Dipl.-Ing. Gernot Christian WALZL

STRAIGHT SKELETONS
From Plane to Space

Doctoral Thesis

Graz University of Technology

Supervisor:
Univ.-Prof. Dipl.-Ing. Dr.techn. Franz AURENHAMMER
Institute for Theoretical Computer Science, TU Graz

Graz, 2015

STATUTORY DECLARATION

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Date	Signature

Abstract

The straight skeleton is a piecewise linear structure that is defined by a self-parallel offsetting process. Given a polygon in the plane, the offsetting process shifts each edge in a self-parallel manner. By following the moving vertices of the polygon, the straight skeleton is created.

The present work starts with an explanation of the problems that arise during the offsetting process. Moreover, it provides an overview of the most important work on straight skeletons in the plane. The straight skeleton in the plane has got attention in the field of computational geometry due to the work by Aichholzer et al. in 1995 [AAAG95].

Up to now, little attention has been paid to the straight skeleton in space. The focus of the present work is the investigation of the straight skeleton of general polyhedra in 3-space. In 3-space, each facet of a given polyhedron is shifted in a self-parallel manner. The straight skeleton is defined by following the moving edges and vertices of the polyhedron. Compared to the offsetting process in the plane, the solution of the offsetting process in 3-space turns out to be less intuitively. The present work proves the existence of a solution. In contrast to the straight skeleton in the plane, the solution is not unique any more. A simple example is provided to show this ambiguity. Furthermore, an algorithm is presented to find all possible offset polyhedra. The ambiguity is used to optimize the offset polyhedron (e.g. maximize volume with a given offset).

During the offsetting process, a polyhedron experiences combinatorial and topological changes. Such changes are caused by events (e.g. the length of an edge shrinks to zero). All possible events for non-degenerate polyhedra are analyzed and categorized.

All presented algorithms have been implemented in C++. The implementation is used to generate experimental results. This implementation also features an interactive animation of the offsetting process as well as the capability to render figures for the paper.

Keywords: polygon offsetting, polyhedron offsetting, shrinking process, motorcycle graph, roof construction, wavefront propagation

Acknowledgments

I would like to thank my supervisor Franz Aurenhammer for making all of this possible. We had inspiring and fruitful discussions that lead to many results.

I thank my parents, Brigitte and Norbert, for their support trough my entire life. This thesis would not exist without their financial support.

Thanks to Wolfgang Aigner who kindly provided examples of polyhedra.

Special thanks go to the European Science Foundation (ESF) (especially to Günter Rote) for funding my travel expenses to the Symposium on Computational Geometry (SoCG) in Kyoto, Japan.

Graz, 2015 Gernot Walzl

Contents

1 Introduction

1.1 Skeletal Structures

In biology, the skeleton defines basic properties of an animal. The joints of the bones give hints on how the animal might move. The skeleton tells how strong the animal might be. It shows if the animal might be able to fly. It somehow defines the way of life of an animal. Because it defines basic properties of animals, it is used to classify them. To give an example, all birds have a skeletal structure that is comparable.

When the life of an animal is over, all that remains is its skeleton. A major part of the scientific knowledge of dinosaurs is based on the skeletons, which were found. Thus, it is obvious how fundamentally important skeletal structures for biology and archeology are.

Similar applications of skeletal structures are found in computer science. Such applications include shape matching in computer vision, animation techniques in computer graphics and many more. All of them require a skeletal representation of geometric objects. This is where computational geometry provides the tools.

There are various skeletal representations of geometric objects. Depending on the application, a specific skeletal structure may have advantages over another one. An overview of the most common skeletal structures is given in the next subsections.

1.1.1 Medial Axis

The most widely used skeletal structure is the medial axis. The medial axis of an object is defined as the set of centers of maximal disks contained by the object. A disk is maximal when it touches the boundary of the object at least at two points. An example of the medial axis of a polygon is shown in Figure 1.1.

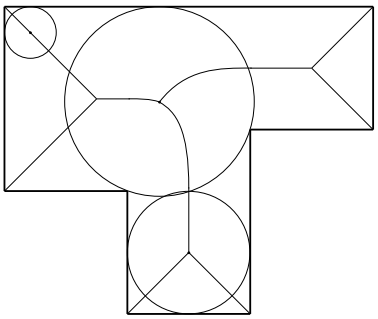


Figure 1.1: Medial axis of a polygon and defining circles

The medial axis was introduced by Blum in the year 1967 [Blu67]. His work was motivated by a question in biology: How do animals or humans perform a selection or a classification on geometric shapes? There are countless many different shapes. A set of basic attributes should be sufficient to perform a selection. The medial axis transformation was presented to extract such basic attributes from geometric shapes. At the end of his work, he used an optical device to visualize grassfire propagation by using photographic defocusing.

In 1987, Aggarwal, Guibas, Saxe and Shor found an algorithm to compute the medial axis of convex polygons in $\Theta(n)$ time and space, where n denotes the size of the polygon [AGSS87]. This time bound is optimal because the combinatorial size of the medial axis is $O(n)$.

In 1995, Chin, Snoeyink and Wang found an algorithm that computes the medial axis of simple polygons in linear time [CSW95]. Their algorithm decomposes a given polygon into so-called *xy monotone histograms*. The medial axis of xy monotone histograms is computed in linear time. The individual medial axes of the histograms are merged to obtain the entire medial axis of the given polygon. Overall, the required time of this algorithm is linear in the size of the polygon.

1.1.2 Straight Skeleton

Another skeletal structure is the straight skeleton. Compared to the medial axis, it consists only of straight line segments. There are applications where this property is mandatory. To give an example, the edges of the roofs of ordinary houses are straight line segments.

The straight skeleton is defined by an offsetting process where each edge is shifted inwards in a self-parallel manner. By following the vertices, the straight skeleton is created. An example is shown in Figure 1.2.

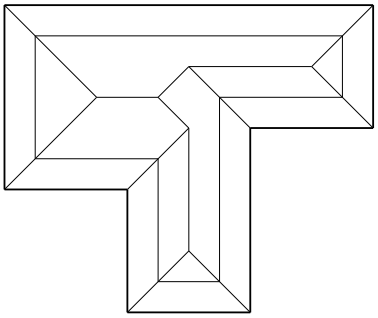


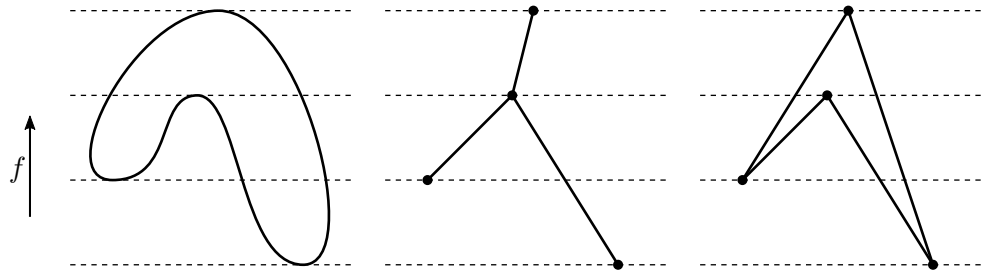
Figure 1.2: Straight skeleton of the same polygon (Fig. 1.1) and a defining offset polygon

Further explanations about the straight skeleton of polygons in the plane are given in Chapter 2.

1.1.3 Reeb Graph

The Reeb graph is a topological skeleton of a given manifold. It is an abstract graph that does not define an embedding. The Reeb graph was introduced by Georges Reeb in 1946 as part of the Morse theory [Ree46].

The height function f is a smooth, scalar-valued function on the manifold. A node of the Reeb graph corresponds to a point where all partial derivatives of f are zero, $\nabla f = 0$. An example is shown in Figure 1.3.



(a) Manifold **(b)** Reeb graph **(c)** Reeb graph of boundary

Figure 1.3: Reeb graph

A typical application for the Reeb graph is a dip-coating process. To prevent corrosion, objects of metal are covered by coatings. The dip-coating process moves an object through a bath of liquid coating. The requirement for the object is that the entire surface should be covered by a uniformly distributed coating. Depending on the geometric shape of the object, there might be bubbles of air when the object is completely dipped into the bath. When the object is removed from the bath, too much liquid might be taken away. The Reeb graph helps to find such potential problems beforehand.

1.2 Outline

The present work is organized as follows. The next chapter explains properties and algorithms of the straight skeleton of polygons in the plane. Starting with an explanation of basic construction algorithms, it moves on to advanced algorithms, and ends with a summary of recent results. Moreover, important properties of the straight skeleton in the plane are highlighted.

In the third chapter, the straight skeleton of polyhedra in 3-space is examined. Papers on the straight skeleton in 3-space are found very rarely. In 3-space, the problem is slightly different. Facets of a given polyhedron are shifted in a way that keeps them parallel to their initial orientation. The essential problem arises at the very first moment. Vertices with a degree higher than three need to be split. Chapter 3 investigates this problem in detail and provides an algorithm to solve it.

Chapter 4 explains data structures and an algorithm to implement the computaion of straight skeletons of polyhedra. The implementation features an interactive animation of the offsetting process as well as the capability to create figures for the paper. In addition, it is explained how the data structures in memory are stored on disk for further analysis. Experimental results are evaluated at the end of this chapter.

Complete examples of shrinking polyhedra are depicted in Appendix A. Equations for geometric operations, like the intersection of planes, angle bisector, etc., are found in Appendix B.

Parts of this work have been published during various conferences and are contained in the proceedings of these conferences [AW13a, AW13b, AW14].

2 Straight Skeletons in the Plane

The straight skeleton in the plane is defined by an offsetting process. This offsetting process shifts every edge of the polygon in a self-parallel manner and at the same time. When every edge is shifted inwards, the polygon shrinks. Therefore, this process is also called *shrinking process*. During the shrinking process, *arcs* are created by following the vertices of the shrinking polygon. A *node* is created when the shrinking process requires combinatorial or topological changes of the polygon. A combinatorial change is required when an edge vanishes. When the polygon is split into two parts, a topological change of the polygon occurs.

The straight skeleton partitions the polygon into cells. These cells have useful properties, which are explained later in this chapter. Figure 2.1 shows an example of a shrinking, convex polygon and the resulting straight skeleton.

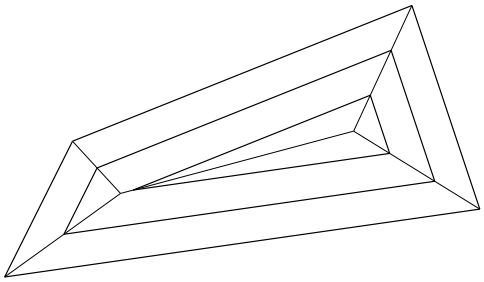


Figure 2.1: Offsetting a polygon

The first ideas on this topic were written down in the 19th and 20th century. When constructing a roof of a house, it is mandatory that the water of raindrops always finds a way to the ground [vP77, Mü16].

The first known algorithm to construct a *bisector skeleton* was explained by Brassel, Heller and Jones in the year 1984 [BHJ84]. Their bisector skeleton is equal to the unweighted case of the straight skeleton in the plane.

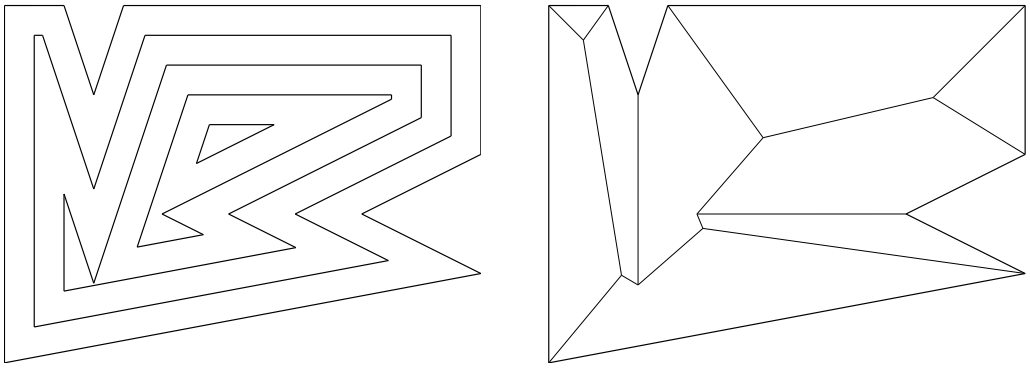
The straight skeleton in the plane became widely known by the work of Oswin Aichholzer, Franz Aurenhammer, David Alberts and Bernd Gärtner in 1995 [AAAG95]. This work defined the straight skeleton in a procedural way. Based on this definition, a construction algorithm handles events that occur during the shrinking process in chronological order. The events require combinatorial or topological changes to the shrinking polygon. Furthermore, fundamental properties of the straight skeleton were proven.

2 Straight Skeletons in the Plane

This chapter provides an overview of the most important work on straight skeletons in the plane. Various algorithms to construct a straight skeleton are explained here. The difficulties of constructing the skeleton efficiently are highlighted. At the end of this chapter, properties and applications of straight skeletons in the plane are summarized.

2.1 Simple Polygons

This section is devoted to the fundamental properties of the straight skeleton found in [AAAG95]. At the beginning of this chapter, it was mentioned that the straight skeleton is defined by a shrinking process. Every edge of the polygon is shifted inwards in a self-parallel manner. When all edges are shifted with the same speed, the vertices of the polygon move on angle bisectors of incident edges. The edges change their length during the shrinking process. This depends on the angle between adjacent edges. When a reflex vertex meets an edge, the edge is split. An example of the shrinking process is shown in Figure 2.2.



(a) Offset polygons **(b)** Straight skeleton

Figure 2.2: Straight skeleton of a simple polygon (input from [AAAG95])

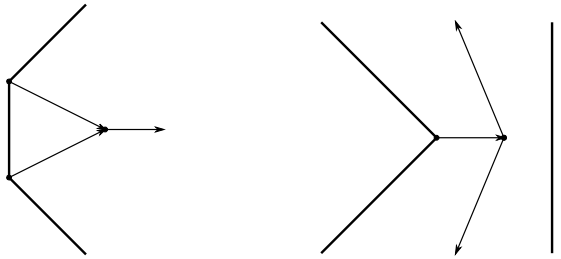
The shrinking process is used to define the straight skeleton. To compute the straight skeleton, the computation of offset polygons is not required. It is sufficient to know which events change the polygon during the shrinking process. The order of the events is substantial for the structure of the straight skeleton.

2.1.1 Events

Events change the polygon during the shrinking process. There are two types of events. The vanishing of an edge induces combinatorial changes on the polygon. This event is called an *edge event*. When a reflex vertex meets an edge, the polygon is split. The so-called *split event* changes the polygon topologically.

Edge Event

The position where an edge vanishes is determined by the intersection of the supporting lines of its adjacent skeleton arcs. If all edges are shifted with the same speed, this is the intersection of its adjacent angle bisectors. The time of the event is determined by the perpendicular distance from the edge to the corresponding edge



(a) Edge event **(b)** Split event

Figure 2.3: Events

event. Only information from the local structural neighbourhood is used to compute this event.

A shrinking triangle has three edges that vanish at the same time and at the same position. If all edges are shifted with the same speed, this position is the center of the incircle.

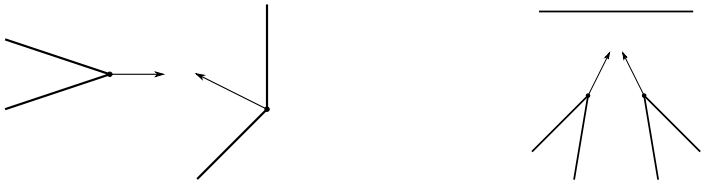
Split Event

When a reflex vertex meets an edge of the polygon, a split event happens. This event splits the edge and, very likely, the polygon. To compute the next split event, it is not sufficient to investigate the local structural neighbourhood of the reflex vertex. A reflex vertex can meet any non-incident edge of the polygon.

The position of a split event is, like the edge event, determined by the intersection of supporting lines of skeleton arcs.

Another way of computing the position is the following. During the shrinking process, the vertex moves on a line. By determining start position and the speed of the vertex and the start position and speed of the edge on this line, the position of the split event can be computed.

When the split events are computed, it is important to pay attention to changes of the polygon during the shrinking process. Cases that need special attention are illustrated in Figure 2.4.



(a) The ray of the reflex vertex points to the vertical edge. This edge may not be involved in the split event.

(b) The path of a reflex vertex may be cut off by another reflex vertex.

Figure 2.4: Challenging split events

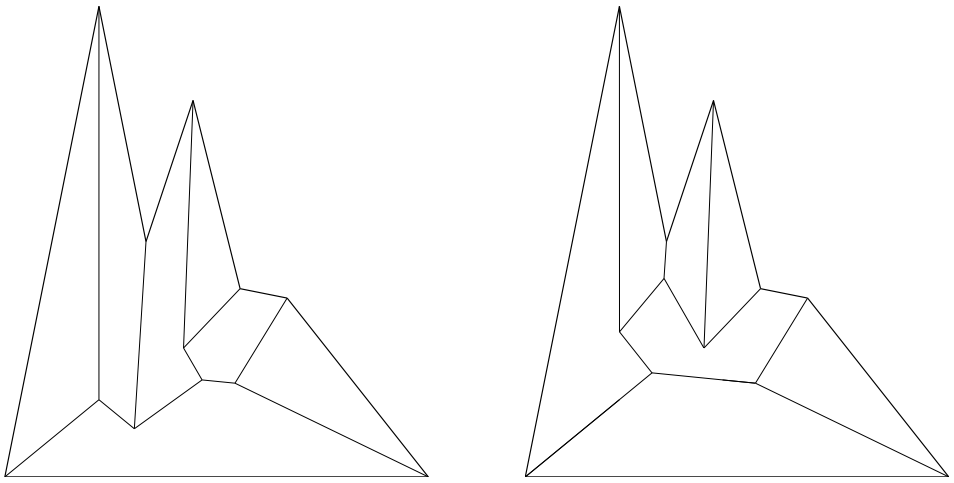
2.1.2 Bisector Graphs

Two edges of a polygon define the supporting line of one edge of the bisector graph. The edges of the bisector graph are also called *arcs*, like the *arcs* of the straight skeleton. The supporting line of an arc halves the angle between the supporting lines of two edges. The arcs are bounded by the vertices of the polygon and by the nodes of the bisector graph. Each node is the intersection point of (at least) three bisectors. In the present work, only bisector graphs are of interest that are:

1. planar (free of crossings)
2. completely inside the polygon
3. do not introduce cycles

Even with these restrictions, the definition of bisector graphs is ambiguous. An example is given in Figure 2.5.

Every straight skeleton is a bisector graph. In contrast to the ambiguous definition of bisector graphs, the straight skeleton is uniquely defined by the shrinking process.



(a) Bisector graph I (straight skeleton) **(b)** Bisector graph II

Figure 2.5: Planar bisector graphs are ambiguous (based on Fig. 2 in [AAAG95])

The straight skeleton of a polygon without holes is an unrooted binary tree. The leaves of this tree represent the n vertices of the polygon. An unrooted binary tree has $n - 2$ nodes and $2n - 3$ arcs.

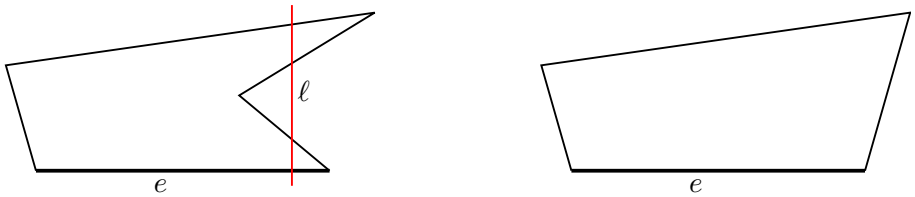
2.1.3 Roof Model

The roof model is an interpretation of the straight skeleton as the roof of a house. The ground plan of the house is used as polygon for the input. The time of the shrinking process is interpreted as third spatial dimension (height). Each offset polygon is understood as a contour line that marks positions with an equal height.

The interpretation of the offset as height is used to prove properties of the straight skeleton [AAAG95]. Every raindrop that hits a facet of the roof (cell of the straight skeleton) is able to drain off at the corresponding edge of the cell. For this reason, the straight skeleton can be used to construct roofs.

2.1.4 Monotonicity

A cell is monotone in a direction if, and only if, an intersection with any line that is perpendicular to this direction results in a line segment, which is connected. An example of this property is given in Figure 2.6.



(a) Not monotone **(b)** Monotone

Figure 2.6: Monotonicity in direction of edge e

A property of plane bisector graphs is the monotonicity of its cells. The roof model of the unweighted straight skeleton has the same slope for every roof facet (cell). A line on a roof facet maximizes its slope when it is perpendicular to the defining edge. The monotonicity of the cells is proven by contradiction [AAAG95]. To create a contradiction, it is assumed that a cell is not monotone. A line that maximizes its slope leaves the roof facet and re-enters the same facet at a higher point. In between these two points, the line is projected vertically onto the roof. The slope of the projection is less than the slope of its originating line. When the line re-enters its corresponding roof facet, the roof would not be continuous at that point. This leads to a contradiction because every plane bisector graph can be interpreted as a roof that is continuous [AAAG95]. Therefore, each cell is monotone in direction of its defining edge.

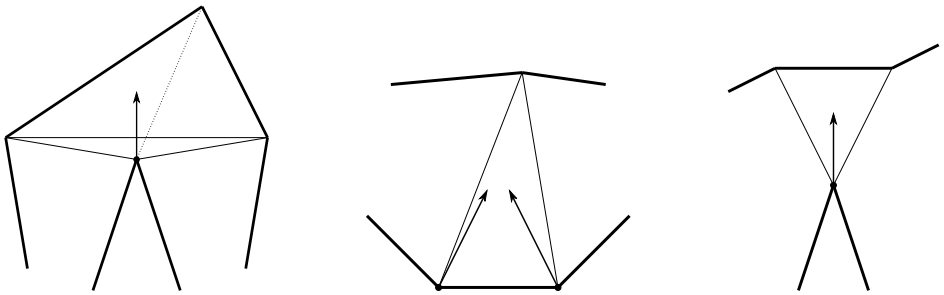
2.2 Triangulated Polygons

An efficient and easy-to-implement algorithm to construct the straight skeleton in the plane is explained by Aichholzer und Aurenhammer in [AA96]. A triangulated polygon is given as input. When the polygon shrinks, the triangles of the triangulation stay connected with the moving vertices until an event occurs.

The triangulation helps to find the next event to construct the straight skeleton. The triangles create a structure where every event is found in a local structural neighbourhood. The position of every event is determined by the collapsing of a triangle. A triangle collapses when all three points of the triangle are on a line.

To keep the triangulation valid during the shrinking process, an additional event is introduced: The flip event handles the necessary flip of an edge inside a quadrangle. This event does not directly influence the resulting straight skeleton.

Edge events and split events of the straight skeleton are also found when a triangle collapses. Figure 2.7 gives an example for every event and shows collapsing triangles.



(a) Flip event **(b)** Edge event **(c)** Split event

Figure 2.7: Collapsing triangles and corresponding events

A priority queue is used to find the next event. The time of the collapse is used as priority.

A triangle collapses when its (signed) area equals zero. In the following equations, the time is denoted by t. All three points of the triangle $p_i(t) = (x_i(t), y_i(t)), i \in \{1, 2, 3\}$ start at position $p_i(0)$. During the shrinking process, the points move on angle bisectors. Their velocities are denoted by $v_i = (v_{xi}, v_{yi})$. Equation 2.1 is quadratic in time t and has at most two solutions for t.

$$\frac{1}{2}\begin{vmatrix} x_1(0) + t \cdot v_{x1} & x_2(0) + t \cdot v_{x2} & x_3(0) + t \cdot v_{x3} \\ y_1(0) + t \cdot v_{y1} & y_2(0) + t \cdot v_{y2} & y_3(0) + t \cdot v_{y3} \\ 1 & 1 & 1 \end{vmatrix} = 0 \tag{2.1}$$

The expected time for computing the straight skeleton of ordinary polygons using this algorithm is $O(n \log n)$. The proven upper bound for the time complexity is $O(n^3 \log n)$ [AA96]. This time complexity is caused by the number of possible flip events. The triangulation influences the number of required flip events.

2.3 Weighted Straight Skeletons

If all edges of a given polygon propagate at the same speed, the roof model consists of facets with equal slopes. Aichholzer and Aurenhammer have generalized the concept of straight skeletons to generate roofs and terrains with prescribed facet slopes [AA96]. The propagation speed of individual edges is adjusted to match the required facet slopes. This may change the topological structure of the straight skeleton. Furthermore, basic properties of the straight skeleton are lost. Figure 2.8 shows an example where a cell is not monotone in direction of its incident edge.

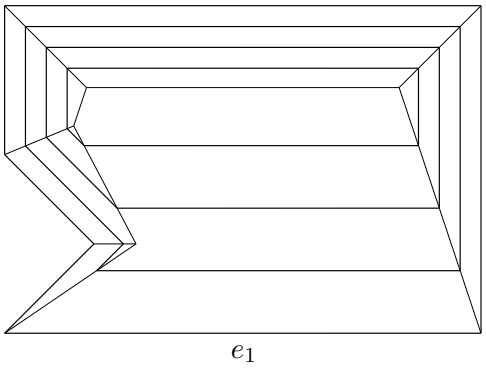


Figure 2.8: Edge e_1 propagates faster than any other edge. The cell belonging to edge e_1 is not monotone in direction of its edge.

Besides the monotonicity of cells, the weighted straight skeleton of a polygon with holes may have cells with holes. This applies if an edge propagates fast over a hole of the polygon. The created cell encloses this hole.

Huber has observed that the definition of the straight skeleton is ambiguous when two parallel edges with different weights became adjacent [Hub11]. This is visualized in Figure 2.9. Edge e_1 is slightly tilted. Edge e_3 propagates twice as fast as edge e_1. When edge e_1 and edge e_3 become parallel, the straight skeleton is ambiguous.

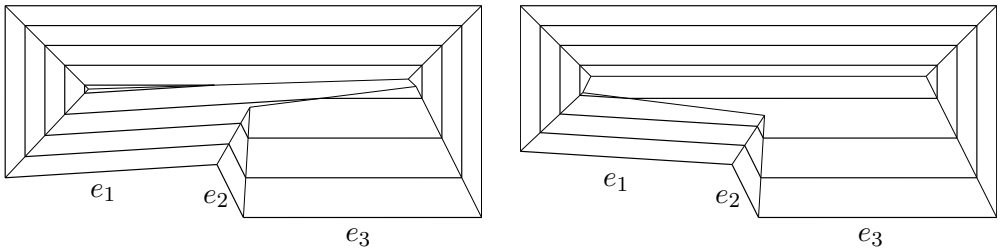


Figure 2.9: The weighted straight skeleton is ambiguous when edge e_1 and edge e_3 become parallel.

2.4 Efficient Data Structures: Half-Space Range Searching

Eppstein and Erickson were the first who have used efficient half-space data structures to compute the straight skeleton in the plane [EE99]. Their paper shows how to find pairwise interactions between objects using efficient data structures. Their time complexity analysis shows a subquadratic upper bound for the computation of the straight skeleton. It can be computed in $O(n^{1+\varepsilon} + n^{8/11+\varepsilon}r^{9/11+\varepsilon})$ time, given a polygon with n vertices and r reflex vertices. ε is an arbitrary small positive constant. This constant hides various other constants used by underlying algorithms.

Prior to understanding their algorithm for the computation of the straight skeleton, a simpler problem must be explained. The explanation begins with a given set of rays in 3-space. As the result, the lowest intersection for a query triangle is determined.

Their algorithm is based on a multi-level data structure. The first level is a half-space range searching data structure by Matoušek [Mat93]. This data structure is initialized with the given set of rays in 3-space. A triangle in 3-space has an oriented supporting plane. The half-space data structure efficiently finds rays that intersect the supporting plane of the query triangle.

The next level of this multi-level data structure efficiently finds intersections inside the triangle. The three edges of the triangle have three supporting lines. Using the relative orientations between the supporting line of the ray and these three supporting lines, it is determined whether the ray intersects the triangle. This is achieved with a data structure by Agarwal and Matoušek [AM94].

The relative orientation of two lines in 3-space is calculated by the (signed) volume of a tetrahedron. The sign of the result determines the orientation of the lines. Two points on a line that are not on the same position define a direction. Two lines and two points on each line span a tetrahedron with four points $p_i = (x_i, y_i, z_i), i \in \{1, ..., 4\}$. Equation 2.2 shows the determinant to calculate the (signed) volume of a tetrahedron.

$$V = \frac{1}{6}\begin{vmatrix} x_1 & x_2 & x_3 & x_4 \\ y_1 & y_2 & y_3 & y_4 \\ z_1 & z_2 & z_3 & z_4 \\ 1 & 1 & 1 & 1 \end{vmatrix} \tag{2.2}$$

Parametric search is used to find the lowest intersecting ray. The relative orientation of a fourth horizontal line is used to find the lowest intersection. This level uses a data structure by Chazelle et al. [CEG+96].

All of the involved data structures can efficiently handle a changing set of rays. Rays need to be removed during the computation of the straight skeleton. By interpreting the time of the shrinking process as a third spatial dimension, the straight skeleton becomes a roof model [AAAG95].

At the beginning of the shrinking process, no interactions or events have taken place. Every edge of the given polygon defines a triangle in 3-space, which is possibly unbounded. The initial direction of incident vertices define the bounds of this triangle. Every reflex vertex of the given polygon defines a ray in 3-space. The slope

of each ray is determined by the intersection of the supporting planes of incident triangles. These triangles and rays are considered for pairwise interactions. By finding the next pairwise interaction, the next split event is found efficiently. The whole sequence of edge and split events define the resulting straight skeleton.

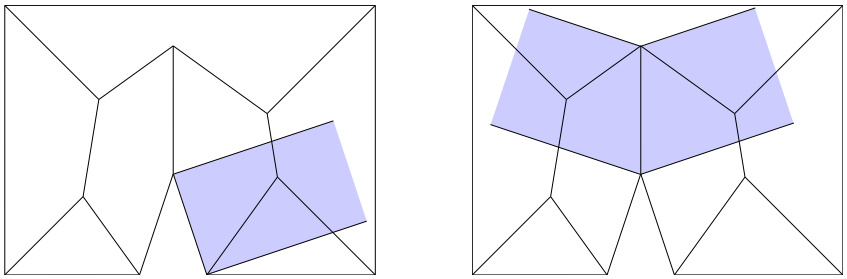
2.4.1 Lower Envelope of Slabs

Eppstein and Erickson have introduced two types of slabs in 3-space. The vertical projection of the lower envelope of these slabs onto the given polygon reveals its straight skeleton [EE99].

Each edge of the polygon defines an edge slab. It is bounded by the defining edge and two rays originating at the incident vertices of this edge. Both rays have a direction perpendicular to the defining edge. The slope of the slab is equal to the slope of the corresponding facet of the roof model.

Each reflex vertex defines two reflex slabs. A reflex slab is bounded by the edge of the roof (arc) created by the reflex vertex. Furthermore, a vertex has two incident edges that are part of the given polygon. The direction of one of these edges is used to define two rays perpendicular to the defining edge. The first ray has its origin at the position of the reflex vertex. The second ray has its origin at the position of the node on the roof created by the reflex vertex. These two rays bind one of the two reflex slabs defined by the reflex vertex.

An example of this definition is visualized in Figure 2.10.



(a) An edge defines an edge slab.

(b) A reflex vertex defines two reflex slabs.

Figure 2.10: Top view of edge and reflex slabs

Although the position of the node on the roof created by a reflex vertex is not locally defined, this model helps to compute the unweighted straight skeleton in time and space $O(n^{1+\varepsilon} + n^{8/11+\varepsilon} r^{9/11+\varepsilon})$ [EE99].

If the edges propagate with different speeds, the created roof is not a lower envelope of its slabs [EE99]. An example is shown in Figure 2.11.

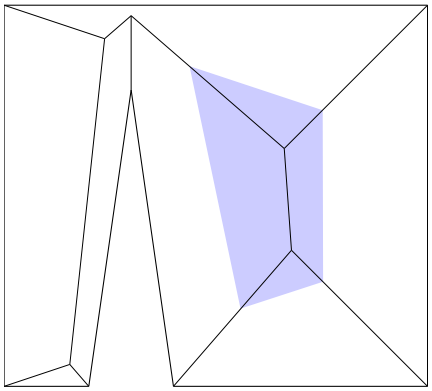


Figure 2.11: The roof of a polygon with weighted edges is not the lower envelope of its slabs. The shaded region is cut off by the edge slab of the leftmost edge. This edge moves faster than all other edges. (based on Fig. 6 in [EE99])

2.4.2 Motorcycle Graphs

A further pairwise interaction problem handled by Eppstein and Erickson is the motorcycle graph [EE99]. A motorcycle has an initial position and a velocity in the plane. Each motorcycle drives on a straight line with constant speed. While a motorcycle moves, it creates a wall where other motorcycles can not pass. An example is shown in Figure 2.12. The length of an arrow corresponds to the speed of a motorcycle.

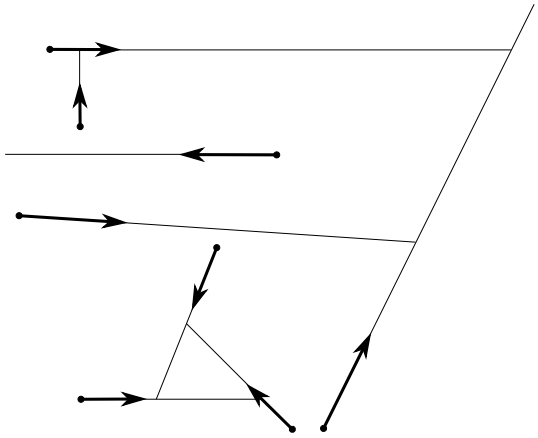


Figure 2.12: Motorcycle graph

This problem is also solved by finding pairwise interactions between rays and triangles. Again, time is interpreted as third spatial dimension. The rays have a direction that lifts the track of a motorcycle into the third dimension. At any one moment in time, the projection of the position of each motorcycle onto the corresponding ray has the same height. Below the projected tracks, triangles are placed. These triangles grow during the movement of the motorcycles in the plane. When a growing triangle hits a ray, the corresponding motorcycle crashes into the wall left by the motorcycle belonging to the ray.

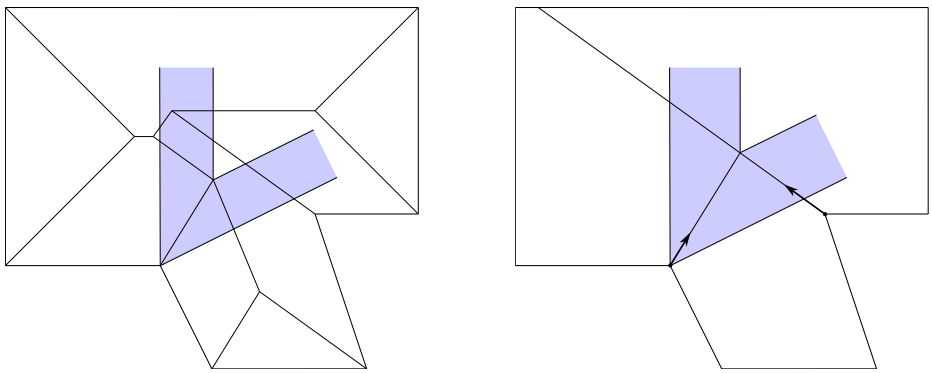
Their complexity analysis shows that the motorcycle graph can be computed in time and space $O(n^{17/11+\varepsilon})$ [EE99].

2.5 Motorcycle Graphs and Straight Skeletons

The difficulty of constructing a lower envelope of slabs is that the reflex slabs are not only defined by local properties. Cheng and Vigneron use the motorcycle graph to define motorcycle slabs [CV02]. When the motorcycle graph of a polygon is given, the motorcycle slabs are locally defined.

To construct a motorcycle graph of a polygon, each reflex vertex is replaced by a motorcycle. The velocity of each motorcycle matches the velocity of the reflex vertex during the offsetting process. These motorcycles are used to compute the motorcycle graph of the given polygon. In addition, the motorcycles run out of fuel, when they reach the boundary of a polygon.

The path of each motorcycle defines two motorcycle slabs. Each motorcycle slab is bounded by rays at the origin and destination of the motorcycle path. The direction of these rays are perpendicular to the edges that are incident to the corresponding reflex vertex. The slope of the motorcycle slab is equal to the slope of the corresponding facet of the roof model. Motorcycle slabs are compared to reflex slabs (defined in Section 2.4.1) in Figure 2.13. The roof of a given polygon is the lower envelope of edge and motorcycle slabs [CV02].



(a) Two reflex slabs **(b)** Two motorcycle slabs

Figure 2.13: Top view of reflex slabs compared to motorcycle slabs

Because all motorcycle slabs are only defined by local properties, an efficient divide-and-conquer algorithm for constructing the straight skeleton of a given polygon and its pre-computed motorcycle graph is possible.

Cheng and Vigneron describe such an algorithm, where the polygon gets divided by a canonical partition [CV02]. This partition is induced by ridge points. A ridge point is a point on an edge of the roof model. Each edge of the roof model has two incident facets. Therefore, there are two paths that a raindrop may take when it hits a ridge point. The raindrop follows the path of the steepest descent. When a raindrop hits a vertex of the roof model, there are three paths of the steepest descent. By following the steepest descents, a ridge point induces a canonical partition of the underlying polygon.

The computation of the roof model starts by computing the boundary of an arbitrary facet f. The current cell of the partition has n_c facets. All slabs of the current cell are intersected with the supporting plane of the facet f. This results in $O(n_c)$ line segments. It takes $O(n_c \log n_c)$ time to find the lower envelope of line segments [Her89]. The facet is bounded by this lower envelope.

After the facet has been computed, a vertex of this facet is chosen. The straight skeleton is an unrooted binary tree. Cheng and Vigneron define the root to be the node that balances the size of both subtrees [CV02]. A vertex (node) of the facet $v(f)$ is chosen that is structurally closest to the root of the straight skeleton. The straight skeleton does not need to be known at this step. Because the facets of the roof model are ordered, the difference between their indices modulo n_c show where the larger subtree is.

A vertical line is placed that passes through the vertex $v(f)$. This line is used to find the ridge points that induce a further partition of the polygon. The line is interpreted as the top view of a plane. All slabs of the current cell are intersected with this plane. Using the lower envelope of the resulting line segments, ridge points are determined. This is visualized in Figure 2.14.

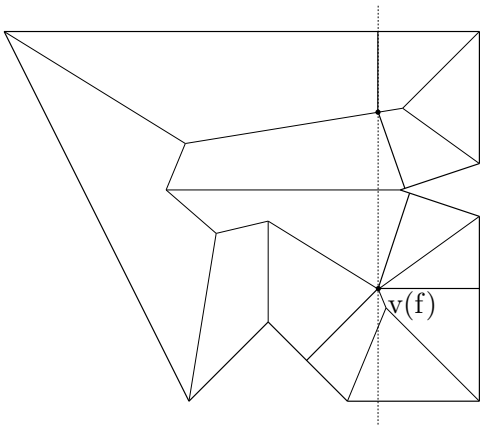


Figure 2.14: Divide and conquer (based on Fig. 8 in [CV02])

Each cell of the partition is recursively partitioned further, until the cells are small enough so that the roof model of each cell is found easily. The roof models of the cells are merged to obtain the roof model of the given polygon.

Given a polygon and its motorcycle graph, the algorithm by Cheng and Vigneron computes the straight skeleton in $O(n \log^2 n)$ time [CV02]. For a polygon with h holes, this algorithm takes $O(n\sqrt{h+1} \log^2 n)$ time to compute the straight skeleton given the polygon and its motorcycle graph [CV07].

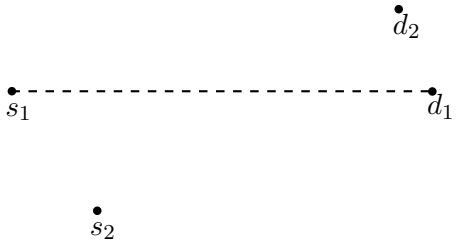
2.6 Accelerated Motorcycles

In 2013, Vigneron and Yan presented *A Faster Algorithm for Computing Motorcycle Graphs* [VY13]. Their algorithm keeps a tentative track and a confirmed track for each motorcycle. Because of the tentative tracks, the events of this algorithm might not be handled in chronological order. An event might create new events that happen earlier, but are processed later. The events are stored by a priority queue. The priority of an event is determined by the time when it happens. The priority queue finds the next event efficiently.

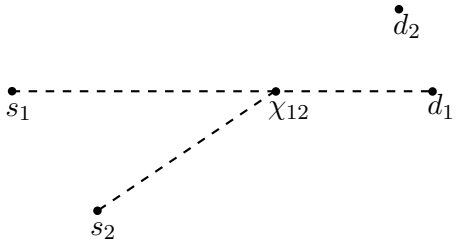
A ray-shooting data structure is used to extend tentative tracks. During the computation, this data structure ensures that no tentative tracks cross.

Each motorcycle i has a stack S_i with target points. At the beginning, each stack contains the starting point s_i and the point d_i, where the motorcycle runs out of fuel. An event of motorcycle i is denoted by (i, p), where p is the position of the event. The algorithm is explained in Figure 2.15 by giving an example.

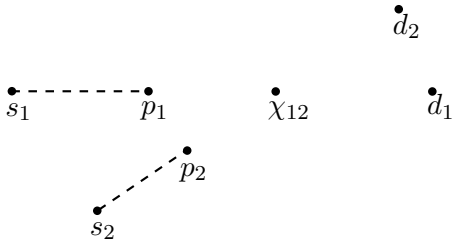
Their time complexity analysis shows that the motorcycle graph can be computed in $O(r^{4/3+\varepsilon})$ time, where r denotes the number of motorcycles [VY13].



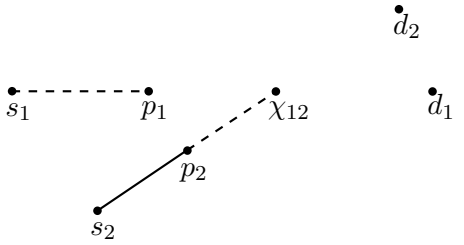
(a) At the beginning, the first event $(1, s_1)$ extends the tentative track of motorcycle 1.



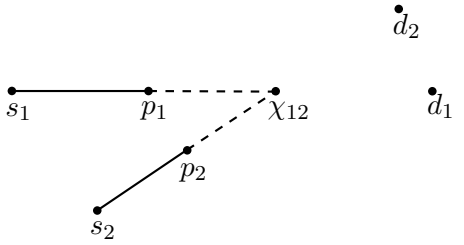
(b) Event $(2, s_2)$ extends the tentative track of motorcycle 2. By querying the ray-shooting data structure, a crossing point χ_{12} is found. Two new events $(1, \chi_{12})$ and $(2, \chi_{12})$ are created.



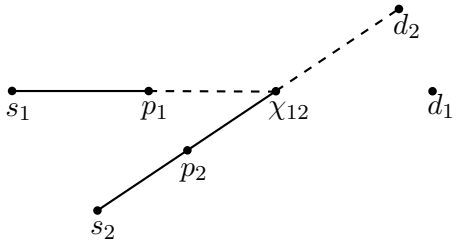
(c) When a crossing point is found, the tentative tracks of involved motorcycles are shortened. This creates two additional events $(1, p_1)$ and $(2, p_2)$



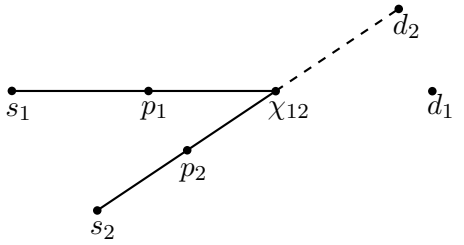
(d) Event $(2, p_2)$ extends the confirmed and tentative track of motorcycle 2.



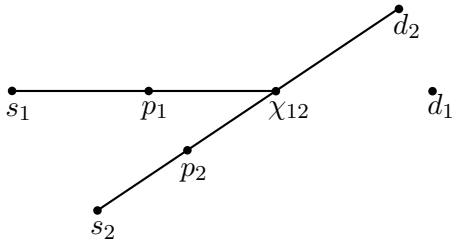
(e) Event $(1, p_1)$ extends the confirmed and tentative track.



(f) While the tentative tracks are extended, no new crossing points are found.



(g) Event $(1, \chi_{12})$ finds the destination for motorcycle 1.



(h) At the end, all motorcycles have found their destinations and no tentative tracks remain.

Figure 2.15: Example for *A Faster Algorithm for Computing Motorcycle Graphs* [VY13]. In this example, both motorcycles have the same speed.

2.7 Properties

As its name implies, the straight skeleton consists only of straight line segments. A remarkable property of the unweighted case (all edges propagate with the same speed) is the monotonicity of its cells. Monotonicity is of use in various algorithms.

2.7.1 Size

The size of the straight skeleton is linear in the size of the given polygon. If the polygon has no holes, the structure of the straight skeleton is an unrooted binary tree. For a polygon with n vertices, its straight skeleton has exactly n connected facets, $n-2$ nodes and $2n-3$ arcs [AAAG95].

If the given polygon has holes, its straight skeleton is not an unrooted binary tree. Each hole introduces a cycle into the structure of the straight skeleton. For a polygon with h holes, there are $n-2+2h$ nodes and $2n-3+3h$ arcs.

2.7.2 Time Complexity

A gap still exists between the lower and the upper bound for the time required for computing the straight skeleton of a polygon.

Lower Bound

A trivial lower bound for computing the straight skeleton is given by its size, which is linear in the size of the given polygon. For convex polygons, the unweighted straight skeleton is equal to its medial axis. The medial axis of convex polygons can be computed in time linear in the size of the input [AGSS87].

For polygons with holes, there is a lower bound, which results from a reduction to sorting [Hub11]. As it is well known, sorting of n numbers requires at least $\Omega(n \log n)$ time. (There are $n!$ possible permutations. A lower bound for the factorial is $n! \geq \left(\frac{n}{2}\right)^{n/2}$. The height of a balanced tree is logarithmic in the number of its leaves.)

The straight skeleton is used for sorting numbers as follows: For each number, a triangle is placed on a line. The distance of each triangle to a predefined origin is equal to the value of the corresponding number.

After the triangles are placed, a bounding box is drawn that includes all triangles. An example of this construction is shown in Figure 2.16. When the straight skeleton is computed, the sorted numbers are obtained by traversing the arcs of the cell on top. This reduction proves the lower bound for computing the straight skeleton of polygons with holes, which is $\Omega(n \log n)$.

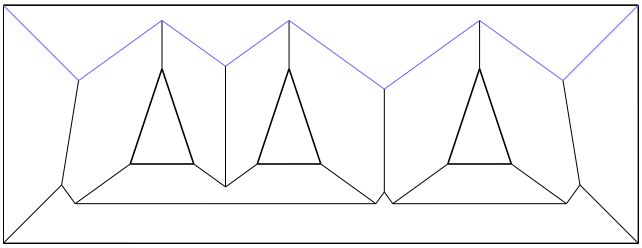


Figure 2.16: Lower bound for computing the straight skeleton of a polygon with holes (based on Fig. 10 in [Hub11])

Upper Bound

The upper time bound for computing the straight skeleton of polygons was lowered over time. Table 2.1 shows the most important algorithms and their time complexity.

Algorithm	**Time**	**Notes**
[AAAG95]	$O(nr \log n)$	Priority queue to process events sequentially
[AA96]	$O(n^3 \log n)$ shows $O(n \log n)$ in most cases	Structural locality due to triangulation
[EE99]	$O(n^{1+\varepsilon} + n^{8/11+\varepsilon} r^{9/11+\varepsilon})$	Subquadratic time bound with various algorithms involved
[CV02, CV07]	$O(n\sqrt{h+1} \log^2 n + r\sqrt{r} \log r)$	Randomized algorithm reduces straight skeleton computation to motorcycle graph computation
[VY13]	$O(n\sqrt{h+1} \log^2 n + r^{4/3+\varepsilon})$	Faster motorcycle graph
[CMV14]	$O(n(\log n) \log r + r^{4/3+\varepsilon})$	Deterministic algorithm for reduction

Table 2.1: Evolution of the upper time bound for computing the straight skeleton of a polygon with n vertices, of which r are reflex. The polygon has h holes. ε denotes a small positive constant, which hides various constants of involved algorithms.

2.8 Applications

2.8.1 Road Centerlines

From an airplane, an areal image of a city is acquired easily. In this image, all streets have an intentional width. A skeletal representation of the streets is essential for many applications. For instance, routing for car navigation needs the streets represented as abstract graph. The nodes of this graph reflect geometric important positions, such as crossings. These nodes are connected with weighted edges. The weight of the edge can be the geometric distance between the nodes or the time required to get from one node to the other. Dijkstra's algorithm is used to compute the shortest path between two nodes.

Haunert and Sester use the straight skeleton to compute road centerlines given a polygonal representation of roads [HS08]. These centerlines can be used to find various paths (shortest, fastest) between two points of interest or to draw a simple street map. For street maps, it is preferable when all streets are drawn with an equal width.

2.8.2 Geometric Modelling

Geometric models for three-dimensional computer graphics are typically represented as a mesh of the surface. This mesh usually consists of a number of triangles and their vertices. Such a representation of geometry is inefficient when the geometry consists of similar, repetitive parts. To give an example, it is inefficient to model each blade of grass as vertices and triangles. A more complex example is found in architecture. Most buildings have evenly distributed windows in the same design. The surface of each single window can explicitly be described with triangles or, for example, as a procedure that evenly distributes a variable number of windows on a wall.

The *Generative Modeling Language* (GML) by Havemann [Hav05] features a description of geometry with procedural models. Intersection-free extrusion is mandatory for many procedural definitions, like an automatic roof construction. This is where the straight skeleton finds its application.

2.8.3 Origami

Origami is the ancient Japanese art of paper folding. In Japanese, *ori* means folding and *kami* means paper. Usually, the folding starts with a square sheet of paper. The result of several folding steps is an easily recognisable sculpture, like the famous Origami crane. Typically, cutting of the paper is not allowed.

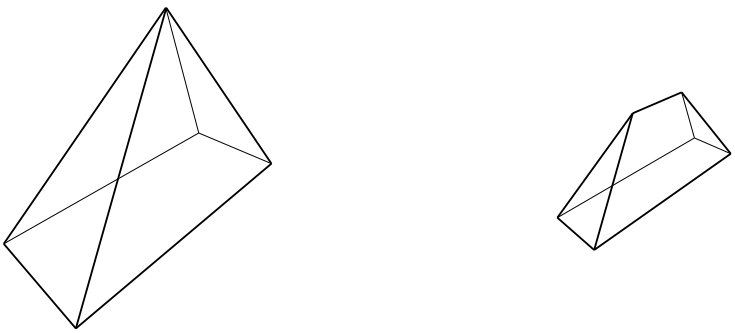
A related art of folding paper is to fold a sheet of paper in a way that a single straight line cut creates a desired polygonal shape when the paper is unfolded. To give an example, a five-pointed star is created by folding a paper four times and cutting a straight line with a scissor.

The work by Demaine et al. answers the question whether any polygonal shape can be created by a single straight line cut when the sheet of paper gets folded many times [DDL98]. Of course, this is more a theoretical point of view because the sheet of paper doubles its thickness at every folding step. To reduce the number of edges to cut, the sheet of paper has to be folded at angle bisectors of the edges of the polygonal shape. When the folding lines are shown inside the polygonal shape, it is obvious that these lines include the straight skeleton. Their work proved that two straight line cuts are sufficient to create any polygonal shape by folding.

3 Straight Skeletons in Space

A polyhedron is a closed and interior-connected subset of the 3-space $\mathbb{R}^3$ with a piecewise linear boundary. It consists of vertices, edges and facets. A vertex is a point that has incident edges and facets. An edge is a straight line segment that connects two vertices. Coplanar edges bound the area of a facet.

The straight skeleton of a polyhedron is defined by an offsetting process, where each facet is shifted inwards in a self-parallel manner. The fundamental problem arises at the very first moment of this process: A point in 3-space is defined by the intersection of three planes. Some vertices of a given polyhedron may have more than three incident facets. When shifting all facets in parallel and at the same time, vertices need to be split into vertices of degree 3. An example is shown in Figure 3.1. In the shown example, the vertex has only convex incident edges. Vertices with convex and reflex incident edges are more challenging. (e.g. the vertex at the center of a star shaped polyhedron.)



(a) Pyramid with a vertex of degree 4 on top

(b) Vertex splits into two vertices of degree 3

Figure 3.1: Polyhedron offsetting

An offsetting process towards the interior of the polyhedron is called *shrinking process*. During the shrinking process, moving vertices trace out the *arcs* of the straight skeleton. By following the edges, skeletal facets are created. The skeletal facets are called *sheets*. A *node* of the straight skeleton is created where an event occurs. Such events cause combinatorial or topological changes of the given polyhedron during the shrinking process. Every facet of the polyhedron gives rise to exactly one cell, which is bounded by skeletal facets. An example is shown in Figure 3.2.

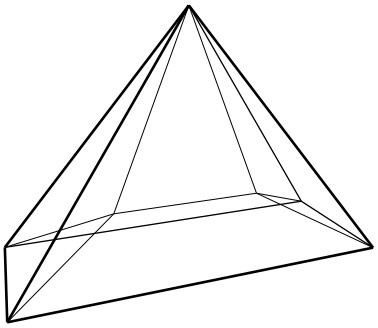


Figure 3.2: Straight skeleton of a pyramid

At the beginning of this chapter, existing works regarding this topic are summarized. Ideas from lower dimensions are used to guide the thoughts into 3-space. A visualization is presented that perfectly fits to the vertex-splitting problem. An algorithm for constructing the straight skeleton in 3-space is presented. Results of generic configurations are shown.

3.1 Prior Work

Letting a polyhedron shrink in a self-parallel manner is an essential task in geometry. However, papers on this topic are rare.

The first paper on straight skeletons of three-dimensional polyhedra was done by Barequet, Eppstein, Goodrich and Vaxman [BEGV08]. At that time, it was the most complete paper on straight skeletons in 3-space. At the beginning of this paper, a voxel-based algorithm to approximate the shrinking process is explained. The next section shows how to handle axis aligned polyhedra. At the end, general polyhedra are discussed. In their paper, a lower bound for the complexity of the straight skeleton in 3-space is proven.

3.1.1 Voxel-based Approach

A voxel is a cubical cell in a regular, axis aligned grid in three-dimensional space. For the three-dimensional space, a voxel is the same as a pixel for the two-dimensional plane. The voxel-based approach considers cases in which the input polyhedron is formed as a union of connected voxels.

To shrink a given polyhedron, voxels of its boundary are analyzed. Each of these voxels has exactly six directly adjacent voxels. The adjacent voxels are used for a case distinction. To give an example, a voxel might have two of its sides as part of the polyhedron's boundary (facets). The resulting straight skeleton for this voxel can easily be obtained by predefined cases. After each voxel on the boundary has been processed, the result is that each facet of the polyhedron has been shifted inwards the length of a voxel. The straight skeleton remains from the previously done case distinctions of directly adjacent voxels.

This algorithm takes $\Theta(V)$ time, where V denotes the volume as number of voxels. This basic algorithm is improved so that the straight skeleton is computed in time proportional to the surface area of the input polyhedron [BEGV08].

3.1.2 Orthogonal Polyhedra

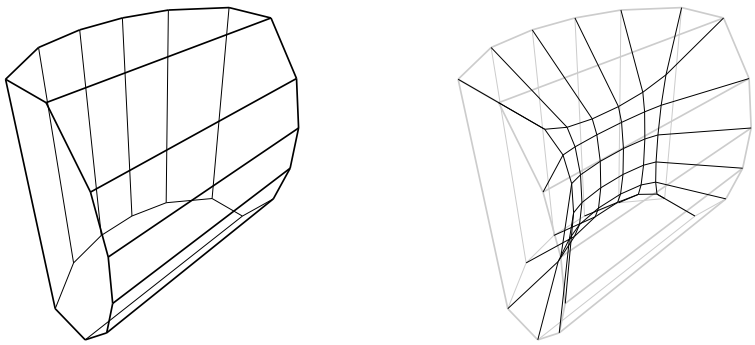
The next step from voxel-based polyhedra to general polyhedra are orthogonal polyhedra. An orthogonal polyhedron is a polyhedron, where all of its facets are aligned to two of the coordinate axes.

The shrinking process continuously moves all facets inwards. Although the facets continuously move inwards, there can be a discontinuous change of the polyhedron's surface when two parallel facets meet. The resulting straight skeleton is equal to the medial axis in the L_∞ metric. The benefit gained from orthogonal polyhedra is that the number of possible events during the shrinking process is limited and can be enumerated. Barequet et al. use an enumeration of possible events to prove the structural complexity of the straight skeleton of orthogonal polyhedra to be $O(n^2)$, where n denotes the number of vertices. Furthermore, the straight skeleton of orthogonal polyhedra can be computed in $O(n^2 \log n)$ time [BEGV08].

An implementation for computing the straight skeleton of orthogonal (axis aligned) polyhedra was done by Jonas Martinez [MVPG11].

3.1.3 Lower Bound for General Polyhedra

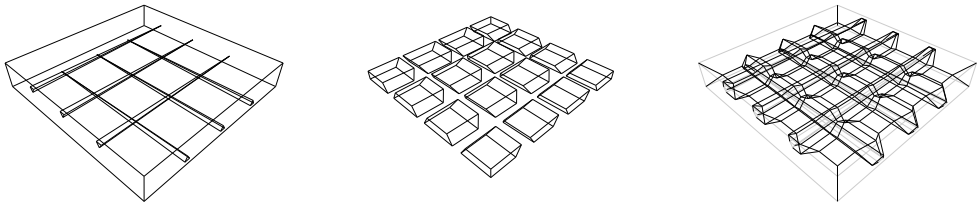
The straight skeleton of a convex polyhedron is equal to its medial axis. In the convex case, both skeletons consist only of piecewise linear parts. In 1994, Held used this property to compute the medial axis of convex polyhedra by using a wavefront propagation of the facets [Hel94]. A lower bound for the structural complexity was shown. An example of his idea is visualized in Figure 3.3. When the polyhedron shrinks, the facets in front get in contact with the facets at the back. This creates a squared number of skeletal facets in between.



(a) Convex polyhedron **(b)** Straight skeleton

Figure 3.3: A convex polyhedron and its skeleton, which is quadratic in the size of the polyhedron.

Figure 3.4 shows a polyhedron where the offsetting process splits a single facet into a squared number of facets. It is a rectangular prism that is small in height and has tetrahedral spikes pointing to the inside. The resulting straight skeleton is quadratic in the size of the polyhedron.



(a) Initial polyhedron **(b)** Offset polyhedron **(c)** Straight skeleton

Figure 3.4: This polyhedron is the *iron maiden pizza box* by Tim Culver.

Barequet et al. constructed a lower bound for the straight skeleton of general polyhedra with a set of triangular prisms [BEGV08]. The prisms are aligned in a way such that the projection to a plane behind creates a set of line segments on that plane. This alignment is illustrated in Figure 3.5.

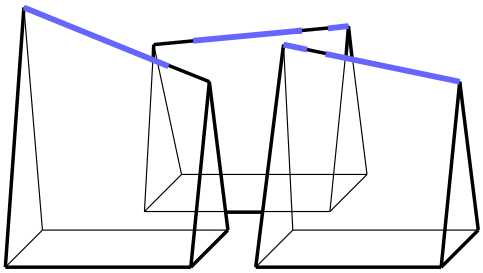


Figure 3.5: A set of triangular prisms. The upper envelope of the projected line segments is highlighted.

In the worst case, the upper envelope of a set of n line segments has a structural complexity of $\Omega(n\alpha(n))$, where $\alpha(n)$ denotes the inverse of the Ackermann function [WS88].

A polyhedron is constructed with a set of prisms at the bottom and a second set of prisms at the top. During the offsetting process, the prisms at the bottom are growing upwards and prisms on top are growing downwards. Both sets of prisms create a squared number of intersections when they crash into each other. Therefore, the lower bound for general polyhedra is $\Omega(n^2\alpha^2(n))$, which is slightly greater than $\Omega(n^2)$ [BEGV08].

3.2 From Plane to Space

Let a vertex with more than three incident edges be given, where all incident edges can be intersected with a single plane. Such a vertex is called a *pointed* vertex. The plane intersects the corresponding polyhedron in a way that the intersection is a bounded, crossing-free polygon. The straight skeleton is defined by shifting each facet in a self parallel manner. This means that edges of this polygon are shifted in a self parallel manner too. Their speed is determined by the inner angle φ_i between the intersecting plane and the corresponding facet. Equation 3.1 shows how to compute the weight w_i of the edge. This weight corresponds to the speed of the edge.

$$w_i = \frac{1}{\sin \varphi_i} \tag{3.1}$$

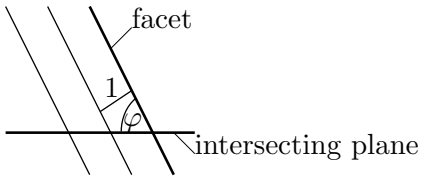
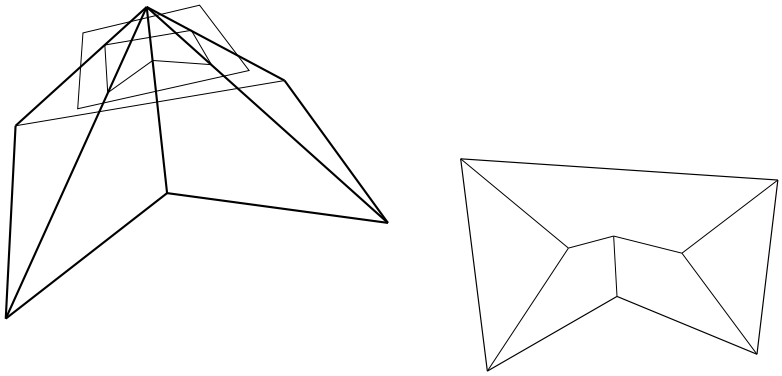


Figure 3.6: Speed on intersecting plane

Computing the weighted straight skeleton in the plane computes a cross-section of a possible straight skeleton in space. The structure of the weighted straight skeleton in the plane shows how the given vertex needs to be split. Each node of the weighted straight skeleton in the plane belongs to a vertex of the shrinking polyhedron in space.



(a) Polyhedron intersected with a plane

(b) Weighted straight skeleton of cross-section

Figure 3.7: From plane to space

Unfortunately, this approach only works for pointed vertices.

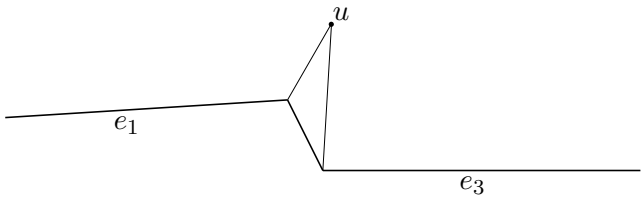


Figure 3.8: Edge e_1 and edge e_3 can not meet at node u.

The weighted straight skeleton in the plane shows an ambiguity issue when edges with different weights become parallel (see Figure 2.9 in Section 2.3). This issue can not occur when a pointed vertex is intersected with a plane that determines the speed of each edge. The proof is done by contradiction. It is assumed that edge e_1 and edge e_3 meet and create a node of the weighted straight skeleton. This assumption is illustrated in Figure 3.8. For this observation, it is required that the node u is vertically below the pointed vertex. The node is contained by the intersection of the polyhedron with the plane. Depending on the location, the plane intersects the polyhedron at a different angle. Because the weighted straight skeleton is a cross-section of the three-dimensional straight skeleton, tilting the intersecting plane does not change the combinatorial structure of the skeleton. The distance between the pointed vertex and the intersecting plane is denoted by h. An edge e_i meets the node u after it has moved a distance of $\frac{h}{\tan\varphi_i}$. With a speed of $\frac{1}{\sin\varphi_i}$, edge e_i reaches the node at time $h\cos\varphi_i$. The geometric configuration from Figure 3.8 does not allow to fulfill the equation $\cos\varphi_1 = \cos\varphi_3$. Therefore, edge e_1 and edge e_3 can not meet during the shrinking process of the polyhedron.

The straight skeleton in the plane is unique and represents exactly one possible bisector graph of the intersected part of the polyhedron. Figure 2.5 in Section 2.1.2 shows that the bisector graph of a polygon in the plane is not unique. It turns out that any crossing-free bisector graph can be used to split a vertex of a polyhedron. An intersecting plane is not needed to compute angle bisector planes. Therefore, angle bisector planes do not require the vertex to be pointed.

3.3 Fundamental Algorithm

As mentioned at the beginning of this chapter, the shrinking process of a non-degenerate polyhedron requires each vertex to be split into vertices of degree 3. The degree of a vertex counts the number of incident edges. It is denoted by $\deg v$. This number is equal to the number of incident facets. There are various combinations to split a vertex.

The beginning of this section explains how all possible combinations are generated. Each combination is checked if it leads to an offset polyhedron without self-intersections of the surface. At the end of this section, the presented algorithm is extended to vertices with coplanar facets.

3.3.1 Generating Unrooted Binary Trees

After each vertex has been split into vertices of degree 3 (non-degenerate case) the offset structure of a single vertex is represented as an unrooted binary tree. A node of this unrooted binary tree represents one vertex of degree 3 after the splitting.

A vertex of degree 4 has exactly two possible unrooted binary trees. A vertex of degree 5 has five possible binary trees. Figure 3.9 is a schematic drawing of a vertex of degree 5. The facets are labeled in counterclockwise order around the vertex seen from outside of the polyhedron. The total number of possible unrooted binary trees is given by the Catalan number C_n. Here, n denotes the degree of the vertex.

$$C_n = \frac{1}{n+1}\binom{2n}{n} = \frac{(2n)!}{(n+1)!n!} \tag{3.2}$$

This number shows exponential complexity based on the degree of the vertex.

To compute all possible unrooted binary trees, a split operation is introduced. The notation $\langle a, b\rangle$ expresses that facet a ($\in \mathbb{N}$) is adjacent to facet b after the split operation. Assuming that edges are undirected, the split operation is commutative, $\langle a, b\rangle = \langle b, a\rangle$. To efficiently compute all unrooted binary trees using split operations, only split operations are used where the left operand is smaller than the right one, $a < b$. Different unrooted binary trees are constructed with a fixed number of split operations. The order of the split operations is immaterial to the result. While creating the lists of split operations, permutations of split operations that lead to equal unrooted binary trees are avoided. This is achieved by an ordering relation between the split operations. A list is extended only if the last split operation is smaller than the one that might be appended. Any ordering relation can be used.

A vertex is split in several steps. The following example shows the procedure on a node with degree 5. Generated split operations of the first step are drawn in Figure 3.10.

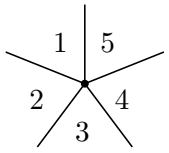
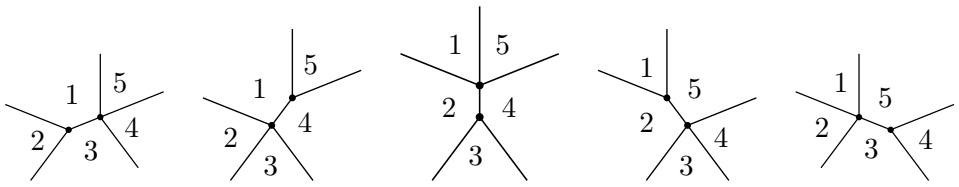


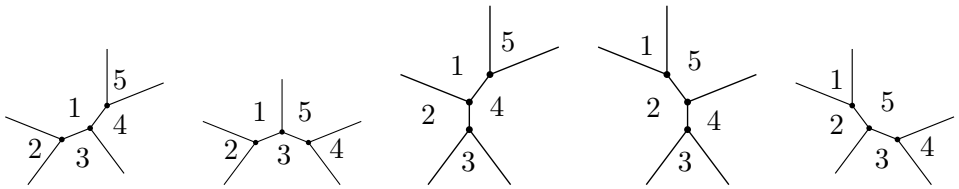
Figure 3.9: Node of degree 5



(a) $\langle 1,3\rangle$ **(b)** $\langle 1,4\rangle$ **(c)** $\langle 2,4\rangle$ **(d)** $\langle 2,5\rangle$ **(e)** $\langle 3,5\rangle$

Figure 3.10: After the first split operation

The next step splits the remaining nodes of degree > 3 separately. Figure 3.11 shows how the lists of split operations are extended.



(a) $\langle 1,3\rangle, \langle 1,4\rangle$ **(b)** $\langle 1,3\rangle, \langle 3,5\rangle$ **(c)** $\langle 1,4\rangle, \langle 2,4\rangle$ **(d)** $\langle 2,4\rangle, \langle 2,5\rangle$ **(e)** $\langle 2,5\rangle, \langle 3,5\rangle$

Figure 3.11: After the second split operation

For $\langle 3,5\rangle$ (Fig. 3.10e) there is no split operation that is bigger than $\langle 3,5\rangle$. The number of split operations has to be the same for each combination of unrooted binary trees. For this reason, the list which includes $\langle 3,5\rangle$ is removed. This step is repeated until all nodes have degree 3.

To split a vertex v into vertices of degree 3, the number of required split operations is $\deg v - 3$.

The generated unrooted binary tree is a schematic representation of the combinatorial structure of the split vertex. This combinatorial structure is equal to the structure of the bisector graph of incident facets.

A node of degree 6 has the following 14 different unrooted binary trees:

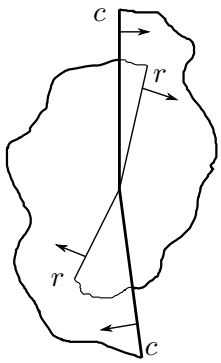
$\langle 0,2\rangle, \langle 0,3\rangle, \langle 0,4\rangle$
$\langle 0,2\rangle, \langle 0,3\rangle, \langle 3,5\rangle$
$\langle 0,2\rangle, \langle 0,4\rangle, \langle 2,4\rangle$
$\langle 0,2\rangle, \langle 2,4\rangle, \langle 2,5\rangle$
$\langle 0,2\rangle, \langle 2,5\rangle, \langle 3,5\rangle$
$\langle 0,3\rangle, \langle 0,4\rangle, \langle 1,3\rangle$
$\langle 0,3\rangle, \langle 1,3\rangle, \langle 3,5\rangle$
$\langle 0,4\rangle, \langle 1,3\rangle, \langle 1,4\rangle$
$\langle 0,4\rangle, \langle 1,4\rangle, \langle 2,4\rangle$
$\langle 1,3\rangle, \langle 1,4\rangle, \langle 1,5\rangle$
$\langle 1,3\rangle, \langle 1,5\rangle, \langle 3,5\rangle$
$\langle 1,4\rangle, \langle 1,5\rangle, \langle 2,4\rangle$
$\langle 1,5\rangle, \langle 2,4\rangle, \langle 2,5\rangle$
$\langle 1,5\rangle, \langle 2,5\rangle, \langle 3,5\rangle$

3.3.2 Checking for Valid Offset Polyhedra

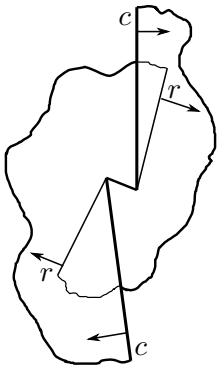
After all unrooted binary trees are generated, each combination is checked if it leads to a valid offset polyhedron when shifting all facets in a self-parallel manner. An offset polyhedron is valid if its surface is free of self-intersections.

The vertex is split using a previously generated unrooted binary tree. Afterwards, each facet is shifted inwards with the same distance in a self-parallel manner. To check if the offset polyhedron has a surface without self-intersections, each polygon on the surface (facet) needs to be free of intersections.

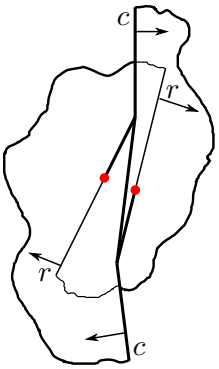
A simple sweep line algorithm for segment intersection is used to determine if a polygon has self-intersections. This algorithm takes $O(n \log n)$ time, where n denotes the number of segments. A facet without self-intersections is a necessary condition, but it is not sufficient. An example for this is shown in Figure 3.12. Each facet is free of self-intersections, but the polyhedron has self-intersections. Some edges cross non-adjacent facets. This can happen when the vertex is split. To detect such cases, a point in polygon algorithm is used.



(a) Two tilted, overlapping facets



(b) Valid solution



(c) Invalid solution

Figure 3.12: Self-intersecting polyhedron with intersection-free facets. In this figure, c denotes a convex edge and r denotes a reflex one. The arrows show the moving directions of the edges.

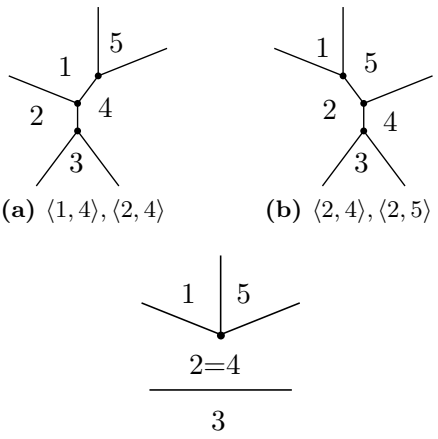
There are various ways to check if a point is inside a polygon. A possible algorithm uses ray casting. The point emanates a ray in any direction in the plane. The number of edges of the polygon that intersect this ray is counted. If this number is even, the point is outside the polygon. If it is odd, the point is inside the polygon. This algorithm is computed in linear time $O(n)$, where n denotes the number of edges of the polygon.

The part of the polyhedron originating from the split vertex v has $\deg v$ facets and $2 \cdot \deg v - 3$ edges. Therefore, the expected value of edges of the relevant part of each facet is bounded by $\frac{2(2 \cdot \deg v - 3)}{\deg v} \leq 4$. $O(\deg v)$ facets are checked against $O(\deg v)$ edges. The resulting time complexity for this step is $O(\deg^2 v)$.

3.3.3 Vertices with Coplanar Facets

The following is a naive idea of how vertices with coplanar facets are split. Each facet is tilted slightly. By tilting the facets, coplanarities are destroyed and angle bisector planes between all facets can be computed. During the shrinking process, edges of the polyhedron move on these angle bisector planes.

Unfortunately, the reality looks different. Between two coplanar facets there is no line that defines an intersection. There is no edge that moves on any angle bisector plane during the shrinking process, because there is no angle bisector plane. For that reason, the edge of the bisector graph is removed if its coplanar facets are adjacent. An example is illustrated in Figure 3.13.



(a) $\langle 1,4\rangle, \langle 2,4\rangle$ **(b)** $\langle 2,4\rangle, \langle 2,5\rangle$

(c) Resulting bisector graph

Figure 3.13: Both unrooted binary trees lead to the same bisector graph if facet 2 and facet 4 are coplanar

The removal of an edge with coplanar facets decomposes the unrooted binary tree into a forest consisting of two trees. The example in Figure 3.13 also shows that different unrooted binary trees can lead to identical bisector graphs.

3.4 Spherical Skeleton

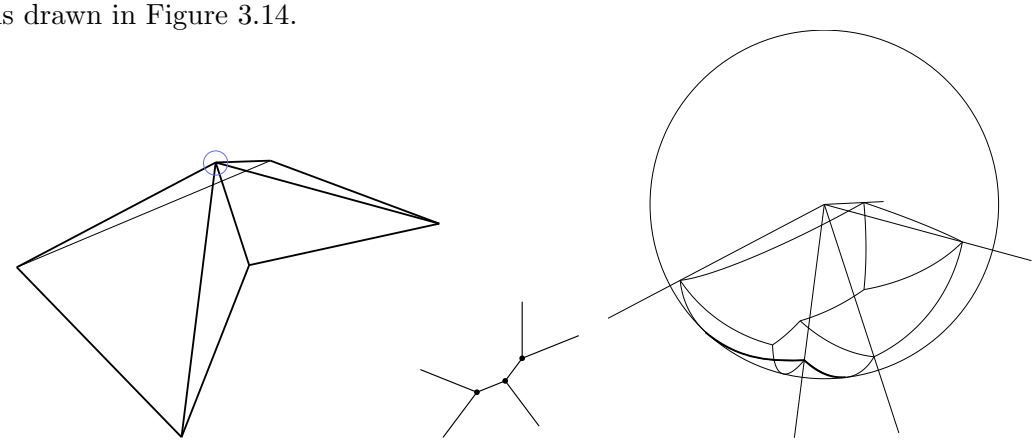
The spherical skeleton is an embedding of the bisector graph of a vertex onto the surface of a sphere centered at that vertex. This embedding represents the intersection of the straight skeleton of the polyhedron with the surface of an infinitesimal sphere.

On a sheet of paper, a sphere is drawn as a circle. Images on the surface of the sphere are shown bent on the paper. After defining which parts of the images are on the front and which parts are on the back of the sphere, the images on the surface of the sphere are uniquely defined by a single drawing.

This method completely visualizes the splitting of a complex vertex with only one figure. To achieve this, a sphere is centered at the vertex. The surface of the sphere is intersected with incident facets of the vertex. This intersection creates a spherical polygon. Circular edges on the front of the sphere are drawn bolder than circular edges on the back.

The bisector graph is embedded onto the surface of the sphere, inside the spherical polygon. This embedding of the bisector graph forms the spherical skeleton. The edges of the bisector graph show adjacent facets. Between the adjacent facets, there are angle bisector planes. The edges of the polyhedron move on these bisector planes during the shrinking process. The circular arcs of the spherical skeleton result from the intersections of the angle bisector planes with the surface of the sphere. The nodes of the bisector graph are embedded on the intersections of circular arcs. The result is an intersection of the straight skeleton of the polyhedron with the surface of the sphere.

Such a figure shows how the vertex at the center of the sphere is split. An example is drawn in Figure 3.14.



(a) Pyramid with five-sided base and intersecting sphere on top

(b) Structure of bisector graph

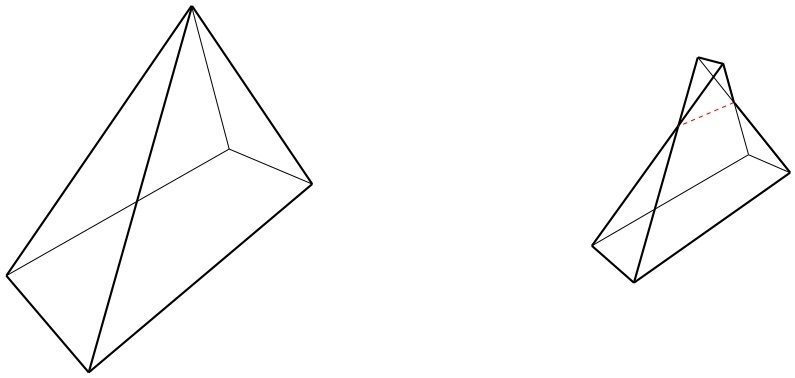
(c) Sphere with intersected facets and embedded bisector graph

Figure 3.14: Example of a spherical skeleton

If the spherical skeleton is free of crossings and completely inside the spherical polygon, the local offset surface of the polyhedron's vertex is free of intersections.

3.5 Existence of a Solution

The idea begins with the assumption that an arbitrary unrooted binary tree with appropriate size is used to split a vertex. While shifting the facets inwards, there are, depending on the used unrooted binary tree, parts of the polyhedron that shrink and parts that increase in size. To make it possible that parts of the polyhedron increase their size while the facets are shifted inwards, the inside of the polyhedron's surface has to be turned to the outside. Exactly where this happens, the surface of the polyhedron is self-intersecting. The self-intersections separate the growing parts from the shrinking parts. Because the growing parts can be separated, they can be removed from the polyhedron. When all growing parts are removed, the solution consists of shrinking parts only and has a surface without self-intersections.



(a) Pyramid with a vertex of degree 4 on top

(b) Vertex split with an arbitrary unrooted binary tree

Figure 3.15: Example of an invalid offset

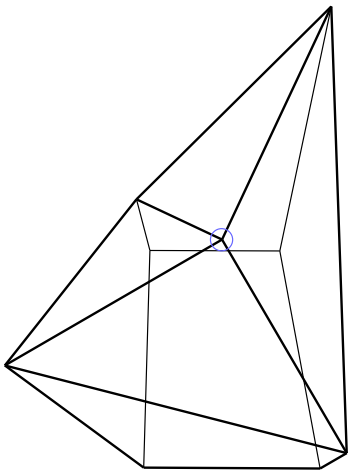
3.6 Ambiguity

The ambiguity of the straight skeleton in 3-space is shown by providing examples of polyhedra that have more than one combinatorially different offset surface.

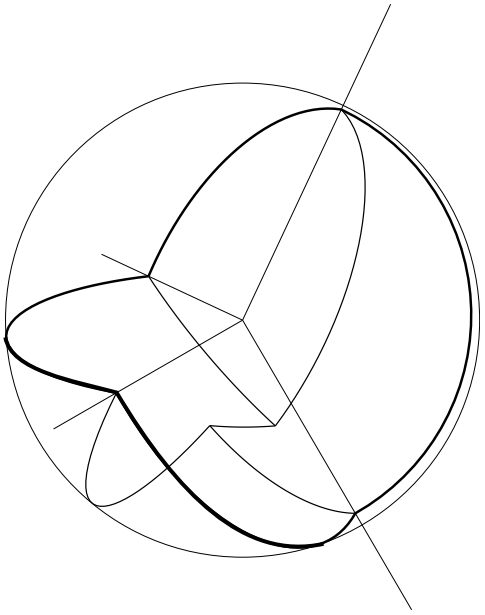
A polygon in the plane may have several embedded bisector graphs without crossings (see Section 2 for details). Such a polygon is used as the base of a pyramid. The vertex on top of the pyramid is going to be split. When checking all combinations of bisector graphs of the vertex, there are several combinations that do not lead to a self-intersecting surface of the polyhedron when the facets are shifted inwards.

Several valid combinations may exist only if a vertex has reflex and convex incident edges. If a vertex has only convex incident edges, there is exactly one combination to split this vertex. If the polyhedron is convex, the straight skeleton defined by the shrinking process is equal to the medial axis. The medial axis is uniquely defined.

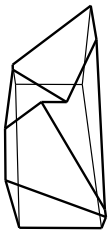
The most simple example to prove ambiguity is a saddle point with two convex and two reflex incident edges. There are exactly two combinations to split this vertex. Both combinations lead to a polyhedron that has an intersection-free surface during the shrinking process. One combination traces out a convex edge. The other one traces out a reflex edge. This example is shown in Figure 3.16.



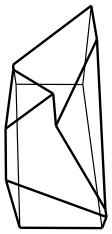
(a) Polyhedron with a saddle point and a parallelogram as its base



(b) Spherical polygon of the saddle point (convex solution is drawn)



(c) Convex solution



(d) Reflex solution

Figure 3.16: Two different, valid offsets from the same polyhedron

3.7 Lower Bound for the Number of Valid Offset Polyhedra

Let there be a pyramid, where the vertex on top has k locally possible offset surfaces. The base of this pyramid is a polygon (like the one shown Figure 2.5). The polygon is used to construct another pyramid with a vertex of high degree on top. A series of n of such polygons is placed on the plane. These polygons are connected with a narrow corridor between each other. The connected polygons form the base of another pyramid. The vertex on top of this pyramid is split at the very first moment of the shrinking process. This construction leads to k^n possible offset surfaces. In the worst case, the number of locally possible offset surfaces originating from one vertex is exponential in the degree of the vertex.

Overall, the lower bound for the number of possible offset polyhedra originating from the same polyhedron is exponential in the number of its vertices.

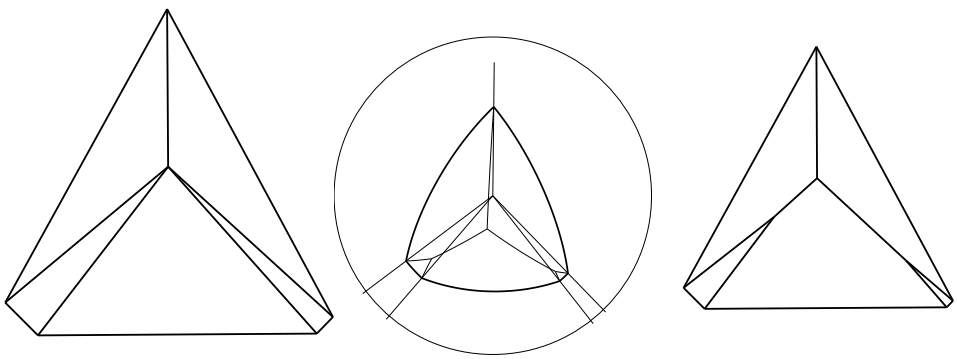
3.8 Convex Vertices

Although the degree of a vertex is usually not that big in general, checking all combinations is not efficient. For vertices with convex incident edges only, there is a unique solution that can easily be computed with a faster algorithm.

To achieve this, three adjacent facets incident to the vertex are shifted inwards in a self parallel manner using the same distance. After these three facets have been shifted, they intersect at a point. This point defines a distance between the origin, the position of the vertex, and itself. The algorithm searches for three adjacent facets incident to the vertex that maximize this distance. This point of intersection is the first identified vertex of the solution.

To continue this procedure, the facet in between the three adjacent facets is removed from the list of incident facets of the vertex. The next point of intersection that maximizes the distance after the facets have been shifted is identified. It is used to further split the vertex.

This step is repeated until all vertices have degree 3. An example is shown in Figure 3.17.



(a) Pyramid with a vertex of degree 5 on top

(b) Vertex intersected with a sphere

(c) Split vertex after an offset

Figure 3.17: Splitting convex vertices

The presented algorithm has quadratic time complexity, $O(\deg^2 v)$, based on the degree of the vertex. A priority queue can be used to speed up the computation to a linear-logarithmic time complexity, $O(\deg v \cdot \log(\deg v))$.

This algorithm can be adopted for vertices with only reflex incident edges. Instead of searching for the point that maximizes the distance after three adjacent facets have been shifted, the point that minimizes the distance is used for reflex vertices.

3.9 Events

After all vertices have been split into vertices of degree 3, every facet is shifted inwards at the same time. This process continues until an event occurs. Such an event can be the vanishing of an edge or when a reflex vertex creates a tunnel or other changes. The vanishing of an edge changes the combinatorial structure of the polyhedron. The creation of a tunnel changes the polyhedron topologically.

An event happens exactly when four supporting planes of facets meet at a point. This is the reason why the number of events is finite. When the shrinking process is stopped exactly at the moment where four facet planes meet, a vertex exists with four incident facets. The splitting of this vertex also explains how the polyhedron is changed by the event.

In this section, events and all possible solutions are shown. The events are divided into two categories, vanish events and contact events. Vanish events are comparable to edge events of the straight skeleton in the plane. Contact events are comparable to split events.

For every event there are several geometric configurations. The events are distinguished by the structure of the bisector graph.

3.9.1 Vanish Events

Vanish events describe the vanishing of edges. Thereby, the length of an edge is continuously decreased by the shrinking process until both vertices of the edge meet at a point. Exactly at this point, the structure of the polyhedron has to change in order to retain a valid surface during the shrinking process. A surface is valid if it is free of self-intersections.

During the shrinking process, all edges of the polyhedron move on angle bisector planes. The exact position where an edge vanishes is computed by intersecting three bisector planes. One bisector plane is given by the edge where it moves on. The other two bisector planes are found at adjacent edges. Figure 3.18 visualizes this. The bisector plane where e moves on is intersected with the bisector planes of edges e_{11} and e_{12}.

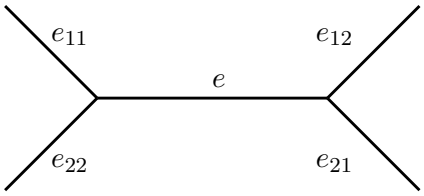


Figure 3.18: Calculate vanish events

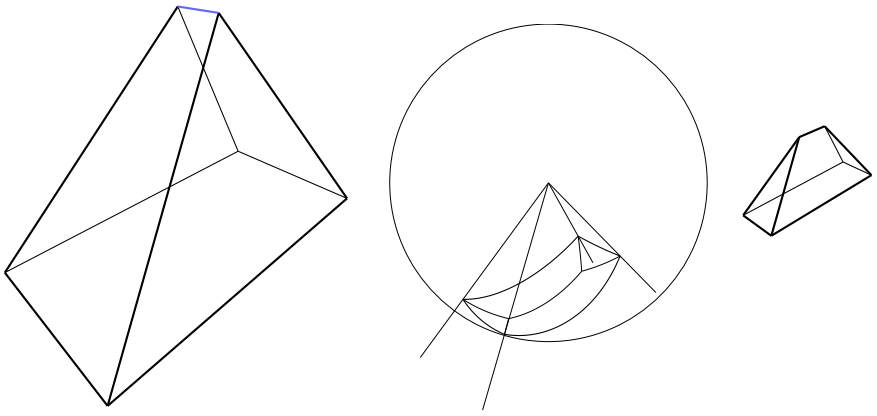
A non-degenerate polyhedron can have one to six edges vanishing simultaneously, even when the facet planes are tilted slightly. The last event is the vanishing of a tetrahedron with its six edges.

Convex polyhedra can have one, three, or six edges vanishing at the same time, for non-degenerate cases.

1 Edge vanishes

When both vertices of an edge meet at a point, a vertex of degree 4 exists at this moment.

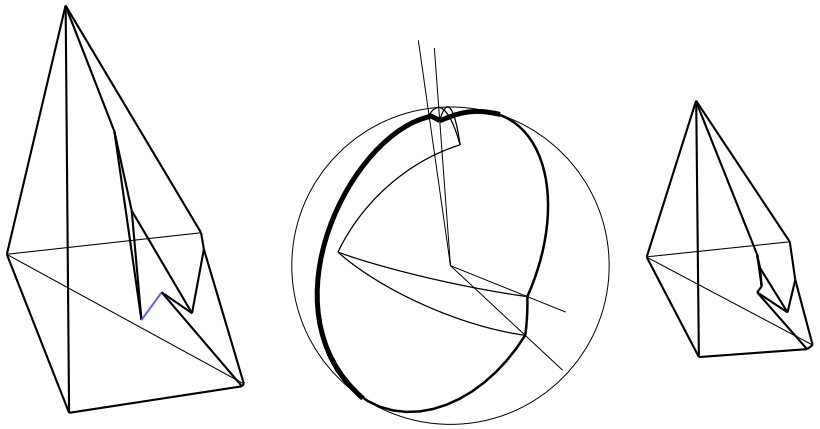
In the simplest case, this event generates a new edge that is defined by the intersection of the facets that were not adjacent before the event. In this case, the the moving directions of both involved vertices are forced to change. An example with such a geometric configuration is shown in Figure 3.19.



(a) Before **(b)** Event **(c)** After

Figure 3.19: Edge event

When two vertices of an edge meet, they do not necessarily change their moving direction. There are geometric configurations where this event does not change the moving direction of the vertex. The edge continuously decreases the length until the edge is vanished. Adjacencies between facets are not changed by the event. After the event the length of the edge increases again. An example for such a geometric configuration is shown in Figure 3.20. In the shown example, it is impossible for the vertices to change their moving directions. The structure of the bisector graph is equal to the one shown in Figure 3.19.



(a) Before **(b)** Event **(c)** After

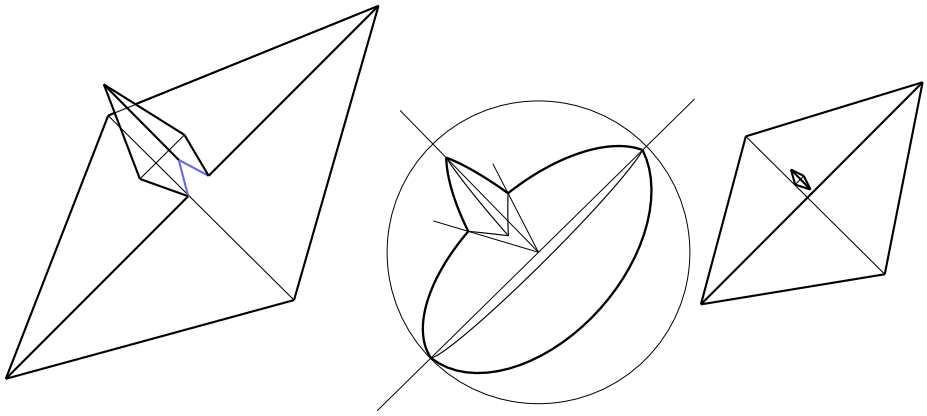
Figure 3.20: Edge event (impossible to flip)

It may happen that none of these combinations lead to self-intersections of the polyhedron's surface. This is the case with saddle points. The example with the saddle point in Figure 3.16 shows that geometric configurations for this event also exist, where both combinations lead to valid offset polyhedra.

2 Edges vanish

Figure 3.21 shows a tetrahedron with a wedge attached. During the shrinking process, the wedge moves away from the edge of the tetrahedron.

This event has a bisector graph that is not connected. This is because the two coplanar facets of the vertex to be split are one and the same facet.

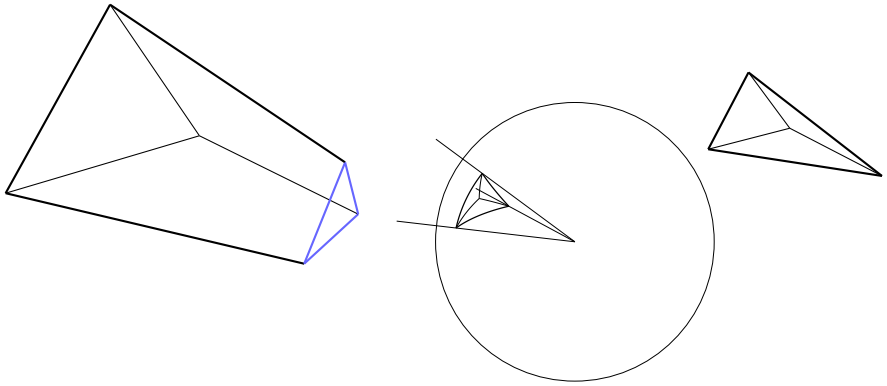


(a) Before **(b)** Event **(c)** After

Figure 3.21: Edge merge event

3 Edges vanish

How a triangle vanishes in a point is shown in Figure 3.22.



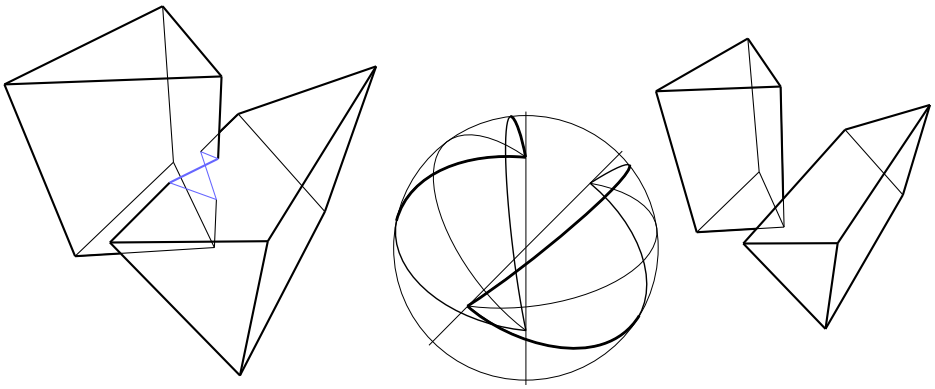
(a) Before **(b)** Event **(c)** After

Figure 3.22: Triangle event

The shown example has three convex edges. There are geometric configurations where three edges vanish simultaneously and some of these edges (or all) are reflex.

4 Edges vanish

Figure 3.23 shows two intersecting prisms. The intersection has four edges. When both prisms are separated by the shrinking process, four edges vanish at the same time. This event has a disconnected spherical polygon. Therefore, the spherical skeleton is disconnected too.

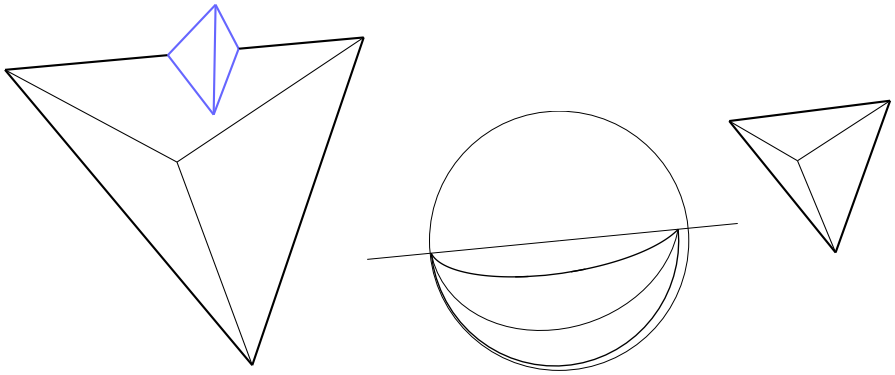


(a) Before **(b)** Event **(c)** After

Figure 3.23: Double edge merge event

5 Edges vanish

Figure 3.24 shows two tetrahedra that share a facet. The smaller tetrahedron vanishes due to the shrinking process. This is the reason why five edges vanish at the same time and at the same position.

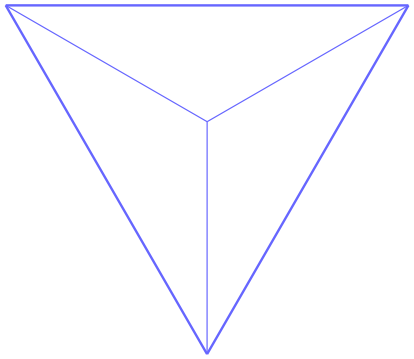


(a) Before **(b)** Event **(c)** After

Figure 3.24: Double triangle event

6 Edges vanish

The last event of a shrinking process is the vanishing of a tetrahedron. The angle bisector planes of a tetrahedron intersect at the center of its insphere. This is the position where all six edges vanish simultaneously.



(a) Before

(vanished completely)

(b) After

Figure 3.25: Tetrahedron event

This event may also occur when the tetrahedron is attached to some other object.

3.9.2 Contact Events

A contact event happens when elements of a polyhedron, which are not adjacent before the event, touch each other. However, most of the contact events happen in a local structural neighbourhood. For example, this can be a vertex and an edge incident to the same facet. There are only two contact events that have a non-local influence. These events are called an *edge-edge contact event* and a *vertex-facet contact event*. The bisector graph of these two events has cycles.

The position of contact events is, comparable with vanish events, determined by the intersection of three bisector planes. Another way of calculating the position is the following. During the shrinking process, two edges that are likely to meet move on bisector planes. The intersection of these two bisector planes is a line. By determining the start point and the velocity of both edges on that line, the point of contact is determined. This is visualized in Figure 3.26.

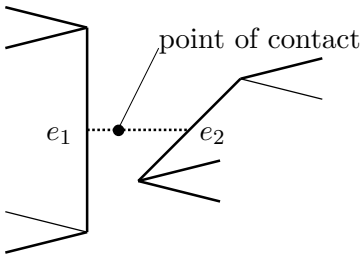
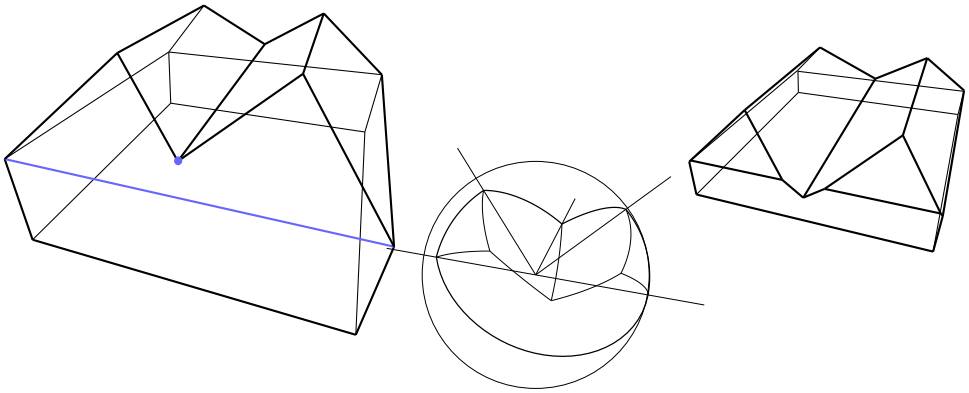


Figure 3.26: Calculate contact events

Care has to be taken on the movement of the vertices that are incident to the involved edges. It is not sufficient when just the supporting lines of the edges get in contact.

Vertex-Edge Contact

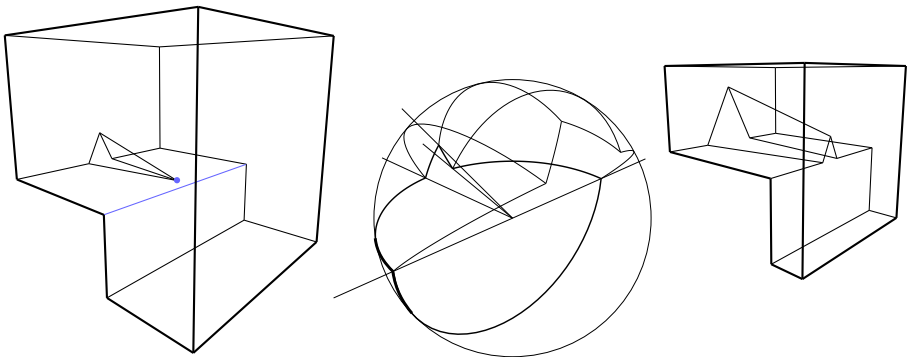
This event happens when a vertex crashes into an edge. The vertex and the edge have to share a common facet. Figure 3.27 shows a polyhedron with a convex edge that is split by a vertex incident to a reflex edge.



(a) Before **(b)** Event **(c)** After

Figure 3.27: Surface event

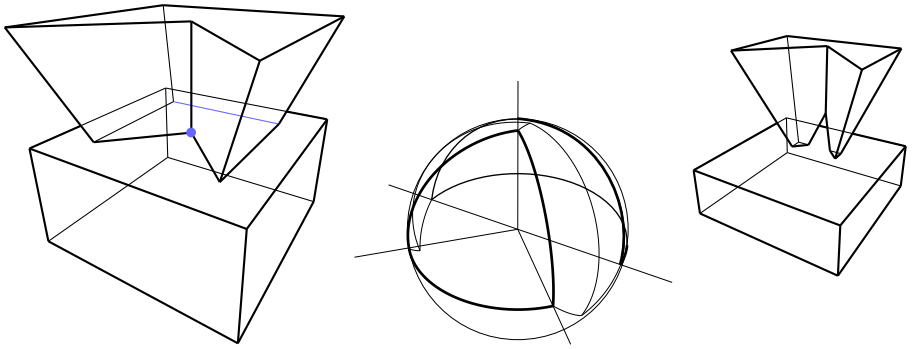
The edge that is split by the vertex does not necessarily have to be convex. Figure 3.28 shows a geometric configuration where a reflex edge is split. The structure of the bisector graph is equal to the one from Figure 3.27.



(a) Before **(b)** Event **(c)** After

Figure 3.28: Surface event with a reflex edge

Figure 3.29 shows that this type of event can also change the polyhedron topologically. The structure of the bisector graph stays the same.



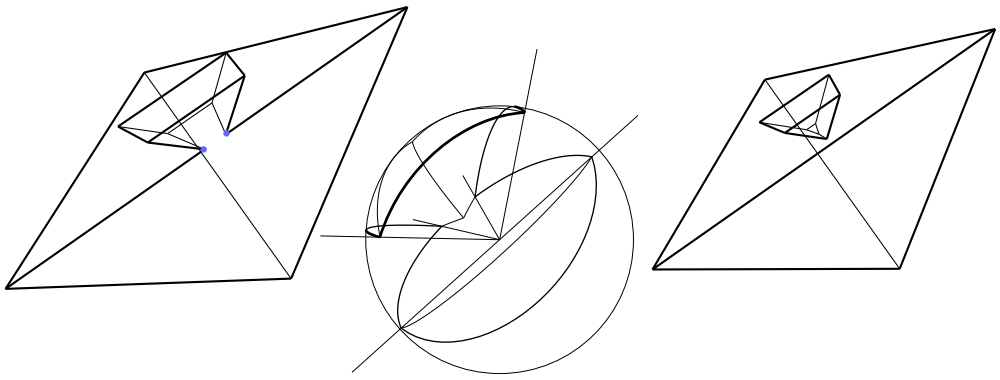
(a) Before **(b)** Event **(c)** After

Figure 3.29: Surface event which causes topological changes

A vertex-edge contact event may also occur when the incident edge of the vertex is convex. This convex edge crashes into another edge when it propagates faster.

Vertex-Vertex Contact I

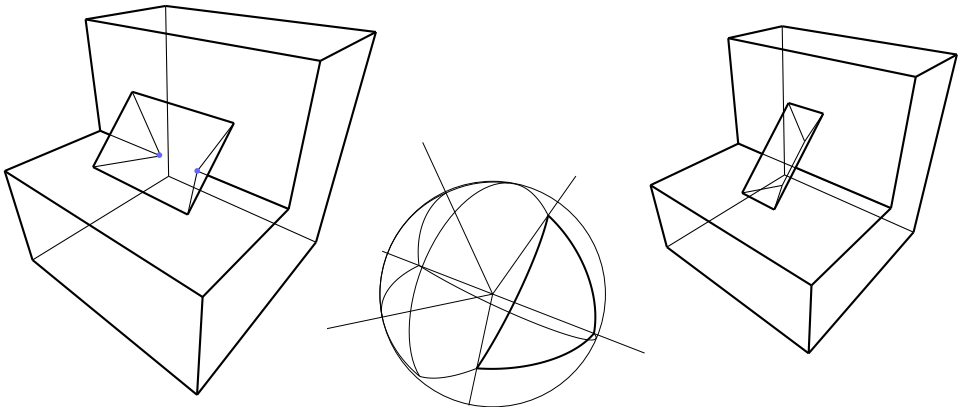
Polyhedra do exist where two vertices meet, even when the facet planes are tilted slightly. Type I of the vertex-vertex contact event changes the moving directions of the vertices. Figure 3.30 shows a geometric configuration where only combinatorial changes on the surface happen.



(a) Before **(b)** Event **(c)** After

Figure 3.30: Vertex event

Figure 3.31 shows how this event can lead to topological changes. The structure of the bisector graph stays the same.

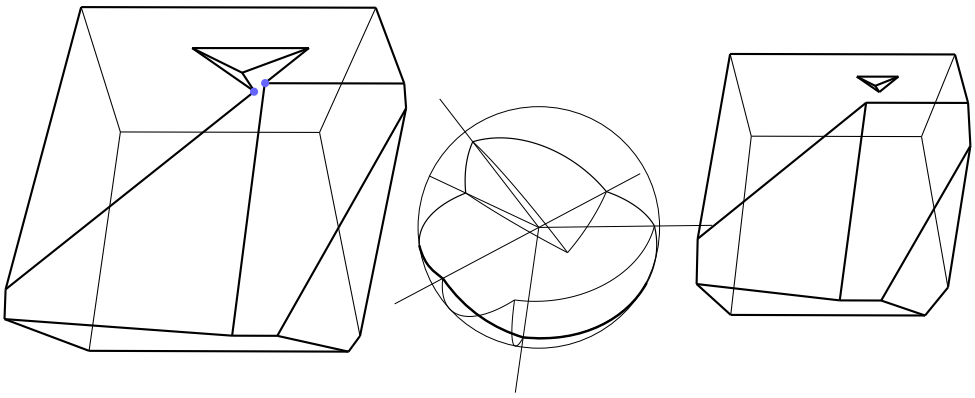


(a) Before **(b)** Event **(c)** After

Figure 3.31: Vertex event which causes topological changes

Vertex-Vertex Contact II

Not every vertex-vertex contact event changes the moving directions of the vertices. Figure 3.32 shows a tetrahedron attached to a polyhedron. They share a common facet. During the shrinking process, the moving directions of both involved vertices do not change. The structure of the bisector graph differs from type I. Therefore, this event is called *vertex-vertex contact event II*.

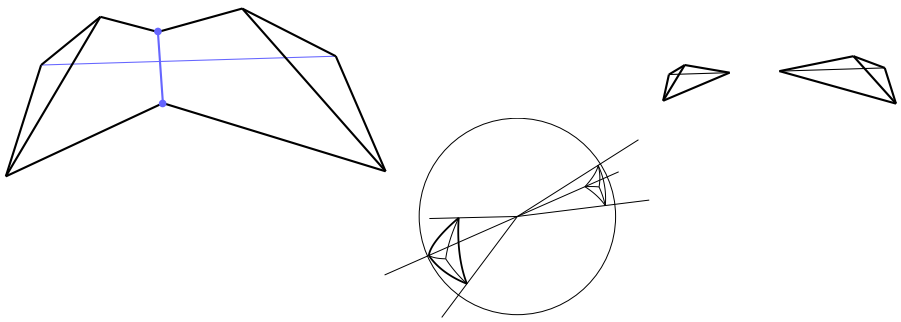


(a) Before **(b)** Event **(c)** After

Figure 3.32: Flip vertex event

Vertex-Vertex-Edge Contact I

This event describes how a reflex edge splits a polyhedron. The reflex edge crashes into another edge. Exactly where this happens, both vertices incident to the reflex edge meet. The reflex edge vanishes at this point. An example is shown in Figure 3.33. The shown polyhedron splits into two parts. There are polyhedra that stay connected after this event.

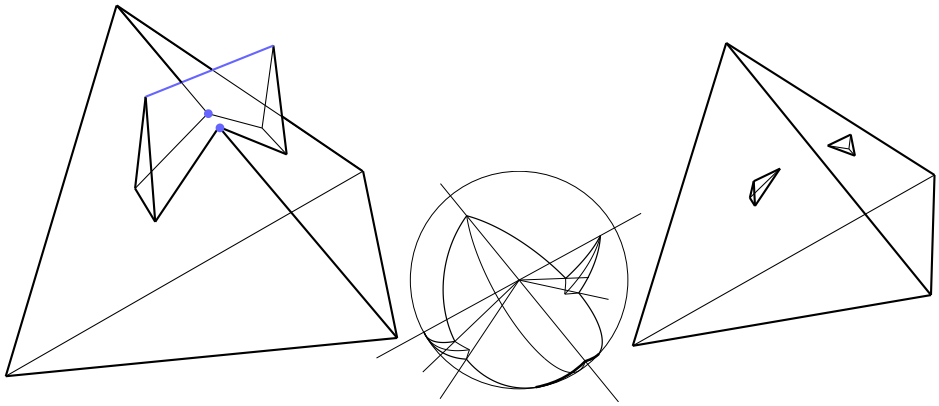


(a) Before **(b)** Event **(c)** After

Figure 3.33: Polyhedron split event

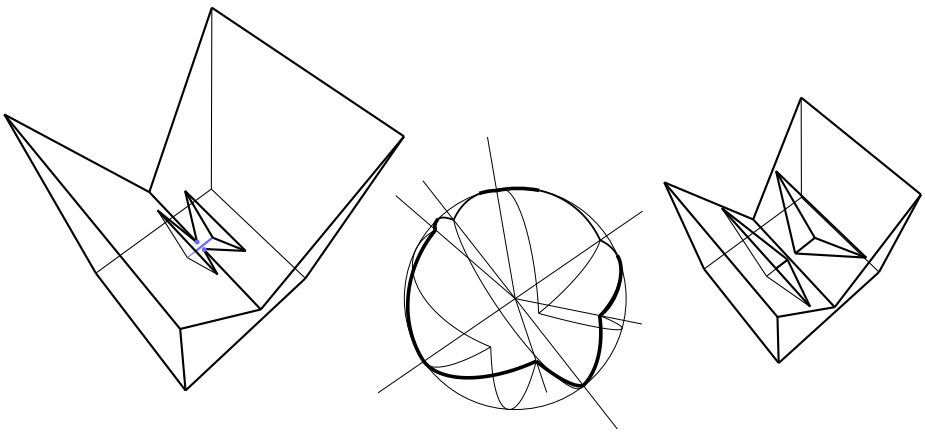
Vertex-Vertex-Edge Contact II

An edge is split where two vertices meet. This also happens when the facets are tilted slightly. An example is shown in Figure 3.34. The bisector graph consists of three parts that are not connected. Exactly at the time and at the position where this event happens, a vertex with degree 8 exists. This degree denotes the highest occurring degree of a vertex of all events.



(a) Before **(b)** Event **(c)** After

Figure 3.34: Split merge event

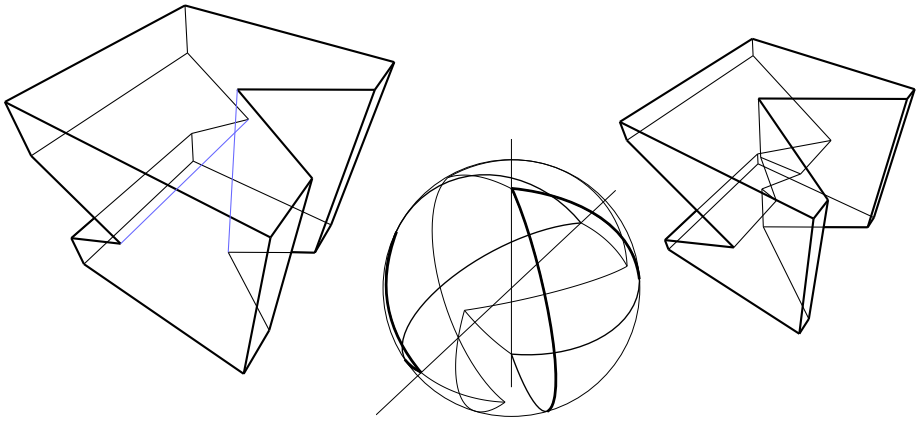


(a) Before **(b)** Event **(c)** After

Figure 3.35: Split merge event in a reflex case

Edge-Edge Contact

The edge-edge contact event in Figure 3.36 shows how two reflex edges meet. Both edges do not share a common structural neighbourhood of the polyhedron. Therefore, this event is non-local.

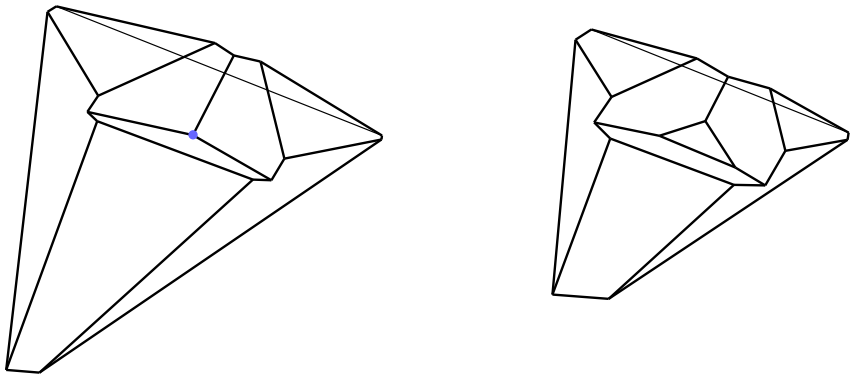


(a) Before **(b)** Event **(c)** After

Figure 3.36: Edge split event

Vertex-Facet Contact

In Figure 3.37 a reflex vertex pierces through a facet. This creates a tunnel. The vertex-facet contact event is also non-local.



(a) Before **(b)** After

Figure 3.37: Pierce event

3.9.3 Notes and Comments

Every event creates one node of the straight skeleton. In the non-degenerate case, every node has four incident arcs and six incident sheets. During the offsetting process, the vertices move on trisector lines. Therefore, every arc has three incident sheets.

Inverse Events

An inverse event is found when the offsetting process is reversed. The facets are shifted in the opposite direction. All inverse events are listed in Table 3.1. E1 denotes a vanish event with one edge. V-E denotes a vertex-edge contact event.

Event	**Inverse**
E1	E1
E2	V-E
E3	V-F
E4	E-E
E5	–
E6	–
V-E	E2
V-V I	V-V-E I
V-V II	V-V II
V-V-E I	V-V I
V-V-E II	V-V-E II
E-E	E4
V-F	E3

Table 3.1: Inverse events

3.10 Monotonicity

The unweighted straight skeleton partitions a given polyhedron into monotone cells. A cell is monotone if the intersection with any line perpendicular to the defining facet is a single line segment. The proof of this property is similar to the proof of the monotonicity of cells of the straight skeleton in the plane (see Section 2.1.4, and [AAAG95]).

In the plane, time is interpreted as a third spatial dimension. This lifts the offset polygons onto a roof model. A line on a facet of the roof model maximizes its slope when it is perpendicular to the defining edge. Because the straight skeleton is defined by a process that continuously moves all edges, the roof model itself is continuous. If a facet of the roof model would not be monotone in direction of its defining edge, there would a point of discontinuity when the line re-enters this facet. Every other facet has a lower slope in direction of that line. Because the roof model is continuous, the intersection with any line perpendicular to a defining edge is connected.

In 3-space, every cell of an unweighted straight skeleton of any polyhedron is monotone. Like the proof for the two-dimensional case [AAAG95], this property is proven by contradiction. A line ℓ perpendicular to a defining facet is considered. During the shrinking process, every point on that line is hit by the offset polyhedron at a specific time. This time has a linear relation to the offset of such a point in the direction of the line.

For this proof, it is assumed that a cell is not monotone. This means that a line ℓ leaves the corresponding cell and re-enters the same cell at some other point. The linear relation between the time of the shrinking process and the offset in the direction of the line changes at positions outside the corresponding cell. A point of discontinuity in this relation (jump in time) is introduced when the line re-enters its corresponding cell (see Figure 3.38). This contradicts to the definition of the straight skeleton. It is defined by an offsetting process that continuously shifts all facets. Because of this contradiction, the assumption of cells not being monotone is incorrect.

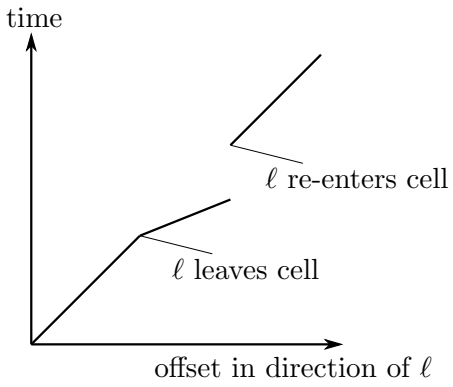


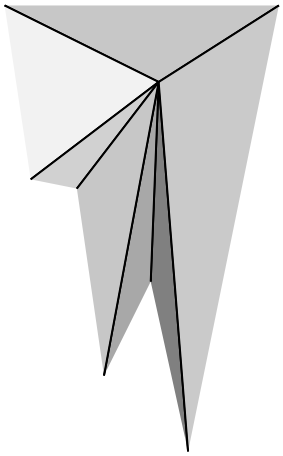
Figure 3.38: Contradiction of continuity and non-monotonicity of cells

3.11 Arrangement of Planes

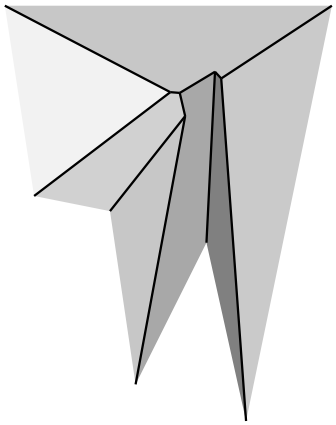
All incident facets of a vertex of high degree intersect at a single point. The supporting planes of these facets form an arrangement of planes in 3-space. The planes divide the space into cells of the arrangement. In the initial position, all cells are unbounded. The offsetting process shifts the planes while keeping them parallel to their initial orientation. When the planes are shifted, a vertex of high degree dissolves into bounded cells of this arrangement.

A selection of bounded cells shows how the vertex of a polyhedron is split by the offsetting process. This selection includes all unbounded cells that are inside the polyhedron (in a local environment). Furthermore, the selection includes all bounded cells that are completely surrounded by previously selected cells. The planes are not allowed to revert their moving direction during the offsetting process. This fact leads to another selection criterion: All planes are required to move away from the initial position of the vertex. For this reason, the surface of the selected cells has to be radially monotone, seen from the initial position of the vertex.

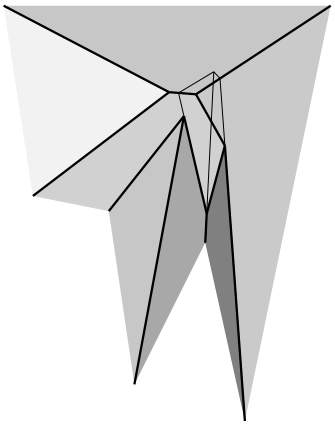
An example of this idea with various selections of cells is illustrated in Figure 3.39. The provided example shows how a pointed vertex with degree 7 dissolves into bounded cells. Figure 3.39b and Figure 3.39c show the cases where the structure of the solution is an unrooted binary tree. It is not possible for the offsetting process to create a local offset as shown in Figure 3.39e.



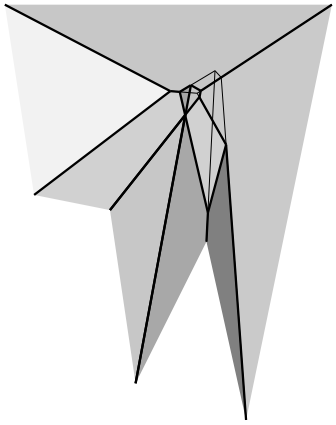
(a) The vertex on top of this pyramid dissolves into bounded cells of an arrangement of planes.



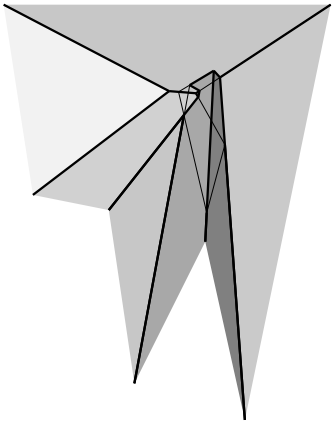
(b) Unbounded cells on the inside



(c) Unbounded cells on the inside and two bounded cells



(d) Only one bounded cell is added. This would be another solution but the structure is no longer a tree.



(e) Another cell is added. This local offset is not valid. It introduces a facet where further offsetting would create an additional supporting plane.

Figure 3.39: Arrangement of planes

3.12 Optimization

The offset of a polyhedron is ambiguous (see Section 3.6). At the beginning of the shrinking process, vertices with reflex and convex incident edges may lead to several valid and combinatorially different offset polyhedra. During the shrinking process, there is only one event that can cause ambiguity: The edge vanish event with one edge only may lead to two different and valid offset polyhedra. This section explains how these ambiguities are used to optimize the solution.

3.12.1 Number of Convex Edges

A possible optimization criterion is to maximize the number of convex edges of the offset polyhedron. At the same time, the number of reflex edges is minimized. When the vertex is split at the beginning of the shrinking process, the combination that leads to the highest number of convex edges is chosen.

If a vanish event with one edge only has two valid solutions, the solution with the convex edge is chosen.

Of course, this idea can also be used to maximize the number of reflex edges at each step. Choosing the local optimum at each step of the computation does not necessarily lead to the global optimum. A chosen reflex edge can split a high number of convex edges when the offsetting process continues. Overall, this might increase the number of convex edges at a specific offset.

3.12.2 Volume / Surface Area

Another optimization criterion is the minimization of the volume. Minimizing the volume of the offset polyhedron also maximizes its surface area. At the beginning of the shrinking process, vertices of degree > 3 are split. After vertices have been split, facets move away from their initial position. Seen from the initial position of the vertex that has just been split, previous incident facets are entirely visible (because of radial monotonicity). Therefore, non-overlapping pyramids can be constructed. The base of each pyramid is defined by a shifted facet. The top is defined by the initial position of the vertex. Equation 3.3 shows how the volume V and the base area B of a pyramid are connected.

$$V = \frac{1}{3} \cdot B \cdot h \tag{3.3}$$

If each facet is shifted with the same speed, the height h is equal for all pyramids. This observation leads to a direct connection between the volume and the base area. The bigger the volume is, the bigger is the base area of the pyramid. This volume is removed from the offset polyhedron by the shrinking process. Therefore, maximizing the surface area of the offset polyhedron minimizes its volume.

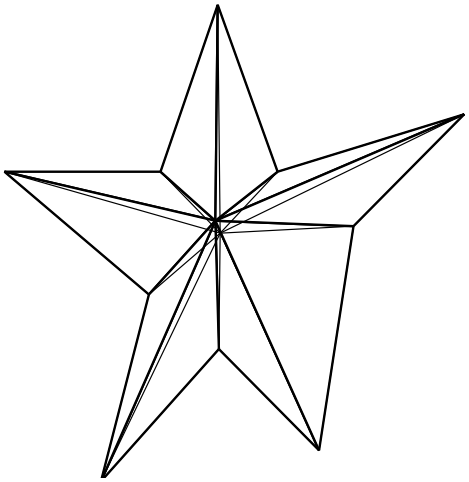
The surface area of the polyhedron is calculated by adding the areas of all facets. Each facet is rotated so that the normal vector points in the same direction as the z-axis. After the rotation, the facet is interpreted as a polygon in the plane (without z-coordinate). The (signed) area A of a planar polygon with n vertices in the plane $(x_i, y_i), ..., (x_n, y_n)$ is calculated in equation 3.4.

$$A = \frac{1}{2}\left(\begin{vmatrix} x_1 & x_2 \\ y_1 & y_2 \end{vmatrix} + \begin{vmatrix} x_2 & x_3 \\ y_2 & y_3 \end{vmatrix} + ... + \begin{vmatrix} x_n & x_1 \\ y_n & y_1 \end{vmatrix}\right) \tag{3.4}$$

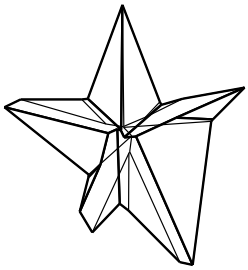
Regarding the vanish event for one edge only, there may be two combinations that lead to two different, valid offset polyhedra. The symmetric difference between these two offset polyhedra is a tetrahedron. This is the reason why the combination that contains the faster moving edge leads to the offset polyhedron with a smaller volume. Because the difference is a tetrahedron, the combination that leads to a reflex edge minimizes the volume of the offset polyhedron.

3.12.3 Example

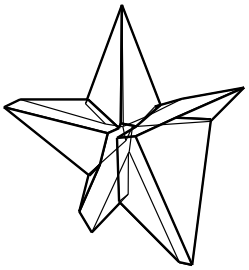
An example with different optimization criteria is shown in Figure 3.40. The visible vertex at the front, in the middle of the sea star has five convex and five reflex incident edges. For this vertex, there are $C_{10} = 1430$ combinations, where five lead to a valid offset polyhedra.



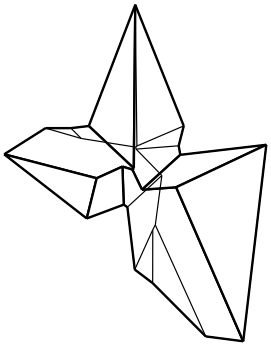
(a) Polyhedron *sea star*



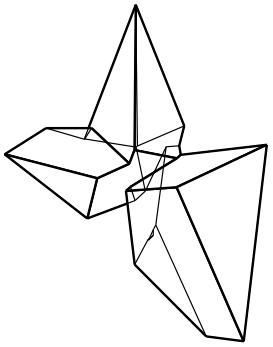
(b) Maximizing the number of convex edges, locally at each step (offset = 1)



(c) Minimizing the volume (offset = 1)



(d) Maximizing the number of convex edges, locally at each step (offset = 1.5, enlarged view)



(e) Minimizing the volume (offset = 1.5, enlarged view)

Figure 3.40: Optimization

4 Implementation

Many results presented in this work were obtained by experimenting with an implementation. This implementation grew from result to result. To obtain new results, simplicity and intelligibility of each step of the computation is important. For this purpose, computational speed does not play an important role.

At the beginning of this chapter, used data structures are presented. These data structures differ from those that are shipped in many software libraries. Afterwards, the implemented algorithm is explained. Although simplicity is a key feature of the algorithm, the time complexity is analyzed. The visualization for screen and for paper is explained. At the end, numbers of experimental results are summarized.

4.1 Data Structures

This section is devoted to used data structures. It is explained why default data structures of many software libraries are not sufficient to create offsets in a self-parallel manner.

In order to provide uniform concepts, data structures for the plane are based on the same ideas as data structures for the space.

4.1.1 In the Plane

Polygon

A polygon in the plane is stored as an ordered list of vertices by many software libraries. This is not sufficient for polygons with holes.

In the implementation the data structure of a polygon has a list of vertices and a list of edges. Because a list of all edges exists, a polygon can have holes. Each edge is directed and has a source and a destination vertex. Every vertex has a point, which defines its geometric position. The incoming and the outgoing incident edge is stored by the vertex. Therefore, incident edges are found in constant time.

Straight Skeleton

The data structure of straight skeletons in the plane consists of a list of nodes and a list of arcs. A list of events is also stored by the straight skeleton. A node is placed at the geometric position where an event happens. The position itself is stored by the node. To present the straight skeleton as a roof model, the time when the event occurs is stored by the node. This time represents the height of the node in the roof model. Every node has a list of incident arcs. For non-degenerate polygons, a node has either one (at the position of a vertex) or three arcs. This data structure is suitable for an arbitrary degree of a node.

During the shrinking process, the vertices trace out the arcs. The moving direction of a vertex is equal to the direction of its arc. An arc has a source node and a destination node. Furthermore, the left and the right polygon edge that define the arc are stored by the arc. The event itself stores the generated node and (optionally) the offset polygon at the time of the event.

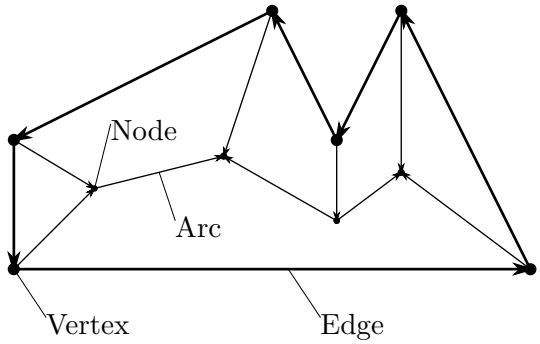


Figure 4.1: Data structure for polygons in the plane

4.1.2 In Space

Polyhedron

A frequently found use case is the rendering of a polyhedron on screen. Many data structures are focused on that use case. For such an application, a polygonal mesh of the surface is sufficient. This polygonal mesh usually consists only of triangles.

When facets of a polyhedron are shifted in a parallel manner, they may become non-convex. A facet may even have holes. The polyhedron itself may have holes or tunnels. Many data structures available in software libraries are not optimized for such requirements. To use such a publicly available software library, the data structures would have to be extended and adopted. Therefore, a data structure for polyhedra with holes and tunnels was implemented.

Halfedge Data Structure As indicated by its name, the halfedge is the central element of this data structure. Every facet of the polyhedron is bounded by counter-clockwise oriented halfedges. Every halfedge has a second halfedge with the opposite direction, an incident facet, a destination vertex and a reference to the next halfedge of the incident facet. The geometric position is stored by the destination vertex. The reference to the next halfedge is used to iterate counter-clockwise over all halfedges around the incident facet. The halfedge data structure is visualized in Figure 4.2.

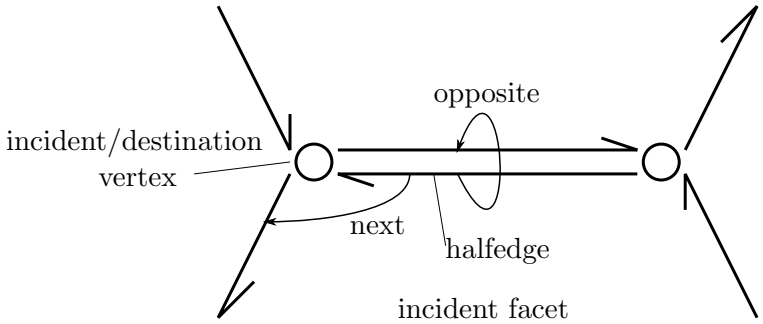


Figure 4.2: Halfedge data structure

CGAL uses this data structure for polyhedra [CGA]. Using this data structure (without extensions), it is not possible for a polyhedron to have holes inside a facet.

Implemented Data Structure In the implemented data structure, a polyhedron has a list of facets, a list of edges and a list of vertices. Every facet has a list of incident edges and a list of incident vertices. A facet can have holes because a list of edges exists. Every edge is directed and has a source and destination vertex. Furthermore, the left and the right (seen from outside) incident facet are stored by the edge. For efficiency reasons, the position of the edge in the lists of incident facets is also stored by the edge itself. Every vertex has a point, which shows its geometric position. The vertex also contains a list of incident edges and a list of incident facets. When these lists are ordered, the next element can be found in constant time. This is useful to iterate over all edges of a facet in a counter-clockwise order. To ensure the consistency of the data structure, a consistency check is implemented. Figure 4.3 shows an edge of this data structure.

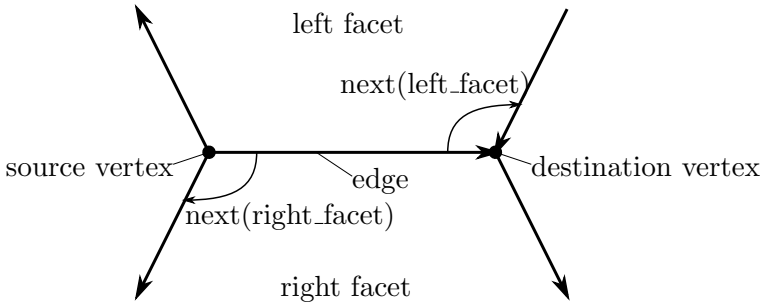


Figure 4.3: Data structure for polyhedra

Straight Skeleton

A straight skeleton in 3-space consists of a list of sheets, a list of arcs and a list of nodes. This data structure also includes a list of events. Each sheet has a list of incident arcs and a list of incident nodes. The facet on the positive side and the facet on the negative side are also stored within the sheet. Each arc has a source and a destination node. Incident sheets are stored in a list as part of the arc. In the non-degenerate case there are 3 incident sheets. Each node has a point, which marks the geometric position in 3-space. A list of incident arcs and a list of incident facets are part of the node. Each event has a node that was created by the event. Optionally, the offset polyhedron, at the time of the event, is stored with the event. To ensure the consistency of the data structure a consistency check is implemented.

4.1.3 File Format

To compute a straight skeleton, a polygon or a polyhedron is a necessary input. This input can be read from a file. The resulting straight skeletons are stored for analysis.

Various tools are handy for geometric modelling. The most common and easy to read file format is Wavefront's .obj file format. The implementation is able to load and save polyhedra using this format.

The resulting straight skeletons are stored on disk using a relational database. Polygons and polyhedra (with holes) can also be stored in this database.

Wavefront .obj File

Wavefront .obj files are simple ASCII text files. The file format is supported by many tools for geometric modelling: Art of Illusion, Blender, etc. Not all features of this format are explained here. Only essential parts are covered. The .obj file consists of a description of vertices and facets. The Cartesian coordinates of one vertex are in one line of the file. This line begins with `v` followed by the coordinates, which are separated by whitespace characters. The order of the vertices plays an important role for the definition of the facets. A line starting with `f` denotes the definition of a facet. The vertices of a facet are given in counter-clockwise order and are referenced by numbers. Reference number 1 corresponds to the first vertex in the file. An example of a tetrahedron is given in Listing 4.1.

```
v -1.0 -1.0 -1.0
v 1.0 1.0 -1.0
v 1.0 -1.0 1.0
v -1.0 1.0 1.0
f 1 2 3
f 1 3 4
f 1 4 2
f 4 3 2
```

Listing 4.1: tetrahedron.obj

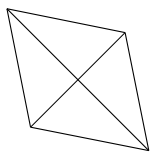


Figure 4.4: Tetrahedron

Every facet is defined as an ordered list of its vertices. It is not possible to store holes inside a facet using this file format. To handle this limitation, the implementation removes edges between coplanar facets when a polyhedron is loaded.

For the data structure it is mandatory to handle facets with holes. This file format can only save states where facets do not have any holes.

Relational Database

Relational databases store their data in tables. Therefore, the data structures have to be mapped to tables. Embedded databases do not require a server software. A software library (e.g. SQLite) is sufficient. This software library stores the data in a binary file on disk.

The implemented data structure of polygons is capable of having holes inside a polygon. This data structure is mapped to the tables defined in Listing 4.3. The straight skeleton data structure is also mapped to the corresponding tables. Contrary to text files, coordinates of a point are not converted to the decimal system and are stored as text. The coordinates are directly stored in binary form. In case floating point precision is not sufficient for the coordinates, their representation can be modified here.

The implemented data structure for polyhedra is also available in tabular form. Polyhedra with holes, tunnels and even holes in facets can be saved and loaded. Listing 4.4 shows the defined tables to save polyhedra and straight skeletons in 3-space.

The strengths of relational databases are visible with SQL queries. Listing 4.2 queries the database for polyhedra that have a straight skeleton where a specified event occurs more often than three times. The returned polyhedra are sorted in descending order by the number of the specified event.

```
SELECT PolyhedronID, COUNT(Events.etype) AS num_events
  FROM Events JOIN StraightSkeletons ON Events.SkelID=StraightSkeletons.SkelID
  WHERE Events.etype=7
  GROUP BY PolyhedronID
  HAVING num_events>3
  ORDER BY num_events DESC;
```

Listing 4.2: get_polyhedra.sql

Conventional file formats do not offer such possibilities.

```
CREATE TABLE Points (PointID INTEGER PRIMARY KEY,
  x REAL, y REAL);
CREATE TABLE Vertices (PolyID INTEGER NOT NULL, VID INTEGER NOT NULL,
  PointID INTEGER, PRIMARY KEY (PolyID, VID));
CREATE TABLE Edges (PolyID INTEGER NOT NULL, EID INTEGER NOT NULL,
  VID_SRC INTEGER, VID_DST INTEGER, PRIMARY KEY (PolyID, EID));
CREATE TABLE SkelEdgeData (PolyID INTEGER NOT NULL, EID INTEGER NOT NULL,
  speed REAL, PRIMARY KEY (PolyID, EID));
CREATE TABLE Polygons (PolyID INTEGER PRIMARY KEY,
  description TEXT, created INTEGER);
CREATE TABLE Nodes (SkelID INTEGER NOT NULL, NID INTEGER NOT NULL,
  PointID INTEGER, height REAL, PRIMARY KEY (SkelID, NID));
CREATE TABLE Arcs (SkelID INTEGER NOT NULL, AID INTEGER NOT NULL,
  NID_SRC INTEGER, NID_DST INTEGER, PRIMARY KEY (SkelID, AID));
CREATE TABLE Events (SkelID INTEGER NOT NULL, EventID INTEGER NOT NULL,
  etype INTEGER, NID INTEGER, PRIMARY KEY (SkelID, EventID));
CREATE TABLE StraightSkeletons (SkelID INTEGER PRIMARY KEY,
  PolyID INTEGER, description TEXT, created INTEGER);
```

Listing 4.3: data2d_schema.sql

```
CREATE TABLE Points (PointID INTEGER PRIMARY KEY,
  x REAL, y REAL, z REAL);
CREATE TABLE Planes (PlaneID INTEGER PRIMARY KEY,
  a REAL, b REAL, c REAL, d REAL);
CREATE TABLE Vertices (PolyhedronID INTEGER NOT NULL, VID INTEGER NOT NULL,
  PointID INTEGER, PRIMARY KEY (PolyhedronID, VID));
CREATE TABLE Edges (PolyhedronID INTEGER NOT NULL, EID INTEGER NOT NULL,
  VID_SRC INTEGER, VID_DST INTEGER, FID_L INTEGER, FID_R INTEGER,
  PRIMARY KEY (PolyhedronID, EID));
CREATE TABLE Triangles (
  PolyhedronID INTEGER NOT NULL, FID INTEGER NOT NULL, TID INTEGER NOT NULL,
  VID_1 INTEGER, VID_2 INTEGER, VID_3 INTEGER,
  PRIMARY KEY (PolyhedronID, FID, TID));
CREATE TABLE Facets (PolyhedronID INTEGER NOT NULL, FID INTEGER NOT NULL,
  PlaneID INTEGER, PRIMARY KEY (PolyhedronID, FID));
CREATE TABLE Polyhedrons (PolyhedronID INTEGER PRIMARY KEY,
  description TEXT, created INTEGER);
CREATE TABLE Nodes (SkelID INTEGER NOT NULL, NID INTEGER NOT NULL,
  PointID INTEGER, offset REAL, PRIMARY KEY (SkelID, NID));
CREATE TABLE Arcs (SkelID INTEGER NOT NULL, AID INTEGER NOT NULL,
  NID_SRC INTEGER, NID_DST INTEGER, PRIMARY KEY (SkelID, AID));
CREATE TABLE Sheets (SkelID INTEGER NOT NULL, SID INTEGER NOT NULL,
  PRIMARY KEY (SkelID, SID));
CREATE TABLE Sheets_Arcs (
  SkelID INTEGER NOT NULL, SID INTEGER NOT NULL, AID INTEGER NOT NULL,
  PRIMARY KEY (SkelID, SID, AID));
CREATE TABLE Events (SkelID INTEGER NOT NULL, EventID INTEGER NOT NULL,
  etype INTEGER, NID INTEGER, PRIMARY KEY (SkelID, EventID));
CREATE TABLE StraightSkeletons (SkelID INTEGER PRIMARY KEY,
  PolyhedronID INTEGER, config TEXT, description TEXT, created INTEGER);
```

Listing 4.4: data3d_schema.sql

4.2 Algorithm

As mentioned at the beginning of this chapter, the most important feature of the algorithm is its simplicity and therefore the simplicity of the implementation. Because of this simplicity, many results on the straight skeleton were found. An algorithm with a low time complexity was not the aim of the present work. This section is devoted to the simple straight skeleton algorithm for polyhedra in 3-space. Algorithm 4.1 explains the procedure.

Algorithm 4.1 SimpleStraightSkeleton(polyhedron)

Require: isValid(polyhedron)
Ensure: isEmpty(polyhedron)
1: skel_result ← initializeSkeleton(polyhedron)
2: polyhedron ← splitVerticesDegGt3(polyhedron)
3: offset ← 0; offset_prev ← 0
4: event ← findNextEvent(polyhedron, offset)
5: **while** ∃ event **do**
6: offset ← getOffset(event)
7: polyhedron ← shiftFacets(polyhedron, offset − offset_prev)
8: handleEvent(skel_result, event, polyhedron)
9: **if** isEnabled(debug) **then**
10: isConsistent(skel_result)
11: isConsistent(polyhedron)
12: isInsideBoundingBox(polyhedron)
13: **end if**
14: event ← findNextEvent(polyhedron, offset)
15: offset_prev ← offset
16: **end while**

4.2.1 Time Complexity Analysis

The given polyhedron has n_v vertices, n_e edges and n_f facets. r_e denotes the number of reflex edges. The number of reflex vertices is denoted by r_v. A vertex is reflex if all incident edges are reflex.

Basics

Euler's formula establishes a relation between the number of nodes, arcs, and faces of a connected planar graph. If a polyhedron is topologically equivalent to a sphere, its surface can be mapped to a connected planar graph in the plane. In this case, one facet of the polyhedron maps to the unbounded face of the graph. This connection is expressed in equation 4.1

$$n_v - n_e + n_f = 2 \tag{4.1}$$

Every edge is incident to two facets and every facet has at least three edges. This gives $2n_e \geq 3n_f$. Applying this relation to Euler's formula bounds the number of edges and facets.

$$n_e \leq 3n_v - 6 \qquad\qquad n_f \leq 2n_v - 4$$

These bounds are only valid for $n_v \geq 3$.

Using the ingredients above, the expected degree of a vertex is bound. A naive idea for this is to sum up all edges, multiply them by two (each edge has two vertices), and divide the result by the number of vertices. The result of this would be $\frac{2(3n_v-6)}{n_v}$. Unfortunately, the relations from above are only valid for $n_v \geq 3$. Therefore, the summation has to start at $n_v \geq 3$. A tetrahedron has four vertices and six edges. The total degree of all four vertices is twelve.

An upper bound for the expected degree of a vertex v_i is calculated as follows.

$$\begin{aligned} E[\deg v_i] &= \frac{1}{n_v - 4} \sum_{i=5}^{n_v} \deg v_i \\ &\leq \frac{1}{n_v - 4} \left(\left(\sum_{i=1}^{n_v} \deg v_i \right) - 12 \right) \\ &\leq \frac{2(3n_v - 6) - 12}{n_v - 4} = 6 \end{aligned}$$

Initialization: Split Vertices

At the beginning, vertices of degree > 3 are split. The required time depends on the method used.

Convex and reflex vertices are split in $O(\deg v_i \cdot \log(\deg v_i))$ time. Such vertices have only one unique solution.

For pointed vertices, the weighted straight skeleton in the plane can be used. Using this method, only one possible offset polyhedron is generated.

To check all combinations of bisector graphs, an exponential number of unrooted binary trees is generated. The degree of the vertex determines the number of unrooted binary trees. The exact number is equal to the Catalan number of the degree of the vertex. On an ordinary computer, the implementation takes approximately one second for a vertex of degree 10.

Finding the Next Event

The required time to find the next event depends on the type of the event. The next vanish event is determined in linear time, $O(n_e)$. This time complexity is achieved because it is sufficient to investigate a local neighbourhood of the edge.

The investigation of a local neighbourhood is also sufficient for most of the contact events. For these local contact events, all edges are investigated once. To find an edge that meets the current investigated edge, all vertices of incident facets are examined. Only incident edges of these vertices are possible candidates to meet the

current edge. The number of incident vertices of a facet is equal to the number of incident edges. The expected value of incident edges of a facet is bounded by 6. This is observed by drawing a graph on the surface of the polyhedron. A node is put on every facet. The nodes of adjacent facets are connected. This creates the edges of the graph. The resulting graph on the surface of the polyhedron is embedded into the plane. If the polyhedron is a topological sphere, the graph can be embedded into the plane without crossings. Therefore, the next local contact event is found in $O(n_e)$ time for topological spheres.

Non-local events are the edge-edge contact event and the vertex-facet contact event. The next edge-edge contact event is found by checking all reflex edges against each other. This takes $O(r_e^2)$ time. The next edge-facet contact event is found by checking all reflex vertices with all facets of the polyhedron. This takes $O(r_v \cdot n_f)$ time.

Under the assumption that the input polyhedron is topologically equivalent to a sphere, the next event is found in $O(n_e + r_v \cdot n_f + r_e^2) < O(n_v^2)$ time.

Number of Events

In the non-degenerate case, an event may happen when four facet-supporting planes meet in a point during the shrinking process. If an event happens, a node of the straight skeleton is created at this point. This observation leads to the trivial upper bound of $O(n_f^4) = O(n_v^4)$ for the total number of events.

4.3 Visualization

The implementation features an interactive animation using OpenGL. The shrinking process of the polyhedron can be observed on screen. The visualization is independent from the algorithm that computes offset polyhedra and the straight skeleton. One thread runs the algorithm. One thread is responsible for the visualization. Another thread handles key strokes. While the implementation is executed, the algorithm and the visualization access the same data structures in memory. Therefore, locking mechanisms are required.

OpenGL is perfectly suited to render three-dimensional geometric objects on screen. This method generates raster images, pixels for the screen. For paper, vector graphics are preferred over raster images. Another feature of the implementation is non-interactive vector graphics rendering. All figures of polygons, polyhedra, straight skeletons and spherical skeletons in the present work were created by this function.

4.3.1 Rendering to PostScript

To render a polyhedron to an PostScript file, the following information is required. The coordinates of the vertices are stored by the data structure. The actual view on screen is determined by geometric transformations and a perspective projection. Using this information, the polyhedron on screen is rendered as vector graphic into a PostScript file. In this section, only essential parts that are relevant for this procedure are covered. OpenGL and PostScript have many more features than those that are covered here.

Several geometric models in three-dimensional space are rendered by OpenGL onto the two-dimensional screen. The geometric models are transformed by translation, rotation and scaling to be placed into the global coordinate system. Afterwards, the global scene is also transformed using such operations. As a result, the z-axis is perpendicular to the screen and points out of it. These transformations are achieved with matrix multiplications. Translation, rotation and scaling are covered by the modelview matrix MV. The perspective projection is done with the projection matrix P. MV and P are 4×4 matrices. The coordinates of the given point of the geometric in 3-space are put into a vector $p_{in} = (x, y, z, 1)$. Equation 4.2 shows how this vector is transformed using matrix multiplications.

$$p_{out} = P \cdot MV \cdot p_{in} \tag{4.2}$$

The result p_{out} is a four-dimensional vector. This vector is the position of the given point on screen after a perspective projection. The fourth coordinate of p_{out} is used to normalize the device (screen) coordinates. The so-called *viewport* is used to create a raster image and contains the resolution of this image.

OpenGL provides functions to create the transformation matrices. The graphics card puts triangles in 3-space onto the screen using these matrices. The currently used transformation matrices are read from the graphics card. Functions to read these matrices are shown in Listing 4.5.

```
float modelview[16];
float projection[16];
int viewport[4];
glGetFloatv(GL_MODELVIEW_MATRIX, modelview);
glGetFloatv(GL_PROJECTION_MATRIX, projection);
glGetIntegerv(GL_VIEWPORT, viewport);
```

Listing 4.5: get_gl_matrices.c

The same transformation matrices are used to map the coordinates of the vertices onto the paper. The rasterisation is omitted for the paper.

PostScript is a scripting language focused on the creation of vector graphics. The commands to create such a vector graphic are contained in an ASCII text file. The following example shows how a line is drawn using PostScript. The default unit size of PostScript is $\frac{1}{72}$ inch. Listing 4.6 creates a line that is 1 mm thick and 50 mm long. The image itself measures 70 mm in width and 20 mm in height.

```
%!PS-Adobe-3.0 EPSF-3.0
%%BoundingBox: 0 0 198 57

2.83465 setlinewidth

newpath
28.5 28.5 moveto
170.23 28.5 lineto
stroke
```

Listing 4.6: line.eps

4.4 Experimental Results

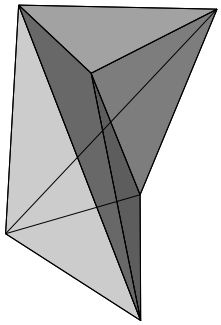
The algorithm from Section 4.2 has been implemented in C++. Experimental results were obtained with an ordinary computer, an Intel i5-2500 at 3.3 GHz with 4 GB of main memory. Table 4.1 shows used examples of polyhedra. Some of them are visualized in Figure 4.5.

The sizes of the resulting straight skeletons are shown in Table 4.2 and Table 4.3. For the measuring of the required time for computation, the application was compiled with the debug flag disabled. The binary was executed without visualization. This accelerates the computation because the data structure would be locked while it gets drawn on the screen. The results in Table 4.2 were obtained by keeping the polyhedra convex during the shrinking process. Table 4.3 shows the sizes of the straight skeletons when the volumes of the offset polyhedra are minimized.

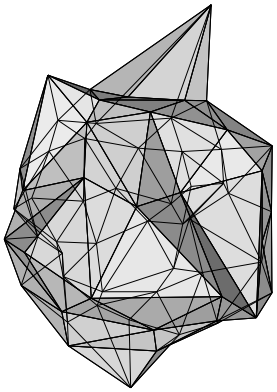
The number of events are counted in Table 4.4 and Table 4.5.

PolyhID	Name	Figure	Vertices	Edges	Facets
1	Shaken Cube		8	12	6
2	Held	3.3	24	36	14
3	Iron Maiden Pizza Box	3.4	32	48	24
4	Saddle Point	3.16	9	16	9
5	Sea Star	3.40	12	30	20
6	Wedge on Tabletop	A.1	16	24	11
7	Iron Maiden	A.3	20	30	15
8	Schönhardt	4.5a	6	12	8
9	Verworrtakelt I	4.5b	66	192	128
10	Verworrtakelt II		66	192	128
11	Armadillo (small)		50	144	96
12	Armadillo	4.5c	99	291	194
13	Asteroid		20	54	36
14	Stanford Bunny	4.5d	152	450	300
15	Pawn (chess)		42	119	79
16	Chinese Lion		89	261	174
17	Convex Piece		38	108	72
18	Hand	4.5e	52	150	100
19	Shaken Sphere		66	192	128
20	Venus (small)		63	183	122
21	Venus	4.5f	142	420	280

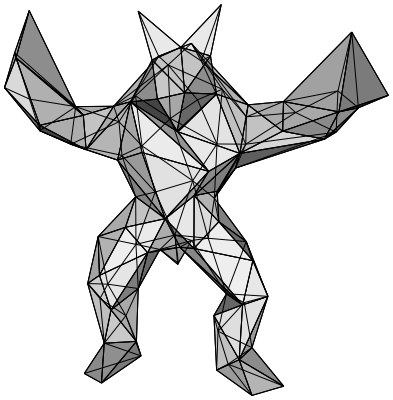
Table 4.1: Examples of polyhedra



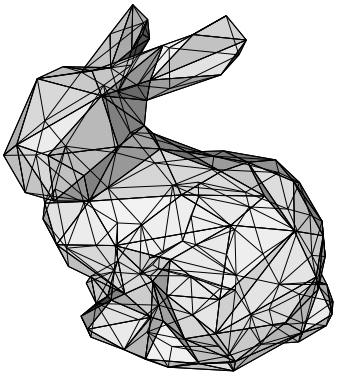
(a) 8, Schönhardt



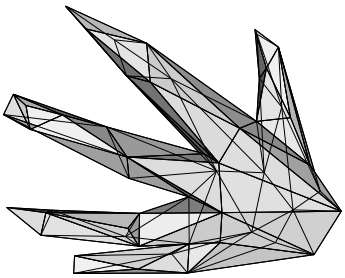
(b) 9, Verworrtakelt I



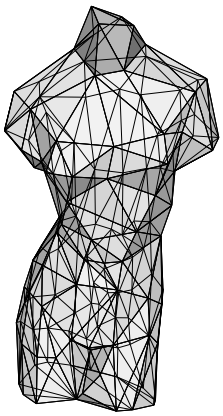
(c) 12, Armadillo



(d) 14, Stanford Bunny



(e) 18, Hand



(f) 21, Venus

Figure 4.5: Examples of polyhedra

PolyhID	Nodes	Arcs	Sheets	Time [s]
1	12	12	13	0.01
2	60	84	61	0.11
3	183	318	184	9.15
4	22	33	28	0.03
5	41	76	66	0.92
6	41	58	42	0.12
7	69	108	70	0.52
8	12	18	19	0.01
9	402	798	596	22.57
10	388	770	583	24.02
11	229	452	368	5.48
12	512	1018	802	25.68
13	86	166	135	0.48
14	949	1892	1400	97.28
15	210	413	325	4.39
16	483	960	744	27.00
17	196	386	299	2.36
18	238	470	386	6.41
19	335	664	522	6.80
20	344	682	526	11.02
21	948	1890	1376	125.60

Table 4.2: Straight skeletons for locally maximizing the number of convex edges

PolyhID	Nodes	Arcs	Sheets	Time [s]
1	12	12	13	0.01
2	60	84	61	0.11
3	183	318	184	9.34
4	19	27	26	0.03
5	43	80	71	0.94
6	41	58	42	0.12
7	69	108	70	0.52
8	14	22	23	0.03
9	475	944	671	38.82
10	467	928	667	39.12
11	270	534	417	7.83
12	587	1168	885	39.35
13	86	166	135	0.51
14	980	1954	1440	123.30
15	232	457	351	5.62
16	519	1032	781	37.71
17	196	386	299	2.37
18	287	568	439	9.30
19	341	676	529	7.13
20	411	816	597	18.37
21	1093	2180	1527	180.00

Table 4.3: Straight skeletons for minimizing the volume

PolyhID	Vanish Events						Contact Events						
	E1	E2	E3	E4	E5	E6	V-E	V-V I	V-V II	V-V-E I	V-V-E II	E-E	V-F
1	1	0	2	0	0	1	0	0	0	0	0	0	0
2	25	0	10	0	0	1	0	0	0	0	0	0	0
3	22	0	38	0	0	16	30	0	0	30	0	9	6
4	3	0	5	0	0	2	2	0	0	1	0	0	0
5	6	0	13	0	0	4	3	0	0	3	0	0	0
6	9	0	6	0	0	3	3	0	0	3	0	0	1
7	5	0	10	0	0	8	11	0	0	11	0	1	3
8	1	0	2	0	0	2	0	0	0	1	0	0	0
9	145	0	140	0	7	4	33	0	0	5	0	1	1
10	119	1	137	0	7	8	36	0	0	10	0	2	2
11	52	0	89	0	0	11	17	0	0	10	0	0	0
12	130	0	196	0	4	18	48	0	0	17	0	0	0
13	33	0	32	0	0	1	0	0	0	0	0	0	0
14	333	0	354	0	6	9	85	0	0	9	0	0	1
15	90	0	73	0	2	1	2	0	0	0	0	0	0
16	138	0	185	0	5	11	45	0	0	10	0	0	0
17	89	0	68	0	0	1	0	0	0	0	0	0	0
18	56	0	92	0	3	9	18	0	0	8	0	0	0
19	144	0	124	0	0	1	0	0	0	0	0	0	0
20	120	0	121	0	4	7	21	0	0	7	0	1	0
21	361	1	317	0	12	13	89	0	0	12	0	0	1

Table 4.4: Event count for locally maximizing the number of convex edges

PolyhID	Vanish Events						Contact Events						
	E1	E2	E3	E4	E5	E6	V-E	V-V I	V-V II	V-V-E I	V-V-E II	E-E	V-F
1	1	0	2	0	0	1	0	0	0	0	0	0	0
2	25	0	10	0	0	1	0	0	0	0	0	0	0
3	22	0	38	0	0	16	30	0	0	30	0	9	6
4	1	1	4	0	0	2	2	0	0	0	0	0	0
5	11	0	9	0	3	3	3	0	0	2	0	0	0
6	9	0	6	0	0	3	3	0	0	3	0	0	1
7	5	0	10	0	0	8	11	0	0	11	0	1	3
8	1	2	2	0	0	2	0	0	0	0	0	1	0
9	188	1	143	0	8	9	47	0	1	9	0	2	1
10	177	5	134	0	8	13	47	0	0	12	0	3	2
11	77	3	82	0	5	14	29	0	0	10	0	0	0
12	191	4	185	0	8	23	59	0	0	18	0	0	0
13	33	0	32	0	0	1	0	0	0	0	0	0	0
14	366	4	338	0	11	12	84	0	0	10	0	3	0
15	94	0	74	0	6	2	13	0	0	1	0	0	0
16	157	3	187	0	3	15	54	0	0	11	0	0	0
17	89	0	68	0	0	1	0	0	0	0	0	0	0
18	87	5	92	0	2	13	29	0	0	7	0	0	0
19	147	0	124	0	1	1	2	0	0	0	0	0	0
20	157	3	129	0	5	9	36	0	0	7	0	2	0
21	438	4	341	0	15	15	124	0	0	12	0	1	1

Table 4.5: Event count for minimizing the volume

4.4.1 Notes and Comments

The experimental results give rise to observations. Minimizing the volume of the polyhedra during the shrinking process tends to increase the number of reflex edges. When the number of reflex edges is increased, more time for computation is required. More contact events with global influence are possible (edge-edge contact event, vertex-facet contact event) and need to be found. Furthermore, the total number of events tends to increase. This also increases the size of the straight skeleton.

5 Conclusion and Further Work

The aim of the present work was to study self-parallel offsets of polyhedra in 3-space. A self-parallel offset of a polyhedron is created by shifting its facets in a way that each facet stays parallel to its initial orientation. An investigation of such an essential topic in the field of computational geometry had not yet been covered.

Literature on self-parallel offsets of polygons in the plane exists. Such an offset is created by shifting all edges of the polygon. The edges are required to stay parallel to their initial orientation. During the offsetting process, events occur that change the polygon combinatorially and topologically: Edges may vanish or a reflex vertex splits an edge into two parts. By following the vertices of the polygon during the offsetting process, a skeletal structure, the straight skeleton, is revealed. As its name implies, this skeleton consists only of straight line segments.

The second chapter of this work studied properties and algorithms of the straight skeleton in the plane. Most important properties were highlighted. The gained insights of cited papers were summarized in a consecutive way. First algorithms use the procedural definition of the straight skeleton for its computation. These algorithms use a priority queue to arrange possible events in chronological order. More advanced algorithms use previously discovered properties for a more efficient computation. A difficulty for an efficient computation was the fact that reflex vertices have a global influence on the offset polygon. This difficulty was extracted with the motorcycle graph. Recent work has discovered that the motorcycle graph can be computed more efficiently when the events are not processed in chronological order.

The current state of the art for the straight skeleton in the plane shows a gap between the lower and upper time complexity bound. The lower bound is linear in the size of the input polygon. For polygons with holes, the lower bound was increased to linear-logarithmic with a reduction to sorting. The lowest upper bound results from a reduction of the straight skeleton computation to a motorcycle graph computation. This reduction leads to a sub-quadratic time bound for computing the straight skeleton.

Before the present work, the offsetting process for polyhedra in 3-space has not received much attention. A substantial difference between the offsetting process in the plane and the offsetting process in 3-space happens in the very first moment. In 3-space, a point is defined by three planes. In order to shift the facets, vertices with a degree higher than three need to be split. The present work has proven the existence of a solution for this problem. Simple examples have shown the ambiguity of the offsetting process in space. Vertices that can be split in exponentially many different ways have been described. The presented algorithm and its implementation were helpful to compute all of them.

After each vertex has been split, all facets are shifted. The shifting continues as long as no self-intersections on the surface occur. To keep the surface intersection free during the whole offsetting process, events need to be detected and handled correctly. Possible events were shown by giving examples. These examples were illustrated by a shrinking polyhedron, which was visualized before and after an event. The presented algorithm was used to compute the solutions to the events.

Properties of the straight skeleton of polyhedra were proven. An important property is the monotonicity of its cells, which also applies to the three-dimensional case.

All presented algorithms were implemented in C++. Rather than having a focus on the efficiency, this implementation was kept easily understandable to gain insight into the offsetting process. Additional checks after each step of the computation were helpful for the evaluation of the ideas. An interactive visualization of polyhedra during the shrinking process was a necessary tool to study and verify the results.

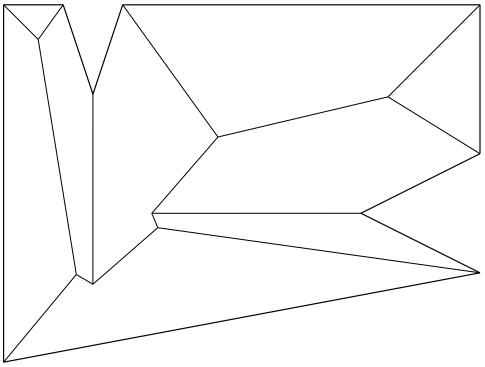
5.1 Mesh Generation Based on Straight Skeletons

The finite element method (FEM) is a common tool for material stress analysis in the field of mechanical engineering. For the calculation, the geometry of the mechanically stressed component is decomposed into finite elements, like tetrahedra or prisms. For these finite elements, there are equations to calculate mechanical stress.

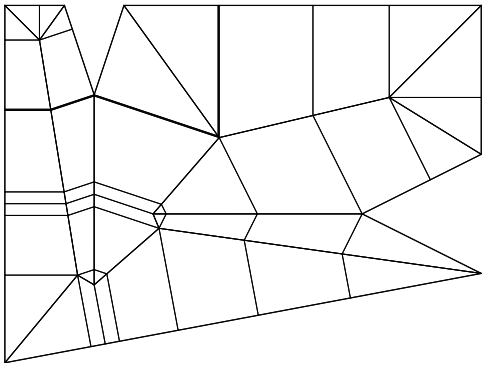
To use this method for a the calculation of a mechanical component, a mesh of finite elements is required. There are many algorithms available for automatic mesh generation. Some of them create a hexahedral mesh from the outside to the inside. This may cause overlappings of the elements. Smoothing algorithms are used to equally distribute the generated mesh afterwards. This removes possible overlappings.

Another approach to automatically generate a mesh was described by Storti et al. [STG+97]. Starting with a skeletal representation of the mechanical component, a mesh is constructed from the inside to the outside. Storti et al. have used the medial axis as skeletal structure. A drawback of the medial axis is that it consists of parabolic parts. The straight skeleton consists only of piecewise linear parts.

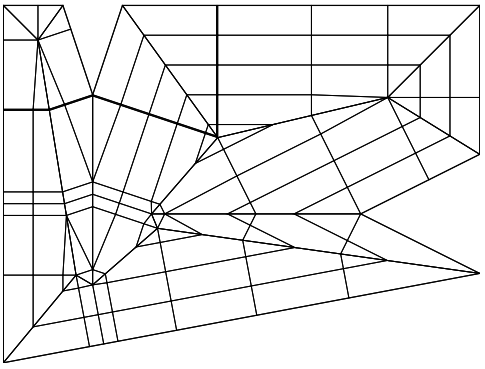
Figure 5.1 shows, how the straight skeleton could possibly be used for mesh generation. It is very likely that this idea can be generalized to create meshes for polyhedra in 3-space, but this requires further investigation.



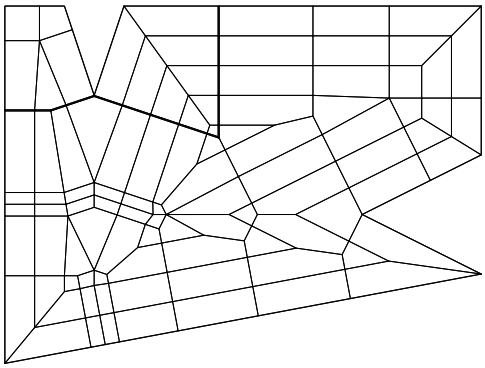
(a) Given polygon and its straight skeleton



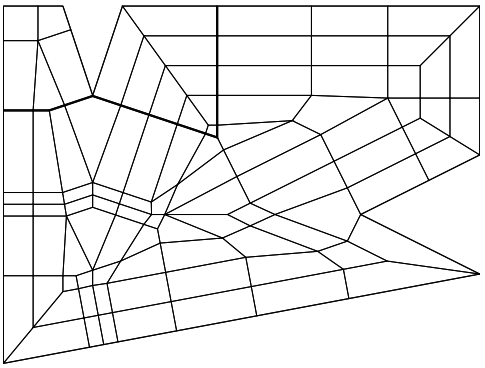
(b) Step 1: Nodes emanate rays



(c) Step 2: Offset polygons are drawn



(d) Step 3: Adjacent triangles are merged



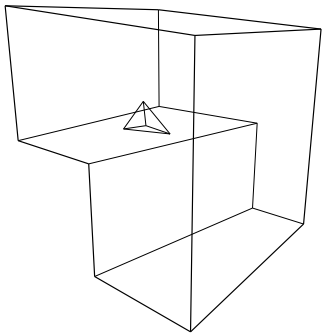
(e) Step 4: Edges with two incident five-sided cells are removed

Figure 5.1: Mesh generation based on straight skeletons

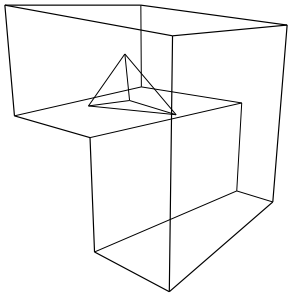
A Examples

The following figures visualize the shrinking process of selected polyhedra. The constructed straight skeleton is drawn afterwards. The complexity of the shown examples is kept low so that the visualization is still clearly understandable.

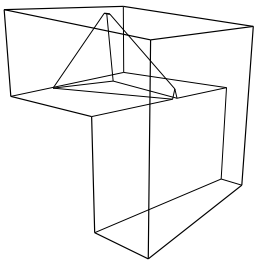
The size of the polyhedra, as well as the size of the resulting skeletons, and the number of events are counted in the tables of Section 4.4.



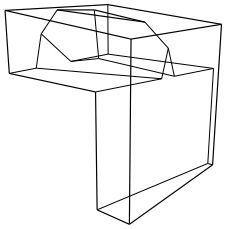
(a) Offset = 0



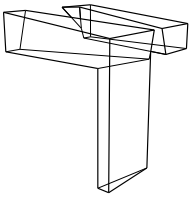
(b) Offset = 1



(c) Offset = 2



(d) Offset = 3



(e) Offset = 4

(f) Offset = 5

Figure A.1: Wedge on Tabletop

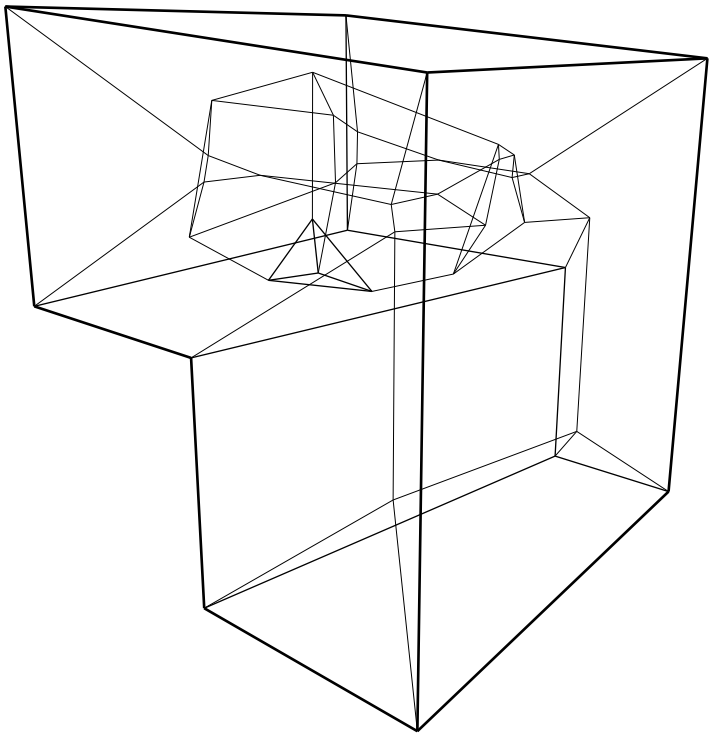
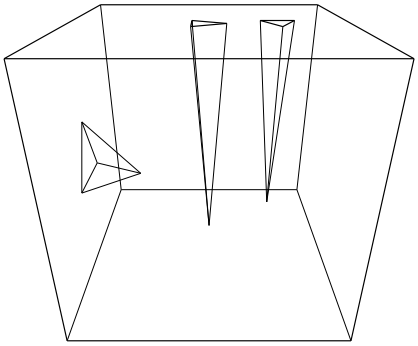
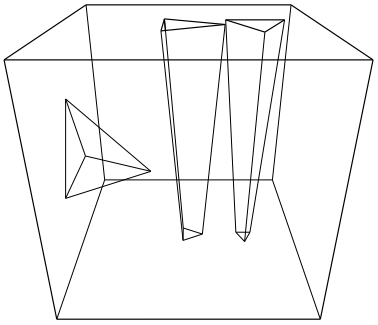


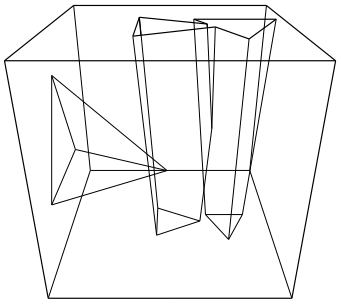
Figure A.2: Straight skeleton of Wedge on Tabletop



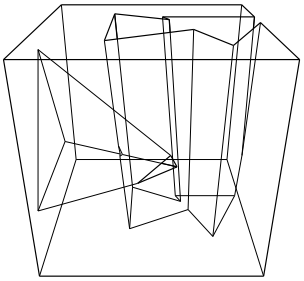
(a) Offset = 0.0



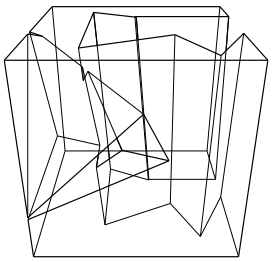
(b) Offset = 0.5



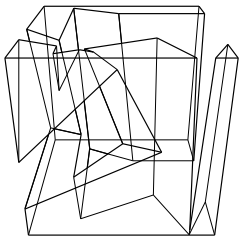
(c) Offset = 1.0



(d) Offset = 1.5



(e) Offset = 2.0



(f) Offset = 2.5

Figure A.3: Iron Maiden

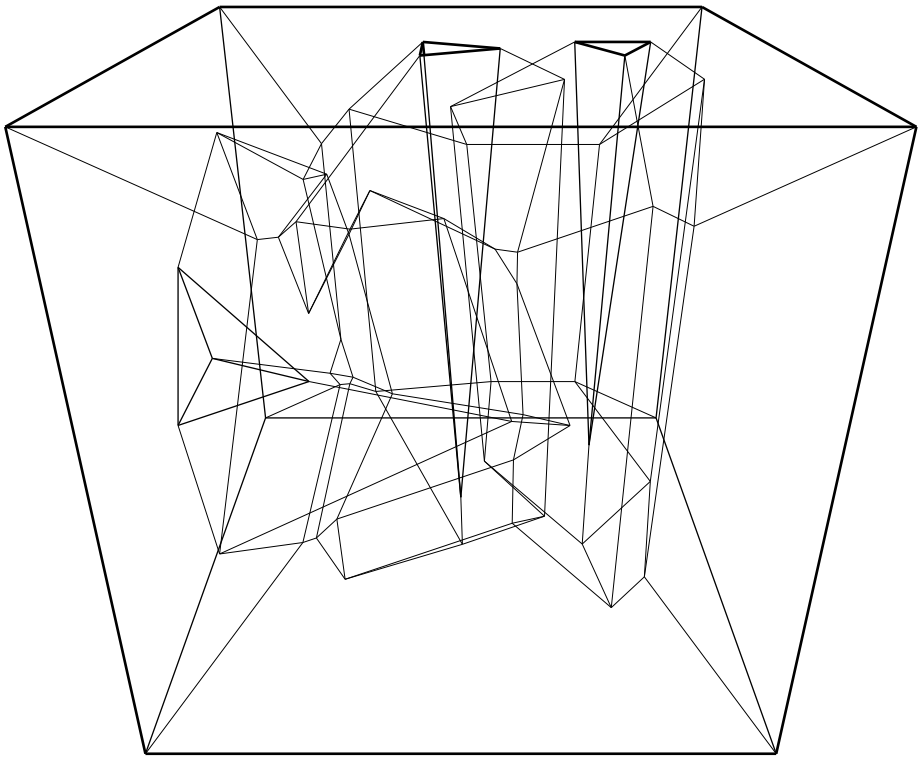


Figure A.4: Straight skeleton of Iron Maiden

B Euclidean Geometry

This appendix includes the equations that are required to compute the geometry of the straight skeleton. It starts with an introduction and gives definitions of basic geometric elements and their notation. Most important for the skeleton is the angle bisector and the intersection of lines in the plane and planes in space. Finally, the representation of the numbers during the calculation is examined to have an exact representation of geometric elements.

The ancient Greek mathematician Euclid of Alexandria brought geometry into a logical system ∼300 BC. He used intuitively appealing axioms to deduce theorems. These theorems explained Euclidean's geometry. It was the only geometry for a long time.

B.1 Basic Elements

B.1.1 Point

A point p defines one position in space. In a Cartesian coordinate system a point is uniquely defined by numerical coordinates with straight line axes that are perpendicular to each other.

$$p = (x, y, z) \tag{B.1}$$

B.1.2 Vector

A vector $\vec{v}$ describes a direction and a distance in space but without a position. It can be interpreted as the difference of two points.

$$\vec{v} = (v_x, v_y, v_z) \tag{B.2}$$

Length

$$|\vec{v}| = \sqrt{v_x^2 + v_y^2 + v_z^2} \tag{B.3}$$

Scalar product

$$\vec{u} \cdot \vec{v} = u_x v_x + u_y v_y + u_z v_z \tag{B.4}$$

The result is a scalar. It is used to calculate the inner angle φ between two vectors $(\vec{u}, \vec{v})$.

$$\cos\varphi = \frac{\vec{u} \cdot \vec{v}}{|\vec{u}|\,|\vec{v}|} \tag{B.5}$$

Another application is the orthogonal projection. A vector $\vec{v}$ is projected orthogonally onto a vector $\vec{u}$.

$$\text{proj}_{\vec{u}}\vec{v} = \frac{\vec{u}}{|\vec{u}|}\frac{\vec{u}\cdot\vec{v}}{|\vec{u}|} \tag{B.6}$$

Cross product

$$\vec{u}\times\vec{v} = \begin{pmatrix} u_x \\ u_y \\ u_z \end{pmatrix} \times \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix} = \begin{pmatrix} u_y v_z - u_z v_y \\ u_z v_x - u_x v_z \\ u_x v_y - u_y v_x \end{pmatrix} \tag{B.7}$$

The result is a vector that is perpendicular to both vectors $\vec{u}$ and $\vec{v}$.

B.1.3 Line

In the Plane

In two-dimensional space a line separates the space into two half-spaces. Because the line has an orientation, one side is positive. The other one is negative.

$$l : ax + by + c = 0 \tag{B.8}$$

The normal vector $\vec{n}$ is perpendicular to the line and points to the positive side.

$$\vec{n} = (a, b) \tag{B.9}$$

Initialization Two points (p_1, p_2) are used to initialize a line.

$$ax_1 + by_1 + c = 0 \tag{B.10a}$$
$$ax_2 + by_2 + c = 0 \tag{B.10b}$$

$$\begin{pmatrix} x_1 & y_1 \\ x_2 & y_2 \end{pmatrix} \begin{pmatrix} \frac{a}{c} \\ \frac{b}{c} \end{pmatrix} = \begin{pmatrix} -1 \\ -1 \end{pmatrix} \tag{B.11}$$

$$\frac{1}{c}\begin{pmatrix} a \\ b \end{pmatrix} = \frac{1}{x_1 y_2 - x_2 y_1} \begin{pmatrix} y_2 & -y_1 \\ -x_2 & x_1 \end{pmatrix} \begin{pmatrix} -1 \\ -1 \end{pmatrix} \tag{B.12}$$

There are two solutions for this equation. The selected solution defines the left side of the line with a direction from p_1 to p_2 to be the positive side.

$$a = y_1 - y_2 \tag{B.13a}$$
$$b = x_2 - x_1 \tag{B.13b}$$
$$c = x_1 y_2 - x_2 y_1 \tag{B.13c}$$

In Space

In three-dimensional space a line is defined by a point p_0 and a direction $\vec{v}$.

$$(x, y, z) = p_0 + \lambda\vec{v} \tag{B.14}$$

B.1.4 Plane

A plane separates the three-dimensional space into two half-spaces.

$$h : ax + by + cz + d = 0 \tag{B.15}$$

The normal vector $\vec{n}$ is perpendicular to the plane and points to the positive side.

$$\vec{n} = (a, b, c) \tag{B.16}$$

Initialization Three points (p_1, p_2, p_3) are used to initialize a plane.

$$ax_1 + by_1 + cz_1 + d = 0 \tag{B.17a}$$
$$ax_2 + by_2 + cz_2 + d = 0 \tag{B.17b}$$
$$ax_3 + by_3 + cz_3 + d = 0 \tag{B.17c}$$

$$\underbrace{\begin{pmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \end{pmatrix}}_{A} \begin{pmatrix} \frac{a}{d} \\ \frac{b}{d} \\ \frac{c}{d} \end{pmatrix} = \begin{pmatrix} -1 \\ -1 \\ -1 \end{pmatrix} \tag{B.18}$$

$$\frac{1}{d}\begin{pmatrix} a \\ b \\ c \end{pmatrix} = \frac{1}{\det(A)} \begin{pmatrix} y_2z_3 - z_2y_3 & z_1y_3 - y_1z_3 & y_1z_2 - z_1y_2 \\ z_2x_3 - x_2z_3 & x_1z_3 - z_1x_3 & z_1x_2 - x_1z_2 \\ x_2y_3 - y_2x_3 & y_1x_3 - x_1y_3 & x_1y_2 - y_1x_2 \end{pmatrix} \begin{pmatrix} -1 \\ -1 \\ -1 \end{pmatrix} \tag{B.19}$$

This equation has two solutions. The following solution defines the positive side of the plane as follows. The orientation of the points (p_1, p_2, p_3) define the rotation of a screw that is tightened clockwise. The screw moves to the positive side of the plane when it is rotated clockwise.

$$a = y_2z_3 + z_1y_3 + y_1z_2 - z_2y_3 - y_1z_3 - z_1y_2 \tag{B.20a}$$
$$b = z_2x_3 + x_1z_3 + z_1x_2 - x_2z_3 - z_1x_3 - x_1z_2 \tag{B.20b}$$
$$c = x_2y_3 + y_1x_3 + x_1y_2 - y_2x_3 - x_1y_3 - y_1x_2 \tag{B.20c}$$
$$d = x_3y_2z_1 + y_3z_2x_1 + z_3x_2y_1 - x_1y_2z_3 - y_1z_2x_3 - z_1x_2y_3 \tag{B.20d}$$
$$d = -x_ia - y_ib - z_ic \qquad i \in \{1, 2, 3\} \tag{B.20e}$$

B.1.5 Sphere

A sphere is defined by its center (point) and its radius.

B.2 Distance

The Euclidean distance between two points $p_1 = (x_1, y_1, z_1)$ and $p_2 = (x_2, y_2, z_2)$ is defined as:

$$d(p_1, p_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2} \tag{B.21}$$

In the 19th century other ways of measuring distances came up. Non-euclidean geometries began to evolve. In the year 1906 Maurice Fréchet found conditions that must be fulfilled by distance functions. A distance is valid and defines a metric space, if (and only if) the following conditions hold:

$$\begin{aligned} d(p_1, p_2) &\geq 0 && \text{(B.22}a\text{)} \\ d(p_1, p_2) &= 0 \Leftrightarrow p_1 = p_2 && \text{(B.22}b\text{)} \\ d(p_1, p_2) &= d(p_2, p_1) && \text{(B.22}c\text{)} \\ d(p_1, p_3) &\leq d(p_1, p_2) + d(p_2, p_3) && \text{(B.22}d\text{)} \end{aligned}$$

B.2.1 Point-Plane

The orthogonal distance between a plane h and a point p_0 is calculated as follows. A vector $\vec{v}$ from any point on the plane to the point p_0 is given by

$$\vec{v} = \begin{pmatrix} x_0 - x \\ y_0 - y \\ z_0 - z \end{pmatrix} \tag{B.23}$$

$\vec{v}$ is projected onto the normal vector $\vec{n}$ of the plane. The length of the projection is the distance between the plane h and the point p_0.

$$\begin{aligned} d(h, p_0) &= |\text{proj}_{\vec{n}}\vec{v}| && \text{(B.24}a\text{)} \\ &= \frac{|\vec{n} \cdot \vec{v}|}{|\vec{n}|} && \text{(B.24}b\text{)} \\ &= \frac{|ax_0 + by_0 + cz_0 - ax - by - cz|}{\sqrt{a^2 + b^2 + c^2}} && \text{(B.24}c\text{)} \end{aligned}$$

$$d(h, p_0) = \frac{|ax_0 + by_0 + cz_0 + d|}{\sqrt{a^2 + b^2 + c^2}} \tag{B.25}$$

B.3 Intersection

In two-dimensional space two lines (l_1, l_2) intersect at a point $p = (x, y)$.

$$l_1: \quad a_1 x + b_1 y + c_1 = 0 \tag{B.26a}$$
$$l_2: \quad a_2 x + b_2 y + c_2 = 0 \tag{B.26b}$$

$$\begin{pmatrix} a_1 & b_1 \\ a_2 & b_2 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} -c_1 \\ -c_2 \end{pmatrix} \tag{B.27}$$

$$\begin{pmatrix} x \\ y \end{pmatrix} = \frac{1}{a_1 b_2 - a_2 b_1} \begin{pmatrix} b_2 & -b_1 \\ -a_2 & a_1 \end{pmatrix} \begin{pmatrix} -c_1 \\ -c_2 \end{pmatrix} \tag{B.28}$$

The lines (l_1, l_2) are parallel, if $a_1 b_2 - a_2 b_1 = 0$.

In three-dimensional space three planes (h_1, h_2, h_3) intersect at a point $p = (x, y, z)$.

$$h_1: \quad a_1 x + b_1 y + c_1 z + d_1 = 0 \tag{B.29a}$$
$$h_2: \quad a_2 x + b_2 y + c_2 z + d_2 = 0 \tag{B.29b}$$
$$h_3: \quad a_3 x + b_3 y + c_3 z + d_3 = 0 \tag{B.29c}$$

$$\underbrace{\begin{pmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{pmatrix}}_{A} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} -d_1 \\ -d_2 \\ -d_3 \end{pmatrix} \tag{B.30}$$

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \frac{1}{\det(A)} \begin{pmatrix} b_2 c_3 - c_2 b_3 & c_1 b_3 - b_1 c_3 & b_1 c_2 - c_1 b_2 \\ c_2 a_3 - a_2 c_3 & a_1 c_3 - c_1 a_3 & c_1 a_2 - a_1 c_2 \\ a_2 b_3 - b_2 a_3 & b_1 a_3 - a_1 b_3 & a_1 b_2 - b_1 a_2 \end{pmatrix} \begin{pmatrix} -d_1 \\ -d_2 \\ -d_3 \end{pmatrix} \tag{B.31}$$

Two planes (h_1, h_2) intersect at a line.

$$\vec{v} = \vec{n}_1 \times \vec{n}_2 \tag{B.32}$$

Depending on the direction of $\vec{v}$, one coordinate of the point p_0 is set to 0.

$$p_0 = (x, y, 0) \tag{B.33}$$

$$h_1: \quad a_1 x + b_1 y + c_1 0 + d_1 = 0 \tag{B.34a}$$
$$h_2: \quad a_2 x + b_2 y + c_2 0 + d_2 = 0 \tag{B.34b}$$

$$\begin{pmatrix} a_1 & b_1 \\ a_2 & b_2 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} -d_1 \\ -d_2 \end{pmatrix} \tag{B.35}$$

$$\begin{pmatrix} x \\ y \end{pmatrix} = \frac{1}{a_1 b_2 - a_2 b_1} \begin{pmatrix} b_2 & -b_1 \\ -a_2 & a_1 \end{pmatrix} \begin{pmatrix} -d_1 \\ -d_2 \end{pmatrix} \tag{B.36}$$

A plane h and a line l intersect at a point $p = (x, y, z)$.

$$l : p = p_0 + \lambda_0 \vec{v} \tag{B.37}$$

$$h : ax + by + cz + d = 0 \tag{B.38}$$

$$a(x_0 + \lambda_0 v_1) + b(y_0 + \lambda_0 v_2) + c(z_0 + \lambda_0 v_3) + d = 0 \tag{B.39}$$

$$\lambda_0 = \frac{-ax_0 - by_0 - cz_0 - d}{av_1 + bv_2 + cv_3} \tag{B.40a}$$
$$\lambda_0 = \frac{-ax_0 - by_0 - cz_0 - d}{\vec{n} \cdot \vec{v}} \tag{B.40b}$$

B.4 Projection

B.4.1 Point-Plane

p'_0 denotes the position on the plane h of an orthogonal projected point p_0.

$$p'_0 = p_0 + \lambda_0 \vec{n} \tag{B.41}$$

$$h : ax + by + cz + d = 0 \tag{B.42}$$

$$a(x_0 + \lambda_0 a) + b(y_0 + \lambda_0 b) + c(z_0 + \lambda_0 c) + d = 0 \tag{B.43}$$

$$\lambda_0 = \frac{-ax_0 - by_0 - cz_0 - d}{a^2 + b^2 + c^2} \tag{B.44}$$

B.5 Bisector

A point on the intersection of two planes (or lines in 2-dim.) is calculated. This point is part of the angle bisector. The normal vector $\vec{n}$ of the angle bisector is calculated as follows.

$$\vec{n} = \frac{\vec{n}_1}{|\vec{n}_1|} + \frac{\vec{n}_2}{|\vec{n}_2|} \tag{B.45}$$

The angle bisector is initialized with the intersection point and its normal vector.

B.6 Side & Orientation

The (signed) area of a triangle (p_1, p_2, p_3) is positive, if the points are ordered counter-clockwise.

$$A = \frac{1}{2} \begin{vmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ 1 & 1 & 1 \end{vmatrix} \tag{B.46}$$

The (signed) volume of a tetrahedron (p_1, p_2, p_3, p_4) is calculated as follows.

$$V = \frac{1}{6} \begin{vmatrix} x_1 & x_2 & x_3 & x_4 \\ y_1 & y_2 & y_3 & y_4 \\ z_1 & z_2 & z_3 & z_4 \\ 1 & 1 & 1 & 1 \end{vmatrix} \tag{B.47}$$

A point p is on the positive side of a plane h, iff $ax + by + cz + d > 0$. It is on the negative side, iff $ax + by + cz + d < 0$.

B.7 Number Representation

Rational numbers $\mathbb{Q}$ (fraction of two integers) are sufficient to have an exact representation of intersections and projections. $\mathbb{Q}$ is not sufficient for an exact representation of the bisector. The normal vectors can not be normalized without calculating the square root.

Bibliography

[AA96] Oswin Aichholzer and Franz Aurenhammer. Straight skeletons for general polygonal figures in the plane. In *Computing and Combinatorics*, volume 1090 of *Lecture Notes in Computer Science*, pages 117–126. Springer, 1996.

[AAAG95] Oswin Aichholzer, Franz Aurenhammer, David Alberts, and Bernd Gärtner. Straight skeletons of simple polygons. In *Proc. 4th Internat. Symp. of LIESMARS*, pages 114–124, 1995.

[AGSS87] Alok Aggarwal, Leonidas Guibas, James Saxe, and Peter Shor. A linear time algorithm for computing the voronoi diagram of a convex polygon. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, STOC '87, pages 39–45, New York, NY, USA, 1987. ACM.

[AM94] Pankaj K. Agarwal and Jiří Matoušek. On range searching with semi-algebraic sets. *Discrete & Computational Geometry*, 11:393–418, 1994.

[AW13a] Franz Aurenhammer and Gernot Walzl. Structure and computation of straight skeletons in 3-space. In *Proc. 24th International Symposium on Algorithms and Computation ISAAC'13*, volume 8283 of *Lecture Notes in Computer Science*, pages 44–54, Hong Kong, 2013. Springer Berlin Heidelberg.

[AW13b] Franz Aurenhammer and Gernot Walzl. Three-dimensional straight skeletons from bisector graphs. In *Proc. 5th International Conference on Analytic Number Theory and Spatial Tessellations*, Kiev, Ukraine, 2013.

[AW14] Franz Aurenhammer and Gernot Walzl. Polytope offsets and straight skeletons in 3D. In *Proc. 30th Annual Symposium on Computational Geometry*, pages 98–99, Kyoto, Japan, 2014.

[BEGV08] Gill Barequet, David Eppstein, Michael Goodrich, and Amir Vaxman. Straight skeletons of three-dimensional polyhedra. In *Algorithms - ESA*, volume 5193 of *Lecture Notes in Computer Science*. Springer, 2008.

[BHJ84] Kurt Brassel, Martin Heller, and Patrick Jones. The construction of bisector skeletons for polygonal networks. In *Proceedings of the 4th International Symposium on Spatial Data Handling*, volume 1, pages 117–128, 1984.

[Blu67] Harry Blum. A transformation for extracting new descriptors of shape. In *Models for the Perception of Speech and Visual Form*, pages 362–380. MIT Press, 1967.

[CEG+96] Bernard Chazelle, Herbert Edelsbrunner, Leonidas J. Guibas, Micha Sharir, and Jorge Stolfi. Lines in space: Combinatorics and algorithms. *Algorithmica*, 15(5):428–447, 1996.

[CGA] CGAL - Computational Geometry Algorithms Library. `http://www.cgal.org`. accessed in August 2014.

[CMV14] Siu-Wing Cheng, Liam Mencel, and Antoine Vigneron. A faster algorithm for computing straight skeletons. In *Algorithms - ESA 2014*, volume 8737 of *Lecture Notes in Computer Science*, pages 272–283. Springer Berlin Heidelberg, 2014.

[CSW95] Francis Chin, Jack Snoeyink, and Cao An Wang. Finding the medial axis of a simple polygon in linear time. In *Algorithms and Computations*, volume 1004 of *Lecture Notes in Computer Science*, pages 382–391. Springer, 1995.

[CV02] Siu-Wing Cheng and Antoine Vigneron. Motorcycle graphs and straight skeletons. In *Proceedings of the 13th annual ACM-SIAM Symposium on Discrete Algorithms*, pages 156–165, 2002.

[CV07] Siu-Wing Cheng and Antoine Vigneron. Motorcycle graphs and straight skeletons. *Algorithmica*, 47(2):159–182, 2007.

[DDL98] Erik D. Demaine, Martin L. Demaine, and Anna Lubiw. Folding and cutting paper. In *Post-Conf. Proc. of the Japan Conference on Discrete and Computational Geometry (JCDCG 1998)*, volume 1763 of *Lecture Notes in Computer Science*, pages 104–118. Springer, 1998.

[EE99] David Eppstein and Jeff Erickson. Raising roofs, crashing cycles, and playing pool: Applications of a data structure for finding pairwise interactions. *Discrete & Computational Geometry*, 22:569–592, 1999.

[Hav05] Sven Havemann. *Generative Mesh Modeling*. PhD thesis, TU Braunschweig, 2005.

[Hel94] Martin Held. On computing voronoi diagrams of convex polyhedra by means of wavefront propagation. In *CCCG*, pages 128–133, 1994.

[Her89] John Hershberger. Finding the upper envelope of n line segments in $O(n \log n)$ time. *Information Processing Letters*, 33(4):169–174, 1989.

[HS08] Jan-Henrik Haunert and Monika Sester. Area collapse and road centerlines based on straight skeletons. *Geoinformatica*, 12(2):169–191, 2008.

[Hub11] Stefan Huber. *Computing Straight Skeletons and Motorcycle Graphs: Theory and Practice*. PhD thesis, Faculty of Natural Sciences, University of Salzburg, June 2011.

[Mat93] Jiří Matoušek. Range searching with efficient hierarchical cuttings. *Discrete & Computational Geometry*, 10(1):157–182, 1993.

[MVPG11] Jonas Martinez, Marc Vigo, and Núria Pla-Garcia. Skeleton computation of orthogonal polyhedra. *Computer Graphics Forum*, 30(5):1573–1582, 2011.

[Mü16] Emil Müller. *Lehrbuch der darstellenden Geometrie für technische Hochschulen*. B. G. Teubner, 1916.

[Ree46] Georges Reeb. Sur les points singuliers d'une forme de pfaff completement intégrable ou d'une fonction numérique. *C. R. Acad. Sci. Paris*, 222:847–849, 1946.

[STG+97] Duane W. Storti, George M. Turkiyyah, Mark A. Ganter, Chek T. Lim, and Derek M. Stal. Skeleton-based modeling operations on solids. In *Proceedings of the Fourth ACM Symposium on Solid Modeling and Applications*, 1997.

[vP77] Gustav Ad. v. Peschka. *Kotirte Ebenen*. Buschak & Irrgang, Brünn, 1877.

[VY13] Antoine Vigneron and Lie Yan. A faster algorithm for computing motorcycle graphs. In *Proceedings of the 29th annual Symposuim on Computational Geometry*, pages 17–26. ACM, 2013.

[WS88] Ady Wiernik and Micha Sharir. Planar realizations of nonlinear davenport-schinzel sequences by segments. *Discrete & Computational Geometry*, 3(1):15–47, 1988.