Philipp M. Wagner, BSc

Uniform cubic B-spline fitting in a class A modeling environment

MASTER'S THESIS

to achieve the university degree of

Master of Science

Master's degree programme: Information and Computer Engineering

submitted to

Graz University of Technology

Supervisor

Priv.-Doz. Dipl.-Inform. Dr.-Ing. Sven Havemann

Institute of Computer Graphics and Knowledge Visualization

Faculty of Computer Science and Biomedical Engineering

Graz, May 2018

AFFIDAVIT

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master‘s thesis.

08.05.2018

Date

[illegible]

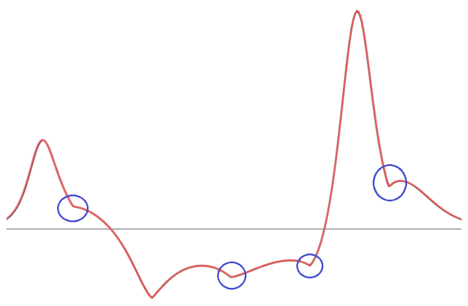
Signature

PHILIPP M. WAGNER

UNIFORM CUBIC B-SPLINE FITTING IN A CLASS A MODELING ENVIRONMENT

UNIFORM CUBIC B-SPLINE FITTING IN A CLASS A MODELING ENVIRONMENT

Philipp M. Wagner, BSc



Diploma Thesis

Supervisor:
Priv.-Doz. Dipl.-Inform. Dr.-Ing. Sven Havemann

Institute of Computer Graphics and Knowledge Visualization
Faculty of Computer Science and Biomedical Engineering
Graz University of Technology

May 2018

ABSTRACT

B-splines have been a least common denominator in computer aided geometric design for a very long time, the uniform cubic type being particularly popular for its simplicity, fast means of evaluation and inherent $\mathbb{C}^2$ continuity. While most of the existing approaches to B-spline fitting first and foremost concern themselves with the spatial error of the fit, automotive design in addition puts a strong emphasis on curvature quality, which directly relates to the aesthetic quality of the final product's shape. In German automotive design the term class A describes shapes of the highest quality, such as the outer body of a car. B-splines may be described either by a set of control points, or by the projections of these control points to the limit shape of the spline under subdivision, namely limit points or Greville points. In this work we present a novel approach to distribute limit points over a discrete input curve, in order to obtain an initial solution for a B-spline curve which can further be optimized by constrained least squares techniques. Our approach is driven by the local curvature estimates of the input data and an error term, and we show that it obtains strong links to B-spline artifact analysis. Although our approach performs well, it does not yield shapes of class A quality. Further investigation shows that B-splines obtain inherent traits which make them hardly applicable to class A design. These deficiencies result in unexpected and uncontrollable curvature behavior, and while showing most prominently in the cubic case, they exist on all B-spline degrees. In response we further discuss the specific demands of class A design on a design curve, and present a viable alternative to classic B-spline modeling which is based on clothoid segments. It shows that such a representation can ease the process of class A design in a very natural way, due to its superior control over curvature.

The current of the flowing river does not cease, and yet the water is not the same water as before. The foam that floats on stagnant pools, now vanishing, now forming, never stays the same for long. So, too, it is with the people and dwellings of the world.

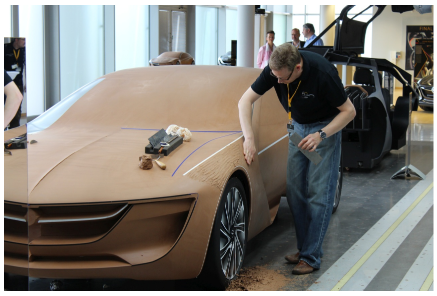
—
Hōjōki by Kamo no Chōmei

ACKNOWLEDGMENTS

To all people that bore with me during the course of this work with Zen-like patience. First and foremost my supervisor and my mum. Then my fellow students. They always believed that we would have that Chinese celebration dinner some day.

CONTENTS

1 INTRODUCTION


(a) Clay model.

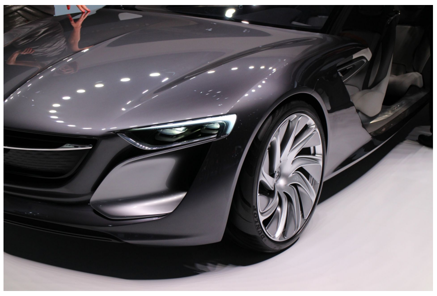

(b) Final product.

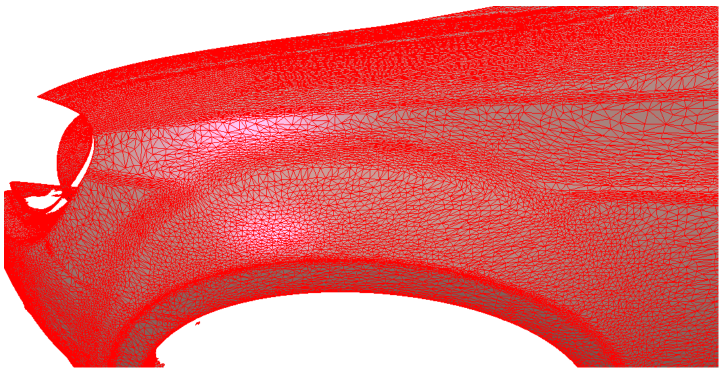
Figure 1: From clay model to automobile.

This thesis evolved as part of the *SurfaceReconstruction* software framework, a project cooperation between CGV[1] and Volkswagen AG[2]. In the automotive industry it is often desired to convert physical objects to digital 3D models. Such physical objects could be prototype designs of a car body made of clay (figure 1), or final car parts already in production. A laser scanner is most commonly used for digitization of such objects, generating a point cloud that is further converted into a 3D triangle mesh consisting of up to several millions of triangles. A detail of such a triangle mesh is shown in figure 2.

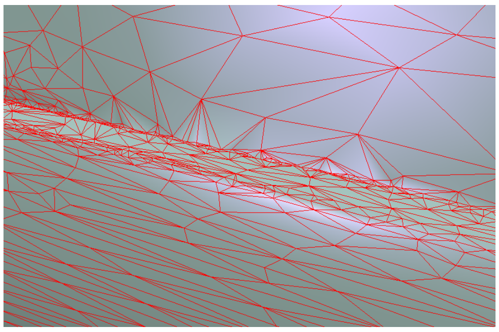
When being used as input to further processing or as a final data representation for storage, triangle meshes show some severe drawbacks.

- **Regularity**: Scanned objects, such as hand-crafted clay models, usually do not obtain the high surface quality that needs to be achieved for the final digital model (figure 2b). The laser scanner might introduce additional irregularities due to its limited spatial accuracy and irregular sampling (figure 2c). This makes further processing inevitable.
- **Controllability**: Manipulation of a triangle mesh is a very tedious and time consuming task, since every vertex has to be manipulated completely independent of its neighborhood. It is easy to imagine that even with only a few hundred vertices it is nearly impossible to produce a coherent, smooth and regular

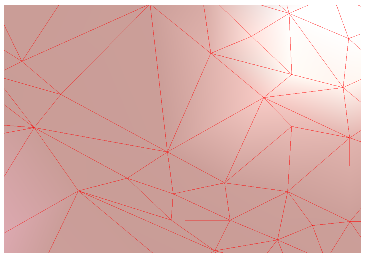
1 CGV: Institute of Computer Graphics and Knowledge Visualization, Graz University of Technology, http://www.cgv.tugraz.at
2 Volkswagen AG, http://www.volkswagen.de



(a) Triangle mesh obtained from a scan.



(b) The highlight clearly shows the irregularities in the triangle mesh.



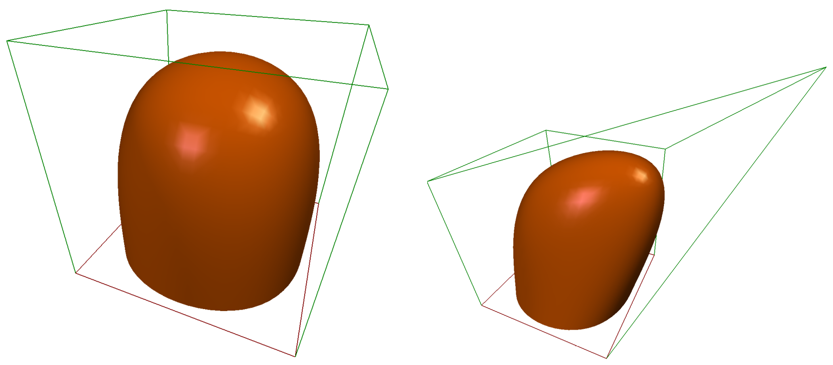
(c) Inhomogeneous sampling of the laser scanner.

Figure 2: Triangle mesh details.

result when manually editing a certain mesh region. Also automation of modeling processes becomes very involved. It is a very complex spatial representation and thus hard to control.

- **Disk space**: Triangle meshes devour much disk space, since all vertex positions and triangle indices have to be stored.
- **Refinement**: Changing mesh resolution (e.g. needed for level of detail calculation) is computationally intensive. Further, a triangle mesh is only defined at discrete positions and new positions on the surface can only be estimated.

Therefore usually a parametric surface representation is desired, which is defined and can be manipulated via a small set of *control points*. The analytic surface definition of such a parametric representation changes implicitly if one of the control points is varied, thus by moving only a few points in space whole parts of a 3D model can be edited without loosing consistency and smoothness. This makes modeling much more efficient and convenient, and enables the creation of high quality surfaces. A surface now can be defined uniquely by the control points, which reduces the amount of data needed for storage drastically. As for the last of the above-mentioned drawbacks, since being defined analytically, the surface can be evaluated at runtime in



(a) The cube represents the control mesh of the bronze colored model.

(b) The shape can be manipulated by moving one of the control meshes' vertices.

Figure 3: Manipulation of a 3D shape through control points.

as much detail as needed. This often can be done using very efficient iterative schemes, making resolution changes computationally cheap. Figure 3 shows how a 3D shape can be edited via its control points. As we can see, the control points form a sparse mesh on their own, which we will refer to as the *control mesh* of the parametric surface.

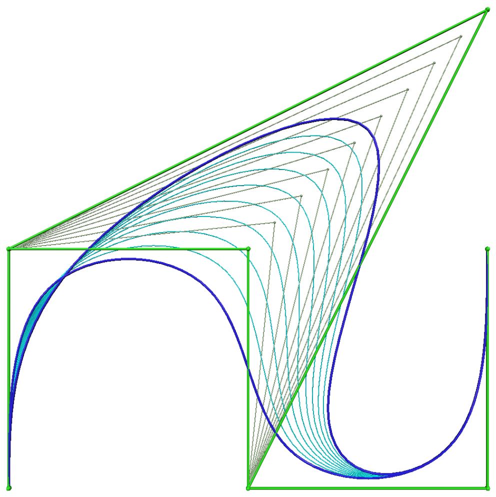


Figure 4: Curve manipulation using control points. The curve remains consistent and smooth.

Similar to surfaces, we can express many parametric curves via control points. Figure 4 shows the manipulation of a curve through its *control polygon*. Note how whole regions of the curve can be manipulated by moving a single point, without loosing its smoothness and consistency.

Parametric curves often serve as a basis to generate their parametric surface equivalent, uniting curves and surfaces in a common mathematical framework. This property makes parametric curves interesting to be studied before generalizing problems to the surface case.

Subsequently we will utilize the term *geometric primitive* for our parametric data representation, whenever it is necessary to transcend dimensional categories such as »curve« and »surface« or specific types of parametric curves or surfaces in order to discuss common problems and properties.

In this work we will concern ourselves with the well-known problem of fitting a parametric design curve to discrete data. This can be seen as a first step towards the even more complicated task of parametric surface fitting. A classic parametric curve used for data fitting and modeling is the *uniform cubic B-spline curve*. B-splines have been a least common denominator in computer aided geometric design (CAGD) for a very long time, the uniform cubic type being particularly popular for its simplicity, fast means of evaluation and inherent $\mathbb{C}^2$ continuity. The uniform cubic B-spline is thus part of most major modeling software packages.

Instead of by its control points a B-spline curve may also be defined by so-called *limit points* (also called *Greville points*), which are interpolated by the generated spline and correspond uniquely to its set of control points. For each limit point a corresponding control point can be obtained and vice versa. Limit points and control points thus are equivalent representations of the B-spline, and we can change between these representations without altering the curve.

B-splines and other basic definitions and mathematical foundations for our work will be treated in chapter 2.

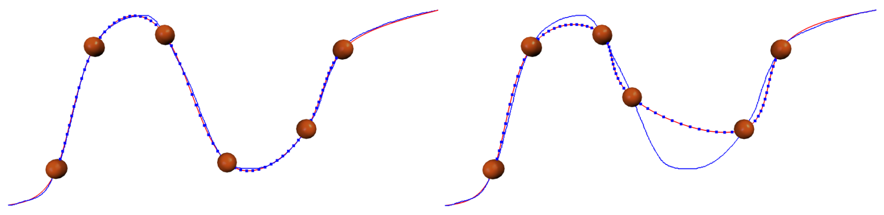

(a) A good configuration? (b) Moving one limit point of a cubic B-spline curve influences all B-spline segments.

Figure 5: Fitting a cubic B-spline to a stroke curve using limit point placement.

Given discrete data, it is a very important and interesting question where to place the control points of a B-spline curve in order to obtain a high quality fit to the data. This still is an unsolved problem in CAGD, and the broad range of solutions suggested to answer it (genetic algorithms, neural networks, etc.) shows the complexity of the task and how much we still lack basic understanding for it.

But why is this task is so challenging? One reason is rooted in the interdependency of control points, limit points and spline segments. Moving one control point will influence a certain number of B-spline segments around it, the number depending on the degree of the polynomial B-spline curve (four in the cubic case). Instead of control points we can also place limit points, which has both advantages and disadvantages, as we will discuss later. Moving a limit point in contrast influences all segments, each limit point depending on several control points. In the uniform cubic case a limit point for instance is a linear combination of three consecutive control points, which causes a chain of propagation over the whole curve, the effect dampening with distance to the moved limit point (see figure 5).

This propagation over multiple nearby segments implies that from an optimization point of view we potentially have to deal with many local minima, and from a model point of view with quite unintuitive behavior.

An important question besides *where* to place the control points is *how many* of them to spend in a certain region of the data in order to obtain a certain quality of the fit. This also attributes to sparseness of the control polygon or mesh, which is much desired, as we will explain later on.

The reproduction of the intended curvature profile with the design primitive - or the generation of an even fairer curvature profile - is another crucial problem of high quality fitting. An important notion in this context is the one of *class A* design, describing surfaces in automotive design that meet the highest quality standards (e.g. the body of a car). While not being defined consistently throughout the industry, it is common opinion that this notion is strongly related to curvature quality. Flaws in curvature can easily be spotted in the reflections of the final product, thus curvature quality is a central factor in automotive design. This also means that a good distance error of the fit alone cannot ensure acceptable shape quality.

Finally, an important aspect of high quality design is the quality of the control mesh itself. It has its own design language and provides new means of communication as an abstraction of shape. Professional surface engineers in the automotive industry are able to infer quality of shape from directly judging the the control mesh itself. It is for instance understood that regularity and sparsity of the control mesh most likely will lead to fairer shapes.

These challenges apply to the curve and surface case alike.

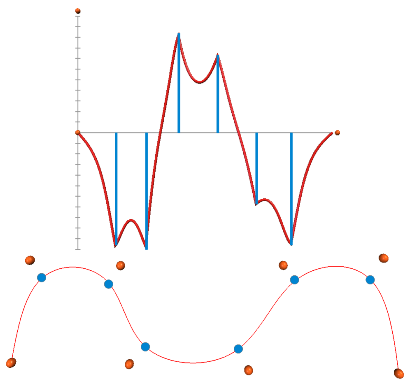
Concentrating on the 2D (or planar) case, the problem we want to solve in this work thus can be summarized as: *Given sampled curve data, place the control points of a uniform cubic B-spline curve such that the curvature of the data is reproduced in a fair way*. Additionally we demand regularity and sparsity of the control polygon, and that we have means to influence the accuracy of the fit.

There are many existing approaches to uniform cubic B-spline fitting that focus on minimizing the distance error of the fit, but to our best knowledge none of them gives proper attention to the important aspects of correct curvature reproduction and curvature quality (chapter 3).

Exploiting the duality of control and limit points, we make use of a method which has its complement in B-spline artifact analysis and provides us with a measure for required limit point spacing at a certain position along the discrete curve. The obtained spacing both respects a certain distance error tolerance and the local curvature estimate of the discrete data. Using this local curvature-driven approach an initial set of B-spline limit points can be retrieved by consolidating all required limit point spacings along the discrete curve. The limit point distribution is regular and its density depends both on the demanded accuracy and the desired degree of regularity. The initial limit points are further optimized globally by using robust constrained least squares methods directly on the limit positions. The control points can always be retrieved from the limit points by making use of an iterative procedure.

The algorithm will be described in full detail in chapter 4. In order to obtain a smooth data basis to interpolate and a reliable curvature profile we can base our measurements on, we will also inspect and compare various methods for discrete data smoothing and robust curvature estimation.

Analysis (section 5.1) shows an interesting result, namely that class A shapes can not be obtained easily with uniform cubic B-splines. This fundamental problem is rooted in the inherent behavior of the B-spline itself - to be precise in the curvature it generates and the difficulty to control it - which makes large scale modeling using cubic B-splines very difficult. Figure 6a clearly emphasizes this. The $\mathbb{C}^2$ continuity of the uniform cubic B-spline leads to continuous but not necessarily smooth curvature, which shows in the strong bends. Further, the curvature extrema that are introduced between the segments cannot be hidden (or even controlled) by regularity of the control polygon alone. B-spline artifact analysis tells us that the shown curvature artifacts are always present, but can be reduced by spending


(a) B-splines introduce unwanted curvature extrema between their segments, which makes them hard to control curvature-wise.

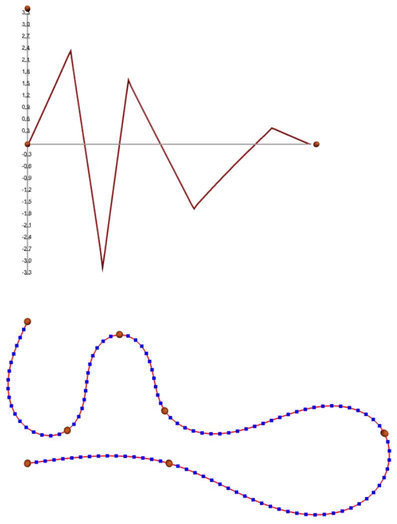

(b) PCC curves as an alternative to classic spline modeling. Clothoidal curves always yield linear curvature change between their segments.

Figure 6: Are B-spline curves suited for class A design?

more limit points or leveraging the B-spline degree, both options not necessarily being desirable.

These findings spawn some interesting questions. We will subsequently discuss what properties a parametric class A surface or curve should obtain in our opinion, and what challenges class A design could pose to automated modeling (section 5.2). We will further discuss why controlling curvature variation and keeping it to a minimum for us is the key to class A design.

Thus, despite generating regular limit point distributions and reproducing the discrete input curvature quite accurately, the presented approach can not ensure shapes which are desired in the high quality design sector (section 5.3).

In order to emphasize our findings, we we will present an alternative to B-spline modeling which makes use of piecewise clothoid curves (PCCs) in section 5.4. We will show how this change of paradigm could make modeling more intuitive and reliable in terms of class A quality (see figure 6b).

Final thoughts will be given in section 5.5.

2 BASIC DEFINITIONS

In this chapter we will discuss some basic definitions that we will need later on.

2.1 PARAMETRIC CURVES

Most curves used in computer aided design can be expressed as parametric curves.

Definition 1 (Parametric curve) *Let the interval* $I \subset \mathbb{R}$ *be called the parameter interval. The function*

$$\mathbf{c} : I \to \mathbb{R}^n$$

is called a parametric curve in $\mathbb{R}^n$*.*

In this work we are interested in parametric 2D curves, which for each parameter value $t \in I$ yield a curve position $\mathbf{c}(t) \in \mathbb{R}^2$, or parametric space curves $\mathbf{c}(t) \in \mathbb{R}^3$.

A parametric curve is said to have an *arc-length parametrization*, if the arc-length between two curve positions $\mathbf{p}(s_0)$ and $\mathbf{p}(s_1)$ equals $|s_1 - s_0|$.

2.2 DISCRETE CURVES: POLYGONAL CURVES

Rather than having a parametric representation, our input data will consist of discrete samples.

A *polygonal curve* in $\mathbb{R}^n$ is a discrete curve representation defined by an ordered sequence of spatial positions.

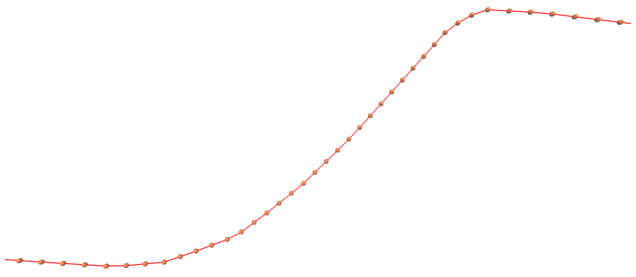
$$P: \quad \mathbf{p}_0, \mathbf{p}_1, \ldots, \mathbf{p}_{m-1}, \mathbf{p}_m \qquad m \geqslant 1, \mathbf{p}_i \in \mathbb{R}^n.$$

Each pair of consecutive points forms a line segment, its length is given by

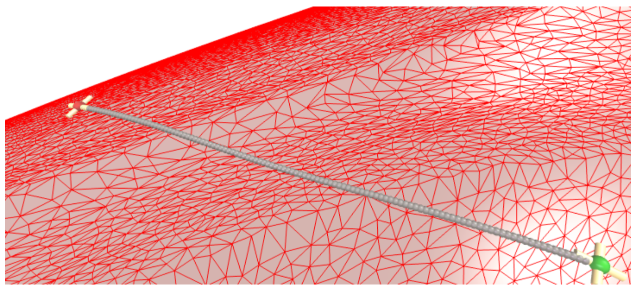
$$d_i = \|\mathbf{p}_{i+1} - \mathbf{p}_i\| \qquad i = 0, \ldots, m-1.$$

A polygonal curve is sometimes also called *piecewise linear curve*.

Such a discrete representation can be obtained in various ways. We can evaluate a parametric curve $\mathbf{c}(t)$ at several parameter values t



(a) Part of a polygonal curve consisting of sampled positions of an analytical curve.



(b) Curve samples taken from a triangle mesh forming a polygonal curve.

Figure 7: Retrieval of polygonal curves.

(figure 7a), or sample a surface along an arbitrary path (figure 7b). In our case we intersect a triangle mesh with a plane (which is especially challenging, as explained in section 4.2).

Throughout this text the term *discrete curve* will be synonymous with *»represented as a polygonal curve«*.

2.3 HAUSDORFF DISTANCE

In order to evaluate the quality of our data fit, some error metric has to be chosen to measure the distance between the resulting parametric curve and the discrete input curve.

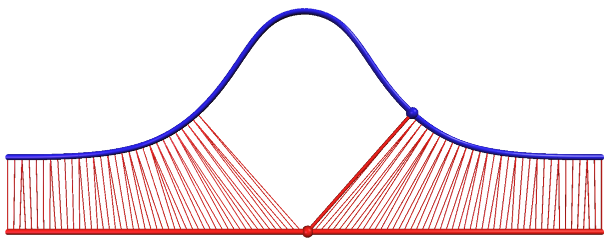
The *one-sided Hausdorff distance* from X to Y can be defined as

Definition 2 (One-sided Hausdorff distance)

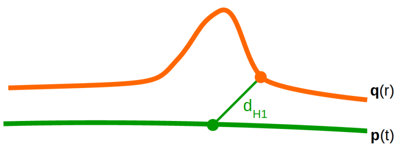
$$d_{H1}(X, Y) = \sup_{x \in X} \inf_{y \in Y} d(x, y),$$

where X and Y are subsets of a metric space M, and $d(x, y)$ is a distance metric.

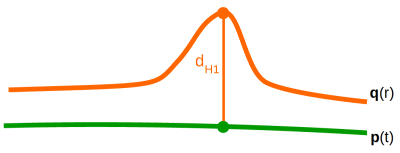
Let now X and Y be the parametric curves $\mathbf{p}(t)$ and $\mathbf{q}(r)$, M the Euclidean space of dimension two or three and $d(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|$



(a) Minimum distances to the blue curve for various positions on the red curve. The supremum of these minimum distances over the whole parameter range equals the one-sided Hausdorff distance d_{H1} (shown here in bold red).



(b) One-sided Hausdorff distance d_{H1}, $\mathbf{p}(t)$ to $\mathbf{q}(r)$.



(c) One-sided Hausdorff distance d_{H1}, $\mathbf{q}(r)$ to $\mathbf{p}(t)$.

Figure 8: One-sided Hausdorff distance between parametric curves: The maximum distance which is needed in order for all points of a curve to reach a point on the other curve.

the Euclidean distance between two curve points. By computing the minimum Euclidean distance to $\mathbf{q}(r)$ for each parameter t of $\mathbf{p}(t)$ and choosing the supremum, we can obtain the one-sided Hausdorff distance between parametric curves. This is illustrated in figure 8a. Switching the direction of measurement can result in quite different results, as illustrated in figures 8bc.

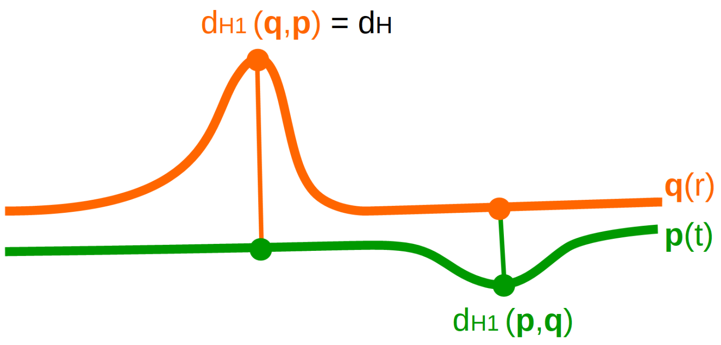


Figure 9: Hausdorff distance d_H between two parametric curves $\mathbf{p}(t)$ and $\mathbf{q}(r)$. The Hausdorff distance is just the maximum of the one-sided Hausdorff distances $d_{H1}(\mathbf{p}, \mathbf{q})$ and $d_{H1}(\mathbf{q}, \mathbf{p})$.

Having defined the one-sided Hausdorff distance, the *Hausdorff distance* between X and Y can be computed as

Definition 3 (Hausdorff distance)

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

which is the maximum of $d_{H1}(X,Y)$ and $d_{H1}(Y,X)$. The Hausdorff distance between two parametric curves is illustrated in figure 9.

If the two curves are represented as polygonal curves P and Q, we can express their discrete Hausdorff distance as follows.

Definition 4 (Hausdorff distance between polygonal curves)

$$d_H(P,Q) = \max\left\{ \max_{p\in P} \min_{q\in Q} \|\mathbf{p}-\mathbf{q}\|\ ,\ \max_{q\in Q} \min_{p\in P} \|\mathbf{q}-\mathbf{p}\| \right\}$$

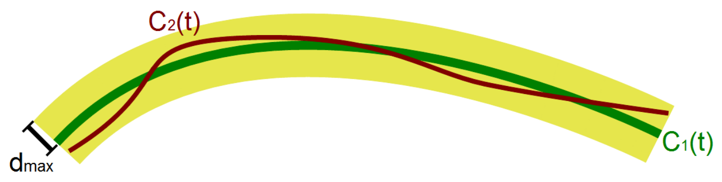


Figure 10: Ensuring a maximum error bound using Hausdorff distance

The Hausdorff distance assures a true maximum error bound. Provided a maximum distance threshold d_{max}, a tube can be imagined around the input curve, inside which the constructed parametric curve has to lie (see figure 10). It is thus our measure of choice to express the final spatial error of the parametric data fit. Remember that the error bound also has to hold vice versa. The input curve equally has to lie inside a restricted tube of width d_{max} around the fitted parametric curve.

2.4 CURVATURE

Although the distance error between the design curve and the discrete data is an important measure, it cannot really capture the visual quality of the result. We thus need an additional measure of quality in order to ensure visually pleasing results.

Curvature is maybe the most important and common descriptor of shape quality. Basically it does exactly what its name tells us, describing the »curviness« of a geometric object. But the so called *curvature profile* - curvature plotted over the parameter space - is above all an enormously useful and intuitive tool to analyze the behavior of a shape and to judge its aesthetic qualities. Flaws which may not even be visible to the eye when inspecting the shape itself, become very

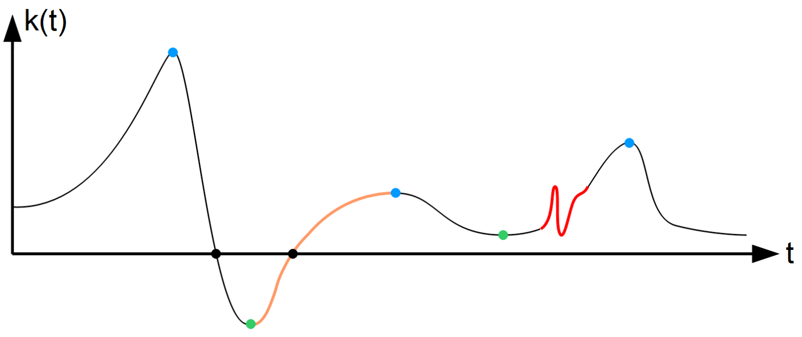


Figure 11: *Shape analysis using a curvature profile. black*: Inflection points. *blue, green*: Curvature extrema, regions where the shape bends strongly. *orange*: Transition behavior between curvature extrema. *red*: Spontaneous variation. The orange section between the two extrema is smooth and strictly monotonic. This is the behavior one would expect from a high quality shape. The high frequency red section may signal a flaw in the model. Except for the red section, this profile may be called »fair«, with its few clearly defined extrema and pleasant transitions. One can also see that the curve at least has to be of continuity class $\mathbf{G}^3$, since it joins smoothly at any curve position.

evident and exposed in the curvature profile. Every feature on the shape can be identified easily as a curvature extremum and we can assess amplitude and frequency of such features at a glance. Figure 11 illustrates how a shape can be analyzed by consulting its curvature profile.

To us curvature is of the utmost importance out of many reasons.

- Curvature directly relates to the reflections on a surface and thus to surface quality and aesthetics of the final consumer product. The human visual perception reacts very sensitive to flaws in surface reflections, thus the reflections on a high quality surface are investigated thoroughly in automotive design. So-called *reflection lines* can be created on a physical object by projecting multiple parallel light sources onto it, or for a computer model by simulating this process in software. The created pattern of stripes makes it easy to spot flaws such as curvature discontinuities on the surface.

- Controlling curvature and its variation for us is the key to high quality modeling.

- Accordingly, reproducing the intended curvature trend of the input data and ensuring a certain curvature quality for us is the key to high quality parametric fitting and automated modeling.

- In order to achieve these goals, we are basing our measurements on the curvature values of the discrete data. We thus rely on accurate curvature estimation.

Since curvature and its quality are so fundamental to our work, we will quickly review some important definitions for it. They can be found e.g. in [8]. In a later chapter we will deal with the problem of robustly estimating curvature from discrete and noisy data, and discuss the topic of curvature quality in more detail.

The curvature of an arc-length parametrized curve just equals its second derivative.

Definition 5 (Curvature of an arc-length parametrized curve)

$$\kappa = \|\mathbf{k}''(s)\| = \left\| \frac{d^2\mathbf{k}(s)}{ds^2} \right\| = \left\| \frac{d\mathbf{T}(s)}{ds} \right\|.$$

Thus we can also interpret curvature as rotation speed of the unit normal vector **T**.

In the general case, the curvature of a planar curve $\mathbf{k}(t) = (x(t), y(t))$ is defined as

Definition 6 (Curvature of a planar curve)

$$\kappa = \frac{x'y'' - x''y'}{(x'^2 + y'^2)^{3/2}}.$$

The curvature of a graph $y = f(x)$ is defined as

Definition 7 (Curvature of a graph)

$$\kappa = \frac{f''}{(1 + f'^2)^{3/2}}.$$

The curvature of a general space curve $\mathbf{k}(t) = (x(t), y(t), z(t))$ is defined as

Definition 8 (Curvature of a space curve)

$$\kappa = \frac{\|\mathbf{k}' \times \mathbf{k}''\|}{(\mathbf{k}'\mathbf{k}')^{3/2}}.$$

Only absolute curvature values are returned, so the sign has to be retrieved some other way if needed, e.g. by fixing some convention of sign externally. In the two-dimensional case the sign is fixed by the curve's parametrization.

The curvature of space curves is measured in its *Frenet frame*, which is spanned by three vectors at each curve point **p**.

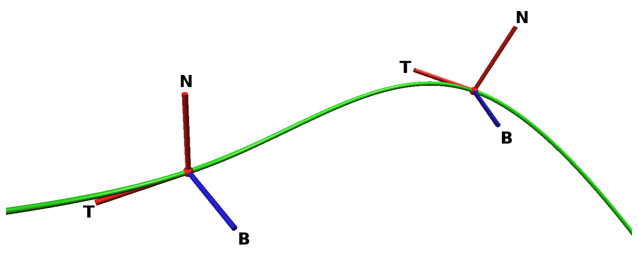


Figure 12: The Frenet frame at an arbitrary position of a curve is defined by the normal **N**, the tangent **T** and the binormal **B**. The curvature describes how fast **N** and **T** rotate around **B**.

- The unit tangent vector $\mathbf{T}(\mathbf{p})$
- The unit normal vector $\mathbf{N}(\mathbf{p})$
- Their cross-product, the so-called binormal vector $\mathbf{B}(\mathbf{p})$

The three vectors form an orthonormal basis. Similar to the planar case, curvature measures how fast the tangent vector rotates in the plane defined by the binormal vector. Figure 12 visualizes the Frenet frame of a curve.

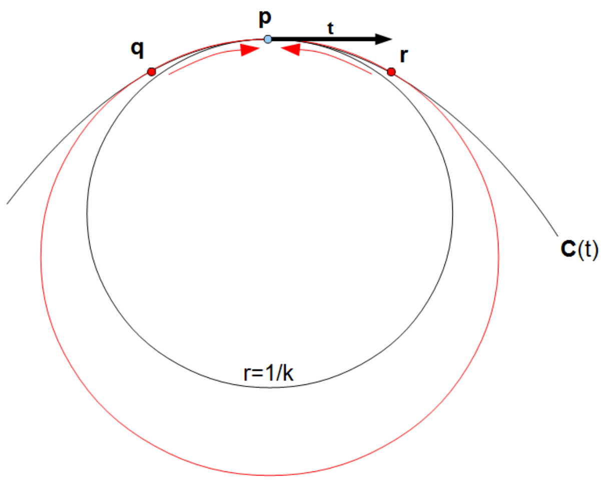


Figure 13: Constructing the circle of curvature: As **q** and **r** move closer to **p**, their circumscribed circle more and more resembles the circle of curvature.

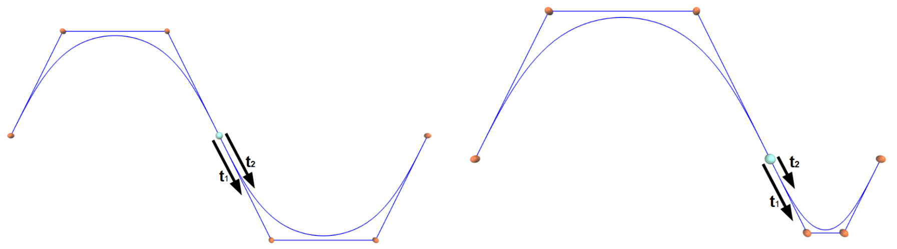
The *radius of curvature* is in all cases given by

$$r = \frac{1}{|\kappa|}$$

and defines the radius of a special circle, namely the *circle of curvature*, that is attached to the curve at the point **p** the curvature value is measured at. At this point the circle and curve share a common tangent. The circle of curvature can be visualized by choosing a point **p** and two neighbors **q** and **r** on a parametric curve. Three points uniquely define a circle, and when **q** and **r** move closer to **p**, in the limit the three points will define the circle of curvature. This discrete definition of the circle of curvature is illustrated in figure 13.

The radius of curvature, being just the inverse of the absolute curvature, also leads to a discrete definition of curvature (section 4.4.4).

2.5 CONTINUITY CLASSES



(a) Spline curves connected in a $\mathbf{C}^1$ manner. The tangents are equal in direction and magnitude.

(b) Spline curves connected in a $\mathbf{G}^1$ manner. The tangents only equal in direction, but the curves could be reparametrized to connect in a $\mathbf{C}^1$ manner.

Figure 14: Parametric and geometric continuity.

When describing a parametric curve or surface, one important aspect is its degree of continuity over the whole parameter space. The continuity property is also strongly connected to the parametric primitive's curvature quality.

For univariate functions the continuity class can be defined as follows.

Definition 9 (Continuity (differentiability) class)

A function $f : \mathbb{R} \to \mathbb{R}$ *is said to be of differentiability class* $\mathbb{C}^n$ *if all derivatives* $f', \ldots, f^n$ *exist and are continuous.*

This concept can be generalized for multivariate functions.

Definition 10 (Continuity (differentiability) class of multivariate functions)

A multivariate function $f : \mathbb{R}^n \to \mathbb{R}^m$ *with component functions* $f_1, \ldots, f_m$ *is said to be of continuity class* $\mathbb{C}^n$ *if all partial derivatives* $f_i', \ldots, f_i^n$ *exist and are continuous.*

For parametric curves and surface we distinguish two kinds of continuity. For ease of explanation only curves are treated.

Definition 11 (Parametric continuity)

A curve is said to be of continuity class $\mathbb{C}^n$ *if* $\frac{d^n s}{dt^n}$ *exists and is continuous throughout the curve.*

For an alternative definition, let $\mathbf{p}$ be an arbitrary curve point that splits the curve into two segments. Parametric continuity $\mathbf{C}^n$ states that the first n derivatives of both segments are equal in both direction and magnitude at the join $\mathbf{p}$. A curve is said to have $\mathbf{C}^n$ continuity if this property holds for every curve point $\mathbf{p}$.

Definition 12 (Geometric continuity)

A curve is said to be of continuity class $\mathbf{G}^n$ *if it can be reparametrized to have* $\mathbf{C}^n$ *continuity.*

This means that for every join $\mathbf{p}$ a local parametrization must exist which makes the split curve segments join in a $\mathbf{C}^n$ fashion at $\mathbf{p}$.

Parametric continuity means continuity also on the parametrization, while geometric continuity demands continuity just on the geometry itself. If we watch an object move, parametric continuity will assure that the object moves smoothly. Geometric continuity in contrast only assumes smoothness of the trace the object leaves behind, and assures that important geometric properties such as slope ($\mathbf{G}^1$) or curvature ($\mathbf{G}^2$) vary continuously.

The difference between parametric and geometric continuity is illustrated in figure 14.

High quality geometries used for industrial applications will at least be of class $\mathbf{G}^2$, in order to ensure curvature continuity.

2.6 B-SPLINES

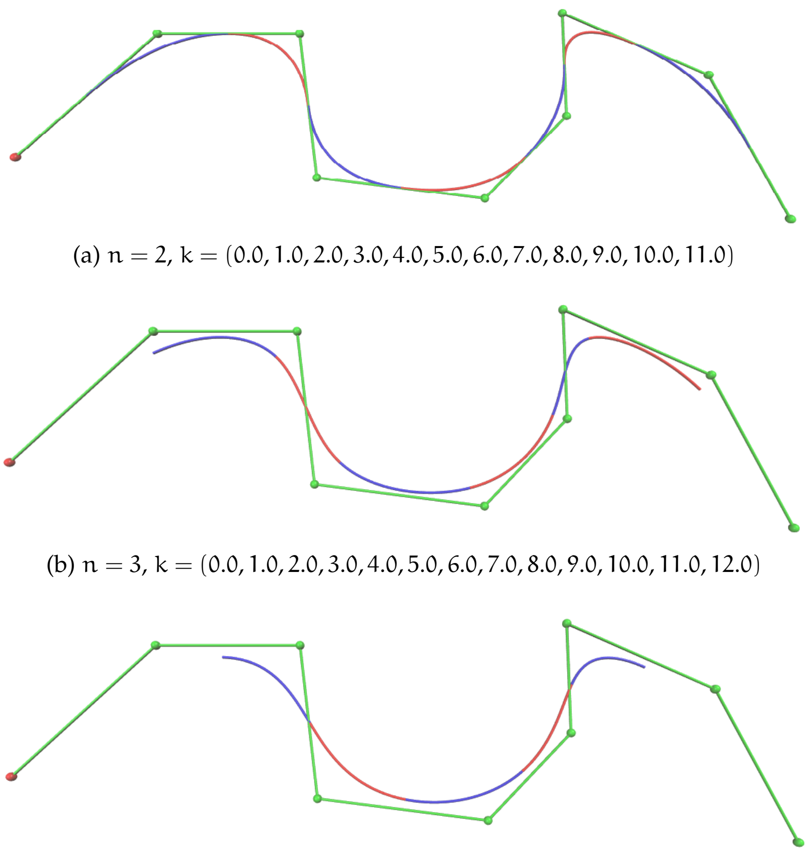
The parametric design curve we will use throughout this work is the *uniform cubic B-spline curve*. We will shortly review its most important properties, for a good basic introduction to B-spline curves we refer to [18] or [10].

2.6.1 *General B-spline curves*

B-spline curves are one of the most broadly used design curves in the field of CAGD. They are a generalization of Bézier curves and themselves generalized by the hugely popular NURBS. Usual drawbacks of Bézier curve design are:

- Moving one control point influences the whole curve, which makes the design process very painful (no local support).

- Adding one more control point also increases the degree of the curve by one, which results in higher complexity and computational costs.
- It is difficult to maintain the continuity of several consecutive Bézier curves when editing their control points.



(a) $n = 2$, $k = (0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0, 11.0)$

(b) $n = 3$, $k = (0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0, 11.0, 12.0)$

(c) $n = 4$, $k = (0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0, 11.0, 12.0, 13.0)$

Figure 15: B-spline curves with varying degree. The higher the degree n, the more the curve moves away from the control polygon.

B-Spline curves eliminate these drawbacks. They consist of multiple polynomial curve segments, which connect to each other with endpoint conditions depending on the curve's degree. Instead of increasing the total degree of the curve by one, adding an additional control point just results in one more curve segment of the same degree. The individual curve segments are thus much more decoupled as in the case of Bézier curves. The B-spline polynomials result from control points that are weighted by the so called B-spline basis functions.

Definition 13 (B-spline Basis Functions) *Let* $l = (t_0, t_1, \cdots, t_{k-1}, t_k)$ *be a non-decreasing sequence of numbers, the* ***knot vector****, and* ***n*** *the degree of the B-spline curve. Then the B-spline basis functions can be defined as*

$$N_i^0(t) = \mathbb{1}_{[t_i, t_{i+1}]}(t) \tag{1}$$

$$N_i^n(t) = \frac{t - t_i}{t_{i+n} - t_i} N_i^{n-1}(t) + \frac{t_{i+n+1} - t}{t_{i+n+1} - t_{i+1}} N_{i+1}^{n-1}(t) \tag{2}$$

with $\mathbb{1}_{[x,y]}$ *being*

$$\mathbb{1}_{[x,y]}(t) = \begin{cases} 1 & \textit{if } t \in [x, y] \\ 0 & \textit{else} \end{cases}$$

For equal knots we define $\frac{0}{0} = 0$.

The basis functions thus recursively blend together the parametric borders defined by the knot vector depending on the relative position of the parameter t.

The general B-spline curve is then defined as follows.

Definition 14 (B-spline curve) *Let* $l = (t_0, t_1, \cdots, t_{k-1}, t_k)$ *be a knot vector and* $\mathbf{P} = (\mathbf{p}_0, \cdots, \mathbf{p}_{k-n-1})$ *a set of control points called* ***control polygon****. Then a B-Spline curve of degree* n *is defined by*

$$c(t) = \sum_{0}^{k-n-1} \mathbf{p}_i N_i^n(t) \qquad t \in [t_n, t_{k-n}], \tag{3}$$

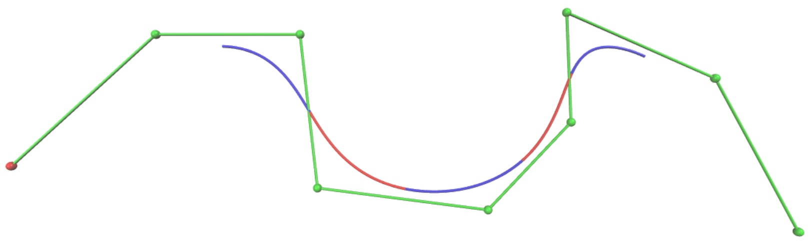
where n *is bounded by* $1 \leqslant n \leqslant k - 1$.

Figure 15 shows B-spline curves of various degrees.

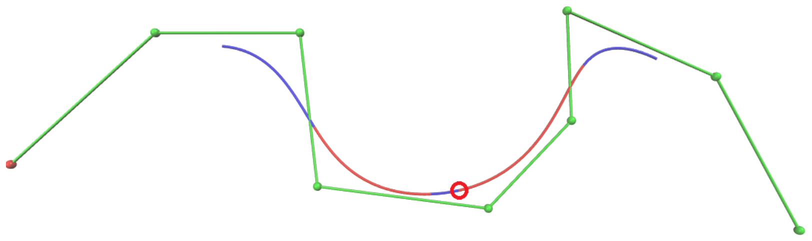
There are basically two ways to modify a B-spline curve. Adjusting the control points is the method of choice to change the curve's overall shape, while adjusting the knots is more of a fine-tuning operation, which influences the time spent in each segment (figure 16).

Some well-known and important properties of B-splines are stated below (see e.g. [10]).

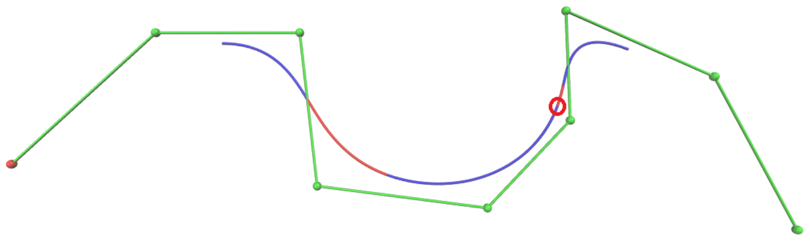
- **Local support**: $N_i^n(t)$ is zero outside the interval $[t_i, t_{i+n+1}]$. From this follows the important fact that only $n + 1$ successive control points contribute to a single segment or in other words, that there are $n + 1$ overlapping basis functions in each segment. In a cubic B-spline, each segment is influenced by four control points. Conversely, each control point influences four segments of the curve.



(a) $n = 4$, $k = (..., 6.0, 7.0, 8.0, ...)$



(b) $n = 4$, $k = (..., 6.0, 6.2, 8.0, ...)$



(c) $n = 4$, $k = (..., 6.0, 7.8, 8.0, ...)$

Figure 16: Manipulation of a knot. The time spent in a segment can be changed through the knots. Since this also affects the curves shape, the knots can be seen as tuning parameters.

- **Partition of unity**: The intervals $[t_n, t_{k-n}]$ are called *inner intervals*. For a parameter t on the inner intervals the following is always true:

$$\sum_{0}^{k-n-1} N_i^n(t) = 1 \qquad t \in [t_n, t_{k-n}].$$

- **Strong convex hull**: Each segment lies in the convex hull of the control points that contribute to it.

- **Variation diminishing**: No straight line intersects a B-spline curve in the plane more often than it intersects its control polygon. In the case of space curves this is true for planes. The B-spline curve thus always obtains less variation than its control polygon.

- **Continuity**: A B-spline curve is $\mathbf{C}^{n-1}$ continuous if all knots differ from each other. With m being the highest number of successive knots that coincide, the continuity reduces to $\mathbf{C}^{n-m}$. Such a knot vector is said to have a *multiplicity of m*. Since knots define segment borders, having a certain multiplicity at a knot will result in reducing the continuity at the respective border.

- The B-spline curve never varies more than its control polygon does.
- Given a degree n and j control points there have to exist $n+j+1$ *different* knots.

2.6.2 *Uniformity of B-spline knot vectors*

A B-spline with all knots spaced equally is said to be *uniform*. The basis functions of such a B-spline are overlapping copies of each other, shifted in parameter space.

Uniformity usually also brings some disadvantages.

- One will loose the possibility to alter the knot vector, which takes away a certain degree of freedom and locality.
- Many fitting algorithms rely on fine-tuning the knots, inserting new ones, removing others and so on.
- Covering the same parameter space between unequally spaced segment borders may lead to overshooting behavior, resulting in undesired bends and oscillations.

As for the last point, imagine a B-spline curve of degree $n = 3$. The parametric velocity needs to change very abruptly when for instance running from quite long segments into very short ones, abrupt changes not being possible because of the $\mathbb{C}^2$ continuity property. Usually this effect can be reduced by methods such as chord length parametrization on the knots, which is not possible in the uniform case.

Fixing the knot values also has its advantages.

- No strategy has to be implemented to find corresponding knot values when fitting the curve to some shape.
- For a user knot values are very abstract and unintuitive. Automatic adjustment of knots on the other hand will represent some inner magic that is hard to grasp.
- The curve can be expressed, parametrized and evaluated in a much faster and simpler way.

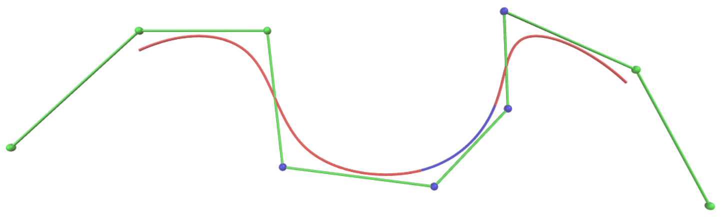


Figure 17: Uniform cubic B-spline curve. A single segment is defined by four control points (exemplarily colored in blue).

2.6.3 *Uniform cubic B-spline curves*

As the name suggests, the uniform cubic B-spline curve has a uniform knot vector and is of degree $n = 3$.

A good choice for a knot vector is $l = (t_0, t_1, \cdots, t_{k-1}, t_k) = (-3, -2, -1, 0, 1, 2, 3, 4)$. With $n = 3$ and $k = 7$ such a B-spline segment is formed by 4 successive control points and can be conveniently evaluated in the interval $[0, 1]$. Figure 17 shows a uniform cubic B-spline curve.

$$\mathbf{c}(t) = \sum_{i=0}^{3} \mathbf{p}_i N_i^3(t) \qquad t \in [0, 1].$$

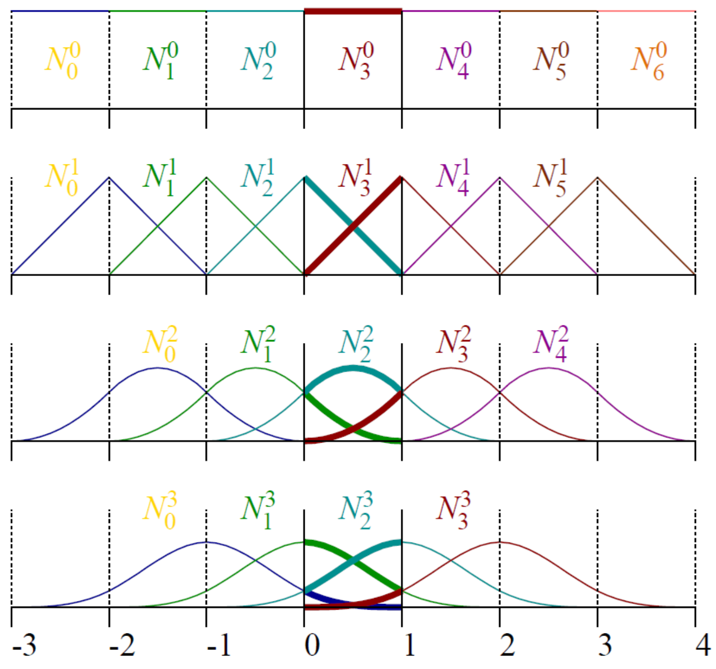


Figure 18: Recursive combination of basis functions at different stages of the uniform cubic case.

The cubic basis functions are shown in figure 18. As one can see the interval $[0, 1]$ is the only interval where all basis functions contribute. Further, moving a single control point will only influence two neighboring segments in each direction of the curve. The four cubic polynomials that form the basis functions are listed below.

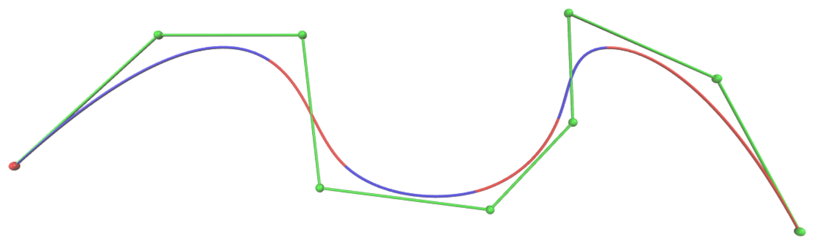
$$N_0^3(t) = \frac{1}{6}(-t^3 + 3t^2 - 3^t + 1)N_3^0$$
$$N_1^3(t) = \frac{1}{6}(3t^3 - 6t^2 + 4)N_3^0$$
$$N_2^3(t) = \frac{1}{6}(-3t^3 + 3t^2 + 3t + 1)N_3^0$$
$$N_3^3(t) = \frac{1}{6}t^3 N_3^0$$

They can be easily retrieved by using recursion 2.

Using these basis functions, a uniform cubic B-spline segment can be expressed in an efficient matrix form as

$$\mathbf{c}(t) = (\mathbf{p0}, \mathbf{p1}, \mathbf{p2}, \mathbf{p3})\,\frac{1}{6}\begin{pmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 0 & 4 \\ -3 & 3 & 3 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}\begin{pmatrix} t^3 \\ t^2 \\ t^1 \\ 1 \end{pmatrix} \qquad t \in [0, 1]. \quad (4)$$

2.6.4 *Clamping*


(a) Interpolating the endpoints using clamping. Since the curves degree is $n = 3$, clamping is achieved by having knot multiplicity 4 at both ends.

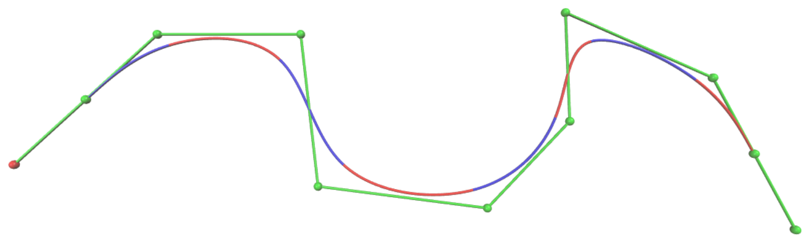

(b) Interpolating the endpoints by geometrical construction. The new endpoints should be seen as »virtual«, because they are only used to achieve the interpolation at their successor/predecessor.

Figure 19: B-spline endpoints.

When constructing a B-spline curve there are multiple ways to handle the endpoints. It is often desirable that the curve's endpoints coincide with the first and last control point, which is not the case by default. This can be achieved in two ways:

- **Knot multiplicity**: Having a knot multiplicity of $n+1$ at the begin and end of the knot vector.
- **Constructional solution**: Adding two additional virtual control points in such a way, that both the first and last chords of the control polygon are prolonged by a new chord of the same length. Virtual means that the additional control points are just used for curve construction, they are not really part of the original control polygon.

Since knot multiplicity is incompatible with uniform B-splines, the constructional approach will be the tool of choice for them when clamping is needed. Both methods are visualized in figure 19.

2.6.5 *Bicubic uniform B-spline surface*

Although the focus in this text lies on B-spline curves, the connection between B-spline curves and surfaces is a quite interesting one. We will quickly introduce the *bicubic uniform B-spline surface*, in order to show how easily B-spline curves can be extended to the surface case. This emphasizes the fact that findings for the curve case can also be very relevant for the surface case, as generalization is usually not too painful.

B-spline curves can be extended to B-spline surfaces by using a two dimensional parameter space (u, v) and combining two basis functions $M_i^m(u)$ and $N_j^n(v)$. The combined basis functions are required to form partitions of unity and to yield only non-negative values.

$$L_{ij}(u,v) = M_i^m(u) \cdot N_j^n(v)$$

Such surfaces are called *tensor product surfaces*.

Definition 15 (Tensor product surface)

$$\mathbf{s}(u,v) = \sum_i \sum_j \mathbf{p}_{ij} \cdot M_i^m(u) \cdot N_j^n(v) = \sum_i \sum_j \mathbf{p}_{ij} \cdot L_{ij}(u,v) \tag{5}$$

Figure 20 illustrates how easily we can construct a B-spline surface by interconnecting multiple control polygons to form a control mesh. The surface will inherit the curve's continuity properties, e.g. in the case of cubic B-spline curves this means that the resulting bicubic surface will also be $\mathbf{C}^2$ continuous.

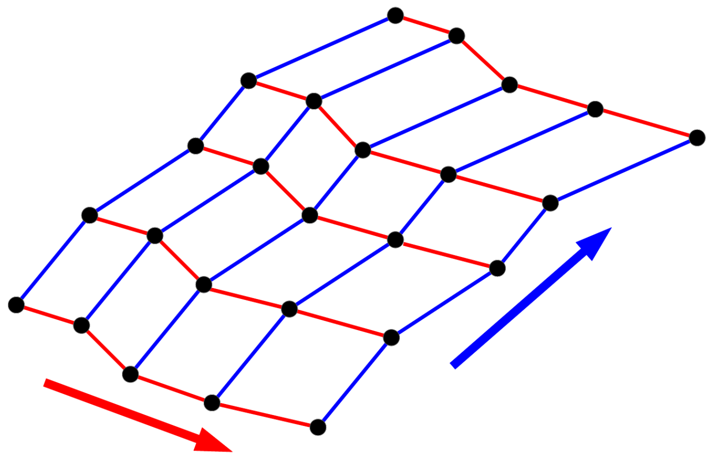


Figure 20: Construction of a B-spline control mesh by interconnection of multiple control polygons.

The property of simple extension from the curve to the surface case is extremely useful, uniting curves and surfaces into a common framework with common properties. An especially important implication is that a set of carefully chosen high quality B-spline curves can be used to span a high quality B-spline surface.

2.6.6 *Subdivision and limit points*

In this section we will shortly discuss the Catmull/Clark subdivision scheme. Although originally introduced for B-spline surfaces it can easily be applied to B-spline curves. The concept of limit points our algorithm is based on can be explained via the subdivision process in a very simple way. Subdivision also enables us to compute the limit points efficiently from control points. For a deeper explanation we refer to [18].

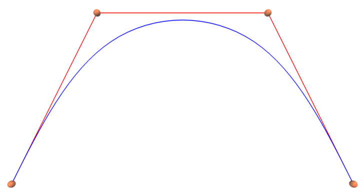
The Catmull/Clark subdivision scheme is a generalization of the uniform cubic B-spline and can be evaluated in a very fast and stable way for both curves and surfaces. We will focus on the curve case. A new control polygon is obtained in each step of subdivision, which yields the same curve but lies closer to it. In the limit the control polygon converges to the curve itself. Two rules are applied to the control polygon in order to generate the new control vertices.

Definition 16 (Edge point rule) *Edge points are generated by two successive control points* $(\mathbf{p}_1, \mathbf{p}_2)$ *that form an edge.*

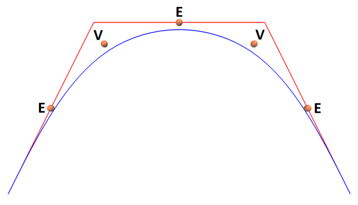
$$\mathbf{p}_E = \frac{1}{2}(\mathbf{p}_1 + \mathbf{p}_2) \tag{6}$$

Definition 17 (Vertex point rule) *Vertex points are generated by three successive control points* $(\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3)$ *forming a corner.*

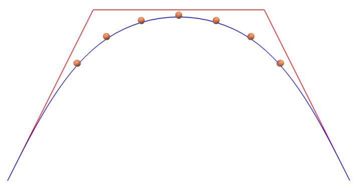
$$\mathbf{p}_V = \frac{1}{8}(\mathbf{p}_1 + 6\mathbf{p}_2 + \mathbf{p}_3) \tag{7}$$



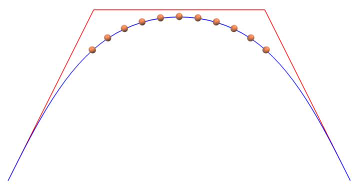
(a) A control polygon.



(b) One step of subdivision. The edge rule (**E**) is applied 3 times , the vertex rule (**V**) 2 times.



(c) Two steps.



(d) Three steps.

Figure 21: Subdivision of a uniform cubic B-spline segment. One can see how fast the subdivided points converge to the limit curve.

The number of control vertices thus approximately doubles in each step of subdivision. Figure 21 illustrates this process.

One important question is: To which curve points do the original control points converge? This can be answered by applying the vertex rule multiple times on a control point and its neighbors, in the limit resulting in the following simple rule.

Definition 18 (Projection to limit position)

$$\mathbf{p}_{i_\infty} = \frac{1}{6} \cdot (\mathbf{p}_{i-1} + 4\mathbf{p}_i + \mathbf{p}_{i+1}) \tag{8}$$

This is an especially useful property of subdivision, as it is now possible to refine curve parts individually, and then project the control points onto the limit curve to obtain shapes at varying levels of detail. Figure 22 shows the projection of control vertices onto the limit curve.

The projection of a control point onto the limit curve is named *limit point* or *Greville point*. It is the boundary of a B-Spline segment corresponding to an inner knot, which can also be nicely seen in figure 22.

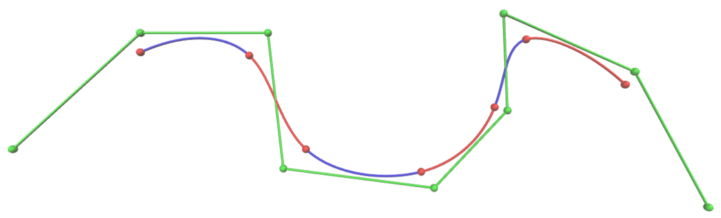


Figure 22: Limit positions of the inner control vertices of a uniform cubic B-spline curve. As we can see, the limit positions represent the borders between the alternately colored B-spline segments.

A B-spline curve can be defined uniquely by either a set of control points or a set of limit points, and since they have such a unique correspondence to each other, we can say that control and limit points represent a different basis of the B-spline.

When comparing the control point basis and limit point basis of a B-spline, a big difference is the area of influence of a single point of the basis. While moving a control point only affects the curve locally (local influence), moving a limit point affects the whole curve (global influence). The local influence of the control point can be explained by the local support of the B-spline, the global influence of the limit point by its dependence on multiple control points, which causes a propagation over all other limits. However, the influence of a limit point dampens rather quickly with distance.

2.6.7 *Obtaining control points from limit points*

As discussed in the last section, it is quite easy to obtain the limit position of a control point under subdivision. What is missing is the reverse direction - to retrieve the control points from a set of given limit points.

The desired method solves the following problem: Given n limit points $\mathbf{L} = (\mathbf{l}_0, \cdots, \mathbf{l}_{n-1})$, find a set of n control points $\mathbf{P} = (\mathbf{p}_0, \cdots, \mathbf{p}_{n-1})$, such that $\mathbf{p}_{i_\infty} = \mathbf{l}_i$.
The B-spline curve defined by the control points $\mathbf{P}$ has to interpolate each given limit point by definition.

This can either be done by solving a linear system of equations (including endpoint conditions, see for example [10]) or by using a simple iterative method by Lin et al. that can be found in [25]. Because of its ease of implementation the second method was chosen for our implementation (the other method supposedly being a little bit faster). It sums up as follows.

Algorithm 1 (Obtain control points from limit points) *Starting with a set of limit points* $\mathbf{L}$*:*

```
1 Set the initial control points to $\mathbf{P} = \mathbf{L}$.
2 Calculate the limit positions $\mathbf{L}'$ from $\mathbf{P}$ using 8.
3 Calculate the offset vectors to the true limit points as $\mathbf{d}_i = \mathbf{L}_i - \mathbf{L}'_i$.
4 Correct the control points: $\mathbf{p}_i = \mathbf{p}_i + \mathbf{d}_i$.
5 Repeat from 2 unless the error is smaller than some threshold $\epsilon$.
```

The algorithm converges very fast and reliably ([25] shows that in fact it always converges). Also it works for both curves in 2D and 3D, for surfaces, and for arbitrary subdivision schemes.

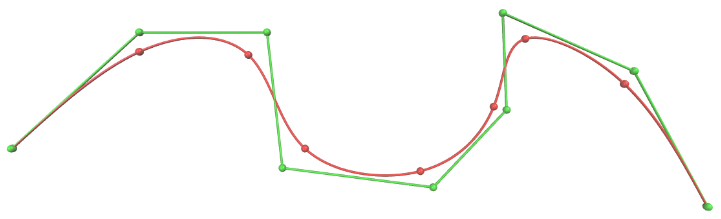
If endpoint interpolation is desired, the first and last control point can be excluded from the iterative process. Regardless of the choice on endpoint positions, the other points will always adapt to form a valid control polygon. Another option can be to fixate *two* points at each end of the control polygon to form tangential conditions (see [10]).

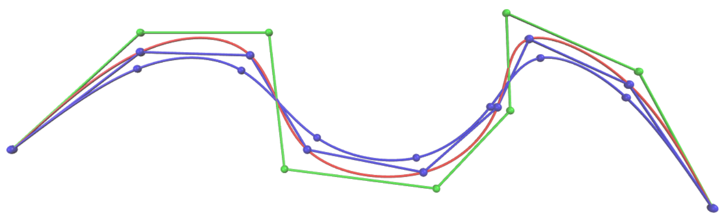
Figure 23 shows results for the algorithm at different iterations.

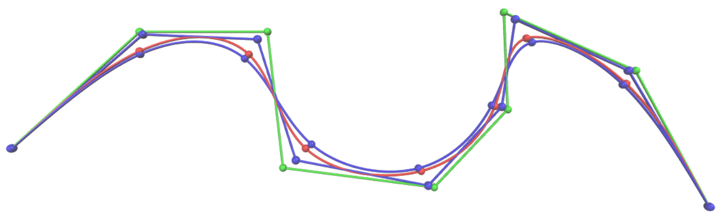
The robust and efficient computation of control points from a set of limit points allows us to choose the limit positions of a spline first and compute the corresponding control points afterwards. A major advantage of limit point placement is that limit points are a very »direct« means of interaction, since they are interpolated by be the generated shape. It is more intuitive for a user not having to deal with an abstract control polygon, which for higher B-spline degrees will move more and more away from the shape it generates. A disadvantage of limit point placement is of course the global influence of limit points we discussed earlier.

We can even go a step further and choose limit positions directly on the data to be interpolated by our spline. The input data could e.g. be a parametric spline or a discrete representation such as a polygonal curve. In this case a limit point can be parametrized by its parametric position along the input data, which reduces the search-space in the 2D curve case from $\mathbb{R}^2$ to a certain parameter range in $\mathbb{R}$. This is very convenient for data fitting and optimization, but the restricted domain also benefits a user manually editing a B-spline. The benefits are even more evident in the 3D case. We will make use of this method later on.

An inherent drawback of limit point placement on the input shape is that interpolation of the shape might not even be desired in the context of fitting. A much better fit maybe could be obtained by choosing interpolation positions away from the shape. This is true in terms of


(a) The given B-spline limit points (red). The original curve (also in red) and control points (green) are additionally shown as ground truth.


(b) In the first step the given limit points are directly used as a control polygon, resulting in a first estimate (blue) with a rather high error.


(c) In the second step this first guess is corrected by the limit position error offsets.

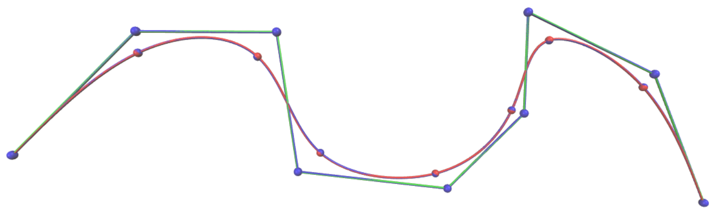

(d) After 20 iterations the difference is negligible.

Figure 23: Reconstructing the control points from given limit points using an iterative approach.

distance (e.g. for a least squares fit), but also in terms of curvature quality, since the interpolated shape might itself contain flaws (e.g. unwanted waves, bends, discretization artifacts). We will try to mitigate this fact by computing a more fair version of the input data first.

3 RELATED WORK

B-spline fitting and interpolation has been a very well-known and thoroughly investigated field for a long time now. Basic ideas in B-spline fitting are of a very general nature. The classical approach to fit a B-spline curve to given data points is to obtain a least squares solution in terms of the distance error between the data points and the curve. The following is described in more detail for instance in [10].

Imagine P+1 data points $\mathbf{p} = (\mathbf{p}_0, \dots, \mathbf{p}_P)$ that should be approximated by a B-spline curve of degree N. The degree N of the curve is most often declared beforehand and very crucial for the result. On one hand a curve of too low degree may not model the data well enough, on the other hand a curve of too high degree may be harder to control and result in overfitting (e.g. capture noise on the data, obtain unwanted oscillations). Another important choice to make is the number of knots used, which in combination with the degree gives the number of control points C of the spline curve.

We want the distance of the B-spline curve $\mathbf{c}(t)$ to each data point $\mathbf{p}_i$ to be as small as possible. For this, a B-spline curve parameter t_i has to be assigned to each data point $\mathbf{p}_i$. Let $\mathbf{t} = [t_0, \cdots, t_P]$ denote the vector of all of these assigned parameter values. Now, assumed that we were able to assign a B-spline parameter value to each data point, we want to minimize the functional

$$e_{sqr} = \sum_{i=0}^{P} ||\mathbf{p}_i - \mathbf{g}(t_i)||^2. \tag{9}$$

This is a very general expression, which could also be used for e.g. Bézier curve fitting. It is independent of the notion of a knot vector and applies to nearly any parametric curve fitting task. The functional can be further extended in various ways in order to obtain more satisfying results, as will be shown below.

Such a functional is traditionally minimized by solving a set of linear equations. Much of this is discussed in de Boor's classic work [6], and there is also a good introduction given in [10]. When focusing on B-splines, equation 9 can be rewritten as

$$e_{sqr} = \sum_{i=0}^{P} \left|\left| \mathbf{p}_i - \sum_{j=0}^{C} \mathbf{c}_j N_j^n(t_i) \right|\right|^2. \tag{10}$$

The key here is to find good choices for the parameter values t_i and the knot vector $\mathbf{k}$, in order to leave the control points to be the only remaining unknowns. This is not an easy task and the choice will greatly influence the result of the approximation. By setting the partial derivatives for each control point $\mathbf{c}_j$ to zero, we can obtain a set of so-called normal equations.

$$\sum_{j=0}^{C} \mathbf{c}_j \sum_{i=0}^{P} N_j^n(t_i) N_k^n(t_i) = \sum_{i=0}^{P} \mathbf{p}_i N_k^n(t_i) \quad , \quad k = 0, \cdots, C \tag{11}$$

The symmetric coefficient matrix $\mathbf{M}$ of the $C+1$ linear equations is singular only if there exists a knot interval $[k_{j-1}, k_{j+N}]$ that contains none of the assigned parameter values t_i (*Schoenberg-Whitney theorem*). It has the form

$$\mathbf{M}_{j,k} = \sum_{i=0}^{P} N_j^n(t_i) N_k^n(t_i). \tag{12}$$

A stable evaluation of $\mathbf{M}$ will not automatically lead to good results. This is in the very nature of the pure least squares approximation, but the optimization can be refined in many ways in order to influence the final result. We often do not want the curve to be nearest to all data points at all cost, and may thus decide to accept a greater distance error if we can trade it for more meaningful shapes. This may for instance be needed to enforce a smooth and oscillation-free curve shape, to handle stronger noise, or to smoothly fill gaps in the data. Some approaches to reach these goals are listed below.

ENERGY TERMS / VARIATIONAL APPROACHES There are various *fairness* or *energy terms* that can be added to equation 9 to enforce smoother variation of the curve. Often such an energy term is a quadratic function involving the curve's first or second derivative, or of a mix of both. A classic choice is e.g.

$$F_{en} = \int ||\mathbf{g}''(t)||^2 dt. \tag{13}$$

The functional to be minimized then becomes

$$e_{sqr} = \sum_{i=0}^{P} ||\mathbf{p}_i - \mathbf{g}(t_i)||^2 + \lambda \cdot F_{en}. \tag{14}$$

Such variational approaches are widely used in curve and surface fitting, so there exists much literature on them. Energy terms are discussed e.g. in [16], [34], [42] or [43].

LINEAR SMOOTHING TERMS Additional smoothing terms can be incorporated into the linear system 11, which typically form linear constraints on the curves shape, for instance on the angles between successive control polygon chords. Another often used method is for instance the so-called *cage regularization*, where we want a control point to lie in the mid of its neighboring control positions. This can be realized as weights $[1, -2, 1]$ on each control point triplet in the linear system's matrix. Refer to [10] for a broader explanation of such methods.

GEOMETRIC CONSTRAINTS Another way to influence the fitting result is to add additional geometric constraints to equation 9. Such constraints could for instance be sets of first or higher order derivatives, specified at the data points $\mathbf{p}$. The curve will then not only have to adapt to the given positions, but e.g. also to given tangent directions. Constraints on the endpoints are another example. The constraints are usually incorporated into 9 by making use of *Lagrange multipliers*, as e.g. explained in [32]. For the linear system approach only such constraints can be incorporated, that can be formed into linear terms.

A crucial task that remains is the assignment of parameter values to the data points, and the decision on a knot vector. There are several techniques for assigning parameter values to the data points. A uniform parameterization is usually the most simplistic but also worst way to do this, since it completely ignores the geometry of the data points, resulting in overshooting behavior and oscillations. A better way to do it is *chord length parameterization*, which relates the distance in parameter space to the distance of the data points. For data with sharp bends, even better results can be achieved by *centripetal parameterization* ([23]), which smoothes out the centripetal forces working on an object that moves along the spline curve. Whatever the chosen method, it is highly unlikely that $\mathbf{p}_i$ will exactly equal $\mathbf{g}(t_i)$. In this case a method called *parameter correction* (or *intrinsic parameterization*, Hoschek [20]) can be used, which corrects the parameters t_i in a way that the new positions $\mathbf{g}(t_i')$ will lie closer to the respective $\mathbf{p}_i$. The curve is then recalculated, and the process is iterated a few times. Many of these reparameterization schemes operate locally, a more global strategy can for instance be found in [39]. It can be said though that there is no optimal strategy for this task, so parameter assignment remains an uncertainty factor.

The other problem is to choose a good knot vector and, to complicate the matter even more, to decide on the number of knots needed. There are multiple methods for knot selection, two methods that achieve a positive definite and well-conditioned matrix $\mathbf{M}$ (with at least one t_i in every knot span) are presented in [32]. For this we assume the outer knots to be of the form

$$k_0 = \cdots = k_N = 0 \qquad k_{K-N} = \cdots = k_K = 1. \tag{15}$$

This will assure interpolation of the first and last data point. The first method is called *knot averaging* and can be used if $C = P$ (which actually equals an interpolation task).

$$k_{j+N} = \frac{1}{N} \sum_{i=j}^{j+N-1} t_i \qquad j = 1, \cdots, P - N$$

The parameter values distributed over the data points are just locally averaged. The other method is called *knot placement* and can be used if $P > C$ (most often the case).

$$\begin{aligned} d &= \frac{P+1}{C-N+1} \quad , \quad i = \mathit{int}(j \cdot d) \quad , \quad \alpha = j \cdot d - i \\ k_{j+N} &= (1-\alpha) \cdot t_{i-1} + \alpha \cdot t_i \qquad j = 1, \cdots, P - N \end{aligned}$$

Both methods reflect the distribution of parameter values, making them a much better choice than uniformly distributed knots. In the case of uniform B-splines the knot values are already fixed though, leaving only the total number of knots (or equally: the number of control points) to be chosen.

Summarizing, the classical approach to B-spline fitting includes the following steps.

1 Distribution of parameter values over the given data points.

2 Calculation of the knot vector.

3 Solving a system of linear equations, optionally by incorporating additional constraints or smoothing terms.

4 Using parameter correction in order to obtain better (nearer) curve positions.

5 Repeat from 2 if needed.

There are some approaches that deal with joint optimization of parameter values, knots and control points. Refer to e.g. [1], [13] or [37].

One approach to handle this complex problem is the technique of *knot removal*. For this, the data points are initially interpolated by the B-spline, each assigned parameter value representing a knot. Then knots are gradually removed, depending on their importance for the approximation, until a satisfying result has been produced. The approximation goal can be defined by a user-given error bound, the least squares error described by 9 being initially zero and growing as knots are removed. See e.g. [30], [9], [27].

Reversing the problem, there exist a lot of solutions that adaptively add knots based on heuristic rules stemming from the curve's local properties. This includes both adding them iteratively based on some criteria, or estimating the number of knots and distributing them afterward. Razdan ([36]) uses circular arcs and curvature distribution to place the knots. The total number of knots is thereby estimated by dividing the curve into circular arc segments. Similar heuristics are used in [24] or [31] (*dominant points*).

Methods based on modification of the knot vector are unfortunately not suitable in the context of this work, since the framework the presented approach is built upon relies on uniform B-splines.

Some other contributions make use of intermediate curve representations, which are finally converted into B-splines, such as polynomials ([30]) or Bézier-splines ([7]).

There also exist popular iterative methods which rely on *footpoint computation*. The knots remain fixed in all of these methods, which simplifies the task and makes them perfectly suitable for fitting uniform B-splines. The footpoint of a certain data point is its closest point on the current approximating design curve due to some distance metric. Calculating the footpoints can be formulated as a parameter correction task as follows.

In each iteration new control points are calculated by minimizing the error

$$f = \sum_i err_i^2 + \lambda \cdot f_s, \tag{16}$$

where err_i is an error metric describing the distance of the i^{th} data point to the design curve, and f_s an energy term for fairing of the result. There are three well-known optimization methods for this problem, each having its own error metric.

PDM The *point distance minimization* incorporates just the distance to the footpoints into the error metric.

$$err_i = ||\mathbf{g}(\mathbf{c}, t_i) - \mathbf{p}_i||^2$$

This is of course a bad approximation to the true err_i^2, but PDM is quite simple and can be used with unorganized data points too. It converges slowly though, being a variant of steepest descent ([43]). There are many contributions to curve fitting based on or alike PDM, such as [14], [20], [33] or [38].

TDM *Tangent distance minimization* additionally makes use of the normal vector information $\mathbf{n}_i$ at each data point.

$$err_i = \left[(\mathbf{g}(\mathbf{c}, t_i) - \mathbf{p}_i)^T \cdot \mathbf{n}_i \right]^2$$

Here the distance between a data point and an assigned tangent line is measured, which is a good approximation in low curvature regions, but yields unstable behavior in high curvature ones. Wang et al. state in [43] that this equals a variant of Gauss-Newton minimization without step control. TDM is e.g. used to fit B-splines in [5].

SDM *Squared distance minimization* has been introduced by Wang et al. in [43]. SDM also incorporates curvature information into the distance metric.

$$err_i = \begin{cases} \frac{d}{d-r} \left[(\mathbf{g}(\mathbf{c}, t_i) - \mathbf{p}_i)^T \cdot \mathbf{t}_i \right]^2 + \left[(\mathbf{g}(\mathbf{c}, t_i) - \mathbf{p}_i)^T \cdot \mathbf{n}_i \right]^2 & \text{if } d < 0 \\ \left[(\mathbf{g}(\mathbf{c}, t_i) - \mathbf{p}_i)^T \cdot \mathbf{n}_i \right]^2 & \text{if } 0 \leqslant d < r \end{cases}$$

Here d denotes the signed distance between the the data point $\mathbf{p}_i$ and the footpoint $\mathbf{g}(\mathbf{c}, t_i)$ on the fitted curve, and r the radius of curvature at the footpoint. The distinction made via the sign assures a positive semi-definite error metric, which is a second order approximation to the squared distance function. Wang et al. show that SDM actually equals a Newton optimization scheme, and that in terms of convergence it is both more stable and faster than the other methods. A more detailed explanation of SGM is given in section 4.11.

In all these methods, footpoint and control point calculation are separate stages that are alternated. Zheng et al. ([47]) deal with the joint optimization of footpoints (described by their parameter values t_i) and control points, which promises to be even faster than SDM.

It is possible to implement an adaptive scheme based on such iterative algorithms, by inserting new control points where they are most badly needed, followed by another optimization phase ([44]).

Iterative methods based on footpoints are very effective, but results strongly depend on the initial control point distribution provided. With bad initialization, the optimization is likely to end up in a local minimum. We can imagine the initial B-spline curve to »snap« to the data points.

In the last years many new approaches to the optimization problem of B-spline fitting emerged, stemming from such fields as computational biology (genetic algorithms, e.g. [4] or [45]), statistics (Gaussian mixture models, [46]) or optimal control ([13]). These methods yield promising results, but they also have disadvantages. They tend to behave a little like closed boxes, which makes it hard to comprehend the results and to influence them accordingly from the outside. However, controllability and comprehensibility are very important aspects of high quality modeling. This is one major reason why usually simpler (lower degree) parametric polynomials are chosen for modeling and data fitting.

Further, these techniques are very dependent on the formulation and parametrization of the problem - e.g. as a cost or reward function - so the problem of asking the right question remains. Methods based on machine learning seem especially promising, but often need a lot of training data to capture a problem adequately, which in practice can be a problem.

Although there exists a broad range of literature, the suggested solutions mainly concentrate on minimization of the distance error. What is completely amiss is a special focus on curvature quality and accurate reproduction of the (intended) input curvature, or at least an evaluation of results in terms of these aspects. Even if curvature values are incorporated into the approach, they are used mainly as a tool to again reach a certain distance error or assure better convergence (e.g. in SDM) - as a means to an end. Most interestingly an explanation for how curvature values are retrieved from the discrete data is rarely given, an aspect which in our opinion is essential for accurate spline fitting.

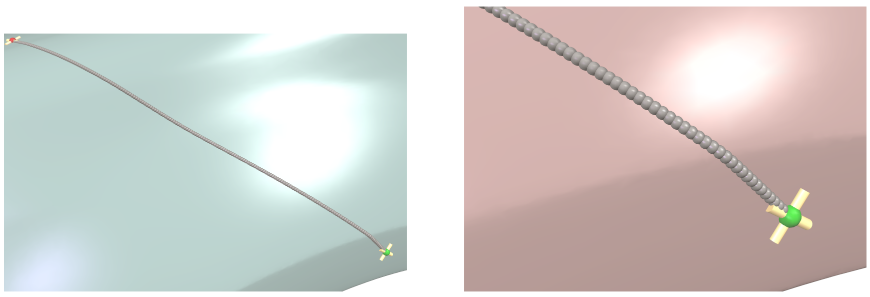
Thus, techniques focusing on curvature quality are on one hand badly underrepresented in literature, but on the other hand very sought-after in high quality CAGD. We will try to fill this gap with the presented work, and see it as an additional contribution to spawn a discussion about this extremely important topic.

4

UNIFORM CUBIC B-SPLINE FITTING IN A CLASS A MODELING ENVIRONMENT

4.1 PROBLEM DEFINITION

This work builds upon the *SurfaceReconstruction* toolkit, a collaborative effort between Volkswagen AG and the Institute of Computer Graphics and Knowledge Visualization at Graz University of Technology. The software package provides surface engineers with convenient modeling tools and assists them with the task of converting 3D scans (triangle meshes) to uniform cubic B-spline patches, a time-consuming process which for critical car parts is still carried out manually by experienced staff. The goal of the software is the partial - and ultimately full - automation of the needed workflow.



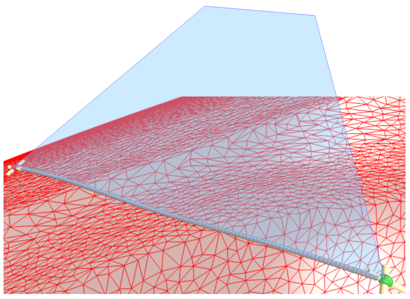
(a) Pearl chains are discrete curves sampled along a triangle mesh. Manipulators (here in red and green) can be dragged across the mesh in order to span the chain.

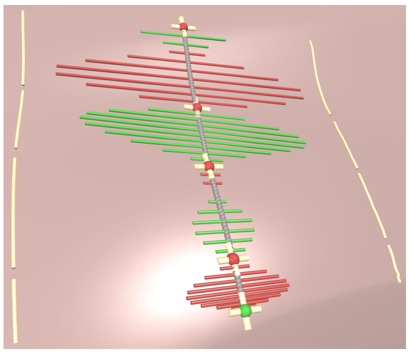
(b) Detail: The small gray pearls represent the mesh samples the chain consists of.

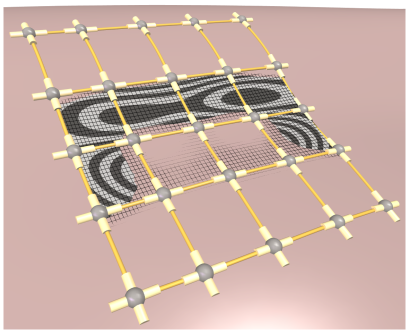
Figure 24: *SurfaceReconstruction* toolkit: Pearl chains

When editing a project manually, the user interacts through so-called *pearl chains* that are placed onto a 3D triangle mesh obtained from a scan. A chain consists of a path sampled along the mesh (visualized as small grey pearls), and draggable manipulators at its end points and optionally at discrete positions in-between (visualized by bigger colored pearls). Since they consist only of mesh samples, the chains cling to the model and may be dragged along it using the manipulators. Such a pearl chain is visualized in figure 24.

The path that is spanned between the manipulators of a pearl chain can be seen as a discrete polygonal curve. There are of course multiple ways to connect two positions on a mesh using a path along its surface. The software provides two modi operandi here: To follow the


(a) A pearl chain generated via a planar cut through the manipulator positions, resulting in a discrete planar 3D curve.


(b) Each chain represents a uniform cubic B-spline curve via the limits that are placed on it, here a total of five. The distance error of the constructed spline curve to the chain samples is displayed as bars, alternating their color with each segment (red and green). The error is zero at the limit positions. A predefined error bound is visualized as reference (shown to the left and right of the curve).


(c) Several pearl chains interconnected to a limit point mesh, the gray intersections representing the limit points. The uniform cubic B-spline patch that is defined this way is shown with its reflection lines (being partially occluded by the mesh).

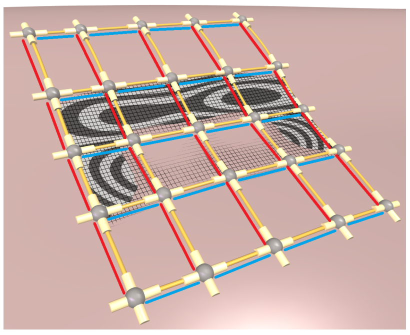

(d) The individual chains (B-spline curves) that contribute limits to the limit point mesh are shown in red in one grid direction and in blue in the other one. These individual curves span the uniform bicubic B-spline patch in the sense of a tensor product surface.

Figure 25: *SurfaceReconstruction* toolkit: Forming B-spline curves and patches using pearl chains

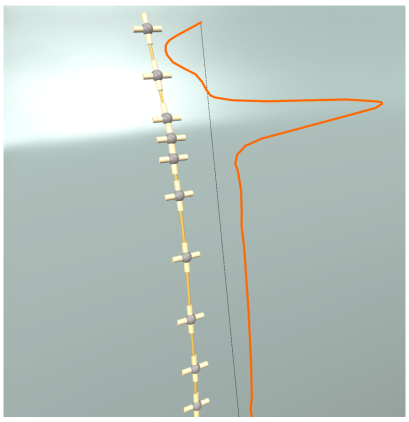
flow of curvature on the surface, resulting in a 3D space curve, or to make a planar cut through the two positions, resulting in a planar 3D curve (see figure 25a). Since our algorithm operates on planar curves, we make use of the second approach.

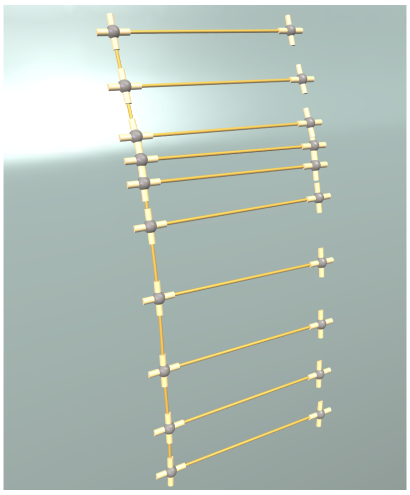
As mentioned before, we can interpret the small pearls as samples of a polygonal curve. Let us now interpret the bigger manipulator pearls as the limit positions for a uniform cubic B-spline curve. We can easily obtain this B-spline curve's control polygon and generate the corresponding polynomial curve, which interpolates the mesh at the limit positions. Subsequently the spatial distance of the spline curve to the discrete surface samples of the generating pearl chain can be evaluated. This is demonstrated in figure 25b. We use a discrete Hausdorff distance to measure the distance error.

Furthermore, multiple pearl chains may be interconnected to form a grid, the intersection points spawning new limit positions. By doing so we can form a limit point mesh for a Catmull/Clark subdivision patch (a uniform bicubic B-spline patch), which also interpolates the mesh at the user-defined limit positions. Figures 25cd show how several chains can be combined to form a limit point mesh.

From a user perspective choosing points of interpolation on the surface is much more intuitive than editing an abstract control polygon in full three dimensions. On one hand this considerably limits the search space for editing. On the other hand limit points in comparison represent more direct means of manipulation, as the control polygon usually lies away from its generated curve and thus from the modeled surface. In this setting the distance of the control points to the surface depends on both the B-spline degree and the variation of the surface, a planar surface e.g. always forcing the control points onto it.

Summing up, the central paradigm of the *SurfaceReconstruction* toolkit is the construction and interconnection of so-called pearl chains on a surface, in order to form a limit point mesh for a uniform cubic B-spline patch, which interpolates the surface at the chosen limit positions. In a nutshell: Uniform cubic B-spline surface fitting via interpolation point placement on the discrete mesh to be modeled.


(a) An initial chain with adapted limit positions. The corresponding curvature profile is shown as an overlay in orange. As we can see more points are spent in high curvature regions.


(b) The chain is prolonged into perpendicular direction at the limits. The assumption that the covered region is homogeneous enough must hold.

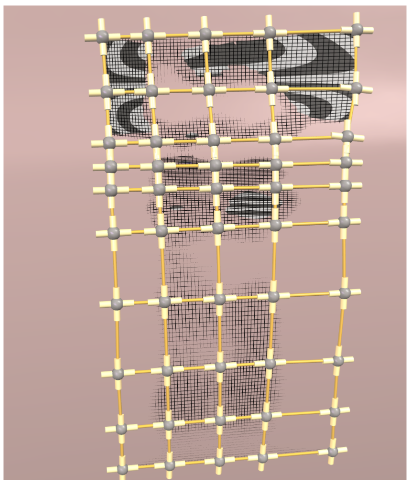

(c) More vertical chains are added to obtain an initial patch.

Figure 26: Modeling in homogeneous surface regions extending from an initial chain.

Let us now make use of the concept of pearl chains, and assume that the user spans a pearl chain across the mesh. By automatically finding a suitable distribution of limit positions along this chain, a B-spline curve fitting algorithm can assist the user in multiple ways.

- It helps the user to obtain some intuition for the geometry of the surface region that is currently modeled.
- It generates an initial solution for further manual refinement.
- It reduces the time spent on the modification of a single chain.

- Assumed that the surface region is homogeneous enough, it can even provide an initial solution for a B-spline patch, as shown in figure 26.

In order to obtain a basic understanding for the problem, we will focus on 2D curves which are obtained via planar cuts through the endpoints of the provided pearl chain. This way we will not have to deal with torsion. However, we assume that when having solved the problem in two dimensions, we can always build upon our results and deal with torsion later. We further assume that any progress for the curve fitting case can be of great use for the surface case too, since B-spline curves and surfaces are so closely tied.

The task to be achieved by the proposed algorithm is stated in a very general way as follows.

Problem formulation: *Given* N *ordered 2D curve samples* $\mathbf{p} = [\mathbf{p}_0, \cdots, \mathbf{p}_{N-1}]$, *a set of limit positions for a uniform cubic B-spline curve shall be retrieved from the discrete curve such that...*

- *The distance error of the fit is smaller than a specified maximum error bound.*
- *As few as possible limit points are used.*
- *The spacings between limit points vary regularly.*
- *Similar curves (stemming from similar mesh regions) yield similar limit point distributions. The algorithm is predictable, the result repeatable.*
- *The curvature of the produced spline is faithful to the curvature of the discrete input data.*

The first condition meets the need of industrial applications, where exact error bounds are very common. Furthermore the number of limit points spent will depend on this quantity.

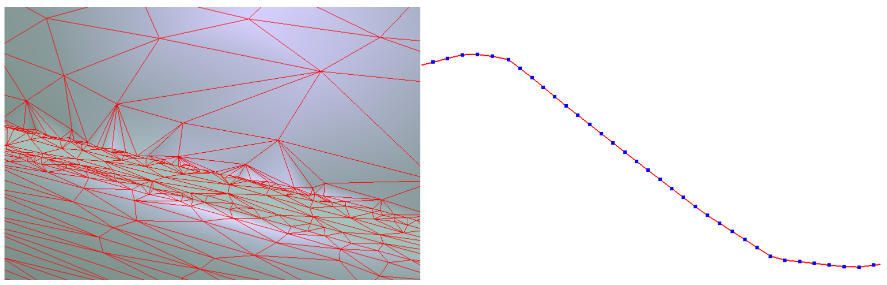
We want the number of limit points to be as low as possible, since many limit positions are harder to control and consolidate. A high number of limit points implies that many control points need to be moved in order to model a certain change of shape later on. Even worse, these control points have to be changed in a regular and consistent fashion, thus the control points are more tightly coupled and the effort needed for consolidation is very high. Few control points in contrast mean that each control point - as a degree of freedom - has a clear and isolated function. Finally, many limit points complicate the produced geometry unnecessarily and increase the probability that undesired oscillations are introduced.

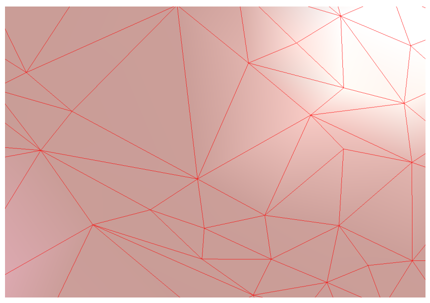
We also want the mesh widths to vary regularly for reasons stated in section 2.6.2, uniform parameter intervals leading to overshooting behavior if the segment lengths are changing too rapidly.

Of course curves in similar surface regions should obtain similar limit point distributions, in order to make the algorithm predictable and to generate matching sets of chains in homogeneous regions.

The last point is especially important to us. We want to make sure that the original design idea is conveyed to our digital model, and thus that neither an important feature gets lost nor an undesired feature is introduced into our final shape.

4.2 INPUT DATA ANALYSIS


(a) Missed regions closed by the triangulation process.


(b) Inhomogeneous mesh sampling.

(c) Hand-crafted clay models by nature obtain irregularities.

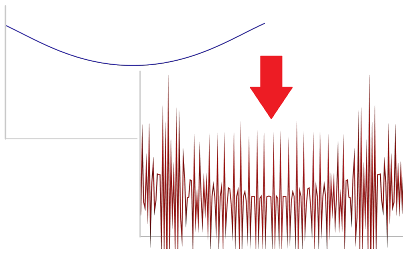
Figure 27: Data flaws introduced by data source and scanning process.

We now want to analyze the workflow for input data acquisition regarding flaws which might get introduced into our discrete input data. For this let a clay model of a car body be the source for a digital model, which is still common practice in the automotive sector. This model must be deemed irregular and flawed, as it is a real physical object - in this case even a hand-crafted prototype - and not fair in a high quality CAGD sense. This physical object is first captured by a 3D scanner, and converted into a triangle mesh. The resulting mesh contains various kinds of flaws.

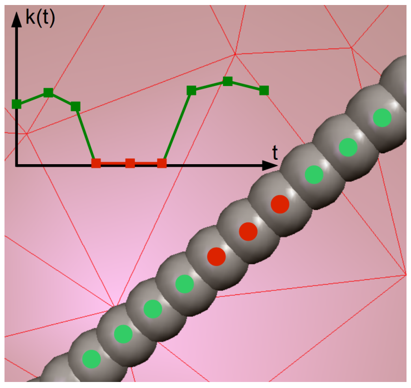
- Flaws introduced by the scanner, such as missed regions (figure 27a).

- Noise generated due to the scanner's limited spatial accuracy.
- An inhomogeneous or non-optimal sampling rate, which does not adapt to the local traits of the scanned geometry (figure 27b).
- Irregularities already present in the scanned object (figure 27c).

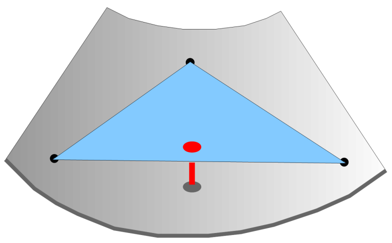
These factors could be used to model an estimate for the lowest error bound it makes sense to achieve with the fitting algorithm.



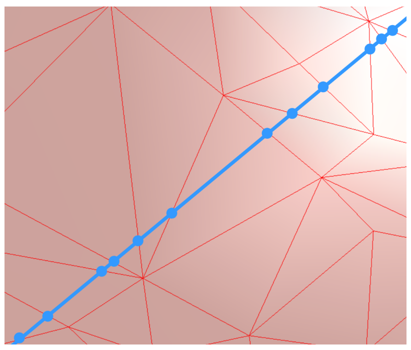
(a) Effect of closely sampled single precision data on discrete curvature.



(b) Zero curvature when sampling multiple points from a planar triangle.



(c) Triangle samples can obtain a substantial discretization error. The shown sample lies far away from the true surface the triangle was obtained from.



(d) Irregularly sampled polygonal curve obtained by cutting the mesh with a plane.

Figure 28: Data flaws introduced by curve sampling.

In a next step a discrete curve is sampled from the triangle mesh. In our case this is done by the user spanning a pearl chain. We can identify at least three sources for data flaws.

- Noise produced by numerical inaccuracies during sampling.
- A non-optimal sampling rate, which does not adapt to the local traits of the sampled triangle geometry.

- Irregular point spacings obtained by cutting the triangles with a plane.

Numerical inaccuracy depends on the sample rate and the numerical precision that is used for data storage. Even very small perturbations can influence computations such as discrete curvature estimation, as shown in figure 28a.

Non-optimal sampling can be poisonous in several ways. Too high sampling rates will result in excessive triangle subsampling and introduce zero-values into the discrete curvature profile (figure 28b). Triangle subsampling further generates curve positions that cannot be trusted, as triangle samples may lie far away from the true surface (figure 28c). Too low sampling rates on the other hand may result in badly sampled or omitted features, which is equally disastrous.

Even when obtaining the curve samples by an exact plane cut through the mesh, it is possible that triangles will get cut in different ways, e.g. near their corners or at their full length. This results in irregularly sampled polygonal curves (figure 28d).

Because of the various flaws which might have been introduced into our input data beforehand, we need our algorithm to be robust against perturbations. This problem is investigated from two different points of view.

In section 4.3 we discuss methods for preprocessing of the input data itself, in order to gain higher numerical accuracy for our computations, and smoother curve samples.

In section 4.4 we investigate and compare methods for robust discrete curvature estimation, in order to obtain a smooth curvature profile which can be trusted.

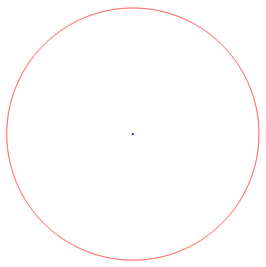
4.3 INPUT DATA PREPROCESSING

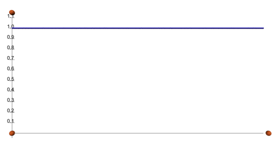
4.3.1 *Data normalization*

As a very first step, the input data is normalized before being processed further. This enforces good numerical precision and better conditioning of the algorithm. Figures 29 shows the effect of huge coordinates on the discrete curvature profile of sampled single precision data. Although a robust curvature estimator would smooth out the noise, we want to prevent such artifacts beforehand.

Such a normalization can be achieved in several ways, here a normalization procedure will be utilized that was proposed by Hartley in [17] for fundamental matrix estimation. The normalization employs a translation and isotropic scaling, such that:

- The data centroid rests at the origin $(0, 0)$.


(a) Test circle, $r = 1$


(b) Discrete curvature profile, circle constructed at origin.

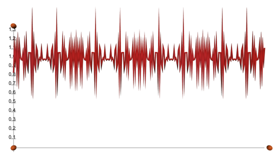

(c) Discrete curvature profile, circle constructed at $(1000, 1000)$

Figure 29: Precision errors as curvature noise

- The average distance of a point to the origin is $\sqrt{2}$, which equals the euclidean norm of a position on the unit circle.

The points are transformed to have zero mean and a standard deviation of $\sqrt{2}$ with respect to their distance to the origin. This normalization is similar to a studentization, which normalizes a random variable to have zero mean and a standard deviation of one.

4.3.2 *Data smoothing*

When estimating discrete curvature values from noisy samples, one possibility is to use noise robust curvature estimators like those presented in the next section. If the amount of noise is substantial, robust curvature estimators need to operate on a broader spatial scale in order to cancel out the noise. As we will see, this often results in a drop of amplitude, since a broader neighborhood of the point of interest has to be incorporated into the estimate. Alternatively we can transform the input data into a smoother representation, which also yields a smoother curvature profile. Compared to a robust curvature estimator this is an invasive procedure which changes the input data, but since the input data can only be trusted to a certain degree (section 4.1), it should not be prohibited to move the input samples a little, as long as they are moved economically and with care. It also makes sense to fit a smoothed version of the data since we do not want to fit noise-afflicted positions anyway.

There exist several techniques to obtain a smooth representation from noisy curve samples. In the next sub-sections some of them will be inspected.

In all upcoming examples a non-robust curvature estimator (an estimator based solely on a sample and its two neighbors) will be used, in order to highlight the effect of the smoothing algorithm on the curvature profile, without the influence of a robust curvature estimator's inherent smoothing.

4.3.2.1 *Laplacian smoothing*

The most basic method is Laplacian smoothing.

Definition 19 (Laplacian smoothing)

$$Q_k = \frac{1}{2}(P_{k-1} + P_{k+1})$$

Each point is simply moved to the centroid of its neighbors. Applying this rule to one vertex after another may result in long propagation time and oscillatory behavior, therefore the smoothed positions are calculated out-of-place. Depending on the smoothing technique this usually leads to overshooting, since the movement of one vertex does not account for the movements of its neighbors. A damping factor α is introduced to slow down the movement of vertices.

Definition 20 (Laplacian smoothing with damping)

$$Q_k = (1 - \alpha) \cdot P_k + \alpha \cdot \frac{1}{2} \cdot (P_{k-1} + P_{k+1})$$

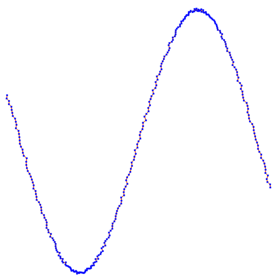
If α is set too high, the smoothing procedure will become unstable, if set too low, smoothing will be slow. The optimal value will depend on the respective smoothing algorithm. For Laplacian smoothing a factor of $\alpha = 0.9$ proved fine.

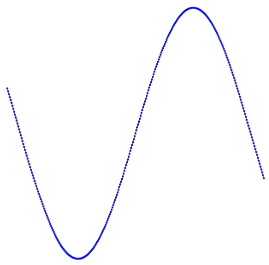
Figure 30 shows the effect of Laplacian smoothing on a sine curve consisting of 300 data points with added Gaussian noise. After only 80 iterations the noise is smoothed out as shown in b) and c). However, soon the data becomes strongly oversmoothed, as shown in d) and e). This effect, which is also called shrinking, is better visualized in f) using a circle. Just like the shape, curvature will successively shrink and degrade with more iterations.

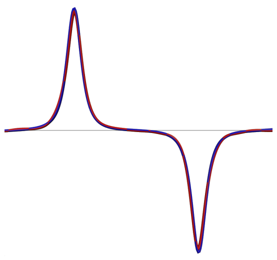
Laplacian smoothing is very effective, but also aggressive on the shape of a curve, and thus has to be applied very carefully.

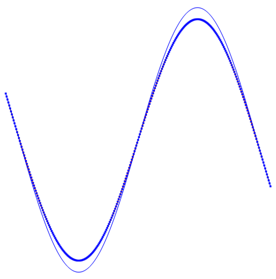
4.3.2.2 *Centroid preserving smoothing*

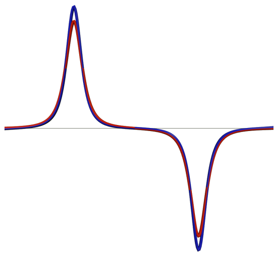
A centroid preserving smoothing technique that does inflict less shrinking on the data was introduced by van Overveld in [41] in the context of mesh smoothing. It is quite easy to adapt it for curve smoothing purposes.


(a) Test sine curve with added Gaussian noise, 300 samples.


(b) Curve after 80 iterations.


(c) Curvature after 80 iterations.


(d) Curve after 500 iterations.


(e) Curvature after 500 iterations.

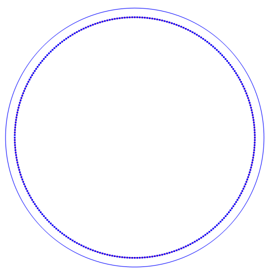

(f) Shrinking effect on a circle.

Figure 30: Various results for Laplacian smoothing.

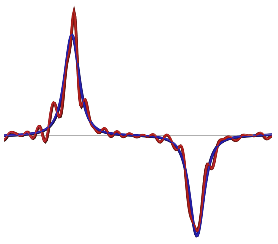
Definition 21 (Centroid preserving smoothing (van Overveld))

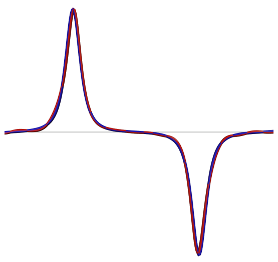
$$\begin{aligned}
\overline{P} = \frac{1}{2} \cdot (P_{k-1} + P_{k+1}) \\
D = P_k - \overline{P} \\
D_C = -\frac{2}{3} \cdot D \\
D_N = \frac{1}{3} \cdot D \\
Q_{k-1} = P_{k-1} + D_N \\
Q_k = P_k + D_C \\
Q_{k+1} = P_{k+1} + D_N
\end{aligned}$$

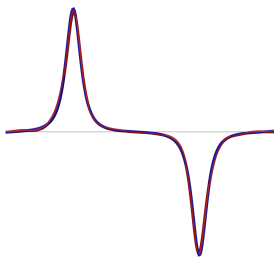
Offsets are not only calculated for the current vertex, but also for its two neighbors. This reveals the following effect.

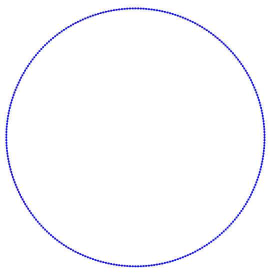
$$\begin{aligned}
\frac{1}{2} \cdot (Q_{k-1} + Q_{k+1}) = \frac{1}{2} \cdot (P_{k-1} + P_{k+1} + \frac{2}{3} \cdot D) = \\
P_k + \frac{1}{2} \cdot (P_{k-1} + P_{k+1}) - P_k + \frac{1}{2} \cdot \frac{2}{3} \cdot D = \\
P_k - D + \frac{1}{3} \cdot D = P_k - \frac{2}{3} \cdot D = Q_k
\end{aligned}$$

After the smoothing step the point P_k lies in the center of its neighbors, and as a nice side effect, the center of gravity of the three points remains the same. This cannot prevent shrinking completely, since the movement of a point consists of a total of three contributions, but it will slow the effect down. To make the smoothing step stable for all points, a damping factor has to be introduced once again. A factor of $\alpha = \frac{2}{3}$ performed well.


(a) Sine curve with added noise, 300 samples, curvature after 200 iterations


(b) Sine curve with added noise, 300 samples, curvature after 3500 iterations


(c) Sine curve with added noise, 300 samples, curvature after 8000 iterations


(d) Circle, $r = 1$ unit, 10000 iterations

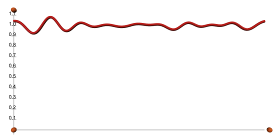

(e) Circle curvature, 10000 iterations

Figure 31: Various results for centroid preserving smoothing.

Figure 31 shows results for the same test data as in the last section. In a) and b) one can see that compared to Laplacian smoothing it takes far more iterations to smooth out the noise. Even though the data becomes smooth soon and the overall shape is preserved quite well, the algorithm also tends to preserve strong oscillations caused by the noise. With more iterations it becomes harder and harder to smooth these unwanted features out, because of the shape preserving property. This effect can even better be seen in d) and e). After 10000 iterations the circle is smoothed out and on average yields the expected constant curvature of one, but oscillations unfortunately remain.

Centroid preserving smoothing maintains the overall shape of the object much better, but it may also preserve oscillations that are caused by the noise itself. Once snapped into a smooth result, it becomes harder and harder to smooth out these oscillations. If the signal-to-noise ratio is not expected to be as severe as in the discussed examples, this smoothing technique will perform quite well though.

4.3.2.3 *MLS curves*

Another interesting option for estimating smooth curve samples from noisy data is to make use of robust implicit shape representations.

Moving least squares (MLS) surfaces are surface representations which are implicitly defined by a set of 3D positions. A point is typically pulled from an initial position to a smooth surface position by iteratively querying its current neighborhood and computing a new position from it, which lowers the point's distance error in the least squares sense.

Robust implicit MLS surfaces (RIMLS) were introduced by Öztireli et al. in [29]. They improve former approaches by making use of robust statistics in order to get better outlier suppression and by introducing a sharpness parameter for better reconstruction of details and sharp bends. Although introduced for surfaces, it is quite easy to adapt the method to generate implicit curves. We will shortly review the cornerstones of the text, for more details refer to [29].

The essence of the problem is to find an approximation to the signed distance function $f(\mathbf{x})$, which describes the local distance of a point to the smooth surface. The input to the algorithm is in our case a polygonal curve. We generate normals for each vertex by using its neighboring segment directions. Since this estimate yields rather noisy normals, we use the method described in [29] to smoothen them. By making use of a local neighborhood an initial normal can be retrieved as

$$\mathbf{n}_j^0 = \frac{\sum_{i,i\neq j} \phi_i(\mathbf{p}_j)\mathbf{n}_i}{\sum_{i,i\neq j} \phi_i(\mathbf{p}_j)}, \tag{17}$$

where

$$\phi_i(\mathbf{x}) = \left(1 - \frac{\|\mathbf{x} - \mathbf{x}_i\|}{h_i^2}\right)^4$$

is a weight function. The spatial radii h_i are set to a multiple f_s of the average local point spacing, this factor thus influences the amount of smoothing.

Smooth normals can then be retrieved iteratively by using

$$\mathbf{n}_j^k = \frac{\sum_i \phi_i(\mathbf{p}_j) w_n(\|\mathbf{n}_j^{k-1} - \mathbf{n}_i\|)\mathbf{n}_i}{\sum_i \phi_i(\mathbf{p}_j) w_n(\|\mathbf{n}_j^{k-1} - \mathbf{n}_i\|)}. \tag{18}$$

Here w_n is a normal weight function defined as

$$w_n(x) = e^{-\left(\frac{x}{\sigma_n}\right)^2}.$$

Using the obtained smooth normals, the iterative RIMLS definition is

$$f^k(\mathbf{x}) = s_0 = \frac{\sum \mathbf{n}_i^T(\mathbf{x}-\mathbf{x}_i)\phi_i(\mathbf{x})w(\mathbf{r}_i^{k-1})w_n(\Delta\mathbf{n}_i^k)}{\sum \phi_i(\mathbf{x})w(\mathbf{r}_i^{k-1})w_n(\Delta\mathbf{n}_i^k)}. \tag{19}$$

The weight function $w(\mathbf{r}_i^{k-1})$ is used for outlier rejection and weights the residuals

$$\mathbf{r}_i^{k-1} = f^{k-1}(\mathbf{x}) - (\mathbf{x}_i - \mathbf{x})^T\mathbf{n}_i.$$

It is *Welsch's function*, which is defined as

$$w(x) = e^{-\left(\frac{x}{\sigma_r \cdot h_i}\right)^2}.$$

The function $w_n(\Delta\mathbf{n}_i^k)$ is introduced to reproduce sharp features more accurately. It penalizes samples whose normals deviate strongly from the predicted RIMLS gradient. This difference can be expressed as

$$\Delta\mathbf{n}_i^k = \|\nabla f^k(\mathbf{x}) - \mathbf{n}_i\|.$$

The symmetric weight function $\phi_i(\mathbf{x}) = \phi(\|\mathbf{x}-\mathbf{x}_i\|)$ gives more weight to closer samples, and reflects the fact that the approximations used for this approach are only valid at values near $\mathbf{x}$.

Setting $w_i = w(\mathbf{r}_i^{k-1})w_n(\Delta\mathbf{n}_i^k)$ constant for one projection step as an approximation leads to the following gradient definition.

$$\nabla f^k(\mathbf{x}) = \frac{\sum w_i\phi_i(\mathbf{x})\mathbf{n}_i + \sum w_i\nabla\phi_i(\mathbf{x})\left(\mathbf{n}_i^T(\mathbf{x}-\mathbf{x}_i) - f^k(\mathbf{x})\right)}{\sum \phi_i(\mathbf{x})w(\mathbf{r}_i^{k-1})w_n(\Delta\mathbf{n}_i^k)} \tag{20}$$

Then a simple steepest descent approach can be used to obtain a smooth position for a given point. Some details important for implementation are listed below.

- Setting the initial weights to one results in a pure least squares approach to initialize good starting positions in the first iteration.

- For fast dismissal a spatial data-structure should be queried (in our implementation a quadtree is utilized) in order to retrieve the neighborhood. This is the bottleneck of the algorithm.
- The choice of σ_n depends on the desired amount of sharpness, smaller values yielding sharper results.

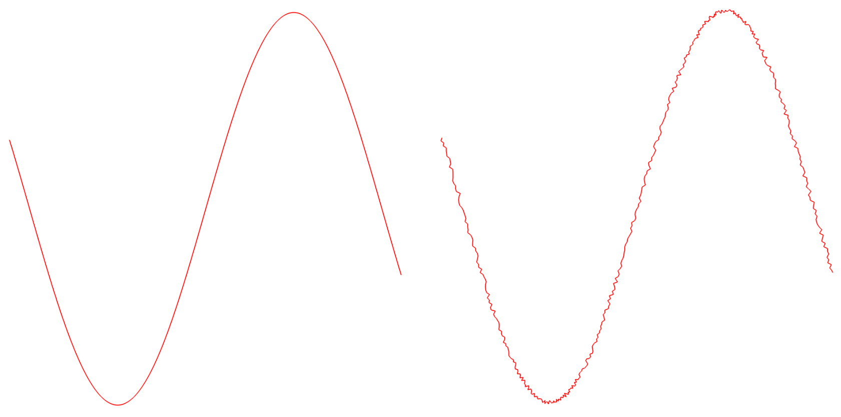

(a) Sine curve (b) Sine curve with added noise

Figure 32: Test data for the RIMLS algorithm.

Figure 32 shows synthetic curve data used for validation of the implemented RIMLS code. Normally distributed noise was added to a sine curve in normal direction.

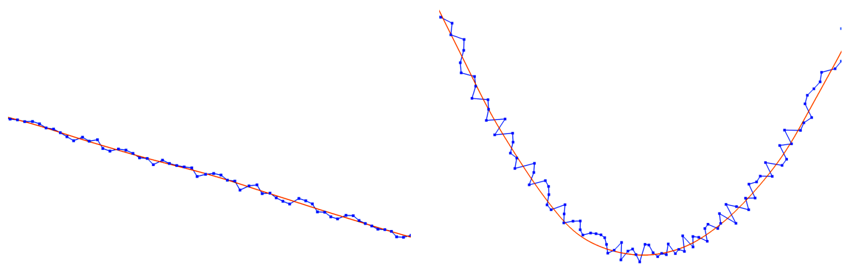

(a) (b)

Figure 33: Various results for the RIMLS algorithm. $h = 10.0$, $\sigma_r = 0.5$, $\sigma_n = 1.5$, 40 iterations

Figure 33 shows results for a straight detail and a high-curvature detail of the curve. As expected the algorithm produces a smooth curve from the noisy samples. In some part of b) one can see that the resulting curve slightly deviates from the data, which may indicate that the spatial scale h has been chosen too big for this curve region.

In figure 34 the spatial radius has been chosen smaller, and the produced curve now follows the samples more accurately. On the other

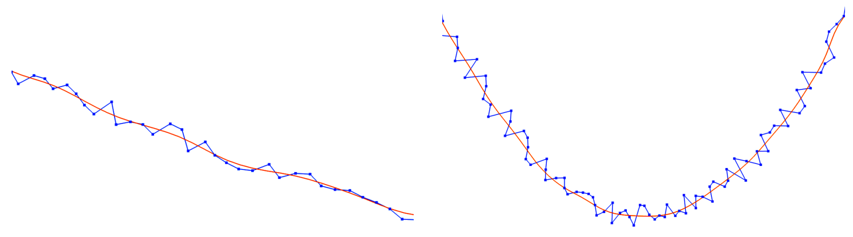
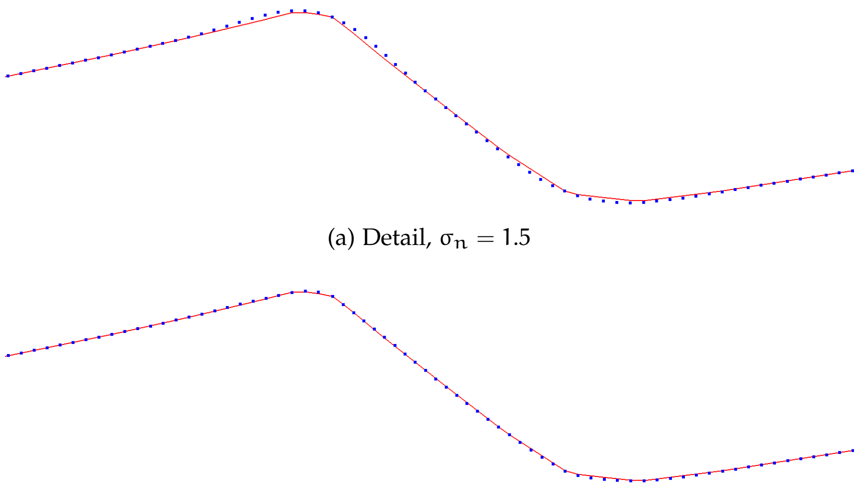

(a) (b)

Figure 34: Various results for the RIMLS algorithm. $h = 5.0$, $\sigma_r = 0.5$, $\sigma_n = 1.5$, 40 iterations

hand the solution now obtains more oscillations.

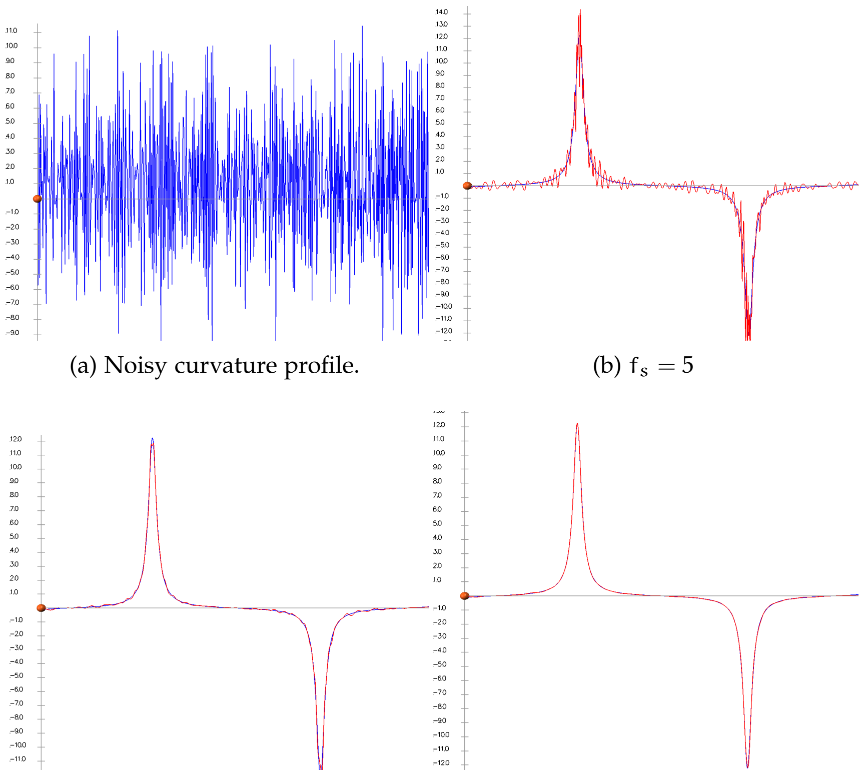

(a) Detail, $\sigma_n = 1.5$

(b) Detail, $\sigma_n = 0.2$

Figure 35: RIMLS results for non-synthetic data, detail. $h = 10.0$, 20 iterations

Figure 35 shows the effect of the sharpness parameter σ_n on the S-part of a curve sampled from the scan of a car door. A higher σ_n will make the smooth curve deviate from the original data in regions of strong change, especially if higher spatial scales are needed in the presence of noise. If we choose a lower σ_n, the MLS-curve will stick to the original bend more closely, as shown in 35b, although oscillations in the guidance points will have a stronger impact on the result.

Since we are very concerned about how our smoothing procedure alters the curvature of the original data, we again have a look at our noisy sine curve. Figure 36 shows how the discrete curvature converges to the ground truth curvature of the sine curve with increasing spatial scale f_s. There is no visible shrinking effect in the final result. We of course only assume pure Gaussian noise, it is expected that other kinds of noise will not get filtered out as perfectly as in this



(a) Noisy curvature profile. (b) $f_s = 5$

(c) $f_s = 10$ (d) $f_s = 15$

Figure 36: Curvature behavior of RIMLS, demonstrated on a sine curve with added Gaussian noise. Ground truth=blue, MLS-curve=red.

example.

Another advantage of implicit curves is that we are able to retrieve a smooth representation of our original data in as much detail as we wish, independent of the initial point density. If our input data is not sampled very densely, we can always retrieve a denser version by projecting more points onto the MLS-curve. One could for instance double the density easily by projecting all vertices and segment midpoints. In case of initial positions which are far away from the MLS shape or very densely sampled curves we have to take care of point ordering though.

One disadvantage with this method is that we cannot guarantee that one parameter set will always lead to the desired results. The effect of the spatial scale f_s for instance depends on the point density. It has more impact on a certain shape feature if the feature's resolution is low.

Furthermore, we cannot increase f_s as much as we want, since more compact features may get mixed up if they lie very close to each other.

Finally, if the noise on the data is rather low, the MLS-curve is not given much room to adapt to the given samples. In this case the sharpness parameter σ_n needs to be chosen quite low in order to force the MLS estimate to follow the input data more closely.

4.3.3 *Resampling*

Samples obtained from a laser scanner or a photogrammetric setup will often be distributed in a non-feature-adapted and inhomogeneous way, which is not very desirable.

- Low-curvature parts are sampled too densely, introducing unnecessary noise and oscillations.
- High-curvature parts are sampled too sparsely, diminishing data support for important features.

We thus want to move higher sample densities away from unimportant shape regions into important ones (features) and obtain smoothly varying samples. For this we make use of a curve resampling approach introduced by Baran et al. in [3]. It enforces smooth, global compliance to a given guidance function, and allows to choose a desired amount of variation. Any roughly smooth function can act as a guidance function, which makes the method pretty versatile.

Baran et al. consider a resampling function $r(s)$.

$$r(s) = \min_i \left(|s - s_i| \cdot \beta + \frac{2 \cdot \pi}{\gamma \cdot \kappa_i} \right) \tag{21}$$

The first term enforces smoothness of variation by weighing distance to surrounding samples against the locally needed spacing. The second term represents the guidance function, in this case curvature. It divides the circle of curvature at the i^{th} sample by a factor γ, this way relating the local point spacing to the local curvature. It is important to note that any function yielding sample spacings could act as a guidance term here.

Using $r(s)$, we can generate a resampled point distribution. Starting at arc length s_j, the next point at s_{j+1} can be retrieved by running along the curve as long as the following condition holds.

$$s_{j+1} - s_j \leqslant \operatorname*{argmin}_{s \in [s_{j-1}, s_j]} \quad r(s) \tag{22}$$

This way all locally needed spacings are taken into account.

The number of samples produced can be varied in two ways. The factor γ directly influence the number of samples via the guidance function. The factor β defines the allowed amount of variation in the produced sample distances. Setting $\beta = 0$ spaces the samples equally using the highest curvature value. Setting β to a high value enables the samples to vary more freely, resulting in fewer samples needed to realize a change of density.

In our implementation, a small stepsize ds is used for sub-sample accuracy. The resampling algorithm can be summarized as follows.

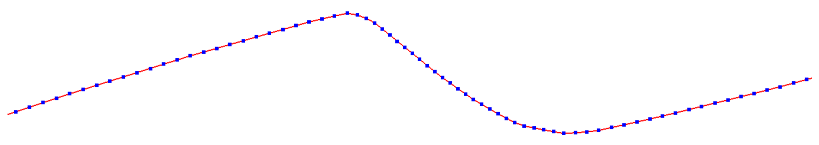
1 Insert a sample at length $s = 0$.

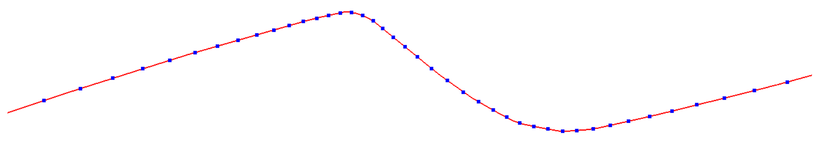
2 Take a step ds along the polygonal curve and check equation 22 for the current position s_{cur}.

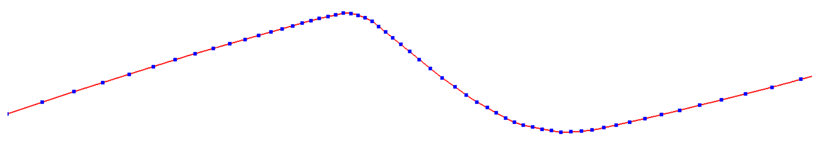
3 If equation 22 holds continue; otherwise, insert a sample at $(s_{cur} - ds)$.

4 Alternate 2 and 3 until the curve's end is reached.

5 Insert a sample at the endpoint if needed.


(a) Input curve, uniform sampling.


(b) Curvature adapted resampling, $\beta = 0.1$, $\gamma = 100$.


(c) Curvature adapted resampling, $\beta = 0.1$, $\gamma = 150$.

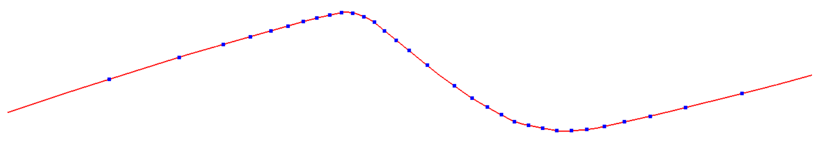

(d) Curvature adapted resampling, $\beta = 0.6$, $\gamma = 100$.

Figure 37: Various results for curvature guided resampling.

Figure 37 shows results for the sharp S-part of a curve.

4.4 CURVATURE CALCULATION FROM DISCRETE DATA

In this section we will investigate various methods to calculate curvature profiles from noisy discrete data. It will both cover methods which are not robust against noise but very fast to calculate, and methods which are computationally more involved but on the other hand more robust.

Since our approach to B-spline fitting is curvature-driven, it is important to extract a curvature profile as accurate and stable as possible. But this can be problematic for many reasons.

- Curvature calculation - involving the first and second derivative - is extremely sensitive to noise. This calls for either preliminary smoothing of the data or robust means of estimation.
- Many robust methods may introduce a smoothing effect that will lower the curvature profile's amplitude.
- This results in a tradeoff between robustness and accurate amplitude.
- No curvature estimator can fully compensate data-loss which happened in previous processing stages.

We will analyze each method with respect to the first three points. As for the last problem, in order to be able to conduct a valid curvature estimation the input data is expected to be sampled densely enough. However, it is unlikely that we are able to estimate the *intended* (designed) curvature from our discrete data at all. We will discuss this issue later on.

4.4.1 *Method of comparison*

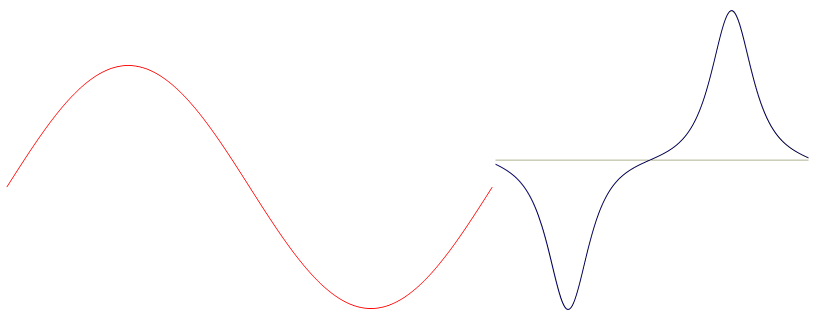

(a) Sinusoidal curve (b) Curvature profile

Figure 38: Signal used to validate the discussed curvature estimation techniques.

At the end of each sub-section results will be presented for both noise-free and noise-afflicted data. The chosen test data consists of a sinusoidal signal.

$$y(t) = \frac{1}{4} \cdot \sin(2\pi t) \quad \mathbf{g}(t) = (t, y(t)) \qquad t \in [0,1]$$

The curvature of this graph can be calculated as (equation 7)

$$\kappa(t) = -\frac{\pi^2 \cdot \sin(2\pi t)}{\left(1 + \frac{\pi^2}{4} \cdot \cos^2(2\pi t)\right)^{\frac{3}{2}}}.$$

The curvature estimated from the discrete values will be compared to the ground-truth profile, and we will discuss the influence of noise, as well as other characteristics of each method. In order to obtain noisy samples the curve is distorted by noise following a Gaussian distribution. Figure 38 shows the test signal and its curvature profile.

4.4.2 *Signed curvature*

Some methods described in this section only yield absolute curvature values. In order to obtain the sign one must rely on some convention, e.g. a fixed quantity. In case of a sufficiently planar curve, a reference normal **N** can be provided that represents the plane the curve lies in. For a point triplet the sign of curvature then easily can be chosen using the following rule.

$$\mathbf{V}_1 = \mathbf{p}_k - \mathbf{p}_{k-1} \qquad \mathbf{V}_2 = \mathbf{p}_{k+1} - \mathbf{p}_k$$

$$S = (\mathbf{V}_1 \times \mathbf{V}_2) \cdot \mathbf{N} \qquad \operatorname{sign}(\kappa) = \begin{cases} -1 & \text{if } S < 0 \\ 1 & \text{if } S \geqslant 0 \end{cases}$$

In the presence of noise it is hard to obtain useful information this way. The method can be easily extended to be more robust by using a sphere with radius r at the point of interest $\mathbf{p}$. The sphere intersects the curve at the positions $\mathbf{s}_1$ and $\mathbf{s}_2$. Retrieving the sign from the triplet $(\mathbf{s}_1, \mathbf{p}, \mathbf{s}_2)$ is more robust, but the ball radius has to be chosen wisely - feature adapted at best.

4.4.3 *Curvature from segment angles*

The most basic way to assign curvature values to discrete curve samples is to look at three consecutive points forming two neighboring segments. In our discrete curve scenario this configuration is the minimal Frenet frame.

A single curve sample $\mathbf{p}_k$ and its two neighbors $\mathbf{p}_{k-1}$, $\mathbf{p}_{k+1}$ form two segments $\overline{\mathbf{p}_{k-1}\mathbf{p}_k}$ and $\overline{\mathbf{p}_k\mathbf{p}_{k+1}}$. A useful relation for arc-length parametrized discrete curves comes from the field of discrete differential geometry and can be found e.g. in [40]. By calculating the segment direction vectors

$$\mathbf{D}_1 = \frac{\mathbf{p}_k - \mathbf{p}_{k-1}}{\|\mathbf{p}_k - \mathbf{p}_{k-1}\|} \qquad \mathbf{D}_2 = \frac{\mathbf{p}_{k+1} - \mathbf{p}_k}{\|\mathbf{p}_{k+1} - \mathbf{p}_k\|}$$

one can define the curvature normal as follows.

Definition 22 (Curvature normal)

$$\mathbf{N}_c = \kappa \cdot \mathbf{N} = \mathbf{D}_1 - \mathbf{D}_2$$

The curvature normal $\mathbf{N}_c$ describes the length gradient in a discrete sense and its length yields the discrete mean curvature. It can easily be calculated as

$$\kappa = 2\sin\left(\frac{\alpha}{2}\right),$$

where α is the angle between $\mathbf{D}_1$ and $\mathbf{D}_2$ (see figure 39). This formula is scale invariant, since it depends only on the angle between he segments. The curvature characteristic obtained this way has to be seen as a distribution of a fixed amount of curvature over the discrete curve. At higher sampling densities the curvature values will decrease with the segment angles, as the total curvature will be distributed over more values. For accurately scaled curvature values, one has to account for the segment lengths l:

$$\kappa = \frac{2}{l}\sin\left(\frac{\alpha}{2}\right).$$

For very small angles $\sin(\alpha)$ becomes approximately α and the formula simplifies to

$$\kappa \approx \frac{\alpha}{l}.$$

For curves which are not parametrized by arc-length this formula changes to (see [22])

$$\kappa \approx \frac{2\alpha}{l_1 + l_2},$$

l_1 and l_2 being the segment lengths which differ in this case.

The calculation of the segment angle α can be a source for numerical problems. A basic way to calculate α is to use the following well-known relation.

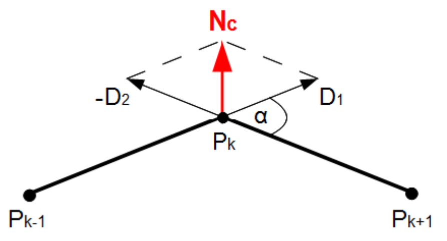


Figure 39: The mean curvature is calculated from the normalized segment directions.

$$\alpha = \operatorname{acos}(\, \mathbf{V}_1 \cdot \mathbf{V}_2 \,) \qquad \mathbf{V}_1 = \frac{\mathbf{p}_k - \mathbf{p}_{k-1}}{||\mathbf{p}_k - \mathbf{p}_{k-1}||}, \; \mathbf{V}_2 = \frac{\mathbf{p}_{k+1} - \mathbf{p}_k}{||\mathbf{p}_{k+1} - \mathbf{p}_k||}.$$

The acos operation loses much precision if the angle is near any multiples of π (including zero), where the derivative of cosine is zero. This has the counter-intuitive effect that curvature computation with acos becomes less precise when the sampling density increases. There is also a chance for bit roundings, resulting in a dot product out of the interval $[0, 1]$ on which the acos function is defined.

The numerically more stable method uses the atan2 operation.

$$\alpha = \operatorname{atan2}(\, ||\mathbf{V}_1 \times \mathbf{V}_2||, \, \mathbf{V}_1 \cdot \mathbf{V}_2 \,)$$

An alternative to trigonometric functions for angles near zero is to calculate the projection of the point $\mathbf{p}_{k+1}$ on the line given by $\overline{\mathbf{p}_{k-1}\mathbf{p}_k}$ (see figure 40). Since the angle is by definition very small, this leads to the relation

$$\sin(\alpha) = \frac{||\mathbf{p}_{k+1} - \mathbf{p}_{proj}||}{||\mathbf{p}_{k+1} - \mathbf{p}_k||} \approx \alpha.$$

There is a small error within the calculation of $\mathbf{p}_{proj}$, but generally this operation is very stable.

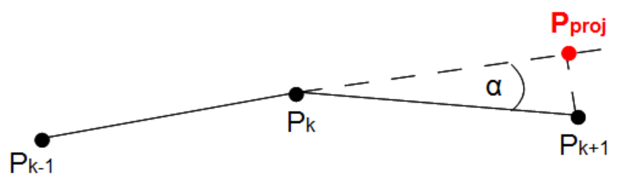
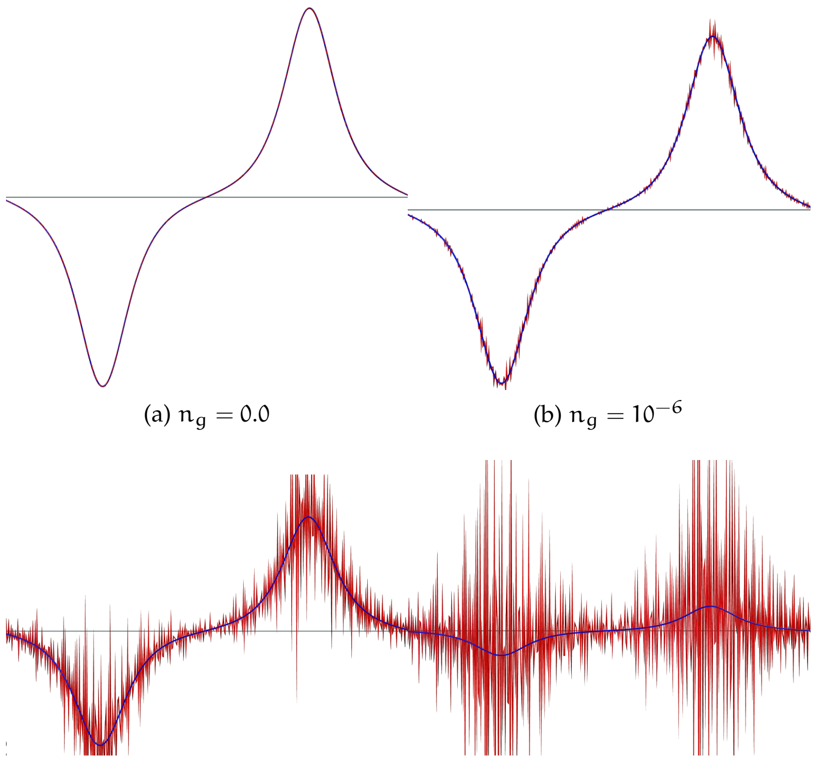


Figure 40: Angle through projection: $\mathbf{p}_{k+1}$ is projected onto the line defined by the first segment.

The following table compares the methods proposed for calculation of the segment angle in terms of precision. The table shows the absolute error of the result (double precision, 30 decimals shown). One can see that near zero atan2 has far better precision than acos. Still better when having very small angles is the projection method, but the error grows very fast with the angle getting bigger.

Reference angle	Error acos	Error atan	Error projection
10^{-}8	0.0000000100000000000000000000	0.0000000000000000000000000333619	0.000000000000000000000000000000
10^{-}7	0.0000000000399718806241097540000	0.0000000000000000000000005783802	0.0000000000000000000000172053567
10^{-}6	0.000000000004444930334196474100	0.0000000000000000000000047045897	0.000000000000000000166441974135
10^{-}5	0.000000000000413743512513815510	0.0000000000000000000000576544887	0.000000000000000166667284899440
10^{-}4	0.000000000000262206882183264880	0.00000000000000000000002474924393	0.000000000000166666661483190480
10^{-}3	0.000000000000007831409132297296	0.00000000000000000000013552527156	0.000000000166666658339004180000
10^{-}2	0.000000000000001441555208536727	0.0000000000000000000282061971436	0.000001666658333357440300000000
10^{-}1	0.000000000000000555111512312578	0.0000000000000000008558420899057	0.000166583353171836920000000000


(a) $n_g = 0.0$ (b) $n_g = 10^{-6}$

(c) $n_g = 10^{-5}$ (d) $n_g = 10^{-4}$

Figure 41: Curvature from angles: Results for a sinusoidal curve at various noise levels. Reference curvature=blue, calculated curvature=red

Figure 41 shows results for the sinusoidal curve. On the noiseless curve the true curvature can be retrieved very accurately if the curve is sampled dense enough. However, with added noise this method soon becomes unusable, since a noise amplitude as small as 10^{-4} will distort the curvature profile beyond recognition.

4.4.4 *Curvature from circumcircle*

Another fast method makes use of the fact that three consecutive points form a triangle, and that the circumcircle of this triangle approximates the local circle of curvature. The radius of the circumcircle through three points **a**, **b**, **c** can be calculated as

$$r = \frac{a \cdot b \cdot c}{4F} \quad a = \left\|\overline{\mathbf{ab}}\right\|, b = \left\|\overline{\mathbf{bc}}\right\|, c = \left\|\overline{\mathbf{ac}}\right\|,$$

where F is the area of the triangle $\overline{\mathbf{abc}}$. One can invert the formula to obtain a measure for the curvature value.

$$|\kappa| = \frac{4F}{a \cdot b \cdot c} \tag{23}$$

The area F can be expressed as

$$F = \frac{\left\|\overline{\mathbf{ab}} \times \overline{\mathbf{ac}}\right\|}{2}, \tag{24}$$

so when inserting into 23 one obtains

$$|\kappa| = \frac{2\left\|\overline{\mathbf{ab}} \times \overline{\mathbf{ac}}\right\|}{a \cdot b \cdot c}. \tag{25}$$

Numerically this expression is remarkably stable, since distances and the cross product can be evaluated very accurately even at angles near zero. Only very small segment lengths may introduce stability issues.

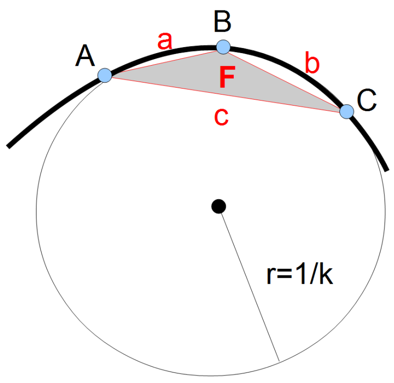
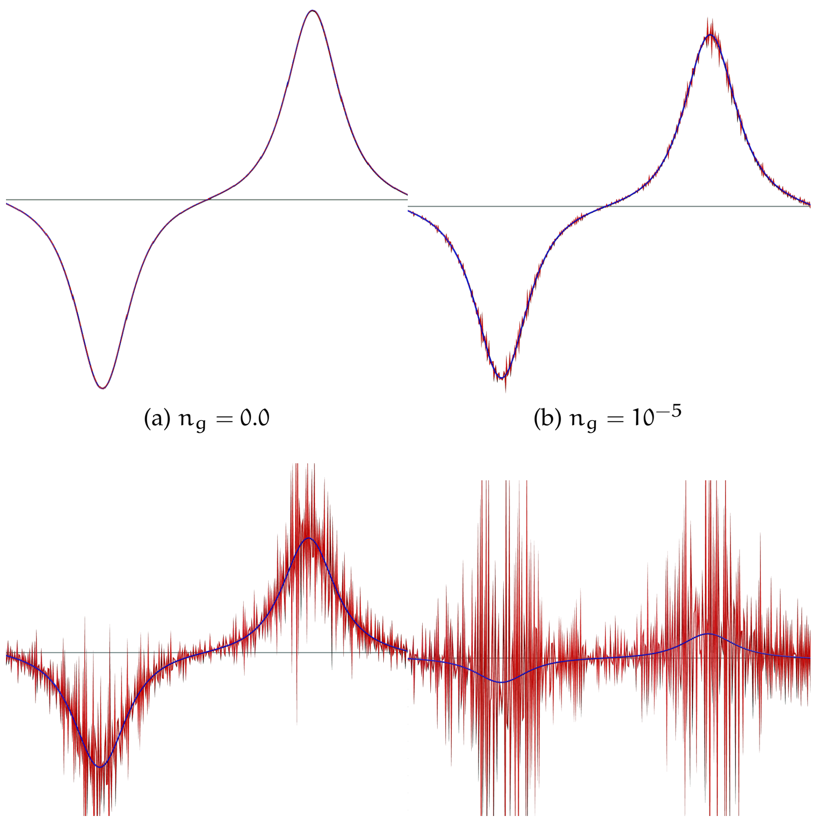


Figure 42: Curvature from circumcircle.

The configuration is visualized in figure 42.

As shown in figure 43 the three point method also suffers from a strong sensitivity to noise, but its robustness increases nearly in the order of magnitude compared to the segment angle based method. An explanation for this could lie in its increased numerical robustness.



(a) $n_g = 0.0$

(b) $n_g = 10^{-5}$

(c) $n_g = 10^{-4}$

(d) $n_g = 10^{-3}$

Figure 43: Curvature from circumcircle: Results for the sinusoidal curve at various noise levels n_g. Reference curvature=blue, calculated curvature=red

4.4.5 *Robust three point method*

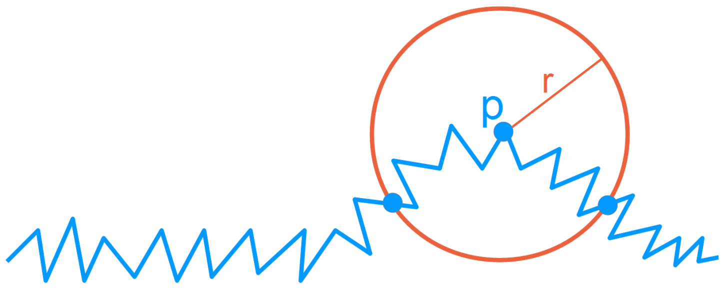


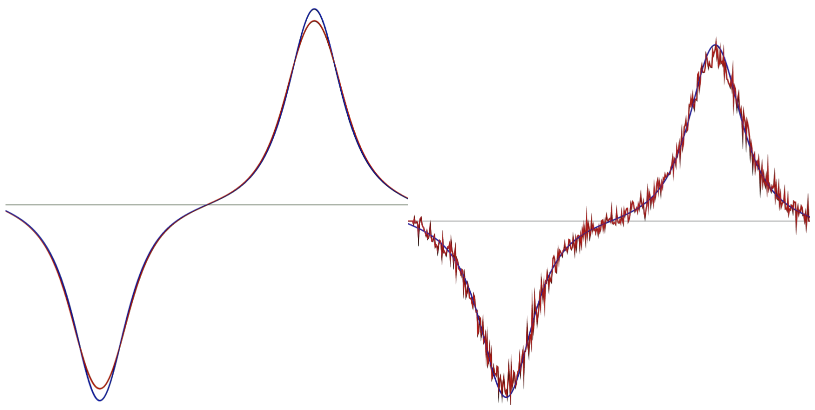
Figure 44: Robust retrieval of three points for discrete curvature calculation using spheres of radius r.

Curvature estimation from three consecutive samples is very sensitive to noise. Similar to the robust version of sign computation (see 4.4.2) it can be extended to use a sphere of radius r around the vertex of interest $\mathbf{p}$ for retrieval of three more robust positions. This is shown in figure 44.

There are some downsides to this method.

- The method obtains an inherent smoothing effect.
- Big radii which are not adapted to the local feature geometry will generate erroneous curvature.
- Properties of the captured curve segment are not taken into account.

The choice of the correct spatial radius is thus of the utmost importance, as with all neighborhood-based methods.



(a) $n_g = 0.0$, $r = 0.05$ (b) $n_g = 10^{-3}$, $r = 0.05$

Figure 45: Performance of the neighborhood-extended three point method in terms of smoothing and noise suppression. Reference curvature=blue, calculated curvature=red

Figure 45 shows results for the circumcircle method. In 45a the smoothing effect is visualized, 45b shows the behavior in presence of noise. We can see that even substantial amounts of noise can be suppressed quite successfully, in exchange for a notable loss of amplitude.

4.4.6 *Curvature from integral invariants*

Extending the three point methods to ball neighborhoods already achieves a great amount of robustness against noise, but one big problem is that we just jump back and forth on the curve. We do not really take into account the geometric properties of the curve segment which is captured by the ball neighborhood.

Integral invariants were introduced by Manay et al. in [28]. The original intent was to define unique descriptors for planar curves that are robust against noise, to be used e.g. for curve matching. Integral invariants are obtained by integrating over local neighborhoods of a shape, which are represented by balls of a certain radius. Compared to methods from the field of discrete differential geometry, they have the advantage of not having to change the geometry itself in order to implement multiscale behavior. Instead the different scales are represented by the radius of the ball neighborhood involved. The use of integration introduces an implicit smoothing effect, making integral invariants less sensitive to noise. Further, they can be described and visualized in an elegant way.

Inspired by the work of Manay et al., Pottmann et al. studied the behavior of various integral invariants and their relation to curvature in [35]. All of the following definitions and explanations can be found in more detail in their text.

The key concept behind integral invariants can easily be explained in two dimensions. A curve in the plane splits a planar area into two parts, one part named the domain D. Around the curve point of interest $\mathbf{p}$, there is a neighborhood region $B_r(\mathbf{p})$ represented by a disk of radius r (see figure 46). Two important quantities regarding curvature estimation in 2D are introduced, one being the intersection of the disk $B_r(\mathbf{p})$ with the domain D, which yields a planar area. We can describe this disk by a unit disk B, scaled by the radius r and shifted to $\mathbf{p}$.

$$A_r(\mathbf{p}) = B_r(\mathbf{p}) \cap D = (\mathbf{p} + rB) \cap D \qquad \text{(Area invariant)}$$

The other one is the intersection of the disk's outline $S_r(\mathbf{p})$ (a circle) with the domain D, represented by a circular arc. Similarly, we can describe this outline by a unit circle S, scaled by the radius r and shifted to $\mathbf{p}$.

$$CA_r(\mathbf{p}) = S_r(\mathbf{p}) \cap D = (\mathbf{p} + rS) \cap D \qquad \text{(Arc invariant)}$$

The radius r represents a certain scale of estimation, since features that are much smaller than r will not have much influence on the result of the integration. It is important to note that the terms »disk« and »circle« can always be changed to »ball« and »sphere« when treating surfaces. In this case the area invariant for instance becomes the volume invariant.

Pottmann et al. define the indicator function $1_D(\mathbf{x})$ as the function yielding $1_D(\mathbf{x}) = 1$ if the point $\mathbf{x}$ lies in the domain D, and $1_D(\mathbf{x}) = 0$ otherwise. The area invariant of Manay et al. can then be described by the following integral.

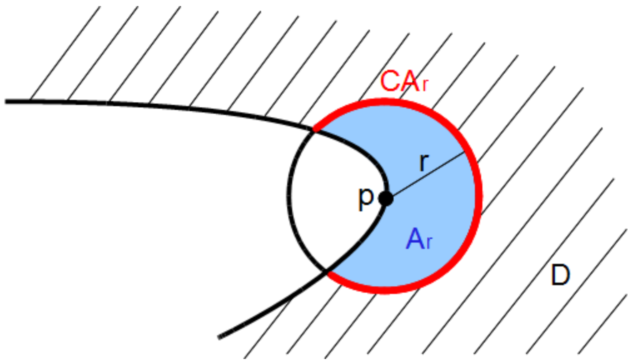


Figure 46: Integral invariants: The disk with radius r intersects the domain D resulting in the area A_r. The circle with the same radius intersecting with D results in CA_r, which is the length of a circular arc.

Definition 23 (Area invariant)

$$A_r(\mathbf{p}) = \int_{\mathbf{p}+rB} 1_D(\mathbf{x})\, d\mathbf{x}$$

The arc invariant can be described in a similar way.

Definition 24 (Arc invariant)

$$CA_r(\mathbf{p}) = \int_{\mathbf{p}+rS} 1_D(\mathbf{x})\, d\mathbf{x} = \frac{d}{dr} A_r(\mathbf{p})$$

Pottmann et al. propose estimates for the two invariants by using the Taylor expansion

$$CA_r = \pi r - \kappa r^2 + O(r^3), \tag{26}$$

which can be derived by locally describing the curve by the parabola

$$y = \kappa \frac{x^2}{2}.$$

By integration the area invariant can be expressed as

$$A_r = \frac{\pi}{2} r^2 - \frac{\kappa}{3} r^3 + O(r^4). \tag{27}$$

It is important to note that these estimates only hold for smooth curves.

Calculating the length CA_r of the circular arc (figure 46) can be achieved easily.

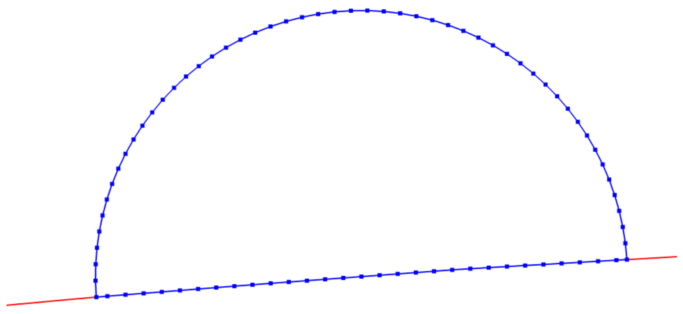


Figure 47: Polygon used to approximate the area for A_r.

The fast and precise calculation of A_r remains. There exist various approaches to approximate the area from discrete data. We can for instance close the curve segment captured by the ball neighborhood using samples from a circular arc. The discrete curve and the sampled arc then form a closed polygon (see figure 47), whose area can easily be calculated as follows.

Definition 25 (Area for simple polygons) *Be* $\mathbf{P}$ *a simple* 2D *polygon with* n *vertices* $\mathbf{p}_0, \ldots, \mathbf{p}_{n-1}$. *The area of this polygon is then given by*

$$A = \frac{1}{2} \sum_{i=0}^{n-1} x_i y_{i+1} - x_{i+1} y_i \quad \mathbf{p}_n = \mathbf{p}_0$$

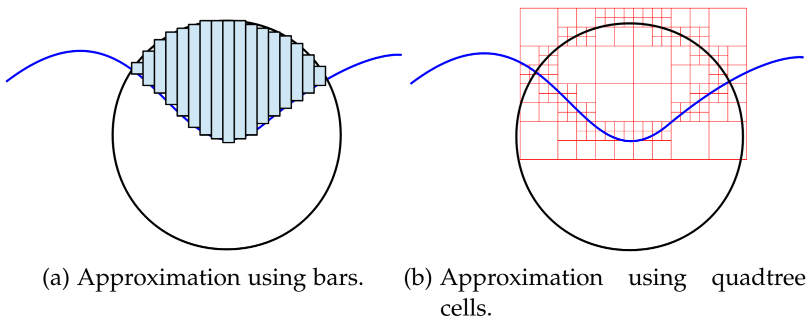

(a) Approximation using bars. (b) Approximation using quadtree cells.

Figure 48: Methods to approximate the area for A_r.

Assumed the neighborhood is adapted to the captured feature such that the curve piece inside approximately forms a graph, the area can be approximated by rectangular bars, as shown in 48a. This should be feasible for any data which varies reasonably smooth.

For the 3D case Pottmann et al. suggest to discretize the volume invariant using octree cells. The same can be achieved for 2D curves using a quadtree (see figure 48b). Further, since the desired integrals can be obtained by convolution of the indicator function with the respective neighborhood, a grid approach is suggested, which involves calculation of the FFT. The problem here remains to find a fast approximation to the indicator function.

Irrespective of the used method a poor estimate will result in a curvature offset, since equation 27 compares the estimate with the area of a half circle ($r^2\pi/2$).

For our implementation the polygonal approach and the graph-like approach have been implemented, and both performed well in our scenario.

Equations 26 and 27 already produce signed curvature values by comparing the invariants to half-circle counterparts. If the correct side of the domain can be identified for all computations, the curvature signs will be coherent.

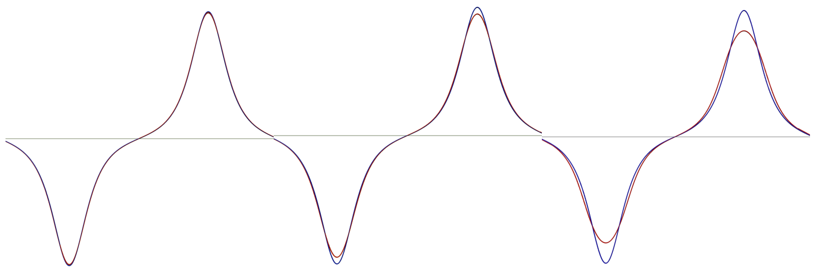

(a) $r = 0.02$ (b) $r = 0.05$ (c) $r = 0.1$

Figure 49: Curvature from arc invariants: Influence of scale on the resulting curvature. Reference curvature=blue, estimated curvature=red

Figure 49 shows curvature values computed from CA_r at various scales of estimation. No noise was added to the test data in order to visualize the effect of the neighborhood radius on the estimation. The inherent smoothing effect results in a drop of amplitude, getting bigger as the radius widens.

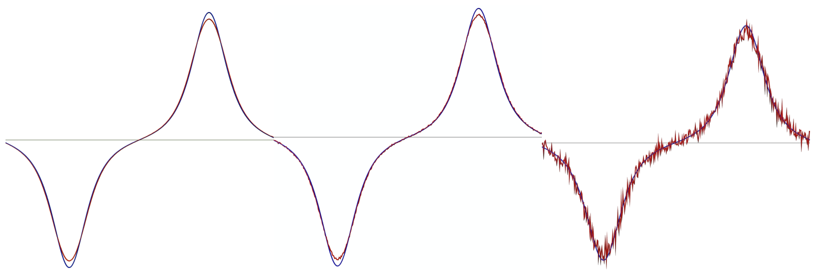

(a) $n_g = 10^{-5}, r = 0.05$ (b) $n_g = 10^{-4}, r = 0.05$ (c) $n_g = 10^{-3}, r = 0.05$

Figure 50: Curvature from arc invariants: Results for a sinusoidal curve at various noise levels n_g with fixed radius. Reference curvature=blue, estimated curvature=red

For figure 50 we added Gaussian noise to our test data. In terms of robustness this method is superior to the methods described in the last sections.

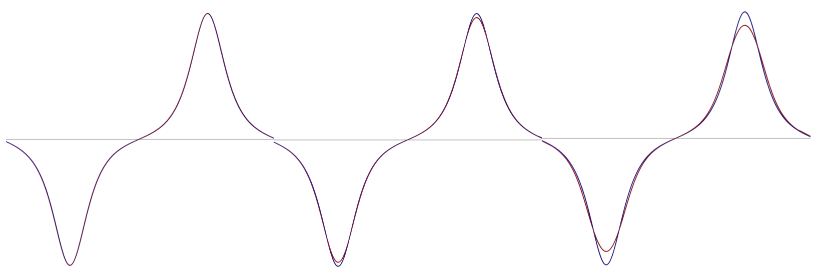


Figure 51: Curvature from area invariants: Influence of scale on the resulting curvature. Reference curvature=blue, estimated curvature=red

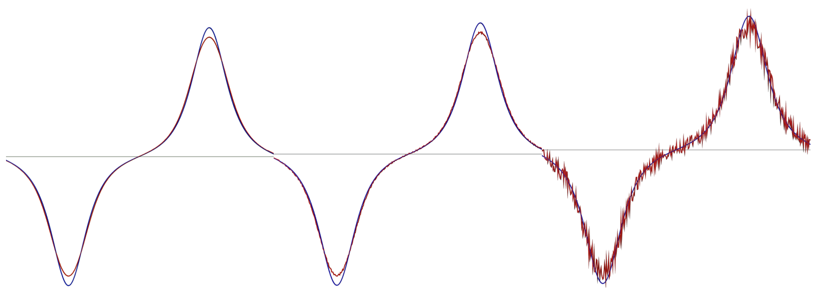
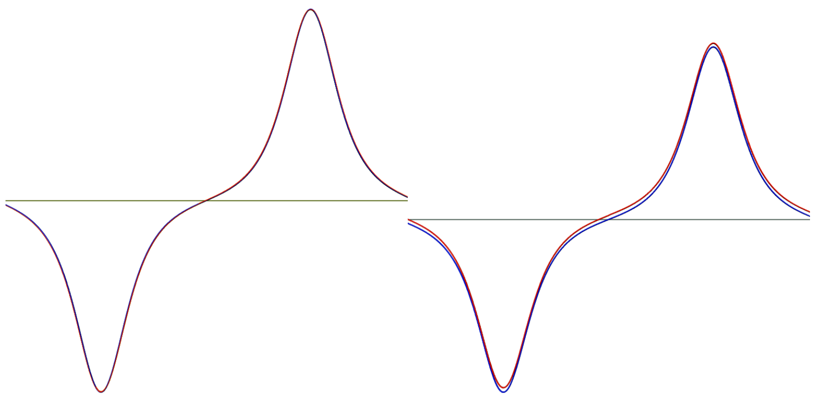


Figure 52: Curvature from area invariants: Results for a sinusoidal curve at various noise levels n_g with fixed radius. Reference curvature=blue, estimated curvature=red

Results for the area invariant are shown in figure 51. We again see the influence of the spatial scale on the estimated curvature values, but compared to the arc invariant the smoothing effect here is lower. This shows that the area invariant describes the local curve shape more accurately, incorporating more data into the estimate. Figure 52 shows results for the noisy data set.

The area of the intersected domain cannot be calculated exactly for discrete data, which results in an inherent error in the estimated curvature values. Figure 53 visualizes the effect of this error on the curvature profile. The outline of the intersected domain is sampled with varying density. Then the area is calculated from the resulting polygon. The error increases as the sampling density gets lower, which can be seen in 53b and 53c. Since the area of the obtained polygon is always smaller than the true area of the domain, the error shows as a


(a) $r = 0.02$, $s = 2$. With the precise area calculation the curvature is pretty accurate.

(b) $r = 0.02$, $s = \frac{1}{5}$. Here the error gets bigger, as the area is calculated less precisely.

(c) $r = 0.02$, $s = \frac{1}{8}$

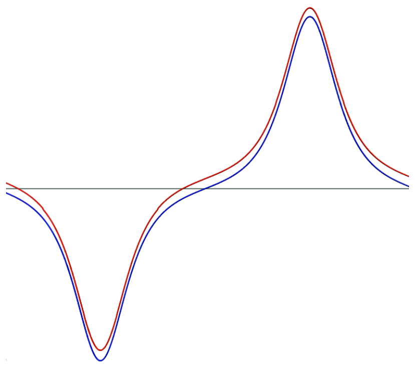
Figure 53: Curvature from area invariants: Influence of the domain area calculation error on the resulting curvature. The sample rate s of the constructed polygon is measured in samples/degree. Reference curvature=blue, estimated curvature=red

constant curvature offset.

Integral invariants are a very elegant framework for estimating curvature from discrete data at various scales. We prefer the area invariant over the arc invariant, since it incorporates more data of the captured local geometry into the estimate, and thus obtains a more accurate amplitude even for higher spatial radii. Drawbacks are the more involved computation and the need for an approximation to the true domain intersection area.

4.4.7 *Curvature estimation using line integrals*

Another method for discrete curvature estimation is the one presented by Lin et al. in [26]. Similar to integral invariants it makes use of a ball neighborhood and integration inside, but the integrals are approximated instead of being actually carried out, resulting in



constant complexity. It is thus more similar to the arc invariant than to the area invariant. The method makes use of principal component analysis to determine a normalized frame for the estimation.

We look at an arc-length parametrized two dimensional curve $\mathbf{c}(s)$. To calculate the curvature at a curve point $\mathbf{p}(s_0)$ a local coordinate system is used for analysis. In this coordinate system the curve's tangent $\mathbf{t}(s_0)$ and normal $\mathbf{n}(s_0)$ coincide with the X and Y axis. In this frame the curve can be seen as a graph $g(x)$ and its Taylor expansion at $x = 0$ written as

$$g(x) = g(0) + xg'(0) + \frac{x^2}{2}g''(0) + O(x^3).$$

Since $g''(0)$ equals the curvature κ (arc-length parametrization), this simplifies to the parabolic equation

$$g_a(x) = \frac{\kappa}{2}x^2,$$

a common term to locally approximate a curve using its curvature.

A line integral is an integral over some function $f(x, y)$ along a path D.

$$I(f) = \int_D f(x, y)\, dl$$

In this setting the path D is given as

$$D = \left\{(x, y)|x^2 + y^2 = r^2, y \geqslant g(x)\right\},$$

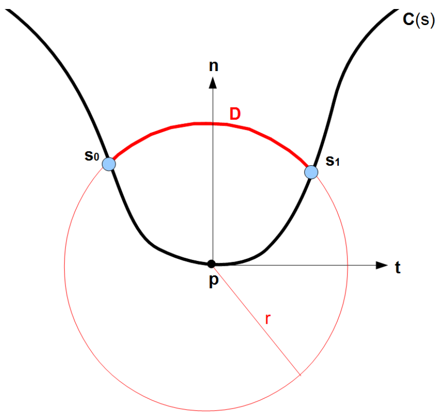


Figure 54: The local coordinate system at $\mathbf{p}$ uses the curve's tangent $\mathbf{t}$ and normal $\mathbf{n}$. The line integrals are evaluated along the arc D.

which describes the part of the circle B centered at $\mathbf{p}(s_0)$ which lies above the curve $g(x)$ (see figure 54).

Using the local curve approximation $g_a(x)$ this line integral can roughly be approximated by

$$I(f) \approx \hat{I}(f) = \int_{B_+} f(x,y)\,dl - \int_0^{\frac{1}{2}\kappa r^2} f(r,y)\,dy - \int_0^{\frac{1}{2}\kappa r^2} f(-r,y)\,dy. \quad (28)$$

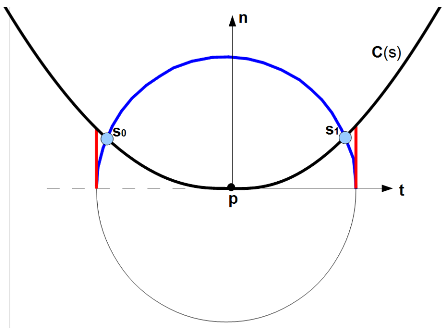


Figure 55: The line integral over D is approximated by integrating over the half circle (blue), and then subtracting the integrals over the two line segments (red).

The first integral represents the upper half B_+ of the circle B. The two integrals that are subtracted represent the parts of the circle between $y = 0$ and $g(x)$. They are approximated by straight vertical line segments at $x = \pm r$, reaching from the x-axis to $g(x)$ (see figure 55).

In the 2D case the covariance matrix of the path D can be written as

$$\hat{\Sigma}(D) = \begin{bmatrix} I(x^2) & I(xy) \\ I(xy) & I(y^2) \end{bmatrix} - \frac{1}{l(D)} \begin{bmatrix} I^2(x) & I(x)I(y) \\ I(x)I(y) & I^2(y) \end{bmatrix}, \quad (29)$$

$l(D)$ being the length of the path. In this normalized setting the covariance matrix is a diagonal matrix, and both $I(x)$ and $I(xy)$ evaluate to 0. We are interested in the first pivot element of the covariance matrix, which gives

$$I(x^2) = \frac{\pi}{2} r^3 - \kappa r^4.$$

From this follows an estimate for curvature.

$$\Sigma_{1,1} \approx \frac{\pi}{2} r^3 - \kappa r^4 \to \kappa \approx \frac{\pi}{2r} - \frac{\Sigma_{1,1}}{r^4}. \quad (30)$$

In order to obtain this information in any coordinate system, Lin et al. make use of PCA. Figure 56 visualizes the angles ϕ_0 and ϕ_1, which are the rotation angles of the ball intersections $\mathbf{s_0}$ and $\mathbf{s_1}$ relative to the

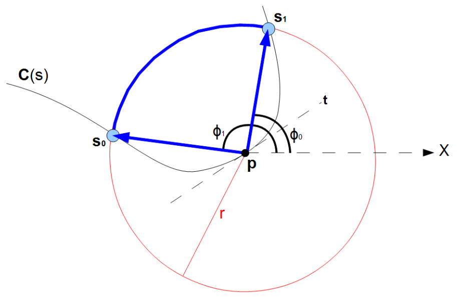


Figure 56: Curvature calculation: the line integrals can be parameterized by the angles ϕ_0 and ϕ_1. The coordinate system does not matter, PCA is used to retrieve it.

normalized coordinate system. Lin et al. express the line integrals of the covariance matrix using these two angles as follows.

$$I(x^2) = \frac{r^3}{2}(\phi_1 - \phi_0 + \sin(\phi_1)\cos(\phi_1) - \sin(\phi_0)\cos(\phi_0))$$
$$I(y^2) = \frac{r^3}{2}(\phi_1 - \phi_0 - \sin(\phi_1)\cos(\phi_1) + \sin(\phi_0)\cos(\phi_0))$$
$$I(xy) = \frac{r^3}{2}\left(\sin^2(\phi_1) - \sin^2(\phi_0)\right)$$
$$I(x) = r^2(\sin(\phi_1) - \sin(\phi_0))$$
$$I(y) = -r^2(\cos(\phi_1) - \cos(\phi_0))$$
$$l(D) = r(\phi_1 - \phi_0)$$

By inserting into 29 and carrying out eigenvalue decomposition on the corresponding covariance matrix, one can obtain the normalized coordinate frame that was used earlier. The eigenvectors approximate the curve's tangent and normal at **p**. The eigenvalue corresponding to the tangent eigenvector can then be inserted into expression 30 to obtain the curvature value. For the desired tangent eigenvector **v** the following rule must hold (which can easily be seen when looking at 56).

$$\text{sign}(\overrightarrow{\mathbf{ps_0}} \cdot \mathbf{v}) \neq \text{sign}(\overrightarrow{\mathbf{ps_1}} \cdot \mathbf{v}) \tag{31}$$

The other eigenvector approximates the curve's normal and is orthogonal to the first one.

For the calculation of eigenvalues and eigenvectors the following relations are useful. Let M be a 2x2 matrix.

$$M = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

Then the eigenvalues of M can be calculated as

$$e_0 = \frac{1}{2}\mathrm{tr}(M) + \sqrt{\mathrm{tr}^2(M)\frac{1}{4} - \det(M)},$$
$$e_1 = \frac{1}{2}\mathrm{tr}(M) - \sqrt{\mathrm{tr}^2(M)\frac{1}{4} - \det(M)}.$$

The eigenvectors can be calculated from the eigenvalues and M.

$$\begin{array}{ll} \begin{pmatrix} e_0 - d \\ c \end{pmatrix}, \begin{pmatrix} e_1 - d \\ c \end{pmatrix} & \text{if } c \neq 0 \\ \begin{pmatrix} b \\ e_0 - a \end{pmatrix}, \begin{pmatrix} b \\ e_1 - a \end{pmatrix} & \text{if } b \neq 0 \\ \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix} & \text{if } b = 0, c = 0 \end{array}$$

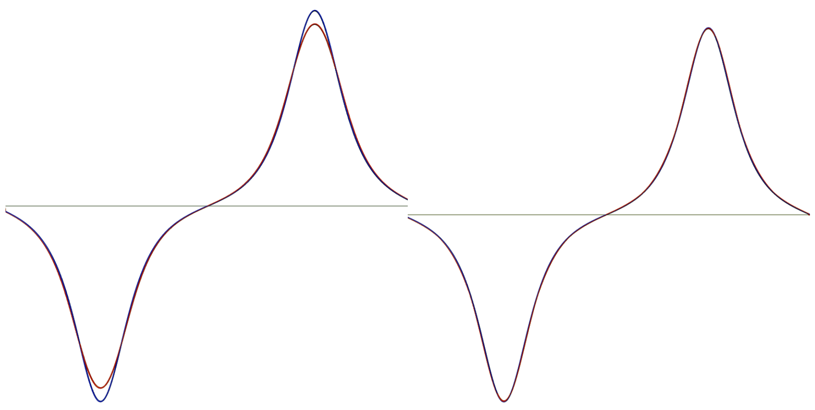

(a) $r = 0.05$ (b) $r = 0.01$

Figure 57: Influence of the radius of estimation. Reference curvature=blue, calculated curvature=red

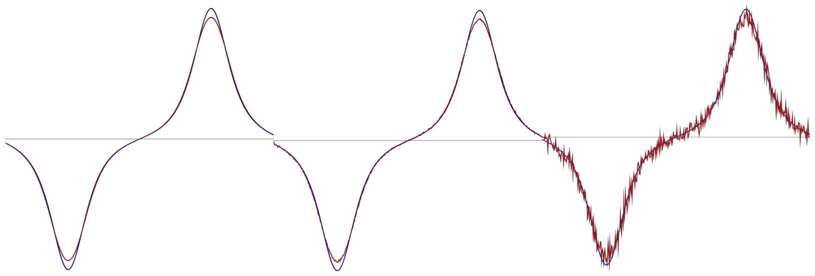

(a) $n_g = 10^{-5}$, $r = 0.05$ (b) $n_g = 10^{-4}$, $r = 0.05$ (c) $n_g = 10^{-3}$, $r = 0.05$

Figure 58: Results for the sine curve at various noise levels n_g with fixed radius. Reference curvature=blue, calculated curvature=red

Figure 57 shows the influence of the neighborhood radius, figure 58 the amount of noise suppression. The method introduces an equal amount of smoothing as the arc integral invariant method, also noise suppression is very similar.

4.5 SMOOTH CURVATURE ESTIMATION BY ADAPTIVE CURVE RESAMPLING

We investigate another method for curvature estimation, which is based on the idea that not all samples need the same amount of data support when the curve is preprocessed. It represents a mixture of the last two sections and involves both curvature estimation and curve smoothing. At the end we obtain a smooth curve and a corresponding curvature profile. This is an advantage, since pure curvature estimation gives us an estimated profile which we cannot validate. After all we do not obtain the true curvature profile of our discrete input data, and we cannot generate the curve that is represented by the estimated curvature profile easily. With a corresponding smooth curve at hand we can at least compare the generated curve to the original samples and see in which way it deviates from them.

In the last sections we analyzed several methods for curve smoothing and curvature estimation regarding their ability to reconstruct the accurate curvature profile of the input data and their robustness against noise. The review of robust scale-based curvature estimators has revealed some important issues. Scale-based curvature estimators obtain an inherent smoothing effect, which already shows at low scales and gets more severe at bigger ones. On the other hand we need a certain scale of estimation in order to cancel out enough noise present in the data. It is very critical to find an optimal spatial scale radius for the estimator in every curve region, too small radii causing noisy estimates (variance of the signal), too big ones damping the signal amplitude (estimator bias) or even mixing up multiple features.

Methods have been suggested to adapt the radius to local curve properties (see for instance [21] or [26]), but we have the impression that it is not easy to find a general solution to this problem for a wide range of scenarios (varying sample densities, noise levels, etc.). Accurate curvature amplitude is not an issue in many applications, but in our case we want to base metric quantities on it. Thus we rely on accurate estimation.

Even if a robust curvature estimator is quite elegant in the sense of not changing the geometry itself, we should ask ourselves if it is a problem to make changes to the geometry in the first place. As mentioned earlier, there exist several sources for flaws (noise, weakly sampled regions, scanning-inaccuracies, etc.) which diminish the trust in our input samples to some degree.

Furthermore, since we want to place limit points onto the discrete curve, it might be wiser to fit a smooth and balanced version of the input data.

For an alternative approach we first project the input data onto a MLS-curve (section 4.3.2.3). For this we use a rather small spatial scale (representing a multiple of the local point spacing) of $f_s = 5$ in order to stay true to the original data. We thus on average incorporate 11 samples into our curve estimate, which smooths down a substantial amount of noise, but is small enough not to let the MLS-curve deviate too much from data with less noise. The sharpness parameter is set to a rather small value of $\sigma_n = 0.2$ in order to make the MLS estimate adapt more closely to the shape of the input samples. There is the option to add more samples in weakly sampled regions before continuing, an advantage of the MLS-definition of our initial curve estimate. This way we can also mitigate the effect of missing regions.

An upper bound for the noise level d_n is estimated by keeping track of the maximum MLS projection distance, which is the distance of a noisy curve sample $\mathbf{p}_i$ to its least squares solution $MLS(\mathbf{p}_i)$.

$$d_n = \max_i MLS(\mathbf{p}_i)$$

Then a curvature profile for the MLS-curve is computed using one of the robust estimators described in the last sections (the area integral invariant in the current implementation), adapting the radius of estimation r_κ to either the estimated noise level d_n or to the average point spacing d_{avg}.

$$r_\kappa = f_\kappa \cdot \max(d_n, d_{avg})$$

The curvature scale is set to $f_\kappa = 10$. Since we only need a first rough estimate of the true curvature, it does not matter if the generated profile gets a little oversmoothed or still obtains moderate noise.

Using this first - and possibly still noise-afflicted - curvature estimate, we resample the MLS-curve with the curvature-guided approach of section 4.3.3 and obtain a feature-adapted sample distribution. The falloff value (rate of change) is set to a rather strict value of $\beta = 0.1$ (10 percent) in order to obtain smoothly changing samples. This also alleviates potential noise on the guidance function. The minimum and maximum sample distances produced by the resampling algorithm are furthermore limited to multiples of the average point spacing d_{avg}, in our case $s_{min} = 0.7 \cdot d_{avg}$ and $s_{max} = 20 \cdot d_{avg}$. A minimum sample density lower than the average sample distance of the original curve allows us to push more samples into high-curvature regions. The point density parameter γ is chosen as a design parameter.

We evaluated our default parameters for various sample densities and noise levels and they proved to work well for our test data.

What did we achieve with this resampling? On one hand we lowered variation in low-curvature regions, which should already reduce a certain amount of noise. On the other hand we pushed data support from feature-poor regions into feature-rich regions of the curve.

Having achieved better support for strong features, we apply n_i iterations of centroid preserving smoothing (section 4.3.2.2) to the the resampled curve. Here the density parameter γ of the resampling algorithm comes into play. As γ decreases, samples first vanish in feature-less regions. The smoothing step will afflict this regions more aggressively, since they have lower data support. By tuning the density of the resampling process we are able to vary between a smoother curve with less variation (low γ), and a curve which is more faithful to the original data (high γ).

We finally estimate the resulting curvature profile using one of the DDG-based methods, which yield more accurate curvature amplitude for smooth data.

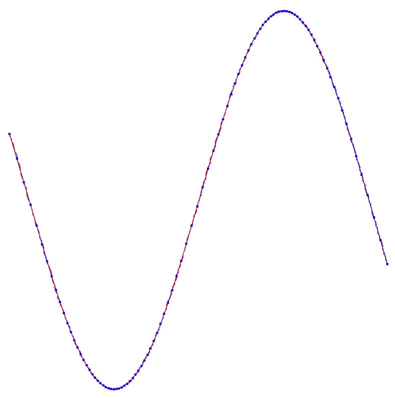
For the examples below we used a fixed amount of smoothing iterations $n_i = 30$.

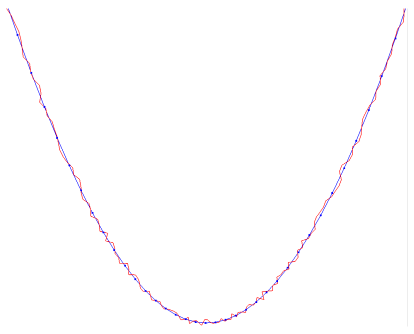
We will first inspect results for our sine curve. Figure 59 shows that even under heavy noise the original curvature can be retrieved. In 59a we see the curvature-adapted point distribution. The samples are moved from low-curvature regions into high-curvature regions, just as expected.

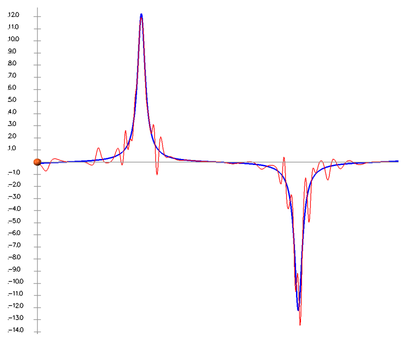
Figure 60 shows results for a B-spline curve with known curvature profile. This is not an easy setup, since there are many rapid high-curvature regions around the S-part. No noise has been added in order to show the basic accuracy of the method. In 60b we see that the algorithm accurately reproduces the ground truth curvature. In 60c γ has been reduced and the curvature profile now obtains much fewer oscillations while maintaining the overall shape and amplitude.

Figure 61 shows results for the same spline curve, but a substantial amount of noise has been added. Even though details get distorted a little, the overall profile stays faithful to the ground truth. The amplitude is successfully maintained as well.

One drawback of the method is that it depends on several parameters. On the other hand it achieves remarkably stable curvature ampli-


(a) Input data (red), result curve (blue), $\gamma = 25$


(b) Input data (red), result curve (blue), $\gamma = 25$


(c) Input curvature (blue), result curvature (red), $\gamma = 230$

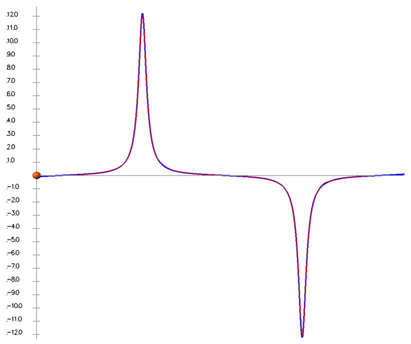
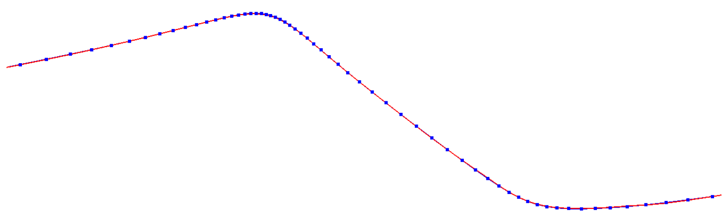

(d) Detail: Input curvature (blue), result curvature (red), $\gamma = 25$

Figure 59: Robust curvature retrieval for artificial data with known ground truth.

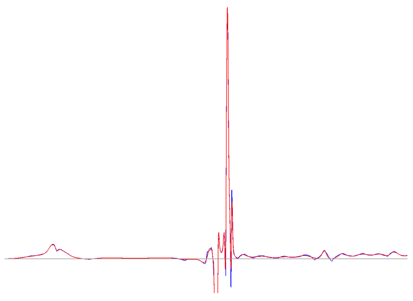
tudes even at substantial noise levels. The most important parameters to tune are the resampling density γ and the number of smoothing iterations n_i. Even though we adapt the other parameters to both the noise level and the point density of the input data, a certain dependence of the results on these factors remains.

It is suggested to use parameter sets for different applications. Once a set of parameters has been chosen, possibly with visual feedback, it is assumed that this parameter set will achieve good results for a certain 3D scan or even a certain domain of models. For our metric 3D scans of car parts we are using rather conservative values of $\gamma = 150$ and $n_i = 50$, which proved to yield smooth curvatures and at the same time preserved important features.

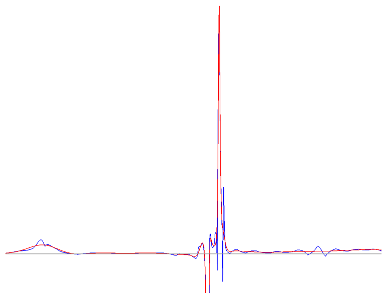
The inherent loss of information that comes with this step of preprocessing might cause additional concern, but since we reduce information in feature-less regions first this is acceptable to us.



(a) Detail: Input data (red), result curve (blue), $\gamma = 20$

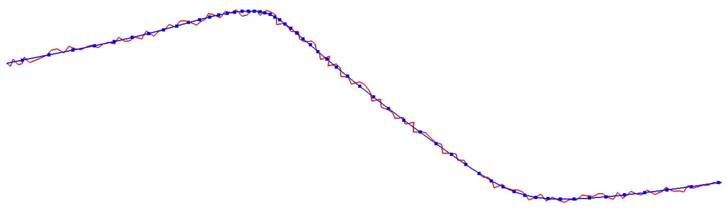


(b) Input curvature (blue), result curvature (red), $\gamma = 400$

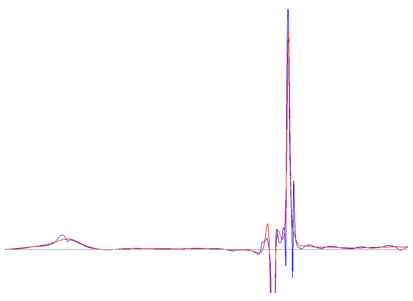


(c) Input curvature (blue), result curvature (red), $\gamma = 20$

Figure 60: Robust curvature retrieval for artificial data with known ground truth.



(a) Detail: Input data (red), result curve (blue), $\gamma = 20$



(b) Input curvature (blue), result curvature (red), $\gamma = 20$

Figure 61: Robust curvature retrieval for artificial data with known ground truth. Gaussian noise added.

4.6 CHOOSING THE RIGHT PREPROCESSING METHOD

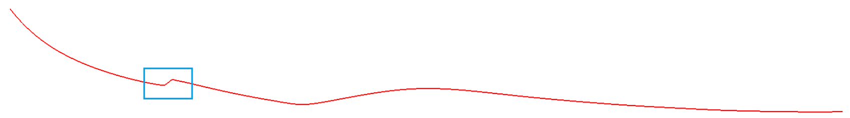
After investigating various approaches to the problems of curve beautification and curvature estimation, we are still concerned about this initial stages, since the removal of unneeded features (noise, oscillations, etc.) requires a certain definition of »important feature« and »unneeded feature«. We will elaborate on this in the results chapter of this work.

We make use of the resampling-based approach of section 4.5 though, for reasons stated below.

- It yields results which are faithful to the ground truth curvature even under substantial noise.
- It is stable at high curvature and preserves curvature amplitude.
- It yields a smooth curvature profile we can measure with.
- It yields a smooth intermediate shape for our limit positions.
- The possibility to get a matching smooth curve and curvature profile.

Varying sampling densities are no problem for us, since our algorithm interpolates between provided curvature values and more information is provided in important feature-rich regions.

We decide against robust curvature estimators, because they already show a bias at lower scales and yield a curvature profile which geometrically does not fully correspond to the curve that generated it.


(a) Input curve.

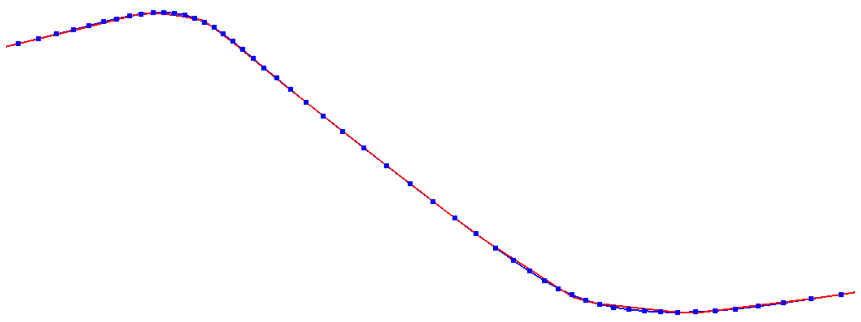
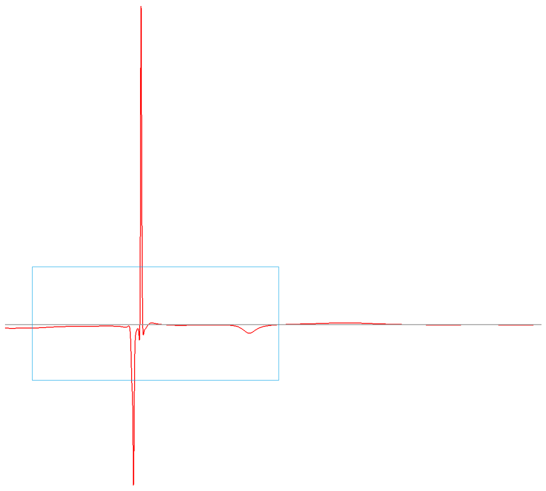
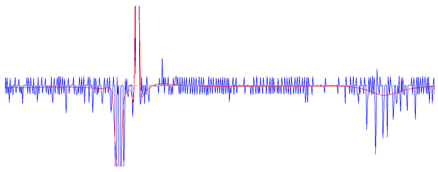

(b) Detail S-part from a). Input curve (red), smoothed curve (blue)

Figure 62: Preprocessing result, curve.

Figure 62a shows a curve sampled from the scan of a car door. Figure 62b compares this input curve to the preprocessed curve in the S-part. We see that the smooth curve follows the input data quite accurately, but does not obtain the strong dents of the original curve


(a) Smooth curvature profile of preprocessed curve.


(b) Detail from a). Input curvature (blue), smooth curvature (red)

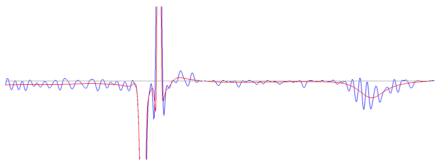

(c) Detail from a). Slightly smoothed input curvature (blue), smooth curvature (red)

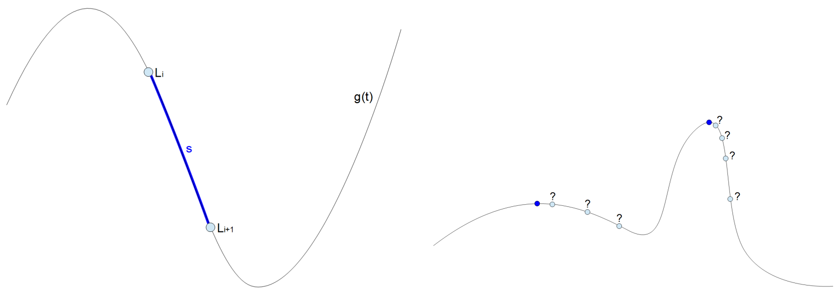
Figure 63: Preprocessing result, curvature.

(which stem from missing regions in the scan).

Figure 63 shows results for the curvature of the same test data. In figure 63a we compare the obtained smooth curvature profile to the raw curvature of the input data. The input curvature is distorted by noise, so a true comparison is not easy. In 63b we applied a small amount of centroid preserving smoothing (50 iterations). This way we obtain a curvature trend which is easier to compare. As we can see, our result curvature approximates the oscillations in a smooth and expected way.

4.7 THE REQUIRED MESH WIDTH: A LOCAL MEASURE FOR LIMIT POINT SPACING

The key principle the presented B-spline fitting technique is based on, is the spacing of B-spline limit points along a discrete curve to obtain a satisfying data fit. For a user the manipulation of limit points is very intuitive and they can be parameterized conveniently by a vector of (normalized) arc lengths. But where exactly to place the limit positions?



(a) Mesh width between two limit points.

(b) Starting from the blue limit points, where to put the next ones?

Figure 64: Motivation

A closely related question that this section will try to answer is: *If one limit point was chosen on the curve, where to put the next one*? We could also ask for the adequate local mesh width of the limit point distribution at some curve position $\mathbf{p}(t)$, the mesh width between two limit points being measured in arc length along the curve (see figure 64a). In order to stay true to the common nomenclature, we will talk of *mesh widths*, even though we actually treat polygons rather than limit meshes. Figure 64b motivates this question by showing two quite different limit point starting positions and possible positions for their neighbors. It is by intuition that one would assign a broader mesh width to the left limit point than to the right one, since more limit points will surely be needed in high-curvature regions. This already gives a hint that the limit point distribution should somehow be related to local curvature. Further, it is also desired to realize as broad mesh widths as possible in order to obtain sparse limit point distributions. This reduces the complexity of the produced geometry and prevents unnecessary waves on it. We are interested in the *maximum limit point mesh width* that can be realized in a certain curve region. But how can we quantify this local property?

In order to give an answer to this question, we locally approximate the curve region around a curve position $\mathbf{p}(t)$ by the circle of curvature at $\mathbf{p}(t)$. Since it is an important quality aspect of the final limit distribution (section 4.1), we assume that the limit points are regular, and thus that they so not vary to rapidly in a certain curve region. We place four equally spaced limit points on this circle of radius $r = 1/\kappa$, such that a uniform cubic B-spline segment $\mathbf{g}(t)$ is uniquely defined. We are interested in the maximum distance error of this B-spline segment to the circular curve approximant.

The spline segment will not fully resemble our circular approximation, a fact that will be discussed in section 4.8. Assuming the spline is parameterized between 0 and 1, its maximum deviation e_{tol} from the circle can easily be retrieved as

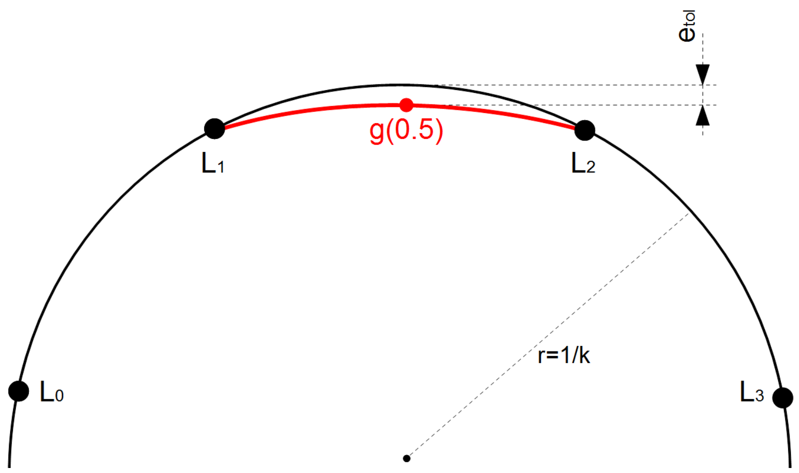
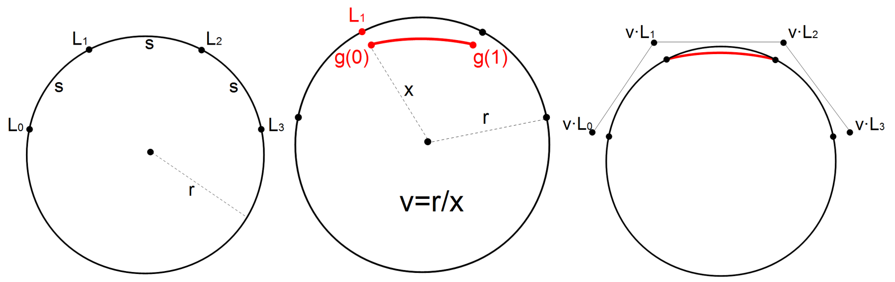


Figure 65: Construction of a cubic uniform B-spline segment on a circle. The distance e_{tol} measures the maximum deviation of the B-spline segment from the circle.

$$e_{tol} = r - \|\mathbf{g}(0.5)\|.$$

This basic construction is depicted in figure 65.

The maximum deviation e_{tol} is now fixed at some user defined value. What is the maximum arc-length s_{tol} the limit points can be spaced along the circle, without causing the error between spline and circle to exceed e_{tol}?



(a) Limit points are sampled along the circle at distances s.

(b) The limit points are used as control points and yield a B-spline segment $\mathbf{g}_l(t)$.

(c) The limit points are scaled to retrieve the control points, such that the spline endpoints equal L_1 and L_2.

Figure 66: Obtaining the control points from limit points sampled along a circle.

In order to answer this, we first need to parameterize the error $e_{cur}(s)$ in a convenient way. We will make use of the fraction α of a full circle instead of the arc-length s, in order to describe our spacing along the circle. This also frees the resulting terms of specific units.

$\alpha = \frac{1}{3}$ means that we cover the whole circle with four limit points and will be the maximum spacing. The value α thus lies in the range $[0, \frac{1}{3}]$. An arc-length s can always be retrieved from α via $s = 2\pi r\alpha$.

Let us imagine a circle at the origin of the coordinate system and limit points placed along it. We directly use the limit points $\mathbf{L}$ along the circle as a control polygon for the B-spline curve $\mathbf{g}_l(t)$. The end-points of $\mathbf{g}(t)$ are supposed to lie on the circle, but the ones of $\mathbf{g}_l(t)$ deviate from this position by $r - ||\mathbf{g}_l(0)||$. We can use the difference in scale to correct for this deviation and to obtain the true control polygon $\mathbf{C}$ as

$$C_i = \frac{r}{||\mathbf{g}_l(0)||} \cdot L_i.$$

The steps to obtain the control points from a given limit configuration are depicted in figure 66.

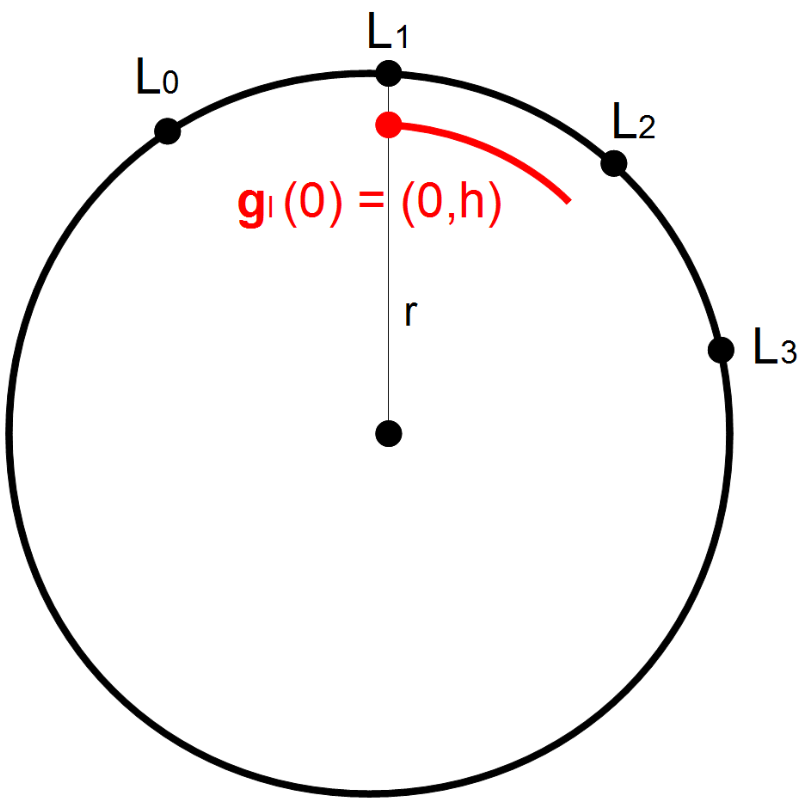


Figure 67: Limit point distribution for control point scale retrieval.

In order to express $||\mathbf{g}_l(0)||$ in a more convenient way using α, let us relocate the limit points such that L_1 is moved to $(0, r)$ (see figure 67). The B-spline segment endpoint $\mathbf{g}_l(0)$ now lies somewhere on the y-axis at $(0, h)$. By using the equation for a uniform cubic B-spline with four control points (equation 4) and the circle fraction α, we can express $||\mathbf{g}_l(0)||$ as

$$||\mathbf{g}_l(0)|| = h = \frac{1}{6}r\cos(2\pi\alpha) + \frac{2}{3}r\cos(0) + \frac{1}{6}r\cos(-2\pi\alpha) = \frac{2}{3}r + \frac{1}{3}r\cos(2\pi\alpha).$$

Having obtained the control points, we can also express the point of maximum deviation on $\mathbf{g}(t)$ using α. For this we will switch back to the limit point distribution shown in figure 65, such that $\mathbf{g}(0.5)$ lies on the y-axis at $(0, k)$. We can rewrite $||\mathbf{g}(0.5)||$ as

$$\begin{aligned} ||\mathbf{g}(0.5)|| = k &= \frac{1}{48}r_2\cos(3\pi\alpha) + \frac{23}{48}r_2\cos(\pi\alpha) + \frac{23}{48}r_2\cos(-\pi\alpha) + \frac{1}{48}r_2\cos(-3\pi\alpha) \\ &= \frac{r_2}{24}\cos(3\pi\alpha) + \frac{23}{24}r_2\cos(\pi\alpha), \end{aligned}$$

where $r_2 = \frac{r^2}{h}$.

To obtain the current error $e_{cur}(r, \alpha)$ we can now use the two retrieved terms.

$$e_{cur}(r, \alpha) = r - \frac{\frac{r}{24}\cos(3\pi\alpha) + \frac{23}{24}r\cos(\pi\alpha)}{\frac{2}{3} + \frac{1}{3}\cos(2\pi\alpha)}$$

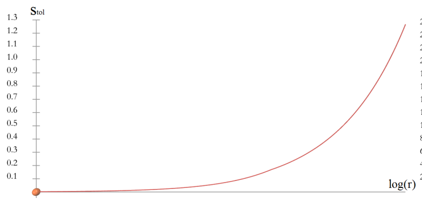
The function $e_{cur}(r, \alpha)$ is non-linear and cumbersome to invert to $\alpha(r, e_{cur})$. We thus formulate an optimization problem with the following functional to be minimized.

$$f(\alpha) = |e_{tol} - e_{cur}(r, \alpha)| \qquad \alpha \in \left[0, \frac{1}{3}\right]$$

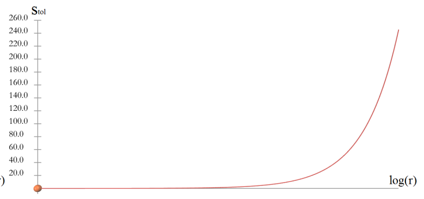
It converges fast and reliable with simple optimization techniques, since there is only a single global minimum in the given range. If e_{tol} cannot be realized on the given circle, the reachable minimum will be at $\alpha = \frac{1}{3}$, as desired.

The spacing s_{tol} obtained by this optimization is used to approximate the local mesh width that is required at a certain curve sample $\mathbf{p}(t)$, *required* meaning that it must not be exceeded by limit point mesh widths in this curve region. Equally, it measures how far from this sample the next limit position can be spaced along the curve at maximum. It will from now on be referred to as the curve position's *required mesh with*. The required mesh widths depend on the local curvature and the error e_{tol}, which the user may vary to influence the spacing and thus the density of the limit positions.

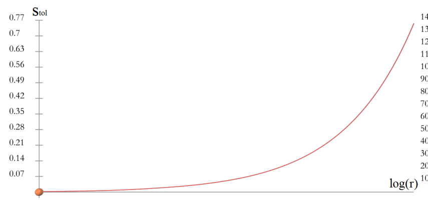
It is clear that the circle of curvature soon becomes a very inaccurate estimate for the curve as one moves away too far from the position $\mathbf{p}(t)$, but in terms of *local* curvature behavior it is at least an estimate that enforces closer mesh widths.



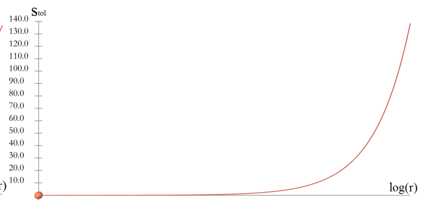
(a) $e_{tol} = 10^{-2}$, $r \in [0.001, 1]$



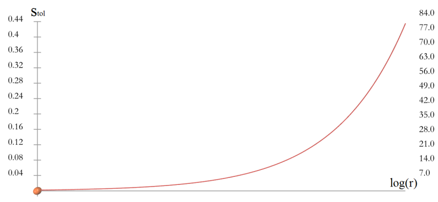
(b) $e_{tol} = 10^{-2}$, $r \in [0.001, 1000]$



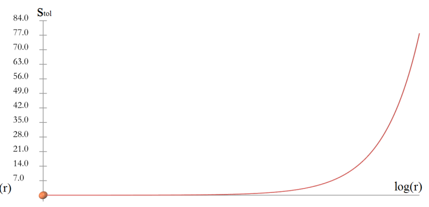
(c) $e_{tol} = 10^{-3}$, $r \in [0.001, 1]$



(d) $e_{tol} = 10^{-3}$, $r \in [0.001, 1000]$



(e) $e_{tol} = 10^{-4}$, $r \in [0.001, 1]$



(f) $e_{tol} = 10^{-4}$, $r \in [0.001, 1000]$

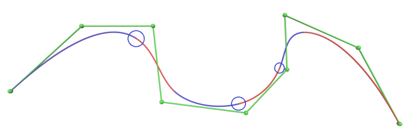
Figure 68: Required mesh width charts. Required mesh width plotted over $\log(r)$ at various error bounds.

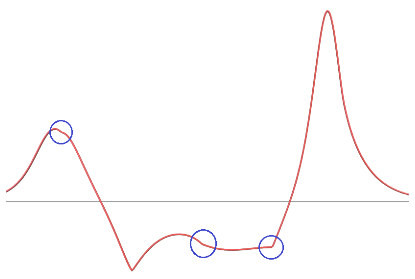
Figure 68 shows the required mesh width plotted over the radius of curvature r for various error bounds e_{tol} and parameter ranges. One can see that the retrieved mesh widths vary pretty similar on all error levels, but with different absolute scale. Note that the scale itself does not vary linearly with the error.

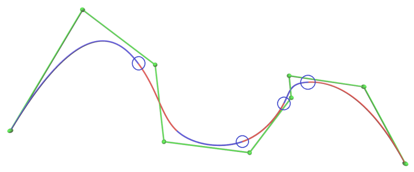
4.8 CONNECTION TO SUBDIVISION ARTIFACT ANALYSIS

The results obtained from this empirical approach can also be related to the theoretical results of Augsdörfer et al. in [2]. For this let us shortly discuss a phenomenon which is inherent to B-splines.

When modifying the control points of a uniform cubic B-spline curve, one can often identify irregularities in the associated curvature profile. These irregularities show as strange bends that are not strongly visible at one time, and suddenly very present at another, which is shown in figure 69. This behavior is very unpredictable and thus not very pleasing for high-quality design. It calls for further analysis.


(a) A uniform cubic B-spline curve.


(b) Curvature profile: There are several parts of the curve that yield unpleasant bends.


(c) The same curve slightly manipulated at some control points.

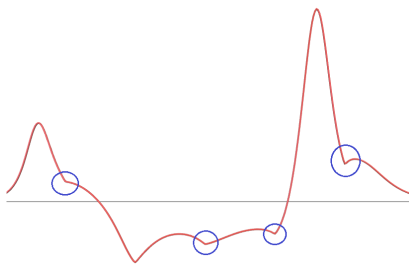
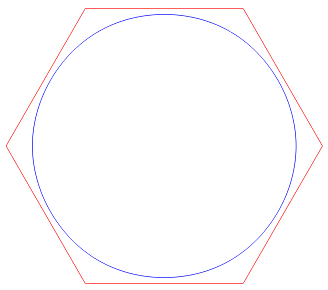

(d) Curvature profile: When modifying the curve, new irregularities are suddenly introduced.

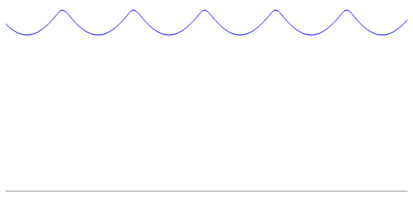
Figure 69: Irregularities in the curvature profile of a B-spline curve.

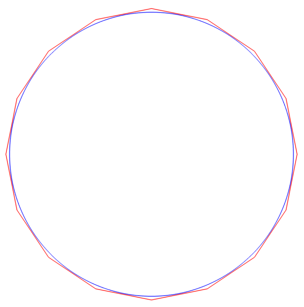
In order to visualize the phenomenon in a more standardized setting, a closed B-spline curve can be investigated whose control polygon consists of equally spaced points sampled from a circle. What kind of shape does such a control polygon yield? Intuitively the answer would be a circle, but this is not the case. This can easily be seen when looking at the curvature profile in figure 70. What should be a straight horizontal line consists of many ripples with equal amplitude, which are the mysterious »bends« mentioned earlier. The curvature maxima manifest at the segment borders, thus inside a segment the curvature profile »sags«. The figure also shows empirically that the amplitude of the ripples varies inversely proportional with the sampling density. The higher sampling density in 70c results in smaller ripples and thus the curve resembles a circle much more closely.

In [2] Augsdörfer et al. investigate this behavior from the viewpoint of *subdivision artifact analysis*. *Artifacts* means features of the produced shape that can not be avoided by the designer by movement of control (limit) points. One can imagine that such unpredictable »features« in a shape are counterproductive for high-quality modeling.

Each subdivision process can be represented by a special matrix called subdivision mask. In one step of subdivision the coefficients


(a) Control polygon consisting of circle samples, $n = 6$.


(b) The closed curve defined by this control polygon is not exactly a circle, since its curvature profile is not constant, but instead consists of many ripples.


(c) Control polygon consisting of circle samples, $n = 16$.

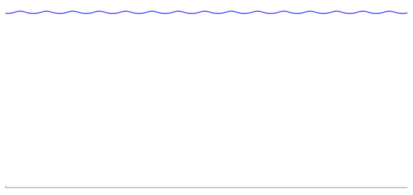

(d) A higher control point sampling density lowers the amplitude of the observed ripples.

Figure 70: B-spline curves do not reproduce circles.

of this mask are used to produce new vertices as a linear combination of old ones. By splitting up this mask, it can be shown that each subdivision scheme consists of a sampling stage for refinement and a subsequent filter stage that smoothes out the refined samples.

The filter stage can be further split up into a convolution of a number of smoothing matrices and a so called kernel. The number of smoothing matrices that can be extracted, corresponds to the number of $(1 + z)$ factors in the z-transform of the mask. The kernel is what remains after sorting out the smoothing parts.

The mask for our Catmull/Clark curve subdivision scheme takes the form:

$$\mathbf{M} = \frac{1}{8}\begin{pmatrix} 1 & 4 & 6 & 4 & 1 \end{pmatrix}$$

It is easy to spot the two subdivision rules introduced earlier. Imagine three vertices. New points with zero values are inserted on each edge, then the mask is applied to all positions (which is done by centering the mask on a position and using the factors for a linear combination). At the old vertex positions the inserted zero positions cancel out the factors with value 4, yielding the vertex rule

$$\frac{1}{8}\begin{pmatrix} 1 & 6 & 1 \end{pmatrix}.$$

At the newly inserted zero positions all factors but those with value 4 are canceled out, which yields the edge rule

$$\frac{1}{2}\begin{pmatrix} 1 & 1 \end{pmatrix} = \frac{1}{8}\begin{pmatrix} 4 & 4 \end{pmatrix}.$$

The z-transform applied to this mask yields

$$f(z) = (1z^{-2} + 4z^{-1} + 6z^{0} + 4z^{1} + 1z^{2})\frac{1}{8} = 2\frac{z+1}{2}^{4}.$$

This hints at four smoothing stages. The mask can be split up into

$$\mathbf{M} = \frac{1}{2}\begin{pmatrix} 1 & 1 \end{pmatrix} \star \frac{1}{2}\begin{pmatrix} 1 & 1 \end{pmatrix} \star \frac{1}{2}\begin{pmatrix} 1 & 1 \end{pmatrix} \star \frac{1}{2}\begin{pmatrix} 1 & 1 \end{pmatrix} \star 1$$

which reveals the four predicted smoothing terms and a unity kernel.

To sum it up: Each step of subdivision consists of a sampling stage that refines the geometry by adding points, and a number of stages that smooth out the new geometry. Further, the number of smoothing stages is directly related to the degree of the limit curve.

Augsdörfer et al. studied the effects of such subdivision components using control meshes laid out in a grid. They identified two different kinds of artifacts, both originating from the sampling stage.

LONGITUDINAL ARTIFACTS Artifacts associated with the approximating error introduced in the sampling stage. They occur on directions aligned with the grid. For the B-spline curves used in this work only this kind of artifacts is of importance.

LATERAL ARTIFACTS Artifacts that occur when extruding in directions not aligned with the grid. This type of artifacts is not relevant for our curves.

Both kinds of artifacts are smoothed out to a certain extend in the subsequent smoothing stages. Augsdörfer et al. conclude that artifacts are indeed always present, but can be reduced in two ways:

- Using higher degree curves, which means a change of curve type and more inherent complexity (e.g. broader support).
- Using a higher sample rate, which means more complex means of manipulation.

Another option in our opinion might be to »hide« them by making use of clever control polygon construction, but this is yet subject to research.

Augsdörfer et al. most notably also present methods to quantify the artifact errors on the limit curve. In the case of uniform cubic B-spline curves the amplitude of the artifact error can be quantified as

$$\mathrm{err}_a = \sin^4\left(\frac{\pi\omega}{2}\right)\left(1 + 2\sin^2\left(\frac{\pi\omega}{2}\right)\right)\frac{1}{3}, \tag{32}$$

where ω relates to the control point sampling density and is measured in cycles per point. Using this equation we can now quantify the ripples on our circle.

We now can use our results from last section to get expressions for $\|\mathbf{g}_l(0)\|$ and $\|\mathbf{g}_l(0.5)\|$ for $r = 1$ and a certain α. The curve $\mathbf{g}_l(t)$ represents the limit curve of our initial control points sampled along the circle (see figure 66b). The maximum error can then be expressed as

$$e_{cur}(\alpha) = (\|\mathbf{g}_l(0)\| - \|\mathbf{g}_l(0.5)\|) \cdot \frac{1}{2} \tag{33}$$

$$= \left(\frac{2}{3} + \frac{1}{3}\cos(2\pi\alpha) - \frac{1}{24}\cos(3\pi\alpha) - \frac{23}{24}\cos(\pi\alpha)\right) \cdot \frac{1}{2}. \tag{34}$$

Note that the error has been halved in the equation, since for the control point case it is assumed that the error of the generated spline will be distributed around the true signal.

Equation 32 can be rewritten using

$$\sin^4(x) = \frac{1}{8}(3 - 4\cos(2x) + \cos(4x))$$
$$\sin^6(x) = \frac{1}{32}(10 - 15\cos(2x) + 6\cos(4x) - \cos(6x))$$

in order to obtain the same terms.

$$\mathrm{err}_a = \sin^4\left(\frac{\pi\omega}{2}\right)\left(1 + 2\sin^2\left(\frac{\pi\omega}{2}\right)\right)\frac{1}{3} \tag{35}$$

$$= \frac{1}{3}\sin^4\left(\frac{\pi\omega}{2}\right) + \frac{2}{3}\sin^6\left(\frac{\pi\omega}{2}\right) \tag{36}$$

$$= \left(\frac{2}{3} + \frac{1}{3}\cos(2\pi\alpha) - \frac{1}{24}\cos(3\pi\alpha) - \frac{23}{24}\cos(\pi\alpha)\right) \cdot \frac{1}{2} \tag{37}$$

4.9 THE ACCUMULATED MESH WIDTH

With the required mesh width at hand as a measure, let us consider again the question of where to put the neighbors of a limit point that already has been chosen to lie at some curve point $\mathbf{p}$. The curvature $\kappa(\mathbf{p})$ can be used together with the user defined error limit e_{tol} to calculate the local maximum limit mesh width s_{tol}. It would not suffice just to run along the curve for s_{tol} units to find the next limit point position, since this would most likely violate the required mesh widths of other positions. We can imagine a curve position of zero curvature, which would be a free ticket to jump to the curves end, no matter how many strong features lying inbetween.

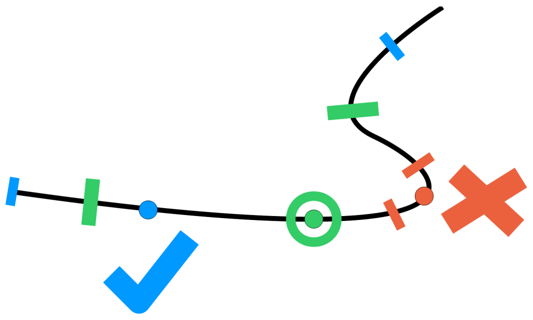


Figure 71: The accumulated mesh width: The required mesh width of the green limit cannot be realized, since the orange limit requires a smaller mesh width. The mesh width of the blue limit would agree with the green limit though.

To prevent this, all mesh width requirements that are passed along the way have to be accounted for when running from one limit position to the speculated next one. This can be achieved by not running farther along the curve than the shortest mesh width that is encountered, leading to the definition of a more restricted mesh width s_{acc} at some arc-length parameter s_0.

$$s_{acc}(s_0) = \max_{s_1 \geqslant s_0} (s_1 - s_0) \quad , \quad s_1 - s_0 \leqslant \min_{s \in [s_0, s_1]} w(s) \tag{38}$$

The function $w(s)$ returns the required mesh width at arc-length s. The length s_{acc} is the »true« local mesh width that can be spent at a certain position on the curve, accounting for all other local mesh widths inside the spanned curve interval. It is named the *accumulated mesh width*. Following the accumulated mesh width ensures that no important feature is ignored - the threshold feature size of course depending on the tolerated error e_{tol} - and that all curve positions are satisfied in terms of their required mesh widths. Figure 71 illustrates the importance of the accumulated mesh width.

By using the sample positions as support points, equation 38 can also be rewritten for a discrete curve setting as

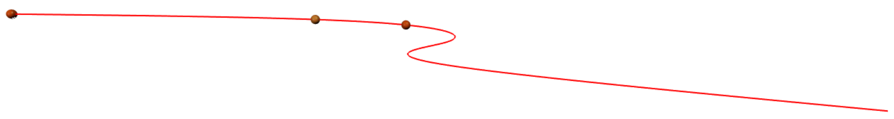
$$s_{acc}(s_0) = \max_{s_1 \geqslant s_0}(s_1 - s_0) \quad , \quad s_1 - s_0 \leqslant \min_{s_i \in [s_0, s_1]} w(s_i), \tag{39}$$

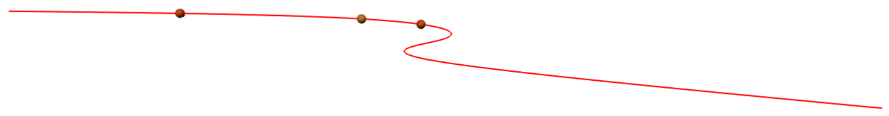
s_i being the arc-length at the i^{th} sample. Should it be desired to use sample positions as limit positions, this equation can be further be rewritten as

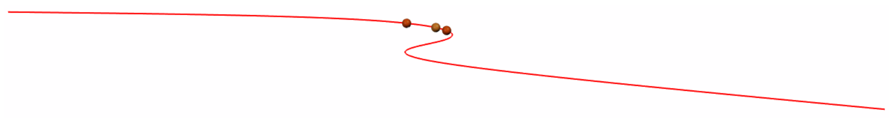
$$s_{acc,i} = \max_{j \geqslant i}(s_j - s_i) \quad , \quad s_j - s_i \leqslant \min_{k \in [i,j]} w(s_k). \tag{40}$$

The accumulated mesh width at a certain curve position can be calculated in forward and backward direction, usually yielding different results $s_{acc,f}$ and $s_{acc,b}$. In order to assign one accumulated mesh width to each sample, we choose the smaller value as the final accumulated mesh width.

$$s_{acc} = \min(s_{acc,f}, s_{acc,b}) \tag{41}$$


(a) Result 1


(b) Result 2


(c) Result 3

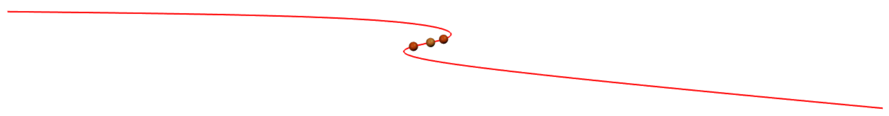

(d) Result 4

Figure 72: Various results for the accumulated mesh width of a cubic B-spline curve. $e_{tol} = 10^{-5}$ units, total curve length $l = 10.0$ units.

Figure 72 shows various results for the accumulated mesh width on a sampled cubic B-spline curve. Equation 40 has been used to retrieve the shown neighbor limits in both curve directions. In 72a the limit is placed in a low curvature region, thus the required mesh width s_{tol} will be quite high. The distance $s_{acc,b}$ will also be high, since the whole region is of low curvature and won't stop the next limit point from running far along the curve. In fact the limit has to be stopped at the end of the curve. The next limit point in forward direction is stopped early by the high curvature S-part. In 72b and 72c the start limit is placed closer to the S-part. $s_{acc,b}$ is getting shorter because s_{tol} shrinks more and more with the higher curvature. $s_{acc,f}$ also shrinks because of s_{tol}, but is further held back by the first peak of the high curvature S-part. Image 72d shows that inside the S-part both accumulated mesh widths are limited by the two high curvature regions.

The behavior on the other side of the curve can be deduced in a similar way.

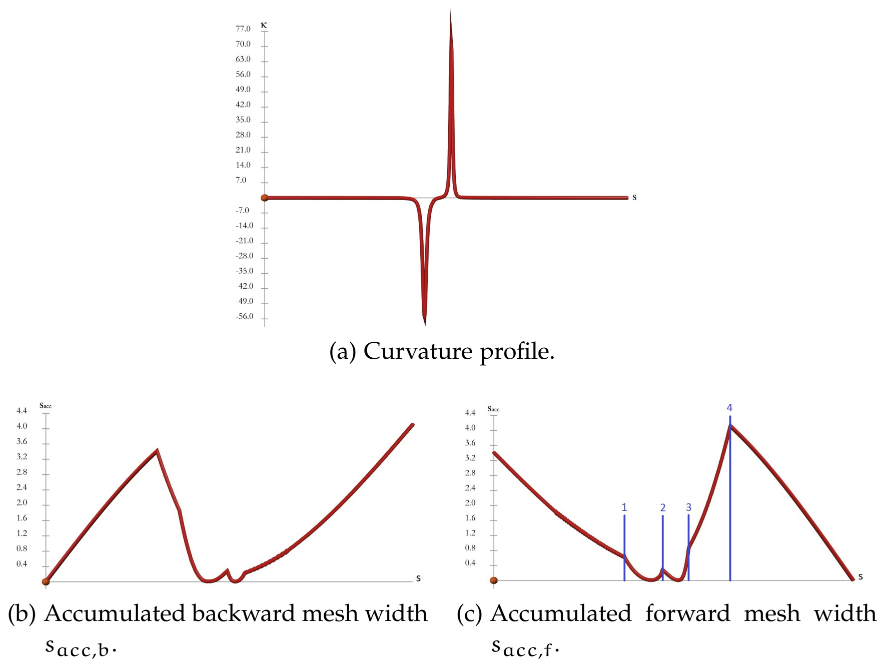
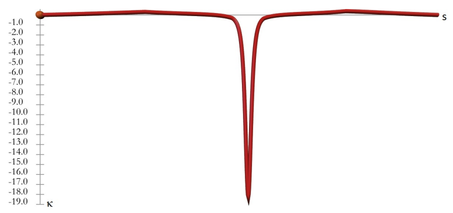


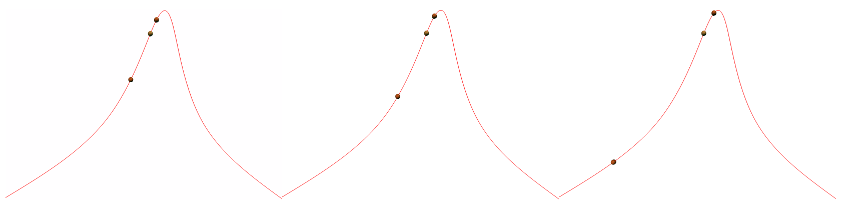
Figure 73: Characteristics for the curve shown in figure 72. $e_{tol} = 10^{-5}$ units, total curve length $l = 10$ units.

Figure 73a shows the curvature profile of the same curve. In image 73b and 73c one can see the accumulated mesh widths plotted against arc-length. Let us consider the accumulated forward mesh width of 73c.

At the start of the curve $s_{acc,f}$ is quite high, but it shrinks as the curvature increases. At position 1 there is a little bend in the graph, which indicates a segment border or a change of influence. $s_{acc,f}$ then shrinks until a minimum is reached near the high curvature peak. Between 1 and 3 $s_{acc,f}$ first grows a little bit, but soon again drops because it is limited by the second high curvature peak. After the second curvature peak the accumulated mesh width grows until it is stopped by the end of the curve at 4.



(a) Curvature profile.



(b) $e_{tol} = 10^{-5}$ (c) $e_{tol} = 10^{-4}$ (d) $e_{tol} = 10^{-2}$

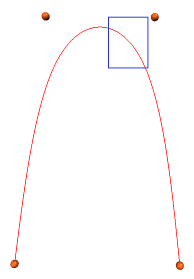
Figure 74: Effect of the error bound e_{tol} on the accumulated mesh width. The accumulated mesh widths grow as the error bound is lowered.

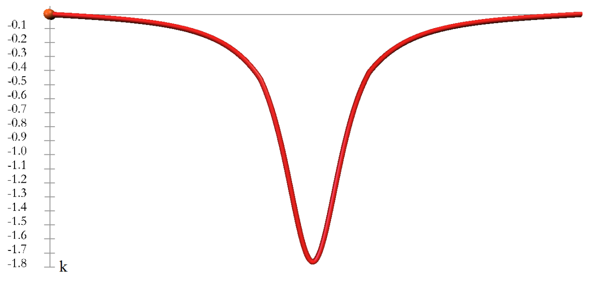
The effect of the error bound e_{tol} on the accumulated mesh width is visualized in figure 74. The curve contains a high curvature peak in its middle, as shown in 74a. All required mesh widths s_{tol} grow with a lowered error bound, which influences both how far limits are suggested to move locally and how strong they are limited by mesh widths of other regions. This results in the accumulated mesh widths growing as the error bound is stepwise weakened in 74b, 74c and 74d.

At last, let us have a look at figure 75. This example shows $s_{acc,f}$ and $s_{acc,b}$ for a position on a uniform cubic B-spline segment with a high curvature peak in its center. The section we are looking at is at the right flank of the spline. All required mesh widths $w(s_i)$ are plotted along the curve at the respective samples $\mathbf{p}(s_i)$. We can see how the required mesh widths limit the accumulated mesh width, in backward direction the first sample with its high curvature, in forward direction the point of interest itself, since curvature declines.

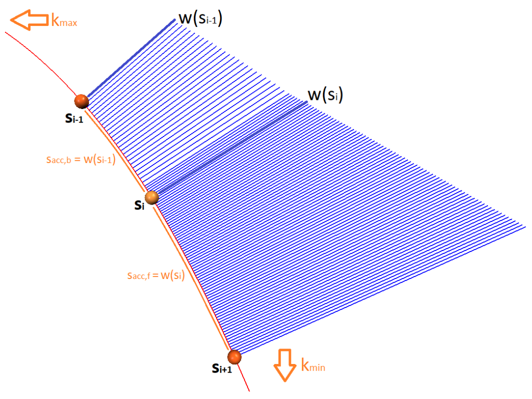
4.10 LIMIT POINT DISTRIBUTION

The accumulated mesh width introduced in the last section only dictates how far to space the next limit points at a certain position of the curve at maximum, but a valid overall limit point distribution has yet to be found. Valid here means that no required mesh width is violated by too large mesh widths of the distribution.


(a) Uniform cubic B-spline.

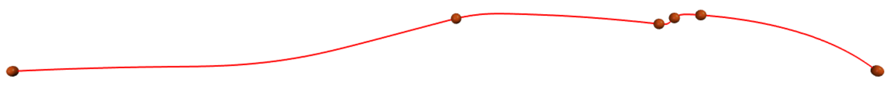


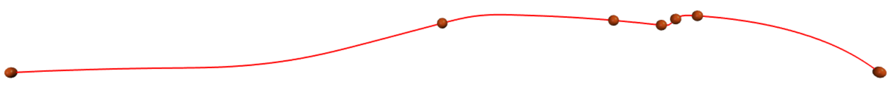
(b) Curvature profile.



(c) Accumulated mesh width in forward and backward direction. The required meshwidth of each curve sample is plotted in normal direction. The two limiting mesh widths are painted in bold blue.

Figure 75: Accumulated mesh width depending on the required mesh width.


(a) $e_{tol} = 10^{-2}$


(b) $e_{tol} = 10^{-3}$

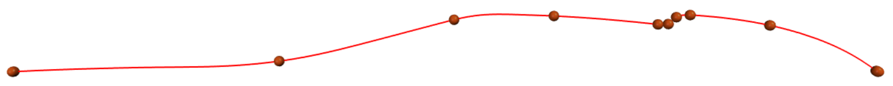

(c) $e_{tol} = 10^{-4}$

Figure 76: Valid limit point distributions for changing error bounds.

A first simple approach that yields such a distribution is described below. Given a curve and an error bound e_{tol}:

1 Calculate the curve's curvature profile.

2 Start at the beginning of the curve and add a limit position $\mathbf{l}_0$ there.

3 At $\mathbf{l}_i$ calculate the accumulated forward mesh width $s_{acc,f}$ as described in the last section, using e_{tol} and the local curvature values.

4 From $\mathbf{l}_i$ run $s_{acc,f}$ units along the curve to obtain the next limit position $\mathbf{l}_{i+1}$.

5 Alternate 3 and 4 until the end of the curve is reached.

6 If needed insert a limit point at the end of the curve.

Such a limit point distribution will always respect the required mesh widths. Note that we could also run from the end of the curve to its beginning, using $s_{acc,b}$.

Figure 76 shows how the error bound influences the resulting distributions. When lowering the error bound new limit points will eventually pop in and the distribution may change substantially.

Looking at figure 76 we observe that the obtained mesh widths vary quite rapidly, but the original construction the measure is derived from relies on four *equally spaced* points and is only valid in a local homogeneous curve region around the investigated position. Of course we can't fully comply to this assumption, since the mesh widths have to vary between each two of these local estimates, but we will try to reconcile the limits a little bit, such that the mesh widths vary more smoothly. This also makes sense regarding our initial requirements for mesh width regularity. For convenience it should also be possible for the user to choose the allowed amount of variation.

For this we again make use of the approach of Baran et al. introduced in section 4.3.3. Instead of relating the local spacing to a fraction of the local circle of curvature, we will directly make use of the required mesh width.

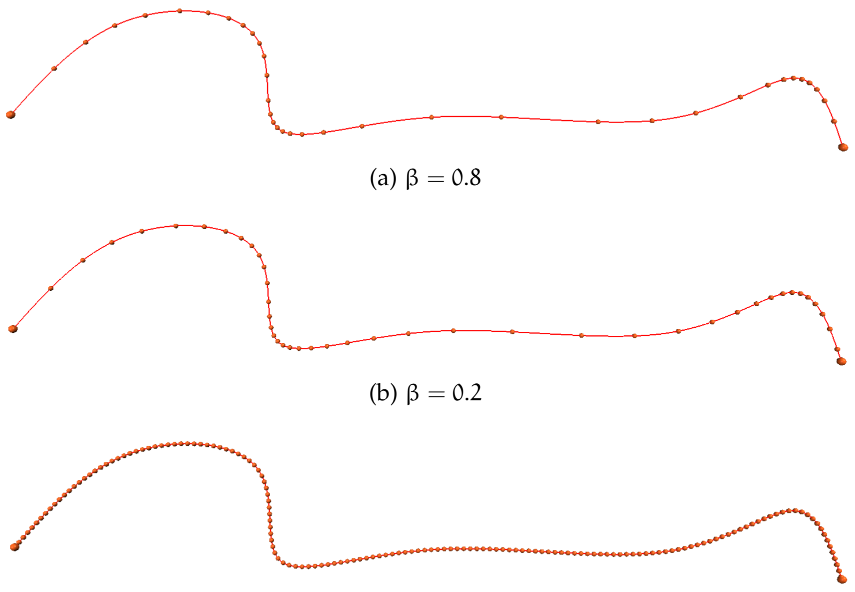
In [3] the resampling function $r(s)$ is considered.

$$r(s) = \min_i \left(|s - s_i| \cdot \beta + \frac{2 \cdot \pi}{\gamma \cdot \kappa_i} \right) \tag{42}$$

We adapt the second term to our needs.

$$r(s) = \min_i \left(|s - s_i| \cdot \beta + mw(s_i) \right) \tag{43}$$

The factor β defines the amount of variation that is allowed on the generated mesh widths, $\beta = 0$ resulting in equally spaced points using the smallest required mesh width on the curve. Setting β to a high value enables the limits to move more freely and will result in distributions similar to the ones of our initial solution. The resampling is then carried out as described in section 4.3.3 in order to obtain a set of limit points. Note how this approach naturally extends our idea of the accumulated mesh width by adding additional smoothing functionality. The error bound e_{tol} now influences our guidance function, the required mesh width. But even if e_{tol} would allow a rapid change in mesh width, this mesh width might get further limited by the smooth variation constraint.



(a) $\beta = 0.8$

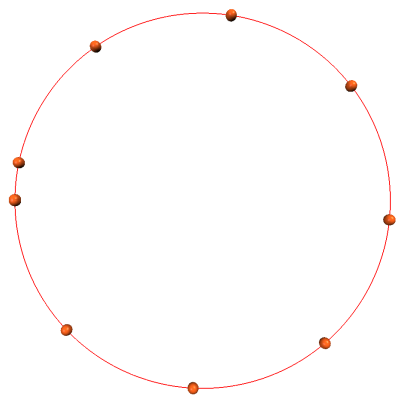
(b) $\beta = 0.2$

(c) $\beta = 0.0$

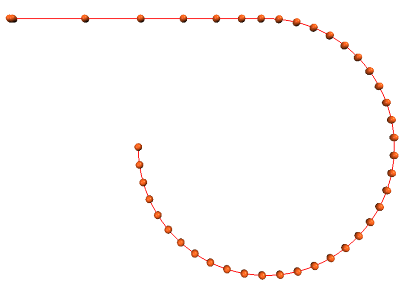
Figure 77: Limit point distributions obtained by the presented algorithm. At the end of the curve the last interval is cut off. $e_{tol} = 10^{-5}$ units

Figure 77 shows resulting limit point distributions for a fixed error tolerance e_{tol} and varying falloff factor β. In 77a we see a result for a falloff factor of $\beta = 0.8$. The value β actually represents an allowed rate of change, which here means that the mesh width is allowed to vary up to 80 percent from segment to segment. As we can see the mesh widths are given enough room, but they still vary smoothly. In 77b β is reduced, in this case more limit points have to be spent to realize a certain change of mesh width. In 77c β is set to zero, thus equal mesh widths are enforced. Since the needs of every required mesh width have to be satisfied, the smallest mesh width is applied all over the curve.

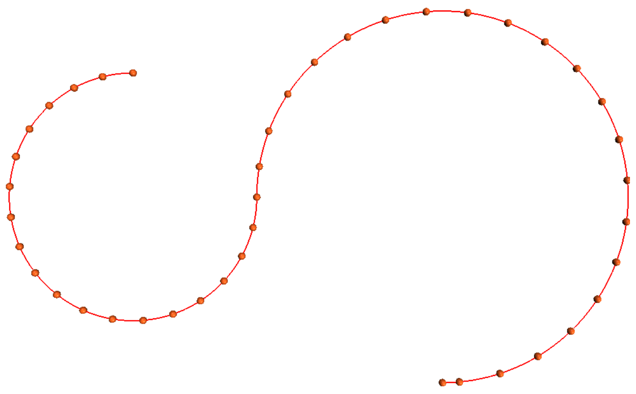
Figure 78 shows how the algorithm operates on certain curve configurations. In 78a we see a circle, and independent of the used falloff



(a) Limit point distribution on a circle, $e_{tol} = 10^{-3}$ units, $\beta = 0.8$.



(b) Limit point distribution on a circle segment with attached line, $e_{tol} = 10^{-6}$ units, $\beta = 0.3$



(c) Limit point distribution on two attached circle segments, $e_{tol} = 10^{-5}$ units, $\beta = 0.2$

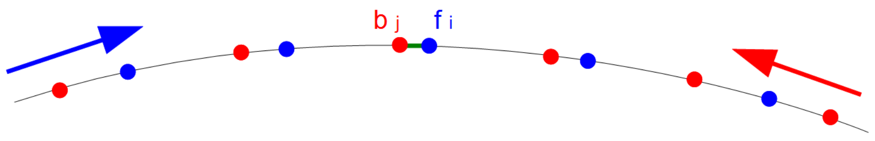
Figure 78: Limit point distribution on some special curve configurations.

rate β we obtain exactly the spacings dictated by the required mesh width.

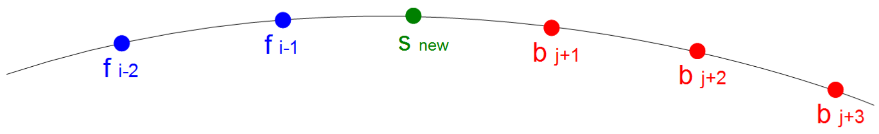
In 78b we see a transition from a circle segment into a straight line. First the limits are spaced equally along the circle, then at the transition they speed up into the zero curvature segment. Note that the realized mesh widths should be restricted in a practical scenario, in order to prevent too big mesh widths in regions of zero curvature and too small (practically unrealizable) mesh widths in high curvature regions.

In 78c we see a transition between two circles, the limits varying smoothly between the two required mesh widths.

On a general curve applying the resampling from begin to end will likely yield a different result than running vice versa. There is also the problem that the last segment will be cut off at the curve's end (see for instance 78a), destroying the smooth variation property. Two simple approaches to fix this have been tested.



(a) Inside the green interval the mesh required widths cannot be violated.



(b) The two chosen points have been replaced by the green »glue point«.

Figure 79: Finding an interval to glue together two limit point distributions that were produced in different curve directions.

The first approach obtains one result for each resampling direction, and glues them together at the most suitable position. Forward and a backward limit point distributions can be described by two sets of arc-length positions $\mathbf{f} = (f_0, f_1, \ldots, f_{k-1}, f_k)$ and $\mathbf{b} = (b_0, b_1, \ldots, b_{m-1}, b_m)$. For each forward limit position f_i, the nearest backward limit position b_j can be found, such that $f_i - b_j \geqslant 0$. It is safe to exchange those two positions by a single new glue point L_{new} at a position $s \in [b_j, f_i]$. This way both involved mesh widths are shortened and no required mesh width is violated. We are searching for the index pair (i,j) given by

$$\underset{i,j}{\operatorname{argmin}} \quad f_i - b_j \quad , \quad f_i \geqslant b_j.$$

Then a new unified limit distribution can for instance be obtained as

$$\mathbf{fb} = \left(f_0, \ldots, f_{i-1}, \frac{1}{2}\left(b_j + f_i\right), b_{j+1}, \ldots, b_m \right).$$

Figure 79 visualizes this. In 79a the two chosen points can be replaced by a point placed in the green area, without violating the required mesh widths. In 79b the two distributions glued together at s_{new}.

Figure 80 shows a result for this strategy. As we can see, the situation now is much better at the endings, but depending on the chosen glue-point the glued section may look a little bit unnatural.

Since the glue-point approach can obviously lead to non-optimal results in some situations, a second approach is discussed. Imagine a limit point distribution retrieved by the smooth resampling described above. As already stated, it can happen that the last interval is cut off

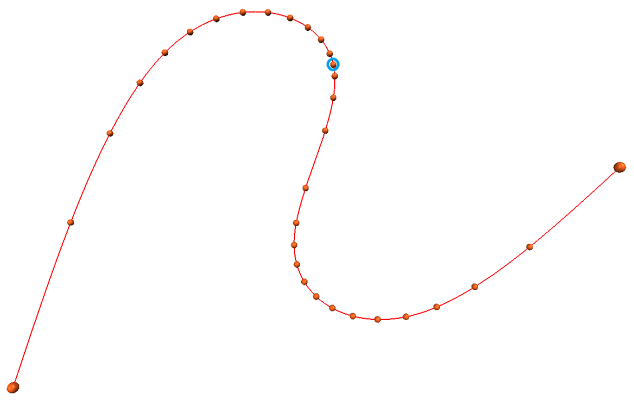


Figure 80: Limit point distribution after the glue-point strategy. $\beta = 0.8$, $e_{tol} = 10^{-5}$ units

and thus too short. In order to fix this, we can rescale the current distribution by applying a scale factor gamma to our mesh width weights in the resampling function.

$$r(s) = \min_i \left(|s - s_i| \cdot \beta + \gamma \cdot mw(s_i)\right) \tag{44}$$

The trick is now to find a distribution which delivers the same number of limit points, but with the last point lying at the curves end. This can be formulated as an optimization problem with the free variable $\gamma \in [0, 1]$. The strategy has been found to converge fast and very reliably.

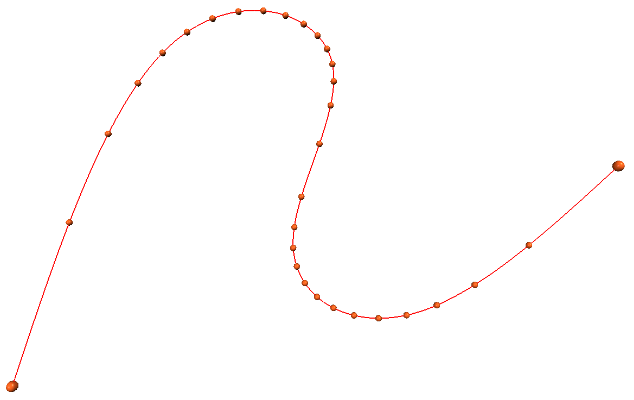
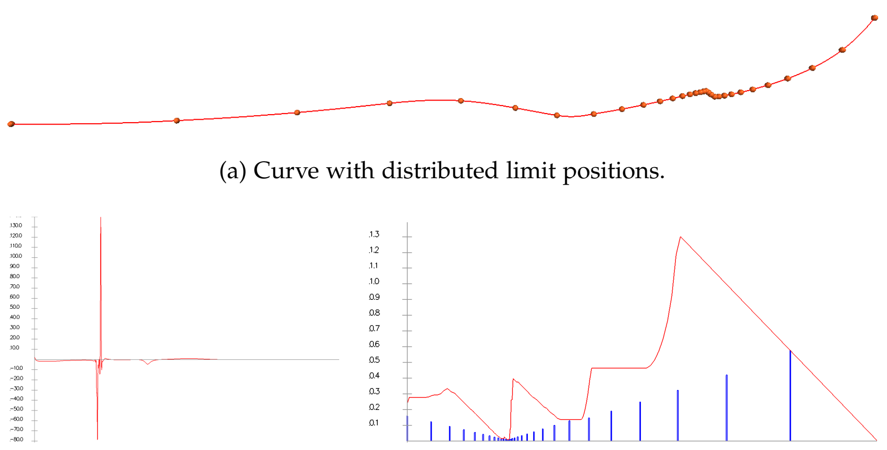


Figure 81: Limit point distribution with additional endpoint optimization. $\beta = 0.8$, $e_{tol} = 10^{-5}$ units

Figure 81 shows a result for this endpoint optimization. The result now looks more natural and varies smoothly over all segments, at the cost of a little bit smaller mesh widths than actually needed and higher computation times.

In figure 82 we want to validate the limit point distribution strategy discussed up to this point. Figure 82a shows the test curve and



(a) Curve with distributed limit positions.

(b) Preprocessed curvature profile.

(c) Accumulated (red) vs. realized (blue) forward mesh widths.

Figure 82: Validation of the limit distribution strategy. $\beta = 0.3$, $e_{tol} = 10^{-4}$ units

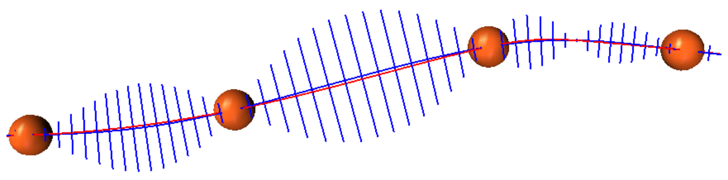
resulting limit positions, figure 82b the estimated curvature profile. In 82c we see that the smooth meshwidth-based resampling strategy produces valid mesh widths. The accumulated mesh width is plotted in red, the mesh widths of the resulting distribution are plotted at their respective positions as blue peaks. The resampling algorithm produces smoothly varying mesh widths and at the same time globally adjusts them to comply to all required mesh widths. The compliance to all required mesh widths clearly shows, as all realized mesh widths stay below the accumulated mesh width at their respective positions.

4.11 OPTIMIZATION

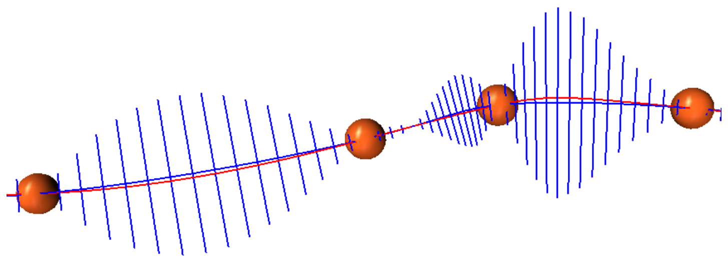
As stated earlier, optimizing limit points has its difficulties. Moving a single limit potentially influences even more neighboring limits than in the control point case, resulting in a wave-like behavior, in which the error is pushed along the curve. Figure 83 shows this behavior, which leads to many local minima for an optimization.

With our algorithm we can now provide an initial limit point distribution for such an optimization in order to further reduce the distance error. In fact we can also provide an initial control point distribution, as control points are easy to obtain. It is clear that a further limit/control point optimization might violate the required mesh widths of our initial distribution, but further optimization will be needed for various reasons.

- Our mesh widths are based on curvature *estimates*, the accuracy depending on factors such as sample density and noise level.



(a) Limit point distribution with distance error plotted in normal direction. Since only the signed distance is computed, the distances are plotted to both sides of the curve.



(b) As one limit point is moved, the error is pushed into other segments of the spline.

Figure 83: Limit point movement and non-linear behavior of the distance error.

- Our mesh widths are further based on local curve estimates, which will not hold when moving away from the point of interest. The circle of curvature is a tight but also very rough estimate. We can react on a change of mesh widths, but the radial distribution of limit points will usually not hold.
- We rather provide an estimate for the required number of limits and distribute them into the curve regions accordingly.
- It is expected that under further optimization the limits will not move that far from their initial positions, but that the B-spline will rather »snap« to the data a little closer.
- Our hypothesis is that our initial distribution is precise enough, such that a further optimization is more a detail operation.
- In order to enforce this, we constrain the distances the limits are allowed to move to a fraction of their neighboring segment lengths (e.g. 20 percent).

Since an optimization solely based on the euclidean distance shows rather slow convergence and gets stuck in local minima more easily, we rely on *squared distance minimization* (SDM) introduced by Wang et al. in [43]. We will quickly outline the most important steps, for more details consult [43].

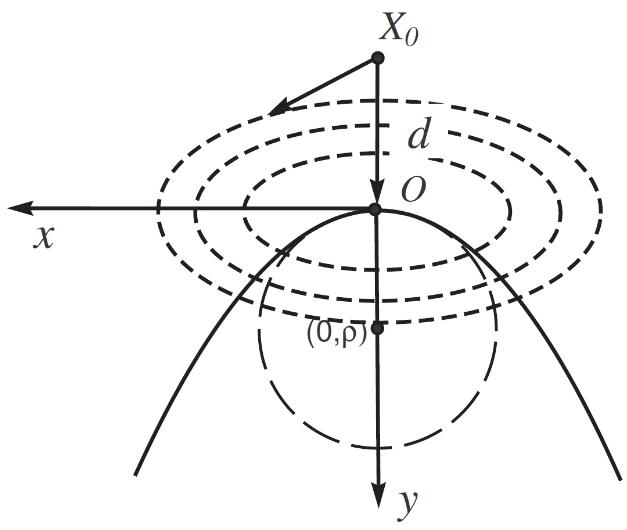


Figure 84: SDM error measure (image from [43]).

The distance of a single discrete position X_0 to a footpoint O on the B-spline approximant is measured as shown in figure 84. The local frenet frame serves as the coordinate system in which the distance is measured. The distance d is declared positive if the discrete position X_0 and the center of the circle of curvature at O lie on the same side of the curve. A second order approximation to the squared distance function at O can be expressed as

$$g(x,y) = \frac{d}{d-\rho}x^2 + y^2. \tag{45}$$

The error term suggested by Wang et al. is

$$\epsilon_i = \begin{cases} \frac{d}{d-r_i}\left[(\mathbf{g}(t_i)-\mathbf{p}_i)^T \cdot \mathbf{T}_i\right]^2 + \left[(\mathbf{g}(t_i)-\mathbf{p}_i)^T \cdot \mathbf{N}_i\right]^2 & \text{if } d < 0 \\ \left[(\mathbf{g}(t_i)-\mathbf{p}_i)^T \cdot \mathbf{N}_i\right]^2 & \text{if } 0 \leqslant d < r \end{cases},$$

where d is defined as described above, $\mathbf{p}_i$ is a sample of our discrete curve, and $\mathbf{g}(t_i)$ is the footpoint (nearest point) of $\mathbf{p}_i$ on the fitted B-spline curve $\mathbf{g}(t)$. The normal $\mathbf{N}_i$, tangent $\mathbf{T}_i$ and radius of curvature r_i of the B-spline curve have to be calculated at the footpoint.

The error to be minimized by the optimization is

$$\epsilon_{tot} = \frac{1}{2}\sum_i \epsilon_i + \alpha \cdot f_s, \tag{46}$$

where f_s is an energy functional to enforce minimum variation, which in this implementation is chosen as

$$f_s = \int \|\mathbf{g}''(t)\|^2 dt. \tag{47}$$

One step of optimization involves minimizing this error term by moving the control points accordingly. The footpoint properties t_i, $\mathbf{N}_i$, $\mathbf{T}_i$ and r_i are not calculated in each iteration of this minimization step, instead they will be calculated before each step and then fixed during the iterations. The optimization step itself originally involves solving a linear system of equations with additional constraints, as described in chapter 3. This is possible in the control point case.

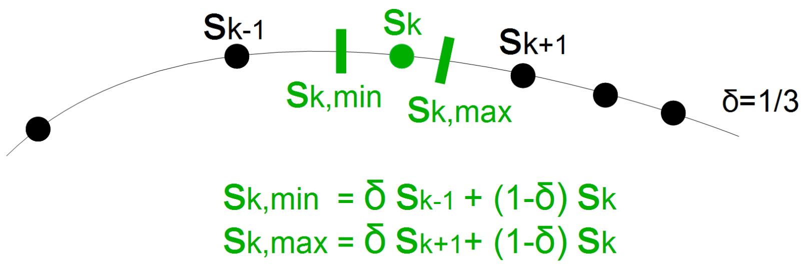


Figure 85: Explanation of the utilized parameter constraint. A limit point is allowed to move into a certain percentage δ of its initial neighbour intervals.

If we want to parameterize the optimization in terms of limit points rather than control points, we need a different solution. We are using a constrained multivariate LM optimization, with the normalized arc-lengths $s = (s_0, s_1, \ldots, s_{k-2}, s_{k-1})$ of the limit positions as parameters. The parameters are constrained not to move further than a certain percentage δ from their original positions into their neighboring segments. This is visualized in figure 85.

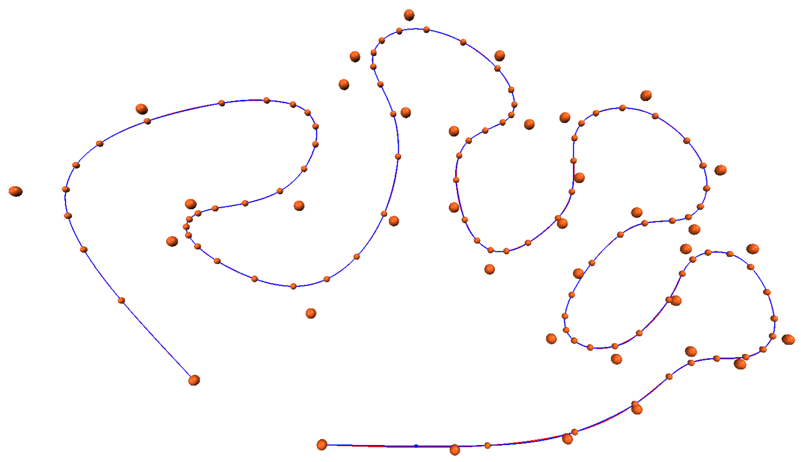
The error function minimized by the LM algorithm then includes the following steps.

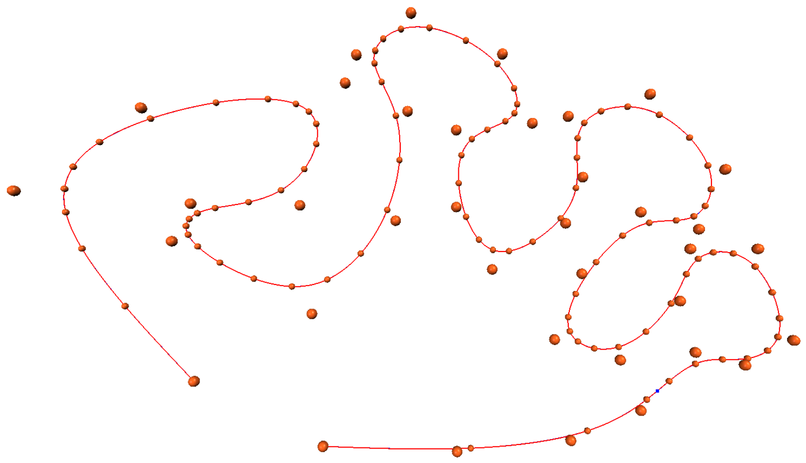
1 Obtain the limit positions by sampling the discrete curve using the normalized arc-lengths s.

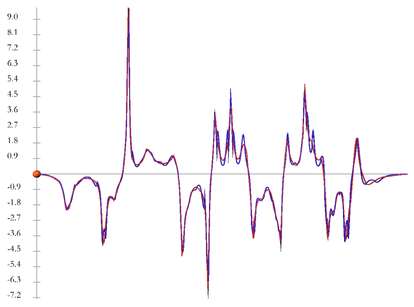
2 Calculate the control points iteratively.

3 Calculate the SDM error term f_s.

The runtime obtained for one optimization step is of course much worse than in the original approach, but SDM will generally converge pretty fast, good improvements can be obtained for most of our test


(a) Initial limit point distribution, $\epsilon = 0.01127$ units. Input spline=red, fitted spline=blue.


(b) After further optimization, $\epsilon = 0.00117$ units. Input spline=red, fitted spline=blue.


(c) Initial distribution curvature.

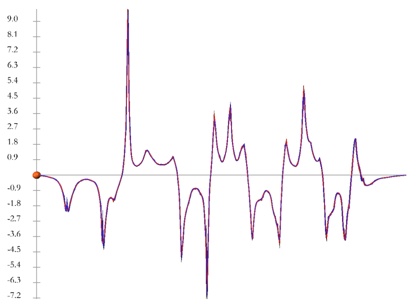

(d) Optimized distribution curvature.

Figure 86: SDM limit point optimization. $e_{tol} = 10^{-4}$ units, $\beta = 0.7$, 50 iterations

curves in fewer than 10 iterations.

Figure 86 shows results for limit point SDM. The initial limit point distribution is obtained by the strategy introduced in the last sections.

The big spheres show the control points of the test spline curve, the small ones the generated limit positions. The maximum fitting error after the optimization is roughly one order of magnitude lower than before and the resulting curvature also follows the true curvature more accurately. Looking at the curvature of the initial limit distribution, we can see that our distribution algorithm alone cannot satisfy our initial criteria. It still contains many additional oscillations which are not present in the input spline. Furthermore, the error is still quite high and does not lie below the provided error tolerance. We will discuss this in more detail in the next chapter.

5 RESULTS AND FURTHER ANALYSIS

5.1 RESULTS

We will now analyze the algorithm introduced in the last chapter regarding the curve and curvature quality it generates.

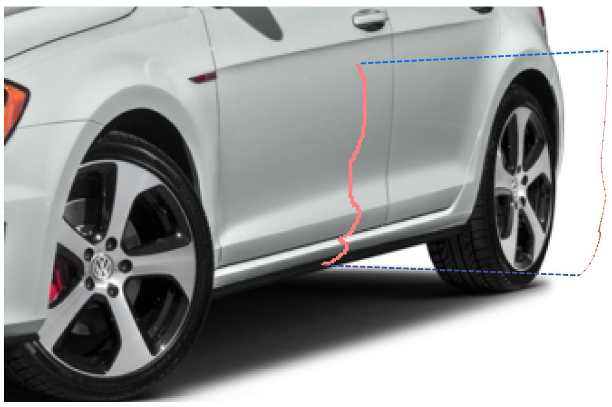


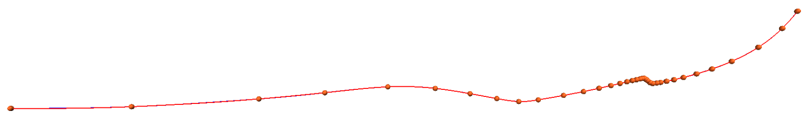
Figure 87: Cross-section of a Golf 7 car door.

The first curve represents the cross-section of a car door of a Volkswagen Golf 7. The location of the cross-section is depicted in figure 87, results for our algorithm in figure 88.

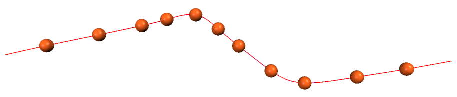
It is a difficult curve to fit, as it obtains both long low-curvature regions and a rapid S-shaped part. The curve has a total length of 1.553 units. What catches the eye is that in order to reach a final fitting error in the magnitude of 10^{-4} units we need to spend a lot of limit points, even in regions of low curvature (figures 88ab). In figure 88c we identify a dent in the curvature profile that resembles a typical B-spline artifact. The curvature profile is captured quite nicely though (figure 88d).

We can also see that our maximum error bound $e_{tol} = 10^{-5}$ units is not met even after further optimization. We will discuss this issue in more detail below.

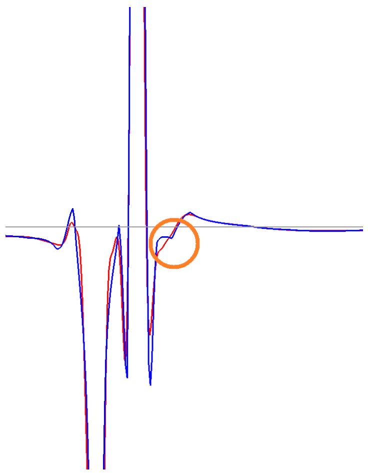
Figure 89 shows another cross-section of the door model. The curvature is generally lower here, but the curve obtains a clear bend at the characteristic lines of the door. In figures 89abc we see the limit point distributions for several error boundaries e_{tol}. Figures 89def show the corresponding curvature profiles.



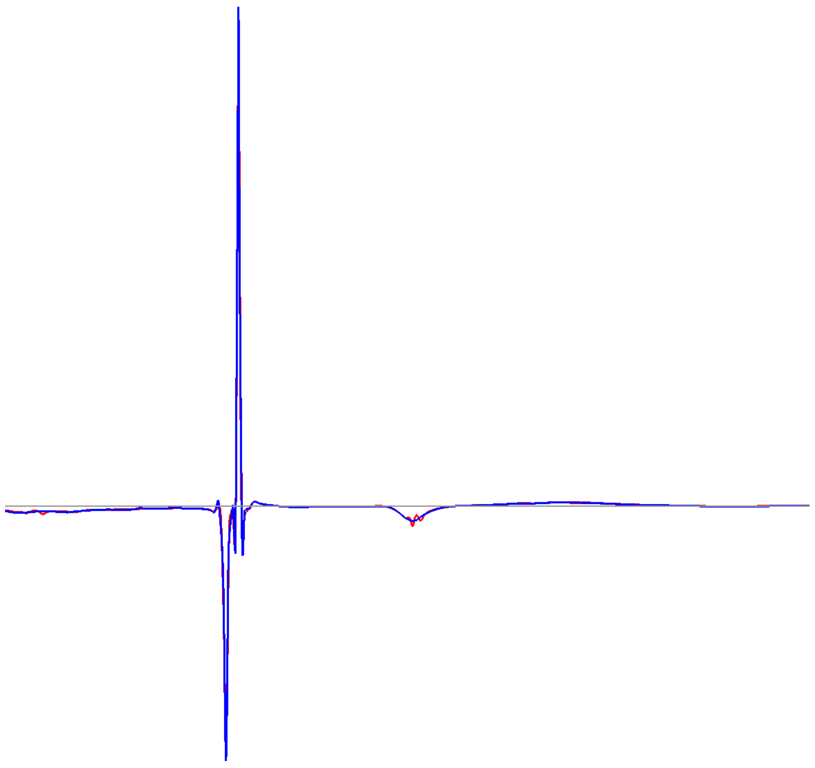
(a) Curve with distributed limit positions.



(b) Curve Detail.



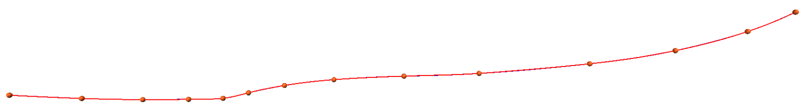
(c) Curvature detail.



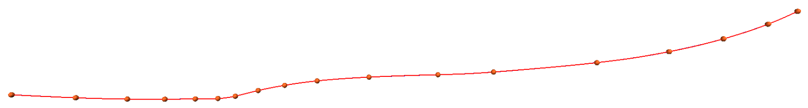
(d) Curvature profile: Curve (red), spline (blue).

Figure 88: Example: Sampled door with S-part. e_{tol} : 10^{-5} units, curve length: 1.553 units, error after limit distribution: 0.001079 units, error after further optimization: 0.000283 units

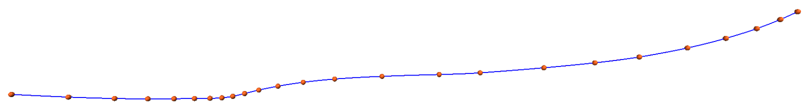
Again the actual fitting errors do not truly meet the specified error boundary, but they correlate, the fitting error decreasing with lower



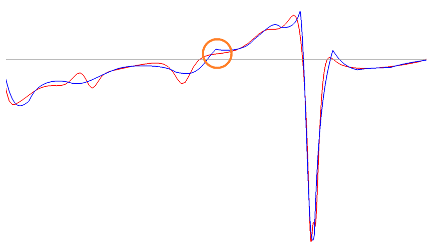
(a) Curve with distributed limit positions. e_{tol} : 10^{-5} units, final error: 0.000312 units



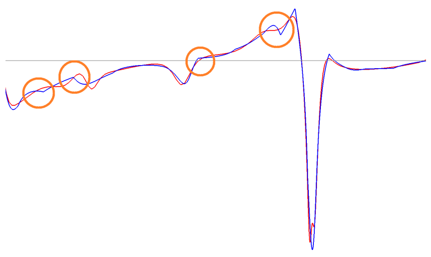
(b) Curve with distributed limit positions. e_{tol} : 10^{-6} units, final error: 0.000135 units



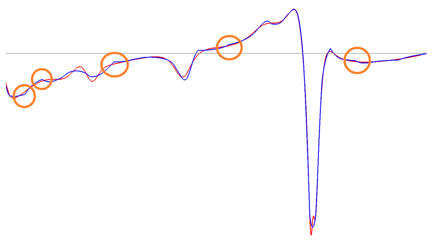
(c) Curve with distributed limit positions. e_{tol} : 10^{-7} units, final error: 0.000053 units



(d) Curvature profile, curve (red), spline (blue). e_{tol} : 10^{-5} units



(e) Curvature profile, curve (red), spline (blue). e_{tol} : 10^{-6} units

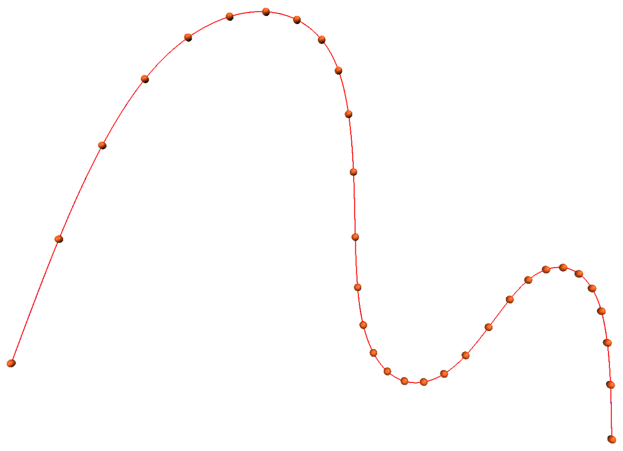


(f) Curvature profile, curve (red), spline (blue). e_{tol} : 10^{-7} units

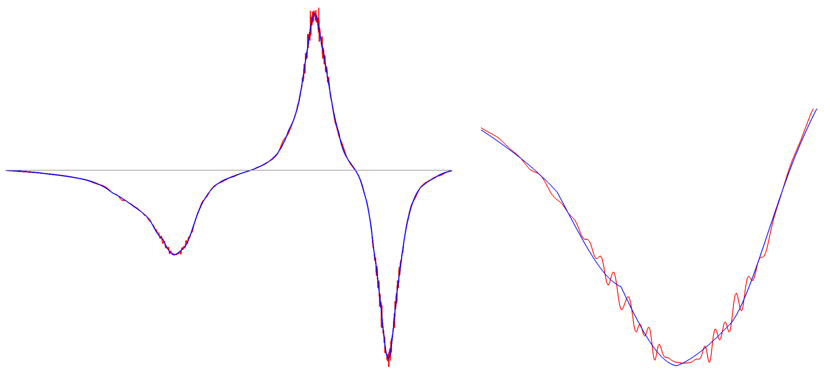
Figure 89: Example: Sampled door with characteristic line. Curve length: 3.1 units

e_{tol}. Further, the curvature profile of the spline converges more and more to the original curvature as e_{tol} gets lower.

In the curvature profiles we marked several regions where the $\mathbf{G}^2$ continuity of the cubic B-spline shows clearly, and in which surroundings we suspect B-spline artifacts. We can see that even with decreasing error bound new artifacts can not be prevented, although the curvature profile of the spline fits the profile of the curve better. This is to be expected, we refer to the verdict of section 4.8 (artifact analysis). The curvature that is produced might suffice for many fields of application, it remains full of ripples though and will not be please a surface engineer - and thus not meet the challenges of high quality automotive design. Nonetheless it could easily serve as an initial solution for further manual refinement.



(a) Curve with distributed limit positions.



(b) Curvature profile, curve (red), spline (blue).

(c) Curvature detail, curve (red), spline (blue).

Figure 90: Example: Generic spline curve. e_{tol} : 10^{-5} units, curve length: 1.68 units, final error: 0.000108 units

Figure 90 shows results for a generic spline curve. Here the input data obtains more noise compared to the last examples, and we can see that even if some of the noise remains after preprocessing the algorithm can cope with it quite gracefully.

Furthermore, we can see that the produced number of limits is far higher than the original set of limits which generated the input curve. We cannot claim to generate optimally sparse limits, as our limit point distribution is based on local curve estimates and these estimates need to be consolidated. We generally want to enforce homogeneous mesh widths because of the uniformity property of our

B-splines. We further do not make use of any assumption about the geometry that generated our input samples.

The fact that the spatial error of our fit does not comply to our provided error bound is also a cause for concern. One explanation might be that our local curve estimate - the circle of curvature - does only hold for a very compact region around the point of interest, and the curve will soon develop away from it. Additionally, even if we limit the rate of change, the assumption of locally equal mesh widths in the construction will not hold after consolidating the local mesh widths. Under the idealized conditions of our local estimate the error bound holds, but the consolidated mesh widths do not longer fully resemble these local constructions. Thus the error bound for the moment can only be seen as a guidance parameter for the final fitting error.

This issue needs to be analyzed in more detail though and will be subject to future research.

We identified several issues with our approach, which make it hardly applicable for industry-grade high quality design tasks. In the next section we will try to gain more insight into the requirements of high-quality CAGD, in order to be able to analyze our work in this context.

5.2 CLASS A MODELING

In this section we shortly want to discuss what »fair« means in the context of design curves and surfaces. This is of course very subjective and depends on a certain domain of application. Since the resulting curvature quality does not seem to be controllable by the proposed design, at least not in a degree that would satisfy surface engineers, some important questions arise.

- What does a beautiful curvature profile even look like and what are the means to describe it?
- What does this mean for industry-grade high quality B-spline fitting?
- What existing CAGD curves (or surfaces) naturally generate fair curvature?

A first answer is given by evaluating surface quality measures in the automotive industry, where quality requirements are traditionally very high. Notice that the following insights are mostly based on a cooperative project with Volkswagen AG and may not apply to the automotive industry in general. Exact definitions actually vary strongly depending on the country or even company.

In German automotive design surfaces are divided into three categories.

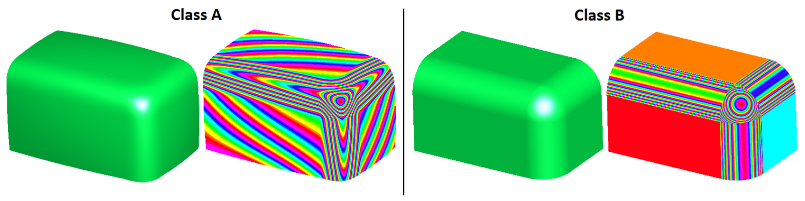
- **Class C**: Surfaces that are only used temporarily or for tools that are involved in the production of the higher class surfaces.
- **Class B**: Surfaces not visible to the customer, like surfaces on the interior of objects.
- **Class A**: Surfaces with design intent, which are visible to the customer, like the chassis of a car.

An important notion is the one of *class A* design, describing surfaces that meet the highest quality standards. While not being defined consistently throughout the industry, it is common opinion that class A shape quality can be expressed by a certain curvature quality, depicted in a very compact and intuitive way by a curvature profile.
In our opinion class A quality can be described on spot by two properties: Sparsity and minimum variation. These properties may be expressed in several related ways.

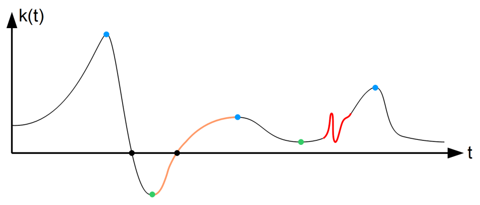
- **Sparse shape**: No unnecessary variation on the shape (ripples, bends, oscillations, etc.).
- **Sparse design geometry**: A minimum number of control/limit points. No use of *unnecessary* control points which may lead to additional shape variation. Sparsity is also required for controllability.
- **Sparse curvature**: A minimum number of purposeful curvature extrema (representing strong features on the shape) and zero crossings (representing points of inflection). A minimum amount of curvature variation between these extrema.

These categories of course go hand in hand, curvature reflecting the shape, the used design geometry generating shape and curvature, and so on. What class A design should achieve in a nutshell: *Describing an object accurately while using a representation which is as simple as possible, obtaining a minimum set of features placed with care and purpose*.

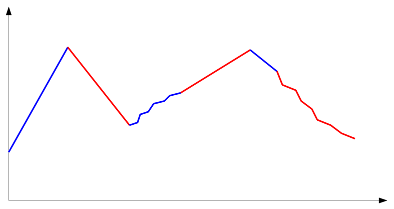
One implication for fair curvature profiles is for instance that we need continuous curvature. This is why class A quality is also connected to the continuity class of the used geometry. The difference between class A and class B surfaces is visualized in figure 91a by making use of highlight analysis. The reflections on a surface immediately reveal curvature-discontinuous areas, therefore in practice curves and surfaces used for class A design are of continuity class $\mathbf{G}^2$



(a) Comparison between class A and class B surfaces. The class B surface is only tangent continuous. The class A surface is curvature continuous, which results in a smoother and visually more pleasing look. The difference can be seen even more clearly via highlight analysis. (Image from Wikipedia)



(b) Curvature fairness. The graphic depicts curvature extrema (blue, green), fair transitions (orange) and spontaneous ripples (red).



(c) A curvature monotonic $\mathbf{C}^2$ curve that is not fair.

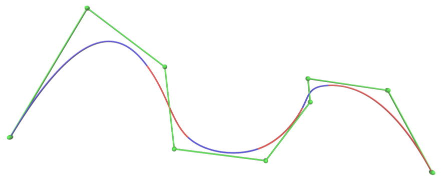
Figure 91: Curvature and class A.

or higher.

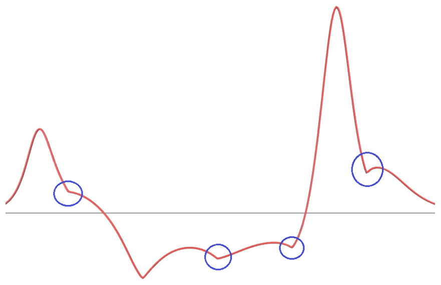
Furthermore, the curvature profile needs to vary smooth and strictly monotonic between a few clearly defined extrema. This is depicted in figure 91b. Curvature monotonicity alone cannot ensure fair curves, as shown in 91c. We favor transitions which obtain a minimum amount of variation, and behave like e.g. lower-degree polynomials.

Many design curves inherently obtain smooth curvature transitions between their segments, although curvature monotonicity might not be guaranteed. This results in bends at the segment borders and undesired curvature extrema, which most prominently shows in the case of B-splines (section 4.8). Since undesired curvature extrema ultimately result in undesired shape features they need to be prevented at all cost.

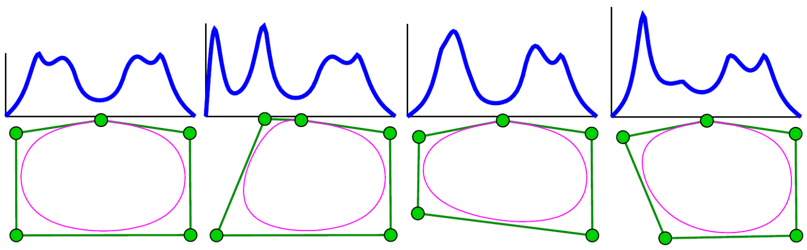
When we look at literature regarding this topic, Farin in [11] for instance states that a curve or surface with fair shape *is supposed to have a curvature characteristic that varies monotonically for the most part*. Farin in this work also discusses another interesting topic. Since the notion of class A is more widespread in the context of surfaces, literature more seldom refers to »class A curves«. Farin states that *ultimately feature curves will be used to define surfaces*, which is of course true for B-spline curves (section 2.6.5). He also states that in order to be part of a class A surface, a curve on the surface has to be class A itself. By defining class A via continuity classes, curvature monotonicity and minimum variation, it is safe to say that every curve embedded into a class A surface needs to be class A itself and has to inherit these properties. We can also state that a set of class A curves can be used to span a class A surface.



(a) Uniform cubic B-spline curve.



(b) Curvature profile. Artifacts are introduced as bends, in between new maxima or minima can form.



(c) B-spline curvature is hard to control, as we also pointed out in [19].

Figure 92: B-spline artifacts might introduce unwanted curvature extrema.

We can now further ask: How can we achieve such fair shapes? The most simple answer in the context of design curves is: Place the control points in such way that a fair shape is formed. But this is not equally easy for all design curves. The artifacts inherent to B-splines result in unpredictable behavior in terms of curvature, since new curvature extrema are possibly introduced between the spline segments during each modification. This behavior is caused by the pull the spline exerts on its segments, and controlling this peculiarity of spline curves is a huge additional effort and the bane of B-spline fitting and optimization when fair curvature is demanded. Examples are given in figure 92. These deficits are often mitigated by increasing the B-spline degree. Increasing the degree dampens the artifacts, but cannot hide them completely. In the case of cubic B-splines the artifacts are just more apparent than on higher degrees.

In state of the art software short Bézier patches are still used to model high quality shapes, since shape variation in this case is just easier to control.

For Bézier curves we can again refer to [11], where a simple scheme for control point placement is introduced, that ensures strictly monotonic curvature variation between the Bézier curve segments. Although such special rules of construction could be a viable option for curve design, they also have their disadvantages. For other design curves, such as B-splines, rules might become much more complex and limit the domain of possible shape configurations, and ultimately the flexibility of the design curve.

Imagine a design curve which inherently produces monotonic curvature transitions between its segment borders. Analytic curves which are known for their smooth and monotonic curvature progression are spirals, such as the logarithmic spiral or the clothoid. A design curve consisting of spiral segments will be discussed in section 5.4, and it can be shown that such a curve has many desirable qualities when it comes to intuitive and fair modeling.

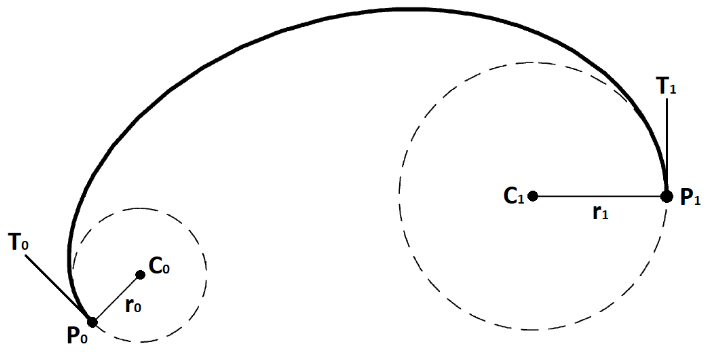


Figure 93: $\mathbf{G}^2$ Hermite-like data interpolation: Connecting circles with a curve. (Image adapted from [15])

In addition to strict curvature monotonicity on the segments and smooth transitional behavior, we can demand constraints on the segments endpoints in order to obtain some kind of holy grail for CAGD design curves. How many of these constraints can be met by a curve is a matter of its degrees of freedom (DOF). Four DOF are needed to match given endpoints with a curve. The remaining DOF can for instance be used to specify tangents or curvature values at the endpoints.

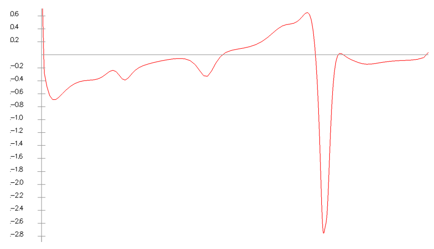
The interpolation of given positions and the first k derivatives at these positions can be achieved with a polynomial using $\mathbf{G}^k$ *Hermite interpolation*. If the polynomial curve is required to match given positions and tangent values, one speaks of $\mathbf{G}^1$ Hermite interpolation, $\mathbf{G}^2$ Hermite interpolation in addition requires to match given curvature values. The problem of $\mathbf{G}^2$ Hermite interpolation is equal to the task of joining circles in a $\mathbf{G}^2$ manner, since a point on a circle inherently fixes position, tangent and curvature. This is visualized in figure 93.

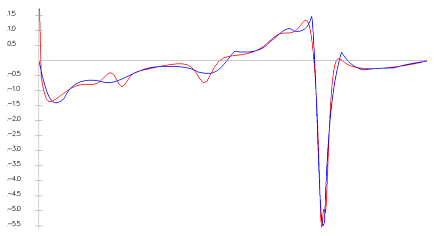
A design curve which interpolates endpoints in a $\mathbf{G}^2$ Hermite fashion using a single segment would give us total control over curvature design. Unfortunately the expectations on a design curve are a little contradictory, since on one hand it should be of low degree and on the other hand very expressive. Further, some flexibility is usually taken away when forcing the curve to meet such conditions. Additional DOF can be spent on a »shape parameter« that influences the form of the segment transitions (e.g. [15]).

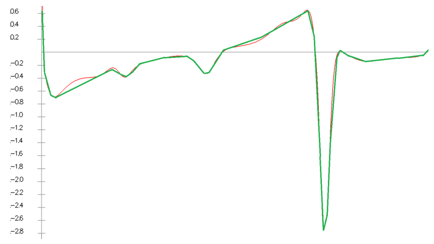
In practice it is often difficult to achieve $\mathbf{G}^2$ Hermite-like interpolation with a single design curve segment, as we have to weigh curve complexity (controllability) against degrees of freedom (expressiveness) and flexibility of the approach.

5.3 FURTHER ANALYSIS REGARDING CLASS A BEHAVIOR

Using the insight gained in the last section we can further evaluate our proposed algorithm for B-spline fitting, but this time from the viewpoint of class A design.


(a) Curvature profile of a car door cross-section.


(b) Curvature profile of the fitted uniform cubic B-spline (blue). We can clearly spot several B-spline artifacts.


(c) Is this the curvature profile we want to model?

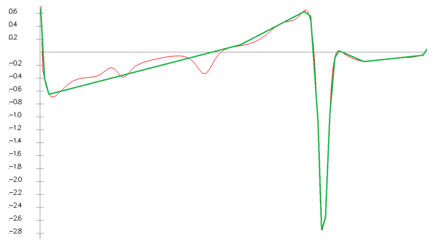

(d) Or is this curvature profile more desirable?

Figure 94: B-spline fitting curvature quality in the light of class A design.

We again have at look at the curvature profile of the door cross-section shown in figure 94. In 94b we see the curvature profile of the uniform cubic B-spline obtained by our fitting approach. Referring to the last section, it is obvious that neither the input curvature of 94a nor the result curvature of 94b is a desirable curvature profile, as they contain too much variation and are not sparse in the class A curvature sense. We identify several different problems which lead to this behavior.

B-SPLINE UNIFORMITY We already discussed the advantages of sparse geometry in the context of class A. Homogeneously varying limit point mesh widths lead to denser limit point distributions, but on the other hand are very important in the presence of B-spline uniformity. Thus our locally required mesh width is based on this assumption (see figure 65), and we further enforce homogeneity during the consolidation of all required mesh widths, resulting in globally smooth mesh width variation. This results in overly complex B-splines, which might be more prone to oscillatory behavior.

B-SPLINE ARTIFACTS B-spline artifacts are always present, they can be suppressed or - theoretically - hidden by sophisticated construction. In order to suppress them we either need to spend more control/limit points, or use higher degree splines. Since in our case the second measure is not an option, a lot of limit points have to be spent in order to reach a certain level of quality. This again contradicts limit point sparsity as demanded by class A design.

In fact, even very dense limit point distributions cannot completely assure acceptable curvature, although we observed a strong improvement. Some of the B-spline artifacts in the curvature plot of 94b are still clearly visible.

With our presented approach we can *enforce* artifact suppression by distributing the right amount of limit points into each curve region (as confirmed by artifact analysis), but we cannot guarantee fair and artifact-free curvature.

FLAWED INPUT DATA Figure 94a shows an estimated smooth version of the true curvature profile of the possibly flawed input data. It is not fair in the sense of class A modeling though. The question is: Given noisy and potentially corrupted input data, what is the curvature profile we want to model with our design curve? Is it the curvature of 94c or the curvature of 94d? Which feature is important to the surface engineer? What does the domain of class A curvature graphs even look like for a certain sector of CAGD, and how can we describe it mathematically (e.g. minimum and maximum feature size, feature frequency, amount of variation, etc.)? What if the input data is incomplete?

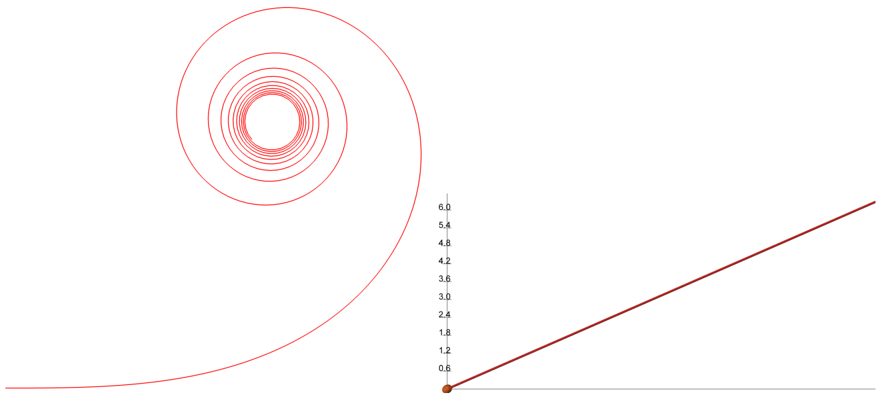
This hints at a complex task on its own, which in our opinion could prove very fruitful for automated modeling when being investigated further. We need to obtain a better understanding for the curvature domain we want to model, in order to obtain class A quality also for corrupted input data in an automated way.

THE REQUIRED MESH WIDTH As the given examples clearly show, complying to the required mesh widths and enforcing homogeneity alone cannot assure class A curvature. We base our locally required mesh width on a local curve estimate, but as we move away from this estimate, it soon becomes invalid. We mitigate this fact by taking into account all the required mesh widths for our final limit point distribution, and by enforcing homogeneity on the mesh widths. Nonetheless the resulting segments differ from the local estimates. This might also explain why the maximum distance error bound used in our local estimates cannot hold for the final fit.

SURFACE INTERPOLATION A final issue with our approach regarding class A design is rooted in its initial requirements. Placing limit points directly on the shape is of course more intuitive for the user. The downside to this approach is that this way oscillations on the input shape are easily reproduced by the fitted design curve or surface. Many well-established software packages thus approximate the input data instead of interpolating it. We try to mitigate the negative aspects of limit point placement by obtaining an intermediate smooth curve representation. Nevertheless, low-frequency oscillations on the input data might get preserved.

In the next section we will further investigate the topic of inherently curvature monotonic design curves. Being unsatisfied with the properties of B-splines, we will discuss a special kind of curve which obtains many properties desirable for class A design.

5.4 PIECEWISE CLOTHOID CURVES: AN ALTERNATIVE TO B-SPLINE MODELING?



(a) Clothoid curve. (b) Curvature profile, curvature over arc-length.

Figure 95: Approximation of a clothoid curve.

In this section we want to present an alternative to classic B-spline modeling, which developed in the wake of this work. Unsatisfied with the properties of B-splines, a more viable alternative for class A modeling was sought. When looking for an analytical curve that by nature obtains curvature monotonicity, spiral segments soon come into mind.

A very well-known and popular spiral curve is the clothoid. It is broadly used in many fields of engineering like road design or stroke curve beautification. It is not as widely used for CAGD though. The clothoid's key characteristic is that its curvature varies linearly with its arc-length. Another useful property of the clothoid curve is that its arc-length varies linearly with its parameter t. Figure 95 shows a clothoid curve and its curvature profile.

The generalized clothoid curve in the plane obtains 5 DOF. Starting with the normalized clothoid, an arbitrary clothoid curve can be defined by a translation, rotation and scaling. The starting position can then be chosen by a fifth DOF, for example a parameter value. A clothoid segment is defined by 6 DOF, an additional parameter being needed for the segment's end point.

Clothoid curves are analytically defined via Fresnel integrals, which cannot be solved directly, but there exist approximations ([12]). Parameters for these approximations are not easy to find though.

Havemann et al. in [19] overcome this by using a refinement scheme which successively adds points to a polygonal line, approximating a clothoid segment up to an arbitrary small error.

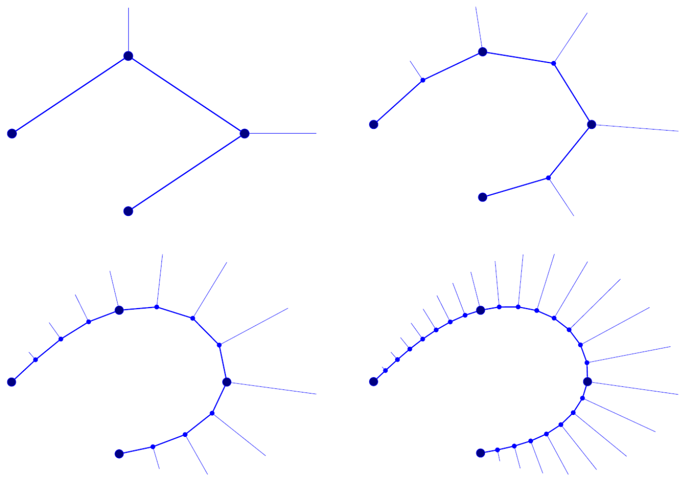


Figure 96: Iterative construction of a PCC.

Starting with a set of user defined control points (or fix points) as the initial polyline, two steps are alternated.

- New points are inserted between every two neighboring points of the polyline.

- Each point which is not a control point is optimized on the perpendicular bisector of its neighbors, such that its discrete curvature (the curvature defined by the circle through three consecutive points) approximately becomes the arithmetic mean of its neighboring curvature values.

This leads to a curvature distribution in which each inner curvature value is the arithmetic mean of its neighbors. In addition the curvature does not obtain discontinuities at the control points, and the inner points quickly converge to an equal spacing. The whole polygonal line thus converges to a chain of clothoid segments, which is named *Piecewise Clothoid Curve* (short: PCC). Figure 96 shows how a PCC is iteratively formed.

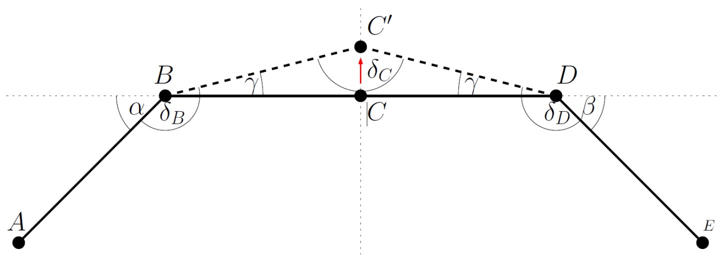


Figure 97: Basic construction for the optimization of PCC vertices.

Figure 97 visualizes how a PCC vertex C is optimized. The setting is normalized, such that the neighbors lie at $B = (-1, 0)$ and $D = (1, 0)$. By using approximations for the discrete curvature and setting $|\overline{BC}|$ and $|\overline{CD}|$ to 1, the angle γ can be calculated as

$$\gamma = \frac{\beta(|\overline{BA}| + 1) + \alpha(|\overline{DE}| + 1)}{2|\overline{BA}||\overline{DE}| + 3(|\overline{BA}| + |\overline{DE}|) + 4}.$$

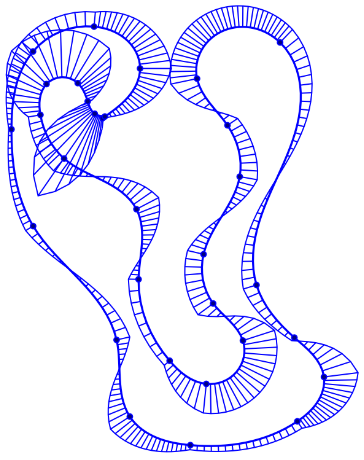
The distance along the bisector which optimizes the point then just equals $\tan\gamma$.

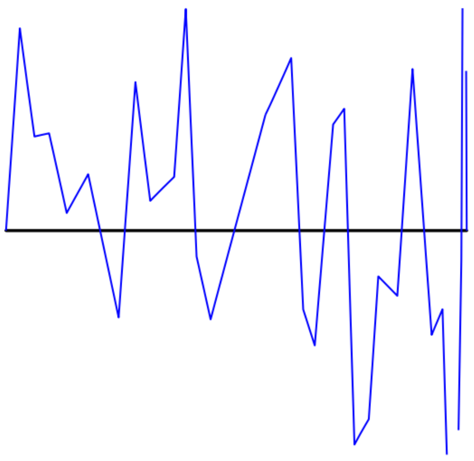
PCCs obtain many properties which are very desirable for class A design.

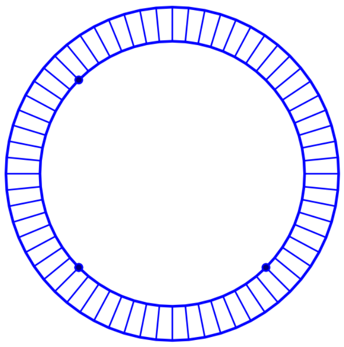
- Curvature extrema can only form at segment borders, which makes PCCs much more intuitive to edit than splines (figure 98b).

- Three points can be used to form a circle in a completely natural way (figure 98c).

- Inserting a new control point on the curve neither changes the curve nor the curvature profile.


(a) Example PCC.


(b) Curvature profile. Each line segment corresponds to a PCC segment. Curvature extrema thus only form at segment borders.


(c) PCCs naturally form circles.

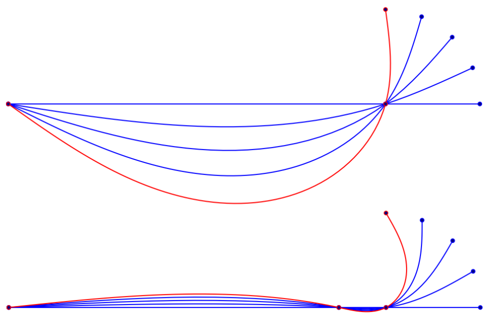

(d) Changes on the PCC can be mitigated by using multiple closely spaced control points.

Figure 98: Advantages of PCCs.

- The curve can be evaluated in as much detail as needed in a very efficient way, not unlike traditional subdivision schemes.
- Moving a control point will change the whole curve, but the changes become damped rather quickly with distance. This resembles the global behavior of limit points, but with PCC design a new control point can always be added to the curve without consequences, in order to »pin down« important areas (figure 98d).
- The PCC always interpolates its control points, making editing less abstract compared to B-spline control point editing.
- The scheme is very fast to evaluate.

In a former section we discussed the ability of a design curve to interpolate endpoints with additional constraints, a property which is very useful for flexible design. PCCs offer the possibility to specify either tangent or curvature constraints at the control points. A tangent constraint will unfortunately destroy the $\mathbf{G}^2$ continuity at a control point, but Havemann et al. provide a way to restore $\mathbf{G}^2$ continuity by inserting an additional control point in an unobtrusive way.

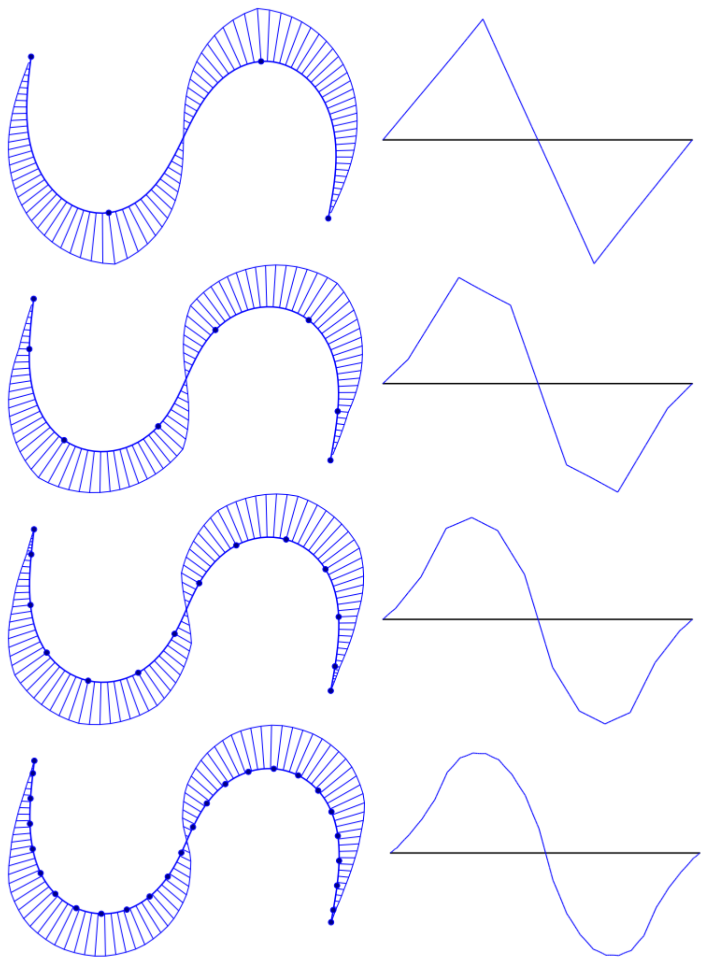


Figure 99: Iterative construction of a clothoid spline.

It might further be desirable to obtain smooth curvature profiles rather than piecewise linear ones. By inserting new control points in a special way between each two consecutive ones and then removing the old control points, a new PCC with attenuated curvature spikes can be obtained. Repeating this procedure and using the final control points as a new polygonal curve will in the limit yield a $\mathbf{G}^3$ continuous clothoid spline. This process can be achieved with only minor curve changes and repeated up to a desired accuracy. Figure 99 illustrates this process.

The flexibility of PCCs also shows in the fact that it is easy to approximate a B-spline curve and its curvature artifacts with PCCs. It

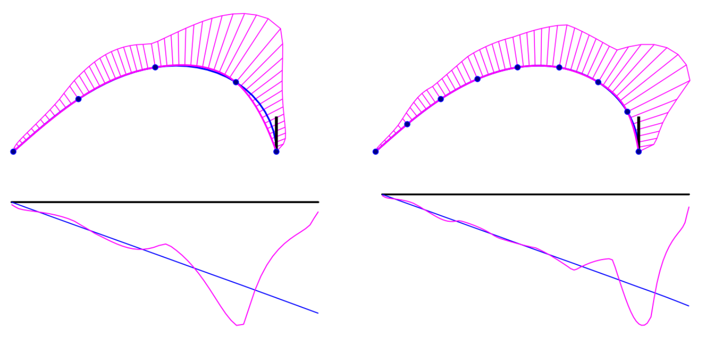


Figure 100: B-spline artifacts are hard to control and cannot be completely avoided. B-spline curves thus cannot approximate PCC curves well.

is not possible to achieve the inverse case though, as shown in figure 100. It is very hard to control B-spline curvature artifacts, which mostly results in curvature profiles simply unacceptable for class A design. PCCs in contrast are much easier to control in terms of curvature.

With their desirable properties and flexibility PCCs could be a superior alternative to spline editing in the future, and they could render class A modeling both easier and more intuitive for manual and automated processes alike. The missing link here is the generalization of PCCs to surfaces, which could be a very interesting topic for future work.

5.5 final thoughts & future research

We presented an approach to fit uniform cubic B-spline curves to noisy data points using a local curvature-based measure rooted in B-spline artifact analysis. We used this measure to distribute limit points over the discrete input curve, in order to further optimize these limits and suppress B-spline artifacts effectively. We also pointed out several problems we encountered, most notably the acquisition of a faithful curvature profile and a smooth input curve from the noisy input samples.

Our approach does not yield curvature profiles which meet the high demands of class A automotive design. We identified several issues within our approach, but also discovered that the uncontrollable curvature behavior of the B-spline is a major issue regarding class A modeling. We thus investigated the properties of class A curves and presented a viable alternative to classic B-spline modeling. Design curves which obtain more intuitive curvature behavior than B-splines might yield more successful approaches to automated class A conversion in the future.

We gathered several ideas for future research on this topic, which we shortly want to outline below.

SIMPLE SCHEMES FOR B-SPLINE CURVATURE MONOTONICITY If replacing B-spline design is not an option, we should at least gather more insight on their curvature behavior. Even if many spline curves have already been inspected regarding the conditions needed for curvature monotonicity, these rules are often very complex. We suggest to further investigate simple schemes for control or limit point placement, in order to achieve curvature-monotonous B-spline curves, similar to the *typical curves* Farin suggested for Bézier design.

THE DOMAIN OF CLASS A SHAPES In order to be able to model class A shapes, we need to understand how the domain of class A shapes can be described through simple rules for a certain sector, such as automotive design. It could prove fruitful to gather information from the experts who manually generate class A shapes at the moment, namely the surface engineers, and to compile their rich experience into concrete mathematical rules for the description of class A shapes and the evaluation of shapes regarding their class A quality. This could for instance help us to discriminate between desired features, undesired features, and flaws, such as missing regions in the input data.

GENERALIZATION OF PCCS TO THE SURFACE CASE In order to become even more useful, the PCC scheme has to be extended to the surface case. Here fast evaluation even more becomes a major issue. In our opinion it could also be interesting to investigate if PCC surfaces could be spanned similar to tensor product surfaces.

Part I

APPENDIX

BIBLIOGRAPHY

[1] M. Alhanaty and M. Bercovier. Curve and surface fitting and design by optimal control methods. *Computer-Aided Design*, 33(2):167 – 182, 2001. URL http://dx.doi.org/10.1016/S0010-4485(00)00089-0.

[2] U.H. Augsdörfer, N.A. Dodgson, and M.A. Sabin. Artifact analysis on B-splines, box-splines and other surfaces defined by quadrilateral polyhedra. *Computer Aided Geometric Design*, 28(3): 177 – 197, 2011. URL http://dx.doi.org/10.1016/j.cagd.2010.04.002.

[3] Ilya Baran, Jaakko Lehtinen, and Jovan Popovic. Sketching clothoid splines using shortest paths. *Computer Graphics Forum*, 29:655–664, 2010. URL http://dx.doi.org/10.1111/j.1467-8659.2009.01635.x.

[4] Matthias Bein, Dieter W. Fellner, and André Stork. Genetic B-spline approximation on combined B-reps. *The Visual Computer*, 27(6-8):485–494, 2011. URL http://dx.doi.org/10.1007/s00371-011-0592-9.

[5] Andrew Blake and M. Isard. *Active Contours: The application of techniques from graphics, vision, control theory and statistics to visual tracking of shapes in motion*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1st edition, 1998. ISBN 3540762175. URL http:/dx.doi.org/10.1007/978-1-4471-1555-7.

[6] C. Deboor. *A practical guide to splines*. Springer-Verlag Berlin and Heidelberg GmbH & Co. K, December 1978. ISBN 3540903569. URL http://dx.doi.org/10.1002/zamm.19800600129.

[7] Chongyang Deng and Xunnian Yang. A local fitting algorithm for converting planar curves to B-splines. *Computer Aided Geometric Design*, 25(9):837 – 849, 2008. URL http://dx.doi.org/10.1016/j.cagd.2007.11.001.

[8] M.P. do Carmo. *Differential geometry of curves and surfaces*. Prentice-Hall, 1976. ISBN 9780132125895. URL https://books.google.at/books?id=1v0YAQAAIAAJ.

[9] Matthias Eck and Jan Hadenfeld. Knot removal for B-spline curves. *Computer Aided Geometric Design*, 12(3):259–282, May 1995. URL http://dx.doi.org/10.1016/0167-8396(94)00012-H.

[10] Gerald Farin. *Curves and surfaces for CAGD: A practical guide*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 5th edition, 2002. ISBN 1-55860-737-4.

[11] Gerald Farin. Class A Bézier curves. *Computer Aided Geometric Design*, 23(7):573–581, October 2006. URL http://dx.doi.org/10.1016/j.cagd.2006.03.004.

[12] W.H.J. Fuchs and W.K. Hayman. Rational approximation to the Fresnel integral. In B. Fuglede, M. Goldstein, W. Haussmann, W.K. Hayman, and L. Rogge, editors, *Approximation by Solutions of Partial Differential Equations*, volume 365 of *NATO ASI Series*, pages 69–77. Springer Netherlands, 1992. ISBN 978-94-010-5074-6. URL http://dx.doi.org/10.1007/978-94-011-2436-2_7.

[13] R. Goldenthal and M. Bercovier. Spline curve approximation and design by optimal control over the knots. *Computing*, 72(1-2):53–64, April 2004. URL http://dx.doi.org/10.1007/s00607-003-0046-y.

[14] A. Ardeshir Goshtasby. Grouping and parameterizing irregularly spaced points for curve fitting. *ACM Trans. Graph.*, 19(3):185–203, July 2000. URL http://dx.doi.org/10.1145/353981.353992.

[15] Zulfiqar Habib and Manabu Sakai. G2 Pythagorean hodograph quintic transition between two circles with shape control. *Computer Aided Geometric Design*, 24(5):252–266, July 2007. URL http://dx.doi.org/10.1016/j.cagd.2007.03.004.

[16] H. Hagen and G. Schulze. Variational principles in curve and surface design. In Hans Hagen and Dieter Roller, editors, *Geometric Modeling*, Computer Graphics-Systems and Applications, pages 161–184. Springer Berlin Heidelberg, 1991. ISBN 978-3-642-76406-6. URL http://dx.doi.org/10.1007/978-3-642-76404-2_7.

[17] Richard I. Hartley. In defense of the eight-point algorithm. *IEEE Trans. Pattern Anal. Mach. Intell.*, 19(6):580–593, June 1997. URL http://dx.doi.org/10.1109/34.601246.

[18] Sven Havemann. *Generative mesh modeling*. PhD thesis.

[19] Sven Havemann, Johannes Edelsbrunner, Philipp Wagner, and Dieter Fellner. Curvature-controlled curve editing using piecewise clothoid curves. *Computers & Graphics*, 37(6):764 – 773, 2013. URL http://dx.doi.org/10.1016/j.cag.2013.05.017.

[20] J. Hoschek. Intrinsic parametrization for approximation. *Computer Aided Geometric Design*, 5(1):27 – 31, 1988. URL http://dx.doi.org/10.1016/0167-8396(88)90017-9.

[21] Yu-Kun Lai, Shi-Min Hu, and Tong Fang. Robust principal curvatures using feature adapted integral invariants. In *2009 SIAM/ACM Joint Conference on Geometric and Physical Modeling*, pages 325–330. ACM, 2009. URL http://dx.doi.org/10.1145/1629255.1629298.

[22] Torsten Langer, Alexander Belyaev, and Hans-Peter Seidel. Analysis and design of discrete normals and curvatures. 2005. URL http://hdl.handle.net/11858/00-001M-0000-0014-6837-B.

[23] E.T.Y. Lee. Choosing nodes in parametric curve interpolation. *Computer-Aided Design*, 21(6):363 – 370, 1989. ISSN 0010-4485. URL http://dx.doi.org/10.1016/0010-4485(89)90003-1.

[24] Weishi Li, Shuhong Xu, Gang Zhao, and Li Ping Goh. Adaptive knot placement in B-spline curve approximation. *Computer-Aided Design*, 37(8):791 – 797, 2005. URL http://dx.doi.org/10.1016/j.cad.2004.09.008.

[25] Hongwei Lin, Guojin Wang, and Chenshi Dong. Constructing iterative non-uniform B-spline curve and surface to fit data points. *Science in China Series : Information Sciences*, 47:315–331, 2004. URL http://dx.doi.org/10.1360/02yf0529.

[26] Wei-Yang Lin, Yen-Lin Chiu, Kerry R. Widder, Yu Hen Hu, and Nigel Boston. Robust and accurate curvature estimation using adaptive line integrals. *EURASIP J. Adv. Signal Process*, 2010:25:1–25:14, February 2010. URL http://dx.doi.org/10.1155/2010/240309.

[27] Tom Lyche and Knut Mørken. Knot removal for parametric B-spline curves and surfaces. *Computer Aided Geometric Design*, 4(3):217 – 230, 1987. URL http://dx.doi.org/10.1016/0167-8396(87)90013-6.

[28] Siddharth Manay, Byung-Woo Hong, Anthony J. Yezzi, and Stefano Soatto. Integral invariant signatures. In Tomás Pajdla and Jirí Matas, editors, *Computer Vision - ECCV 2004*, volume 3024 of *Lecture Notes in Computer Science*, pages 87–99. Springer Berlin Heidelberg, 2004. ISBN 978-3-540-21981-1. URL http://dx.doi.org/10.1007/978-3-540-24673-2_8.

[29] Cengiz Oztireli, Gaël Guennebaud, and Markus Gross. Feature preserving point set surfaces based on non-linear kernel regression. *Computer Graphics Forum*, 28(2):493–501, 2009. URL http://dx.doi.org/10.1111/j.1467-8659.2009.01388.x.

[30] Hyungjun Park. An error-bounded approximate method for representing planar curves in B-splines. *Computer Aided Geometric Design*, 21(5):479–497, May 2004. URL http://dx.doi.org/10.1016/j.cagd.2004.03.003.

[31] Hyungjun Park and Joo-Haeng Lee. B-spline curve fitting based on adaptive curve refinement using dominant points. *Computer-Aided Design*, 39(6):439 – 451, 2007. URL http://dx.doi.org/10.1016/j.cad.2006.12.006.

[32] Les Piegl and Wayne Tiller. *The NURBS book*. Springer-Verlag, London, UK, UK, 1995. ISBN 3-540-55069-0.

[33] Michael Plass and Maureen Stone. Curve-fitting with piecewise parametric cubics. *SIGGRAPH Comput. Graph.*, 17(3):229–239, July 1983. URL http://dx.doi.org/10.1145/964967.801153.

[34] Helmut Pottmann, Stefan Leopoldseder, and Michael Hofer. Approximation with active B-spline curves and surfaces. In *Proceedings of the 10th Pacific Conference on Computer Graphics and Applications*, PG '02, pages 8–25, Washington, DC, USA, 2002. IEEE Computer Society. ISBN 0-7695-1784-6. URL http://dx.doi.org/10.1109/PCCGA.2002.1167835.

[35] Helmut Pottmann, Johannes Wallner, Qi-Xing Huang, and Yong-Liang Yang. Integral invariants for robust geometry processing. *Computer Aided Geometric Design*, 26(1):37–60, January 2009. URL http://dx.doi.org/10.1016/j.cagd.2008.01.002.

[36] Anshuman Razdan. Knot placement for B-spline curve approximation, 1999. URL https://www.researchgate.net/publication/2465899_Knot_Placement_for_B-Spline_Curve_Approximation.

[37] Biplab Sarkar and Chia-Hsiang Menq. Parameter optimization in approximating curves and surfaces to measurement data. *Computer Aided Geometric Design*, 8(4):267 – 290, 1991. URL http://dx.doi.org/10.1016/0167-8396(91)90016-5.

[38] Eric Saux and Marc Daniel. An improved Hoschek intrinsic parametrization. *Computer Aided Geometric Design*, 20(8–9):513–521, 2003. URL http://dx.doi.org/10.1016/j.cagd.2003.06.004.

[39] Thomas Speer, Markus Kuppe, and Josef Hoschek. Global reparametrization for curve approximation. *Computer Aided Geometric Design*, 15(9):869 – 877, 1998. URL http://dx.doi.org/10.1016/S0167-8396(98)00024-7.

[40] John M. Sullivan. Curvature measures for discrete surfaces. In *ACM SIGGRAPH 2005 Courses*, SIGGRAPH '05, New York, NY, USA, 2005. ACM. URL http://doi.acm.org/10.1145/1198555.1198662.

[41] C.W.A.M. van Overveld. Pondering on discrete smoothing and interpolation. *Computer-Aided Design*, 27(5):377 – 384, 1995. URL http://dx.doi.org/10.1016/0010-4485(95)96801-R.

[42] Tzvetomir Ivanov Vassilev. Fair interpolation and approximation of B-splines by energy minimization and points insertion. *Computer-Aided Design*, 28(9):753 – 760, 1996. URL http://dx.doi.org/10.1016/0010-4485(95)00087-9.

[43] Wenping Wang, Helmut Pottmann, and Yang Liu. Fitting B-spline curves to point clouds by curvature-based squared distance minimization. *ACM Trans. Graph.*, 25(2):214–238, April 2006. URL http://doi.acm.org/10.1145/1138450.1138453.

[44] Huaiping Yang, Wenping Wang, and Jiaguang Sun. Control point adjustment for B-spline curve approximation. *Computer-Aided Design*, 36(7):639 – 652, 2004. URL http://dx.doi.org/10.1016/S0010-4485(03)00140-4.

[45] Fujiichi Yoshimoto, Toshinobu Harada, and Yoshihide Yoshimoto. Data fitting with a spline using a real-coded genetic algorithm. *Computer-Aided Design*, 35(8):751 – 760, 2003. URL http://dx.doi.org/10.1016/S0010-4485(03)00006-X.

[46] Xiuyang Zhao, Caiming Zhang, Bo Yang, and Pingping Li. Adaptive knot placement using a GMM-based continuous optimization algorithm in B-spline curve approximation. *Computer-Aided Design*, 43(6):598–604, June 2011. URL http://dx.doi.org/10.1016/j.cad.2011.01.015.

[47] Wenni Zheng, Pengbo Bo, Yang Liu, and Wenping Wang. Fast B-spline curve fitting by L-BFGS. *Computer Aided Geometric Design*, 29(7):448 – 462, 2012. URL http://dx.doi.org/10.1016/j.cagd.2012.03.004.

COLOPHON

This document was typeset using the typographical look-and-feel `classicthesis` developed by André Miede. The style was inspired by Robert Bringhurst's seminal book on typography "*The Elements of Typographic Style*". `classicthesis` is available for both LaTeX and LYX:

http://code.google.com/p/classicthesis/

Happy users of `classicthesis` usually send a real postcard to the author, a collection of postcards received so far is featured here:

http://postcards.miede.de/

Final Version as of May 8, 2018 (`classicthesis` 3.0).