G2-Continuous Approximation of Open and Closed Curves

Masterarbeit

zur Erlangung des akademischen Grades Diplom-Ingenieur
an der Naturwissenschaftlichen Fakultät
der Paris-Lodron-Universität Salzburg

Eingereicht von Dominik S. Kaaser

Gutachter: Ao.Univ.-Prof. Dipl.-Ing. Dr. Martin Held
Fachbereich Computerwissenschaften

Salzburg, Juli 2012

G2-Continuous Approximation of Open and Closed Curves

Master's Thesis

Dominik S. Kaaser

July 2012

Department of Computer Sciences
University of Salzburg
Jakob Haringer Str. 2
5020 Salzburg
Austria

Dominik S. Kaaser
E-Mail: dominik@kaaser.at
Web: http://kaaser.at

ABSTRACT

In this thesis we present a new algorithm to compute an approximation of multiply-connected open polygonal chains and polygons using G^2 continuous primitives. Our algorithm follows a divide and conquer approach to successively refine the approximation consisting of uniform cubic B-spline as approximation primitives. The approximation is confined to a tolerance zone which is constructed using the Voronoi diagram of the input. Furthermore, the approximation is based on a set of approximation nodes placed within the tolerance zone.

An introduction to differential geometry is given in order to identify basic properties of curves including their class of continuity. We also discuss primitives suitable to yield G^2 continuity, such as Bézier curves and cubic splines. Bernstein polynomials are introduced as basis of Bézier curves and various properties including de Casteljau's algorithm for Bézier curve evaluation are stated. Furthermore, we review a definition of uniform cubic B-splines as the basis of our approximation. With de Boor's algorithm as generalization of the de Casteljau algorithm, a numerically stable method for evaluation and refinement of B-spline curves is given.

Our approximation resides within a tolerance zone assembling the set of points that do not exceed a given tolerance threshold with respect to a specific metric. The Voronoi diagram is defined and used as a utility data structure in order to give a definition of the input's tolerance zone. Based on the Voronoi diagram, we describe an algorithm to compute the tolerance zone boundary. This boundary is related to but not identical with the traditional offset curves as defined in the literature.

A set of so-called approximation nodes serves as the base of our approximation primitives. These A-NODES are placed on the medial axis of the tolerance zone which is a subset of the Voronoi diagram for polygonal input. A definition of the medial axis is given along with a description of our method to find suitable A-NODES on it. The advantages and disadvantages of this approach are discussed.

To check whether a primitive lies within the tolerance zone, we deal with various types of intersection tests. Some of these methods require the computation of the primitive's convex hull. Thus, an introduction to convex hulls is given in order to describe an algorithm for intersection testing between approximation primitives and the tolerance zone boundary.

An analysis of the runtime complexity and memory consumption is provided, showing that our algorithm runs in $O(n \log n)$ time and has

a linear memory footprint. These claims are supported by empirical tests of our implementation running on a large variety of inputs with a defined automatic setup of tolerances.

Finally, having compared our divide and conquer algorithm to previously published greedy methods, we conclude that our new approach is both, faster and more efficient in terms of smaller output size when used for approximation with spline curves than the methods published in literature.

ACKNOWLEDGEMENT

It is my pleasure to take this opportunity to express my gratitude to all those people that made it possible for me to write this thesis.

First of all, I would like to thank my advisor, Dr. Martin Held, for his support and guidance throughout the whole time I had been implementing the algorithm and writing this thesis. Whenever I encountered difficulties with this subject, his suggestions in many of our discussions helped me to carry on. Without his invaluable advice when proof-reading my drafts and pointing out shallows where I could have run aground, this thesis would not have become what it is now.

I also want to thank my parents, Heidemaria and Wilfried, for their unlimited and unconditional support throughout my entire educational career. My deep and sincere gratitude goes to my girlfriend Linda for her understanding support, her patience and her encouragement whenever I faced problems regarding this thesis.

Finally, I am grateful to my colleagues Peter and Stefan from our work group for many hours of interesting discussions that helped me to keep my work on track. They provided invaluable assistance and handy tools that I really appreciated when creating this thesis.

CONTENTS

1 INTRODUCTION

1.1 ABOUT THIS THESIS

In this thesis we describe an algorithm consisting of several sub-algorithms capable of approximating geometric input data such as multiply-connected simple planar polygons or polygonal chains by G^2 continuous primitives. We do not discuss more complex input primitives such as *biarcs*, *splines*, or *Bézier curves* or input data of higher dimension, e. g., curves in 3D. Many of these more sophisticated input primitives can be sampled, generating a pure polygonal representation which can be used as input to our algorithm.

For the input, we compute the approximation consisting of one or several approximation primitives. This approximation and thus all resulting approximation primitives are subject to one major restriction: The distance in terms of the Hausdorff distance between the approximation and the input must not exceed a user-defined maximum approximation tolerance. Thus, one major part of our work focuses on the generation of a so-called *tolerance zone*. Based on this user-defined approximation tolerance, the boundary of the tolerance zone is computed using a special decomposition of the underlying space allowing a fast and robust generation of the according tolerance zone boundaries.

The resulting approximation primitives are defined upon at least two approximation nodes per primitive. These approximation nodes are vertices generated within the tolerance zone and used, e. g., as start- and endpoint of the according approximation primitive. Subsequently during the process of finding an approximation, primitives defined upon different sets of approximation nodes are tested and the best primitive is selected.

By choosing appropriate approximation primitives, we can achieve different desirable properties of the entire approximation. One of the most common requirements for many practical applications is tangent continuity. In this thesis we discuss various primitives suitable to compute the even stronger C^2 continuous approximations, i. e., approximations with continuous second derivatives. Such approximations are used within many applications, e. g., in the field of tool path generation. For example in the field of rapid prototyping and CNC milling, at least C^1-curves are required, as abrupt moves may either damage the machine head and the material or cause severe processing speed penalties.

As an additional feature, the tolerance zone does not need to be symmetric. It might very well occur that the distance on one side is required to be larger than on the other side. Even a *negative* tolerance can be specified, resulting in a tolerance zone that is disconnected from the original input polygon. Such types of tolerance zones are of great importance in practical applications, for example when the outline of a piece of material is specified. The resulting tool path is required to be completely contained within the material, possibly with some additional distance to the material's border. This guarantees that, for example, a cutting machine's head does not leave the workpiece.

1.2 BASIC DEFINITIONS AND NOTATIONS

Many different geometric objects will occur throughout this thesis. Among these probably most common is the point or vertex in the Euclidean space, denoted with an upper case Latin letter, e. g., P, $P(p_x, p_y)$. Any point can be defined by means of a position vector, starting in the origin $\mathcal{O}$ and pointing at the specific point, e. g.,

$$p = \overrightarrow{\mathcal{O}P} = \begin{pmatrix} p_x \\ p_y \end{pmatrix} \quad .$$

In literature, these two terms get mixed up frequently and it can often be seen that basic mathematical operations are performed directly on points, rather than on the position vectors implicitly defined by the corresponding points.

We will denote the components of a vector $x \in \mathbb{R}^n$ as $x_1, x_2, \ldots, x_n$ where $\mathbb{R}^n$ denotes the n-dimensional Euclidean vector space. For a vector $a \in \mathbb{R}^2$ we also write a_x and a_y for the x- and y-coordinates. For a given vector $p \in \mathbb{R}^2$ let $\|p\|$ denote the norm of that vector, i. e., the Euclidean distance $\|p\| = d(\mathcal{O}, P) = \sqrt{p_x^2 + p_y^2}$ from the origin $\mathcal{O}$ to the tip of the vector, the point P.

The input $\mathcal{P}$ for our algorithm consists of multiply-connected simple planar polygons or polygonal chains, i. e., a number of consecutively connected line segments in a plane. As the input is multiply-connected, we do not accept one, but a set of intersection free simple open or closed polygonal chains as input. We define a polygonal chain as a curve defined by a sequence of vertices with a line segment connection between every two consecutive vertices. A closed polygonal chain has a connecting line segment between the first and the last vertex, alternatively we say the first and the last vertex coincide. A simple polygonal chain does not have any intersections except for successive (or the first and the last) line segments in the enclosed nodes. All defining vertices are required to lie within the Euclidean plane $\mathbb{R}^2$, sometimes in literature also denoted by $\mathbb{E}^2$.

We assume our input to be simple, i. e., all input polygonal chains are simple and pairwise disjoint. A definition of the term *simple* in the context of (parametric) curves is introduced when defining curves in Section 3.4.2.

There do exist algorithms to check for simplicity of a given input polygon based on the triangulation algorithm by Chazelle [Cha91] that run in $O(n)$ time. Unfortunately, there do not exist any known implementations of Chazelle's triangulation algorithm and according to Steven Skiena [Ski08]

> [this] algorithm is sufficiently hopeless to implement that it qualifies more as an existence proof [Ski08, p. 575].

Thus we resort to another algorithm to perform simplicity checks, the Shamos-Hoey algorithm [SH76] which runs in $O(n \log n)$ time. Additionally, one can use the Bentley-Ottmann algorithm [BO79] to modify a complex polygon and extract a set of simple polygons. The Bentley-Ottman algorithm runs in $O((n + k) \log n)$ time with n the total number of vertices and k the number of intersections found.

In addition to the previous definition, we also use the term *polygon* to describe a closed polygonal chain. In opposition to simple open polygonal chains, simple polygons as simple closed polygonal chains do have important additional properties: Most notably, simple polygons have an interior and an exterior. As simple polygons can be classified as Jordan curves, this is an implication from the Jordan curve theorem which states that any simple closed curve in the plane, i. e., a closed curve without self-intersections, divides the plane into two regions. For this theorem, many different proofs exist, e. g., [Bro25].

With the definition of an interior and an exterior, the problem of determining whether a given point lies within the polygonal chain arises. A finite procedure to determine, whether a point lies within a Jordan curve, is described in [BJMR75]. For simple polygons, this problem can be solved by conducting a point-in-polygon test. Many solutions and algorithms for this problem can be found in the literature, e. g., the ray-casting algorithm [SSS74]. Nevertheless, as we explicitly allow open polygonal chains as input, we must not rely on interior- and exterior properties and thus for our thesis point-in-polygon-tests are of little importance.

In some chapters, most notably in Chapter 6, we deal with sets of points or functions at a given level or degree n. Often these functions are recursively defined or require recursive data structures consisting of sets of points at a given level n. We use the convention that for a function $f_{n,k}(t)$ or an intermediate point $p_{n,k}$ the first index n denotes the level and the second index k denotes the index of the function or point at level n. Usually we can assume the first level at $n = 0$ to coincide with the input points p_i, i. e., $p_i = p_{0,i}$. Sometimes the additional notations $b_k^n(t)$ or p_k^n can be observed in the literature.

1.3 OUTLINE

Following the same scheme as Heimlich and Held, [Hei06], [HH08a], our system consists of the following sub-algorithms:

1. PREPROCESSING: The input data is read and subjected to a unit-square-transformation, i. e., all input primitives are scaled to fit into the unit square $[0, 1]^2$. Furthermore, degenerate vertices are removed, such as duplicate vertices, defining zero length edges, and inner co-linear-points, i. e., vertices p_j with an angle at p_j defined by the neighboring vertices p_{j-1}, p_{j+1} of 180 degrees.

2. TOLERANCE ZONE: We compute the Voronoi diagram of the input and extract a tolerance zone boundary. This tolerance zone boundary consists of straight line segments and separates the tolerance band from the outside. Thus, any approximation primitive starting within the tolerance zone and intersecting the tolerance zone boundary must be discarded, as it violates the maximum approximation tolerance.

3. APPROXIMATION NODES: We compute the inner Voronoi diagram of the tolerance zone boundaries and use sampled vertices on the medial axis as initial approximation nodes.

4. APPROXIMATION ALGORITHMS: The actual approximation is carried out by a greedy algorithm that attaches the longest possible valid approximation segment to the output. Different algorithms are presented in the work by Held and Eibl [HE05]. Additionally, we present a top-down approach implementing a divide and conquer scheme that can be used with approximation primitives based on more than two approximation nodes.

2

PRIOR AND RELATED WORK

2.1 POLYNOMIAL INTERPOLATION

The problem of approximating an input polygonal chain is closely related to the problem of finding a curve that *best fits* a series of data points. Such a curve may possibly be subject to given constraints and can be used to either interpolate the points, i. e., the data points lie on the curve, or smooth the input with respect to a given error criterion.

Many different error metrics have been investigated, such as *least absolute deviation* [Kar58] or more commonly *least squares*. The latter was first developed by Legendre [BM89, p. 487] and sometimes is credited to Gauss. For details on the historical events leading to the discovery of the least squares method see [Pla72].

Fitting a sufficiently smooth curve or a surface through a set of data points is a common problem with many references in the literature. Such a curve fitting problem is specified in [Gal99, p. 9] as follows:

> Given $n+1$ data points $p_0, \ldots, p_n$ and a sequence of $n+1$ reals $t_0, \ldots, t_n$, with $t_i < t_{i+1}$ for all i, $0 \leq i \leq n-1$, find a [...] curve $\gamma : [t_0, t_1] \to \mathbb{R}^2$, such that $\gamma(t_i) = p_i$, for all i, $0 \leq i \leq n$.

There are many different types of curves that are suitable to solve this problem. As an obvious attempt to approximate the input polygonal chain, we could choose (equidistant) knots in the curve's parametric domain to compute the set of data points and then compute the polynomial interpolation. The simplest method on these generated data points is the linear interpolation, which connects the data points with straight line segments. Clearly, this method is not suitable to construct curves of higher class of continuity, and thus one would probably generalize it to use polynomials as interpolating functions.

As a very important theorem, it can be shown that there exists a unique interpolating polynomial of minimal degree. This *unisolvence theorem* implicates that all methods of computing a polynomial interpolation yield the same result. Nevertheless, the resulting polynomial interpolations can be in different forms. As a choice discussed widely in literature, we resort to *Lagrange polynomials*. For a given set of points in $\mathbb{R}^2$, the Lagrange polynomial is the polynomial in Lagrange form of minimal degree which passes through all of the data points.

The Lagrange method, although published in 1795 by Joseph-Louis Lagrange, was actually discovered by Edward Waring in 1779 and rediscovered in 1783 by Leonard Euler. For this reason, it is sometimes

referred to as Waring-Lagrange-interpolation. The interpolation is computed for a given set of $n+1$ data points $p_0, \ldots, p_n$ with respect to the $n+1$ discrete knots $t_0, \ldots, t_n$ with the mapping $\lambda(t_i) = p_i$ as

$$L_n(x) = \sum_{i=0}^{n} \lambda(t_i)\, l_{n,i}(x) \quad .$$

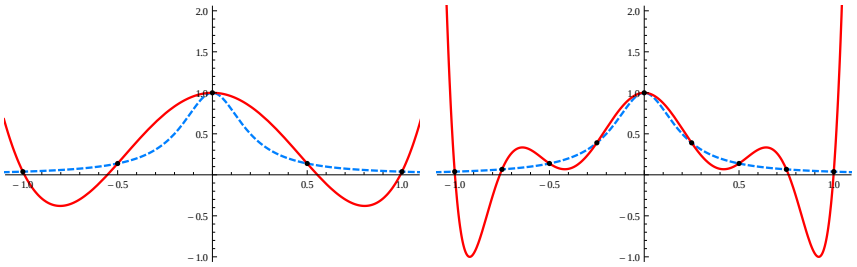
The *basis functions* $l_{n,i}$ are defined as

$$l_{n,i}(x) = \prod_{k=0, k\neq i}^{n} \frac{x - t_k}{t_i - t_k} \quad .$$

It can be observed that the basis functions $l_{n,i}(x)$ form polynomials of degree n and satisfy the condition

$$l_{n,i}(t_k) = \begin{cases} 1, & i = k \quad , \\ 0, & i \neq k \quad . \end{cases}$$

The resulting polynomial in Lagrange form interpolates the given data points p_i. The coefficients of the polynomial in power basis can be computed by evaluating the resulting *Vandermonde* matrix [Shm07]. Unfortunately, this matrix tends to be ill-conditioned.



(a) 5 knots (b) 9 knots

Figure 1: Runge's phenomenon: The interpolation polynomial oscillates at the edges of the interval $[-1, 1]$. The function to be approximated at the uniformly distributed approximation nodes is the Runge function

$$f_5(x) = \frac{1}{1 + (5x)^2} \quad .$$

Another even more problematic disadvantage of polynomial interpolation that can be observed especially when using polynomials of higher degrees is Runge's phenomenon. Named after Carl David Tolmé Runge, this phenomenon describes the oscillation that occurs when the degree of the interpolating polynomials is high [Run01].

The effect can be compared to the Gibbs phenomenon for sine and cosine as basis functions, see Section 6.1.1. Its main implication is that

with higher degree the quality of the interpolation does not necessarily increase. An example of an interpolation of the Runge function can be seen in Figure 1.

We can conclude that Polynomials of especially higher degree are unsatisfactory in a number of ways. The main problem is that polynomials of high degrees tend to oscillate at the interval borders and thus are not suitable for approximation within a maximum approximation tolerance. Furthermore, the condition numbers of the Vandermonde matrices used to compute the coefficients of interpolation polynomials may be large [Gau74] and thus the algorithm may suffer from serious numerical problems. To remedy these shortcomings, different approaches can be followed:

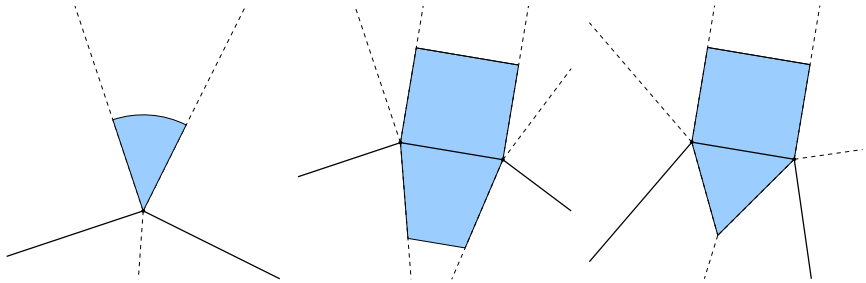
- NON-UNIFORM POINT DISTRIBUTIONS: Uniform point distributions tend to be prone to the Runge phenomenon. Furthermore, the resulting polynomial is ill-conditioned, i. e., minor changes in the input points cause huge changes in the interpolant. As a possible solution to minimize the oscillation the interpolation knots can be distributed more densely at the edges of the interval. An example of such a distribution is the *Chebyshev distribution* [BT04].

- PIECEWISE POLYNOMIALS: Due to their impact in the field of computer aided geometric design, we resort to piecewise polynomial functions. Many of the problems associated with polynomials of higher degree can be avoided by using so-called *spline curves*. These *splines* are compounds of polynomial curves at a given (low) degree and possibly cause some additional work to maintain continuity properties at the joints. For further details see Chapter 6.

2.2 TOLERANCE BAND GENERATION

Held and Eibl [Eib02], [HE05] presented an algorithm for computing a tolerance band which consists of so-called *T-parts*. These T-parts are trapezoids, triangles, disc-sectors and ring-sectors, attached to the input. The tolerance band is computed as the result of cropping and shrinking all these T-parts until they are free of intersections.

The weakness of this tolerance band generation algorithm is its vulnerability against noisy input data [Hei06]. At regions of high noise, the tolerance band becomes unnecessarily tight. As a result, the approximation curve is forced to preserve many small details and thus the number of approximation primitives required increases. Additionally, a large number of tolerance band primitives located at a single region of noisy input data causes problems with the use of hashing techniques. A sample of such a noisy input data problem can be seen in Figure 3.



(a) disc sector (b) trapezoid (c) triangle

Figure 2: The T-part approach as proposed by Held and Eibl builds a tolerance zone based on T-part primitives, disc-sectors, trapezoids and triangles.

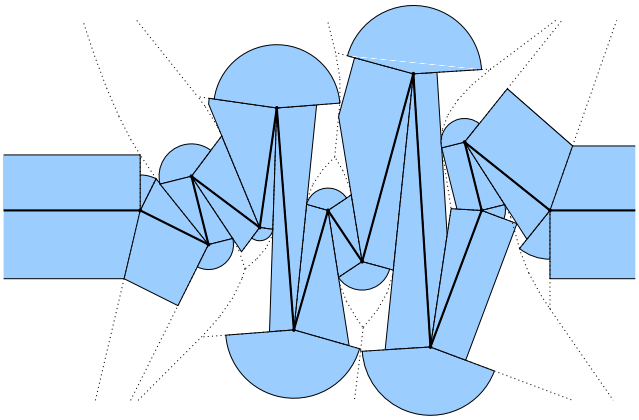


Figure 3: The T-part approach is suboptimal when applied to noisy input data, as the actual tolerance zone generated may become unnecessarily tight.

Heimlich and Held implemented a second approach to overcome these issues [HHo8a]. They use the Voronoi diagram of the input data and compute a conventional offset, which consists of zero to several offset curves. Based on the conventional offset, a so-called *pseudo offset curve* is generated. This curve, in the implementation of Heimlich consisting of points, straight line segments and circular arcs, forms the border of the tolerance zone and is used for intersection tests to confine the approximation to the tolerance zone.

The approach by Heimlich and Held is focused on closed polygonal chains, i. e., polygons. This makes it possible to define a signed distance $d_s(P, q)$ between a point q and the input polygons P as follows, using the minimum Euclidean distance $d(P, q)$ between a polygon and a point. The following definition of d_s makes it necessary to use concepts of interior and exterior of a polygon.

For two simple polygons without intersections it is clear that either both polygons lie in the exterior of each other (they are not *nested*) or one polygon P_1 lies in the interior of P_2, the other. If the first polygon lies in the interior of the other, we can say P_1 is *nested* within the other

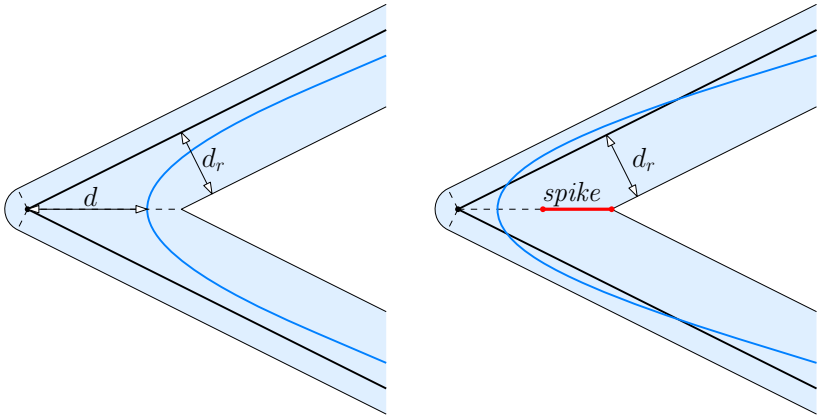
polygon P_2, which leads us to the definition of the *nesting level* $nl(P)$ of a polygon P as described by Heimlich and Held: We say $nl(P)$ is the number of polygons of $\mathcal{P}$ that have P in their interior [HHo8a].

$$d_s(P,q) := \begin{cases} d(P,q) & \text{if } q \in int(P) \text{ and } nl(P)\text{is even, or} \\ & \text{if } q \in ext(P) \text{ and } nl(P)\text{is odd,} \\ 0 & \text{if } q \in P, \\ -d(P,q) & \text{if } q \in ext(P) \text{ and } nl(P)\text{is even, or} \\ & \text{if } q \in int(P) \text{ and } nl(P)\text{is odd.} \end{cases}$$

This yields the definition of a tolerance zone $\mathcal{TZ}(P, d_l, d_r)$ around P with given left and right tolerance d_l, d_r as the set of points $x \in \mathbb{R}^2$ which have a signed distance $d_s(P,x)$ between d_l and d_r:

$$\mathcal{TZ}(P, d_l, d_r) := \{x \in \mathbb{R}^2 : d_l \geq d_s(P,x) \geq d_r\}$$

This work by Heimlich and Held also includes a functionality to add so-called *spikes* to the offset curve.



(a) Without the spike the maximum of the (minimal) distances d between the input and the approximation can become larger than our approximation tolerance d_r.

(b) The spike, as a subset of the Voronoi diagram, ensures that the Hausdorff distance between the input and the approximation does not exceed the tolerance.

Figure 4: Spikes are added to prevent *shortcuts* of the approximation.

A spike is a subset of the Voronoi diagram which ensures that the distance from any point of the input to the approximation does not exceed the maximum distance, i. e., that the Hausdorff distance between the input and the approximation does not exceed the approximation tolerance.

To generate these spikes we focus on only those parts of the Voronoi diagram that do not coincide with our tolerance zone boundary and

that are closer to the input than our requested approximation tolerance. All resulting sets of Voronoi edges have in common that one vertex v lies exactly on the tolerance zone boundary. Basically, we deal with trees rooted in v that consist of nodes and edges of the Voronoi diagram.

We start with one vertex v on the tolerance zone boundary and depict the set of nodes of the original polygonal input chain that can be reached as leaves of the tree T_v rooted in v as S_v. S_v can be easily obtained by in-order traversing T_v. The set of nodes S_v is stored in a list structure, e. g., a queue. We start in a top-down approach and recursively apply a series of steps, keeping track of the current end point of the spike. This can be achieved in linear time [HH08a].

The generated spikes are connected to the tolerance zone boundaries and require the approximation to converge to the input such that the input $\mathcal{P}$ is within the tolerance zone of the approximation $\mathcal{A}$, i. e., $\mathcal{P} \in \mathcal{TZ}(\mathcal{A}, d_l, d_r)$. A sample spike and the impact on the approximation can be seen in Figure 4.

2.3 APPROXIMATION ALGORITHMS

A large number of algorithms can be found in the literature that compute a polygonal approximation of (polygonal) curves by line segments. These algorithms are focused on two optimization problems and can thus be classified into two groups.

One class deals with the problem of finding the minimum number of approximation primitives that is sufficient to approximate an input within a given maximum tolerance ϵ. The other class deals with finding an approximation of minimal approximation error consisting of a fixed number of primitives.

Our approach, as an extension of the prior work by Held, Eibl and Heimlich [HE05], [HH08a], focuses on the first class of problems. The main objective is to find a suitable approximation, i. e., an approximation that does not exceed a predefined tolerance zone, with a minimal number of primitives. Nevertheless, our work lies within the field of greedy algorithms, and thus we do not claim and do not present an optimal algorithm in this case.

Drysdale et al. presented an algorithm in 2006 [DRS06] which returns the minimum number of circular arcs within a given tolerance zone required to approximate the input based on a given set of approximation nodes which form the start and end vertices for approximation primitives found. The algorithm, as later published in 2008 [DRS08] runs in $O(n^2 \log n)$ optimal time. As its asymptotic runtime complexity has been shown to be optimal, this can be seen as a theoretical bound for comparison of other algorithms following greedy approaches. Nevertheless, their result depends on the proper placement of approximation nodes.

A more recent result can be obtained from the work by Georg Maier, independent of the choice of approximation nodes. In his thesis, he presents an algorithm that computes a *smooth minimum arc path*, i. e., a path consisting of circular arcs from a start point to an end point within a given polygon as tolerance zone. As a result, the path consisting of the minimum possible number of arcs is constructed, running in $O(n^2)$ asymptotic runtime complexity, with n the number of vertices given [Mai10].

Further references can be obtained from [BR69], [Aki70], [Ram72], [DRS06], [BM99] and [CC96].

3 MATHEMATICAL ASPECTS

In the following chapter we describe some of the basic mathematical aspects and mathematical concepts used throughout this thesis and required for understanding our algorithm. We start by defining the space in which our input lies and give a short introduction to *Voronoi diagrams* which are utilized throughout the computation of our approximation.

We also define and deal with basic properties of curves, as both, our input and our output can be seen as curves in the space. One main goal of this thesis is to present a way of generating a G^2 continuous approximation which is defined in the following chapter as well.

Finally, in this chapter we discuss basic properties of suitable G^2 continuous curves to achieve an appropriate approximation. Among these possible approximation primitives we will focus on *Bézier curves* and *cubic splines* which are discussed later on in Chapter 6.

3.1 METRIC SPACE

A *metric space*, e. g. $\mathbb{R}^2$ with the Euclidean distance, is an ordered pair (M, d) of a non-empty set M and a function $d : M \times M \to \mathbb{R}$ which defines a metric on M such that for any $x, y, z \in M$ the following properties hold:

1. $d(x, y) \geq 0$
2. $d(x, y) = 0 \Leftrightarrow x = y$
3. $d(x, y) = d(y, x)$
4. $d(x, z) \leq d(x, y) + d(y, z)$

3.1.1 *Euclidean Space*

Probably the best known example of such a metric space is the Euclidean space, i. e., the Euclidean plane or the three dimensional space used in Euclidean geometry. As one of the chief contributions to mathematics, Euclid of Alexandria defined the Euclidean geometry based on five axioms [Malo7, 2011-10-05]. These postulates can be used to deduct all other rules and theorems [Heao8].

Nowadays the Euclidean space is usually defined upon Cartesian coordinates following the ideas of analytic geometry. These coordinates, although previously used by the Greek astronomer Hipparchos and later discussed in the 14th century as latitudes and longitudes in

the work of Oresme, were named after René Descartes for his tremendous impact in the field of analytic geometry.

Independently, Pierre de Fermat was working in the same field, searching for the same solutions as Descartes even more systematically. He never published his research until 1679, whereas Descartes already boasted out his discoveries in 1637 [Sti10, p. 111].

Using the Euclidean distance, the space defined upon Cartesian coordinates indeed forms a metric space. Furthermore, the definition based upon Cartesian coordinates allows the extension of the dimension of the space up to an arbitrary n-dimensional space, $\mathbb{R}^n$.

Although most of the following concepts can be extended to any metric space with an appropriately defined distance function, we focus on $\mathbb{R}^2$. This is simply a consequence of the fact that our input points used to define the input curve are points of the Euclidean plane and thus lie in $\mathbb{R}^2$.

The Euclidean distance used throughout this thesis as distance function in the space is the first choice as it matches the distance between two points as measured using a ruler. There might be settings in which another metric might be a better fit to constitute a model of the actual situation, such as the *Manhattan distance* induced by the *L1 norm*. This metric represents the shortest path a car could take in a rectilinear grid street layout. See [Mal07, 2011-10-05] for additional details on the *taxicab plane*.

The Euclidean distance (for the two-dimensional case) as described above can be computed using the Pythagorean theorem and thus is defined as follows:

$$d : \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R}$$
$$d\,(x,y) := \left\| \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} - \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} \right\| = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}$$

We take a closer look at the four requirements which we demand from a distance function between two points. It is easy to see that the image of a distance function consists of non-negative reals only, i. e. $d(x,y) \geq 0$ can be expressed by means of the other propositions. Consider the triangle inequality (4) for two points $x, y \in M$, starting in x, moving to y and back to x:

$$d(x,x) \leq d(x,y) + d(y,x) \ .$$

When we use the symmetry property (3) $d(x,y) = d(y,x)$, we get

$$d(x,x) \leq 2 * d(x,y) \ .$$

Finally, using the identity property (2) of the metric which states that $d(x,y) = 0$ if and only if $x = y$, we can conclude that

$$0 \leq d(x,y) \ .$$

This does not mean, the first requirement is obsolete. It is rather a consequence of the fact that there do exist spaces which do not necessarily obey the triangle inequality whereas the other three propositions still are in place.

A prominent example of such a space is the space used in *hyperbolic geometry* which

> was created in the first half of the nineteenth century in the midst of attempts to understand Euclid's axiomatic basis for geometry. It [...] discards one of Euclid's axioms [CFKP97, p. 59].

This means hyperbolic geometry replaces the fifth Euclidean postulate: The Euclidean postulates formed the foundation of the ancient Greek's geometry and were accepted as unquestionable truths until the advent of *non-Euclidean geometry* [Kli90, p. 59], [Cox98].

In contemporary language following the paraphrase by Cannon et al. [CFKP97, p. 60] Euclid postulated the following:

1. Each pair of points can be joined by one and only one straight line segment.
2. Any straight line segment can be indefinitely extended in either direction.
3. There is exactly one circle of any given radius with any given center.
4. All right angles are congruent to one another.
5. If a straight line falling on two straight lines makes the interior angles on the same side less than two right angles, the two straight lines, if extended indefinitely, meet on that side on which the angles are less than two right angles .

The fifth postulate can be shown to be equivalent to the *parallel postulate*:

> 5'. Given a line and a point not on it, there is exactly one line going through the given point that is parallel to the given line [CFKP97, p. 60].

All efforts to express the fifth postulate by means of the other four postulates failed [Kli90, p. 60], but led to the discovery of non-Euclidean geometry. By replacing Euclid's fifth postulate with the parallel postulate also known as Playfair's axiom, we get the following differences between hyperbolic and elliptic geometry for a given line l and a point $x \notin l$:

- In EUCLIDEAN geometry, there is exactly one line through x that does not intersect l,

- in HYPERBOLIC geometry, there are infinitely many lines through x not intersecting l and
- in ELLIPTIC geometry, any line through x intersects l.

For more information on non-Euclidean geometry, especially on hyperbolic geometry, please refer to [CFKP97].

For additional details on metric spaces please refer to a textbook on *metric geometry* and *topology*, e. g., [BBI01] or [Mun99]. For details on the *thirteen books of Euclid*, the *Elements*, refer to the translations by Sir Thomas Little Heath [Hea08] who was an expert on the field of Greek mathematics history. Heath's second edition of his translation of Euclid, published in 1926, can be seen as the standard English version of the text [OR03, 2011-10-05]. For a German translation, see for example [Euk03].

3.2 VORONOI DIAGRAM

The Voronoi diagram is a decomposition of a (metric) space according to a given set of input sites. It is determined by the distance function and assigns a region of the space to every input site such that the distance between the points of a given region and the corresponding input site is less than the distance of the points to any other sites with respect to the metric defined on the space.

3.2.1 *Point Voronoi Diagram*

We are given a set $S := \{p_1, p_2, \ldots, p_n\}$ of n distinct points in a metric space. For practical reasons as described above we use the Euclidean plane $\mathbb{R}^2$. We define the Voronoi region, $\mathcal{VR}(p_i)$, also known as Voronoi cell, of a point $p_i \in S$ to be the union of all points of $\mathbb{R}^2$ whose distance to p_i is shorter than the distance to any other $p_j \in S$ for $j \neq i$.

A Voronoi polygon forms the boundary of a Voronoi region. Thus all points of the Voronoi polygon of p_i lie on a bisector of two points $p_i, p_j \in \mathbb{R}^2$ with $i \neq j$. Such a bisector $b(p_i, p_j)$ is the set of points of $\mathbb{R}^2$ that are equidistant to the defining vertices p_i and p_j, i. e.,

$$\forall x \in b(p_i, p_j) \quad d(x, p_i) = d(x, p_j) \ .$$

The union of all these Voronoi polygons defines the Voronoi diagram $\mathcal{VD}(S)$ of the set of points S [Hel10]. So this means, the Voronoi diagram is a decomposition of space into regions which are closer to one input point than to another with respect to a given metric, e. g., the Euclidean metric.

$$\begin{aligned}
\mathcal{VR}(p_i, S) &:= \{x \in \mathbb{R}^2 : \forall j \neq i : d(p_i, x) < d(p_j, x)\} \\
\mathcal{VP}(p_i, S) &:= \{x \in \mathbb{R}^2 : d(p_i, x) = \min_{j \neq i} d(p_j, x)\} \\
b(p_i, p_j) &:= \{x \in \mathbb{R}^2 : d(p_i, x) = d(p_j, x)\} \\
\mathcal{VD}(S) &:= \bigcup_{1 \leq i \leq n} \mathcal{VP}(p_i)
\end{aligned}$$

Additionally, one could describe the Voronoi cell $\mathcal{VR}(p_i, S)$ of a point $p_i \in S$ by examining intersections of half planes. In the Euclidean space, the bisector as defined above between the given vertex p_i and a distinct second point $p_j \in S$ with $i \neq j$ forms a straight line. Along this line the Euclidean space can be divided into two half planes. The half plane which contains p_j can be defined as

$$\mathcal{HP}(p_i, p_j) := \left\{x \in \mathbb{R}^2 : d(x, p_j) < d(x, p_i)\right\} \ ,$$

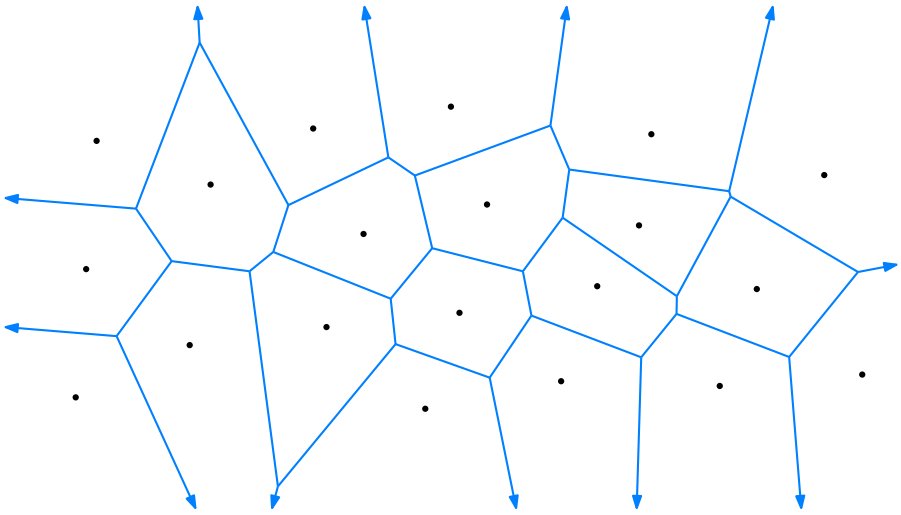


Figure 5: Point Voronoi diagram of a set of points

which is the set of all points closer to p_j than to p_i. Intersecting all half planes defined by p_i and any $p_j \in S$ with $i \neq j$ defines the Voronoi cell of the point p_i.

$$\mathcal{VR}(p_i) := \bigcap_{p_j \in S, i \neq j} \{x \in \mathbb{R}^2 : d(x, p_j) < d(x, p_i)\}$$

The Voronoi diagram can be seen as the set of all points without the union of all Voronoi regions.

$$\mathcal{VD}(S) := \mathbb{R} \setminus \bigcup_{p \in S} \mathcal{VR}(p, S)$$

The first worst-case optimal algorithm for computing Voronoi diagrams of sets of points in the plane was published by Shamos and Hoey [SH75]. Figure 5 shows an example of a point Voronoi diagram created using VRONI [Hel01]; it consists of straight line segments and rays forming the Voronoi edges.

3.2.2 *Complexity Analysis*

The dual graph of the Voronoi diagram of the set of points S is the Delaunay triangulation, named after Boris Nikolaevich Delaunay for his work on this topic from 1934 [Del34]. This dual graph indeed forms a triangulation of S, for a proof see, e. g., Preparata & Shamos [PS85].

The nodes of the Delaunay triangulation, given by the points of S, are connected if and only if the points share a common Voronoi edge. A Delaunay triangulation is given if no circumcircle of any triangle of the triangulation does contain any other point of S. This results in a

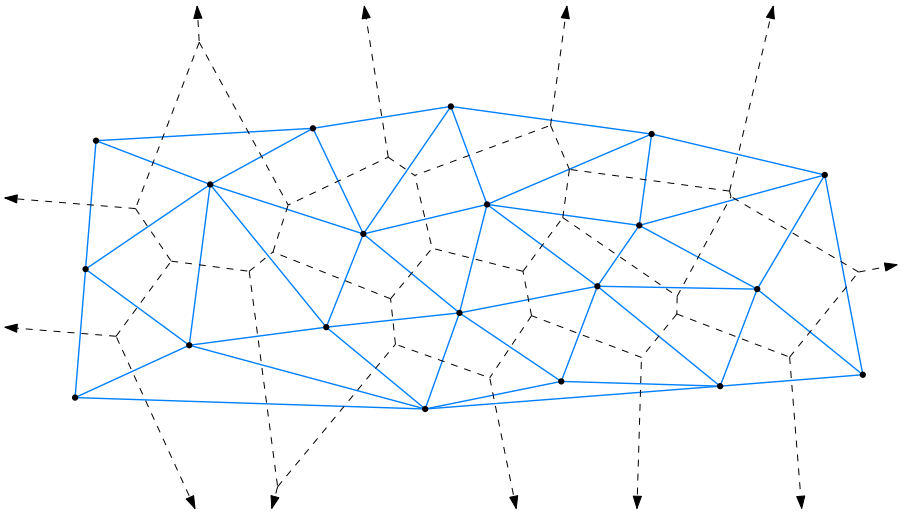


Figure 6: The Delaunay triangulation of the input as the dual graph to the point Voronoi diagram. The nodes are connected if and only if they share a common Voronoi edge.

maximized minimal internal angle of the triangulation. A Delaunay triangulation as the dual graph of the Voronoi diagram of the set of points used in Figure 5 can be seen in Figure 6.

When we apply Euler's formula $V - E + F = 2$ to this triangulation of n nodes, we can conclude that the number of edges and the number of nodes both are in $O(n)$, with at most $3n - 6$ edges and $2n - 5$ nodes.

3.2.3 *Generalization*

The definition of a point Voronoi diagram can be generalized and extended to cover more complex input structures called *sites* such as points, straight-line segments and circular arcs.

We define $\mathcal{S}$ as a finite set of subsets of $\mathbb{R}^2$. For two points $x, y \in \mathbb{R}^2$, let $d(x, y)$ denote the Euclidean distance between x and y. For a subset of the Euclidean plane $\mathcal{Q} \subset \mathbb{R}^2$, the distance between a point x and $\mathcal{Q}$ is defined as

$$d(x, \mathcal{Q}) := \inf_{y \in \mathcal{Q}} d(x, y) \ .$$

Furthermore, for a finite set $\mathcal{S}$ of subsets of $\mathbb{R}^2$, the distance between a point $x \in \mathbb{R}^2$ and $\mathcal{S}$ is defined as

$$d(x, \mathcal{S}) := \min_{\mathcal{Q} \in \mathcal{S}} d(x, \mathcal{Q}) \ .$$

The generalized Voronoi diagram is defined upon a *proper input set* $\mathcal{S}$ of disjoint subsets of $\mathbb{R}^2$. For such a proper set of input sites the following properties are required:

- The set $\mathcal{S}$ consists of finitely many points, open straight line segments and circular arcs, less than a semi-circle, oriented counterclockwise.

- Straight line segments and circular arcs do not contain their start and end points and thus are open in the relative topology defined by their supporting lines respectively circles.

- For every segment and arc $s \in \mathcal{S}$ the end points of s are included in $\mathcal{S}$ as well [Hel11].

The generalization is done by confining the Voronoi edges of the corresponding sites to a so-called *cone of influence* which is defined differently for each type of input site. These cones of influence $\mathcal{CI}(s)$ are necessary to avoid deficiencies in the resulting Voronoi diagram, such as two-dimensional bisectors.

The cone of influence $\mathcal{CI}(s)$ for a circular arc c is the closure of the cone bounded by the rays originating in the center through its endpoints. For a straight-line segment s, the $\mathcal{CI}(s)$ is defined as the strip bounded by the normals through its endpoints. Only for a point p, the cone of influence is not restrictive, as it is the entire plane.

The topological interior of a subset $\mathcal{Q} \subset \mathbb{R}^2$ is denoted by $int(\mathcal{Q})$ and $cl(\mathcal{Q}$ stands for the closure of $\mathcal{Q}$. The cone of influence of a site s is used in the following definition of the generalized Voronoi region of s:

$$\mathcal{VR}(s, \mathcal{S}) := cl\{x \in int\mathcal{CI}(s) : d(x, s) \leq d(x, \mathcal{S})\}$$

The Voronoi polygon $\mathcal{VP}(s, \mathcal{S})$ is defined as the boundary of the Voronoi cell of s and again, the Voronoi diagram is defined as the union of all Voronoi polygons:

$$\mathcal{VP}(s, \mathcal{S}) := \partial\mathcal{VR}(s, \mathcal{S})$$
$$\mathcal{VD}(\mathcal{S}) := \bigcup_{s \in \mathcal{S}} \mathcal{VP}(s, \mathcal{S})$$

With Yap's algorithm [Yap87], there does exist an (theoretical) algorithm which can be proven to compute the Voronoi diagram of n points, straight line segments and circular arcs in $O(n \log n)$ time. This algorithm, which uses the divide and conquer approach, tends to lend itself to a tremendously large number of implementation problems, and thus there do not exist any known implementations. For a survey of algorithms and software for computing Voronoi diagrams of points, line segments and circular arcs in the Euclidean plane, please be referred to [Hel11].

The implementation VRONI by Martin Held which uses random incremental construction runs in $O(n \log n)$ expected time on n points, straight-line segments and since the ARCVRONI extension has been added by Stefan Huber [HH08b] also on circular arcs input [Hel01].

A sample output (with additional offset curves) of the computation of a generalized Voronoi diagram can be seen in Figure 7.

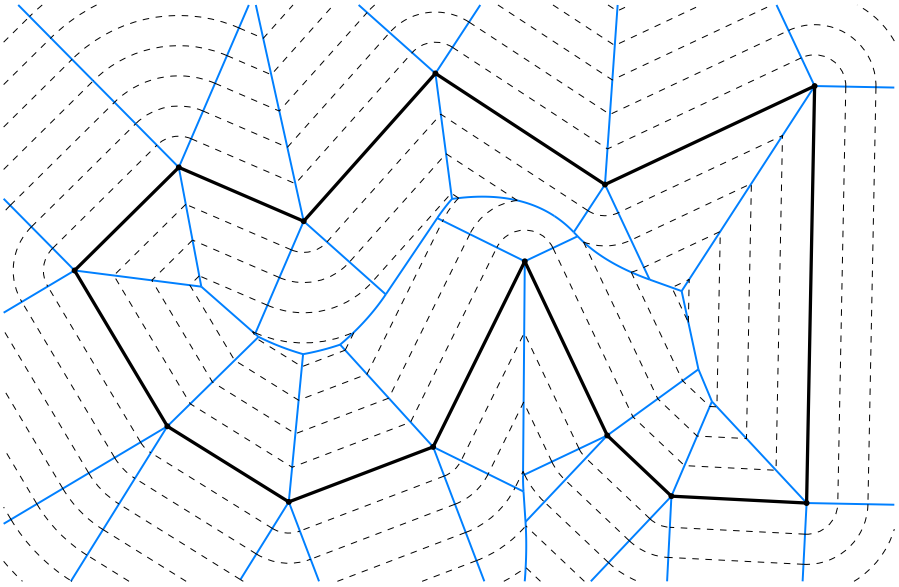


Figure 7: Generalized Voronoi diagram of a closed polygonal chain with various offset curves

3.2.4 *Offsetting*

In the following section we consider Minkowski sums and differences: The Minkowski sum $A \oplus B$ of two sets A and B in Euclidean space, also called dilation of A by B, is defined as the union of the addition of every element of A with every element of B. This special type of additions defined on sets is also often referred to as *translation* of A by B in the literature.

$$A \oplus B := \{a + b : a \in A, b \in B\}$$

A sample Minkowski sum of a square and a circle can be seen in Figure 8.

The Minkowski difference is defined as the intersection of all translations of A by $-b$, i. e.,

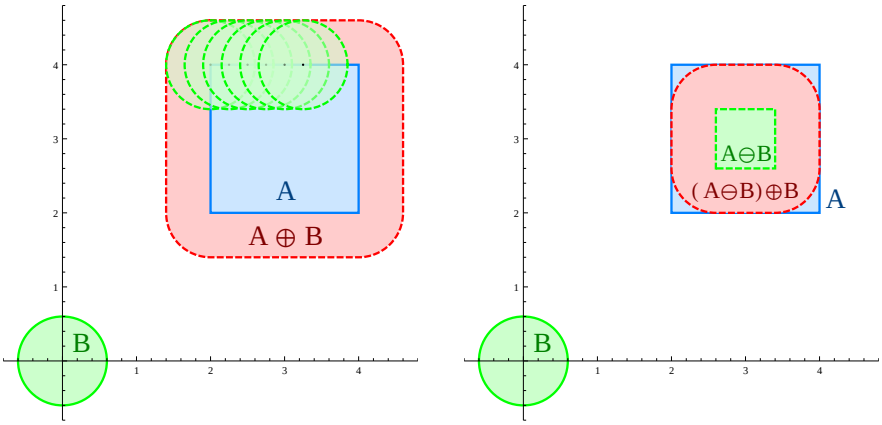
$$A \ominus B := \bigcap_{b \in B} \{a - b : a \in A\} \ .$$

It is clear that neither $(A \ominus B) \oplus B = A$ nor $(A \oplus B) \ominus B = A$ hold, with $(A \ominus B) \oplus B$ illustrated in Figure 8b. Still, one can express the Minkowski difference using Minkowski sums as

$$A \ominus B = \overline{\overline{A} \oplus -B}$$

where the complement of A, denoted by $\overline{A}$, is defined as $\{a : a \notin A\}$ and $-B$ denotes the set B reflected about the origin, i. e., $-B := \{b : -b \in B\}$.

Minkowski sums and differences of an area A with a circular disk B_r centered at the origin are usually called *offsets* [RR86, p. 130].



(a) Minkowski sum $A \oplus B$ (b) $(A \ominus B) \oplus B \neq A$

Figure 8: Figure 8a shows a sample Minkowski sum $\mathcal{A} \oplus \mathcal{B}$. The dashed circles within $\mathcal{A} \oplus \mathcal{B}$ show the translation of selected points (on the border) of $\mathcal{A}$ by $\mathcal{B}$. Figure 8b illustrates that even for simple input $(A \ominus B) \oplus B = A$ does not hold.

A very basic example can be seen in Figure 8. Unfortunately, offsetting does not need to preserve topology. Additionally the boundary of an offset curve may contain circular arcs, even for purely polygonal input, as can be seen in Figure 8: the border of the offset of the square contains circular arcs.

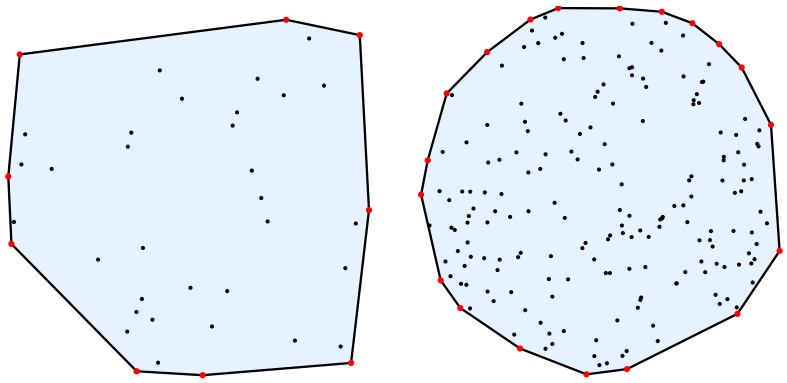
With a Voronoi diagram given, offsetting can be performed in $O(n)$ time. For a given polygon we start to construct an inner offset starting with the offset distance $d = 0$ and continuously increase d until the resulting offset degenerates to a point. If we keep track of the endpoints of the offset primitives, the set of these points forms the Voronoi diagram [Hel98]. This process again can be seen in Figure 7 [Hel91].

Thus by intersecting offset primitives with bisectors of the appropriate Voronoi cell one can compute an offset curve in linear time [Hel91]. Alternatively, without resorting to Voronoi diagrams, there do exist algorithms to compute offset curves. For example, there exists a package 2D *Minkowski Sums* [Wei10] within the CGAL library [cga]. For implementation details please refer to the manual [The10] and the package description [Wei10].

3.3 CONVEX HULL

3.3.1 *Introduction*

Convex hulls of points in a plane were one of the first problems studied in computational geometry. Following the introduction by Mark de Berg et al. in their book on computational geometry [dBCvKO08], one can visualize a convex hull of a finite set of n planar points, i. e., $P_1, \ldots, P_n \in \mathbb{R}^2$, when thinking of a set of nails protruding from a wooden board. The convex hull can then be imagined as a rubber band, running around the nails and informally speaking, connecting the *outmost* points [dBCvKO08, p. 3]. Two samples of convex hulls of sets of points can be seen in Figure 9.



(a) convex hull of 50 random points evenly distributed in a square

(b) convex hull of 200 random points evenly distributed in a circle

Figure 9: Two sets of points with corresponding convex hulls

3.3.2 *Definition*

For a precise definition of the convex hull, we first need to give a definition of convex sets: A convex set in a vector space $\mathcal{V}$ is defined as a set for which the convex combination of any two points of the set lies within the original set [PS85, p. 18]. The convex combination of n points $x_1, \ldots, x_n$ in $\mathcal{V}$ is defined as

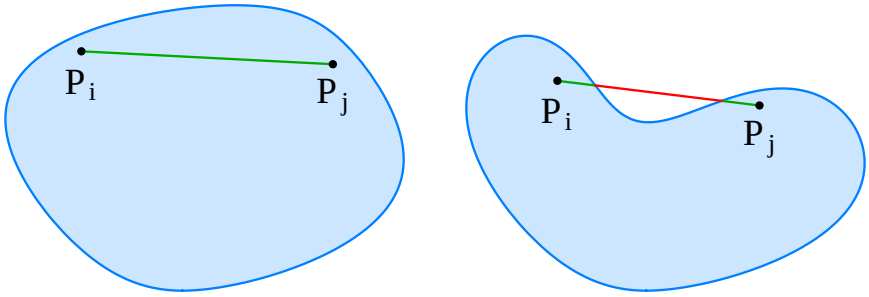
$$\sum_{i=1}^{n} \alpha_i x_i = \alpha_1 x_1 + \alpha_2 x_2 + \ldots + \alpha_n x_n$$

with

$$\forall i \in \{1, 2, \ldots, n\}\ \alpha_i \geq 0 \quad \text{and} \quad \sum_{i=1}^{n} \alpha_i = 1 \ .$$

In the Euclidean space, this lends itself to a geometric interpretation: A set is convex, if and only if for every pair of points in the

set, the straight line segment connecting these points is completely contained within the set. Following this definition, it can be observed that the complete vector space as well as the empty set are convex. It also can be shown that the intersection of convex sets always again yields a convex set [PS85, p. 18]. Two such sets in the Euclidean space, one of them convex, the other one violating above property and thus non-convex, i. e., concave, can be seen in Figure 10.



(a) convex set (b) concave set

Figure 10: Two sets, one of them convex, one concave

If we are given a finite set $\mathcal{S}$ of n points in the plane, we can define the convex hull of these points as the intersection of all convex sets containing $\mathcal{S}$. The resulting set can be seen as the minimal convex set containing all points [dBCvKO08, p. 2]. The previous definition can be extended easily to also cover convex hulls of infinite sets of points in higher dimensions.

Additionally, from an algebraic view, the convex hull can also be seen as the set of all convex combinations of $\mathcal{S}$, i. e., for $n \in \mathbb{N}$ and $\mathcal{S} = \{p_1, p_2, \ldots, p_n\}$, the convex hull is defined as

$$\mathcal{CH}(\mathcal{S}) := \left\{ \sum_{i=1}^{n} \alpha_i p_i : \forall i \in \{1, 2, \ldots, n\}\ \alpha_i \geq 0 \wedge \sum_{i=1}^{n} \alpha_i = 1 \right\} .$$

Conducting a basic induction, the latter can be shown to be a conclusion of the previously given definition.

3.3.3 *Algorithms*

Dating back in the early 1970s, a lot of research has been conducted on convex hulls as one of the first problems studied in computational geometry. There does exist profound knowledge on that topic and many different algorithms have been proposed.

One of the first algorithms published was Graham's scan, developed by Ronald Graham in 1972 [Gra72]. For 3D data, a divide and conquer algorithm was developed and published in 1977 by Preparata and Hong [PH77]. Both algorithms are in $O(n \log n)$ time complexity and thus are worst case optimal. Nevertheless, if the number of

vertices belonging to the convex hull is very small, this runtime complexity can be improved.

The algorithm by Jarvis, known as *gift wrapping algorithm*, sometimes also called *Jarvis march*, runs in $O(nh)$ time and thus is output-sensitive, i. e., its time complexity is determined by the number of convex hull points found [Jar73].

In their work entitled *The Ultimate Planar Convex Hull Algorithm?*, David G. Kirkpatrick and Raimund Seidel [KS86] give a divide and conquer algorithm and prove the lower bound for the time complexity to be $\Omega(n \log h)$. As h denotes the number of convex hull points found, this algorithm is output-sensitive as well.

In his work published in 1996, Timothy M. Chan gives an output-sensitive algorithm that runs in worst-case optimal time $O(n \log h)$, where h again denotes the number of vertices on the convex hull. Additionally, this algorithm can be extended to run on 3D data [Cha96].

Further generalizations include the extensions of the algorithms to higher dimensions and algorithms that accept more complex input than simply points in a plane. Especially for simple polygons as input, there do exist algorithms to compute the convex hull in linear time, e. g., [MA79], [GY83] or [Mel87].

3.3.4 *Applications*

Convex hulls are used in a variety of applications, most notably in intersection testing and collision detection in CAD applications or computer animation. Further applications which are not limited to geometric problems exist in many fields.

Many algorithms in the computational geometry rely on convex hulls and thus many applications exist that make use of these algorithms. An example of such an application is an improvement over the Douglas-Peucker line-simplification algorithm. This line-simplification algorithm is widely used in the field of GIS to simplify map data. An improvement of this algorithm was proposed by Hershberger and Snoeyink that uses convex hulls [HS92][HS94].

Convex Hulls are also used frequently in pattern recognition or image processing. In their work, Shi Pu and George Vosselman use convex hulls of point clouds generated from terrestrial laser scanning to extract features of buildings [PV06]. Haritaoglu et al. use convex hulls to extract human body parts and thus human body posture from monochromatic images [HHD98].

In the field of pattern recognition an example of an application which does not lie within the domain of geometric problems is the use of convex hulls for ROC analysis. Such a receiver operating characteristics (ROC) graph allows visualization, comparison and selection of classifiers, based on their performance. The ROC convex hull

of predefined points in ROC-space can be used to interpolate optimal classifiers [Faw06].

3.3.5 *Intersection Tests*

Consider two objects, $\mathcal{P}_1$ and $\mathcal{P}_2$. If these two objects are not very likely to collide, or if the number of intersection tests is rather high and many different objects are frequently tested against one of them, it comes with great advantage to first find a suitable simpler representation of the object and then test against this simplification. Ideally, such a simplification would consist of primitives that allow numerically stable, accurate and computationally faster intersection tests, in comparison to testing against the object itself which possibly consists of mathematically highly sophisticated primitives and thus is not suitable for intersection testing.

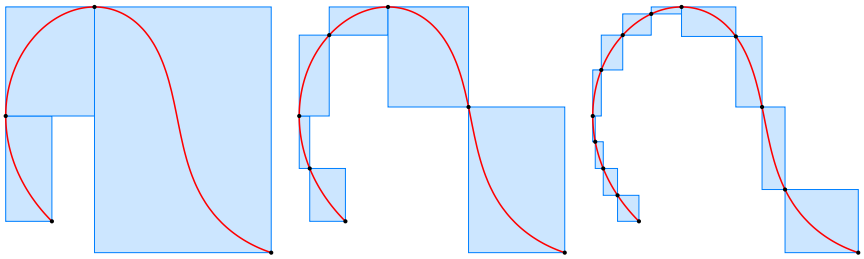
Many different types of simplifications lend themselves for this purpose and, especially in the environment of computer graphics and virtual reality, many approaches to the problems of collision detection have been discussed [HKM95]. As a very obvious choice for the simplicity of intersection tests, one could resort to axis aligned bounding boxes and hierarchies thereof, e. g., R-trees [Gut84a] and improved R^*-trees [BKSS90a]. As an alternative bounding spheres (circles in 2D) or hierarchies of bounding spheres as proposed by Philip M. Hubbard [Hub96] can be implemented.

These simplifications work well for complex input data. As we develop an algorithm to approximate with as few primitives as possible, we try to find *long* primitives defined only upon a few anchor nodes but spanning the maximum number of approximation nodes. Thus, bounding spheres or bounding boxes do not provide a good fit for these primitives and result in many false positives, i. e., intersections found, where there are none. The problem with these false positives is a consequence of the fact that our approximation is restricted to the tolerance zone. That means, for a given tolerance zone characterized by a maximum width of δ, any bounding sphere (circle) could not exceed the radius $\delta/2$ and thus for a small approximation tolerance, bounding sphere intersection tests are prone to fail.

In addition, especially for our approximation primitives, bounding boxes come with great numeric problems and a great chance to give incorrect results and, most problematically, false negatives, i. e., intersections are not found due to a numeric instability. The main problem regarding the generation of axis aligned bounding boxes lies within the generation of stationary points. A description of this approach can be found in [Sed11, p. 85], referred to as *interval subdivision method*. The algorithm starts with determining all points on the curve, for which the tangent vectors are parallel to either the x- or the y-axis. At this point it must be clear that the approach fails for curves

which are almost straight lines parallel to one of the axis. But even for simple polynomial curves without numerical or any other difficulties, finding stationary points is not a trivial task. For algorithms on polynomial root finding please be referred to, e. g., the algorithm by Jenkins and Traub [JT70] and its derived work.

The advantages of the interval subdivision method algorithm lie within the fact that any further subdivision process of the axis aligned bounding boxes can be done by simply evaluating the underlying curve at the subdivision parameter as visualized in Figure 11. Because of the choice of the initial intervals as previously described, the curve is always bounded by the axis aligned bounding boxes and their subdivisions.



(a) initial intervals (b) first iteration (c) second iteration

Figure 11: The interval subdivision method sets up the initial intervals based on axis aligned tangents. The further interval subdivision is processed by evaluating the curve at the subdivision parameter.

Due to these problems we resort to convex hulls on the first hand to gain a suitable approximation of the primitive and speed up intersection testing. If conducted properly, a negative intersection test using the convex hull as simplification indicates that no intersection between the corresponding objects exists. However, the converse does not hold. It might be very likely that the convex hulls of two objects intersect, but the objects themselves do not. In this case, one can either continue to perform an intersection test directly on the objects, or resort to checking a more fine-grained simplification of the object, resulting in a hierarchical system of simplifications to be checked at each intersection test.

3.4 CURVES

3.4.1 *Historical Perspective*

Interest in curves or curved shapes arose long before curves were regarded as mathematical objects, as shown by various ornaments assembling waves or spirals on prehistoric pottery or the systems of folds in the drapery of Greek or Gothic statues [Loc61, p. ix]. It was also for Greek geometers to first describe the curves defined by the intersection of a plane with a cone.

Menaechmus, a Greek geometer and mathematician, is known for discovering conic sections and using parabolas and hyperbolas to solve the problem of doubling the cube. For a given cube with volume V, this problem basically consists of constructing a new cube with volume $2V$ [BM89, p. 93]. Still, this problem cannot be solved using pure geometric means: Stated first by Gauss, in 1837 [Wan37] the French mathematician Pierre Wantzel was the first to come up with a proof that the cube root of 2 (as well as the trisection of an angle) is not constructible, i. e., it cannot be constructed with ruler and compass [OR99, 2011-29-08].

Although Menaechmus already used analytical means to solve the problem of doubling the cube, Greek mathematicians still could not be considered the first analytical geometers, as their main goal was to extract algebraic information as a by-product from an existing curve based on geometric primitives rather than defining a curve by algebraic equations [Sti10, p. 110].

Another well-known example of a set of curves discovered by ancient Greeks are the spirals of Archimedes, named after Archimedes of Syracuse who can be seen as one of the leading mathematicians of that era and of all antiquity [BM89, p. 120]. As many pre-Hellenic cultures, such as the Egyptians or Babylonians, lacked the concept of angle measure, we first find a systematic study of angles (and arcs) with the Greeks. Although Euclid does not describe trigonometry directly, there are theorems equivalent to some of the trigonometric laws, such as for example the law of cosines.

This Archimedean spiral provides a solution to the problem of trisecting an angle. It is a curve defined by a point moving uniformly along a ray, starting in the origin, while the ray itself uniformly rotates about its end point [BM89, p. 126].

An example of such a trisection can be seen in Figure 13. In order to trisect the angle defined by the two rays both starting in the origin O through points A and B we start by trisecting the line segment between the origin and the first intersection of the ray defined by B with the Archimedean spiral. These points are denoted C and D and the lengths of $\overline{OC}$, $\overline{CD}$ and $\overline{DB}$ are the same. The intersection of the Archimedean spiral with the origin-centered circle through point C is

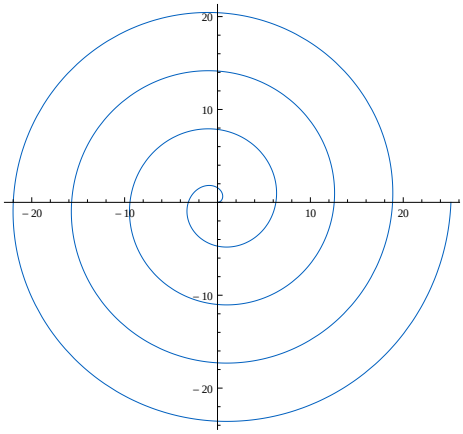


Figure 12: Archimedean spiral

called C'. It can be shown that the ray starting in the origin O through C' is one third of the angle defined by the two original rays [BM89, p. 126].

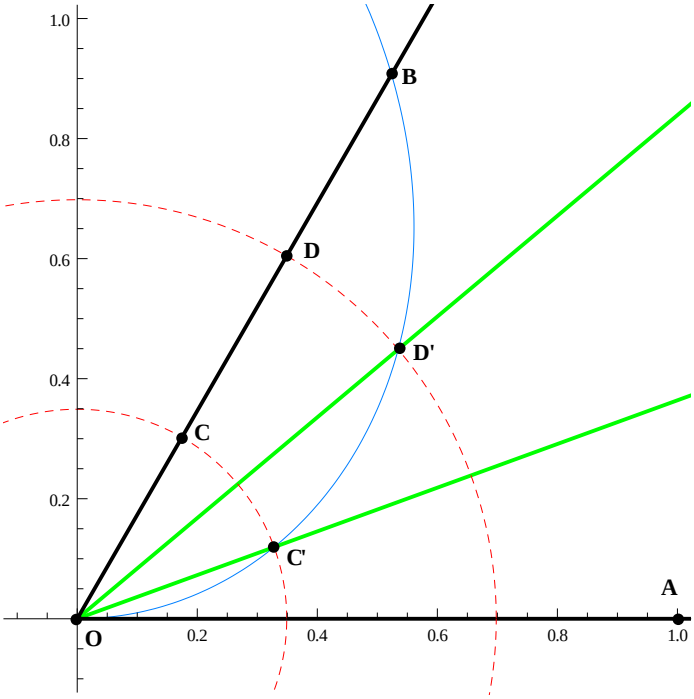


Figure 13: Trisecting an angle

Under the rulership of Ptolemy I an institute known as *the Museum* was established in Alexandria. Euclid, also known as Euclid of Alexandria, was called to teach mathematics and there he wrote one of the most successful textbooks – the *Elements*. Historically, what we now denote a *curve* was called a *line*. In his *Elements*, Euclid distinguished between straight lines and curved lines, which still can be seen in use when talking of *straight line segments* [Heao8].

Throughout history many prominent examples of curves have been examined. So for example in early Roman times, splines had been used to build ships. The idea was to reuse templates for bending the ships' planks, allowing easy recreation based on a stored basic vessel geometry. From the 13th to 16th century, Venetians perfected this technique which also became popular in England around 1600 when the wooden beam used to draw curves, the classical *spline*, probably was invented [Now06, p. 8] [FHK02, p. 2] [Far02, 2011-08-30].

The mathematics that had been developed to solve historical problems had permanence and significance that far exceeded the original problems [PS85, p. 1].

> [...] Geometry is at the heart of mathematical thinking. It is a field in which intuition abounds and new discoveries are within the compass (so to speak) of nonspecialists.

Although many Greek geometers studied curves to solve geometrical problems that previously could not be solved with compass and straight edge constructions, it was for analytic geometry in the seventeenth century to further advance the theory of curves. The centerpiece of this new invention was the ability to describe curves based on mathematical equations rather than on mechanical or theoretic constructions or primitives.

Kepler tried a variety of curves before using elliptic curves as the best fit for the planetary orbits [Loc61, p. ix]. Again it was for analytical geometry that a precise distinction between algebraic and transcendental curves could be made.

In the first half of the seventeenth century the mathematicians Fermat and Descartes both realized that many geometric problems could not only be translated and routinely solved by algebraic operations, but also curves of higher degree such as cubic or quartic curves could be described by equations [Sti10, p. 110].

3.4.2 *Definitions of Curves*

When talking about curves as mathematical objects, we have to deal with different representations of the same curve. A curve can be seen as the path of a particle moving in a vector space, e. g., $\mathbb{R}^n$, most commonly the Euclidean plane $\mathbb{R}^2$ or Euclidean space $\mathbb{R}^3$. So for an analytic representation we need to *map* a parameter (e. g., the time) to the current position of the particle in the space. A parametric representation of such a curve can be defined as a vector-valued function

$$\gamma : I \to \mathbb{R}^n \quad .$$

The domain I is a non-empty interval over $\mathbb{R}$, e. g., $I = [a, b]$ with $a, b \in \mathbb{R}$ and the image of γ is a subset of $\mathbb{R}^n$. Although there are no

restrictions on the choice of a and b, usually the interval I is normalized to $[0,1]$ [PT97, p. 1].

Such a vector-valued function can be decomposed into coordinate functions and thus the function can be seen as

$$\gamma(t) = \big(\gamma_1(t), \gamma_2(t), \ldots, \gamma_i(t)\big)$$

with

$$\gamma_i : I \to \mathbb{R} \quad \forall i \in \{1, 2, \ldots, n\} \quad .$$

If the curve is defined by a purely polynomial function, we use the *degree of the curve*, $\deg(\gamma)$, to describe the degree of the polynomial function. Additionally, sometimes the non-standard term *order of the curve* is used. The order of a curve usually coincides with the number of control vertices or similar control structures used to define the curve and usually can be set to $\deg(\gamma) + 1$.

For a closed interval $[a, b]$ the *start point* is given by $\gamma(a)$ and $\gamma(b)$ denotes the *end point* of the curve γ; in the special case that $\gamma(a) = \gamma(b)$, we call γ *closed*. An additional property of such parametric curves is the class of differentiability, in short the class of the curve: If γ is r times continuously differentiable, we speak of a curve of class C^r and if in addition $\forall i \leq r\ \gamma^{(i)}(a) = \gamma^{(i)}(b)$, i. e., the first r derivatives are identical, the curve is a *closed C^r-curve*. Sometimes the additional term of parametric continuity can be found in the literature [PM02, p. 17].

Furthermore, a curve is called *regular*, if $\forall t \in I\ \gamma'(t) \neq 0$, i. e., the curve consists only of regular points, for which the derivative is not equal to 0. Points on the curve, for which all derivatives of coordinate functions $\gamma_i'(t)$ vanish, are called singular [Str88, p. 3].

Therefore, for a curve in analytic representation in $\mathbb{R}^3$, the coordinates of the point P at parameter t, $P = (x, y, z) = \gamma(t)$, can be expressed by three functions [Str88, p. 1]

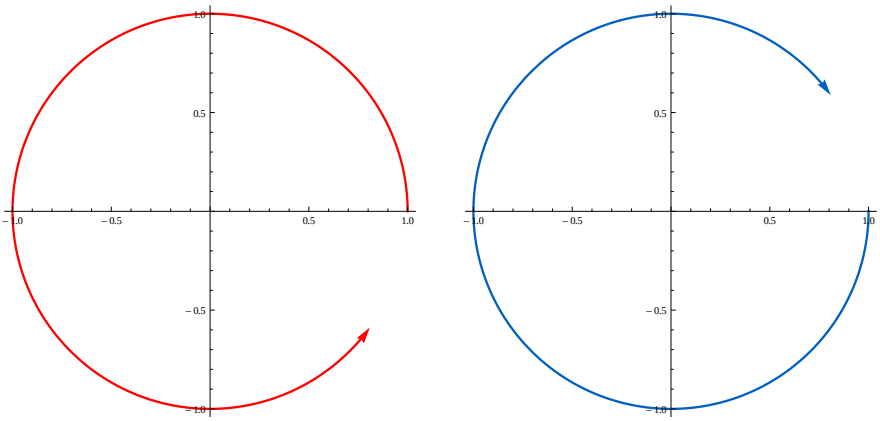
$$x = \gamma_1(t), \qquad y = \gamma_2(t), \qquad z = \gamma_3(t) \quad .$$

The image of the parametric curve is defined as the set of points $\gamma(I) \subset V$, the real value $t \in I$ is called parameter of the curve and $\gamma(t)$ is a position vector of a point on the curve. It is important to distinguish between the curve and its image: it is possible to find different parametric representations of curves with the same image.

Based on the idea that different parametrizations of a curve can have the same image, we define an equivalence relation on parametric curves: We call γ_1 and γ_2 equivalent, if a strictly increasing bijective function $\phi : I_1 \to I_2$ exists, such that

$$\gamma_2(\phi(t)) = \gamma_1(t) \quad \forall t \in I_1 \quad .$$

As such a strictly increasing bijective function is required to be continuous, the parametric curves γ_1 and γ_2 have the same image



(a) The curve $\gamma_1(t)$ defined as

$$\gamma_1(t) = \begin{pmatrix} \cos 2\pi t \\ \sin 2\pi t \end{pmatrix}$$

(b) The curve $\gamma_2(t)$ defined as

$$\gamma_2(t) = \begin{pmatrix} \cos 2\pi t \\ -\sin 2\pi t \end{pmatrix}$$

Figure 14: The parametric curves $\gamma_1(t)$ and $\gamma_2(t)$ have the same image for parametric domain $I = [0, 1]$, but are not equivalent to each other. Above, the curves are both plotted for parameters $t \in [0, 0.9]$.

and, furthermore, the parametric curve γ_2 can be seen as a so-called re-parametrization of γ_1.

A *curve*, sometimes denoted as $\tilde{\gamma}$, is defined as an equivalence class under the above equivalence relation [Lino8, p. 147]. As ϕ is required to be strictly increasing, γ_1 (Figure 14a) and γ_2 (Figure 14b) in above example are representatives of different curves in terms of the definition of equivalence classes. So for the interval $[0, 1]$ as parametric space, these two parametric curves γ_1 and γ_2 are not representatives of the same curve but they do have the same image, a unit circle. Note that for a strictly increasing ϕ regular points on the curve remain regular [Str88, p. 4].

It can be observed that in the literature the distinction between the terms for *curves*, *parametric curves* or *geometric curves* tends to be ambiguous. Whereas we define curves based on (equivalent) maps from a parametric domain into space, the concept of *geometric curves* handles *lines* in the plane as a subset of the space which can locally be *reshaped* into a straight line at any point. This means, a geometric curve can be called a one-dimensional submanifold of the space. The connection between those two different definitions of curves can easily be made via the image of the parametric curve: The image of the curve is (at least locally) a geometric curve, whereas a geometric curve can be parametrized by arc length [Bero3, p. 12f]. For an in-depth discussion of these properties, see Chapter 8 of [BG88].

So this means other ways of defining (geometric) curves do exist: Beside the previously defined parametric definition the implicit definition is one of the most common forms. Such an implicit definition can be seen as an equation or a system of equations describing the re-

lationship between the coordinates of points belonging to the image of a curve [PT97, p. 1]. As an example, take the following (abstract) curve in three dimensional space [Str88, p. 4]:

$$F_1(x,y,z) = 0, \qquad F_2(x,y,z) = 0 \ .$$

Also, using the well-known equation for the circle $x^2 + y^2 = r^2$, we can describe the parametric equation for the unit circle (Figure 14) in an implicit way (Figure 15):

$$F(x,y) = x^2 + y^2 - 1 = 0$$

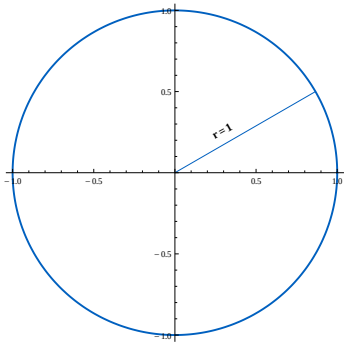


Figure 15: Implicit Definition of γ: $F(x,y) = x^2 + y^2 - 1 = 0$

This implicit definition of a curve shows that it is not only possible to define the curve based on a function with a parameter space (one-dimensional, e. g., $I = [0,1]$ for a curve) as a domain but also as the zero locus of one function or a set of functions which assembles the set of points for which the function value is equal to zero [Gal99, p. 63]:

$$\tilde{\gamma} = \bigcap_{i=1}^{d-1} F_i^{-1}(0) = \{x \in V : F_i(x) = 0 \ \ \forall i = 1, 2, \ldots, d-1\}$$

In the above example (Figure 15), the point $P\left(\frac{\sqrt{3}}{2}, \frac{1}{2}\right)$ is indeed a point of the curve γ as for P_x, P_y the equation $F(P_x, P_y) = 0$ holds:

$$\left(\frac{\sqrt{3}}{2}\right)^2 + \left(\frac{1}{2}\right)^2 - 1 = \frac{3}{4} + \frac{1}{4} - 1 = 0 \ .$$

Of course it could also be possible that the resulting set for $\tilde{\gamma}$ equals the empty set. As expanding our curves to complex numbers does not really help, especially when in need of visualizations, we have to rule out such exceptions, which can cause serious problems. As we only deal with curves, not surfaces or objects of higher degree, it is also necessary to have a set of $n-1$ equations to define a curve in $\mathbb{R}^n$ [Gal99, p. 67f].

Throughout this thesis, we will focus on parametric representations of curves. To describe the following basic properties, we will consider $\tilde{\gamma}$ a curve with parametric representation $\gamma : I = [a,b] \to \mathbb{R}^n$.

3.4.3 *Basic Properties of Curves*

Many of the basic properties of curves were first discovered in the seventeenth century: In his publication *La Géométrie* from 1637, René Descartes presented examples of his work as well as showed off with the benefits his new method came with. One of the greatest advantages of the newly discovered analytic geometry was the connection between algebra and geometry which eventually made it possible to handle geometric problems algebraically [DM06, p. lxiii]:

> Descartes's mathematics is presented in a more elegant, flexible, and developable notation than used by previous generations, with a more successful use of diagrams, and a broader range of expressions, including those for negative and imaginary numbers and variables.

So with any curve being representable as equation of two variables, Descartes defined a class of geometric curves for which all points must have a relation to the points of a straight line expressable by a single equation. He additionally distinguished between geometric and mechanic curves. Whereas the geometric curves basically represent the Euclidean rule and compass constructions and intersections of these, Descartes regarded mechanic curves as of no concern to the true geometer [Jes07, p. 419].

> *La Géométrie* was written to boast about his discoveries, not to explain them. There is little systematic development, and proofs are frequently omitted [...] [Sti10, p. 111].
>
> Discartes's conceit is so great that it is a pleasure to see him come a cropper occasionally [...] [Sti10, p. 112].

One of these *croppers* is related to the length of curves. Descartes thought this length not to be computable:

> [...] geometry should not include lines that are like strings, in that they are sometimes straight and sometimes curved, since the ratios between straight and curved lines are not known, and I believe cannot be discovered by human minds, and therefore no conclusion based upon such ratios can be accepted as rigorous and exact [DSL25, p. 91].

To give an exact definition of the length of curves, we first define a decomposition of the interval I to be the n-tuple of reals $x_1, x_2, \ldots, x_n$ such that $a = x_0 < x_1 < x_2 < \ldots < x_n = b$ for a natural number $n \in \mathbb{N}$. A given curve γ can be approximated by connecting a finite number of n points, given by $\gamma(x_1), \gamma(x_2), \ldots, \gamma(x_n)$ lying on the curve. For this approximation, which is a polygonal chain and

consists only of straight line segments, the actual length can be easily found. Following the basic rules in metric spaces (and using the triangle inequality), the length of such an approximation cannot decrease when adding additional points between existing ones.

Using this property, we can conclude that the length of a curve is the supremum over all decompositions of I of the length of the polygonal chain defined by the points on the curve given by the decompositions' values as parameters [Bero3, p. 4]. If an upper bound L on all polygonal approximations (over all possible decompositions of the parameter space) exists, the curve is called *rectifiable* with length L.

It is important to note that this definition is invariant of the actual parametrization used: If we consider two parametrizations $\gamma_1(t)$ and $\gamma_2(u)$ of the same curve $\tilde{\gamma}$, following the definition there exists a strictly increasing bijective function $\phi(t)$ which maps one parameter to the other. Thus, we only have to apply the inverse function ϕ^{-1} on the decomposition of the parameter space to get for γ_2 exactly the same polygonal chain approximation as for γ_1. The inverse function ϕ^{-1} exists (by definition) and, furthermore, must be strictly increasing.

So for a given curve $\tilde{\gamma}$ with parametric representation $\gamma(t) : [a, b] \to V$ and a decomposition $\mathcal{P} = \{a = x_0 < x_1 < x_2 < \ldots < x_n = b\}$, the length of the polygonal approximation with respect to $\mathcal{P}$ and the metric defined on V is defined as

$$l(\mathcal{P}) = \sum_{i=1}^{n-1} d\big(\gamma(x_{i-1}), \gamma(x_i)\big)$$

and thus the *length* L of $\tilde{\gamma}$ is defined as [WZ77, p. 21]

$$L = L(\tilde{\gamma}) = \sup_{\mathcal{P}} l(\mathcal{P}) \ .$$

For a given decomposition $\mathcal{P}$ of $[a, b]$, we define a so-called *refinement* to be a decomposition $\mathcal{Q}$ of $[a, b]$ which contains at least all of the points already contained in $\mathcal{P}$ and possibly some more. The norm of a decomposition can be defined as the maximum distance between two consecutive points of that partition:

$$\|\mathcal{P}\| = \max_{i=1,2,\ldots,n-1} x_{i+1} - x_i$$

We call curves that have the same length *isometric* with an *isometry* mapping one curve onto another preserving the curve's length [Bero3, p. 11]. An important observation is that all curves are isometric to an interval of the same length on the one-dimensional Euclidean space, the line $\mathbb{R}^1$.

It is quite important that there do exist functions that are non-rectifiable, i. e., curves that do not have an upper bound on the length of their polygonal approximations: Informally speaking, these curves

have infinite length. An example of such a curve is the curve defined as the graph of the function $f(x) = x \sin \frac{1}{x}$ on the interval $(0, 1]$. Another prominent example of such a curve is the Koch snowflake curve: The length of the curve increases with every iteration by one third of the original length regardless of the number of the iteration step.

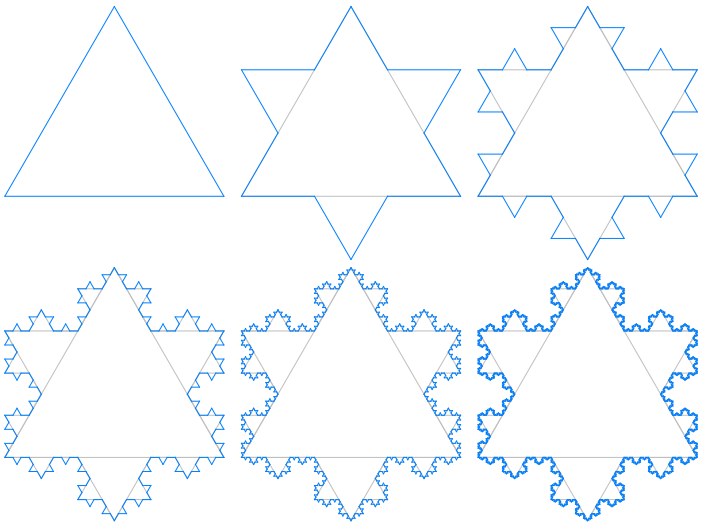


Figure 16: Koch snowflake curve, iterations 1 to 5

Using the above definition, it can be shown that

$$L(\tilde{\gamma}) = \lim_{\|\mathcal{P}\| \to 0} l(\mathcal{P})$$

holds for a rectifiable curve.

Given a curve $\gamma(t) : [a, b] \to \mathbb{R}^n$, it can be shown [WZ77, p. 22] that $\tilde{\gamma}$ is rectifiable if and only if all coordinate functions are of bounded variation, i. e., the total variation V of the real-valued functions γ_i defined as

$$V(\gamma_i) = \sup_{\mathcal{P}} \sum_{i=1}^{n-1} |\gamma_i(x_{i+1}) - \gamma_i(x_i)|$$

with respect to the previously defined decomposition is a finite value. As a result of this statement, one can show that the length of the curve is bounded by the sum of the total variations of its coordinate functions.

Following the idea of Riemann integrals, it can be seen that this total variation on the interval $[a, b]$ of a continuous and differentiable function ϕ simplifies to

$$V(\phi) = \int_a^b \left\|\phi'(x)\right\| dx \ .$$

Consider a curve $\tilde{\gamma}$ with regular parametric representation $\gamma(t)$ with parametric domain $[a, b]$ of class C^r, $r \geq 1$. The vector $\gamma'(t)$

is the tangent vector at parameter t based in the point with position vector $\gamma(t)$. The length of the curve can be expressed by means of the integral [Str88, p. 5] [NF90, p. 3] [Ber03, p. 11]

$$L(\tilde{\gamma}) = \int_a^b \left|\gamma'(x)\right| dx$$

which can be seen as the integral over the *speed*, i. e., the length of the tangent vector [Lin08, p. 149]. Of course this integral can be used to denote the arc length function $L_{\tilde{\gamma}}(b)$ using the upper bound as parameter which gives the length of the curve up to an arbitrary parameter b [Rau08, p. 59]. The length of the tangent vector $\gamma'(t)$ can be computed as the square root of the dot product, i. e.,

$$\left\|\gamma'(t)\right\| = \sqrt{\gamma'(t) \cdot \gamma'(t)} \ .$$

Again it is an important observation that the length of the curve in the definition stated above is independent of the choice of the underlying parametrization [NF90, p. 7].

This lends itself to the definition of a *unit tangent vector* $T(t)$, the vector pointing in the same direction as $\gamma'(t)$ but at length 1.

$$T(t) = \frac{\gamma'(t)}{\left\|\gamma'(t)\right\|} \ .$$

If for any parameter the tangent vector is a unit vector, i. e.,

$$\forall t \in [a,b] : \ \left\|\gamma'(t)\right\| = 1 \ ,$$

this special form of a parametrization is the *arclength parametrization* and γ is at *unit speed*. Clearly an arclength parametrization is of great advantage from a formal point of view, as many proofs and theorems can be simplified in terms of arc length. Nevertheless, this would require finding a solution of the arc length integral which tends to be not only difficult in the general case but sometimes even impossible [Rau08, p. 60].

Formally, the arc length parametrization property can be written using a parameter s and the arc length function $L_{\tilde{\gamma}}(s)$ as

$$\forall s \in [a,b] : \ s = L_{\tilde{\gamma}}(s) = \int_a^s \left\|\gamma'(t)\right\| \ dt \ .$$

Geometric continuity

We previously defined the class C^r of a curve $\gamma(t)$ which describes the differentiability of the curve, i. e., γ is of class C^r, if it is r times continuously differentiable.

Another important property of curves that we use throughout this thesis is the *geometric continuity* of a curve $\gamma(t)$. This term is closely related to the class C^r. In contrast to the previously defined class of

differentiability, depicted as C^r, we characterize the class of geometric continuity as G^r.

A curve is of class G^0, if at any parameter t_0 the left and the right limit points of the curve are identical. For a curve consisting of a set of segments, it follows that any two consecutive curve segments touch at a common join point. Similarly, a curve of class G^0 is also of class G^1, if at any parameter the limit points of the curve's tangents point in the same direction. At this point, the difference between the class C^1 and G^r becomes obvious: For a C^1-curve at any parameter t_0 the limit points of the left and the right tangents have to be identical, as the curve is differentiable. In contrast, for a G^1-curve only the directions have to be the same, but the tangents may have different lengths.

This leads to the following generalization: A curve of class G^{r-1} is also of class G^r, if for any parameter t_0 the normalized r-th derivative of the curve is continuous, i. e., for an appropriate $\lambda \in \mathbb{R}$

$$\lim_{t \to t_0^-} \gamma^{(r)}(t) = \lambda \lim_{t \to t_0^+} \gamma^{(r)}(t) \ .$$

Concludingly, a curve $\gamma(t)$ is of class G^2, if at any parameter t_0 the left and the right limit points are identical, the left and right tangents point into the same direction and the left and the right segments share a common center of curvature:

$$\begin{aligned} \lim_{t \to t_0^-} \gamma(t) &= \lim_{t \to t_0^+} \gamma(t) \\ \exists \lambda_1 \in \mathbb{R} : \lim_{t \to t_0^-} \gamma'(t) &= \lambda_1 \lim_{t \to t_0^+} \gamma'(t) \\ \exists \lambda_2 \in \mathbb{R} : \lim_{t \to t_0^-} \gamma''(t) &= \lambda_2 \lim_{t \to t_0^+} \gamma''(t) \end{aligned}$$

From this definition can be derived that any curve of parametric continuity up to class G^r can be reparametrized to be of class C^r [BD89].

Beyond mathematical aspects, G^2 continuity is important for aesthetic reasons [PM02, p. 16]. Especially in automotive design, the use of visually pleasing shapes and curves is subject to additional aerodynamic constraints and one major goal is to create both, functional and aesthetic surface design. For the creation of functional surfaces, a major problem lies within fluid dynamics. (For example flow separation, see [LH08], can be reduced by using primitives of class G^2 [PM02, p. 16].) For the latter case, methods for surface quality assessment have been proposed in the literature, e. g., the Highlight-line algorithm [BC94].

4

TOLERANCE ZONE

The tolerance zone can be seen as a region around the input, in which it is safe for the approximation to reside. In the following section we start with giving a basic definition of such a tolerance zone of the input. Our definition is based on definitions postulated in the literature and extended to be suitable for more complex input curves. We focus especially on multiply-connected open polygonal chains and polygons.

Based on our definitions we give an algorithm to compute the boundary of such a tolerance zone. The algorithm takes account of traditional offsetting but also incorporates the *pseudo offsets* as described in [HHo8a]. The resulting boundary is sampled and approximated using straight line segments which lend themselves for fast intersection tests especially with more sophisticated approximation primitives. For more details on these intersection tests for splines and Bézier-curves, see Chapter 6.

4.1 DEFINITION

In their work [HHo8a], Heimlich and Held define a signed distance $d_s(\mathcal{P}, x)$ between the input $\mathcal{P}$ and a given point x depending on interior/exterior properties and the nesting level of polygons as follows:

$$d_s(\mathcal{P}, x) := \begin{cases} d(\mathcal{P}, x) & \text{if } x \in int(\mathcal{P}) \text{ and } nl(\mathcal{P}) \text{ is even, or} \\ & \textit{if } x \in ext(\mathcal{P}) \text{ and } nl(\mathcal{P}) \text{ is odd,} \\ 0 & \textit{if } x \in \mathcal{P}, \\ -d(\mathcal{P}, x) & \text{if } x \in ext(\mathcal{P}) \text{ and } nl(\mathcal{P}) \text{ is even, or} \\ & \textit{if } x \in int(\mathcal{P}) \text{ and } nl(\mathcal{P}) \text{ is odd,} \end{cases}$$

This definition of the signed distance is used to define the tolerance zone as the set of points x, such that the signed distance lies in the interval of the given maximum left and right tolerance d_l and d_r:

$$\mathcal{TZ}(\mathcal{P}, d_l, d_r) := \{x \in \mathbb{R}^2 : d_r \leq d_s(\mathcal{P}, x) \leq d_l\}$$

Clearly, this approach does not extend to open polygonal chains as it uses the interior/exterior and nesting levels of polygons which do not exist for open polygonal chains. Thus we have to redefine the signed distance and the tolerance zone to take these polygonal chains as input into account:

Consider a set $\mathcal{P}$ of m open or closed simple polygonal chains $p_1, p_2, \ldots, p_m$ that are pairwise disjoint. Let $\mathcal{VD}(\mathcal{P})$ denote the Voronoi diagram of $\mathcal{P}$, $P \in \mathcal{P}$ a simple polygonal chain and n the number of vertices $P_1, P_2, \ldots, P_n$ of P. The straight line segment between two consecutive vertices P_i, $P_{i+1} \in P$ is denoted by $\overline{P_i P_{i+1}}$.

The Voronoi cell of an input site s has been defined in Section 3.2. We use the Voronoi diagram to give a definition of a tolerance zone that is constructed by joining subsets of Voronoi cells. For each input element we use the intersection of the element's tolerance zone defined as follows with the element's Voronoi cell. The tolerance zone is defined as the union of all these individual tolerance zones.

Straight Line Segments

To give a definition of the input elements' tolerance zones, we first define the signed distance $d_s\left(\overline{AB}, C\right)$ between a straight line defined by the points $A, B \in \mathbb{R}^2$ and a point $C \in \mathbb{R}^2$ as

$$d_s\left(\overline{AB}, C\right) = \frac{\det\left(A, B, C\right)}{\|B - A\|} \quad .$$

In this notation, $\det\left(A, B, C\right)$ denotes the determinant of the matrix defined by the homogeneous coordinates of the points A, B and C, i. e.

$$\det\left(A, B, C\right) = \det \begin{pmatrix} A_x & B_x & C_x \\ A_y & B_y & C_y \\ 1 & 1 & 1 \end{pmatrix} \quad .$$

In literature, sometimes the transposed matrix is used, as the computation of the determinant is invariant under transposition of the matrix. The determinant of such a 3×3 matrix can be computed using Sarrus' rule. Referring to the above definition, this yields the expression

$$\det\left(A, B, C\right) = A_x B_y + B_x C_y + C_x A_y - A_x C_y - B_x A_y - C_x B_y$$

Basically, the determinant returns twice the signed area of the triangle defined by the points A, B and C. Alternatively, this can be seen as the signed area of the parallelogram defined by the four vertices A, B, C and $D = B + C - A$. For such a signed area, the absolute value determines twice the triangle's area, whereas the sign determines its orientation, i. e. the position of the third point with respect to the straight line defined by the other two points: A positive or negative sign indicates whether the point lies left or right of the line whereas a zero value indicates that the point lies exactly on the line. Thus we can conclude that the resulting absolute value indeed gives the distance between the straight line and the point. For details refer to a

textbook on linear algebra or on mathematics for computer graphics, e. g. [Mor99] or [Vin10].

This signed distance between a point and a straight line is used to describe the tolerance zone $\mathcal{TZ}_{\text{line}}(\overline{AB}, d_l, d_r)$ of the straight line segment $\overline{AB}$:

$$\mathcal{TZ}_{\text{line}}(\overline{AB}, d_l, d_r) = \{x \in \mathbb{R}^2 : d_r < d_s(\overline{AB}, x) < d_l\}.$$

The intersection of the straight line's tolerance zone, denoted by $\mathcal{TZ}_{\text{line}}(\overline{AB}, d_l, d_r)$, with the Voronoi cell $\mathcal{VR}(\overline{AB}, \mathcal{P})$ of the straight line segment $\overline{AB}$ with respect to the Voronoi diagram of the input $\mathcal{P}$ gives the straight line segment's tolerance zone. Thus, for a segment $s \in \mathcal{P}$, the tolerance zone is defined as

$$\mathcal{TZ}_{\text{seg}}(s, d_l, d_r) := \mathcal{TZ}_{\text{line}}(s, d_l, d_r) \cap \mathcal{VR}(s, \mathcal{P}) \quad .$$

It can easily be seen that this definition allows the use of one-sided tolerance zones, i. e. tolerances d_l and d_r, such that $sign(d_l) = sign(d_r)$, where the whole tolerance zone is located on one side of the input with respect to the input's orientation.

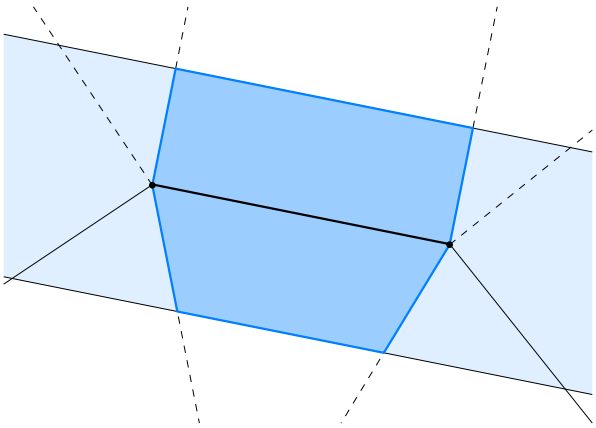


Figure 17: The tolerance zone of a straight line segment

Points with Two-Sided Tolerances

In addition to the tolerance zones of the line segments we also need to examine the tolerance zones of the input vertices P_i. These tolerance zones are defined differently, depending on the approximation tolerance type. For open polygonal chains we take only inner vertices into account, i. e., those vertices that are neither the start nor the end vertex of the polygonal chain and thus omit the *caps* around the start and end vertices that can be seen when computing traditional offset curves.

If the input lies within the tolerance zone and thus we do not have a one-sided tolerance zone, we distinguish between the vertices P_i, for which the triangle $\triangle P_{i-1}, P_i, P_{i+1}$ is oriented clockwise and those

vertices P_i, for which the same triangle is oriented counter-clockwise. Note that we rule out degeneracies for our input and thus expect the determinant used to compute the orientation never to return zero.

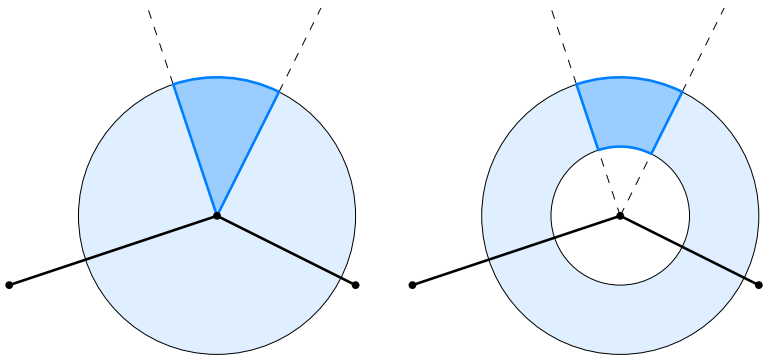
For clockwise orientation, we use the circle centered in P_i with radius $|d_l|$, for counter-clockwise orientation we use the same circle but with radius $|d_r|$ and thus by defining d_i as giving the appropriate tolerance for P_i we get

$$d_i(d_l, d_r) := \begin{cases} |d_l|\,, \text{ if triangle at } P_i \text{ is oriented CW} \\ |d_r|\,, \text{ if triangle at } P_i \text{ is oriented CCW} \end{cases}$$
$$\mathcal{TZ}_{\text{point}}(P_i, d_l, d_r) := \{x \in \mathcal{VR}(P_i, \mathcal{P}) : d(x, P_i) < d_i(d_l, d_r)\}$$

Points with One-Sided Tolerances

For one-sided tolerance zones, we define the lower tolerance d_{min} and the upper tolerance d_{max} to give the following definition of the set of points that belong to the ring sector around the given vertex P_i:

$$d_{\min} := \min(\,|d_l|\,, |d_r|\,)$$
$$d_{\max} := \max(\,|d_l|\,, |d_r|\,)$$
$$\mathcal{TZ}_{\text{point}}(P_i, d_l, d_r) := \{x \in \mathcal{VR}(P_i, \mathcal{P}) : d_{\min} < d(x, P_i) < d_{\max}\}$$



(a) Tolerance zone of a vertex

(b) One-sided tolerance zone of a vertex

Figure 18: Regular (Figure 18a) and one-sided (Figure 18b) tolerance zones of a vertex. The resulting tolerance zone is the union of the intersection of the tolerance zones of the individual input elements with the corresponding Voronoi cells.

Resulting Tolerance Zone

Finally, we conclude that the tolerance zone of the polygonal input chains $\mathcal{P}$ is the union of all intersections of the individual tolerance

zones of line segments s and points p with the Voronoi cells of the corresponding sites. The tolerance zone primitives as described above are visualized in Figure 17 and Figure 18.

For segments s and points p contained in the input, the union of the elements' tolerance zone primitives yields the tolerance zone of the input curve:

$$\mathcal{TZ}(\mathcal{P}, d_l, d_r) := \bigcup_{s \in \mathcal{P}} \mathcal{TZ}_{\text{seg}}(s, d_l, d_r) \cup \bigcup_{p \in \mathcal{P}} \mathcal{TZ}_{\text{point}}(p, d_l, d_r)$$

It must be observed that this definition covers the interior of the tolerance zone, but not the closure. The algorithm described in the following section computes the boundary of the tolerance zone as defined, which can be used to check the candidate approximation primitives for intersections. Thus, we clearly demand the tolerance zone to only consist of the interior and do not allow primitives that touch the tolerance zone boundary, as we would discard those primitives that touch the tolerance zone boundary because they form a (degenerate) intersection with the boundary.

4.2 TOLERANCE ZONE BOUNDARY COMPUTATION

We implemented an algorithm that computes the border of the tolerance zone of a polygonal chain or a polygon. The algorithm runs in three stages and returns a list of vertices that define the tolerance zone, starting in the first and ending in the last vertex of the input. The algorithm requires the Voronoi diagram of the input. Although the algorithm runs on one particular polygon or polygonal chain, the Voronoi diagram computation is required to be carried out on the entire multiply-connected input.

As a preprocessing, we use VRONI by Martin Held [Held01] to compute the Voronoi diagram of the input. The resulting graph returned by VRONI consists of nodes and edges. The clearance disk of a point p in the Voronoi diagram is the largest disk centered at p such that no site is contained within its interior. The edges, which are in the general case portions of conic curves, are parametrized by the clearance radius, i. e., evaluation of an edge at a given parameter t returns the center of the clearance disk with the specified radius t. In the resulting data structures, parabolic arcs are split at their apex such that as an guaranteed property the parameter is always strictly increasing or decreasing when traversing an edge.

4.2.1 *Collect Nodes of Voronoi Cells*

In the first step we add all nodes of Voronoi cells adjacent to the current polygon or polygonal chain P to a list. We start in the first input vertex and the corresponding Voronoi cell and run along the Voronoi edges in the appropriate direction, adding one node after another until we return to a node located directly on the input. Such nodes can be identified by their parameters, as nodes located directly on the input have a node parameter equal to zero. We continue this procedure with the next Voronoi cell until we reach the last vertex of the input chain.

The resulting list forms a path given by consecutive nodes over the Voronoi diagram starting in the first and ending in the last vertex of the input. Clearly, every pair of consecutive nodes in this list is connected by a Voronoi edge.

For the algorithm given in Algorithm 1, we need the set of curve vertices as input. Additionally, we require that the side of the tolerance zone boundary is given, i. e., we need to know whether we compute a left or right boundary. Furthermore, we assume that when traversing a Voronoi polygon, every Voronoi edge is oriented with respect to its start and end nodes according to the current traversal. Thus, a traversal of a Voronoi cell of a straight line segment $\overline{AB}$ starting with an edge incident in A will traverse an edge ending in B followed by an edge starting in B and then end with an edge with

end node incident in A. Note that this does not reflect the edge orientation as given by VRONI, but the particular orientation can be extracted from the context.

Algorithm 1: Collect nodes

```
input  : list of vertices p
output : list of nodes
edge ← getVoronoiDiagramEdge( p0p1 ) ;        // use VRONI
repeat
    edge ← next edge in Voronoi cell of p0p1;
until edge starts in p0 and edge points into the given direction;
for i ← 1 to length(p) do
    repeat
        add start node of edge to output;
        edge ← next edge in Voronoi cell of pi−1pi;
    until edge ends in pi;
    reverse orientation of e;
    if pi has tolerance zone on given side and i < length(p) − 1
    then
        repeat
            add start node of edge to output;
            edge ← next edge in Voronoi cell of pi;
        until edge ends in pi;
        reverse orientation of e;
```

4.2.2 *Skip Nodes outside Tolerance Zone*

In the next step, we make use of the parameters of the nodes in the list to skip all those nodes that leave the conventional offset d, defined as either $|d_l|$ or $|d_r|$ depending on the current computation. We run through the list, until we find a pair of consecutive nodes n_i, n_{i+1} with parameters $p(n_i) < d < p(n_{i+1})$. We can conclude that between nodes n_i and n_{i+1} the path formed by the list of nodes leaves the tolerance zone.

As the node n_i lies within the tolerance zone and n_{i+1} lies outside and, furthermore, every two consecutive nodes in the list are connected by a Voronoi edge, we use VRONI to compute the exact center p of the clearance disk with radius d on the Voronoi edge between n_i and n_{i+1}. The evaluation of the bisector at a given parameter is handled by VRONI and can be carried out in $O(1)$. The resulting point p lies exactly on the border of the tolerance zone and is inserted into the list between nodes n_i and n_{i+1}. We can conclude that the following nodes starting with the former node n_{i+1} lie outside the tolerance

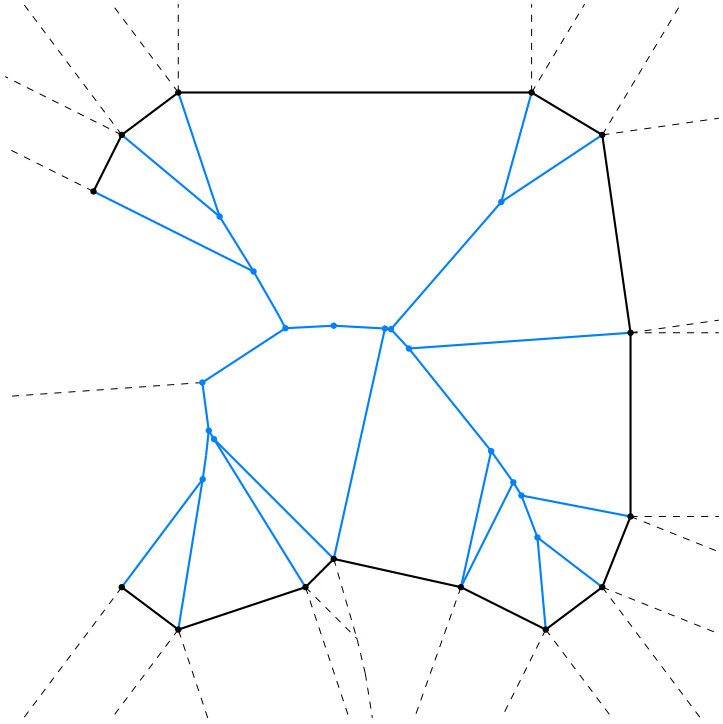


Figure 19: The list after Algorithm 1: All nodes of Voronoi cells on one side of the input get lined up in a list.

zone. Thus, we raise a boolean flag `outside` to indicate that we are currently outside of the tolerance zone.

We move on and remove all following nodes from the list, until we find a pair of consecutive nodes n_j and n_{ji+1} for which the parameters $p(n_j) > d$ and $p(n_{j+1}) < d$. Again, we evaluate the Voronoi edge data and compute the position of the point that lies exactly on the border of the tolerance zone. The point is inserted into the list and as the nodes following n_{j+1} are contained within the tolerance zone, we drop the boolean flag `outside` to indicate that we have returned and do not remove the following nodes from the list.

We continue this procedure, until we reach the end of the list. For the resulting list of nodes $d(n, P) \leq d$ holds for any node n in the list.

Note that in Algorithm 2 the nodes added to the list in lines marked `1` and `2` do not necessarily coincide with a node of the Voronoi diagram, but rather lie on the edge defined by the previous and the current node in the list.

4.2.3 *Removing Trees*

In the previous step, we have used the parameters of the Voronoi nodes to remove all nodes of Voronoi cells that leave the tolerance zone of the polygon chain. The result is the border of the tolerance zone with additional Voronoi edges attached within. In the third step,

Algorithm 2: Skip nodes

```
input  : list of nodes, tolerance d
output : filtered list of nodes

outside ← false;
for v ∈ input do
    p ← getNodeParameter(v) ;                    // use VRONI
    if outside then
        if p < d then      // returning into tolerance zone
            outside ← false;
1           insert(node with parameter d);
        else
            erase(v);
    else
        if p > d then          // leaving the tolerance zone
            outside ← true;
2           insert(node with parameter d);
            erase(v);
```

we clean out all those nodes that do not define the boundary. These structures attached to the border form trees, so we can iterate over the list and remove all leaves as we pass by, ultimately removing the entire trees.

In the list, leaves are stored as triples (n_i, n_{i+1}, n_{i+2}) of nodes for which $n_i = n_{i+2}$, i. e., the first and the third node have the same position. Thus, we scan for such constellations of consecutive nodes and remove them from the list. If our current set of three nodes does not allow any deletion, we move on.

Algorithm 3: Remove trees

```
input  : list of nodes
output : final list of boundary nodes

for i ← 1 to length(input)−2 do
    if node_{i−1} = node_i then      // remove coinciding nodes
        erase(node_i);
        i ← i − 1;
    if node_{i−1} = node_{i+1} then              // remove leaves
        erase(node_i);
        i ← i − 1;
```

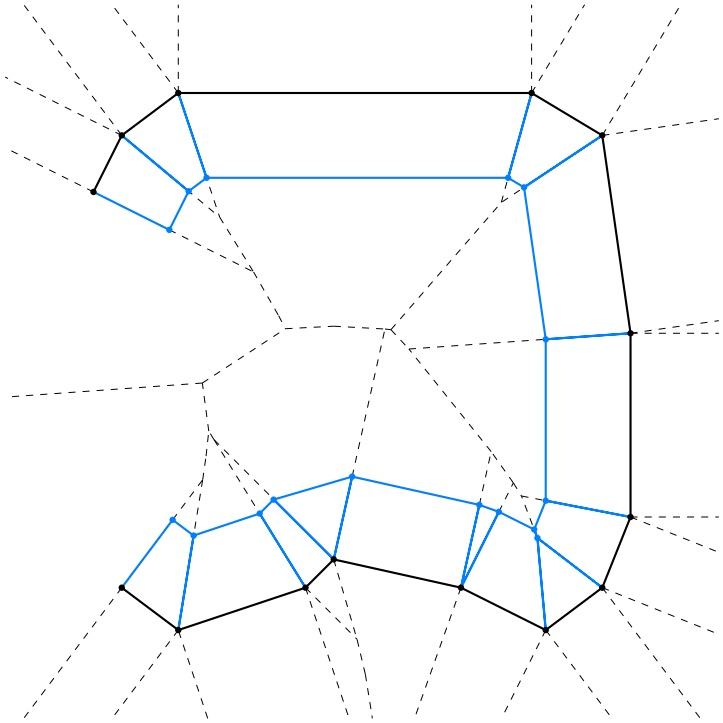


Figure 20: The list after Algorithm 2: The path skips those nodes that exceed the maximum distance to the input.

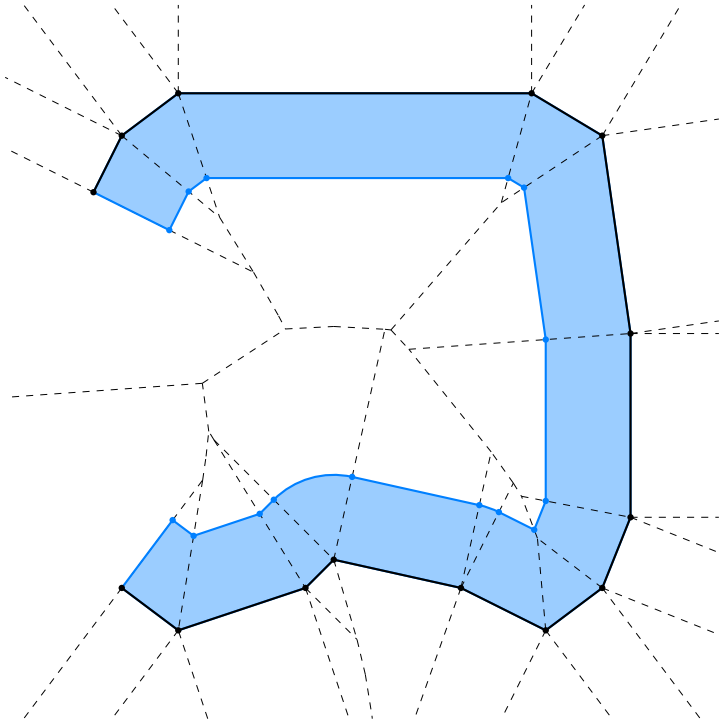
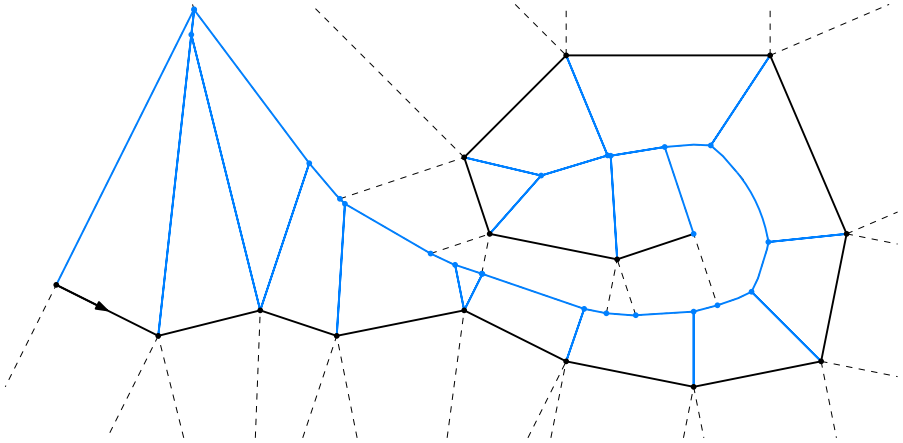
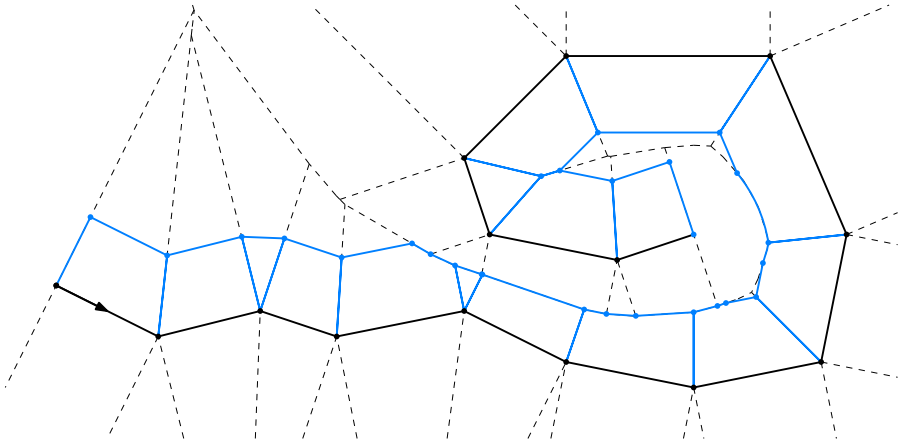


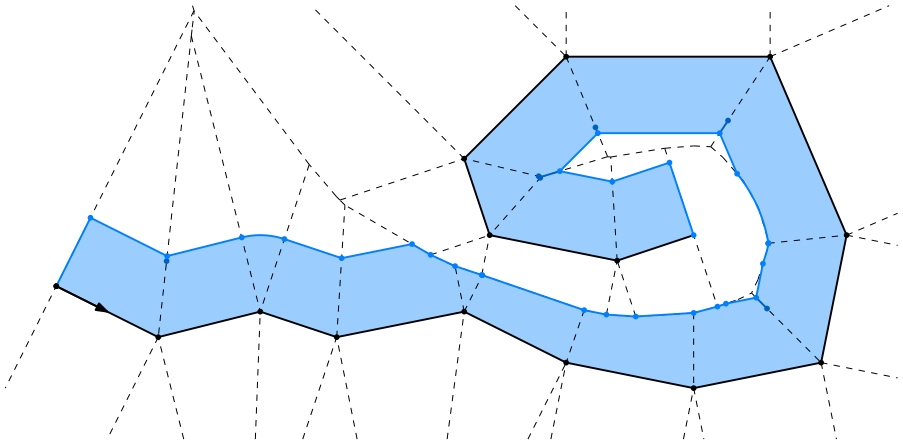
Figure 21: The list after Algorithm 3: Additional Voronoi edges within the tolerance zone have been removed. The one-sided tolerance zone is marked as the colored area, with (sampled) circular arcs added where needed.



(a) Algorithm 1: Collecting all Voronoi nodes



(b) Algorithm 2: Restricting to *conventional offsets*.



(c) Algorithm 3: Removing remaining trees.

Figure 22: This second example shows the construction of the tolerance zone's left boundary with the additional spikes as described in Section 2.2 added in Figure 22c.

4.3 COMPLEXITY ANALYSIS

The algorithm described in the previous section requires the computation of the Voronoi diagram as preprocessing which is in $O(n \log n)$ time and $O(n)$ space. Apart from the preprocessing costs to compute and store the Voronoi diagram, the algorithm for computing the tolerance zone border is in linear time and space.

This can be seen easily, as in the first step we build a list consisting of nodes of the Voronoi diagram. As we build this list, we pass any Voronoi edge twice at most. As the number of Voronoi edges is linear in the number of input sites, we conclude that linear time and space for building and storing this list is not exceeded. So the first step is in $O(n)$. In the second step, we iterate once through the list and remove some nodes, which again is in $O(n)$.

Finally, in the last step we remove the trees attached to the expected result. We iterate through the list and can possibly perform delete operations at every step. The maximum overall number of delete operations is bounded by $n - 2$ and the iteration itself is again an iteration over a list containing a linear number of elements. So this step is as well in $O(n)$.

Therefore, under the assumption of a given Voronoi diagram, we conclude that our algorithm is in linear time and space.

5 APPROXIMATION NODES

One of the fundamental ideas in the approach by Held and Eibl [HE05], and thus also in the work by Heimlich and Held [HH08a], is the computation of so-called *approximation nodes*, abbreviated to A-NODES. These A-NODES lie, speaking informally, in the center of the tolerance zone and are used to find approximation primitives defined upon them. So one of the intrinsics of the algorithms by Held, Eibl, Heimlich, as well as our algorithm is to find approximations that are defined using only these A-NODES.

In the process of finding the approximation, we subsequently try to fit in and add up the longest possible approximation primitives, speaking of *longest* in terms of *passing the maximum number of approximation nodes*. So of course finding proper A-NODES is both a non-trivial as well as a fundamental task. For his work, Heimlich has proposed two different ways of computing a list of approximation nodes, one is based on the work by Eibl and the second utilizes Voronoi diagrams computed by VRONI.

For the T-part approach as proposed by Held and Eibl [HE05], A-NODES are added for every input vertex on those edges of T-part primitives that are incident to the vertex. The distance of the A-NODE to the vertex is computed as the arithmetic mean of the T-parts' heights on the left and on the right side. As a result, for any input vertex, exactly one or three A-NODES are generated. Additionally, tangent vectors are added to the A-NODES. These tangent vectors are perpendicular to the edge which the A-NODE lies on and point in the same direction as the input.

Additionally, Held and Eibl propose a smoothing in combination with a sampling of the A-NODES, such that the maximum distance between two consecutive nodes does not exceed a predefined value. From the work by Heimlich, a value of a 10th of the width of the tolerance zone can be obtained.

Nevertheless, the choice of the loci of these A-NODES is rather arbitrary, especially with respect to the actual approximation primitives used, as this computation of A-NODES does not take any characteristics of the chosen approximation primitive into account. This can dramatically be demonstrated when using straight line segments as approximation primitives, i. e., approximating a polygonal chain with another one. We consider a regular n-gon that we want to approximate using a symmetric tolerance zone. Following the previously described construction, we can conclude that the resulting A-NODES lie exactly on the input vertices. As all approximation primitives, i. e.,

resulting straight line segments, are required to start and end in such an A-NODE and, furthermore, lie within the tolerance zone, they are confined to the area resulting from the intersection of the n-gon with its tolerance zone. Thus, the region for valid approximation primitives is reduced to less than its half. This coincides with an increased amount of approximation primitives required to build a valid approximation.

An example of such an n-gon, an Octagon, can be seen in Figure 23. The A-NODES are too restrictive and thus allow an approximation only within the inner area of the tolerance zone. As a result, we get four approximation primitives, whereas an arbitrary choice of the A-NODES allows an approximation with only three primitives. To give a proof, consider the circumcircle of the inner area and the inscribed circle of the approximation zone. For this example, the radii were chosen to be of ratio 1 : 2 and it can easily be observed that no triangle can be inscribed into such a ring and thus the approximation must consist of at least four A-NODES, one more than the optimal triangle shown dashed in the figure. This problem also extends to splines defined upon this set of A-NODES, when using the triangle and the quadrangle as control points.

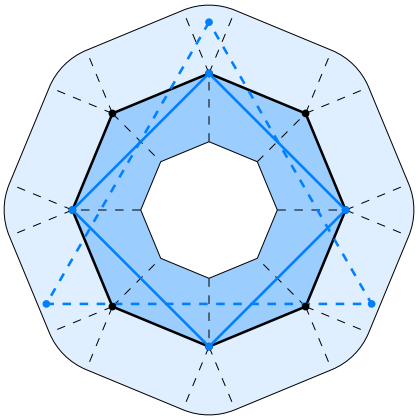


Figure 23: An Octagon with tolerance zone, shaded blue and inner area shaded dark blue.

This problem can be extended to more sophisticated approximation primitives, e. g., spline curves which we use to achieve a G2 continuous approximation. These primitives do not only require a start and an end node, but rather are defined by four anchor nodes. As described in the following chapter, Section 6.2, one can conclude that these splines are entirely contained within the convex hull defined by their four control points, but in general neither the start nor the end point of such a spline segment lie exactly on an A-NODE, as they are *pulled* towards their anchor nodes rather than traveling through them. The placement of A-NODES roughly in the middle of the tolerance zone does not seem to be the best choice.

5.1 MEDIAL AXIS COMPUTATION

The second approach proposed by Heimlich [Hei06] describes the use of the Voronoi diagram to compute the medial axis of the tolerance zone. The medial axis of a polygon, as first described by Harry Blum in 1967 [Blu67], is known under a variety of names, such as *symmetric axis* or *grassfire transform*. The later term describes the original idea by Blum in an intuitive way: We ignite all points on the border of the polygon simultaneously. Assuming a constant burning speed, we track the points where the flames meet. These quench points form the medial axis of the polygon.

Formally, the medial axis can be described as the set of points p in the interior of the polygon, such that there are at least two points, *footprints*, on the boundary of the polygon that are equidistant to p and are closest to p [Lee82]. When examining and decomposing this definition, we get the following properties for points on the medial axis:

- All points p, such that there are at least two points on the boundary that are equidistant to p, describes the set of all points on bisectors of the input polygon.
- If these footprints have to be closest to p, only those p can be accounted that lie on the Voronoi diagram of the polygon.

Thus, for our polygonal input we can conclude that the medial axis is a subset of the Voronoi diagram.

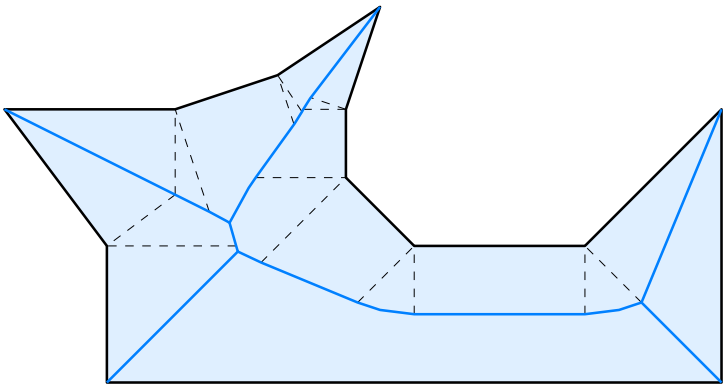


Figure 24: The medial axis of a simple polygon as a subset of the polygon's Voronoi diagram.

More intuitive definitions of the medial axis define it to be the locus of all centers of circles inside the polygon that touch the boundary of the polygon in at least two points [CSW95]. As we have a numerically stable and robust, fast and reliable Voronoi diagram implementation, we can compute the Voronoi diagram of the polygon to filter those edges that belong to the medial axis, using VRONI. As already mentioned in the previous Chapter 3, the generalized Voronoi diagram of

straight line segments can be computed in $O(n \log n)$ time, and the computation using VRONI yields an asymptotic runtime complexity of $O(n \log n)$ expected time, following the random incremental construction approach. Nevertheless, there do exist algorithms that are capable of computing the medial axis in linear time, e. g. the algorithm described in [CSW95].

An important observation is that the number of Voronoi nodes on the medial axis is linear in the number of input vertices. This can easily be concluded when examining the steps that lead to the computation of the medial axis. As discussed in Chapter 4, the number of vertices on the tolerance zone boundary is linear in the number of input vertices. Thus, the input for our second Voronoi diagram computation is in $O(n)$ which again yields a Voronoi diagram with an overall number of nodes (and edges) linear in the input. Thus, we can conclude that the medial axis, as a subset of the Voronoi diagram, is in $O(n)$ with n the number of input vertices.

The medial axis was originally intended as a simpler representation of complex geometric objects in the field of pattern recognition. Associated with information about the defining radii, the medial axis can be used to reconstruct the underlying object. For simple polygons, it can be seen as a tree, rooted in a Voronoi node in the interior of the polygon with the polygon's vertices as leaves. As the medial axis is a subset of the Voronoi diagram, its edges consist of straight line segments and parabolic arcs [Lee82].

We follow the approach by Heimlich and Held [HHo8a] and place an A-NODE on every Voronoi node of the medial axis as defined above. For the computation of the medial axis, we keep track of the input sites, as they can easily be classified into sites belonging to the left and to the right part of the tolerance zone boundary. Thus, we only take those Voronoi edges into account that belong to sites of different classes, one of the left and the other of the right boundary.

As illustrated in Figure 26, this approach of placing A-NODES directly onto nodes of the medial axis may cause a significant increase in the number of approximation primitives. The input polygon consisting of a given number of n ridges on top could be approximated by a single rectangle, resulting in exactly four output segments. When using only approximation nodes on the medial axis, the number of resulting line segments is linear in the number of input vertices. Thus, when extending the input by adding more jagged vertices at the top, the number of output primitives lies in $\Omega(n)$.

This problem can be attenuated by adding sampled A-NODES on the medial axis. Nevertheless, by narrowing the tolerance band, an arbitrary large number of these sampled nodes may be required.

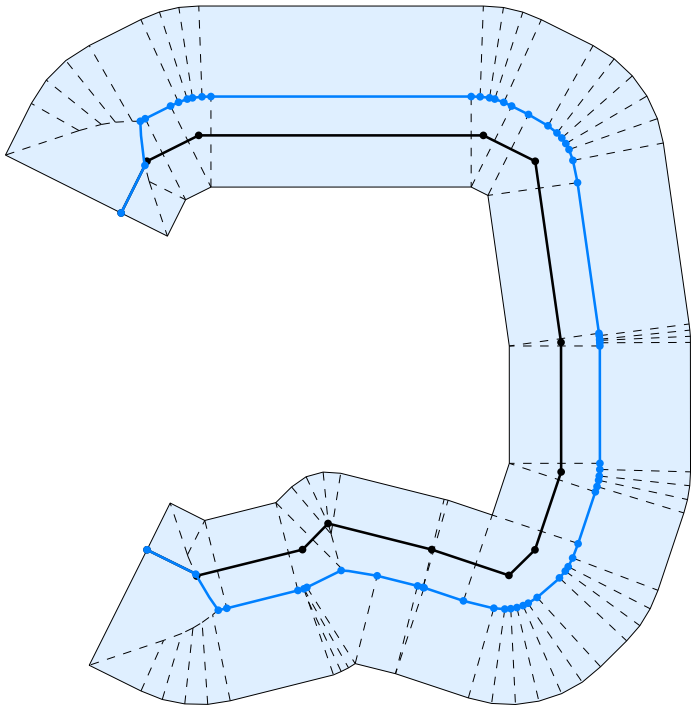


Figure 25: The A-NODES are placed on those nodes belonging to the medial axis that are incident to edges defined by one site of the left and one site of the right boundary.

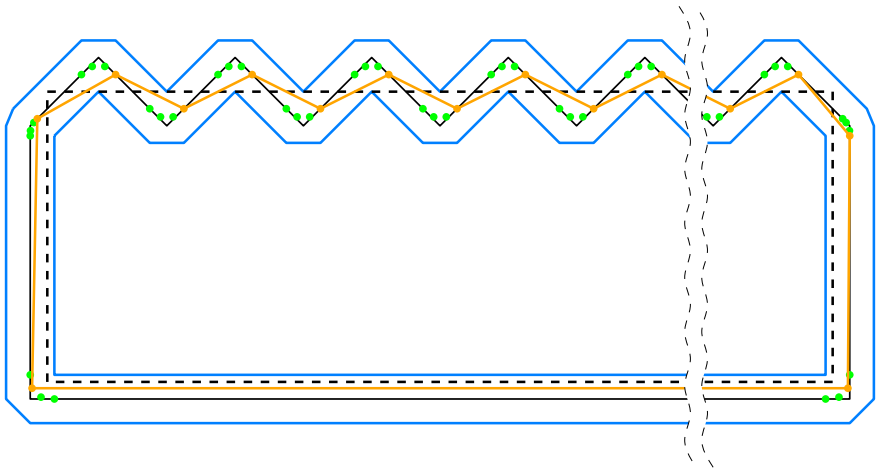


Figure 26: A suboptimal choice of A-NODES may result in a number of output segments in $\Omega(n)$ even if an approximation with a constant number of segments is possible.
The dashed line represents such an optimal approximation, whereas the orange approximation defined only on the green A-NODES requires a linear number of segments.

6

APPROXIMATION PRIMITIVES

6.1 BÉZIER CURVES

6.1.1 *Bernstein Polynomials*

Polynomials and polynomial curves are wide spread in the environment of computational models of scientific and engineering problems. Especially their ability to approximate functions that do not have a closed-form expression motivated the introduction of the Bernstein basis to formulate *well-behaved* polynomials [Far12, p. 382].

Bernstein Basis Polynomials

The *Bernstein basis polynomials* of a given degree n, named after Sergei Natanovich Bernstein, form a basis of the vector space of polynomials $\mathbb{R}[x]$ of degree up to n. These $n+1$ basis polynomials $b_{k,n}$ of degree n are defined as

$$b_{k,n}(x) = \binom{n}{k} x^k (1-x)^{n-k}, \qquad k = 0, 1, \ldots, n$$

where $\binom{n}{k}$ denotes a binomial coefficient, defined as

$$\binom{n}{k} := \frac{n!}{k!(n-k)!} \ .$$

The Bernstein basis polynomials up to degree $n = 5$ can be seen in Figure 27. We use the convention to set $b_{i,n} = 0$ for any $i < 0$ or $i > n$.

One can show that the Bernstein basis polynomials both are linearly independent and span the space of polynomials of a given maximum degree $\deg(\mathcal{P}) < n$. Thus, one can conclude that the Bernstein basis polynomials actually form a basis of the space of polynomials.

The power basis consisting of the polynomials $\{x^i : i \in \mathbb{N}_0,\ i < n\}$ is known to form a valid basis in this space of polynomials. Thus, it can be maintained that the power basis spans the space of polynomials. Furthermore, as any polynomial in power basis can be converted into a polynomial in Bernstein basis and vice versa, the Bernstein polynomials do indeed span the polynomial space.

To show that the Bernstein basis polynomials are linearly independent, one can consider a linear combination of $n+1$ Bernstein basis polynomials and show that for all x

$$0 = \alpha_0 b_{n,0}(x) + \alpha_1 b_{n,1}(x) + \ldots + \alpha_n b_{n,n}(x)$$

holds if and only if all $\alpha_i = 0$ for $i = 0, \ldots, n$.

By applying the definition of the Bernstein basis polynomials, one can conclude that if the linear combination of Bernstein basis polynomials yields zero, one gets

$$0 = \alpha_0 \binom{n}{0} x^0 (1-x)^n + \alpha_1 \binom{n}{1} x^1 (1-x)^{n-1} + \ldots + \alpha_1 \binom{n}{n} x^n (1-x)^0$$

and by collecting the corresponding terms for each power of x, one can write

$$\begin{aligned} 0 =& \alpha_0 \sum_{i=0}^{n} (-1)^i \binom{n}{i} \binom{i}{0} x^i + \alpha_1 \sum_{i=1}^{n} (-1)^i \binom{n}{i} \binom{i}{1} x^i + \ldots \\ &+ \alpha_n \sum_{i=n}^{n} (-1)^i \binom{n}{i} \binom{i}{n} x^i \\ =& \left[\sum_{i=0}^{0} \alpha_i \binom{n}{0} \binom{0}{0} \right] x^0 + \left[\sum_{i=0}^{1} \alpha_i \binom{n}{1} \binom{1}{1} \right] x^1 + \ldots \\ &+ \left[\sum_{i=0}^{n} \alpha_i \binom{n}{n} \binom{n}{n} \right] x^n \end{aligned}$$

which finally gives an expression of type

$$0 = \mu_0 t^0 + \mu_1 t^1 + \ldots + \mu_n t^n \ .$$

From the fact that the polynomials in power form define a basis of the space of polynomials, it can be deduced that $\mu_i = 0$ for all $i = 0, \ldots, n$. Thus, for the values of α_i one gets

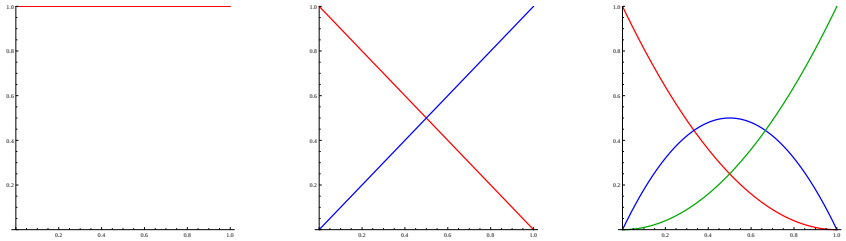
$$\begin{aligned} \sum_{i=0}^{0} \alpha_i \binom{n}{0} \binom{0}{0} &= \alpha_0 && = 0 \\ \sum_{i=0}^{1} \alpha_i \binom{n}{1} \binom{1}{1} &= \alpha_0 * n + \alpha_1 * n && = 0 \\ &\vdots \\ \sum_{i=0}^{n} \alpha_i \binom{n}{n} \binom{n}{n} &= \ldots && = 0 \end{aligned}$$

Beginning from the first line, $\alpha_0 = 0$ and subsequently inserting the previous result into the next line finally gives that $\alpha_i = 0$ for all $i = 0, \ldots, n$ and thus the Bernstein basis polynomials are linearly independent.

Bernstein Polynomials

A *Bernstein polynomial* $B(x)$ of degree n is a linear combination of Bernstein basis polynomials, i. e.,

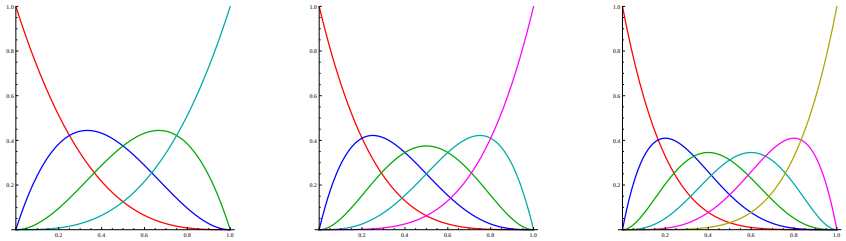
$$B(x) = \sum_{k=0}^{n} b_k b_{k,n}(x) \qquad \text{with } b_k \in \mathbb{R} \ .$$



(a) $b_{0,0}(x) = 1$

(b) $b_{0,1}(x) = 1 - x$
$b_{1,1}(x) = x$

(c) $b_{0,2}(x) = (1-x)^2$
$b_{1,2}(x) = 2x(1-x)$
$b_{2,2}(x) = x^2$



(d) $b_{0,3}(x) = (1-x)^3$
$b_{1,3}(x) = 3x(1-x)^2$
$b_{2,3}(x) = 3x^2(1-x)$
$b_{3,3}(x) = x^3$

(e) $b_{0,4}(x) = (1-x)^4$
$b_{1,4}(x) = 4x(1-x)^3$
$b_{2,4}(x) = 6x^2(1-x)^2$
$b_{3,4}(x) = 4x^3(1-x)$
$b_{4,4}(x) = x^4$

(f) $b_{0,5}(x) = (1-x)^5$
$b_{1,5}(x) = 5x(1-x)^4$
$b_{2,5}(x) = 10x^2(1-x)^3$
$b_{3,5}(x) = 10x^3(1-x)^2$
$b_{4,5}(x) = 5x^4(1-x)$
$b_{5,5}(x) = x^5$

Figure 27: Sample Bernstein basis polynomials $b_{n,k} = \binom{n}{k} x^k (1-x)^{n-k}$, of degree $n = 0, \ldots, 5$ in the unit square $[0,1]^2$

Bernstein used these polynomials in the early 20th century to give a constructive proof [Ber13] of the Stone-Weierstrass approximation theorem [Wei85a], [Wei85b]. This fundamental theorem was first proven by Weierstrass (German: Weierstraß) in 1855 and states that every real- or even complex-valued continuous function can be uniformly approximated by polynomial functions over $\mathbb{R}$. Additional details including further proofs of the Weierstrass theorem can be obtained from reference [Lor86] and more recently [Far12].

Bernstein Approximation

For a given continuous function f on the interval $[0,1]$, the *Bernstein approximation* is defined as

$$B_n(f)(x) = \sum_{k=0}^{n} f\left(\frac{k}{n}\right) b_{k,n}(x) \ .$$

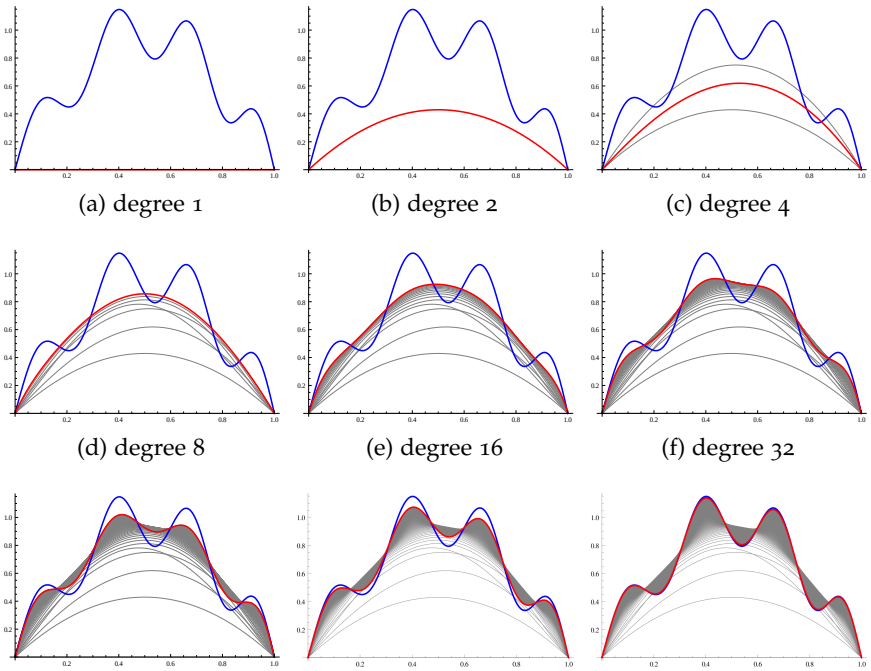
A sample Bernstein approximation of a continuous function over the unit interval with degrees up to 1024 can be seen in Figure 28. This approximation can be shown to converge uniformly to the function f on the interval $[0,1]$ and thus it can be shown that the space of polynomial functions, i. e., the metric space of polynomials on the closed interval $[a,b]$ together with the supremum as a norm, is *dense*.

It must be observed that the Bernstein approximation converges very slowly, i. e., the rate of convergence is in $O(1/n)$ for all functions that are twice differentiable [Phio3, p. 251]. This is basically known as the *Voronovskaya Theorem* [Col75]:

If f is twice differentiable on $[0,1]$, then, for all $x \in [0,1]$,

$$\lim_{n\to\infty} n\big(f(x) - B_n(f,x)\big) = -\frac{1}{2}x(1-x)f''(x) \ .$$

This notoriously slow rate of convergence counterbalances many of the otherwise beneficial properties of the Bernstein approximation and thus prevents wide-spread practical use [GR74b, p. 293].



(a) degree 1 (b) degree 2 (c) degree 4

(d) degree 8 (e) degree 16 (f) degree 32

(g) degree 64 (h) degree 128 (i) degree 1024

Figure 28: Bernstein approximation over the unit interval of the continuous function $f(x) = \sin(\pi x) + \frac{1}{5}\sin(6\pi x + \pi x^2)$ with approximation degrees 1 to 1024

Further Properties of Bernstein Polynomials

As Bernstein basis polynomials are defined upon binomial coefficients, some of the neat properties of binomial coefficients can be propagated to Bernstein basis polynomials. As an example, using the property of binomial coefficients that

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$$

it can be shown that a Bernstein basis polynomial $b_{k,n}(x)$ of degree n can be written as the sum of two basis polynomials $b_{k,n-1}$ of degree $n-1$. The following recursive definition

$$b_{k,n}(x) = b_{k-1,n-1}(x)\, x + b_{k,n-1}(x)\,(1-x)$$

can be shown using the above property, the definition of Bernstein basis polynomials and some simple algebra:

$$\begin{aligned}
b_{k,n}(x) &= \binom{n}{k} x^k\,(1-x)^{n-k} \\
&= \left[\binom{n-1}{k-1} + \binom{n-1}{k}\right] x^k\,(1-x)^{n-k} \\
&= \binom{n-1}{k-1} x^k\,(1-x)^{n-k} + \binom{n-1}{k} x^k\,(1-x)^{n-k} \\
&= x\binom{n-1}{k-1} x^{k-1}\,(1-x)^{n-k} + (1-x)\binom{n-1}{k} x^k\,(1-x)^{(n-1)-k} \\
&= x\, b_{k-1,n-1}(x) + (1-x)\, b_{k,n-1}(x)
\end{aligned}$$

A fundamental property is that over the unit interval all Bernstein basis polynomials are non-negative, i. e.,

$$\forall x \in [0,1] :\ b_{k,n}(x) \geq 0\ .$$

This can be shown using a well-founded induction, using the lexicographical order of $(k,n) \subseteq \mathbb{N}^0 \times \mathbb{N}^0$ as well-founded relation. The base cases can be easily handled, since

$$b_{0,n} = (1-x)^n \geq 0$$

and thus for the induction step we get

$$b_{k,n}(x) = \underbrace{b_{k-1,n-1}(x)}_{\substack{\text{IH}\\ \geq 0}}\ \underbrace{x}_{\geq 0 \forall x \in [0,1]} + \underbrace{b_{k,n-1}(x)}_{\substack{\text{IH}\\ \geq 0}}\ \underbrace{(1-x)}_{\geq 0 \forall x \in [0,1]} \geq 0\ .$$

Similarly, one can conduct a well-founded induction to show that all Bernstein basis polynomials yield positive values over the open unit interval $(0,1)$, i. e.,

$$\forall x \in (0,1)\ \ b_{k,n}(x) > 0\ .$$

Another important property is that Bernstein basis polynomials form a partition of unity, i. e., $b_{k,n}(t)$ sum up to 1 for all values of t.

$$\forall n \in \mathbb{N}\ \ \sum_{k=0}^{n} b_{k,n}(x) = 1$$

To give a proof for above property, one first shows that the sum of the $n+1$ Bernstein basis polynomials of degree n at any point $x \in [0,1]$

equals the sum of the n Bernstein basis polynomials of degree $n-1$, again at any point $x \in [0,1]$.

$$\sum_{k=0}^{n} b_{k,n}(x) = \sum_{k=0}^{n-1} b_{k,n-1}(x)$$

This can be shown using the previously given recursive definition and applying some basic algebra, most notably shifting the index. For this proof, one requires the convention that $\forall i < 0, i > n \quad b_{i,n} = 0$.

$$\begin{aligned}
\sum_{k=0}^{n} b_{k,n}(x) &= \sum_{k=0}^{n} \left[x\, b_{k-1,n-1}(x) + (1-x)\, b_{k,n-1}(x) \right] \\
&= x \sum_{k=1}^{n} b_{k-1,n-1}(x) + (1-x) \sum_{k=0}^{n-1} b_{k,n-1}(x) \\
&= x \sum_{k=1}^{n} b_{k-1,n-1}(x) + \sum_{k=0}^{n-1} b_{k,n-1}(x) - x \sum_{k=0}^{n-1} b_{k,n-1}(x) \\
&= x \sum_{k=1}^{n} b_{k-1,n-1}(x) + \sum_{k=0}^{n-1} b_{k,n-1}(x) - x \sum_{k=1}^{n} b_{k-1,n-1}(x) \\
&= \sum_{k=0}^{n-1} b_{k,n-1}(x)
\end{aligned}$$

Using this intermediate result, one can conclude by induction that indeed for any $n \in \mathbb{N}$

$$\sum_{k=0}^{n} b_{k,n}(x) = 1$$

holds. The induction step has already been shown, as the sum of degree n polynomials equals the sum of degree $n-1$ polynomials at any point $x \in [0,1]$. The base case is simple, as for $n = 0$ the sum reduces to only one term[1], for which

$$\sum_{k=0}^{0} b_{k,0}(x) = b_{0,0} = \binom{0}{0} x^0 (1-x)^0 = 1 \ .$$

This beyond the field of geometric modeling interesting and important property allows the use of approximation primitives based on Bernstein (basis) polynomials, as for any set of n points $P_0, P_1, \ldots, P_n$ with given position vectors $p_1, p_2, \ldots, p_n$ in $\mathbb{R}^2$, the curve obtained by the parametrization

$$\begin{aligned}
&\gamma : [0,1] \to \mathbb{R}^2 \\
&\gamma(t) = p_0 b_{0,n}(t) + p_1 b_{1,n}(t) + \ldots + p_n b_{n,n}(t)
\end{aligned}$$

1 For $x = 0$ or $x = 1$, this term contains the expression 0^0. In this thesis, we use the convention that $0^0 = 1$. See [Knu92] for details and additional references on 0^0.

forms a convex combination of the points P_0 to P_n.

Especially in the field of computer aided geometric design, a lot of research has been conducted to achieve curve (primitives) suitable for many needs, including generalizations [Phi96] and different types of (Bernstein) polynomials, e. g., the q-Bernstein polynomials [OP03]. This might also be a consequence of the fact that the Bernstein basis on a given interval is *optimally stable* [FG96]. For polynomial evaluation, the Bernstein form on the given interval is more stable than the power form [FR87], i. e., the condition numbers are systematically smaller, and thus many applications yield numerically more stable results [FG96, p. 1566], [Far12, p. 396].

> The intrinsic stability of the adopted representation scheme is an important issue that can profoundly influence the accuracy and reliability of various calculations on parametric curves and surfaces [FG96, p. 1566].

The Gibbs Phenomenon

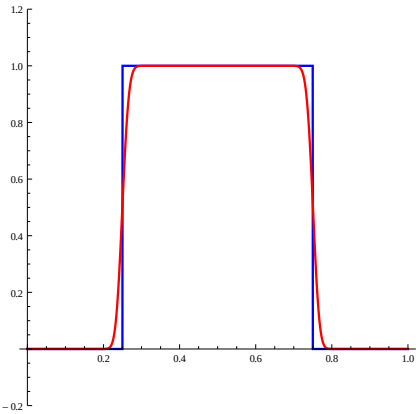
Another interesting result concerning the Bernstein approximation is the absence of the *Gibbs phenomenon* [GP03, p. 7]. The Gibbs phenomenon is an effect which can be seen when the Fourier approximation (based on Fourier series) for non-periodic or discontinuous functions is studied. An example of the Gibbs phenomenon of a Fourier series approximation in comparison to a Bernstein approximation can be seen in Figure 29. It describes the *overshoot* which can be seen in the approximation close to jumps (points of discontinuities) or in the non-periodic case, at the period boundaries. If we are given the Fourier coefficients $\hat{f}_k$, we build the Fourier series as

$$f_n(x) = \sum_{k=-n}^{n} \hat{f}_k \, e^{ik\pi x} \quad .$$

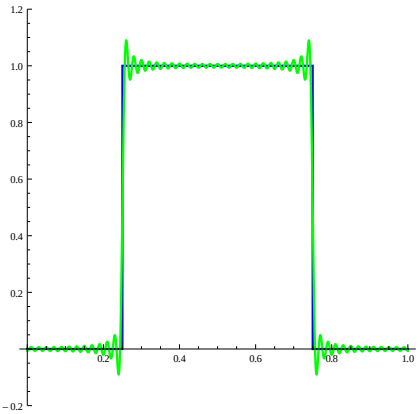
The Gibbs phenomenon, first observed by Henry Wilbraham in 1848 [Wil48, p. 200] and later analyzed by Josiah Williard Gibbs in 1899 [Gib98][Gib99], describes the fact that

$$\sup_{x \in [-1,1]} |f(x) - f_n(x)|$$

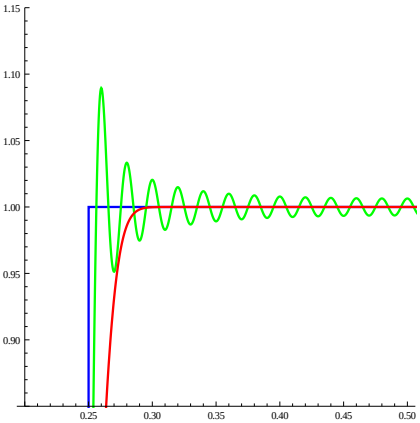
does not tend to zero [GS97, p. 645]. For additional details on the Gibbs phenomenon, see references [Wil48], [Gib99], [HH79], [GS97] and [Jer11].



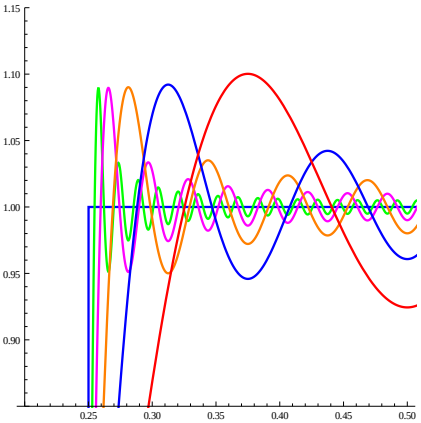
(a) Bernstein approximation of degree 1024 without Gibbs phenomenon



(b) Fourier series approximation of degree 50, *overshoot* clearly visible



(c) comparison of Bernstein and Fourier series approximation at the point of discontinuity



(d) Comparison of different degrees (2^i, $i = 2, \ldots, 6$) of Fourier series approximations

Figure 29: The 1-periodic function $f(x)$ with Bernstein and Fourier series approximations. On the unit interval, $f(x)$ is defined as

$$f(x) = \begin{cases} 1 & x \in [0.25, 0.75] \\ 0 & \text{otherwise} \end{cases}.$$

6.1.2 *Bézier Curves*

Polynomials and polynomial curves in power basis, i. e., curves with polynomials as coordinate functions, are well understood and easy to handle from a mathematical point of view. Still, they are only of little interest to designers, as they are not suitable for artists with little mathematical background as an instrument to define shapes and curves.

Several reasons account for this fact, most notably the problem that the coefficients of a polynomial in power basis give only little insight about the actual geometric shape of the curve [Sed11, p. 17]. Additionally, polynomials in power basis are most likely to suffer from numeric problems. For efficient evaluation, e. g., by using Horner's method, cancellation can possibly occur at any step and the overall numerical error is difficult to determine a-priori [Hig02].

For these reasons the French physicist and mathematician Paul de Faget de Casteljau, working for the Citroën car company, and Pierre Bézier, an engineer at the Renault car company, independently developed curve definitions intuitive enough to be used productively by designers without requiring a lot of mathematical training. These mathematical inventions where pioneered and accompanied by the invention of hardware, capable of visualizing complex sets of geometric objects, e. g., the *Sketchpad* system by Ivan E. Sutherland [Sut64].

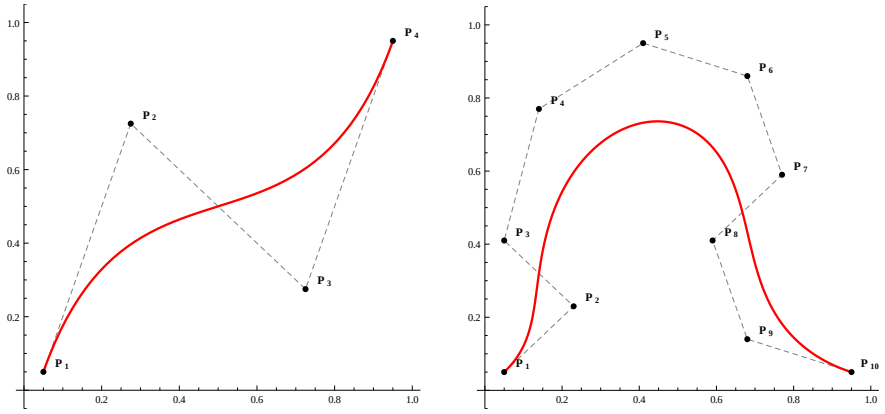
At Citroën, the invention of and development on these curves was conducted secretly by de Casteljau. He focused on creating a mathematical system which could be used at the state of designing. Previously, a lot of effort had to be made in adopting existing designs and blueprints to a suitable mathematical representation. De Casteljau's Bézier curves were recursively defined, which led to the development of the de-Casteljau-algorithm. His main contribution was the use of *control polygons*, instead of defining the curve by a set of points *on* it [Far02, p. 4-6]. As the work by de Casteljau was kept a secret, it was not before the late 1970s that attention was paid to the technical reports by de Casteljau.

The work by Bézier followed mathematically different ideas. Bézier defined a *basic curve* as an intersection of two elliptic cylinders and used affine transformations on the cylinders resulting in affine transformations of the curve. This concept, in polynomial formulations, turned out to be identical to the approach developed by de Casteljau. Furthermore, the algorithm which later became renowned as the *de Casteljau algorithm* had also been discovered by a member of Bézier's team [Far02, p. 6].

Bézier curves are defined upon control points which are forming the control polygon. The reason why Bézier curves are handy for designing free form shapes is that they *mimic* the shape of their control polygon [Sed11]. Most importantly, the start and end points coincide

with the first and last control point. This is a remarkable property, as for many practical purposes the designer is required to have direct control over these two important points [Far96, p. 37].

Additionally, Bézier curves are invariant under affine transformations, e. g., scaling, rotating, translation, etc. This means that first applying an affine transformation to the control polygon and then computing points on the Bézier curve yields the same result as first computing the points on the curve and then applying the transformations [Far96, p. 36].



(a) A degree 4 Bézier curve

(b) A degree 10 Bézier curve

Figure 30: Two sample Bézier curves with corresponding *control polygons*

In 1972, A. R. Forrest discovered the significance of Bernstein polynomials, as the curves as defined by Casteljau and Bézier can be written in Bernstein form: A Bézier curve of degree n with control points defined by position vectors $p_1, \ldots, p_n$ can be expressed explicitly by Bernstein polynomials [For72], parametrized on the unit interval $t \in [0, 1]$ as

$$B(t) = \sum_{i=0}^{n} p_i b_{i,n}(t) \quad .$$

Although we define Bézier curves on the unit interval, it is also possible to use an arbitrary parameter interval $[a, b]$ by linearly mapping $[a, b]$ onto the unit interval $[0, 1]$ such that $B_{[a,b]}(a) = B(0)$ and $B_{[a,b]}(b) = B(1)$. Unless stated otherwise, we usually define Bézier curves $B(t)$ on the unit interval, whereas Bézier curves on the interval $[a, b]$ are denoted as $B_{[a,b]}(t)$, with equation

$$B_{[a,b]}(t) = \sum_{i=0}^{n} p_i \, b_{i,n}\left(\frac{b-t}{b-a}\right) \quad .$$

The class of functions $b_{i,n}(t)$ is also called *blending functions* and it is obvious that different blending functions allow a huge set of possible curves.

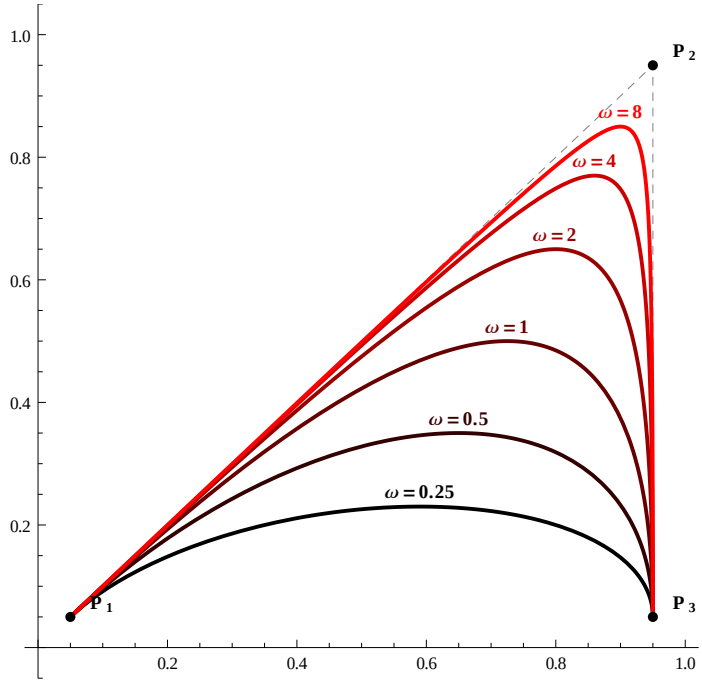


Figure 31: A set of rational Bézier curves with equal weights 1.0 for points P_1, P_3 and $\omega = 2^i, i = -2, \ldots, 3$ for P_2.

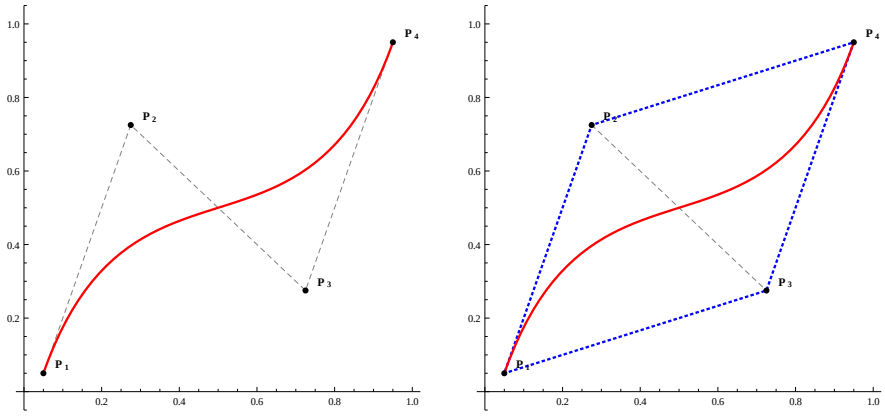
Apart from the possibilities following the use of different blending functions, there do exist generalizations of Bézier curves. One class of these curves consists of the so-called *rational Bézier curves* which have additional scalar weights ω_i attached to every point:

$$R(t) = \frac{\sum_{i=0}^{n} \omega_i p_i b_{i,n}(t)}{\sum_{i=0}^{n} \omega_i b_{i,n}(t)}$$

Rational Bézier curves provide more control over the shape of the curve, see a sample rational Bézier curve in Figure 31. Additionally, they are required to exactly represent all conic sections, as only parabolas can be expressed by polynomial Bézier curves as previously defined [PL02, p. 44]. Furthermore, a perspective projection of a 3D Bézier curve yields a rational Bézier curve [Far83].

The early research by European automotive companies seems to have been conducted unaware of the developments and inventions in aircraft engineering, as James Ferguson had already published early results in 1964 [Fer64]. Further industrial uses of Bézier techniques have been described by Gerald Farin at Daimler-Benz [Far84] and by Hochfeld and Ahlers at Volkswagen [HA90].

One of the most common uses of Bézier curves can be found with font design, e. g., in the graphics language *PostScript*. Individual characters are stored as a sequence of Bézier curves defining their outline, which comes with a great advantage over, e. g., storing the characters' pixel data. As the Bézier curves are invariant under affine transforma-



(a) Bézier curve from Figure 30a (b) convex hull of the control polygon

Figure 32: A Bézier curve lies within the convex hull of its control polygon.

tions, the characters are easily scaled or rotated, which is a common requirement in any text processing system [Far96, p. 120].

6.1.3 *Intersection Tests*

We focus on building an approximation that is entirely contained within our approximation tolerance. Thus we need to come up with a test to check whether a primitive lies within this previously computed tolerance zone. The tolerance zone is represented by a simple polygon sampling its boundary. Instead of running a set of point-in-polygon-tests, e. g., using the ray-casting algorithm [SSS74, p. 13], we rather start with a vertex known to lie within the tolerance zone and check whether the next approximation primitive intersects the tolerance zone boundary. In the case of such an intersection, the approximation primitive leaving the tolerance zone is discarded.

Convex Hull Property

An important property of Bézier curves is that all points of the curve always lie within the convex hull of the defining control polygon. This is an implication of the important fact that the Bernstein basis polynomials are all non-negative and form a partition of unity. These two fundamental properties have been proven in the previous section. A visualization of this property can be seen in Figure 32.

As a direct consequence, to rule out that a given Bézier curve intersects the tolerance zone boundary, we conduct an intersection test with the convex hull of the control polygon which is much faster than testing against the Bézier curve itself. If the test fails, we have to make refinements to our search.

De Casteljau Algorithm

Although de Casteljau's work dates back to 1959, there are only a few technical reports, deposited and attested by a notary in 1963 [BM99]. These technical notes were not easily accessible and thus de Casteljau's work was not noticed until Wolfgang Boehm obtained a copy in 1975 [Far96]. A scan of the original notes has been attached to the paper by Boehm and Müller in 1999, see reference [BM99].

The de Casteljau algorithm runs on a given polygonal chain of length n, the control polygon, which consists of $n+1$ vertices P_k, $k = 0, \ldots, n$ called *control points* or *Bézier points* and n straight line segments. For a given partition of unity t, i. e., $t \in [0, 1]$, subdividing the segments of the control polygon according to t, i. e., at the ratio $t : (1-t)$, gives a new polygonal chain of length $n-1$, consisting of n vertices and $n-1$ segments. Repeating this procedure finally gives a single point, $B(t)$. The trace of this point defined by this algorithm on the control polygon when varying t defines a curve of degree n. It can be shown that this results in the Bézier curve as previously defined.

The subsequently generated points $p_{i,k}$ with i the iteration step and $k = 0, \ldots, i$ can be evaluated recursively as

$$p_{i+1,k} = t\, p_{i,k+1} + (1-t)\, p_{i,k} \quad .$$

These intermediate points can be arranged in the following triangular scheme [BM99]:

$$\begin{array}{ccccccc}
p_{0,0} & & & & & & \\
& p_{1,0} & & & & & \\
p_{0,1} & & \ddots & & & & \\
& & & p_{n-1,0} & & & \\
\vdots & & & & & p_{n,0} & \\
& & & p_{n-1,1} & & & \\
p_{0,n-1} & & \iddots & & & & \\
& p_{1,n-1} & & & & & \\
p_{0,n} & & & & & &
\end{array}$$

In this triangular form, the leftmost column $p_{0,k}$ assembles the input points, i. e., $p_{0,k} = P_k$ whereas the other points are recursively computed, depending on the choice of t. The resulting point $B(t) = p_{n,0}$ can be written using the previously defined Bernstein basis polynomials $b_{n,k}$ as

$$b(t) = \sum_{i=0}^{n} P_i\, b_{n,i}(t)$$

and thus indeed gives the Bézier curve as defined in Section 6.1.2.

The computation of points on the Bézier curve using the de Casteljau Algorithm is both numerically stable [FN90] and sufficiently fast. The storage for the triangular scheme given above is linear in the number of control points, as in every step the previously achieved results can be overwritten [Far96, p. 36]. For a proof that Bézier curves computed by using the de Casteljau algorithm are indeed invariant under affine transformations, see [Gal99, p. 153].

Subdivision Algorithm

The subdivision method can be used to refine the control polygon and achieve a polygonal approximation of the Bézier curve. The main goal is to find two suitable control polygons in such a way that the original curve can be subdivided at any given parameter $t \in [0, 1]$ into two independent consecutive curves. The resulting curves are required to be identical with the original curve, with respect to the partition of the original parameter interval by t. The required control polygons can be easily computed using de Casteljau's algorithm, giving again two Bézier curves as a result. As this process converges fast, it is possible to use consecutive subdivisions to render the curve. It has been proven that the polynomial refinement converges to the curve [CS85] and that the convergence rate is quadratic [Dah86].

To conduct such a subdivision step, we compute the intermediate nodes of the de Casteljau algorithm as previously described. For a control polygon with $n + 1$ control points, this process can be carried out in place, i. e., within a given array of vertices, overwriting the previously generated nodes in every step without requiring additional storage. This can be visualized when arranging the above triangular scheme in the following manner, which allows the next column to be computed from the top down without overwriting any data to be read in a consecutive step.

$$
\begin{array}{ccccc}
p_{0,0} & p_{1,0} & \cdots & p_{n-1,0} & p_{n,0} \\
p_{0,1} & p_{1,1} & \cdots & p_{n-1,1} & \\
\vdots & \vdots & \iddots & & \\
p_{0,n-1} & p_{1,n-1} & & & \\
p_{0,n} & & & &
\end{array}
$$

In this process of evaluating the Bézier curve, we get an array of $n + 1$ vertices, $p_{0,n}, p_{1,n-1}, \ldots, p_{n,0}$. These vertices form the lower border, whereas the vertices $p_{0,0}, p_{1,0}, \ldots, p_{n,0}$ at array index 0, provided that they are properly stored during every step, form the upper border of the triangular scheme. These two polygonal chains of each $n + 1$ vertices form the new control polygon of two Bézier curves which are identical with the original Bézier curve on the parameter intervals $[0, t]$ and $[t, 1]$. These curves are visualized in Figure 33.

Performing Intersection Tests

As the subdivision method returns two Bézier curves, it can be recursively applied, resulting in a polygonal approximation of the Bézier curve. For the proof that the polygonal chain $\mathcal{P}_n(t)$ defined by the nth iteration of the subdivision process converges uniformly to the Bézier curve $B(t)$, i. e.,

$$\lim_{n\to\infty} \sup_{t\in[0,1]} \|\mathcal{P}_n(t) - B(t)\| = 0 \ ,$$

see [Gal99, p. 158]. Additionally, it is possible to compute the Bézier curve for a given parameter interval $[a, b] \subset [0, 1]$, by simply computing the subdivision at parameter a and using subdivision with respect to parameter b on the resulting Bézier curve.

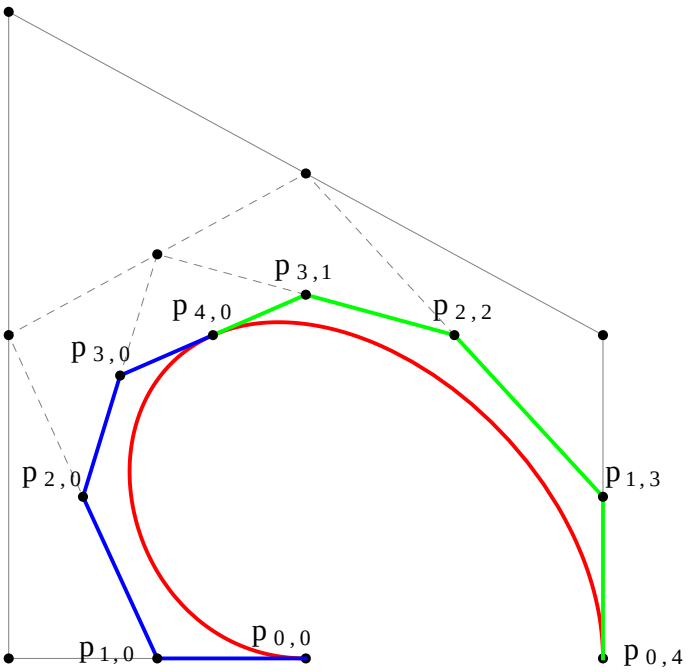


Figure 33: The de Casteljau algorithm used to subdivide a Bézier curve at parameter $t = 0.5$.

As an additional important application, the subdivision method can also be used for intersection testing. Following a binary search approach, the subdivision is subsequently carried out for a given parameter $t = 0.5$. The intersection test can be conducted against the convex hull or, even computationally faster, the axis aligned bounding boxes of the resulting control polygons. If an intersection is reported, the subdivision is carried on with the corresponding division of the curve. The subdivision stops, when the size of the axis aligned bounding box falls below a given intersection tolerance [Han02, p. 81].

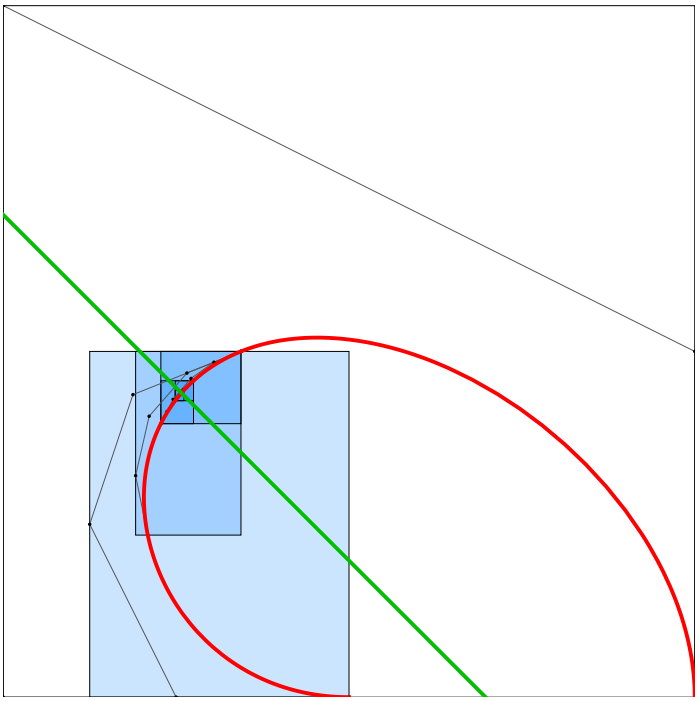
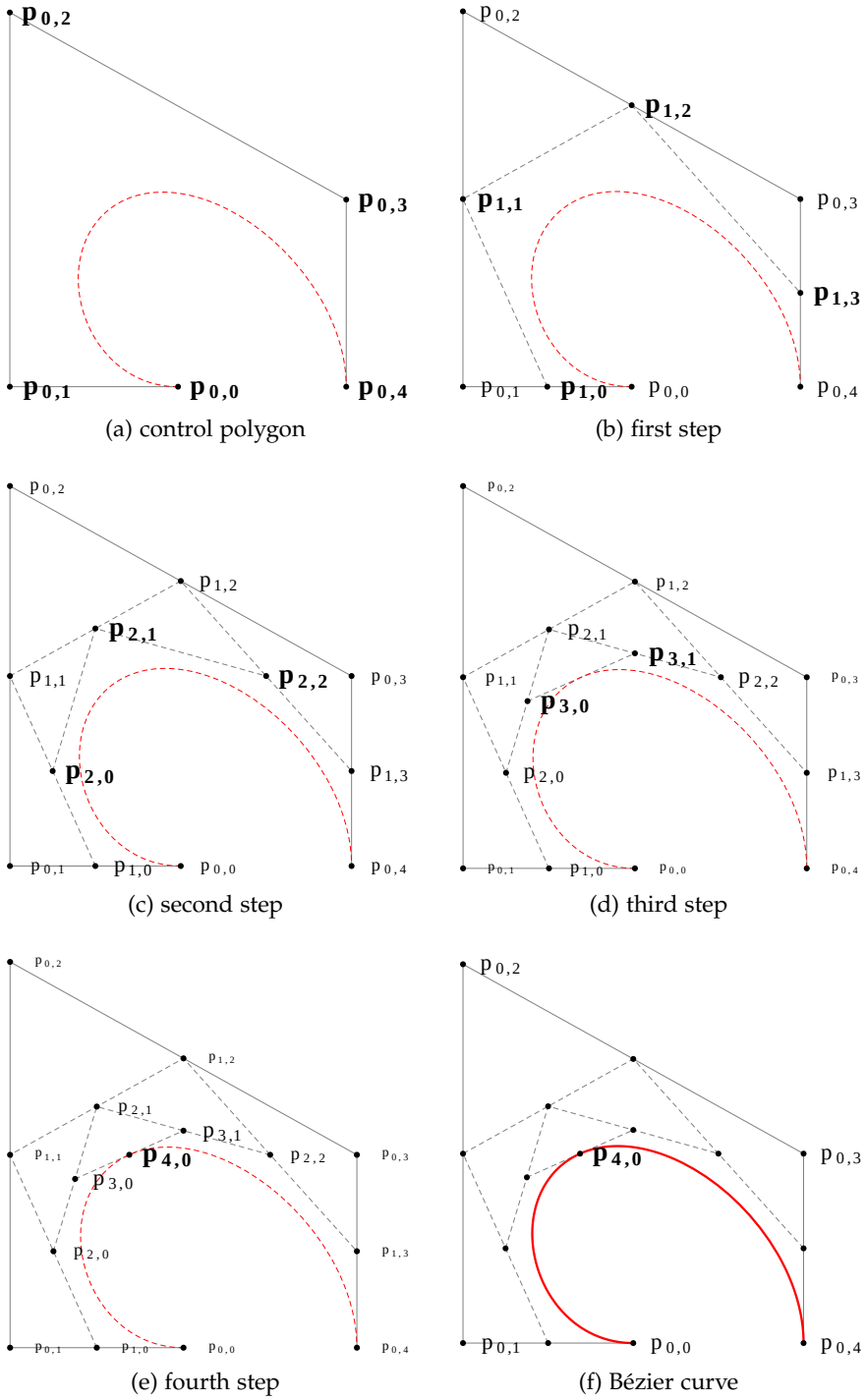


Figure 34: Subsequent subdivisions of a Bézier curve can be used to conduct an intersection test between the Bézier curve and a straight line segment.



(a) control polygon (b) first step

(c) second step (d) third step

(e) fourth step (f) Bézier curve

Figure 35: The de Casteljau algorithm is applied to the input polygon, with $p_{n,k}$ the kth intermediate node after the nth step. The trace of $p_{4,0}$ defines the Bézier curve of degree 4 for the given control polygon of length 4.

6.2 SPLINES

Most shapes are too complex to be represented by a single Bézier curve, and a few additional problems add to the fact that Bézier curves as described in Section 6.1.2 are suboptimal in a number of ways.

1. A control polygon of length n consists of $n+1$ control points. Thus, it takes $n(n+1)/2$ steps to compute a point on the curve. Given a control polygon with a large number of control points, this approach with asymptotic runtime in $O(n^2)$ is too computationally intensive. In addition to the computational inefficiencies, polynomial curves at a high degree are numerically unstable.

2. Bézier curves lack sufficient local control, as moving a control point effects the entire curve. Especially for intersection testing it is desirable that moving a single control point does only effect a small region of the curve, limiting the resulting intersection tests to that specified region.

3. The control points affect the curve, but to define a curve interpolating a shape at a given set of approximation nodes, it is often necessary to solve systems of linear equations. If the degree of the curve is high, this becomes highly impractical.

To overcome these shortcomings, the so-called *splines* have been introduced. The main idea behind splines is to *spline*, i. e., to continuously join a set of shorter parts together, forming a single curve. These parts are easier to handle but require additional care to maintain continuity at the join points, sometimes also called *breakpoints*, resulting in a sufficiently continuous curve [Gal99, p. 187], [PT97, p. 47].

We define a *piecewise polynomial* $f(t)$ of order k with regard to a strictly increasing *sequence of breakpoints* (t_n) as a function with the same image as a polynomial $P_i(t)$ of degree $< k$ on each of the half open intervals $[t_i, t_{i+1})$.

This definition, adopted from the chapter by Carl de Boor [dB02] in the book [FHK02], results in a right-continuous function. This means that for any t

$$\lim_{x \to t+} \gamma(x) = \gamma(t) \ .$$

This choice is arbitrary, but the right-continuous curve definition has become the standard [dB02, p. 141].

Bézier curves as previously defined can be seen as a map from the unit interval $I = [0, 1]$ onto a vector space, most commonly $\mathbb{R}^2$ or $\mathbb{R}^3$. For a piecewise (polynomial) curve, this definition has to be adopted. A spline curve consists of a collection of intervals according

to the given sequence of breakpoints also called *knots*, for which every interval $[t_i, t_{i+1}]$ is mapped onto a (polynomial) curve segment.

At this point, we distinguish between local and global parameters. For every global parameter $t \in [t_0, t_n]$ we evaluate a point on the curve as $\gamma(t)$. For a given interval $[t_i, t_{i+1}]$ we can define the local parameter s as

$$s = \frac{t - t_i}{t_{i+1} - t_i} \ .$$

It can easily be seen that s is in the unit interval for all $t \in [t_i, t_{i+1}]$. Depending on the properties of interest, it can either be more convenient to argue on global coordinates or it can be of advantage to describe certain properties in terms of local coordinates.

6.2.1 *B-Spline Basis Functions*

Based on the definition of a non-decreasing *knot sequence* (t_i), e. g., $t_0 \leq t_1 \leq t_2 \ldots \leq t_j \leq \ldots$ we define the kth B-spline basis function of order 1 for a given knot sequence (t_i) as the characteristic function on the interval $[t_k, t_{k+1})$. The characteristic function $\chi_T(x)$ is defined as

$$\chi_T(x) := \begin{cases} 1, & x \in T \\ 0, & \text{otherwise} \end{cases}$$

resulting in the following definition for the B-spline basis functions of order 1:

$$b_{1,k}(t) := \begin{cases} 1, & t \in [t_k, t_{k+1}) \\ 0, & \text{otherwise} \end{cases}$$

It can be observed that the resulting sequence $(b_{1,k})(t)$ forms a partition of unity for all t in the parametric domain T given by $\inf_i t_i < t < \sup_i t_i$. Furthermore, two identical knots in the knot sequence $t_k = t_{k+1}$ yield the constant function 0 as basis function, as in this case $b_{1,k}(t) = 0$ for all t.

B-spline basis functions of higher degree $n > 1$ are recursively defined by the recurrence relation

$$b_{n+1,k}(t) := \omega_{n+1,k}(t) b_{n,k} + (1 - \omega_{n+1,k+1}) b_{n,k+1}$$

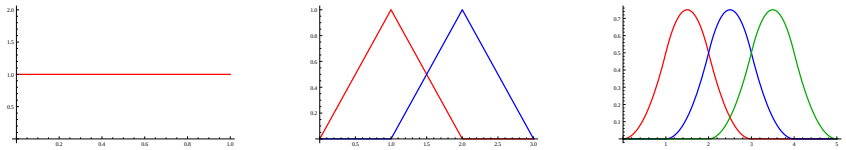
with

$$\omega_{n,k}(t) := \frac{x - t_k}{t_{k+n-1} - t_k} \ .$$

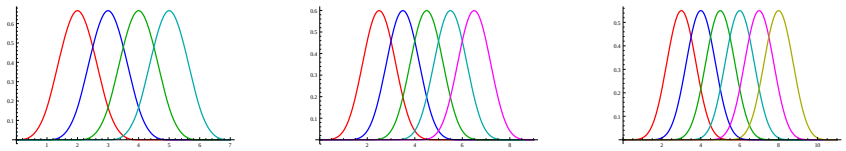
Using an induction on k, it can be shown that this recursive definition of the B-spline basis functions $b_{n,k}$ indeed yields a piecewise

polynomial of order k with break points at $t_k, t_{k+1}, \ldots, t_{k+n}$. Furthermore, the basis functions $b_{n,k}$ are positive in the interior and vanish outside of the interval $[t_k, t_{k+n})$.

Figure 36 shows sample B-spline basis functions up to order 5.



(a) degree 0 (b) degree 1 (c) degree 2



(d) degree 3 (e) degree 4 (f) degree 5

Figure 36: Sample B-spline basis functions over a uniform knot sequence $t_i = i$, i. e., the set of integers

6.2.2 *B-Splines*

Based on the convolution of certain functions describing probability distributions, B-splines along with a suitable knot sequence have been of interest since the 19th century, with their main application in smoothing statistical data. An early appearance can be found in Schoenberg's paper from 1946 on the approximation of equidistant data by analytic functions [Sch46].

Schoenberg defines and uses B-splines as the convolution to a given power k of characteristic functions on given intervals [dB86]. The result forms a probabilistic density distribution of the error which arises for a sum of k real random variables when replacing the variables with their closest integer representations [Sch46].

Starting with the work by Schoenberg in 1946, the discovery of the recurrence relations of B-splines by de Boor [dB72] and Mansfield [dB86, p. 5] as well as by Cox [Cox72] who proved it by a different argument and only for a strictly increasing knot sequence, formed the foundation of the modern spline approximation [Far96, p. 141]. Schoenberg studied several aspects of *Cardinal splines* in different papers published in the late 60s and early 70s where he focused on uniform knot sequences [dB86, p. 2] which led to the publication of *Cardinal Spline Interpolation* [Sch73]. The use of B-splines for curves and surfaces in computer aided geometric design was announced [GR74b, p. 310] and published [GR74a] by Gordon and Riesenfeld in 1974.

A *B-spline* of order n with a given knot sequence (t_i) is defined as a linear combination of the n B-spline basis functions $b_{n,k}$. It must be

observed that in early literature the term *B-spline* is used to describe what has been defined as a B-spline basis function in our work. The linear combination of B-spline basis functions is often called solely a *spline*. We use the convention that a B-spline is based on B-spline basis functions, as this has become the common terminology in computer aided geometric design [dBo2, p. 144].

For practical reasons, (t_n) is expectedly a finite sequence with the only requirement to be non-decreasing. Nevertheless, from the definition of the B-spline basic functions of order k, it can easily be observed that for any knot sequence at most k basic functions are non-zero. A B-spline basis function of order k is non-zero only in the interval $[t_i, t_i + k]$. Thus, the B-spline segment $s(t)$ from the B-spline of order k defined upon the knot-sequence (t_n) as

$$B_k(t) := \left(\sum_i p_i b_{k,i} \right)(t)$$

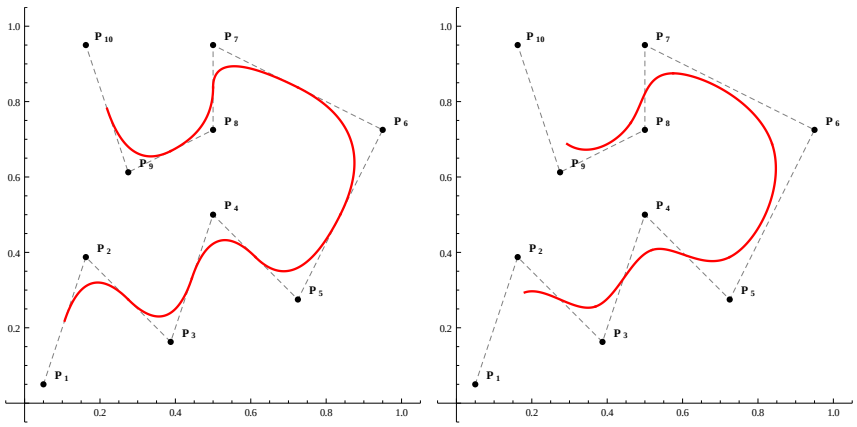
can be evaluated for a $t \in [t_k, t_{k+1}]$ as [PBP02]

$$s(t) := \sum_{i=0}^{k} p_i b_{k,i}(t) \ .$$

Therefore, from a theoretical point of view, it does not make a difference if the knot sequence is finite, infinite or even bi-infinite, as the sum can always be evaluated point wise. Even for an infinite knot sequence the sum

$$\sum_i p_i b_{k,i}(t)$$

is well-defined. From a practical view, this implies that moving a control point results in changing the local behavior of the curve.



(a) quadratic B-spline curve (b) cubic B-spline curve

Figure 37: Sample quadratic and cubic B-spline curves

Two sample B-spline curves can be seen in Figure 37 with uniform knot sequence $t_i = i$ and given control points. Each of the curves consists of a set of curve segments *splined* together at the break points.

6.2.3 *The de Boor Algorithm*

The de Boor algorithm is a fast and numerically stable algorithm for evaluation of B-spline curves. It can be considered a generalization of de Casteljau's algorithm for Bézier curves and was devised by Carl de Boor in 1972 [dB72]. We consider a B-spline segment for a given knot sequence (t_n) and given control points $p_i = p_{0,i}$ defined as

$$s(t) = \sum_i p_{0,i}\, b_{n,i}(t) \quad .$$

We can, analogously to de Casteljau's algorithm, use the recurrence relation on B-splines in a repeated scheme with an according definition of $p_{k,i}$ in the given interval $[t_n, t_{n+1})$ in order to conclude that [PBP02]

$$\begin{aligned} s(t) &= \sum_{i=0}^{n} p_{0,i}\, b_{n,i}(t) \\ &= \sum_{i=1}^{n} p_{1,i}\, b_{n-1,i}(t) \\ &\vdots \\ &= \sum_{i=n}^{n} p_{n,i}\, b_{0,i}(t) \\ &= p_{n,n} \end{aligned}$$

This computation is based on the generation of intermediate nodes $p_{k,i}$ at each level k. The values for $p_{k,i}$, computed recursively following the recurrence relation on the B-spline basis functions allow a definition of $p_{k,i}$ as

$$p_{k,i} = (1-\alpha)p_{k-1,i-1} + \alpha p_{k-1,i} \quad .$$

This definition contains the weights $\alpha = \alpha_{n-k,i}$ which are computed based on the underlying knot sequence and defined as

$$\alpha = \alpha_{n-k,i} = \frac{t - t_i}{t_{i+1+n-k} - t_i} \quad .$$

Since $t \in [t_i, t_{i+1}]$, we can conclude the weights α to be in the unit interval $[0, 1]$. Furthermore, as the linear constants in above definition α and $1 - \alpha$ sum up to 1, the intermediate points form convex combinations of each other [PBP02, p. 64].

Similar to the arrangement in de Casteljau's algorithm, these intermediate nodes can conveniently be displayed in a (lower) triangular scheme:

$$
\begin{array}{lllll}
p_{0,0} & & & & \\
p_{0,1} & p_{1,1} & & & \\
p_{0,2} & p_{1,2} & p_{2,2} & & \\
\vdots & & & \ddots & \\
p_{0,n} & & \cdots & & p_{n,n}
\end{array}
$$

The initial nodes which are points in a given space form the control points. As every intermediate node is a convex combination of the upper left and the left node, the resulting node $p_{n,n}$ is also a node in the given space and, furthermore, a convex combination of the control points. As a remark, it must be observed that for an arbitrary $t \in \mathbb{R}$ this process does not compute the spline segment itself, but returns a polynomial curve $s_n(t)$ that agrees with the spline segment on the corresponding knot interval with respect to t [PBP02].

When applying the de Boor algorithm, we generate a new control polygon at every level that describes the same B-spline curve [Far96, p. 149]. This is precisely how the knot insertion algorithm works: The knot insertion process consists of inserting a knot into the given knot sequence without altering the spline curve, requiring the construction of new control points. It has been shown that the first level of intermediate nodes computed by the de Boor's algorithm yields the new control points [Gal99, p. 234].

As an example, for the first level $k = 1$, the resulting knot sequence consists of the original knots t_i with the additional knot t inserted into the knot interval (t_{n-1}, t_n):

$$t_0 \quad t_1 \quad t_2 \quad \ldots \quad t_{n-1} \quad t \quad t_n \quad t_{n+1} \quad \ldots$$

The corresponding control points for the resulting spline segment consist of the original control points with $n - 1$ points replaced by the intermediate points from the first level of de Boor's algorithm:

$$p_0 \quad p_{1,1} \quad p_{1,2} \quad \cdots \quad p_{1,n} \quad p_n$$

Repeated knot insertions can be used to gain a refinement of the original control polygon while leaving the defined curve unchanged, as it has been shown that the sequence of control polygons P^r consisting of the control polygon with r additional knots inserted converges to the curve defined by these polygons.

The use of the repeated knot insertion can be found in rendering of the defined B-spline curve, as from the convergence directly follows that an arbitrarily close polynomial representation of the curve can be achieved if sufficiently many knots are inserted into the knot

sequence. This lends itself also to an adaptive rendering approach, inserting the more knots the higher the local curvature is, resulting in a polynomial approximation with fewer primitives and/or better error behavior.

6.2.4 *Uniform Cubic B-Splines*

In the previous sections, we discussed very general knot sequences (t_i) which we only required to be non-decreasing. For the following approximation primitive, uniform cubic B-splines, we focus on uniform knot sequences consisting only of integers, i. e., $t_i = i$. The resulting curves were already discussed by Schoenberg in 1973 as *Cardinal splines* [Sch73].

This special choice of the knot sequence results in several simplifications, most notably the fact that all B-spline basis functions are translates of one another. As a direct consequence, many of the associated formulae such as the recurrence relation and thus the evaluation with de Boor's algorithm are simplified. This recurrence relation which previously contained the weight functions $\omega_{n,k}(t)$ simplifies to [dB02, p. 146]

$$b_{n,k}(t) = \frac{t\, b_{n,k-1} + (k-t) b_{n,k-1}(t-1)}{k-1} \quad .$$

As the basis functions are translates of one another, the resulting B-spline is *uniform*.

For our approximation algorithm, we require primitives suitable for G^2 continuous approximation. Thus, we require at least cubic primitives and investigate *uniform cubic B-splines*.

6.2.5 *Intersection Tests*

As previously defined, a spline curve consists of a set of curve segments. A remarkable property of B-splines is that every curve segment of a B-spline is contained within the convex hull of its control points [Spi10, p. 5]. This is a direct consequence of the fact that any point on the curve evaluated using de Boor's algorithm forms a convex combination of its defining control vertices. Again, as with Bézier curves, this lends itself for a suitable simplification for a curve segment to conduct an intersection test with. An example of such a B-spline segment and its convex hull can be seen in Figure 38.

Additionally, as a direct analogon to the refinement procedure of Bézier curves, the de Boor algorithm can be used to insert knots and thus refine the curve by adding additional control points. With these additional control points it is possible to generate a refinement of the curve that can be used to conduct an intersection test with, possibly checking against the refined curve segment's convex hull.

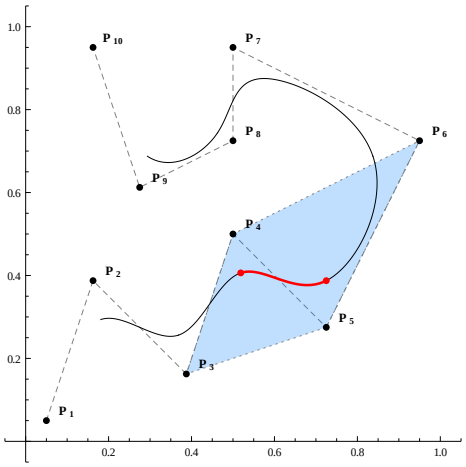


Figure 38: A B-spline segment lies within the convex hull of its defining control points.

As a shortcut to the knot insertion process, the work by Chaikin [Cha74] can be adopted: Chaikin's corner cutting scheme starts with a set of control vertices and repeatedly applies a subdivision process to the given control polygon, resulting in the following two new control vertices [Gal99, p. 235]:

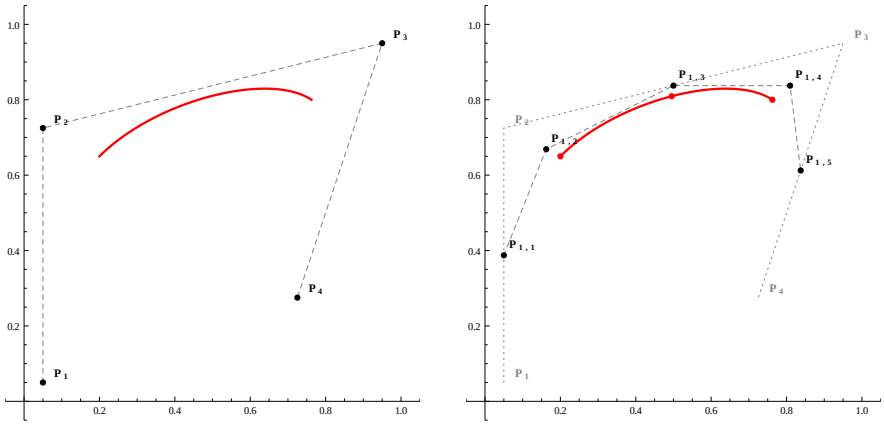
$$p'_{2i} = \frac{3}{4}p_i + \frac{1}{4}p_{i+1}$$
$$p'_{2i+1} = \frac{3}{4}p_i + \frac{1}{4}p_{i+1}$$

These control vertices can be shown to be the result of inserting a knot at the midpoint of every interval of an uniform knot sequence belonging to a quadratic B-spline curve. Thus, the polygon defined by Chaikin's corner cutting scheme defines and, furthermore, converges to a quadratic B-spline curve. The fact that this corner cutting process generates an uniform quadratic B-spline curve were published by Riesenfeld [Rie75].

The curves generated by the corner cutting scheme are, as they are quadratic B-spline curves, of class C^1. Thus, they are tangent continuous, but not suitable for G^2 continuous approximation. To subdivide a cubic spline, for any three consecutive control points the following new vertices have to be computed [Gal99, p. 235]:

$$p'_{2i} = \frac{1}{2}p_i + \frac{1}{4}p_{i+1}$$
$$p'_{2i+1} = \frac{1}{8}p_i + \frac{3}{4}p_{i+1} + \frac{1}{8}p_{i+2}$$

For binary subdivision of an uniform cubic B-spline defined upon a control polygon consisting of four vertices p_1, p_2, p_3, p_4 we have to



(a) Cubic B-spline curve (b) Result of subdivision process

Figure 39: Illustration of the cubic B-spline subdivision process

compute five vertices $p_{1,1}$ to $p_{1,5}$. This computation can be written in matrix form using the so-called *splitting matrix* as follows:

$$\begin{pmatrix} p_{1,1} \\ p_{1,2} \\ p_{1,3} \\ p_{1,4} \\ p_{1,5} \end{pmatrix} = \frac{1}{8} \begin{pmatrix} 4 & 4 & 0 & 0 \\ 1 & 6 & 1 & 0 \\ 0 & 4 & 4 & 0 \\ 0 & 1 & 6 & 1 \\ 1 & 1 & 4 & 4 \end{pmatrix} \begin{pmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \end{pmatrix}$$

These vertices define two cubic B-splines defined upon the control polygons $p_{1,1}, p_{1,2}, p_{1,3}, p_{1,4}$ and $p_{1,2}, p_{1,3}, p_{1,4}, p_{1,5}$, sharing three control vertices. This process is illustrated in Figure 39. Further details on above splitting matrix can be obtained from Catmull and Clark's paper [CC78] on generation of B-spline surfaces.

With this subdivision process defined, we can conduct an intersection test as with the Bézier curves following a binary search scheme. By subsequently inserting knots and thus subdividing the curve until a given intersection tolerance is reached, we can test for intersections using the convex hulls of the resulting spline curve segments. If no intersection is found, the spline curve segment can be omitted, otherwise we continue with refined curve segments. If the maximum intersection tolerance is reached, we report an intersection.

7 APPROXIMATION ALGORITHMS

7.1 GREEDY APPROACH

Held and Eibl [HE05] presented different approximation algorithms. In their original context, these algorithms were used to approximate discrete data sets by G^1 continuous arc splines as presented by Meek and Walton [MW92]. All of these algorithms follow the same idea, implementing a greedy scheme: They start in a given approximation node and traverse the list of approximation nodes as long as the approximation primitive between the given start node and the current node is valid.

The first start node is the first node in the list of approximation nodes. The *longest* approximation primitive found this way, i. e., the primitive which passes the maximum number of approximation nodes, is stored and the process is repeated with the current maximum node as start node. This definition does not cover a length comparison based on the actual curve length, but for complexity reasons rather uses the number of passed A-NODES as a measure of distance. A description of the algorithm in pseudo-code can be seen in Algorithm 4. For Algorithm 4, as well as for Algorithm 5 and Algorithm 6, $AP(i,j)$ or $AP(i,j,k,l)$ denote a primitive defined on the A-NODES with the respective indices.

Algorithm 4: Greedy approach – longest-arc

input : vector of n A-NODES
output: list of resulting approximation primitives r

```
s ← 0;
c ← 0;
while s < n do
    if AP(s, c + 1) is invalid or c + 1 ≥ n then
        add AP(s, c) to r;
        s ← c;
    else
        c ← c + 1;
```

The main difference between the various methods can be seen in the way, the next valid approximation primitive is found. Whereas the *longest arc*-method uses a linear search, the *doubling length* approach does not only increase the interval between each round's start node and the current node by one, but doubles the length in each

step. When an invalid approximation primitive is detected, the doubling process is resumed with the last valid node, until the last node is found, for which a valid approximation primitive can be generated.

This approach can be optimized with the *doubling and bisection* approach by using a binary search scheme once an invalid primitive is reported. This method first uses the *doubling length* method to find the first node, for which the resulting primitive is invalid. Then a binary search is conducted to again find the last approximation node which can be used to define a valid approximation primitive. An in-depth discussion including a complexity analysis of the different approximation methods can be obtained from [Eib02].

Algorithm 5: Doubling and bisection

```
input  : vector of n A-NODES a
output : list of approximation primitives r

start ← 0;
right ← 1;
while start < n do
    next ← 2 * right;
    if start + next > n − 1 then
        next ← n− start −1;
    if AP(start, start +next) is invalid
        or start + next = n − 1 then
        left ← right;
        right ← next;
        while right − left > 1 do
            mid ← (left + right) /2;
            if AP(start, start + mid) is invalid then
                right ← mid;
            else
                left ← mid;
        add AP(start, start + left) to r;
        start ← left;
        left ← 0;
        next ← 1;
    right ← next;
```

These methods have been proven to generate reliable results when used with approximation primitives that are defined upon exactly two A-NODES, a start and an end node. Primitives suitable to form curves of higher classes of continuity, including Bézier curves or cubic B-Splines as defined in Chapter 6, are defined upon more than these two A-NODES and thus cause problems when used with the scheme described above.

The main problem is that when successively adding the node for which the primitive passes the maximum number of A-NODES to the resulting approximation list, chances are good that with the next two rounds only a very small number of A-NODES can be passed. This is a consequence of the fact that the first long step *stretches* the approximation primitive to an extent such that almost no further nodes can be reached without leaving the tolerance zone. Thus, we have developed a top-down approach, allowing a successive refinement of a coarse start primitive violating the boundary conditions until it fits into the tolerance zone.

7.2 TOP-DOWN APPROACH

The following approach implements a divide and conquer algorithm to approximate the input. In our implementation, we focus on uniform cubic B-splines, although this algorithm can be used for any kind of approximation primitive. The general scheme can be described in the following way: We start with one approximation primitive and refine this primitive by subsequently adding new approximation nodes until we gain primitives sleek enough to fit through the tolerance zone.

We require a vector of A-NODES and a list of left and right boundary nodes defining the tolerance zone boundary as input for our algorithm. It is essential that the data type for the A-NODES allows a constant random access. Each A-NODE has a pointer to the corresponding left and right boundary nodes. Thus, intersection tests can be handled by first generating the relevant boundary segments and then using the primitive's methods to find intersections with the boundary. This implies that the relevant intersection code for checking against straight line segments can be encapsulated within the class definitions for the respective approximation primitive.

We can deduce from the polynomial form of the uniform cubic B-splines in power basis that the start point of the curve is incident in the first control point, if the first three control points coincide. Analogously, the end point of the curve coincides with the last control point, if the last three control points coincide. Thus, to approximate polygonal chains with fixed start and end point, we add the first and the last A-NODE three times each. Nevertheless, the idea of Algorithm 6 can easily be extended to cover closed polygons. To support closed polygons, we have defined a circular iterator on the list that allows a traversal from the end of the list back to the start. With appropriate choice of the starting nodes and adapted loop conditions, this can be used to extend the algorithm to closed polygons.

The refinement of a spline segment S that intersects the tolerance zone boundary is done by adding additional A-NODE indices to an index list. This index list contains indices for actual approximation

nodes stored within a vector. For the intersecting spline segment S which is defined upon the four indices stored in n_i, n_{i+1}, n_{i+2} and n_{i+3} the three indices $\lfloor \frac{n_i, n_{i+1}}{2} \rfloor$, $\lfloor \frac{n_{i+1}, n_{i+2}}{2} \rfloor$ and $\lfloor \frac{n_{i+2}, n_{i+3}}{2} \rfloor$ are added at the appropriate positions into the list.

Algorithm 6: Top-down approach (open polygonal chain)

input : vector of A-NODES a
output: list of approximation primitives r
data : empty list of node indices n

for $i \leftarrow 1$ **to** 3 **do**
  `pushFront(0)` into n;
  `pushBack(length(`a`) −1)` into n;
$c \leftarrow 0$;
while $c + 3 <$ `length(`n`)` **do**
  if `AP(`n_c, n_{c+1}, n_{c+2}, n_{c+3}`)` *is invalid* **then**
    `insert(`$\lfloor (n_c + n_{c+1})/2 \rfloor$`)` at position $c + 1$ into n;
    `insert(`$\lfloor (n_{c+1} + n_{c+2})/2 \rfloor$`)` at position $c + 3$ into n;
    `insert(`$\lfloor (n_{c+2} + n_{c+3})/2 \rfloor$`)` at position $c + 5$ into n;
  else
    $c \leftarrow c + 1$;
for $i \leftarrow 3$ **to** length$(n) - 1$ **do**
  if n_i *is redundant* **then**
    `erase(`n_i`)`;
  else
    `pushBack( AP(`n_{i-3}, n_{i-2}, n_{i-1}, n_i`))` into r;

An additional check after the refinement process ensures that no redundant node is added to the list of resulting approximation primitives. Redundant nodes are control points that can be safely removed, i. e., the spline curve remains in the tolerance zone even without the redundant node. A uniform cubic B-spline segment is defined on exactly four control points. So a check for redundancy at node with index i can easily be conducted by checking the three spline segments defined by the approximation nodes $(a_{i-3}, a_{i-2}, a_{i-1}, a_{i+1})$, $(a_{i-2}, a_{i-1}, a_{i+1}, a_{i+2})$ and $(a_{i-1}, a_{i+1}, a_{i+2}, a_{i+3})$ for intersections. In the case of open polygonal chains, the first three nodes and the last three nodes must not be removed to guarantee the spline curve to start and end in the fixed start and end nodes. In the closed case, again circular iterators have been used.

7.3 COMPLEXITY ANALYSIS

For an in-depth discussion of the overall complexity of the algorithm presented in this thesis, we start by giving an overview of the data structures used.

7.3.1 *Data Structures*

The input read from files using our I/O routines is stored in an object of type AOBJECT containing a linked list of CURVE-objects. These CURVE-Objects are used to store the input vertices, approximation nodes and tolerance zone boundaries. As container for the n input vertices and for the approximation nodes we chose to use a double-ended-queue, short *deque*, with the following properties:

- **Constant time random access:** Individual elements can be accessed by their position index in constant time $O(1)$.
- **Linear time iteration:** Iteration over all elements in any order can be done in linear time $O(n)$.
- **Efficient insertion and deletion at both ends:** Elements can be added and removed either at the beginning or the end of the deque in amortized constant time $O(1)$.

The use of this data structure allows to access elements with a given index in constant time, i. e., the data structure behaves like a traditional vector or array of elements. Other than with plain arrays of elements, pointer arithmetics is not supported. Thus, numerical indices have to be computed and used for element access. The handling of allocation and the actual storage of elements is subject to the implementation and the actual library used.

In contrast to the double-ended-queue data structure with constant time random access used for approximation nodes, the nodes defining the tolerance zone boundary are stored in a doubly linked list. This data structure allows constant time insertion and deletion, as long as an iterator to the position is known. As a major drawback, these lists lack the support of direct access to an element by a given index. Thus, random access requires linear time.

- **Linear time random access:** Individual elements can be only accessed by their position index by traversing the list from either the beginning or the end, resulting in linear time $O(n)$.
- **Linear time iteration:** Iteration over all elements in list order either from the beginning to the end or vice versa can be done in linear time $O(n)$.

- **Constant time insertion and deletion:** With a known position, the insertion and deletion operations require only constant time $O(1)$.

These lists are of great use for data that requires a lot of insertion and deletion operations in the middle of the list. An example of such a deletion operation is the removal of collinear or degenerated vertices or segments from a list.

7.3.2 *Preprocessing*

As preprocessing, the entire input is scanned for degeneracies such as collinear vertices or zero-length segments, which are removed. Additionally, the orientation at each vertex is computed as used for the tolerance zone computation in Chapter 4, indicating whether the vertex is convex or concave. As this preprocessing is handled only on a per-vertex-base, the whole process clearly is in linear time, as well as in linear space. The algorithm is defined to work on simple polygons or polygonal chains thus ruling out any self-intersections. A check for self-intersections is not conducted but, as already mentioned in Section 1.2, can be done in $O(n \log n)$ time.

As we resort to the Voronoi diagram to compute the tolerance zone boundary, a smoothing using a pre-approximation as proposed by [HE05] to handle noisy input data is not required.

Thus, we conclude the preprocessing process to run in $O(n)$ linear time and space.

7.3.3 *Tolerance Zone Boundary*

The computation of the tolerance zone boundary uses the Voronoi diagram of the input. The Voronoi diagram can be computed in $O(n \log n)$ time and requires linear $O(n)$ space to be stored. The tolerance zone is defined on a subset of the Voronoi diagram, and thus the number of boundary nodes is linear in the number of input elements. Thus, as already discussed in Section 4.3, the tolerance zone boundary computation runs in $O(n \log n)$ time and requires $O(n)$ linear space.

7.3.4 *Approximation Nodes*

The approximation nodes are placed on the edges of the medial axis of the *inner Voronoi diagram*. This inner Voronoi diagram is computed using the boundary nodes. The number of these boundary nodes is linear in the number of input vertices and thus the computation of the inner Voronoi diagram requires $O(n \log n)$ time and $O(n)$ linear space. Thus, to find the medial axis, we have to traverse a subset of

the Voronoi edges of the inner Voronoi diagram and get a number of A-NODES that is linear in the number of input vertices. So the computation of the approximation nodes can be done in $O(n \log n)$ time and $O(n)$ space.

If not specified otherwise, the approximation nodes are additionally sampled with respect to the width d of the tolerance zone, resulting in a number of $O(n/d)$ A-NODES.

7.3.5 *Approximation*

The TOP-DOWN-algorithm can be seen as a divide and conquer approach. With every split step, an intersection test is conducted and the problem is refined by adding a constant number of node indices to a list functioning as a stack. The merge step does not require additional work, but the intersection test with the tolerance zone boundary can require a linear number of intersection tests against straight line segments. Thus, following the ideas of divide and conquer algorithms [Knu98], the complexity of the top-down approximation can be concluded to be in $O(n \log n)$, requiring linear $O(n)$ space.

Finally, these runtimes sum up and we can conclude our algorithm to require $O(n \log n)$ asymptotic runtime complexity and $O(n)$ linear space.

7.4 COMPARISON AND PRACTICAL RESULTS

7.4.1 *Test Environment and Data Set*

We ran a series of tests on our testing environment RENTIER which is a machine equipped with an INTEL CORE *i7*-2600 processor with four cores hyperthreading enabled backed by 16 GB[1] of memory. Although most of our test data is hosted on a network file system, we do not take system calls for overall runtime measurement into account and thus do not suffer a drawback from slow I/O operations. As 64 bit operating system, we used UBUNTU 10.04 LTS with the LINUX 3.0.0 x64 kernel.

For our test, we simultaneously executed four instances of our implementation to fully utilize all cores available. Each execution was bounded to use at most 4 GB memory and the maximum execution time was limited to 3600 seconds. Additionally, our testing build was not linked against OPENGL as utilized by the NXWINDOW library, which we otherwise use to visualize our algorithm in our graphical user interface.

The tests were run on the data set provided by Martin Held, subsuming a total of 22124 individual input files. These inputs did not

1 Using binary units, this total a number of $16 * 1024^3$ bytes.

only contain single polygons or polygonal chains, but also incorporated multiply-connected structures.

To gain repeatable empiric results, we defined an automatic configuration of the required parameters of our algorithm that is based on properties of the individual input files. Thus, as an approximation tolerance, we specify both, the left and the right tolerance to be half the arithmetic mean of the lengths of all input segments. This is computed as the sum of all lengths of all input segments divided by the number of input segments, with zero length segments removed.

7.4.2 *Data Compression and Runtime Plots*

In the context of data compression, our algorithm can be used to gain simplified representations of the original input while preserving topology. Our tests have shown that our implementation using our automatic setup yields an average data reduction to 55.44%. The number of control points for inputs of a given number of input vertices is plotted in Figure 40.

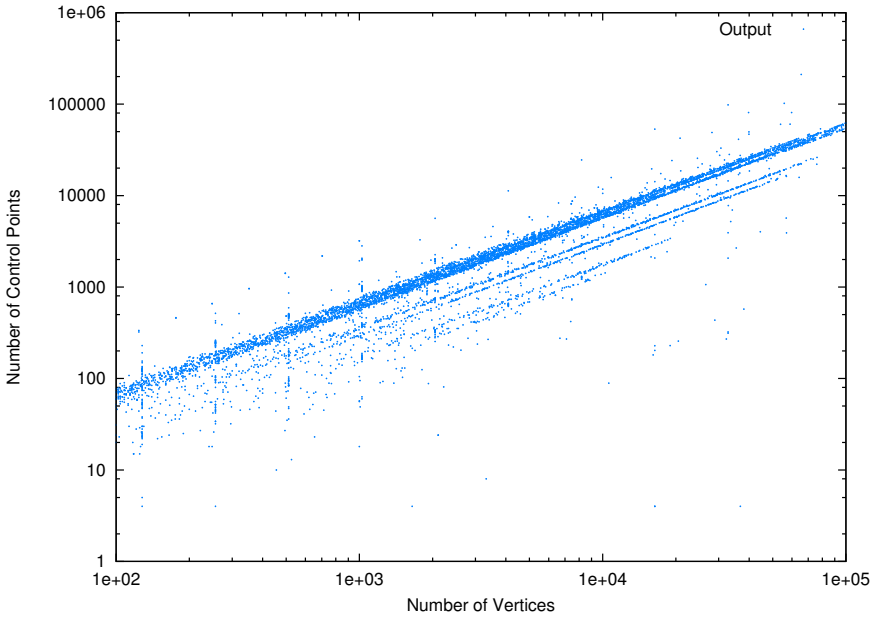
In addition to the number of resulting control points, we have measured our implementation's runtime and memory consumption, plotted in Figure 41 and Figure 42. The corresponding runtime plots are itemized in Figure 43, Figure 44, Figure 45 and Figure 46 to give an overview of the sub-algorithms' contributions to the total.

7.4.3 *Comparison with Previous Work*

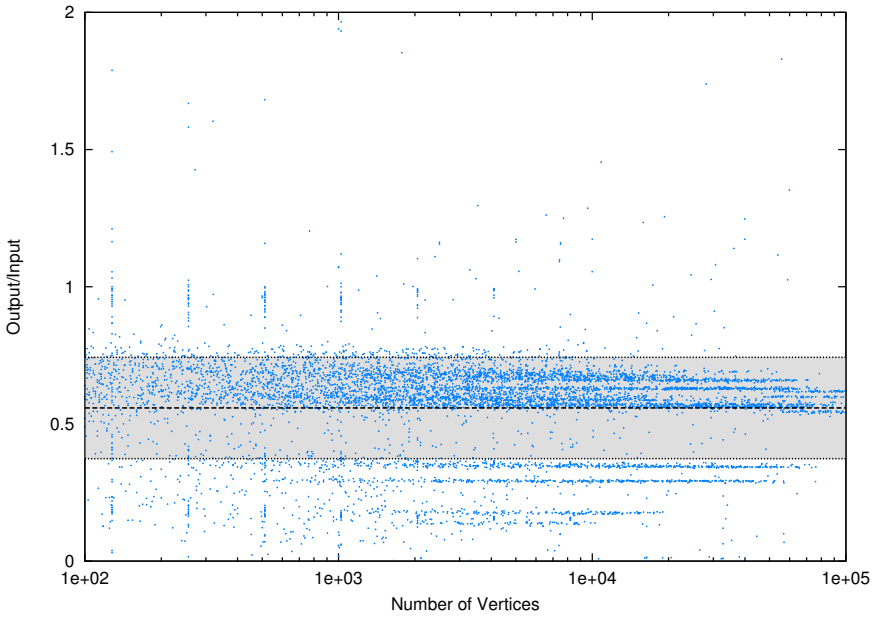
We compared our new divide and conquer approach against the doubling and bisection algorithm presented by Held and Eibl [HE05]. Our tests using the same automatic setup of tolerance zone boundaries and A-NODES for both types of algorithms have shown very similar runtime characteristics for both algorithms when used with straight line segments and biarc primitives.

Although the divide and conquer approach is running faster, it tends to produce a slightly higher number of output primitives. The plots shown in Figure 50 indicate that our approach is both, significantly faster and more efficient in terms of resulting output primitives than the doubling and bisection method when using uniform cubic B-splines as approximation primitives.

The results for straight line segments, biarcs and splines are plotted in Figure 47, Figure 48 and Figure 49 respectively. The trend lines plotted in Figure 47a, Figure 48a and Figure 49a were computed using a linear fitting of the data in the log-log-plot. A comparison of the different approximation primitives can be seen in Figure 51.



(a) Resulting control points for inputs of a given number of input vertices



(b) Our tests have shown that the ratio is independent of the number of input vertices.

Figure 40: Plot showing the achieved data compression using the top down approach with uniform cubic B-splines.
The dashed line in Figure 40b represents the mean μ, the dotted lines bounding the shaded area represent the values $\mu - \sigma$ and $\mu + \sigma$ where σ stands for the standard deviation.

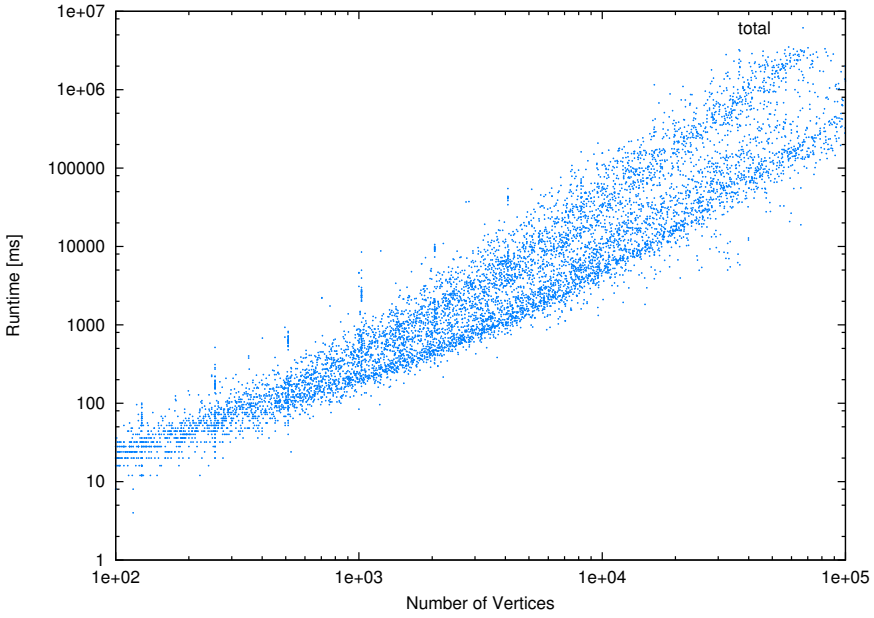


Figure 41: Runtime plot of our implementation

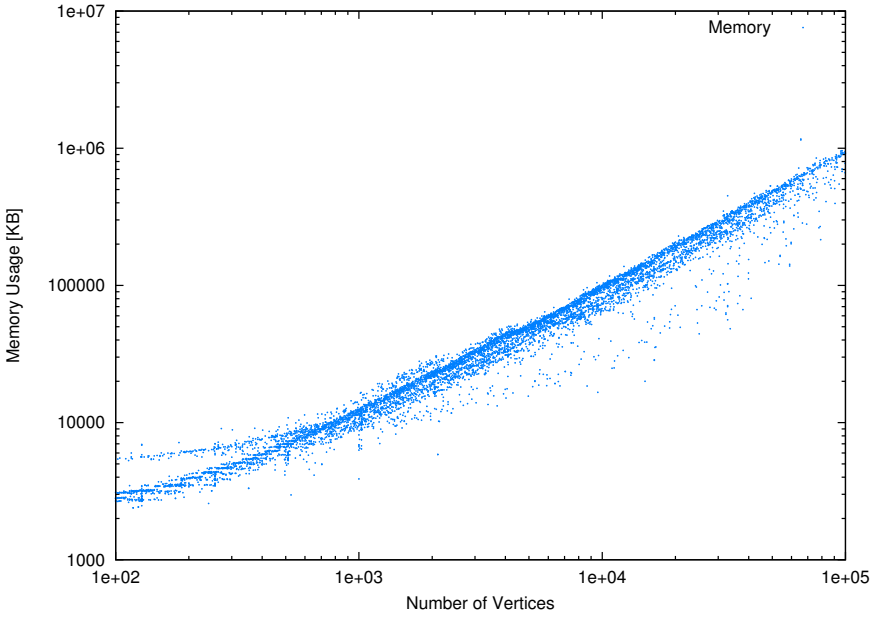


Figure 42: Plot showing a linear memory consumption

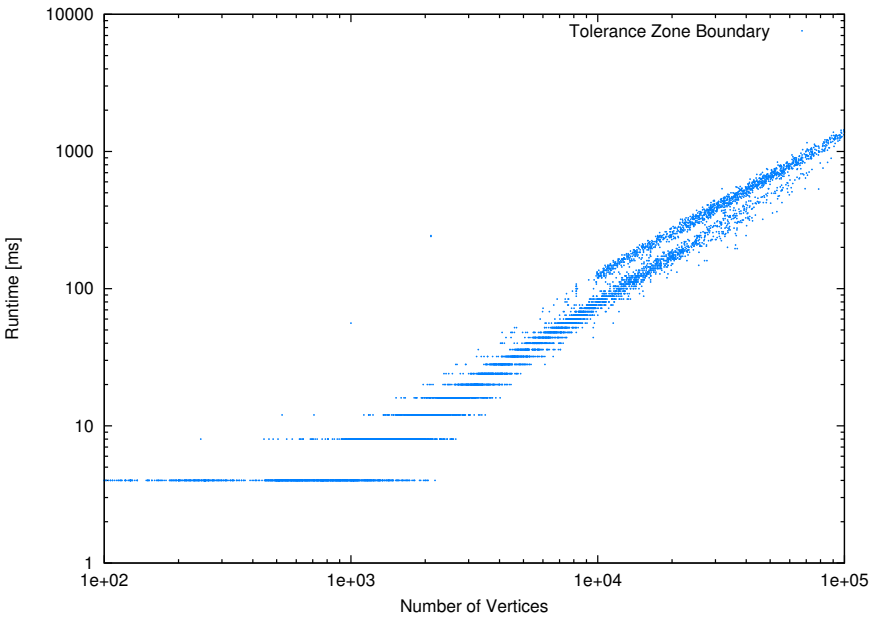


Figure 43: Runtime plot of the tolerance zone boundary computation

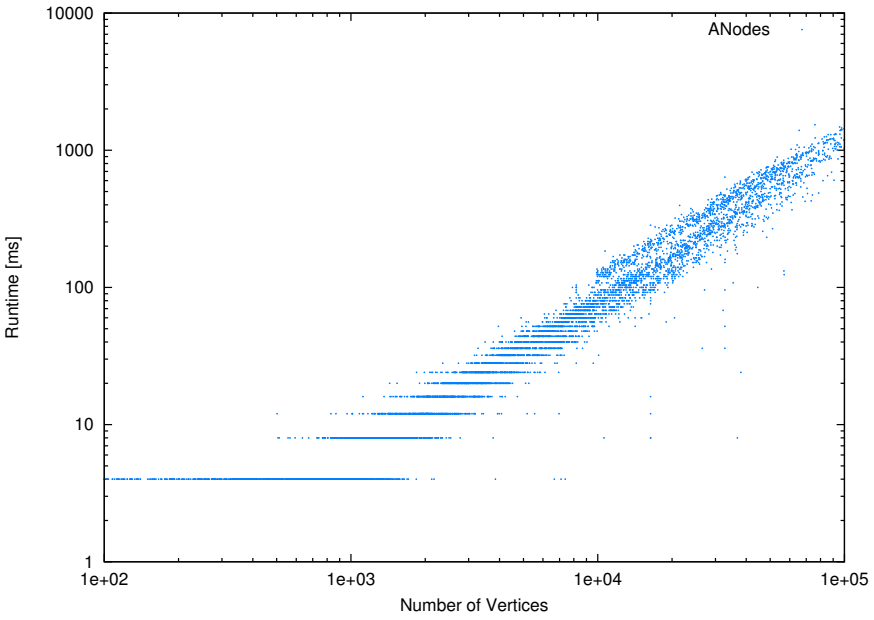


Figure 44: Plot showing the runtime of the A-NODE computation

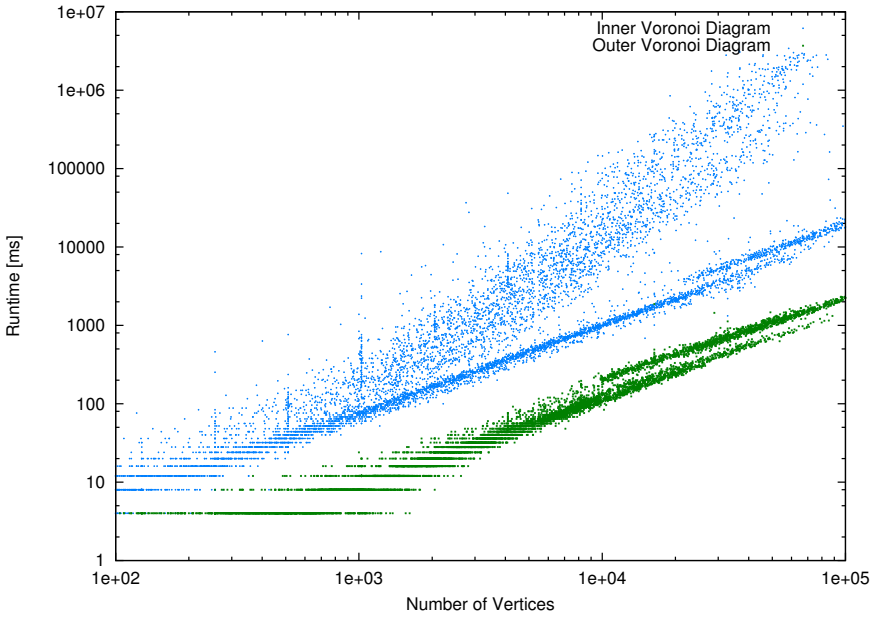


Figure 45: Runtime plot showing the computation of the inner Voronoi diagram used to find the medial axis of the tolerance zone boundary in blue and the outer Voronoi diagram defined on the original input in green.

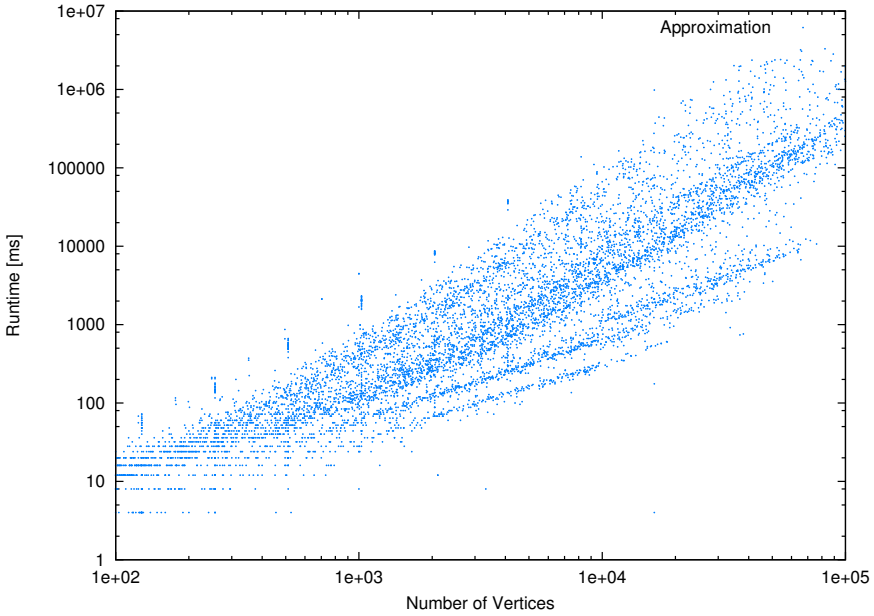
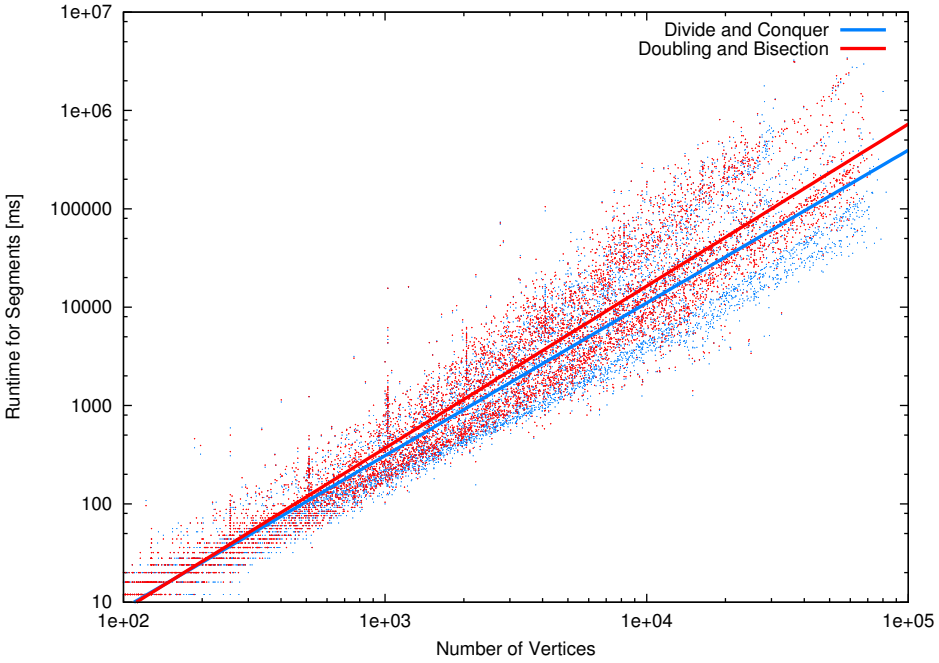
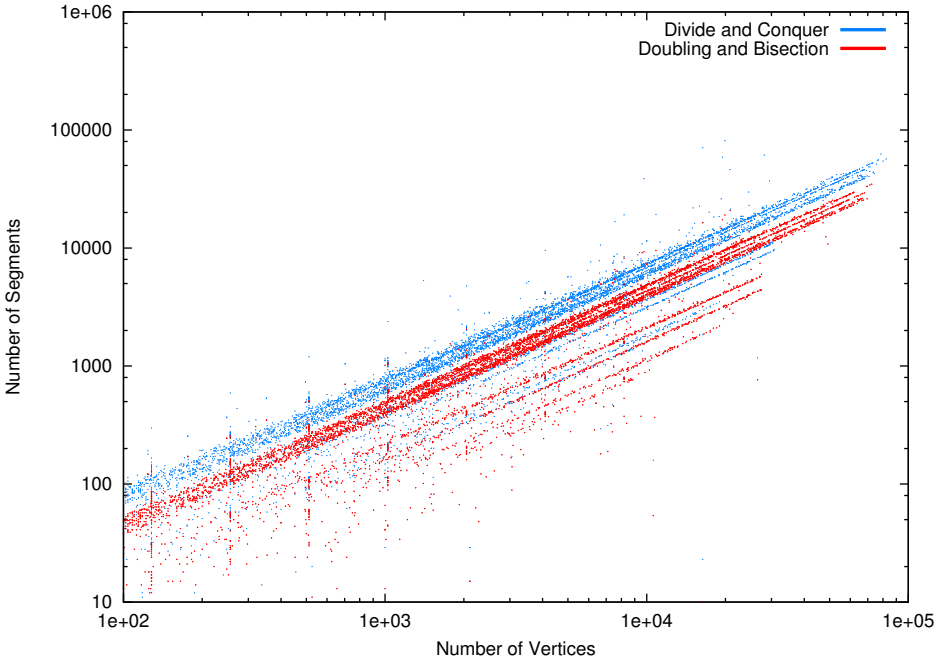


Figure 46: Plot showing the runtime of the approximation using our Top-Down approach

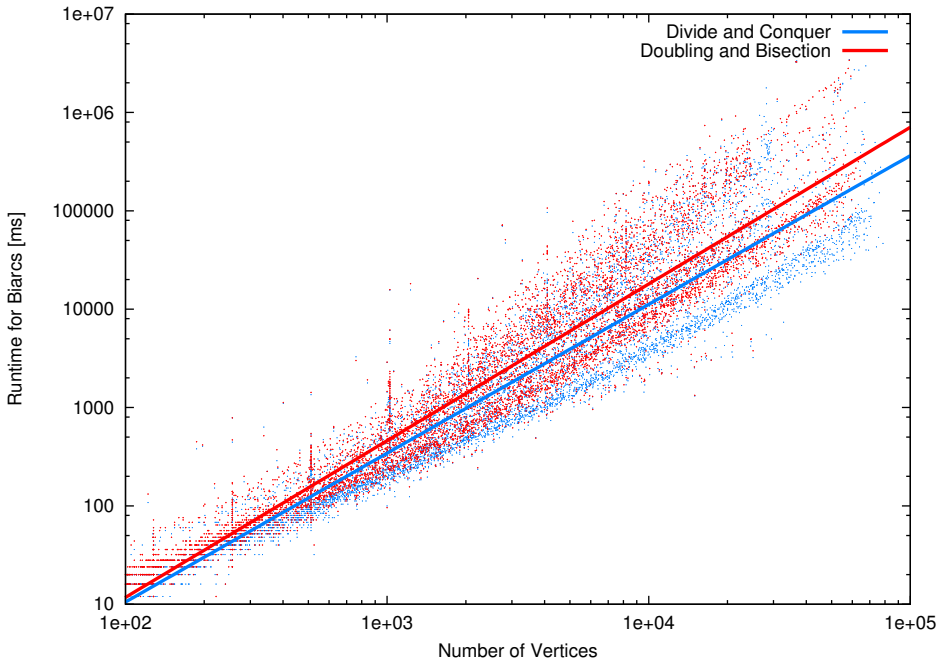


(a) Comparison of runtimes (segments)

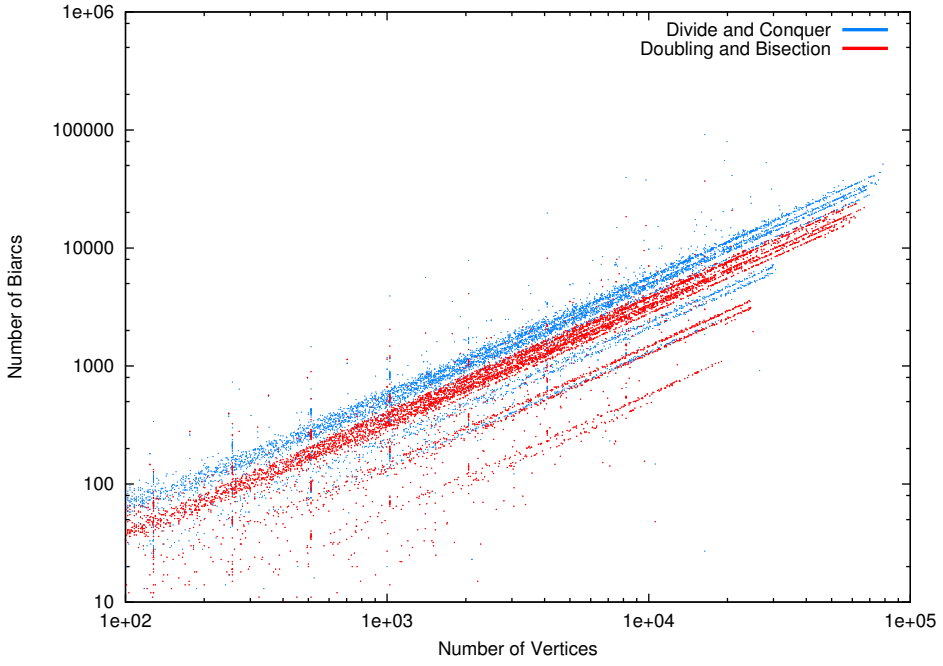


(b) Comparison of output segments

Figure 47: The plots show the comparison of runtimes using SEGMENTS as primitives in Figure 47a and the comparison of resulting output segments in Figure 47b.
The data for our divide and conquer approach is plotted in blue and for the doubling and bisection method in red.

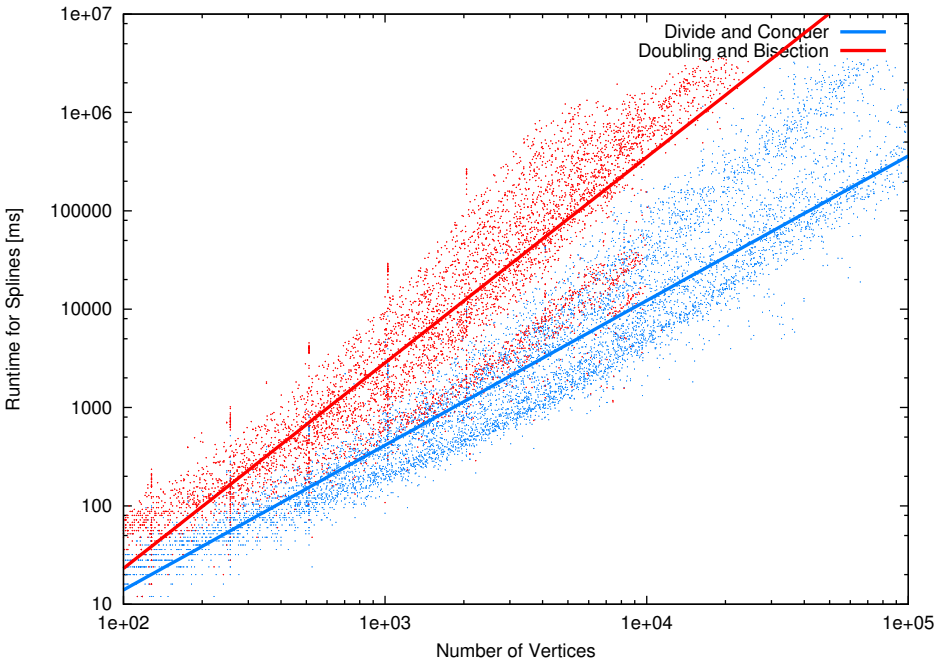


(a) Comparison of runtimes (biarcs)

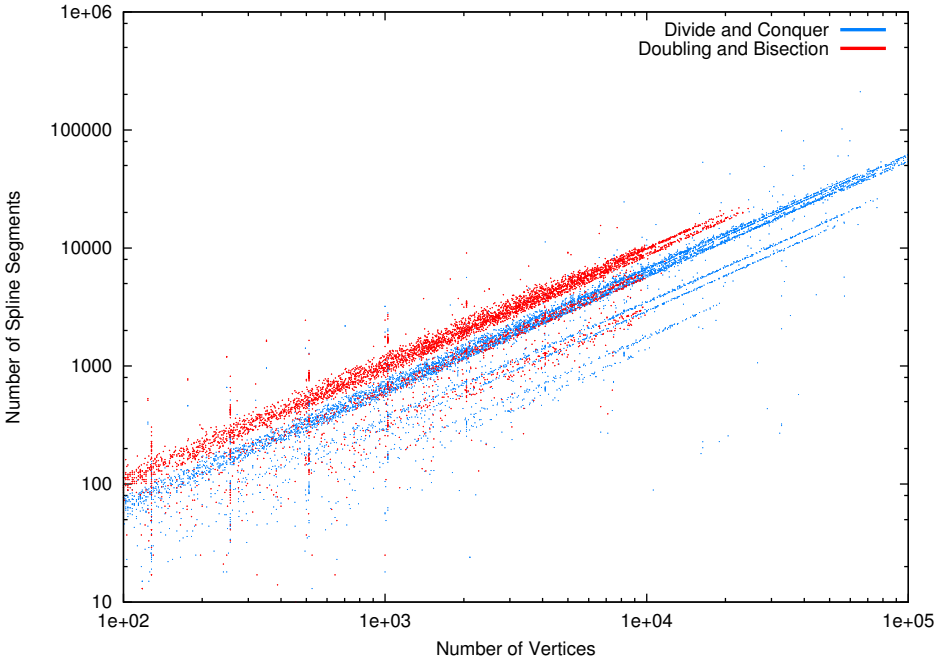


(b) Comparison of output biarcs

Figure 48: The plots show the comparison of runtimes using BIARCS as primitives in Figure 47a and the comparison of resulting output biarcs in Figure 47b.
The data for our divide and conquer approach is plotted in blue and for the doubling and bisection method in red.

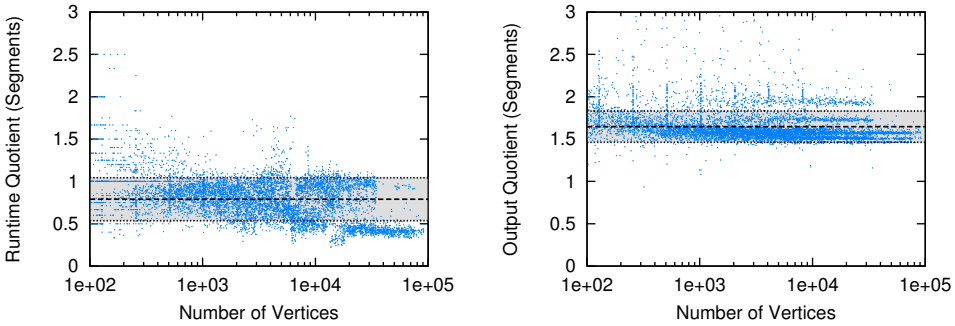


(a) Comparison of runtimes (spline segments)



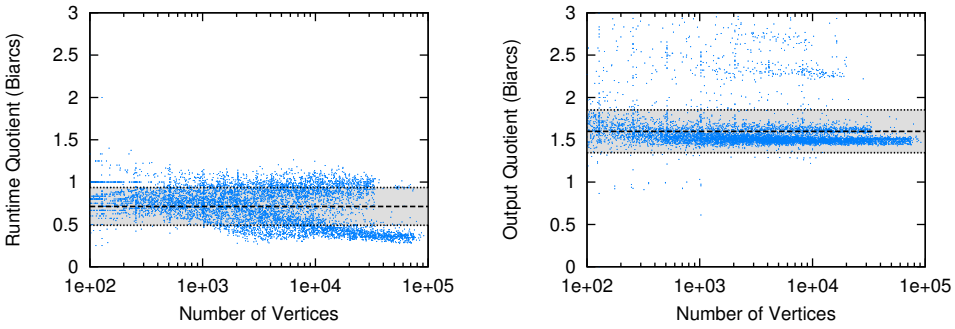
(b) Comparison of output spline segments

Figure 49: The plots show the comparison of runtimes using SPLINE SEGMENTS as primitives in Figure 47a and the comparison of resulting output control points in Figure 47b.
The data for our divide and conquer approach is plotted in blue and for the doubling and bisection method in red.



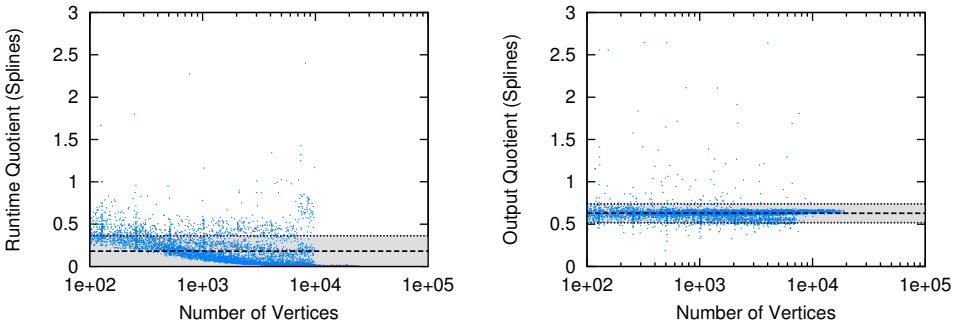
(a) Runtime for Segments

(b) Output for Segments



(c) Runtime for Biarcs

(d) Output for Biarcs



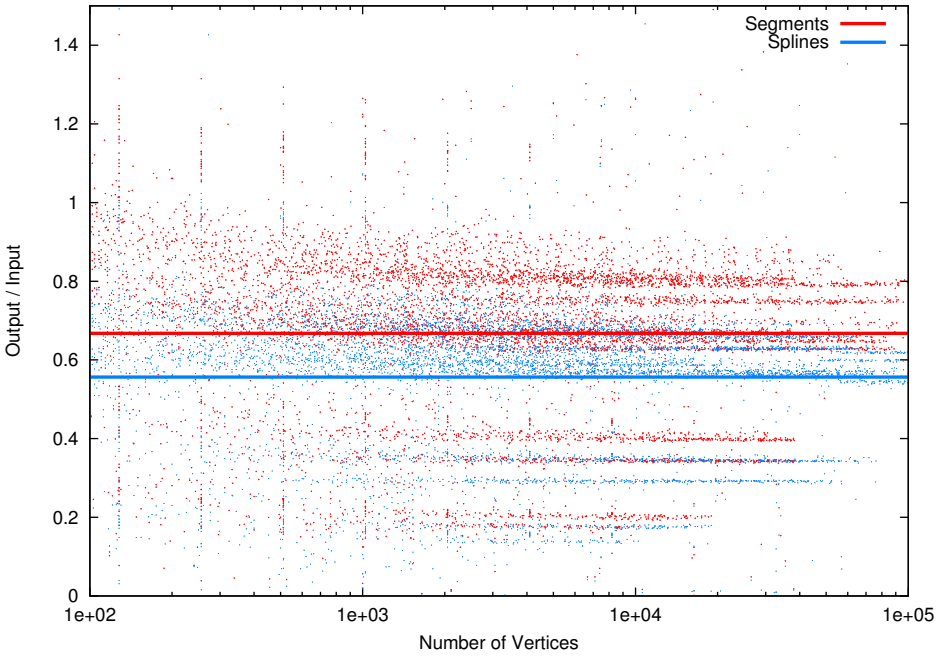
(e) Runtime for Splines

(f) Output for Splines

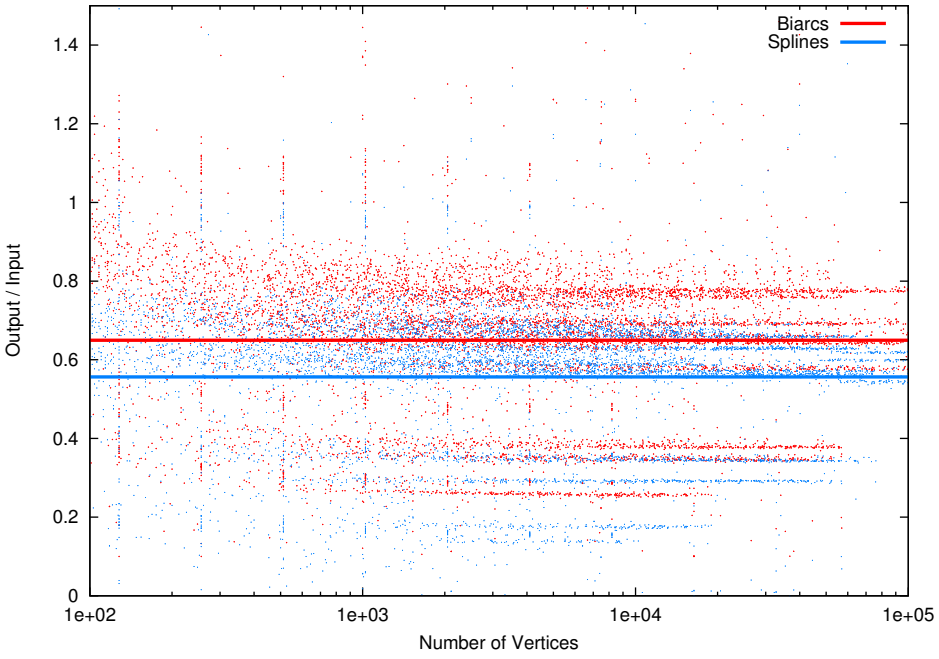
Figure 50: The above plots show quotients of the form

$$\frac{\text{Result Divide and Conquer}}{\text{Result Doubling and Bisection}}$$

of runtimes in the left and output size in the right column for segments in the top, biarcs in the middle and spline segments in the bottom row. The dashed line represents the mean μ and the shaded area shows the range $[\mu - \sigma, \mu + \sigma]$ with σ the standard deviation.



(a) Comparison segments – splines

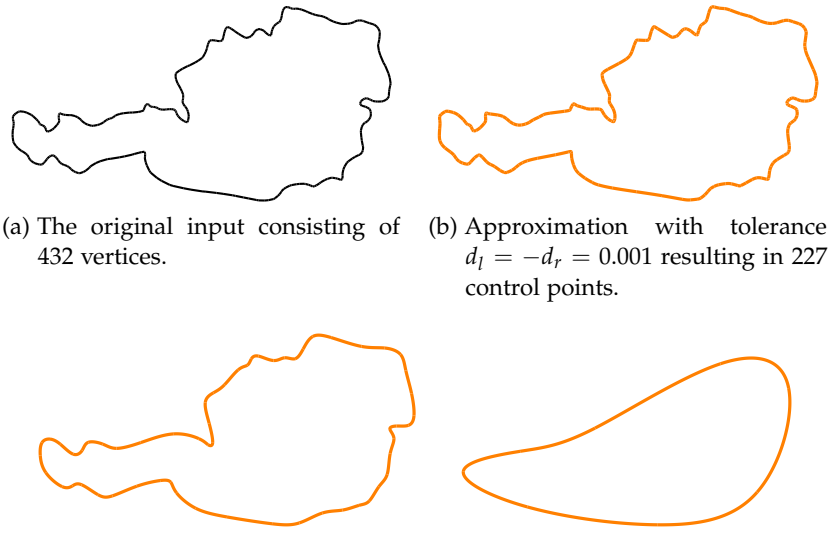


(b) Comparison biarcs – splines

Figure 51: The plots show the number of output elements using the divide and conquer approach normalized to the number of input vertices. The lines represent the averages of the corresponding data points.

7.4.4 *Sample Approximations*

We additionally give a few samples of approximations computed by our implementation. Some of these samples are drawn with corresponding structures such as tolerance zone boundaries. The colors used stand for the individual data structures, with the input black and the approximation in orange. The left and right tolerance zone boundary, including spikes, are drawn in magenta and cyan respectively. Sample approximations of noisy input structures can be seen in Figure 56 and Figure 58. A one-sided tolerance zone is used to approximate the multiply-connected polygons shown in Figure 58. A successive refinement of the input from Figure 52a can be seen in Figure 52b, Figure 52c and Figure 52d.



(a) The original input consisting of 432 vertices.

(b) Approximation with tolerance $d_l = -d_r = 0.001$ resulting in 227 control points.

(c) Approximation with tolerance $d_l = -d_r = 0.01$ resulting in 59 control points.

(d) Approximation with tolerance $d_l = -d_r = 0.1$ resulting in 8 control points.

Figure 52: Successive refinement of the border of Austria.

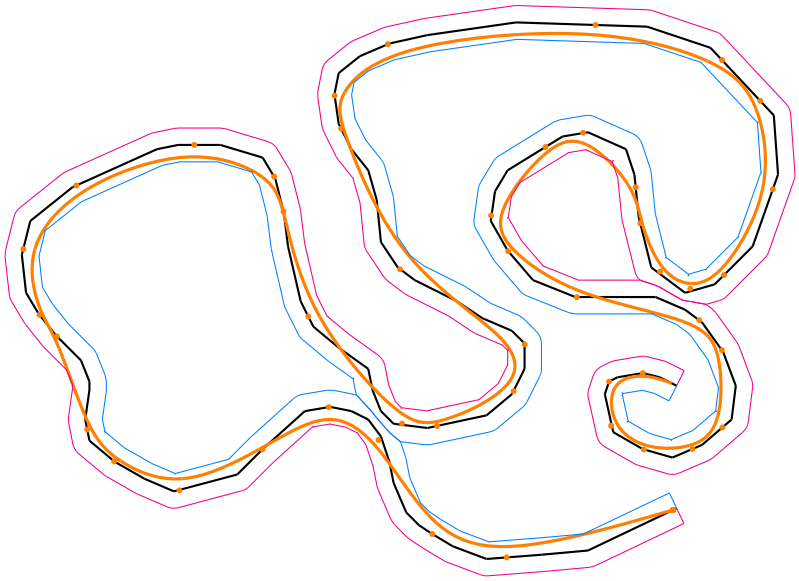


Figure 53: Sample approximation of an open curve.

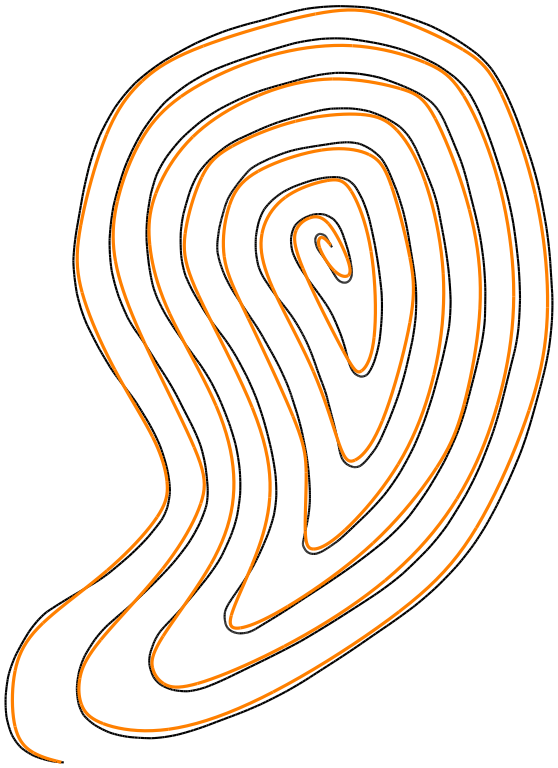


Figure 54: Sample approximation: The input was created using the algorithm by Martin Held and Christian Spielberger [HS09] and is used in the field of pocket machining. The input consists of 3301 vertices and our approximation returns 123 control points for a B-spline curve.

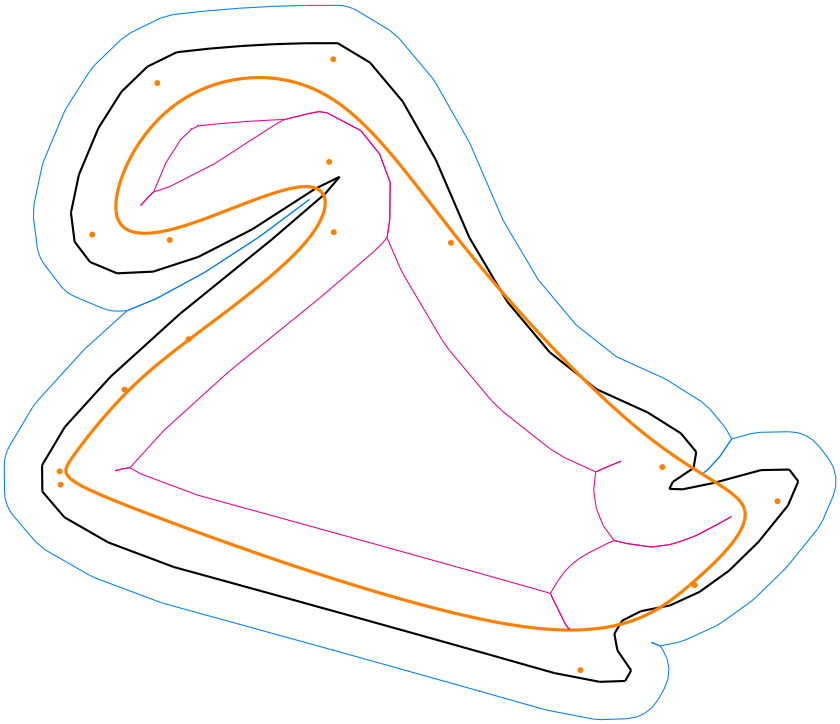


Figure 55: Sample approximation: The resulting curve lies within the tolerance zone.

Figure 56: Sample approximation: A noisy input and its G^2 continuous approximation. Because of the large tolerance threshold, the left tolerance zone depicted in magenta is degenerated to a set of spikes.

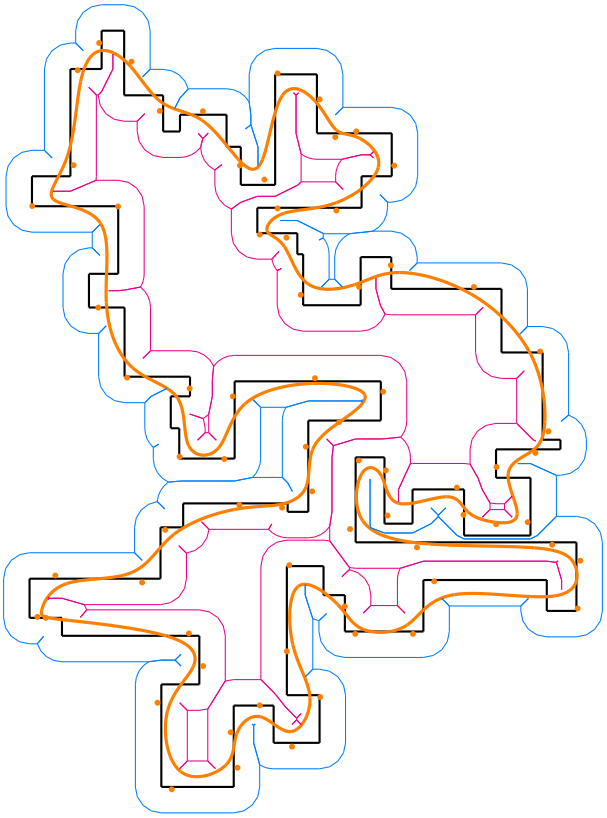


Figure 57: Sample approximation of an axis-aligned input curve.

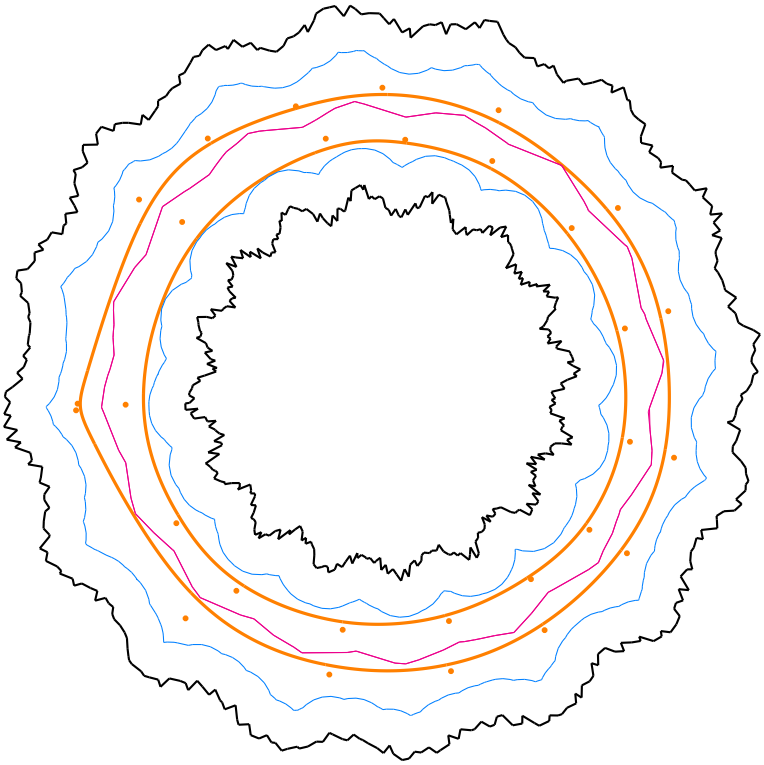


Figure 58: Sample one-sided approximation: The noisy input circles are oriented in opposite directions.

8

CONCLUSION

In this thesis we present a new algorithm to compute an approximation of multiply-connected polygonal chains or polygons. Our algorithm can be used in the environment PowerAPX by Held and Heimlich [HHo8a] to extend the given collection of existing algorithms. In contrast to these greedy algorithms, our algorithm implements a divide and conquer approach. Thus, instead of sampling an approximation by adding one approximation primitive after another, we rather begin with a coarse approximation which is refined until it is entirely confined to the so-called tolerance zone.

We implemented our algorithm using approximation primitives suitable to generate a curve consisting of G^2 continuous primitives, namely uniform cubic B-splines. Our implementation can be used to approximate both, open polygonal chains and polygons as well as multiply-connected structures thereof.

Our algorithm consists of several sub-algorithms that lie in a variety of fields. A major part of this thesis deals with the construction of a so-called tolerance zone. The tolerance zone is based upon a user definable left and a right tolerance threshold which confines the maximum Hausdorff distance between the input curve and the resulting approximation. For the computation of the tolerance zone we rely on the Voronoi diagram, which we use to compute the tolerance zone's boundary. The boundary is sampled using straight line segments, and these segments can be used to conduct fast intersection tests to prevent approximation primitives from leaving the tolerance band.

In order to give a definition of the tolerance zone, we utilize the Voronoi diagram and recall the definitions that can be obtained from literature. We do take open polygonal chains into account and thus do not make use of global properties such as the interior-/exterior-property of a point with respect to a polygon or the nesting-level of polygons. Rather than relying on these properties, we build a tolerance zone from subsets of Voronoi cells. Additionally, we propose the use of the medial axis as a subset of the Voronoi diagram of the tolerance zone, to gain a list of anchor nodes for our approximation primitives.

In this thesis, we give a formal definition of parametric curves in the Euclidean plane. As we aim to compute a G^2 continuous approximation, we define some basic properties such as the different types of continuity, parametric continuity C^r and geometric continuity G^r. Furthermore, we define many properties of curves that we either ex-

plicitly or implicitly use when defining our approximation primitives or algorithm.

In addition to the definition of parametric curves and their properties we give an overview of different types of curves used widely in geometric modeling and suitable as approximation primitives. We give a historic perspective starting with Bézier curves and illustrate the algorithm by de Casteljau to both, evaluate and refine a curve for intersection testing. To overcome the shortcomings of these types of curves, we present splines and focus on B-splines defined upon a uniform knot sequence. To achieve G^2 continuity, we require primitives of at least cubic degree and thus propose uniform cubic B-splines for use with our algorithm.

Intersection testing is another important part of this thesis. We pursue an approach consisting of consecutive refinement and convex hull testing. This can be done, as for our primitives two significant properties hold: Uniform cubic B-Splines are entirely defined within the convex hull of their control points and, furthermore, the de Boor algorithm provides an efficient solution to refine the control polygon and thus split the spline segment into two halves.

We tested our algorithm to confirm a linear memory consumption and runtime in $O(n \log n)$. The number of computed spline control points is linear in the number of input vertices, with an average data compression rate to 55.44% for a specific automated setup of the tolerances.

LIST OF ALGORITHMS

LIST OF FIGURES

All of the above figures used in this thesis have been created by ourselves using software we had either developed or been granted permission to use and thus do not infringe any rights of third parties. The Voronoi diagram figures have been created with Vroni provided by Martin Held. The sample approximations have been computed using our implementation in the environment of PowerAPX.

BIBLIOGRAPHY

[Aki70] Hiroshi Akima. A New Method of Interpolation and Smooth Curve Fitting Based on Local Procedures. *Journal of the Association for Computing Machinery*, 17(4):589–602, October 1970.

[BBI01] Dmitri Burago, Yuri Burago, and Sergei Ivanov. *A Course in Metric Geometry*. American Mathematical Society, June 2001.

[BC94] Klaus-Peter Beier and Yifan Chen. Highlight-line algorithm for realtime surface-quality assessment. *Computer-Aided Design*, 26(4):268–277, 1994.

[BD89] Brian A. Barsky and Tony D. DeRose. Geometric Continuity of Parametric Curves: Three Equivalent Characterizations. *IEEE Computer Graphics and Applications*, 9(6):60–68, November 1989.

[Ber13] Sergei Natanovich Bernstein. Démonstration du Théorème de Weierstrass fondée sur le calcul des Probabilités. *Communications of the Kharkov Mathematical Society*, 13:1–2, 1912/13. http://www.math.technion.ac.il/hat/fpapers/P03.PDF.

[Ber03] Marcel Berger. *A Panoramic View of Riemannian Geometry*. Springer, 1st edition, 2003.

[BFK84] Wolfgang Böhm, Gerald Farin, and Jürgen Kahmann. A survey of curve and surface methods in CAGD. *Computer Aided Geometric Design*, 1(1):1–60, July 1984.

[BG88] Marcel Berger and Bernard Gostiaux. *Differential Geometry: Manifolds, Curves, and Surfaces*, volume 115 of *Graduate Texts in Mathematics*. Springer-Verlag, 1st edition, 1988. Original French edition published by Presses Universitaires de France, 1987.

[BJMR75] G. Berg, W. Julian, R. Mines, and F. Richman. The Constructive Jordan Curve Theorem. *Rocky Mountain Journal of Mathematics*, 5(2):225–236, 1975.

[BKSS90a] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. The R*-tree: An Efficient and Robust Access Method for Points and Rectangles. *SIGMOD Rec.*, 19(2):322–331, May 1990.

[BKSS90b] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. The R*-tree: An Efficient and Robust Access Method for Points and Rectangles. In *Proceedings of the 1990 ACM SIGMOD international conference on Management of data*, SIGMOD '90, pages 322–331, New York, NY, USA, 1990. ACM.

[Blu67] H. Blum. A transformation for extracting new descriptors of shape. In Weiant Whaten-Dunn, editor, *Models for the Perception of Speech and Visual Forms; Proceedings of a Symposium*, pages 362–380. MIT Press, 1967.

[BM89] Carl Benjamin Boyer and Uta C. Merzbach. *A History of Mathematics*. John Wiley & Sons Inc, 2nd edition, March 1989.

[BM99] Wolfgang Boehm and Andreas Müller. On de Casteljau's algorithm. *Computer Aided Geometric Design*, 16(7):587–605, August 1999.

[BO79] Jon L. Bentley and Thomas A. Ottmann. Algorithms for Reporting and Counting Geometric Intersections. *IEEE Transactions on Computers*, C-28(9):643–647, September 1979.

[BR69] Richard Bellman and Robert Roth. Curve Fitting by Segmented Straight Lines. *Journal of the American Statistical Association*, 64(327):1079–1084, September 1969.

[Bro25] Luitzen Egbertus Jan Brouwer. Intuitionistischer Beweis des Jordanschen Kurvensatzes. In *Proc. Akad. Amsterdam*, volume 28, pages 503–508, June 1925.

[BT04] Jean-Paul Berrut and Lloyd N. Trefethen. Barycentric Lagrange Interpolation. *SIAM Review*, 46(3):501–517, September 2004.

[BW93] Thomas Becker and Volker Weispfenning. *Gröbner Bases: A Computational Approach to Commutative Algebra*, volume 141 of *Graduate Texts in Mathematics*. Springer, corrected edition, April 1993.

[CC78] Edwin Catmull and Jim Clark. Recursively generated B-spline surfaces on arbitrary topological meshes. *Computer-Aided Design*, 10(6):350–355, 1978.

[CC96] W. S. Chan and F. Chin. Approximation of Polygonal Curves with Minimum Number of Line Segments or Minimum error. *International Journal of Computational Geometry and Applications*, 6(1):59–77, 1996.

[CFKP97] James W. Cannon, William J. Floyd, Richard Kenyon, and Walter R. Parry. Hyperbolic Geometry. In Silvio Levy, editor, *Flavors of Geometry*, volume 31 of *MSRI Publications*, chapter 2, pages 59–116. Cambridge University Press, 1997.

[cga] CGAL, Computational Geometry Algorithms Library. http://www.cgal.org.

[Cha74] George Merrill Chaikin. An algorithm for high-speed curve generation. *Computer Graphics and Image Processing*, 3(4):346–349, 1974.

[Cha91] Bernard Chazelle. Triangulating a simple polygon in linear time. *Discrete & Computational Geometry*, 6:485–524, 1991. 10.1007/BF02574703.

[Cha96] Timothy M. Chan. Optimal Output-Sensitive Convex Hull Algorithms in Two and Three Dimensions. *Discrete and Computational Geometry*, 16:361–368, 1996.

[Col75] John P. Coleman. An Elementary Proof of Voronovskaya's Theorem. *The American Mathematical Monthly*, 82(6):639–641, June – July 1975.

[Cox72] M. G. Cox. The Numerical Evaluation of B-Splines. *IMA Journal of Applied Mathematics*, 10(2):134–149, 1972.

[Cox98] Harold S. M. Coxeter. *Non-Euclidean Geometry*. Mathematical Association of America, 6th edition, 1998.

[CS85] Elaine Cohen and Larry L. Schumaker. Rates of convergence of control polygons. *Computer Aided Geometric Design*, 2(1-3):229–235, 1985.

[CSW95] Francis Chin, Jack Snoeyink, and Cao Wang. Finding the medial axis of a simple polygon in linear time. In John Staples, Peter Eades, Naoki Katoh, and Alistair Moffat, editors, *Algorithms and Computations*, volume 1004 of *Lecture Notes in Computer Science*, pages 382–391. Springer Berlin / Heidelberg, 1995. 10.1007/BFb0015444.

[Dah86] Wolfgang Dahmen. Subdivision Algorithms Converge Quadratically. *Journal of Computational and Applied Mathematics*, 16(2):145–158, October 1986.

[dB72] Carl de Boor. On Calculating with B-Splines. *Journal of Approximation Theory*, 6:50–62, 1972.

[dB76] Carl de Boor. Splines as linear combinations of B-splines. A Survey. In George G. Lorentz, Charles K. Chui, and Larry L. Schumaker, editors, *Approximation Theory II: Proceedings of an international symposium conducted by the University of Texas*, pages 1–47, Austin, Texas, January 1976. Academic Press (New York).

[dB86] Carl de Boor. Splines as linear combinations of B-splines. A Survey, 1986. emended version of article [dB76] in Approximation Theory, II.

[dB02] Carl de Boor. Spline Basics. In Gerald Farin, Josef Hoschek, and Myung-Soo Kim, editors, *Handbook of Computer Aided Geometric Design*, chapter 6, pages 141–163. Elsevier, 1st edition, August 2002.

[dBCvKO08] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag Berlin Heidelberg, 3rd edition, March 2008.

[Del34] Boris Nikolaevich Delaunay. Sur la sphère vide. *Izvestia Akademii Nauk SSSR, Otdelenie Matematicheskikh i Estestvennykh Nauk*, 7:793–800, 1934.

[DM06] René Descartes and Ian Maclean. *A Discourse on the Method*. Oxford University Press, January 2006.

[DRS06] Robert L. (Scot) Drysdale, Günter Rote, and Astrid Sturm. Approximation of an open polygonal curve with a minimum number of circular arcs. In *Proceedings of the 22st European Workshop on Computational Geometry (EWCG)*, pages 25–28, Delphi, Greece, 2006.

[DRS08] Robert L. (Scot) Drysdale, Günter Rote, and Astrid Sturm. Approximation of an open polygonal curve with a minimum number of circular arcs and biarcs. *Computational Geometry: Theory and Applications*, 41:31–47, 2008.

[DSL25] René Descartes, David Eugene Smith, and Marcia L. Latham. *The Geometry of René Descartes*. The Open Court Publishing Company, 1637, 1925. Translated from the French and Latin. With a facsimile of the first edition, 1637.

[Eib02] Johannes Eibl. Approximation of Planar Curves Within an Asymmetric Tolerance Band. Master's thesis, Universität Salzburg, FB Computerwissenschaften, Salzburg, Austria, 2002.

[Euk03] Euklid. *Die Elemente. Buch I - XIII*. Ostwalds Klassiker der exakten Wissenschaften, Band 235. Harri Deutsch, extended edition, 2003. German translation by Clemens Thaer.

[Far83] Gerald Farin. Algorithms for rational Bézier curves. *Computer-Aided Design*, 15(2):73–77, 1983.

[Far84] Gerald Farin. Some aspects of car body design at Daimler-Benz. In Robert Barnhill, Wolfgang Boehm, and Josef Hoschek, editors, *Surfaces in Computer Aided Geometric Design*, pages 93–98. North-Holland, 1984.

[Far96] Gerald Farin. *Curves and Surfaces for Computer-Aided Geometric Design - A Practical Guide*. Computer Science and Scientific Computing. Academic Press Inc, 4th edition, October 1996.

[Far02] Gerald Farin. A History of Curves and Surfaces in CAGD. In Gerald Farin, Josef Hoschek, and Myung-Soo Kim, editors, *Handbook of Computer Aided Geometric Design*, chapter 1, pages 1–21. Elsevier, 1st edition, August 2002.

[Far12] Rida T. Farouki. The Bernstein polynomial basis: A centennial retrospective. *Computer Aided Geometric Design*, 29(6):379–419, 2012.

[Faw06] Tom Fawcett. An introduction to ROC analysis. *Pattern Recognition Letters*, 27:861–874, 2006.

[Fer64] James Ferguson. Multivariable Curve Interpolation. *Journal of the Association for Computing Machinery*, 11(2):221–228, April 1964.

[FG96] Rida T. Farouki and T.N.T. Goodman. On the optimal stability of the Bernstein basis. *Mathematics of Computation*, 65(216):1553–1566, October 1996.

[FHK02] Gerald Farin, Josef Hoschek, and Myung-Soo Kim, editors. *Handbook of Computer Aided Geometric Design*. Elsevier, 1st edition, August 2002.

[FN90] Rida T. Farouki and C. A. Neff. On the numerical condition of Bernstein-Bézier subdivision process. *Mathematics of Computation*, 55(192):637–647, October 1990.

[For72] A. R. Forrest. Interactive interpolation and approximation by Bézier polynomials. *The Computer Journal*, 15(1), February 1972.

[FR87] Rida T. Farouki and V. T. Rajan. On the numerical condition of polynomials in Bernstein form. *Computer Aided Geometric Design*, 4(3):191–216, 1987.

[Gal99] Jean Gallier. *Curves and Surfaces in Geometric Modeling: Theory & Algorithms*. The Morgan Kaufmann Series in Computer Graphics. Morgan Kaufmann, 1st edition, October 1999.

[Gau74] Walter Gautschi. Norm estimates for inverses of Vandermonde matrices. *Numerische Mathematik*, 23(4):337–347, 1974.

[Gib98] Josiah Williard Gibbs. Fourier's Series. *Nature*, 59(1522):200, December 1898. Letter to the Editors, http://www.nature.com/nature/journal/v59/n1522/index.html.

[Gib99] Josiah Williard Gibbs. Fourier's Series. *Nature*, 59(1539):606, April 1899. Letter to the Editors, http://www.nature.com/nature/journal/v59/n1539/index.html.

[GP03] Henryk Gzyl and José Luis Palacios. On the Approximation Properties of Bernstein Polynomials via Probabilistic Tools. *Boletín de la Asociación Matemática Venezolana*, 10(1):5–14, 2003.

[GR74a] William J. Gordon and Richard F. Riesenfeld. B-Spline Curves and Surfaces. In Robert E. Barnhill and Richard F. Riesenfeld, editors, *Computer Aided Geometric Design: proceedings of a conference held at the University of Utah*, pages 95–126, Salt Lake City, Utah, March 1974. Academic Press.

[GR74b] William J. Gordon and Richard F. Riesenfeld. Bernstein-Bézier Methods for the Computer-Aided Design of Free-Form Curves and Surfaces. *Journal of the Association for Computing Machinery*, 21(2):293–310, April 1974.

[Gra72] Ronald L. Graham. An Efficient Algorithm for Determining the Convex Hull of a Finite Planar Set. *Information Processing Letters*, 1(4):132–133, 1972.

[GS97] David Gottlieb and Chi-Wang Shu. On the Gibbs Phenomenon and its Resolution. *SIAM Review*, 39(4):664–668, December 1997.

[Gut84a] Antonin Guttman. R-trees: a dynamic index structure for spatial searching. *SIGMOD Rec.*, 14(2):47–57, June 1984.

[Gut84b] Antonin Guttman. R-trees: a dynamic index structure for spatial searching. In *Proceedings of the 1984 ACM SIGMOD international conference on Management of data*, SIGMOD '84, pages 47–57, New York, NY, USA, 1984. ACM.

[GY83] Ronald L. Graham and F. Frances Yao. Finding the Convex Hull of a Simple Polygon. *Journal of Algorithms*, 4:324–331, 1983.

[HA90] Hans-Jürgen Hochfeld and Martin Ahlers. Role of Bézier curves and surfaces in the Volkswagen CAD approach from 1967 to today. *Computer-Aided Design*, 22(9):598–607, November 1990.

[Han02] Dianne Hansford. Bézier Techniques. In Gerald Farin, Josef Hoschek, and Myung-Soo Kim, editors, *Handbook of Computer Aided Geometric Design*, chapter 3, pages 43–73. Elsevier, 1st edition, August 2002.

[HE05] Martin Held and Johannes Eibl. Biarc Approximation of Polygons Within Asymmetric Tolerance Bands. *Computer-Aided Design*, 37(4):357—371, April 2005.

[Hea08] Thomas Little Heath. *The Thirteen Books of Euclid's Elements*, volume 1-3. Cambridge University Press, 1908.

[Hei06] Martin Heimlich. Curve Approximation with Biarcs, Elliptic Biarcs and Bézier Curves. Diplomarbeit, Universität Salzburg, FB Computerwissenschaften, Jakob-Haringer-Str. 2, A-5020 Salzburg, February 2006. http://www.cosy.sbg.ac.at/~held/projects/biarcs/biarcs.html.

[Hel91] Martin Held. *On the Computational Geometry of Pocket Machining*, volume 500 of *Lecture Notes in Computer Science*. Springer-Verlag, June 1991.

[Hel98] Martin Held. Voronoi diagrams and offset curves of curvilinear polygons. *Computer-Aided Design*, 30(4):287–300, April 1998.

[Hel01] Martin Held. VRONI: An engineering approach to the reliable and efficient computation of Voronoi diagrams of points and line segments. *Computational Geometry: Theory and Applications*, 18(2):95–123, March 2001.

[Hel10] Martin Held. Voronoi Diagramm. lecture notes on *Algorithmische Geometrie*, June 2010. http://www.cosy.sbg.ac.at/~held/teaching/compgeo/comp_geo.html.

[Hel11] Martin Held. VRONI and ARCVRONI: Software for and Applications of Voronoi Diagrams in Science and Engineering. In *Eighth International Symposium on Voronoi Diagrams in Science and Engineering (ISVD)*, pages 3–12, Qingdao, China, June 2011.

[HH79] Edwin Hewitt and Robert E. Hewitt. The Gibbs-Wilbraham phenomenon: An episode in fourier analysis. *Archive for History of Exact Sciences*, 21(2):129–160, 1979.

[HH08a] Martin Heimlich and Martin Held. Biarc Approximation, Simplification and Smoothing of Polygonal Curves by Means of Voronoi-based Tolerance Bands. *International Journal of Computational Geometry and Applications*, 18(3):221–250, June 2008.

[HH08b] Martin Held and Stefan Huber. Topological Considerations for the Incremental Computation of Voronoi Diagrams of Circular Arcs. In *Proceedings of the 24th European Workshop on Computational Geometry (EuroCG 2008)*, pages 217–220, Nancy (France), March 2008.

[HHD98] Ismail Haritaoglu, David Harwood, and Larry S. Davis. Ghost: a human body part labeling system using silhouettes. In *Proceedings of the Fourteenth International Conference on Pattern Recognition*, volume 1, pages 77–82, August 1998.

[Hig02] Nicholas J. Higham. *Accuracy and Stability of Numerical Algorithms*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2nd edition, 2002.

[HKM95] Martin Held, James T. Klosowski, and Joseph S.B. Mitchell. Evaluation of Collision Detection Methods for Virtual Reality Fly-Throughs. In *Proceedings of the Seventh Canadian Conference on Computer Geometry*, pages 205–210, 1995.

[HS92] John Hershberger and Jack Snoeyink. Speeding Up the Douglas-Peucker Line-Simplification Algorithm. In P. Bresnahan, E. Corwin, and D. Cowen, editors, *Proceedings of the 5th International Symposium on Spatial Data Handling*, pages 134–143, Charleston, SC, 1992. IGU Commission of GIS.

[HS94] John Hershberger and Jack Snoeyink. An $n \log n$ Implementation Of The Douglas-Peucker Algorithm For Line Simplification. In *Proceedings of the tenth annual*

symposium on Computational geometry (SCG '94), pages 383–384, New York, NY, USA, 1994. ACM.

[HS09] Martin Held and Christian Spielberger. A smooth spiral tool path for high speed machining of 2D pockets. *Computer-Aided Design*, 41(7):539 – 550, 2009.

[Hub96] Philip M. Hubbard. Approximating Polyhedra with Spheres for Time-Critical Collision Detection. *ACM Transactions on Graphics*, 15(3):179–210, July 1996.

[Jar73] R. A. Jarvis. On the identification of the convex hull of a finite set of points in the plane. *Information Processing Letters*, 2(1), March 1973.

[Jer11] Abdul J. Jerri, editor. *Advances in the Gibbs Phenomenon*. Sampling Publishing, 1st edition, March 2011.

[Jes07] Douglas M. Jesseph. Descartes, Pascal, and the Epistemology of Mathematics: The Case of the Cycloid. *Perspectives on Science*, 15(4):410–433, Winter 2007. http://muse.jhu.edu/journals/posc/summary/v015/15.4jesseph.html.

[Joy00] Kenneth I. Joy. Bernstein Polynomials. On-Line Geometric Modeling Notes, November 2000. http://idav.ucdavis.edu/education/CAGDNotes/CAGDNotes/Bernstein-Polynomials.pdf.

[JT70] Michael A. Jenkins and Joseph F. Traub. A Three-Stage Algorithm for Real Polynomials Using Quadratic Iteration. *SIAM Journal on Numerical Analysis*, 7(4):545–566, December 1970.

[Kar58] Otto J. Karst. Linear Curve Fitting Using Least Deviations. *Journal of the American Statistical Association*, 53(281):118–132, March 1958.

[Kli90] Morris Kline. *Mathematical thought from ancient to modern times*, volume 1. Oxford University Press, three-volume paperback edition edition, March 1990.

[Knu92] Donald Ervin Knuth. Two Notes on Notation. *The American Mathematical Monthly*, 99(5):403–422, May 1992.

[Knu98] Donald Ervin Knuth. *The Art of Computer Programming*, volume 3, Sorting and Searching. Addison-Wesley, 2nd edition, 1998.

[KS86] David G. Kirkpatrick and Raimund Seidel. The Ultimate Planar Convex Hull Algorithm? *SIAM Journal on Computing*, 15(1):287–299, 1986.

[Lee82] D. T. Lee. Medial Axis Transformation of a Planar Shape. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 4(4):363–369, July 1982.

[LH08] Francois Lekien and George Haller. Unsteady flow separation on slip boundaries. *Physics of Fluids*, 20(097101), 2008.

[Lin08] Johann Linhart. Geometrie. Eine Vorlesung von Johann Linhart im Sommersemester 2008, August 2008. http://www.sbg.ac.at/mat/staff/linhart/geom.pdf.

[Loc61] E. H. Lockwood. *Book of Curves*. Cambridge University Press, 1961.

[Lor86] George G. Lorentz. *Bernstein polynomials*. Chelsea Publishing Company, 2nd edition, 1986.

[MA79] Duncan McCallum and David Avis. A linear algorithm for finding the convex hull of a simple polygon. *Information Processing Letters*, 9(5):201–206, December 1979.

[Mai10] Georg Maier. *Smooth Minimum Arc Paths. Contour Approximation with Smooth Arc Splines*. Industriemathematik und Angewandte Mathematik. Shaker Verlag, Universität Passau, Fakultät für Informatik und Mathematik, March 2010.

[Mal07] Joe Malkevitch. Taxi!, October 2007. http://www.ams.org/samplings/feature-column/fcarc-taxi.

[Man72] Lois Mansfield. On the Variational Characterization and Convergence of Bivariate Splines. *Numeric Mathematics*, 20:99–114, 1972.

[Mel87] Avraham A. Melkman. On-Line Construction of the Convex Hull of a Simple Polyline. *Information Processing Letters*, 25:11–12, April 1987.

[Mor99] Michael Mortenson. *Mathematics for Computer Graphics Applications*. Industrial Press, Inc., 2nd edition, January 1999.

[Mun99] James R. Munkres. *Topology*. Pearson US Imports & PHIPEs, 2nd edition, Dezember 1999.

[MW92] Dereck S. Meek and Desmond J. Walton. Approximation of discrete data by G1 arc splines. *Computer-Aided Design*, 24(6):301–06, 1992.

[NF90] Sergei Petrovich Novikov and Anatoly Timofeevich Fomenko. *Basic Elements of Differential Geometry and Topology*, volume 60 of *Mathematics and its Applications*. Kluwer Academic Publishers, 1 edition, November 1990.

[Now06] Horst Nowacki. *Developments in Fluid Mechanics Theory and Ship Design before Trafalgar*, volume 308 of *Preprint*. Max-Planck-Institut für Wissenschaftsgeschichte, April 2006.

[OP03] Halil Oruç and George M. Phillips. q-Bernstein polynomials and Bézier curves. *Journal of Computational and Applied Mathematics*, 151:1–12, 2003.

[OR99] John J. O'Connor and Edmund F. Robertson. Doubling the cube. MacTutor History of Mathematics archive, School of Mathematics and Statistics, University of St Andrews, Scotland, April 1999. http://www-history.mcs.st-and.ac.uk/HistTopics/Doubling_the_cube.html.

[OR03] John J. O'Connor and Edmund F. Robertson. Thomas Little Heath. MacTutor History of Mathematics archive, School of Mathematics and Statistics, University of St Andrews, Scotland, October 2003. http://www-groups.dcs.st-and.ac.uk/~history/Biographies/Heath.html.

[Pal03] Richard S. Palais. A Modern Course on Curves and Surfaces, Fall 2003. http://virtualmathmuseum.org/Surface/a/bk/curves_surfaces_palais.pdf.

[PBP02] Hartmut Prautzsch, Wolfgang Boehm, and Marco Paluszny. *Bézier and B-Spline Techniques*. Number XIV in Mathematics and Visualization. Springer Berlin Heidelberg, August 2002.

[PH77] Franco P. Preparata and S. J. Hong. Convex Hulls of Finite Sets of Points in Two and Three Dimensions. *Communications of the ACM*, 20(2), February 1977.

[Phi96] George M. Phillips. On generalized Bernstein polynomials. In A. R. Mitchell, D. F. Griffits, and G. A. Watson, editors, *Numerical Analysis (A. R. Mitchell 75th Birthday Volume)*, pages 263–269. World Scientific, 1996.

[Phi03] George M. Phillips. *Interpolation and Approximation by Polynomials*, volume 14 of *CMS Books in Mathematics*. Springer, April 2003.

[PL02] Helmut Pottmann and Stefan Leopoldseder. Geometries for CAGD. In Gerald Farin, Josef Hoschek, and Myung-Soo Kim, editors, *Handbook of Computer Aided Geometric Design*, chapter 3, pages 43–73. Elsevier, 1st edition, August 2002.

[Pla72] Robin L. Plackett. Studies in the History of Probability and Statistics. XXIX: The Discovery of the Method of Least Squares. *Biometrika*, 59(2):239–251, August 1972.

[PM02] Nicholas M. Patrikalakis and Takashi Maekawa. *Shape Interrogation for Computer Aided Design and Manufacturing*. Springer Berlin Heidelberg, 1st edition, February 2002.

[PS85] Franco P. Preparata and Michael Ian Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, August 1985.

[PT97] Les A. Piegl and Wayne Tiller. *The NURBS Book*, volume XIV of *Monographs in Visual Communication*. Springer Verlag, 2nd edition, 1997.

[PV06] Shi Pu and George Vosselman. Automatic extraction of building features from terrestrial laser scanning. In *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, volume 36, part 5, Dresden, Germany, September 2006.

[Ram72] Urs Ramer. An iterative procedure for polygonal approximation of plane curves. *Computer Graphics and Image Processing*, 1(3):244–256, November 1972.

[Rat94] John G. Ratcliffe. *Foundations of Hyperbolic Manifolds*, volume 149 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, 2nd edition, September 1994.

[Rau08] Martin Raussen. Elementary Differential Geometry: Curves and Surfaces, 2008. http://people.math.aau.dk/~raussen/INSB/AD2-11/book.pdf.

[Rie75] Richard F. Riesenfeld. On Chaikin's algorithm. *Computer Graphics and Image Processing*, 4(3):304–310, 1975.

[RR86] Jaroslaw R. Rossignac and Aristides A.G. Requicha. Offsetting operations in solid modelling. *Computer Aided Geometric Design*, 3:129–148, 1986.

[Run01] Carl Runge. Über empirische Funktionen und die Interpolation zwischen äquidistanten Ordinaten. *Zeitschrift für Mathematik und Physik*, 46:224–243, 1901.

[Sch46] Isaac Jacob Schoenberg. Contributions to the problem of approximation of equidistant data by analytic functions. *Quarterly of Applied Mathematics*, 4:45–99, 112–141, 1946.

[Sch73] Isaac Jacob Schoenberg. *Cardinal Spline Interpolation*. CBMS-NSF Regional Conference Series in Applied Mathematics. Society for Industrial and Applied Mathematics, Philadelphia, Pennsylvania, 1973.

[Sed11] Thomas W. Sederberg. Computer Aided Geometric Design, January 2011. http://tom.cs.byu.edu/~557/text/cagd.pdf.

[SH75] Michael Ian Shamos and Dan Hoey. Closest-Point Problems. In *16th Annual Symposium on Foundations of Computer Science*, October 1975.

[SH76] Michael Ian Shamos and Dan Hoey. Geometric Intersection Problems. *17th Annual IEEE Symposium on Foundations of Computer Science (FOCS 1976)*, pages 208–215, October 1976.

[Shm07] Yuriy Shmaliy. *Continuous-Time Systems (Signals and Communication Technology)*. Signals and Communication Technology. Springer Netherlands, 1st edition, September 2007.

[Ski08] Steven S. Skiena. *The Algorithm Design Manual*. Springer London, 2008.

[Spi10] Christian Spielberger. Uniform Cubic B-Splines. Technical report, FB Computerwissenschaften, November 2010. http://www.cosy.sbg.ac.at/~cspiel/reports/splines.pdf.

[SSS74] Ivan E. Sutherland, Robert F. Sproull, and Robert A. Schumacker. A Characterization of Ten Hidden-Surface Algorithms. *Computing Surveys*, 6(1), March 1974.

[Sti10] John Stillwell. *Mathematic and Its History*. Undergraduate Texts in Mathematics. Springer, 3rd edition, August 2010.

[Str88] Dirk J. Struik. *Lectures on Classical Differential Geometry*. Dover Publications, 2nd edition, April 1988.

[Sut64] Ivan E. Sutherland. Sketch pad. a man-machine graphical communication system. In *Proceedings of the SHARE design automation workshop*, DAC '64, pages 6329–6346, New York, NY, USA, 1964. ACM.

[The10] The CGAL Project. *CGAL User and Reference Manual*. CGAL Editorial Board, 3.7 edition, 2010. http://www.cgal.org/Manual/3.7/doc_html/cgal_manual/packages.html.

[Tot94] Vilmos Totik. Approximation by Bernstein Polynomials. *American Journal of Mathematic*, 116(4):995–1018, August 1994. http://www.jstor.org/stable/2375007.

[Vin10] John Vince. *Mathematics for computer graphics*. Springer-Verlag, 3rd edition, February 2010.

[Wan37] Pierre-Laurent Wantzel. Recherches sur les moyens de reconnaître si un Problème de Géométrie peut se résoudre avec la règle et le compas. *Journal de mathématiques*, pages 366–372, 1837.

[Wei85a] Karl Weierstraß. Über die analytische Darstellbarkeit sogenannter willkürlicher Functionen einer reellen Veränderlichen. Erste Mittheilung. In *Sitzungsberichte der Königlich Preußischen Akademie der Wissenschaften zu Berlin*, Digitalisierte Akademieschriften, pages 633–639. Berlin-Brandenburgische Akademie der Wissenschaften, 2001, 1885. http://bibliothek.bbaw.de/bibliothek-digital/digitalequellen/schriften/anzeige?band=10-sitz/1885-2&seite:int=00000109.

[Wei85b] Karl Weierstraß. Über die analytische Darstellbarkeit sogenannter willkürlicher Functionen einer reellen Veränderlichen. Zweite Mittheilung. In *Sitzungsberichte der Königlich Preußischen Akademie der Wissenschaften zu Berlin*, Digitalisierte Akademieschriften, pages 789–805. Berlin-Brandenburgische Akademie der Wissenschaften, 2001, 1885. http://bibliothek.bbaw.de/bibliothek-digital/digitalequellen/schriften/anzeige?band=10-sitz/1885-2&seite:int=00000272.

[Wei10] Ron Wein. 2D Minkowski Sums. In *CGAL User and Reference Manual*. CGAL Editorial Board, 3.7 edition, 2010. http://www.cgal.org/Manual/3.7/doc_html/cgal_manual/packages.html#Pkg:MinkowskiSum2.

[Wil48] Henry Wilbraham. On a certain periodic function. *Cambridge & Dublin Mathematical Journal*, 3:198–201, April 1848.

[WZ77] Richard L. Wheeden and Antoni Zygmund. *Measure and Integral: An Introduction to Real Analysis*, volume 43 of *Monographs and textbooks in pure and applied mathematics*. Marcel Dekker Inc, 1st edition, June 1977.

[Yap87] Chee K. Yap. An O(n log n) Algorithm for the Voronoi Diagram of a Set of Simple Curve Segments. *Discrete & Computational Geometry*, 2(1):365–393, December 1987.