inf \{ d(x, y) : y \in CS, l > 0, y = x + l\hat{v}(x) \}$$

Lemma 43. *Let x be a point of $\mathcal{S} \setminus CS$. The distance $d_{CS}(x)$ is well defined and continuous.*

Proof. First, it is shown that the distance $d_{CS}(x)$ is well defined. It is known from Lemma 42 that the distance from x to $\partial\mathcal{S}$ must increase along the direction $v(x)$. Consider in order to arrive to a contradiction that no point of CS is of the form $x + l\hat{v}(x)$, where $l > 0$. Thus, the distance $d(x, \partial\mathcal{S})$ might grow to infinity. However, $\mathcal{S}$ is a bounded set, which leads to a contradiction.

Let b be a boundary element such that $x \in \lambda(b)$. As shown in the proof of Lemma 42 the intersection of s and $\lambda(b)$ is an open set and it is possible to move from x to CS. Then, there exists a point of CS at a distance $d_{CS}(x)$ from x. In addition, the continuity of $d_{CS}(x)$ is a direct consequence of the continuity of $v(x)$ (see Lemma 41). This concludes the proof of Lemma 43. □

Definition 74. *Let x be a point of $\mathcal{S}$. For any point $x \in \mathcal{S} \setminus CS$ the function $\psi(x)$ maps x to the cube skeleton:*

$$\psi(x) = x + d_{CS}(x)\,\hat{v}(x)$$

And for points $x \in CS$ the function $\psi(x)$ is:

$$\psi(x) = x$$

Lemma 44. *For $x \in \mathcal{S}$ the function $\psi(x)$ is continuous.*

Proof. It is known from Lemma 41 and 43 that $v(x)$ and $d_{CS}(x)$ are continuous when $x \in \mathcal{S} \setminus CS$. Thus, it directly follows that $\psi(x)$ is continuous in $\mathcal{S} \setminus CS$. It remains to show that $\psi(x)$ is also continuous in CS. There exists a neighborhood $N_x \subset \mathcal{S}$ of $x \in CS$. Thus, for any sequence $(y_n)_{n\in\mathbb{N}}$ of points in N_x which converges to x, the corresponding sequence $(\psi(y_n))_{n\in\mathbb{N}}$ converges to x, because the term $\hat{v}(x)$ is bound and the distance $d_{CS}(x)$ converges to zero. This concludes the proof of Lemma 44. □

Definition 75. *(MASSEY [161]) A* deformation retraction *of a space $\mathcal{S}$ onto a subspace CS is a continuous map:*

$$F : \mathcal{S} \times [0,1] \to \mathcal{S}$$

Such that for every $x \in \mathcal{S}$, $t \in [0,1]$ and $a \in CS$:

$$F(x,0) = x,\ F(x,1) \in CS \text{ and } F(a,t) = a$$

In other words, a deformation retraction is a homotopy between a retraction and the identity map on $\mathcal{S}$. The subspace CS is called a deformation retract of $\mathcal{S}$. A deformation retract is a special case of homotopy equivalence (see Definition 26). The next theorem follows a similar retraction as the one defined by Wolter [225] to prove the homotopy equivalence of the medial axis with respect to C^2-smooth shapes.

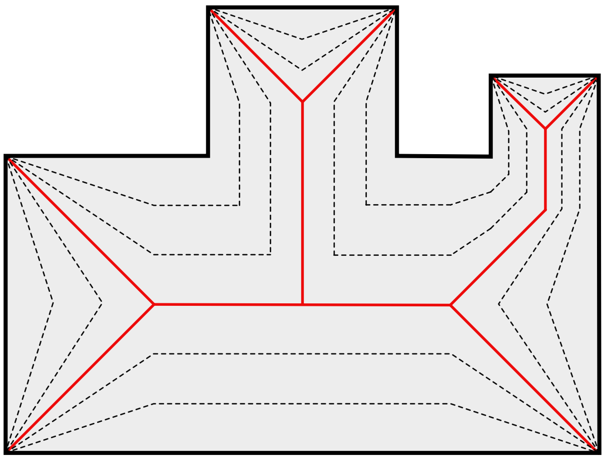


Figure 8.7: The cube skeleton is shown in red. The evolution of $F(x,t)$ (see Theorem 1) is shown with thin dotted lines inside the orthogonal polygon. Observe that as t approaches 1, $F(x,t)$ converges with the cube skeleton.

Theorem 1. *The cube skeleton is a deformation retract of S.*

Proof. The map $F(x,t)$ (see Definition 75) is:

$$F(x,t) = (1-t)\,x + t\psi(x)$$

Note that $F(x,t)$ is just a linear interpolation between x and $\psi(x)$. It is known from Lemma 44 that $\psi(x)$ is continuous when $x \in \mathcal{S}$. Thus, it follows that $F(x,t)$ is continuous in $x \in \mathcal{S}$.

By definition of $F(x,t)$ it is clear that $F(x,0) = x$. If $a \in CS$ then $\psi(a) = a$ and it follows that $(1-t)\,a + ta = a$. Finally, for $F(x,1)$ it directly holds that $F(x,1) = \psi(x) \in CS$. Therefore, the proof of Theorem 1 is now complete. $\square$

An illustration of the deformation retraction defined in Theorem 1 can be seen in Figure 8.7.

8.3.2 Dimension reduction

Theorem 2. *The cube skeleton of an orthogonal shape reduces its dimension. Specifically, the cube skeleton of an orthogonal polygon or polyhedron is composed of line segments or polygons, respectively, that can be either axis-aligned or rotated $\frac{\pi}{4}$ radians along an orthogonal axis.*

Proof. First, the points of CS that are related as *boundary points of* CS are characterized. Let b be a boundary element of $\partial\mathcal{S}$ and let x be a point of CS such that $b \in \sigma_p(x, \mathcal{S})$ and $\dim(b) < n-1$. Then, there exists at least one boundary element $b_2 \in \partial\mathcal{S}$ such that $b \in \partial b_2$, $\dim(b_2) = n-1$ and $d_\infty(x, b_2) = d_\infty(x, \partial\mathcal{S})$. In addition, observe that there exists a neighborhood N_x of x such that:

$$\exists y \in N_x, b_2 \in \sigma_p(y, \mathcal{S})$$

As a consequence, the points of CS such that the set $\sigma(x, \mathcal{S})$ contains at least one boundary element of dimension lower than $n-1$ must be either in the boundary of other points of CS induced by boundary elements of dimension $n-1$ or the set $\sigma(x, \mathcal{S})$ has been found by recursion. In the following, the different cases of points that do not belong to the boundary points of CS are analyzed.

First, consider the cube skeleton points of an n-dimensional orthogonal shape whose $\sigma(x)$ set contains exactly two boundary elements b_1 and b_2 of dimension $n-1$. For example, two edges of an orthogonal polygon or two faces of an orthogonal polyhedron. The shape of the L_∞ bisector that contains these cube skeleton points is analyzed. Only have two cases depending on the orientation of b_1 and b_2 may arise:

- *The supporting lines or planes of* b_1 *and* b_2 *are parallel.* The cube skeleton points are contained in a line segment, if $n = 2$, or in a plane, if $n = 3$, that is also parallel to b_1 and b_2.
- *The supporting lines or planes of* b_1 *and* b_2 *are not parallel.* The cube skeleton points are contained in a line segment, if $n = 2$, or in a plane, if $n = 3$, that is rotated $\frac{\pi}{4}$ radians along an orthogonal axis.

Notice that in both cases the obtained CS fulfills the properties of Theorem 2. Now, the different cases in which the recursion is used to obtain the set $\sigma(x)$ are studied. This study is done depending on the dimension of the space:

- If $n = 1$, in this case the boundary elements must be of dimension $n-1$ and there cannot be boundary points of CS.
- If $n = 2$, in addition to the case of boundary elements of maximal dimension $n-1$, the recursion gives portions of CS which are obtained from a point sweeping in a perpendicular direction to an orthogonal flat and fulfills the properties of the Theorem 2.
- If $n = 3$, similarly to the two-dimensional case, the sweep in a perpendicular direction to an orthogonal flat also preserves the properties of the Theorem 2.

As a consequence, the rest of the points of CS are in the closure of the aforementioned cases. This concludes the proof of Theorem 2. □

Note that the cube skeleton of an orthogonal polygon or polyhedron can also be composed of points or segments and points, respectively. However, these are degenerate cases which correspond to a segment or polygon described in Theorem 2 that has a length or area of zero, respectively. An illustration of some of the cube skeleton polygon types related in Theorem 2 are shown in Figures 8.8 and 8.9.

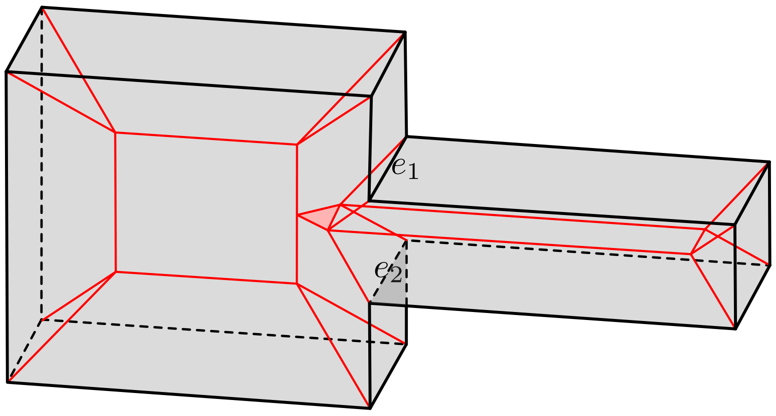


Figure 8.8: A cube skeleton of a simple orthogonal polyhedron is shown in red. The interior of the triangle shaded in red is the set of points whose closest boundary elements are the edges e_1 and e_2. Observe that only the edges bounding the cube skeleton polygons are drawn.

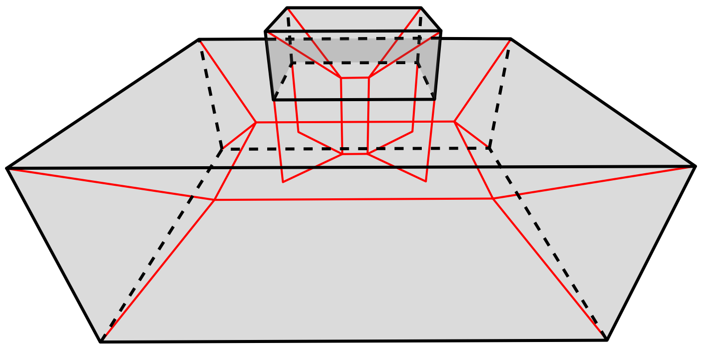


Figure 8.9: A cube skeleton of an orthogonal polyhedron is shown in red. The homotopy equivalence between the cube skeleton and the orthogonal polyhedron is preserved.

In the following, it is proved that the cube axis transform can represent $\mathcal{S}$, analogously to the medial axis transform.

8.3.3 Shape representation

Theorem 3. *The orthogonal shape $\mathcal{S}$ can be reconstructed from its cube transform CT.*

Proof. It is clear that:

$$\mathcal{S} \supset \bigcup_{x \in CS} \mathbb{C}_{x,d_\infty(x,\partial\mathcal{S})}$$

It remains to show that:

$$\mathcal{S} \subset \bigcup_{x \in CS} \mathbb{C}_{x,d_\infty(x,\partial\mathcal{S})}$$

That is, for every point $y \in \mathcal{S}$ there exists a point x such that $y \in \mathbb{C}_{x,d_\infty(x,\partial\mathcal{S})}$. If $y \in CS$ it is obvious that $x = y$ can be selected. If $y \notin CS$, it is shown that if $x = \psi(y)$ is selected the next condition met:

$$\mathbb{C}_{y,d_\infty(y,\partial\mathcal{S})} \subset \mathbb{C}_{\psi(y),d_\infty(\psi(y),\partial\mathcal{S})} \tag{8.6}$$

The maximal cube associated to $\psi(y)$ is empty and contains any maximal cube associated with a point of the segment $[y, \psi(y)]$. Therefore Equation 8.6 is proved and the proof of Theorem 3 is now complete.

□

8.4 The interior cube skeleton of orthogonal polygons

The *interior cube skeleton* of orthogonal polygons, which is a subset of the cube skeleton, homotopy equivalent to $\mathcal{S}$, is introduced. In this section, $\mathcal{S}$ denotes an orthogonal polygon. All the results are based on the simplicial complexes theory [224].

Definition 76. *A* n-simplex *is the n-dimensional convex hull of $n+1$ affinely independent points. The convex hull of any non empty subset of the $n+1$ points that define an n-simplex is called a* face *of the simplex. A simplex s_1 is a* coface *of a simplex s_2 if s_2 is a face of s_1.*

For example, a 0-simplex is a vertex and a 1-simplex is a line segment. The faces of a 1-simplex are its two bounding vertices.

Definition 77. *A* simplicial complex $\mathcal{K}$ *is a set of simplices that satisfies the following conditions:*

- *Any face of a simplex from $\mathcal{K}$ is also in $\mathcal{K}$.*
- *The non empty intersection of any two simplices $\mathcal{K}$ is a face of both simplices.*

Lemma 45. *The cube skeleton of an orthogonal polygon can be represented by a simplicial complex, composed of* 0*-simplices and* 1*-simplices.*

Proof. It is known from Theorem 2 that the cube skeleton of orthogonal polygons is composed of a set of non intersecting line segments. Assume that the segments of the cube skeleton are closed, such that the vertices of $\partial\mathcal{S}$ are also included. Thus, the two-dimensional cube skeleton is a simplicial complex composed of 0-simplices and 1-simplices. □

Definition 78. *Let $\mathcal{K}$ be a simplicial complex, and suppose that s is a simplex in $\mathcal{K}$. s has a* free face *f if f is a face of s and f has no other cofaces. If s and f are removed from $\mathcal{K}$, another simplicial complex is obtained, which is called an* elementary collapse *of $\mathcal{K}$.*

Property 21. *(WHITEHEAD [224]) Two simplicial complexes are homotopy equivalent if they are related by a sequence of elementary collapses.*

Definition 79. *Let $\mathcal{K}$ be the simplicial complex associated to the cube skeleton of an orthogonal polygon. The* interior cube skeleton *is obtained by collapsing all the simplices of K whose intersection with $\partial\mathcal{S}$ is not empty (see Figure 8.10).*

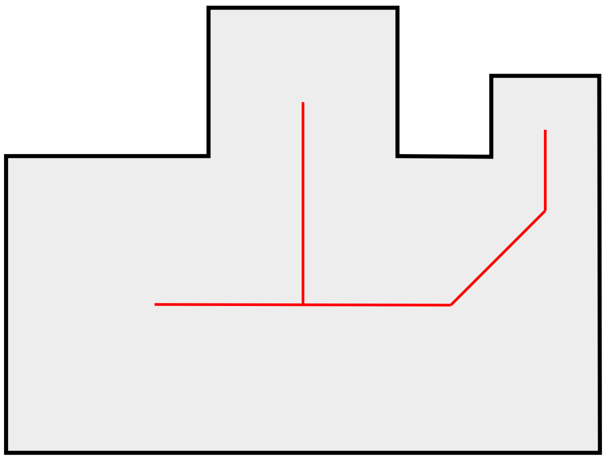


Figure 8.10: Interior cube skeleton of an orthogonal polygon.

Theorem 4. *The cube skeleton and the interior cube skeleton of an orthogonal polygon are homotopy equivalent.*

Proof. Consider a 0-simplex v of $\mathcal{K}$ that is a vertex of $\partial\mathcal{S}$. Observe that v is a convex vertex of the orthogonal polygon and must necessarily be a free face of a 1-simplex e of $\mathcal{K}$. If v and e are removed from $\mathcal{K}$, an elementary collapse is obtained (see Definition 78). Thus, the sequence of collapses proposed in Definition 79 is a valid sequence of elementary collapses and it follows that the two-dimensional CS and the interior CS are homotopy equivalent (see Property 21). □

Observe that the interior cube skeleton of polyhedra could be defined, in analogy to the two-dimensional case, by subdividing the cube skeleton faces into triangles, that are

2-simplices, and collapsing at least the ones that touch the boundary of the orthogonal polyhedron. However, this definition must be carefully done, because the sequence of proper elementary collapses is not trivial. In addition, note that the resulting interior cube skeleton may not be uniquely defined as it depends on the triangulation of the cube skeleton faces.

8.5 Computing the cube skeleton of orthogonal shapes

In this section, algorithms to compute the cube skeleton of orthogonal shapes from their L_∞ Voronoi diagram are introduced. The following definition is required in the next two subsections.

Definition 80. *Let $\mathcal{S}$ be an n-dimensional orthogonal shape and let b be a $n-1$ dimensional boundary element of $\partial\mathcal{S}$. $CS_p(b)$ is a subset of points from the cube skeleton:*

$$CS_p(b) = \{x \in CS : b \in \sigma(x, \mathcal{S})\}$$

Observe that $CS_p(b)$ only contains the parts of the cube skeleton which have not been found by recursion of the set $\sigma(x, \mathcal{S})$ (see Figure 8.13a). The next lemma, about the monotonocity of $CS_p(b)$, has some connection with Lemmas 9 and 19.

Lemma 46. *Let $\mathcal{S}$ be an n-dimensional orthogonal shape and let b be a $n-1$ dimensional boundary element of $\partial\mathcal{S}$. $CS_p(b)$ is monotone with respect to the flat containing b.*

Proof. Let x be an arbitrary point in $CS_p(b)$ and let $y = \operatorname{proj}(x, b)$. It will be shown that:

$$\forall t \in \operatorname{int}([x, y]), \quad t \notin CS_p(b)$$

Consider the n-dimensional cube C_x (see Definition 68) centered at x and with radius $d_\infty(x, \partial\mathcal{S})$. The *facets* of an n-dimensional cube either are its boundary edges (2D) or boundary faces (3D).

Thus, C_x has one of its facets on $\operatorname{flat}(b)$ and its interior does not intersect with $\partial\mathcal{S}$. Assume that there exists $t \in \operatorname{int}([x, y])$ such that $t \in CS_p(b)$. Then $C_t = C_{t,d_\infty(t,\partial\mathcal{S})}$ has also one facet on $\operatorname{flat}(b)$, by using the previous property.

Notice that $C_x \supset C_t$, because $d_\infty(x, \partial\mathcal{S}) > d_\infty(t, \partial\mathcal{S})$.

Moreover, if $t \in CS_p(b)$, then $t \in CS$. Thus, C_t intersects another $n-1$ dimensional boundary element of $\partial\mathcal{S}$, different from b, which intersects with the interior of C_x, giving a contradiction. $\square$

8.5.1 Computing the cube skeleton of orthogonal polygons

In this section, two different algorithms to compute the cube skeleton of orthogonal polygons are presented. In the following, $\mathcal{S}$ denotes an orthogonal polygon and $V_\infty(\mathcal{S})$ is its L_∞ Voronoi diagram (see Chapter 6).

Lemma 47. *The cube skeleton of an orthogonal polygon $\mathcal{S}$ can be extracted from its L_∞ Voronoi diagram.*

Proof. The Voronoi edges are included in the set $E_\infty(\mathcal{S})$, which is analyzed in Section 2. Next, the correspondence between the Voronoi edges and the cube skeleton segments, if it exists, is studied. Let e_v be a Voronoi edge. For a dense set of points $x \in e_v$:

- $|\epsilon(x, \text{sites}(\mathcal{S}))| \geq 2$*, and at least two sites in* $\epsilon(x, \text{sites}(\mathcal{S}))$ *are non-collinear.* The set $\epsilon(x, \partial\mathcal{S})$ must contain two or more one dimensional boundary elements and possibly other vertices on the boundary of these boundary elements. The refined set $\sigma_p(x, \mathcal{S})$ contains at least two non-collinear boundary elements. As $|\sigma_p(x, \mathcal{S}) = 2|$ then $\sigma_p(x, \mathcal{S}) = \sigma(x, \mathcal{S})$. As a consequence, the Voronoi edge coincides with the cube skeleton segment.

- $\epsilon(x, \text{sites}(\mathcal{S})) = \emptyset$. The set $\epsilon(x, \partial\mathcal{S})$ must contain a vertex which is reflex, and cannot contain the two incident polygon edges to the reflex vertex. Thus, x does not have more than one closest boundary element, and it follows that the Voronoi edge does not belong to the cube skeleton (see Figure 8.11b).

- $|\epsilon(x, \text{sites}(\mathcal{S}))| \geq 2$*, and all the sites in* $\epsilon(x, \text{sites}(\mathcal{S}))$ *are collinear.* The Voronoi edge e_v bounds a bisector area (see Property 13), and does not belong to the cube skeleton, as the set $\sigma_p(x, \mathcal{S})$ generally contains only one boundary element that includes a set of collinear polygon edges.

Observe that the cube skeleton segments that coincide with the Voronoi edges are those which are not found by recursion of the set $\sigma(x, \mathcal{S})$. The rest of the parts of the cube skeleton, found by recursion, are called *central segments* (see Figure 8.11a). A central segment is defined by the points such that $\sigma_p(x, \mathcal{S}) = \{e\}$, and $\sigma(x, \mathcal{S})$ contains two vertices on the boundary of e. That central segment is swept perpendicular to e, as defined by the set $\sigma(x, \mathcal{S})$ (see Theorem 2). This is the only possible case of recursion and concludes the proof of Lemma 47. □

Consider a ray with origin in the endpoint, whose set σ_p contains two closest reflex vertices, of a central segment, and that is perpendicular and oriented towards the interior of the one-dimensional boundary element containing both closest vertices. This ray is called the *central swept vertex* (see Figure 8.12a). The remaining endpoint of any central segment can be obtained by computing the intersection of its central swept vertex with the edges of $V_\infty(\mathcal{S})$ that belong to CS.

Two methods to compute the central segments of the cube skeleton, once $V_\infty(\mathcal{S})$ has been computed, are introduced.

Central segment traversal algorithm

Next, an straightforward algorithm to compute the central segments, which is not time-optimal, is provided. As $V_\infty(\mathcal{S})$ has been computed by considering the perturbation technique described in Section 6.5.2, the central segments can be directly derived from the edges of $V_\infty(\mathcal{S})$.

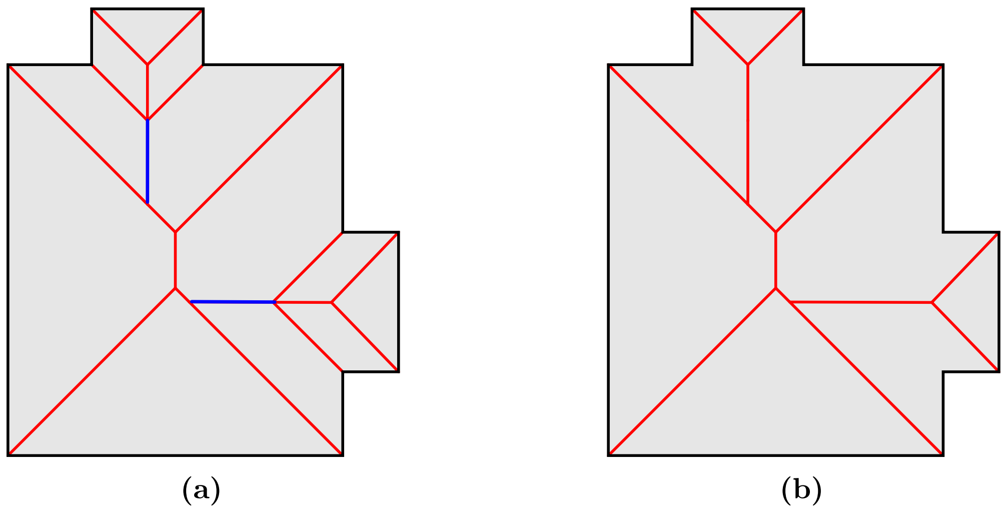


Figure 8.11: Computing the cube skeleton from the L_∞ Voronoi diagram. **(a)** Selection of the central segments of the cube skeleton shown in blue lines (the original computed L_∞ Voronoi diagram is shown in Figure 6.10a). **(b)** Removing the edges of the L_∞ Voronoi diagram that emanate from reflex vertices and do not belong to the cube skeleton.

Lemma 48. *The central segments of the cube skeleton of $\mathcal{S}$ can be straightforwardly computed from $V_\infty(\mathcal{S})$ by the central segment traversal algorithm in time $\theta(n^2)$ and $O(n)$ space, where n is the number of vertices of $\mathcal{S}$.*

Proof. Consider a Voronoi edge e_v of $V_\infty(\mathcal{S})$ bounding a bisector area (see Property 13). Consider also the central swept vertex, with origin in a vertex of the edge e_v, which is L_∞ equidistant to two reflex vertices of the polygon. The remaining endpoint of this central segment is found by traversing the Voronoi edges, until the Voronoi edge intersected by the central swept vertex is found (see Figure 8.12). There can be at most $O(n)$ central segments. Retrieving the intersection of the central swept vertex and the edges of the Voronoi diagram can be done in $O(n)$, as there are at most $O(n)$ Voronoi edges L_∞ equidistant to an edge of $\mathcal{S}$. Thus the worst-case time complexity is $\theta(n^2)$. $\square$

Although the time complexity to compute the central segments with the traversal algorithm is quadratic, in practice a little number of edge traversals are performed to compute the Voronoi edge intersected by the central swept vertex. Experimental results (see Section 8.6) confirm that the traversal algorithm always takes a negligible amount of time compared with the computation of the L_∞ Voronoi diagram, although its theoretical worst-case time complexity is higher.

Central segment binary search algorithm

A second strategy to compute the central segments, with a better theoretical time complexity than the traversal algorithm, is introduced. Assume that all the Voronoi edges that bound a bisector area have been removed (see Lemma 47).

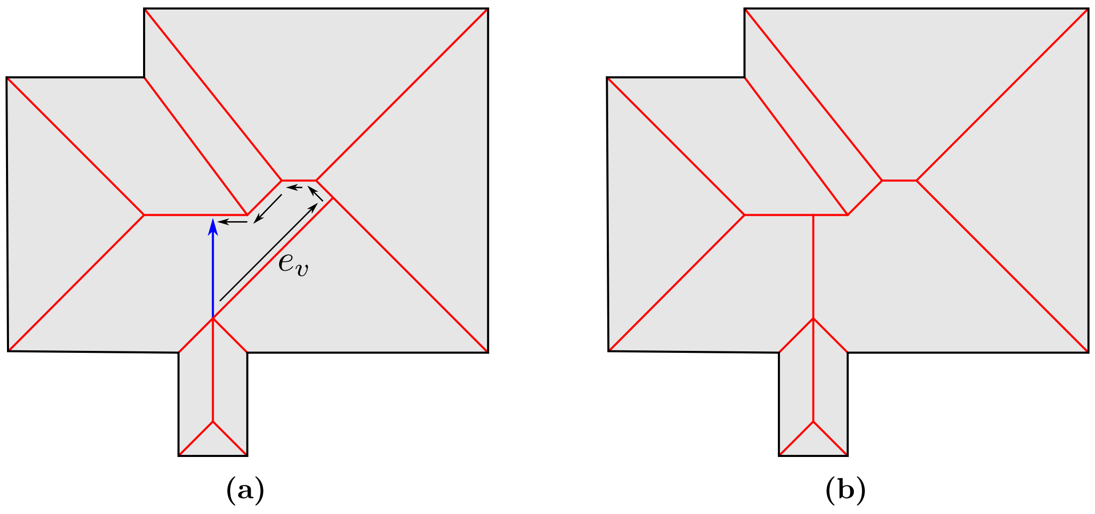


Figure 8.12: Illustration of the traversal algorithm to compute the central segments of the cube skeleton. **(a)** Central swept vertex shown in blue. The Voronoi edges are traversed in the direction of the central swept vertex until one is intersected by the central swept vertex. The black arrows indicate the direction of the edge traversal. **(b)** The edge e_v bounding the bisector area do not belong to the cube skeleton.

Let e be a one-dimensional boundary element of $\partial\mathcal{S}$. Then, $CS_p\left(e\right)$ (see Definition 80) is a set of segments of $V_\infty\left(\mathcal{S}\right)$ (see Lemma 47). Observe that the central swept vertex induced by any pair of reflex vertices of ∂e must intersect $CS_p\left(e\right)$.

Lemma 49. *The central segments of the cube skeleton of $\mathcal{S}$ can be computed from $V_\infty\left(\mathcal{S}\right)$ by a binary search algorithm in time $O\left(n\log n\right)$ and $O\left(n\right)$ space, where n is the number of vertices of $\mathcal{S}$.*

Proof. Let e be a one-dimensional boundary element of $\mathcal{S}$ that has at least one central swept vertex associated to it. The central segments are computed by searching which segment of $CS_p\left(e\right)$ is intersected by the central swept vertices. As the set $CS_p\left(e\right)$ is monotone with respect to the line containing e (see Lemma 46), it is possible to project them onto flat $\left(e\right)$ and obtain a set of disjoint one-dimensional intervals (see Figure 8.13b). Let p be the orthogonal projection of a central swept vertex to the line containing e. The intersection of the central swept vertex can be computed by searching which interval contains p. If the intervals are stored in a binary tree their insertion requires $O\left(k\log k\right)$ time, and searching which interval contains a point requires $O\left(\log k\right)$ time, where k is the number of vertices of $CS_p\left(e\right)$. Thus, as there are at most $O\left(n\right)$ central segments, the binary search algorithm has $O\left(n\log n\right)$ time and $O\left(n\right)$ space complexity. □

Theorem 5. *The cube skeleton of an orthogonal polygon with n vertices can be computed in $O\left(n\log n\right)$ time and $O\left(n\right)$ space complexity.*

Proof. The L_∞ Voronoi diagram of an orthogonal polygon can be computed in $O\left(n\log n\right)$ time and n space (see Lemma 10). The Voronoi edges emanating from the reflex vertices of the orthogonal polygon and bounding bisector areas, which do not belong to the cube skeleton (see Lemma 47), can be erased in $O(n)$ time. The central segments can be computed in $O(n \log n)$ time and n space (see Lemma 49). □

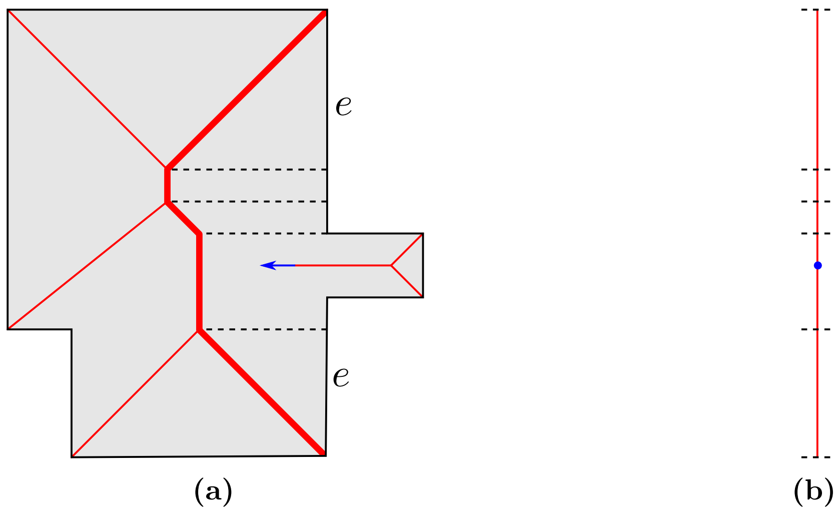


Figure 8.13: Illustration of the binary search algorithm to compute the central segments. The $n-1$ dimensional boundary element e contains two collinear polygon edges. **(a)** The set $CS_p(e)$, shown in thick lines, is monotone with respect to the line containing e. Central swept vertex shown in blue. **(b)** The projection of $CS_p(e)$ onto the line containing e, that is flat(e), induce a set of disjoint intervals. The interval that contains the projected origin x of the central swept vertex is the edge intersected by the central swept vertex.

The central segment binary search algorithm has a better theoretical time complexity than the traversal algorithm, but it is expected in practice to be probably less efficient. This is due to the fact that it may require the initialization of several interval binary trees, while the traversal algorithm does not require any data structure initialization. Finally, observe that the cube skeleton also could be computed by appropriately treating the central swept vertices as a special event of the two-dimensional sweep line algorithm (see Chapter 6).

8.5.2 Computing the cube skeleton of orthogonal polyhedra

A practical algorithm to compute the cube skeleton of orthogonal polyhedra is presented. In this section, $\mathcal{S}$ denotes an orthogonal polyhedron and $V_\infty(\mathcal{S})$ is its L_∞ Voronoi diagram (see Chapter 7). As with orthogonal polygons, the cube skeleton of orthogonal polyhedra can be extracted from $V_\infty(\mathcal{S})$. Unfortunately, it will be seen that some parts of the three dimensional cube skeleton are not included into the L_∞ Voronoi diagram.

Lemma 50. *The cube skeleton of an orthogonal polyhedron can be extracted from its L_∞ Voronoi diagram.*

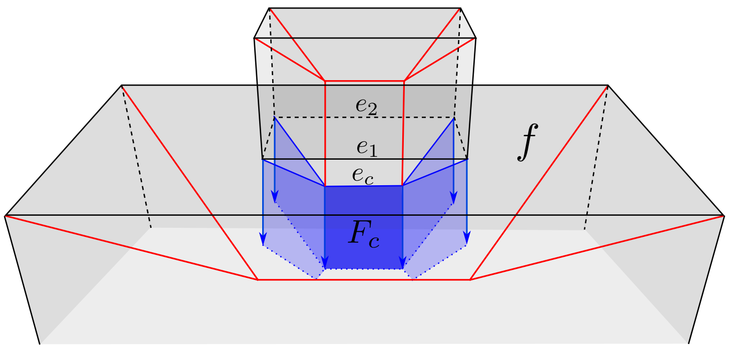


Figure 8.14: Illustration of Lemma 50. A portion of or an orthogonal polyhedron with a face hole. Central faces are shown in blue. The endpoints of the segment bounding the central swept segments are shown with blue arrows. The central part F_c highlighted in a dark blue tone, is associated to the edges e_1 and e_2 and swept perpendicular to the face f, such that $\sigma_p(x, \mathcal{S}) = \{f\}$ and $\sigma(x, \mathcal{S}) = \{e_1, e_2\}$. e_c is the Voronoi edge that coincides with the central swept segment.

Proof. The Voronoi faces are included in the set $E_\infty(\mathcal{S})$, which is analyzed in Section 2. Next, the correspondence between the Voronoi faces and the cube skeleton faces, if it exists, is studied. Let f_v be a Voronoi face. For a dense set of points $x \in f_v$:

- $|\epsilon(x, \text{sites}(\mathcal{S}))| \geq 2$, *and at least two sites in* $\epsilon(x, \text{sites}(\mathcal{S}))$ *are non-coplanar.* In this case, there is a direct equivalence between the Voronoi face and the face of the cube skeleton. The set $\sigma_p(x, \mathcal{S})$ must contain two boundary elements of dimension two and it follows that the set $\sigma(x, \mathcal{S})$ is not defined by recursion. That is $\sigma_p(x, \mathcal{S}) = \sigma(x, \mathcal{S})$. As a consequence, the Voronoi face coincides with the cube skeleton face.

- $\epsilon(x, \text{sites}(\mathcal{S})) = \emptyset$. This Voronoi face does not belong to the cube skeleton because the set $\sigma(x, \mathcal{S})$ contains a single one dimensional boundary element e that includes a reflex edge, and $\text{proj}(x, e) \notin \text{inter}(e, \mathcal{S})$ such that $|\sigma(x, \mathcal{S})| = 1$.

- $|\epsilon(x, \text{sites}(\mathcal{S}))| \geq 2$, *and all the sites in* $\epsilon(x, \text{sites}(\mathcal{S}))$ *are coplanar.* The Voronoi face f_v bounds a bisector volume (see Property 20). This Voronoi face does not belong to the cube skeleton because the set $\sigma_p(x, \mathcal{S})$ will, in general, only contain a single two dimensional boundary element, which encloses the set of polyhedron coplanar faces that induce the bisector volume.

Analogously to the two-dimensional case, the cube skeleton faces that coincide with the Voronoi faces are those which are not found by recursion of the set $\sigma(x, \mathcal{S})$. In addition, there exists parts of the cube skeleton that are not a Voronoi face, which are called *central parts*. It will be shown that the central parts can be computed from the L_∞ Voronoi diagram by retrieving the intersection between some swept segments, and the rest of Voronoi faces that coincide with the cube skeleton. Let F_c be a central part. For all points $x \in F_c$ the next cases, depending on the recursion of the set $\sigma(x, \mathcal{S})$, are distinguished:

- $\sigma_p(x, \mathcal{S}) = \{f\}$ and $\sigma(x, \mathcal{S}) = \{e_1, e_2\}$ (see Figure 8.14). In this case, the set $\sigma(x, \mathcal{S})$ have been found in the first recursion step. Thus, both one dimensional boundary elements e_1 and e_2 belong to the boundary of the two dimensional boundary element f, that is $e_1, e_2 \in \partial f$. The central part F_c is the part of $\sigma\left(\operatorname{proj}(x, f), \operatorname{inter}(f, \mathcal{S})\right)$ such that $\sigma(x, \mathcal{S}) = \{e_1, e_2\}$, that is swept in the direction of the orthogonal axis perpendicular to f. One edge bounding the central part F_c corresponds to a Voronoi edge e_c, which is called *central swept segment*. More precisely, e_c coincides with the set of points in the boundary of F_c such that $\sigma(x, \mathcal{S}) = \sigma_p(x, \mathcal{S}) = \{e_1, e_2\}$, where e_1 and e_2 are reflex polyhedron edges. Thus, the central part F_c is computed by retrieving the intersection between the swept Voronoi edge e_c and the Voronoi faces included in $CS_p(f)$.

- $\sigma_p(x, \mathcal{S}) = \{f\}$, $\sigma_p(x, \operatorname{inter}(f, \mathcal{S})) = \{e\}$ and $\sigma(x, \mathcal{S}) = \{v_1, v_2\}$ (see Figure 8.15b). In this case, the set $\sigma(x, \mathcal{S})$ has been found in the second recursion step. Thus, $v_1, v_2 \in \partial e$ and $v_1, v_2, e \in \partial f$. F_c is the part of $\sigma\left(\operatorname{proj}(x, f), \operatorname{inter}(f, \mathcal{S})\right)$ such that $\sigma(x, \mathcal{S}) = \{v_1, v_2\}$, that is swept in the direction of the orthogonal axis perpendicular to f. In this case, the central swept segment e_c has one endpoint in the Voronoi vertex v_c bounding F_c such that $\sigma_p(x, \mathcal{S}) = \{e\}$. Consider that v_c is swept in the direction of the orthogonal axis perpendicular to $\tau(e) \setminus \tau(f)$. The intersection of the projected swept vertex and the Voronoi edges $CS_p(e)$ (of the orthogonal polygon $\operatorname{inter}(f, \mathcal{S})$) defines the remaining endpoint of e_c. Thus, anagously to the previous case, the central part F_c is computed by retrieving the intersection between the swept segment e_c and the Voronoi faces in $CS_p(f)$.

- $\sigma_p(x, \mathcal{S}) = \{e\}$ and $\sigma(x, \mathcal{S}) = \{v_1, v_2\}$ (see Figure 8.16). In this case, $\sigma(x, \mathcal{S})$ has been found in the first recursion step. Thus $v_1, v_2 \in \partial e$. These points must correspond to a segment bounding a central part of the previous case.

No more cases of cube skeleton recursion, besides of those already mentioned, exist. This concludes the proof of Lemma 50. □

Observe that the set CS_p only contains the aforementioned parts of the cube skeleton which have a direct correspondence with the L_∞ Voronoi diagram. Next, an efficient algorithm to compute the central parts of the cube skeleton from $V_\infty(\mathcal{S})$ is introduced. Thanks to Lemma 46, the problem of computing the intersection of three-dimensional swept segments and vertices can be reduced to a two-dimensional point location and segment intersection problem.

Lemma 51. *(KIRKPATRICK [137]) Given a planar subdivision with n line segments and a query point p, the region of the subdivision that contains p can be determined in $O(\log n)$ time, $O(n)$ space, and $O(n)$ pre-processing time.*

Lemma 52. *(BALABAN [32]) Given a set of n line segments in the plane, it is possible to report all k intersections between the segments in $O(n \log n + k)$ time and $O(n)$ space.*

Lemma 53. *The central parts of the cube skeleton of $\mathcal{S}$ can be computed from $V_\infty(\mathcal{S})$ in time $O(n \log n + k)$ and $O(n)$ space, where n is the number of vertices of $V_\infty(\mathcal{S})$ and k is the number of vertices of the central parts.*

Proof. Let f be a two-dimensional boundary element of $\partial\mathcal{S}$ with at least one central swept segment or vertex associated to it (see Lemma 50). $CS_p(f)$ correspond to a set of faces of $V_\infty(\mathcal{S})$. The faces $CS_p(f)$ are monotone with respect to the plane containing f (see Lemma 46) and it is possible to project them onto the plane $\text{flat}(f)$ and obtain a planar subdivision (see Figure 8.15). As a consequence, computing the intersection of the endpoints of central segments with these Voronoi faces is equivalent to the planar point location problem (see Lemma 51). As there are at most $O(n)$ central parts, the overall time complexity of this step is $O(n \log n)$ time and $O(n)$ space.

Finally, computing the remaining segments that bound each central part is equivalent to report all the intersections between a set of line segments, which are the non-intersecting segments associated to the projected planar subdivision of $CS_p(f)$ plus the set of query segments induced by the central swept segments (see Figure 8.15). This second step can be done in $O(n \log n + k)$ time and $O(n)$ space (see Lemma 52) and dominates the overall time complexity to compute the central parts. □

Lemma 54. *The cube skeleton of an orthogonal polyhedron $\mathcal{S}$ can be computed from $V_\infty(\mathcal{S})$ in $O(n \log n + k)$ time and $O(n)$ space complexity, where n is the number of vertices of $V_\infty(\mathcal{S})$ and k is the number of vertices of the central parts.*

Proof. The Voronoi faces emanating from reflex edges of the orthogonal polyhedron and bounding bisector volumes, which do not belong to the cube skeleton (see Lemma 50), can be erased in $O(n)$ time. The central parts can be computed in $O(n \log n + k)$ time and n space (see Lemma 53). □

8.6 Experimental results

The traversal algorithm to compute the cube skeleton of orthogonal polygons (see Section 8.5.1) has been evaluated. The algorithm is implemented in C++ and its source code consists of about two hundred lines. The experiments were conducted with the same database of orthogonal polygons used in Section 6.7 (see Figure 8.17). Although the worst-case theoretical complexity of the algorithm is quadratic, experimental results suggest that is in average linear with respect to the number of central segments. More specifically, the introduced approach revealed an average runtime of $18.7\, n : \mu s$. The average overhead time to compute the cube skeleton, with respect to the computation of the L_∞ Voronoi diagram, was of $1 - 2\%$. Figure 8.18 and 8.19 show the interior cube skeleton of some orthogonal polygons extracted from cube skeleton. As a future work it remains to implement the algorithm to compute the cube skeleton of orthogonal polyhedra.

8.7 Conclusions

A novel skeletal representation of orthogonal shapes, defined according to the L_∞ metric, has been presented. The cube skeleton shares some important properties with the medial axis (homotopy equivalence, dimension reduction and shape representation) while being only composed of segments and polygons of restricted orientation. The interior cube skeleton of orthogonal polygons retains the properties of the cube skeleton and is composed of less segments, which do not intersect the shape boundary.

Algorithms to compute the cube skeleton of orthogonal shapes have been introduced. Experimental results over several datasets reveal that the overhead to extract the two-dimensional cube skeleton from the L_∞ Voronoi diagram is small. Moreover, there is little empirical evidence that the central segment binary search algorithm is faster than the traversal one.

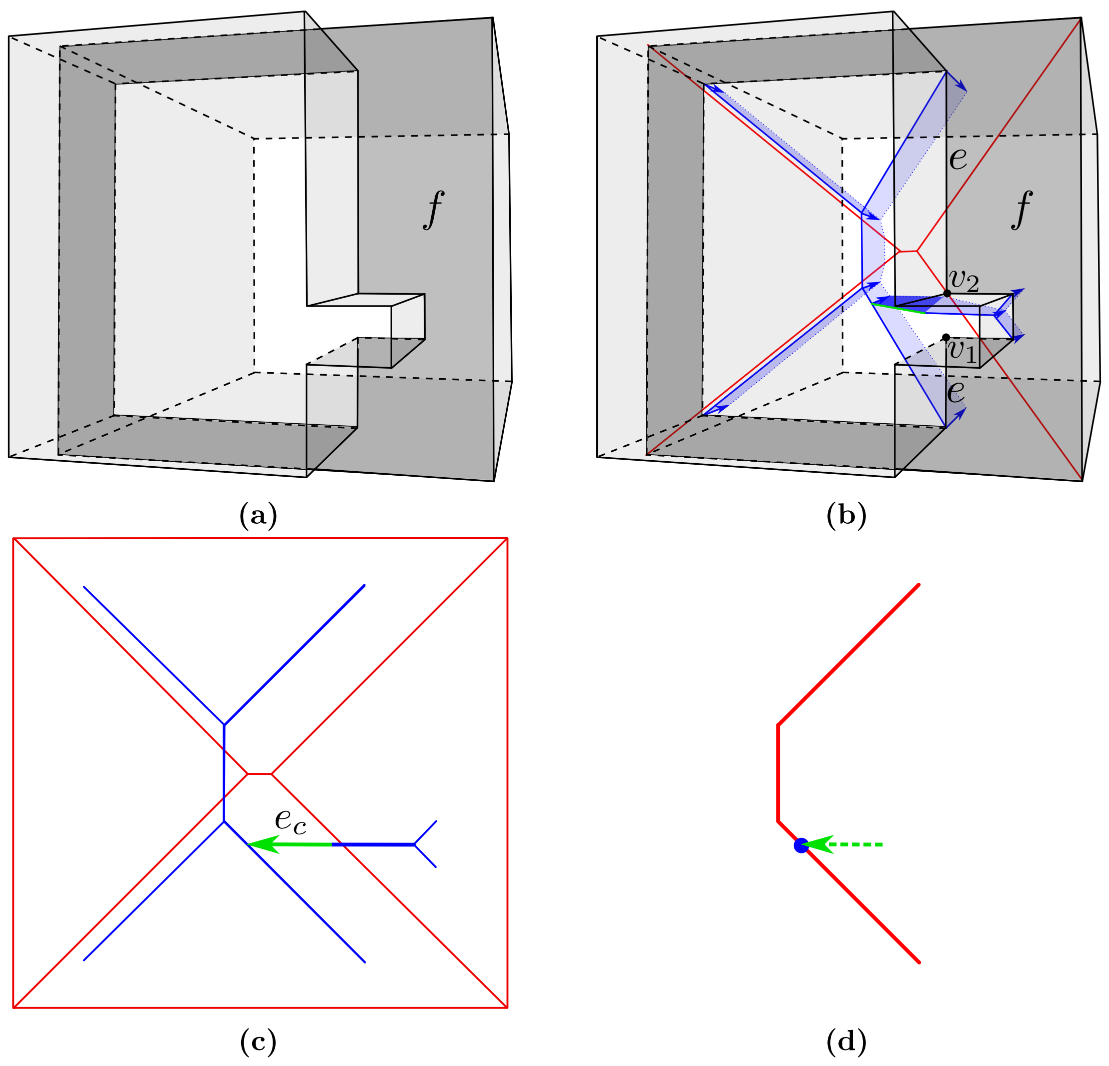


Figure 8.15: Illustration of Lemma 50. Computing the cube skeleton of an orthogonal polyhedron from the L_∞ Voronoi diagram. **(a)** An orthogonal polyhedron with a face f, highlighted in a dark tone, that has a hole with two collinear edges. **(b)** Central faces are shown in blue. The endpoints of the segment bounding the central swept segments are shown with blue arrows. The central part highlighted in a dark blue tone, is associated to the vertices v_1 and v_2 and swept perpendicular to the face f, such that $\sigma_p(x, \mathcal{S}) = \{f\}$ and $\sigma(x, \mathcal{S}) = \{v_1, v_2\}$. The swept segment that induces the central part is shown in green. **(c)** The projection of $CS_p(f)$ onto the plane containing f induces a planar subdivision that is shown in red. The central swept-segments projected onto f are straight line segments shown in blue. **(d)** The projection of $CS_p(e)$ onto the plane containing f induces a monotone polygonal chain that is shown in red. The central swept vertex projected onto e is the point shown in blue.

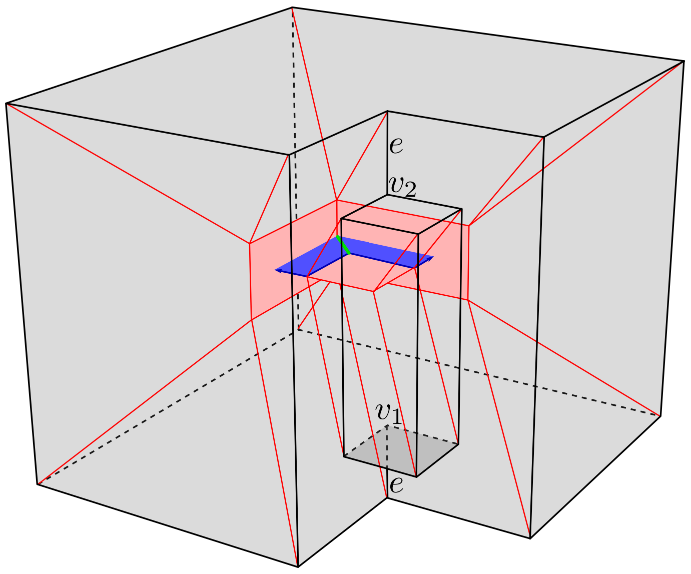


Figure 8.16: Illustration of Lemma 50. A simple orthogonal polyhedron. Voronoi diagram edges shown in red. For illustrative purposes only the Voronoi faces that do not intersect the polyhedron boundary, are shown in red. The central parts induced by central swept segments are shown in blue. The segment with points such that $\sigma_p(x, \mathcal{S}) = \{e\}$ and $\sigma(x, \mathcal{S}) = \{v_1, v_2\}$ bound two central parts and is shown in green.

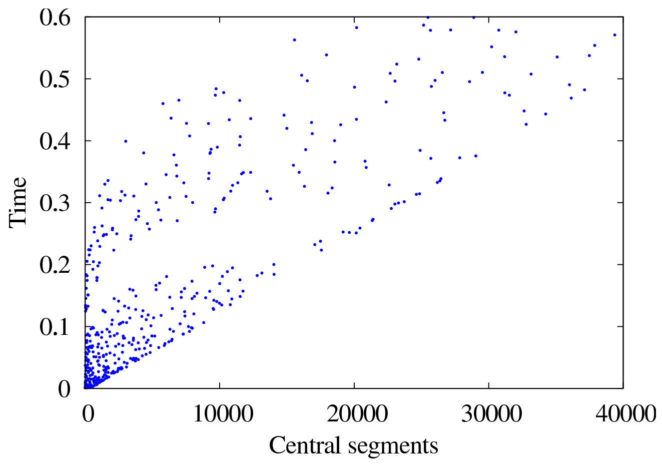


Figure 8.17: Running times to compute the central segments of an orthogonal polygon. The abscissa denotes the number of central segments. The ordinate denotes the running time in seconds. Every blue point depicts the runtime for a single dataset.

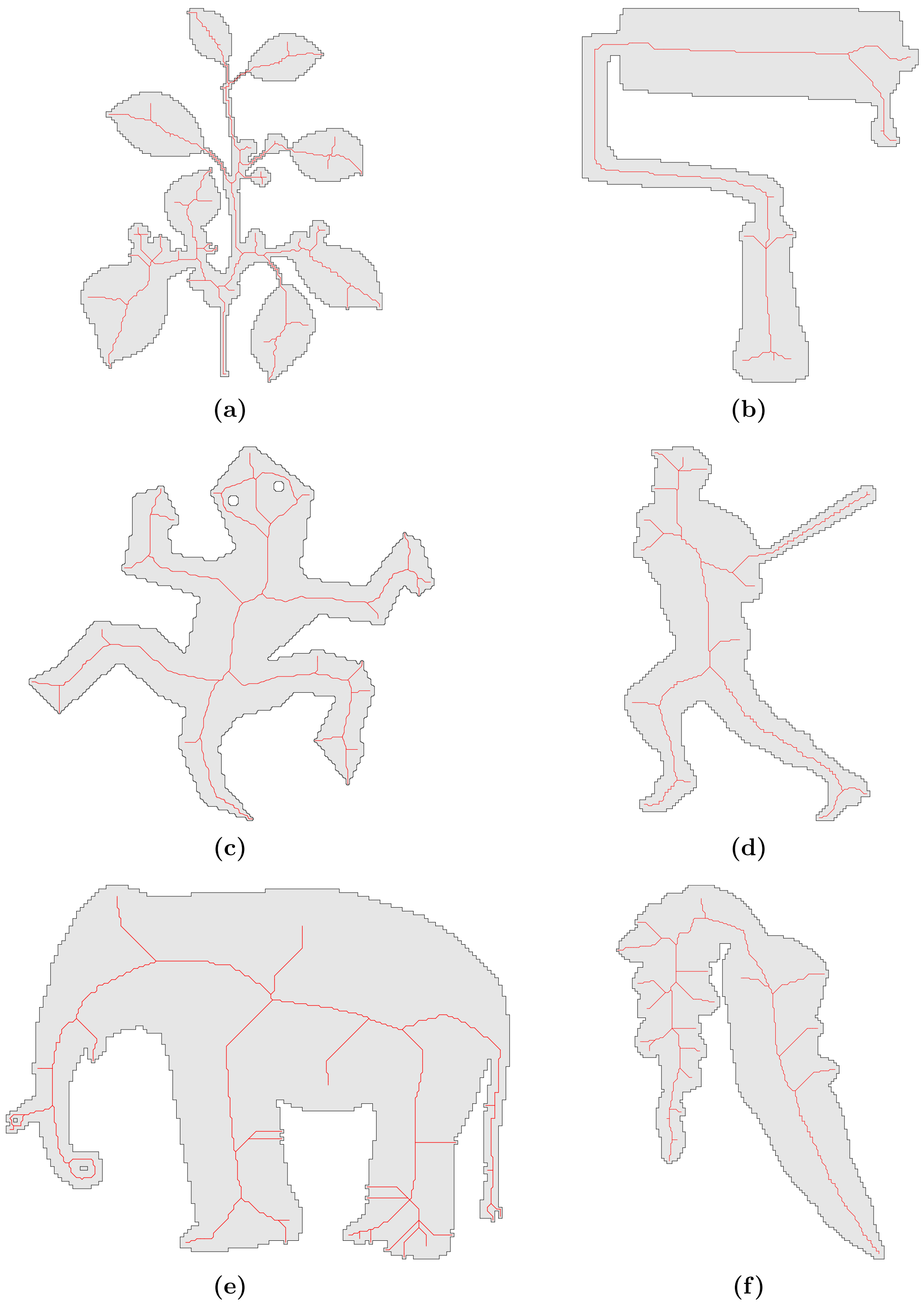
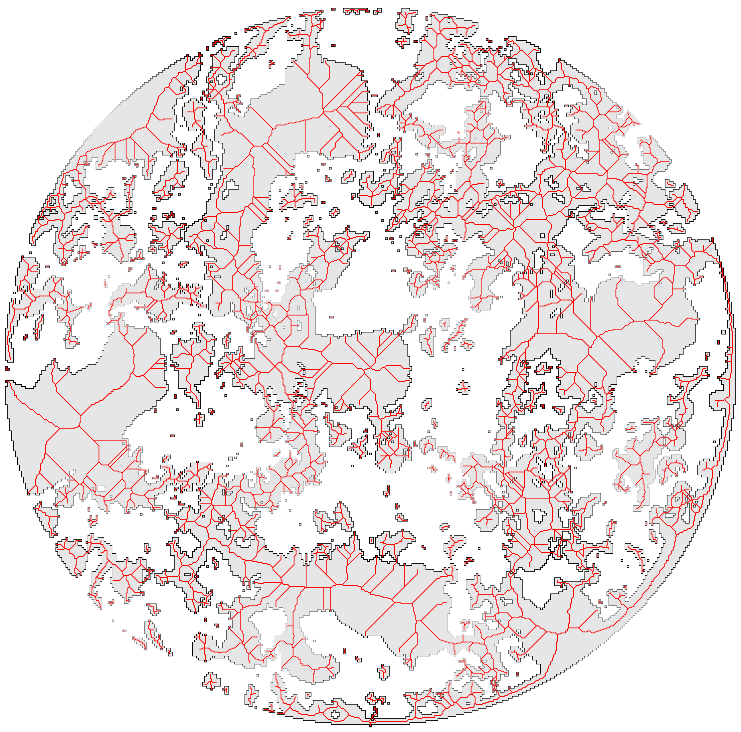
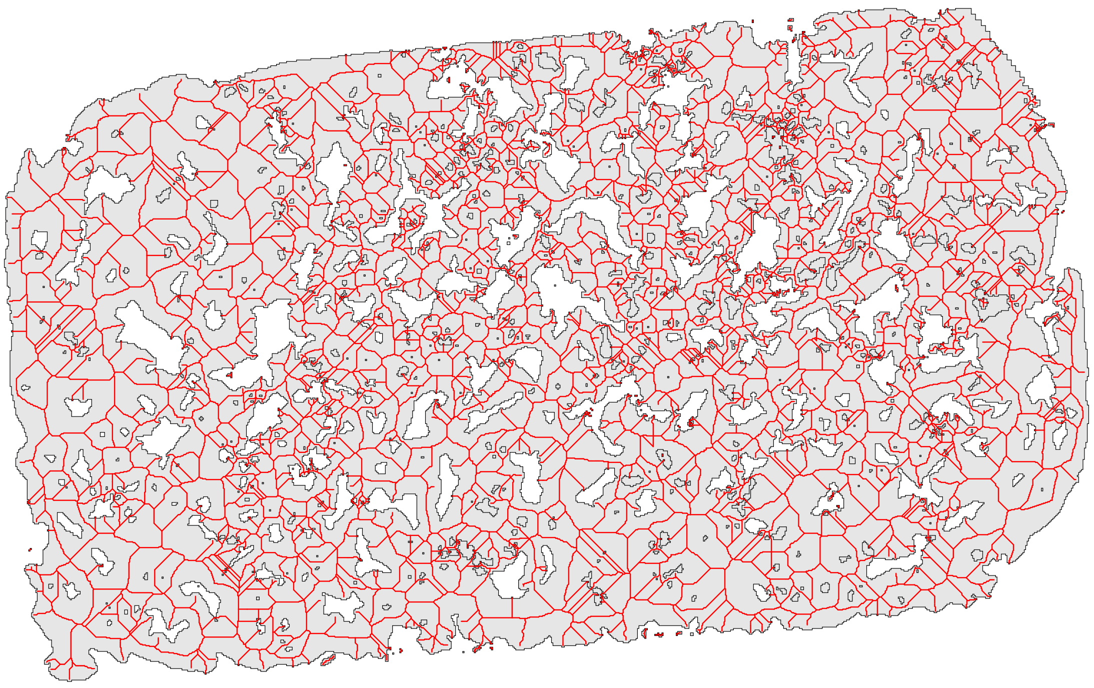


Figure 8.18: Six simple orthogonal polygons and interior cube skeleton shown in red. Shapes are extracted from the database of binary shapes collected by the LEMS Vision Group at Brown University `http://www.lems.brown.edu/~dmc/main.html`



(a)



(b)

Figure 8.19: A biomaterial sample **(a)**, a section of a rock **(b)** and its interior cube skeleton shown in red.

Chapter 9

The scale cube skeleton

Contents

9.1 Introduction

Like the medial axis, the cube skeleton (see Chapter 8) has a high instability under perturbations of the shape. In this chapter, a novel skeletal representation of orthogonal shapes, called the *scale cube skeleton*, is presented. The scale cube skeleton, that is based upon the scale axis transform of Giesen et al. [107] (see Section 3.1.4), tries to overcome these instability problems of the cube skeleton by defining a simplification scheme where shape features are ignored first if they are small relative to their neighborhood. An algorithm to compute the scale cube skeleton of orthogonal polygons is also presented. In addition, the scale cube skeleton of two-dimensional shapes is compared with other relevant skeletal structures and it is shown its ability to obtain more simple and stable skeletons. This chapter is an extended and adapted version of the second part of the article [158].

9.2 Scale cube skeleton definition

In this section, the scale cube skeleton and scale cube transform of orthogonal shapes are introduced.

Definition 81. *Let $\mathcal{S}$ be an orthogonal shape. When $s > 0$, the multiplicatively s-scaled shape of $\mathcal{S}$ is:*

$$\mathcal{S}_s = \bigcup_{\mathbb{C}_{c,r} \in CT} \mathbb{C}_{c,sr}$$

It will be shown in Lemma 58 that when $s \geq 1$, $\mathcal{S}_s$ is an orthogonal shape. The next definition assumes this property.

Definition 82. *(**Scale cube skeleton**) When $s \geq 1$, the* s-scale cube skeleton *CS_s is the cube skeleton (see Definition 67) of $\mathcal{S}_s$ and CT_s is the cube transform (see Definition 69) of $\mathcal{S}_s$.*

Definition 83. *(**Scale cube transform**) When $s \geq 1$, the* scale cube transform *of $\mathcal{S}$ is:*

$$SCT_s = \left\{ \mathbb{C}_{c,r/s} : \mathbb{C}_{c,r} \in CT_s \right\}$$

The scale cube skeleton is able to compute a family of simplified skeletal representations. Bigger cubes of the cube transform tend to cover the smaller ones, as they are grown according to its radius. Bigger cubes are associated with features of the input shape that are more relevant. Thus, the scaling operation tends to preserve relevant features of $\mathcal{S}$, while small cubes are associated to less relevant features and are rapidly covered by bigger cubes. This simplification scheme is expected to produce skeletons with less features: as the scaled shape is more likely to be simpler than the original shape, its skeleton will also be simpler than the skeleton of the original shape.

9.3 Scale cube skeleton properties

Some properties of the scaled shape are shown below. For this purpose, the structure of the union of cubes induced by the cube transform is analyzed.

Lemma 55. *Let $\mathcal{S}$ be an orthogonal polygon and $s \geq 1$. $\mathcal{S}_s$ can be represented by the union of a finite set of rectangles.*

Proof. According to Theorem 2, the next types of segments from CS are identified:

- *A segment defined by two parallel edges of $\partial\mathcal{S}$.* The set of squares from CT associated to the segment can be bounded by an axis-aligned rectangle. Note that this rectangle is the convex hull of the two CT squares associated to the segment endpoints. Growing this rectangle is equivalent to the growth of all the CT squares of the segment. One of the sides of this rectangle has a length equal to the uniform distance from the segment points to both edges of $\partial\mathcal{S}$ multiplied by the scale factor. The other side, must have a length equal or higher.

- *A segment defined by two non parallel edges of $\partial\mathcal{S}$.* In this case, it suffices to consider the biggest square of the two squares associated to the endpoints of the segment. This square encloses all the squares associated with the segment when $s \geq 1$.

- *A segment defined by two vertices of $\partial\mathcal{S}$ that lie on the boundary of a one-dimensional boundary element.* Both vertices of $\partial\mathcal{S}$ must be on the boundary of two different collinear edges. As in the previous case, the biggest square of the two segment endpoints covers all the other squares when grown. This square is associated with the farthest segment endpoint with respect to the one-dimensional boundary element.

□

Figure 9.1a shows the rectangles associated with some of the segment cases related in Lemma 55.

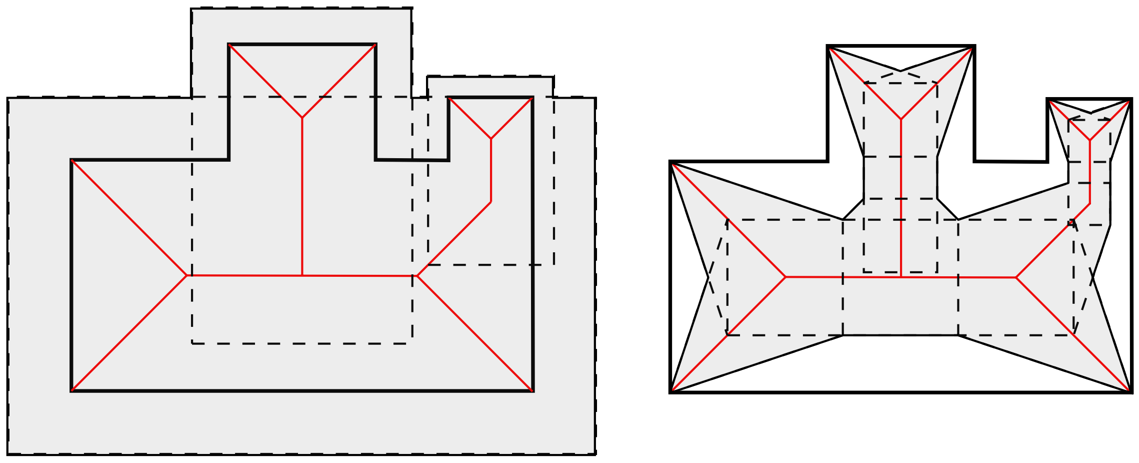


Figure 9.1: **(a)** The s-scaled shape when $s < 1$ is found by obtaining the boundary of the union of rectangles, shown in dashed lines. **(b)** The s-scaled shape when $s < 1$ is found by obtaining the boundary of the union of rectangles and convex polygons, shown in dashed lines.

Definition 84. *A* monotone orthogonal polyhedron *is an orthogonal polyhedron obtained by extruding an orthogonal polygon in a single coordinate.*

Lemma 56. *Let $\mathcal{S}$ be an orthogonal polyhedron and $s \geq 1$. $\mathcal{S}_s$ can be represented by a finite set of monotone orthogonal polyhedra and axis-aligned boxes.*

Proof. A similar proof as Lemma 55 is followed. According to Theorem 2 the next types of polygons from CS appear:

- *A polygon defined by two parallel faces of $\partial\mathcal{S}$.* The union of all the cubes of CT associated to the polygon induce a monotone orthogonal polyhedron. This orthogonal polyhedron is just an orthogonal polygon extruded in the direction of the coordinate aligned with the polygon of CS. The length of the extruded polyhedron is equal to the uniform distance from the polygon to both parallel faces of $\partial\mathcal{S}$ multiplied by the scale factor.

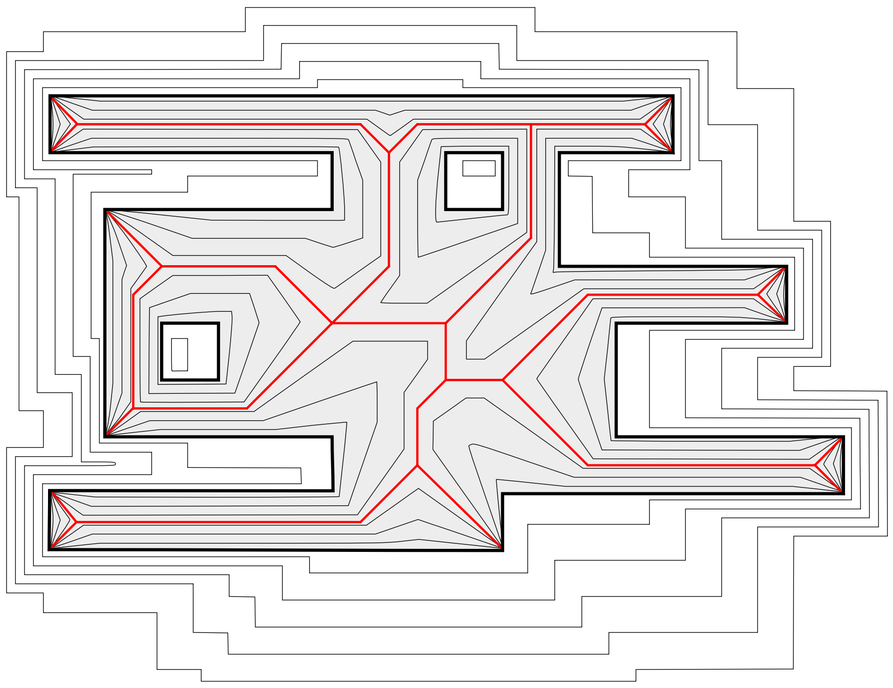


Figure 9.2: Different s-scaled shapes of an orthogonal polygon. Note that when $s > 1$ the scaled shapes are orthogonal polygons, while when $s < 1$ they are not orthogonal polygons.

- *A polygon defined by two non parallel faces of $\partial\mathcal{S}$.* In this case, the bigger cubes of CT must be in the edges of the polygon. Then, there exist two different cases of edges: an edge whose points have a uniform coordinate in an orthogonal direction or the edges without this property. In the first case, the cubes of CT associated to the edge can be simply bounded by an axis-aligned box. In the second case, it is enough to consider the biggest cube of one of the endpoints of the edge.

- *A segment defined by two vertices or two edges of $\partial\mathcal{S}$ that lie on the boundary of a one or two-dimensional boundary element, respectively.* As in the previous case, the biggest cubes of CT are found on the edges of the polygon. In particular, these edges are the farthest to the one or two-dimensional boundary element that contains both vertices or edges from $\partial\mathcal{S}$, respectively.

$\square$

A characterization of the three-dimensional scaled shape by using axis-aligned boxes is provided. It is more desirable to compute the union of axis-aligned boxes than to compute the union of a monotone orthogonal polyhedra.

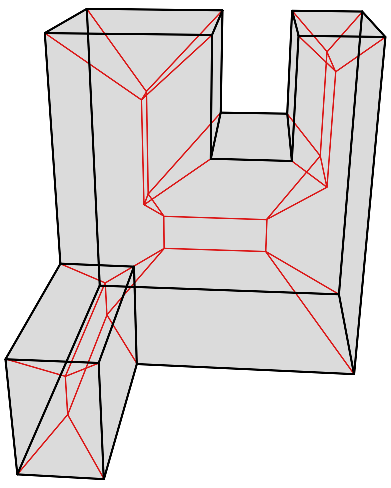
Lemma 57. *Let $\mathcal{S}$ be an orthogonal polyhedron and $s \geq 1$. $\mathcal{S}_s$ can be represented by a finite set of axis-aligned boxes.*

Proof. It is known from Lemma 56 that the shape $\mathcal{S}_s$ can be represented by a finite set of monotone orthogonal polyhedra and axis-aligned boxes. In addition, a monotone orthogonal polyhedron can be divided into a finite set of axis-aligned boxes. It suffices to partition the orthogonal polygon associated to the monotone orthogonal polyhedron in a set of rectangles. Every rectangle induces an axis-aligned box and all the axis-aligned boxes cover the extruded orthogonal polyhedron. □

Figure 9.3 shows the axis-aligned boxes associated with some of the polygon cases related in Lemma 56.

Lemma 58. *Let $\mathcal{S}$ be an orthogonal shape and $s \geq 1$. $\mathcal{S}_s$ is an orthogonal shape.*

Proof. The boundary of the union of axis-aligned rectangles (see Lemma 55) or boxes (see Lemma 57) must be enclosed by axis-aligned edges or faces, respectively. As $\mathcal{S}_s$ is an open set, it is possible to obtain a set of boundary elements that enclose S_s, such that S_s is considered to be an orthogonal shape. □


(a)

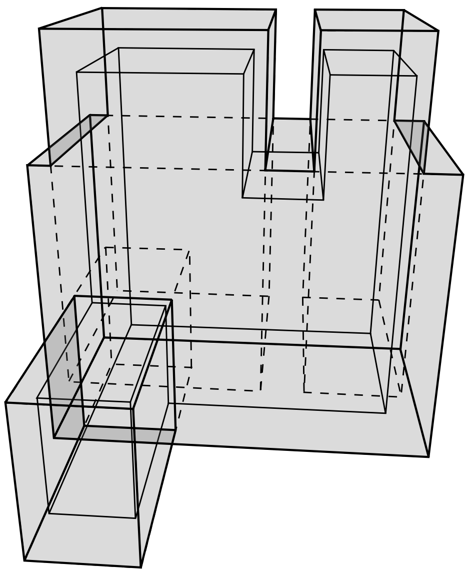

(b)

Figure 9.3: **(a)** Simple polyhedron and its cube skeleton. **(b)** The s-scaled shape when $s > 1$ is found by obtaining the union of axis-aligned boxes, shown in dashed lines.

When $0 < s < 1$, the s-scaled shape is not an orthogonal polygon (see Figure 9.2). In two and three dimensions, the shrunk shape will be a polygon or polyhedron, respectively. The shrunk shape converges to the cube skeleton as the scale factor approaches zero. Thus, the cube skeleton also serves as a skeletal representation of the shrunk shape.

9.4 Computing the scale cube skeleton of orthogonal polygons

A practical algorithm to compute the scale cube skeleton of orthogonal polygons is presented. In this section $\mathcal{S}$ denotes an orthogonal polygon. The presented algorithm requires as input the boundary of the orthogonal polygon and the scale value $s > 1$. The general steps are the following:

1. Compute the cube skeleton $\mathcal{S}$.

2. Compute the s-scaled shape $\mathcal{S}_s$.

3. Compute the cube skeleton of $\mathcal{S}_s$.

Section 8.5.1 and 8.5.2 presented algorithms to compute the cube skeleton of $\mathcal{S}$. Once the cube skeleton is computed, the subset of rectangles that cover the cube transform CT, such that when grown represent the shape $\mathcal{S}_s$, is extracted (see Lemma 55).

Lemma 59. *(WOOD [227]) The boundary of the union of m axis-aligned rectangles can be computed in $O(m \log m + k)$ time and $O(m)$ space, where k is the number of vertices of the boundary.*

Property 22. *(LIPSKI AND PREPARATA [130]) The boundary of the union of m axis-aligned rectangles has less or equal than $m^2 + 4m$ edges.*

Lemma 60. *When $s > 1$, the s-scaled shape of an orthogonal polygon can be computed in $O(n \log n + k)$ time and $O(n)$ space, where n is the number of vertices of the orthogonal polygon and k is the number of vertices of the scaled shape.*

Proof. It is known from Lemma 55 that it is possible to represent the set of squares CT by a finite set of rectangles that are grown. There are $O(n)$ segments in CS, each of them defining a single bounding rectangle. Then, the number of rectangles necessary to cover the s-scaled shape is $O(n)$. Thus, the boundary of $\mathcal{S}_s$ can be computed in $O(n \log n + k)$ time and $O(n)$ space (see Lemma 59). □

Lemma 61. *When $s > 1$, the scale cube skeleton of an orthogonal polygon can be computed in $O(\max(n \log n, k \log k))$ time and $O(\max(n, k))$ space, where n is the number of vertices of the orthogonal polygon and k is the number of vertices of the scaled shape.*

Proof. The scale cube skeleton requires two different types of computations: the cube skeleton computation of $\mathcal{S}$ and $\mathcal{S}_s$ and the computation of the s-scaled shape $\mathcal{S}_s$. From Theorem 5, the cube skeleton can be computed in $O(n \log n)$ time and $O(n)$ space. When $s > 1$ the s-scaled shape can be computed in $O(n \log n + k)$ and $O(n)$ space (see Lemma 60). Finally, the scaled shape can be computed in $O(k \log k)$ time and $O(k)$ space. Thus, the overall time complexity is dominated by the maximum between n and k. □

Definition 85. *The* interior scale cube skeleton *of is the interior cube skeleton of the s-scaled shape. The* interior scale cube transform *is the scale cube transform of the interior cube skeleton (see Figure 9.4 and Figure 9.5).*

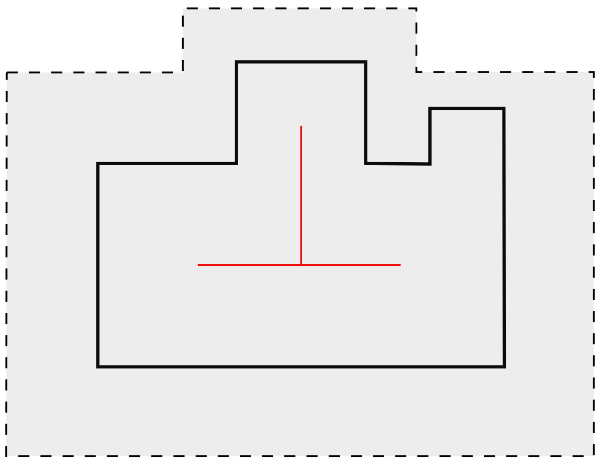


Figure 9.4: Interior scale cube skeleton of an orthogonal shape. Boundary of the scaled shape is shown in dotted lines.

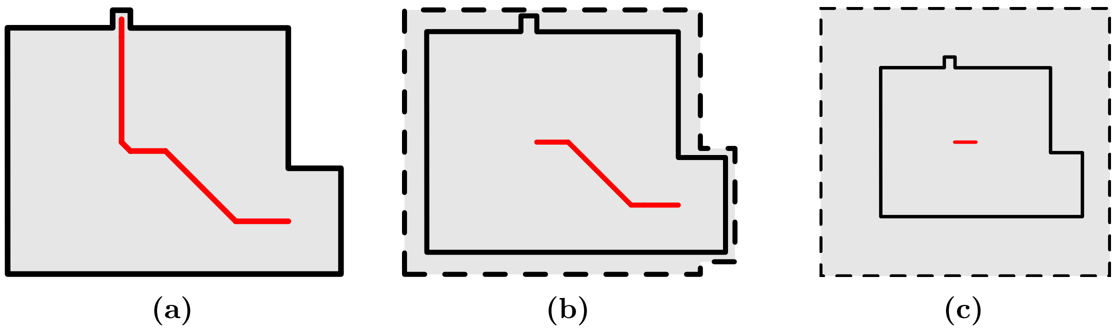


Figure 9.5: **(a)** A simple orthogonal polygon and its interior cube skeleton drawn in red. **(b)** The interior cube skeleton of the s-scaled shape removes an irrelevant branch. **(c)** As the s-scaled shape grows, the interior cube skeleton becomes more simple.

9.5 Experimental results

Although the introduced skeletal representations are applied to orthogonal shapes, it is possible to use them with any kind of shape by considering the approximation framework described in [21]. First, an orthogonal shape approximating an input shape is computed. Secondly, the cube skeleton is computed and simplified with the multiplicative scaling, providing a more stable skeletal approximation of the input shape. This approximation paradigm allows for a trade-off between the skeletal representation resolution and the computation time.

Approximating a shape by an orthogonal shape is straightforward and efficient (see Section 4). A fast way to extract an orthogonal shape is to first discretize the input shape and then extract the boundary of the discretized data. The discretization can be greatly accelerated with the help of modern GPUs [187] when dealing with 3D shapes and is straightforward in the 2D case. On the contrary, the discrete scale axis transform [164] relies on the shape approximation by a union of balls, that is slower.

The medial axis, straight skeleton, interior cube skeleton and interior scale cube skeleton of some shapes have been computed. The medial axis and the straight skeleton of polygons have been implemented using the C++ CGAL library [1]. The theoretical time complexity of the medial axis and straight skeleton algorithms is $O\left(n \log^2 n\right)$ and $O\left(n^2\right)$, respectively, where n is the number of vertices of the polygon. The algorithm to compute the scaled shape has been implemented in C++ and the source code consists of about only three hundred lines of code. Table 9.1 shows the results for six sample objects that are shown in Figure 9.6. The segments of the medial axis and the straight skeleton emanating from the vertices of the input polygon are erased, in order to compare their skeletal structure with the interior cube skeleton. A metric to quantify the similarity between a polygon and the approximated orthogonal polygon is presented. The normalized Hausdorff distance between two sets (d_{HN}) is defined as the Hausdorff distance between the two sets, divided by the diameter of the bounding box of both sets. For all the models in the Table 9.1, the approximation of the input polygon with an orthogonal polygon is accurate ($d_{HN} < 0.001$). Figure 9.6 demonstrates that the interior scale cube skeleton is able to simplify the skeleton while retaining its important topological features. In addition, the time to compute the interior scale cube skeleton is comparable to the performance of CGAL library and is also slightly faster than the straight skeleton computation, although the computation of the union of rectangles is done with a naive algorithm. Thus, the computation time of the SCT can be further improved.

Figure 9.7 illustrates the fact that a less accuracy of the shape approximation produces skeletons composed of less vertices and less accurate. As the scale factor increases the scale cube skeleton and the scaled shape are simplified. In fact, the scaled shape may also be employed for shape simplification.

		Medial axis		**Straight skeleton**		**Cube skeleton**		**Scale cube skeleton**		
Figure	Vertices	Elements	Time	Vertices	Time	Vertices	Time	Vertices	Time	s-scale
(a)	533	530	0.018	528	0.235	1998	0.013	674	0.046	1.3
(b)	573	581	0.013	1112	0.226	2679	0.022	1169	0.069	1.1
(c)	521	537	0.016	517	0.209	2893	0.018	1390	0.059	1.2
(d)	1045	1112	0.066	1103	2.56	1812	0.014	1656	0.138	1.2
(e)	1414	1418	0.051	1412	0.423	1636	0.013	1006	0.086	1.2
(f)	4321	4974	0.176	4623	4.58	5902	0.046	4322	0.197	1.2
(g)	2549	2677	0.091	2559	3.99	5894	0.057	3918	0.161	1.4

Table 9.1: Comparison between the skeleton of some two-dimensional shapes (see Figure 9.6). Running times are in seconds.

9.6 Conclusions

A novel skeletal representation of orthogonal shapes, that extends the cube skeleton and is based upon the scale axis transform, has been presented. The scale cube skeleton can be used as a simple alternative to the scale axis. In addition, an algorithm to compute the scale cube skeleton of orthogonal polygons has been introduced and the ability of the scale cube skeleton to obtain stable skeletal structures has been demonstrated experimentally. Moreover, approximating an orthogonal shape from other classes of shape representations is efficient and robust. The approximation paradigm allows a trade-off between the skeletal representation resolution and the computation time.

Although the worst-case time complexity to compute the scale cube skeleton of orthogonal polygons may be quadratic, with respect to the number of vertices of the input orthogonal polygon (see Property 22), the bigger rectangles tend to cover the smaller ones when they are grown. In fact, the boundary of the scaled shape is expected to have less vertices than the boundary of the input orthogonal shape. In this latter case, the time complexity will be just $O(n \log n)$, where n is the number of vertices of the orthogonal polygon. While the time complexity of discrete methods is bounded by the number of pixels and voxels of the discretized shape the time complexity of the presented algorithm is constrained by the number of vertices of the orthogonal shape.

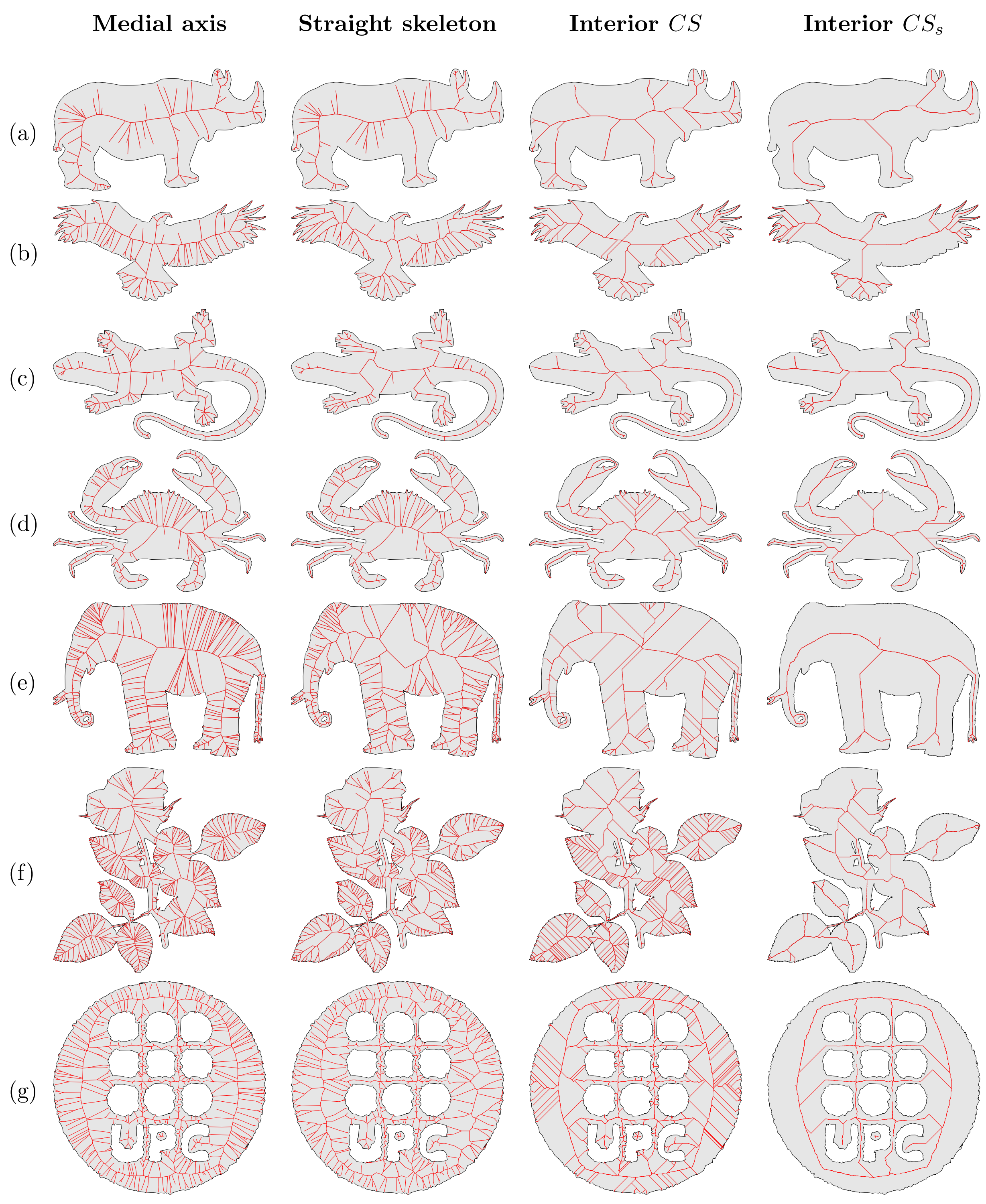


Figure 9.6: Comparison between the medial axis, straight skeleton, cube skeleton and scale cube skeleton of some two-dimensional shapes. The input polygon is shown in gray. The skeleton is shown in red.

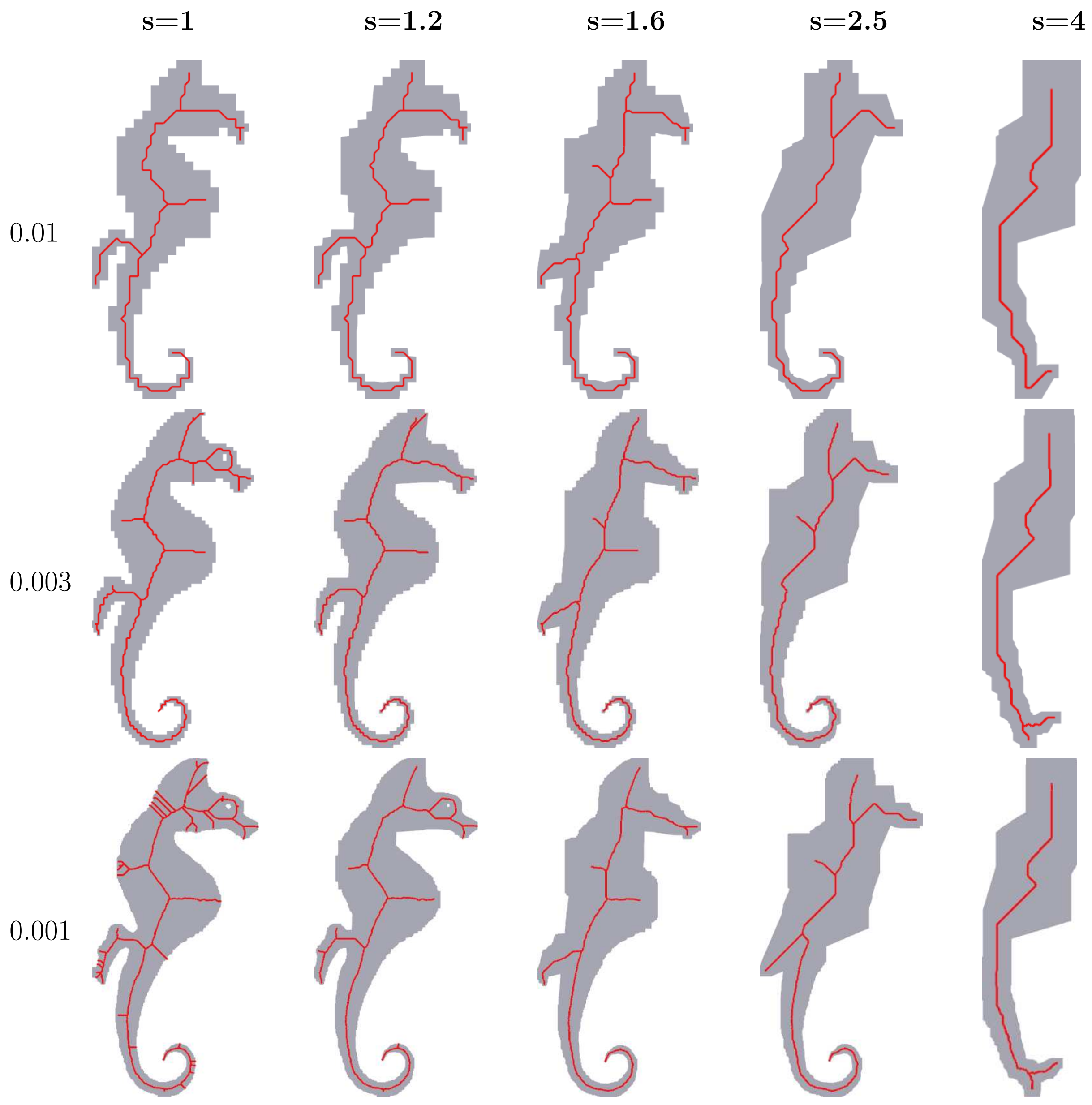


Figure 9.7: The first column indicates the d_{HN} of the orthogonal polygons with respect to the original shape. The first row shows the scale factor values associated to each column. The interior s-scaled shape, drawn in gray, and the interior scale cube skeleton, drawn in red, are shown together.

Chapter 10

Conclusions and future work

The first objective of this thesis was to develop practical and robust algorithms to compute the L_∞ Voronoi diagram of orthogonal shapes, which could serve as a skeletal representation with linear structure and constrained combinatorial complexity. However, the L_∞ Voronoi diagram do not fulfilled some properties, such as dimension reduction and homotopy equivalence, that are important in many applications. In addition, some desirable properties of two-dimensional L_∞ Voronoi diagrams could not directly extended to three dimensions. These aforementioned limitations lead the introduction of the cube skeleton, which appropriately extends and refine the L_∞ Voronoi diagram. Finally, the scale cube skeleton was introduced in order to overcome the stability problems of the cube skeleton.

Thus, the main contributions of this thesis are the design of a sweep line algorithm to compute the L_∞ Voronoi diagram of orthogonal polyhedra, and the introduction of two skeletal representations of orthogonal shapes. A specific implementation to compute the L_∞ Voronoi diagram of orthogonal polygons has also been provided. In addition, an algorithm to extract the B-Rep of orthogonal shapes as well as an algorithm to trihedralize an orthogonal polyhedron have been presented. The presented skeletal structures and algorithms could be used to compute the structural properties of porous biomaterials in an efficient and robust manner, and could be extended to biomedical applications that analyze the structure of shapes. For a more detailed review of the conclusions, see the conclusion section included in each chapter. Next, the future work related to each contribution of this thesis is presented.

Concerning the B-Rep extraction of orthogonal polyhedra, the presented algorithm may be extended in order to identify and classify the three-dimensional shells of the input polyhedron. Once the B-Rep model is computed, identifying the shells can be done by traversing the graph and extracting the connected components. Depending on the desired criterion, non-manifold vertices may separate different shells. After the shells are identified, obtaining the inclusion relationship between them could be done similarly as in 2D, using a sweep-plane algorithm. The status data structure of the sweep-plane algorithm may be required to efficiently maintain a two-dimensional structure, that represents the current intersection of the input orthogonal polyhedron with the sweeping plane.

Although the trihedralization of general polyhedra clearly seems to be an NP-complete problem, it is not still clear whether the trihedralization of orthogonal polyhedra is possibly NP-complete or if it can be solved by a polynomial time algorithm. As orthogonal polyhedra have a constrained structure and the restrictions introduced by the trihedralization are also constrained, an approach to compute the trihedralization with a polynomial time complexity may exist. Moreover, it could be studied how to guarantee the existence of a trihedralization by, for example, introducing new faces to the orthogonal polyhedron.

The performance of the algorithm to compute the L_∞ Voronoi diagram of polyhedra could be improved by using some sort of kinetic data structure [36] that maintains the moving Voronoi wavefront. Apart from the efficient dynamic insertion and deletion of graph vertices and edges, this data structure should withstand the following two queries: obtain the collision between a moving ray of the wavefront and an edge of the wavefront, which corresponds to a ray-face junction event induced by a ray wavefront, and locate the wavefront face that contains a given point, which corresponds to a ray-face junction event induced by a vertex ray with the interior of the polyhedron above the z-plane.

Presently, a non orthogonal shape is approximated by a discrete shape, composed of pixels or voxels, in order to obtain an approximated orthogonal shape. In the Appendix of [164], the approximation of an input shape by a union of balls is analyzed. More precisely, the value of the required Hausdorff distance between both shapes, and a bound on the number of balls required by the homotopy equivalent approximation are given. It will be desirable to give similar bounds in the case of a shape approximated by an orthogonal shape.

The interior cube skeleton of orthogonal polyhedra could be properly defined. In addition, an algorithm to compute the scale cube skeleton of orthogonal polyhedra, which would compute the union of axis-aligned boxes, could be developed. In addition, the homotopy equivalence between the scaled shape and the original shape may be characterized. For large values of the scale factor the interior scale cube skeleton of an orthogonal polygon may lie outside the shape. This property is not desirable and it could be investigated how to avoid it.

Finally, observe that the orthogonal projection of a point and the flat intersection operators may be extended to higher dimensions. Thus, it is expected that the formulation of the cube skeleton and scale cube skeleton could be extended to higher dimensional orthogonal polytopes.

List of publications

- Martinez, J., Vigo, M., Pla-Garcia, N., and Ayala, D. Skeleton computation of an image using a geometric approach. In *Proceedings of Eurographics* (2010), pp. 13–16
- Martinez, J., Vigo, M., and Pla-Garcia, N. Skeleton computation of orthogonal polyhedra. *Computer Graphics Forum 30*, 5 (2011), 1573–1582
- Vigo, M., Pla, N., Ayala, D., and Martinez, J. Efficient algorithms for boundary extraction of 2D and 3D orthogonal pseudomanifolds. *Graphical Models 74*, 3 (2012), 61 – 74
- Martinez, J., Pla-Garcia, N., and Vigo, M. Skeletal representations of orthogonal shapes. *Graphical Models 75* (2013), 189–207

Glossary

Ackermann function In computability theory, the Ackermann function is a total computable function that is not primitive recursive. Its value grows rapidly, even for small inputs. 22

Apollonius diagram The Apollonius Voronoi diagram (also known as additively weighted Voronoi diagram) is defined when positive weights are subtracted from the distances between points. 26

BioCAD Discipline that uses CAD techniques to analyze the structure of biomaterials or to simulate, design and manufacture them. i, iii, 1

CAD Computer aided design (CAD) is the use of computer systems to assist in the creation, modification, analysis, or optimization of a design. 35, 37

CSG Constructive Solid Geometry (CSG) is a technique that creates complex objects by combining simpler objects with successive Boolean operations. 29

dynamical system A concept in mathematics consisting of a function that describes the time dependence of a point in a geometrical space. For example, the mathematical model that describe the swinging of a clock pendulum. 39, 40

FEM Finite element analysis (FEM) is a numerical technique for finding approximate solutions to partial differential equations and their systems, as well as integral equations. 35

GPU A graphics processing unit (GPU) is a specialized electronic circuit designed to rapidly manipulate and alter memory to accelerate the building of images intended for output to a display. 30, 157

k-d tree A k-dimensional tree (k-d tree) is a space-partitioning data structure for organizing points in a k-dimensional space. 38

Morse function A smooth real-valued function on a manifold is a Morse function if it has no degenerate critical points. 23

motion planning A term used in robotics for the process of detailing a task into discrete motions. 34, 39

NC Numerical control (NC) is the automation of machine tools that are operated by abstractly programmed commands encoded on a storage medium. 39

octree A tree data structure that partitions a three-dimensional space by recursively subdividing it into eight octants. 30, 38, 39, 49

origami The traditional Japanese art of paper folding. The goal of this art is to transform a flat sheet of paper into a finished sculpture through folding and sculpting techniques. 35

quadtree A tree data structure that partitions a two-dimensional space by recursively subdividing it into four quadrants or regions. 30, 39

regular grid A tessellation of the Euclidean space by congruent parallelotopes, which are the generalization of parallelepipeds in higher dimensions. 38

run length encoding Run-length encoding (RLE) is a very simple form of data compression in which runs of data, that is, sequences in which the same data value occurs in many consecutive data elements, are stored as a single data value and count, rather than as the original run. 39, 40

spatial enumeration A solid representation scheme represented by a list of spatial cells occupied by the solid. The cells, also called voxels are cubes of a fixed size and are arranged in a fixed spatial grid. 39, 40, 51

urban model Computer simulations used for testing theories about spatial location and interaction between land uses and related activities. 1, 39

visibility problem Given a set of obstacles in the Euclidean space, two points in the space are said to be visible to each other, if the line segment that joins them does not intersect any obstacles. 35, 39

voxel A volume element representing a value on a regular grid in three-dimensional space. 30, 34, 38, 164

Bibliography

[1] Cgal, Computational Geometry Algorithms Library. `http://www.cgal.org`.

[2] Adams, B., Pauly, M., Keiser, R., and Guibas, L. J. Adaptively sampled particle fluids. *ACM Transactions on Graphics 26*, 3 (2007), 48–54.

[3] Aggarwal, A., Guibas, L., Saxe, J., and Shor, P. A linear-time algorithm for computing the Voronoi diagram of a convex polygon. *Discrete and Computational Geometry 4* (1989), 591–604.

[4] Aguilera, A. *Orthogonal Polyhedra: Study and Application.* PhD thesis, Universitat Politècnica de Catalunya, 1998.

[5] Aguilera, A., and Ayala, D. Converting orthogonal polyhedra from extreme vertices model to B-Rep and to alternating sum of volumes. *Computing Supplement 14* (2001), 1–28.

[6] Ahn, H.-K., Bae, S. W., Knauer, C., Lee, M., Shin, C.-S., and Vigneron, A. Realistic roofs over a rectilinear polygon. *Computational Geometry 46*, 9 (2013), 1042 – 1055.

[7] Ahuja, N., and Chuang, J.-H. Shape representation using a generalized potential field model. *IEEE Transactions on Pattern Analysis and Machine Intelligence 19* (1997), 169–176.

[8] Aichholzer, O., Aigner, W., Aurenhammer, F., and Jüttler, B. Exact medial axis computation for triangulated solids with respect to piecewise linear metrics. In *Proceedings of the Seventh International Conference on Curves and Surfaces 2010* (2010), Lecture Notes in Computer Science.

[9] Aichholzer, O., Aigner, W., Hackl, T., and Wolpert, N. Exact medial axis computation for circular arc boundaries. *Curves and Surfaces 6920* (2012), 28–42.

[10] Aichholzer, O., and Aurenhammer, F. Straight skeletons for general polygonal figures in the plane. In *Proceedings of the second Annual International Conference on Computing and Combinatorics* (1996), pp. 117–126.

[11] Aichholzer, O., Aurenhammer, F., Alberts, D., and Gärtner, B. A novel type of skeleton for polygons. *Journal of Universal Computer Science 1* (1995), 752–761.

[12] Albers, S., Kursawe, K., and Schuierer, S. Exploring unknown environments with obstacles. In *Proceedings of the tenth annual ACM-SIAM symposium on Discrete algorithms* (1999), pp. 842–843.

[13] Amenta, N., and Bern, M. Surface reconstruction by Voronoi filtering. *Discrete and Computational Geometry 22* (1999), 481–504.

[14] Amenta, N., Bern, M., and Eppstein, D. The crust and the beta-skeleton: combinatorial curve reconstruction. *Graphical models and image processing 60* (1998), 125–135.

[15] Amenta, N., Choi, S., and Kolluri, R. K. The power crust. In *Proceedings of the sixth ACM symposium on Solid modeling and applications* (2001), pp. 249–266.

[16] Amenta, N., and Kolluri, R. K. The medial axis of a union of balls. *Computational Geometry Theory and Applications 20* (2001), 25–37.

[17] Arcelli, C., and Di Baja, G. S. A width-independent fast thinning algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence 7* (1985), 463–474.

[18] Arcelli, C., Sanniti Di Baja, G., and Serino, L. From 3D discrete surface skeletons to curve skeletons. In *Proceedings of the 5th international conference on Image Analysis and Recognition* (2008), pp. 507–516.

[19] Asarin, E., Bournez, O., Dang, T., and Maler, O. Approximate reachability analysis of piecewise-linear dynamical systems. *Lecture Notes in Computer Science 1790* (2000), 20–31.

[20] Attali, D. *Squelettes et graphes de Voronoi 2D et 3D.* PhD thesis, Université Joseph-Fourier, 1995.

[21] Attali, D., Boissonnat, J. D., and Edelsbrunner, H. Stability and computation of medial axes - a state-of-the-art report. In *Mathematical Foundations of Scientific Visualization, Computer Graphics, and Massive Data Exploration*, Mathematics and Visualization. Springer Berlin / Heidelberg, 2009, pp. 109–125.

[22] Attali, D., di Baja, G., and Thiel, E. Pruning discrete and semicontinuous skeletons. *Lecture Notes in Computer Science 974* (1995), 488–493.

[23] Attali, D., and Lachaud, J. O. Delaunay conforming iso-surface, skeleton extraction and noise removal. *Computational Geometry 19* (2001), 175–189.

[24] Attali, D., and Montanvert, A. Modeling noise for a better simplification of skeletons. In *International Conference on Image Processing* (1996), vol. 3, pp. 13–16.

[25] Attali, D., and Montanvert, A. Computing and simplifying 2D and 3D continuous skeletons. *Computer Vision and Image Understanding 67* (1997), 261–273.

[26] Au, O. K.-C., Tai, C.-L., Chu, H.-K., Cohen-Or, D., and Lee, T.-Y. Skeleton extraction by mesh contraction. *ACM Transactions on Graphics 27* (2008), 44:1–44:10.

[27] Aurenhammer, F. Voronoi diagrams. A survey of a fundamental geometric data structure. *ACM Computing Surveys 23* (1991), 345–405.

[28] Aurenhammer, F. Weighted skeletons and fixed-share decomposition. *Computational Geometry Theory and Applications 40* (2008), 93–101.

[29] Aurenhammer, F., and Imai, H. Geometric relations among Voronoi diagrams. *Geometriae Dedicata 27*, 1 (1988), 65–75.

[30] Ayala, D., Vergara, E., and Vergés, E. Improved skeleton computation of an encoded volume. In *Proceedings of Eurographics* (2007), pp. 33–36.

[31] Ba, W., Cao, L., and Liu, J. Research on 3D medial axis transform via the saddle point programming method. *Computer-Aided Design 44*, 12 (2012), 1161–1172.

[32] Balaban, I. J. An optimal algorithm for finding segments intersections. In *Proceedings of the eleventh annual symposium on Computational geometry* (1995), pp. 211–219.

[33] Barequet, G., Eppstein, D., Goodrich, M. T., and Vaxman, A. Straight skeletons of three-dimensional polyhedra. *Lecture Notes in Computer Science 5193* (2008), 148 – 160.

[34] Barequet, G., Goodrich, M. T., Levi-Steiner, A., and Steiner, D. Straight-skeleton based contour interpolation. In *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms* (2003), pp. 119–127.

[35] Barequet, G., and Vaxman, A. Reconstruction of multi-label domains from partial planar cross-sections. *Computer Graphics Forum 28*, 5 (2009), 1327–1337.

[36] Basch, J. *Kinetic data structures.* PhD thesis, Stanford University, 1999.

[37] Bertrand, G., and Aktouf, Z. Three-dimensional thinning algorithm using subfields. In *Proceedings of SPIE* (1995), vol. 2356, p. 113.

[38] Biasotti, S., Attali, D., Boissonnat, J.-D., Edelsbrunner, H., Elber, G., Mortara, M., Baja, G. S., Spagnuolo, M., Tanase, M., and Veltkamp, R. Skeletal structures. In *Shape Analysis and Structuring*. Springer Berlin / Heidelberg, 2008, pp. 145–183.

[39] Biasotti, S., Falcidieno, B., and Spagnuolo, M. Extended reeb graphs for surface understanding and description. In *Proceedings of the 9th International Conference on Discrete Geometry for Computer Imagery* (2000), pp. 185–197.

[40] Biasotti, S., Falcidieno, B., and Spagnuolo, M. Shape abstraction using computational topology techniques. In *From geometric modeling to shape modeling.* Kluwer Academic Publishers, 2002, pp. 209–222.

[41] Biasotti, S., Marini, S., Mortara, M., and Patane, G. An overview on properties and efficacy of topological skeletons in shape modeling. In *Shape Modeling International* (2003), pp. 245–254.

[42] Biedl, T., Durocher, S., and Snoeyink, J. Reconstructing polygons from scanner data. *Lecture Notes in Computer Science 5878* (2009), 862–871.

[43] Biedl, T., and Genç, B. Reconstructing orthogonal polyhedra from putative vertex sets. *Computational Geometry 44* (2011), 409–417.

[44] Bittar, E., Tsingos, N., and Gascuel, M.-P. Automatic reconstruction of unstructured 3D data: Combining a medial axis and implicit surfaces. *Computer Graphics Forum 14*, 3 (1995), 457–468.

[45] Bitter, I., Kaufman, A., and Sato, M. Penalized-distance volumetric skeleton algorithm. *IEEE Transactions on Visualization and Computer Graphics 7* (2001), 195 –206.

[46] Blum, H. A transformation for extracting new descriptors of shape. In *Models for the Perception of Speech and Visual Form.* MIT Press, 1967, pp. 362–380.

[47] Blum, H. Biological shape and visual science (part I). *Journal of theoretical Biology 38*, 2 (1973), 205–287.

[48] Boissonnat, J., and Karavelas, M. On the combinatorial complexity of Euclidean Voronoi cells and convex hulls of d-dimensional spheres. In *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms* (2003), pp. 305–312.

[49] Bournez, O., Maler, O., and Pnueli, A. Orthogonal polyhedra: Representation and computation. *Lecture Notes in Computer Science 1569* (1999), 46–60.

[50] Bradshaw, G., and O'Sullivan, C. Adaptive medial-axis approximation for sphere-tree construction. *ACM Transactions on Graphics 23* (2004), 1–26.

[51] Brandt, J. Convergence and continuity criteria for discrete approximations of the continuous planar skeleton. *CVGIP: Image Understanding 59* (1994), 116–124.

[52] Brandt, J. W., and Algazi, V. R. Continuous skeleton computation by Voronoi diagram. *CVGIP: Image Understanding 55* (1991), 329–338.

[53] Brédif, M. *Modélisation 3D de bâtiments: reconstruction automatique de superstructures de toits et recalage cinétique de toits polyédriques prenant en compte la topologie.* PhD thesis, Télécom ParisTech, 2010.

[54] Bruck, J., Gao, J., and Jiang, A. A. MAP: medial axis based geometric routing in sensor networks. In *Proceedings of the 11th annual international conference on Mobile computing and networking* (2005), pp. 88–102.

[55] Canny, J., and Donald, B. Simplified Voronoi diagrams. *Discrete and Computational Geometry 3*, 1 (1988), 219–236.

[56] Cao, L., Ba, W., and Liu, J. Computation of the medial axis of planar domains based on saddle point programming. *Computer-Aided Design 43* (2011), 979–988.

[57] Capell, S., Green, S., Curless, B., Duchamp, T., and Popović, Z. Interactive skeleton-driven dynamic deformations. *ACM Transactions on Graphics 21*, 3 (2002), 586–593.

[58] Chang, M.-C., and Kimia, B. Regularizing 3D medial axis using medial scaffold transforms. In *IEEE Conference on Computer Vision and Pattern Recognition* (2008), pp. 1–8.

[59] Chaussard, J., Couprie, M., and Talbot, H. A discrete lambda-medial axis. In *International Conference on Discrete Geometry for Computer Imagery* (2009), vol. 5810, pp. 421–433.

[60] Chazal, F., Cohen-Steiner, D., and Lieutier, A. A sampling theory for compact sets in Euclidean space. *Discrete and Computational Geometry 41* (2009), 461–479.

[61] Chazal, F., and Lieutier, A. Stability and homotopy of a subset of the medial axis. In *Proceedings of the ninth ACM symposium on Solid modeling and applications* (2004), pp. 243–248.

[62] Chazal, F., and Lieutier, A. The λ medial axis. *Graphical Models 67* (2005), 304–331.

[63] Chazal, F., and Soufflet, R. Stability and finiteness properties of medial axis and skeleton. *Journal of Dynamical and Control Systems 10* (2004), 149–170.

[64] Cheng, S.-W., and Vigneron, A. Motorcycle graphs and straight skeletons. *Algorithmica 47* (2007), 159–182.

[65] Cheong, J.-S., and van der Stappen, A. Computing all independent form-closure grasp regions of a rectilinear polyhedron. In *IEEE International Conference on Automation Science and Engineering* (2007), pp. 288 –294.

[66] Chin, F., Snoeyink, J., and Wang, C. A. Finding the medial axis of a simple polygon in linear time. In *Proceedings of the 6th International Symposium on Algorithms and Computation* (1995), pp. 382–391.

[67] Choi, H. I., Choi, S. W., and Moon, H. P. Mathematical theory of medial axis transform. *Pacific Journal of Mathematics 181* (1997), 57–88.

[68] Christ, T., and Hoffmann, M. Wireless localization within orthogonal polyhedra. In *Proceedings of the 23rd Canadian Conference on Computational Geometry* (2011).

[69] Chuang, J.-H., Tsai, C.-H., and Ko, M.-C. Skeletonization of three-dimensional object using generalized potential field. *IEEE Transactions on Pattern Analysis and Machine Intelligence 22* (2000), 1241–1251.

[70] Cline, H. E., Lorensen, W. E., Herfkens, R. J., Johnson, G. A., and Glover, G. H. Vascular morphology by three dimensional magnetic resonance. *Magnetic Resonance Imaging 7* (1989), 45–54.

[71] Cormen, T., Leiserson, C., Rivest, R., and Stein, C. *Introduction to algorithms*. MIT Press, 2001.

[72] Cornea, N., Silver, D., Yuan, X., and Balasubramanian, R. Computing hierarchical curve-skeletons of 3D objects. *The Visual Computer 21* (2005), 945–955.

[73] Cornea, N. D., and Silver, D. Curve-skeleton properties, applications, and algorithms. *IEEE Transactions on Visualization and Computer Graphics 13* (2007), 530–548.

[74] Couprie, M., and Bertrand, G. New characterizations of simple points in 2D, 3D, and 4D discrete spaces. *IEEE Transactions on Pattern Analysis and Machine Intelligence 31* (2009), 637 –648.

[75] Culver, T., Keyser, J., and Manocha, D. Exact computation of the medial axis of a polyhedron. *Computer Aided Geometric Design 21* (2004), 65–98.

[76] Damiand, G. Topological model for 3D image representation: Definition and incremental extraction algorithm. *Computer Vision and Image Understanding 109* (2008), 260–289.

[77] Danielsson, P.-E. Euclidean distance mapping. *Computer Graphics and Image Processing 14* (1980), 227–248.

[78] Das, G. K., Mukhopadhyay, A., Nandy, S. C., Patil, S., and Rao, S. V. Computing the straight skeleton of a monotone polygon in $O(n \log n)$ time. In *Canadian Conference on Computational Geometry* (2010), pp. 207–210.

[79] Davies, E., and Plummer, A. Thinning algorithms: A critique and a new methodology. *Pattern Recognition 14* (1981), 53 – 63.

[80] de Berg, M., Cheong, O., van Kreveld, M., and Overmars, M. *Computational Geometry: Algorithms and Applications.* Springer, 2008.

[81] Demaine, E., Demaine, M., Lindy, J., and Souvaine, D. Hinged dissection of polypolyhedra. *Lecture Notes in Computer Science 3608* (2005), 205–217.

[82] Demaine, E., Demaine, M., and Lubiw, A. Folding and cutting paper. *Discrete and Computational Geometry 1763* (2004), 104–118.

[83] Desbrun, M., Meyer, M., Schröder, P., and Barr, A. H. Implicit fairing of irregular meshes using diffusion and curvature flow. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques* (1999), pp. 317–324.

[84] Dey, T., and Sun, J. Defining and computing curve-skeletons with medial geodesic function. In *Proceedings of the fourth Eurographics symposium on Geometry processing* (2006), pp. 143–152.

[85] Dey, T. K., Woo, H., and Zhao, W. Approximate medial axis for CAD models. In *Proceedings of the eighth ACM symposium on Solid modeling and applications* (2003), pp. 280–285.

[86] Dey, T. K., and Zhao, W. Approximate medial axis as a Voronoi subcomplex. In *Proceedings of the seventh ACM symposium on Solid modeling and applications* (2002), pp. 356–366.

[87] Dietrich, C. A., Scheidegger, C. E., Schreiner, J., Comba, J., Nedel, L., and Silva, C. T. Edge transformations for improving mesh quality of marching cubes. *IEEE Transactions on Visualization and Computer Graphics 15*, 1 (2009), 150–159.

[88] Dijkstra, E. W. A note on two problems in connexion with graphs. *Numerische Mathematik 1* (1959), 269–271.

[89] Edelsbrunner, H., and Mücke, E. P. Simulation of simplicity: a technique to cope with degenerate cases in geometric algorithms. *ACM Transactions on Graphics 9* (1990), 66–104.

[90] Elber, G., and Kim, M.-S. Bisector curves of planar rational curves. *Computer-Aided Design 30*, 14 (1998), 1089–1096.

[91] Eppstein, D. Fast hiearchical clustering and other applications of dynamic closest pairs. In *Proceedings of the ninth annual ACM-SIAM symposium on Discrete algorithms* (1998), pp. 619–628.

[92] Eppstein, D., and Erickson, J. Raising roofs, crashing cycles, and playing pool: Applications of a data structure for finding pairwise interactions. In *Proceedings of the 14th annual ACM symposium on Computational Geometry* (1999), pp. 58–67.

[93] Esperança, C., and Samet, H. Representing orthogonal multidimensional objects by vertex lists. In *Proceedings of the 2nd International Workshop on Visual Form* (1994), pp. 209–220.

[94] Esperança, C., and Samet, H. Orthogonal polygons as bounding structures in filter-refine query processing strategies. *Lecture Notes in Computer Science 1262* (1997), 197–220.

[95] Esperança, C., and Samet, H. Vertex representations and their applications in computer graphics. *The Visual Computer 14* (1998), 240–256.

[96] Etzion, M., and Rappoport, A. Computing the Voronoi diagram of a 3D polyhedron by separate computation of its symbolic and geometric parts. In *Proceedings of the fifth ACM symposium on Solid modeling and applications* (1999), pp. 167–178.

[97] Fabbri, R., Costa, L. D. F., Torelli, J. C., and Bruno, O. M. 2D Euclidean distance transform algorithms: A comparative survey. *ACM Computing Surveys 40*, 1 (2008), 2:1–2:44.

[98] Felkel, P., and Obdrzalek, S. Straight skeleton implementation. In *Proceedings of Spring Conference on Computer Graphics* (1998), pp. 210–218.

[99] Foskey, M., Lin, M. C., and Manocha, D. Efficient computation of a simplified medial axis. In *Proceedings of the eighth ACM symposium on Solid modeling and applications* (2003), pp. 96–107.

[100] Fulton, W. *Algebraic Topology, A First Course.* Springer-Verlag, 1995.

[101] Gagvani, N., and Silver, D. Parameter-controlled volume thinning. *Graphical Models and Image Processing 61* (1999), 149–164.

[102] Gargantini, I., Schrack, G., and Kwok, A. Reconstructing multishell solids from voxel-based contours. *Computer-Aided Design 26* (1994), 293–301.

[103] Ghosh, S. K. Approximation algorithms for art gallery problems in polygons. *Discrete Applied Mathematics 158*, 6 (2010), 718–722.

[104] Giblin, P., and Kimia, B. Transitions of the 3D medial axis under a one-parameter family of deformations. *European Conference on Computer Vision 2351* (2002), 718–734.

[105] Giblin, P., and Kimia, B. A formal classification of 3D medial axis points and their local geometry. *IEEE Transactions on Pattern Analysis and Machine Intelligence 26* (2004), 238 –251.

[106] Giesen, J., and John, M. Surface reconstruction based on a dynamical system. *Computer Graphics Forum 21*, 3 (2002), 363–371.

[107] Giesen, J., Miklos, B., Pauly, M., and Wormser, C. The scale axis transform. In *Proceedings of the 25th annual symposium on Computational geometry* (2009), pp. 106–115.

[108] Giesen, J., Ramos, E. A., and Sadri, B. Medial axis approximation and unstable flow complex. *International Journal of Computational Geometry and Applications 18* (2008), 533–565.

[109] Goldreich, O., and Ron, D. Property testing in bounded degree graphs. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing* (1997), pp. 406–415.

[110] Gong, W., and Bertrand, G. A simple parallel 3D thinning algorithm. In *International Conference on Pattern Recognition* (1990), vol. 1, pp. 188–190.

[111] Google. Operations research tools. `http://code.google.com/p/or-tools/`.

[112] Goswami, S., Dey, T. K., and Bajaj, C. L. Identifying flat and tubular regions of a shape by unstable manifolds. In *Proceedings of the 2006 ACM symposium on Solid and physical modeling* (2006), pp. 27–37.

[113] Greβ, A., and Klein, R. Efficient representation and extraction of 2-manifold isosurfaces using kd-trees. *Graphical Models 66* (2004), 370–397.

[114] Grevera, G. J., Udupa, J. K., and Odhner, D. An order of magnitude faster isosurface rendering in software on a PC than using dedicated, general purpose rendering hardware. *IEEE Transactions on Visualization and Computer Graphics 6* (2000), 335–345.

[115] Guibas, L., Ramshaw, L., and Stolfi, J. A kinetic framework for computational geometry. In *IEEE Annual Symposium on Foundations of Computer Science* (1983), pp. 100–111.

[116] Haciomeroglu, M., Laycock, R., and Day, A. Automatic spatial analysis and pedestrian flow control for real-time crowd simulation in an urban environment. *The Visual Computer 24* (2008), 889–899.

[117] Heijmans, H. J. A. M. Connected morphological operators for binary images. *Computer Vision and Image Understanding 73* (1999), 99–120.

[118] Held, M. On computing Voronoi diagrams of convex polyhedra by means of wavefront propagation. In *Canadian Conference on Computational Geometry* (1994), pp. 128–133.

[119] Held, M. VRONI: An engineering approach to the reliable and efficient computation of Voronoi diagrams of points and line segments. *Computational Geometry 18*, 2 (2001), 95–123.

[120] Hesselink, W., Visser, M., and Roerdink, J. Euclidean skeletons of 3D data sets in linear time by the integer medial axis transform. *Computational Imaging and Vision 30* (2005), 259–268.

[121] Hilaga, M., Shinagawa, Y., Kohmura, T., and Kunii, T. Topology matching for fully automatic similarity estimation of 3D shapes. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques* (2001), pp. 203–212.

[122] Hisada, M., Belyaev, A. G., and Kunii, T. L. A skeleton-based approach for detection of perceptually salient features on polygonal surfaces. *Computer Graphics Forum 21*, 4 (2002), 689–700.

[123] Hoff, K. E., Keyser, J., Lin, M., Manocha, D., and Culver, T. Fast computation of generalized Voronoi diagrams using graphics hardware. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques* (1999), pp. 277–286.

[124] Hoffmann, C. The problems of accuracy and robustness in geometric computation. *Computer 22*, 3 (1989), 31–39.

[125] Hoffmann, C. How to construct the skeleton of CSG objects. In *Proceedings of the fourth IMA Conference* (1994), pp. 421–438.

[126] Huang, H., Wu, S., Cohen-Or, D., Gong, M., Zhang, H., Li, G., and Chen, B. L1-medial skeleton of point cloud. *ACM Transactions on Graphics 32*, 4 (2013), 65:1–65:8.

[127] Hubbard, P. Approximating polyhedra with spheres for time-critical collision detection. *ACM Transactions on Graphics 15*, 3 (1996), 179–210.

[128] Huber, S., and Held, M. Theoretical and practical results on straight skeletons of planar straight-line graphs. In *Proceedings of the 27th annual ACM symposium on Computational Geometry* (2011), pp. 171–178.

[129] Jones, M., Baerentzen, J., and Sramek, M. 3D distance fields: a survey of techniques and applications. *IEEE Transactions on Visualization and Computer Graphics 12* (2006), 581 –599.

[130] Jr., W. L., and Preparata, F. P. Finding the contour of a union of iso-oriented rectangies. *Journal of Algorithms 1*, 3 (1980), 235 – 246.

[131] Ju, T., Baker, M. L., and Chiu, W. Computing a family of skeletons of volumetric models for shape description. *Computer-Aided Design 39*, 5 (2007), 352–360.

[132] Karunakaran, K., and Shringi, R. Octree-to-brep conversion for volumetric nc simulation. *The International Journal of Advanced Manufacturing Technology 32* (2007), 116–131.

[133] Katz, S., and Tal, A. Hierarchical mesh decomposition using fuzzy clustering and cuts. *ACM Transactions on Graphics 22* (2003), 954–961.

[134] Kimia, B. On the role of medial geometry in human vision. *Journal of Physiology-Paris 97*, 2-3 (2003), 155–190.

[135] Kimmel, R., Shaked, D., Kiryati, N., and Bruckstein, A. Skeletonization via distance maps and level sets. *Computer Vision and Image Understanding 62*, 3 (1995), 382–391.

[136] Kirkpatrick, D. G. Efficient computation of continuous skeletons. In *Proceedings of the 20th annual Symposium on Foundations of Computer Science* (1979), pp. 18–27.

[137] Kirkpatrick, D. G. Optimal search in planar subdivisions. *SIAM Journal on Computing 12* (1983), 28–35.

[138] Klein, R., and Wood, D. Voronoi diagrams based on general metrics in the plane. In *5th Annual Symposium on Theoretical Aspects of Computer Science* (1988), pp. 281–291.

[139] Knuth, D. *The art of computer programming.* Addison-Wesley, 2006.

[140] Koltun, V., and Sharir, M. Polyhedral Voronoi diagrams of polyhedra in three dimensions. In *Proceedings of the eighteenth annual symposium on Computational geometry* (2002), pp. 227–236.

[141] Kong, T., and Rosenfeld, A. Digital topology: Introduction and survey. *Computer Vision, Graphics and Image Processing 48* (1989), 357–393.

[142] K.Suresh. Dimensional reduction process. In *Proceedings of the Eighth ACM Symposium on Solid Modeling and Applications* (2003), pp. 76–85.

[143] Lam, L., Lee, S.-W., and Suen, C. Thinning methodologies-a comprehensive survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence 14* (1992), 869–885.

[144] Lazarus, F., and Verroust, A. Level set diagrams of polyhedral objects. In *Proceedings of the fifth ACM symposium on Solid modeling and applications* (1999), pp. 130–140.

[145] Lee, D. T. Medial axis transformation of a planar shape. *IEEE Transactions on Pattern Analysis and Machine Intelligence 4* (1982), 363 –369.

[146] Lee, Y.-H., and Horng, S.-J. The chessboard distance transform and the medial axis transform are interchangeable. In *Proceedings of the 10th International Parallel Processing Symposium* (1996), pp. 424–428.

[147] LESCROART, M. D., AND BIEDERMAN, I. Cortical representation of medial axis structure. *Cerebral Cortex 23* (2013), 629–637.

[148] LEYMARIE, F., AND LEVINE, M. Simulating the grassfire transform using an active contour model. *IEEE Transactions on Pattern Analysis and Machine Intelligence 14* (1992), 56–75.

[149] LEYMARIE, F. F., AND KIMIA, B. B. From the infinitely large to the infinitely small. In *Medial Representations*, vol. 37. Springer Netherlands, 2008, pp. 327–351.

[150] LIEUTIER, A. Any open bounded subset of $\mathbb{R}^n$ has the same homotopy type than its medial axis. In *Proceedings of the eighth ACM symposium on Solid modeling and applications* (2003), pp. 65–75.

[151] LIU, L., CHAMBERS, E. W., LETSCHER, D., AND JU, T. A simple and robust thinning algorithm on cell complexes. *Computer Graphics Forum 29*, 7 (2010), 2253–2260.

[152] LIVNAT, Y., AND TRICOCHE, X. Interactive point-based isosurface extraction. In *IEEE Visualization* (2004), pp. 457–464.

[153] MA, C. M., AND SONKA, M. A fully parallel 3D thinning algorithm and its applications. *Computer Vision and Image Understanding 64* (1996), 420 – 433.

[154] MA, J., BAE, S., AND CHOI, S. 3D medial axis point approximation using nearest neighbors and the normal field. *The Visual Computer 28* (2012), 7–19.

[155] MALANDAIN, G., AND FERNÁNDEZ-VIDAL, S. Euclidean skeletons. *Image and Vision Computing 16* (1998), 317 – 327.

[156] MANGIN, J.-F., RIVIERE, D., CACHIA, A., DUCHESNAY, E., COINTEPAS, Y., PAPADOPOULOS-ORFANOS, D., COLLINS, D., EVANS, A., AND REGIS, J. Object-based morphometry of the cerebral cortex. *IEEE Transactions on Medical Imaging 23* (2004), 968–982.

[157] MARTIN, T., CHEN, G., MUSUVATHY, S., COHEN, E., AND HANSEN, C. Generalized swept mid-structure for polygonal models. *Computer Graphics Forum 31*, 2 (2012), 805–814.

[158] MARTINEZ, J., PLA-GARCIA, N., AND VIGO, M. Skeletal representations of orthogonal shapes. *Graphical Models 75* (2013), 189–207.

[159] MARTINEZ, J., VIGO, M., AND PLA-GARCIA, N. Skeleton computation of orthogonal polyhedra. *Computer Graphics Forum 30*, 5 (2011), 1573–1582.

[160] MARTINEZ, J., VIGO, M., PLA-GARCIA, N., AND AYALA, D. Skeleton computation of an image using a geometric approach. In *Proceedings of Eurographics* (2010), pp. 13–16.

[161] Massey, W. S. *Algebraic topology, an introduction.* Springer-Verlag, 1967.

[162] Matheron, G. Examples of topological properties of skeletons. *Image Analysis and Mathematical Morphology 2* (1988), 217–238.

[163] McAllister, M., Kirkpatrick, D., and Snoeyink, J. A compact piecewise-linear Voronoi diagram for convex sites in the plane. *Discrete and Computational Geometry 15*, 1 (1996), 73–105.

[164] Miklos, B., Giesen, J., and Pauly, M. Discrete scale axis representations for 3D geometry. *ACM Transactions on Graphics 29* (2010), 101:1–101:10.

[165] Milenkovic, V. Robust construction of the Voronoi diagram of a polyhedron. In *Canadian Conference on Computational Geometry* (1993), pp. 473–478.

[166] Milnor, J. W. *Morse theory.* Princeton University Press, 1963.

[167] Montani, C., and Scopigno, R. Quadtree/octree to boundary conversion. In *Graphics Gems II.* Academic Press Professional, Boston, MA, 1991, pp. 202–218.

[168] Morgenthaler, D. *Three-dimensional simple points: serial erosion, parallel thinning, and skeletonization.* University of Maryland, 1981.

[169] Müller, P., Wonka, P., Haegler, S., Ulmer, A., and Van Gool, L. Procedural modeling of buildings. *ACM Transactions on Graphics 25*, 3 (2006), 614–623.

[170] Musuvathy, S., Cohen, E., and Damon, J. Computing medial axes of generic 3D regions bounded by B-spline surfaces. *Computer-Aided Design 43*, 11 (2011), 1485 – 1495.

[171] Ogniewicz, R. L., and Kübler, O. Hierarchic Voronoi skeletons. *Pattern Recognition 28* (1995), 343 – 359.

[172] O'Rourke, J. Uniqueness of orthogonal connect-the-dots. In *Computational Morphology.* North-Holland, 1988, pp. 97–104.

[173] O'Rourke, J. Unfolding orthogonal polyhedra. In *Surveys on discrete and computational geometry* (2008), vol. 453, p. 307.

[174] Palfrader, P., Held, M., and Huber, S. On computing straight skeletons by means of kinetic triangulations. *Lecture Notes in Computer Science 7501* (2012), 766–777.

[175] Papadopoulou, E., and Lee, D. T. The L_∞ Voronoi diagram of segments and VLSI applications. *International Journal of Computational Geometry and Applications 11* (2001), 503 – 508.

[176] Park, S. C., and Choi, B. K. Boundary extraction algorithm for cutting area detection. *Computer-Aided Design 33* (2001), 571–579.

[177] Peschka, G. *Kotirte Ebenen un deren anwedung.* Verlag Buschak & Irrgang, 1877.

[178] Pizer, S., Fletcher, P., Joshi, S., Thall, A., Chen, J., Fridman, Y., Fritsch, D., Gash, A., Glotzer, J., Jiroutek, M., et al. Deformable m-reps for 3D medical image segmentation. *International Journal of Computer Vision 55* (2003), 85–106.

[179] Pizer, S. M., Siddiqi, K., Székely, G., Damon, J. N., and Zucker, S. W. Multiscale medial loci and their properties. *International Journal of Computer Vision 55* (2003), 155–179.

[180] Postolski, M., Couprie, M., and Janaszewski, M. Scale filtered euclidean medial axis. *Lecture Notes in Computer Science 7749* (2013), 360–371.

[181] Reddy, J. M., and Turkiyyah, G. M. Computation of 3D skeletons using a generalized Delaunay triangulation technique. *Computer-Aided Design 27* (1995), 677 – 694.

[182] Rosenfeld, A., Kong, T., and Wu, A. Digital surfaces. *CVGIP: Graphical Models and Image Processing 53* (1991), 305–312.

[183] Samet, H. A quadtree medial axis transform. *Communications of the ACM 26* (1983), 680–693.

[184] Sampl, P. Medial axis construction in three dimensions and its application to mesh generation. *Engineering with Computers 17* (2001), 234–248.

[185] Sanniti di Baja, G. On medial representations. *Lecture Notes in Computer Science 5197* (2008), 1–13.

[186] Schaefer, S., Ju, T., and Warren, J. Manifold dual contouring. *IEEE Transactions on Visualization and Computer Graphics 13*, 3 (2007), 610–619.

[187] Schwarz, M., and Seidel, H.-P. Fast parallel surface and solid voxelization on GPUs. *ACM Transactions on Graphics 29*, 6 (2010), 179:1–179:10.

[188] Sebastian, T., Klein, P., and Kimia, B. Recognition of shapes by editing their shock graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence 26*, 5 (2004), 550–571.

[189] Shaked, D., and Bruckstein, A. Pruning medial axes. *Computer Vision and Image Understanding 69* (1998), 156–169.

[190] Shamir, A., and Shaham, A. Skeleton based solid representation with topology preservation. *Graphical Models 68*, 3 (2006), 307 – 321.

[191] Sharf, A., Lewiner, T., Shamir, A., and Kobbelt, L. On-the-fly curve-skeleton computation for 3D shapes. *Computer Graphics Forum 26*, 3 (2007), 323–328.

[192] Sharir, M., and Agarwal, P. *Davenport-Schinzel sequences and their geometric applications*. Cambridge University Press, 1995.

[193] Sheehy, D. J., Armstrong, C. G., and Robinson, D. J. Computing the medial surface of a solid from a domain Delaunay triangulation. In *Proceedings of the third ACM symposium on Solid modeling and applications* (1995), pp. 201–212.

[194] Sherbrooke, E. C., Patrikalakis, N. M., and Brisson, E. An algorithm for the medial axis transform of 3D polyhedral solids. *IEEE Transactions on Visualization and Computer Graphics 2* (1996), 44 – 61.

[195] Sherbrooke, E. C., Patrikalakis, N. M., and Wolter, F.-E. Differential and topological properties of medial axis transforms. *Graphical Models and Image Processing 58*, 6 (1996), 574–592.

[196] Shinagawa, Y., and Kunii, T. Constructing a reeb graph automatically from cross sections. *IEEE Computer Graphics and Applications 11* (1991), 44–51.

[197] Shinagawa, Y., Kunii, T., and Kergosien, Y. Surface coding based on morse theory. *IEEE Computer Graphics and Applications 11* (1991), 66–78.

[198] Siddiqi, K., and Pizer, S. *Medial Representations: Mathematics, Algorithms and Applications*. Springer, 2008.

[199] Sobiecki, A., Yasan, H., Jalba, A., and Telea, A. Qualitative comparison of contraction-based curve skeletonization methods. In *Proceedings of International Symposium on Mathematical Morphology* (2013), Springer.

[200] Solís Montero, A., and Lang, J. Skeleton pruning by contour approximation and the integer medial axis transform. *Computers & Graphics 36*, 5 (2012), 477–487.

[201] Srinivasan, V., and Nackman, L. Voronoi diagram for multiply-connected polygonal domains. *IBM Journal of Research and Development 31*, 3 (1987), 361–372.

[202] Stolpner, S., and Whitesides, S. Medial axis approximation with bounded error. In *Proceedings of the 2009 Sixth International Symposium on Voronoi Diagrams* (2009), pp. 171–180.

[203] Storti, D. W., Turkiyyah, G. M., Ganter, M. A., Lim, C. T., and Stal, D. M. Skeleton-based modeling operations on solids. In *Proceedings of the fourth ACM symposium on Solid modeling and applications* (1997), pp. 141–154.

[204] Stroud, I., Renner, G., and Xirouchakis, P. A divide and conquer algorithm for medial surface calculation of planar polyhedra. *Computer-Aided Design 39* (2007), 794–817.

[205] Sud, A. *Efficient computation of discrete Voronoi diagram and homotopy-preserving simplified medial axis of a 3D polyhedron.* PhD thesis, University of North Carolina at Chapel Hill, 2006.

[206] Sud, A., Foskey, M., and Manocha, D. Homotopy-preserving medial axis simplification. In *Proceedings of the 2005 ACM symposium on Solid and physical modeling* (2005), pp. 39–50.

[207] Sudhalkar, A., Gürsöz, L., and Prinz, F. Continuous skeletons of discrete objects. In *Proceedings on the second ACM symposium on Solid modeling and applications* (1993), pp. 85–94.

[208] Sudhalkar, A., Gürsöz, L., and Prinz, F. Box-skeletons of discrete solids. *Computer-Aided Design 28*, 6–7 (1996), 507 – 517.

[209] Sugihara, K., and Hayashi, Y. Automatic generation of 3D building models with multiple roofs. *Tsinghua Science & Technology 13* (2008), 368 – 374.

[210] Tagliasacchi, A. Skeletal representations and applications. Tech. rep., Simon Fraser University, 2013.

[211] Tagliasacchi, A., Alhashim, I., Olson, M., and Zhang, H. Mean curvature skeletons. *Computer Graphics Forum 31*, 5 (2012), 1735–1744.

[212] Tagliasacchi, A., Zhang, H., and Cohen-Or, D. Curve skeleton extraction from incomplete point cloud. *ACM Transactions on Graphics 28*, 3 (2009), 71:1–71:9.

[213] Tam, R., and Heidrich, W. Shape simplification based on the medial axis transform. In *IEEE Visualization* (2003), pp. 481–488.

[214] Tamassia, R. A dynamic data structure for planar graph embedding. *Lecture Notes in Computer Science 317* (1988), 576–590.

[215] Tanase, M., and Veltkamp, R. C. A straight skeleton approximating the medial axis. *Lecture Notes in Computer Science 3221* (2004), 809–821.

[216] Vergés, E., Ayala, D., Grau, S., and Tost, D. Virtual porosimeter. *Computer-Aided Design and Applications 5* (2008), 557–564.

[217] Vigo, M., Pla, N., Ayala, D., and Martinez, J. Efficient algorithms for boundary extraction of 2D and 3D orthogonal pseudomanifolds. *Graphical Models 74*, 3 (2012), 61 – 74.

[218] Vyatkina, K. On the structure of straight skeletons. In *Proceedings of the International Conference on Computational Sciences and Its Applications* (2008), pp. 452–460.

[219] Vyatkina, K. Linear axis for planar straight line graphs. In *Proceedings of the fifteenth Australasian Symposium on Computing* (2009), pp. 139–152.

[220] Wan, M., Dachille, F., and Kaufman, A. Distance-field based skeletons for virtual navigation. In *Proceedings of the conference on Visualization* (2001), pp. 239–246.

[221] Wang, S., Wu, J., Wei, M., and Ma, X. Robust curve skeleton extraction for vascular structures. *Graphical Models 74*, 4 (2012), 109–120.

[222] Wang, Y.-S., and Lee, T.-Y. Curve-skeleton extraction using iterative least squares optimization. *IEEE Transactions on Visualization and Computer Graphics 14* (2008), 926 –936.

[223] Ward, A. D., and Hamarneh, G. The groupwise medial axis transform for fuzzy skeletonization and pruning. *IEEE Transactions on Pattern Analysis and Machine Intelligence 32* (2010), 1084–1096.

[224] Whitehead, J. Simplicial spaces, nuclei and m-groups. *Proceedings of the London mathematical society 2*, 1 (1939), 243–327.

[225] Wolter, F. Cut locus and medial axis in global shape interrogation and representation. In *MIT Design Laboratory Memorandum 92-2* (1992).

[226] Wong, W.-T., Shih, F. Y., and Su, T.-F. Thinning algorithms based on quadtree and octree representations. *Information Sciences 176*, 10 (2006), 1379 – 1394.

[227] Wood, D. The contour problem for rectilinear polygons. *Information Processing Letters 19*, 5 (1984), 229–236.

[228] Xie, W., Thompson, R., and Perucchio, R. A topology-preserving parallel 3D thinning algorithm for extracting the curve skeleton. *Pattern Recognition 36* (2003), 1529–1544.

[229] Yang, Y., Brock, O., and Moll, R. N. Efficient and robust computation of an approximated medial axis. In *Proceedings of the ninth ACM symposium on Solid modeling and applications* (2004), pp. 15–24.

[230] Yap, C. An $O(n \log n)$ algorithm for the Voronoi diagram of a set of simple curve segments. *Discrete and Computational Geometry 2*, 1 (1987), 365–393.

[231] Zhang, T. Y., and Suen, C. Y. A fast parallel algorithm for thinning digital patterns. *Communications of the ACM 27* (1984), 236–239.

[232] Zhou, Y., Kaufman, A., and Toga, A. W. Three-dimensional skeleton and centerline generation based on an approximate minimum distance field. *The Visual Computer 14* (1998), 303–314.

[233] Zhou, Y., and Toga, A. Efficient skeletonization of volumetric objects. *IEEE Transactions on Visualization and Computer Graphics 5* (1999), 196–209.