Spline Extensions for the MAPLE Plot System

by

Wolfgang Heidrich

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 1995

Abstract

Traditionally computer algebra systems use lines and polygons to represent mathematical functions graphically. While these geometric primitives can easily be rendered on conventional raster graphics hardware, a smooth representation using splines would provide a wider range of trade-offs between image quality and rendering performance. Since modern computer graphics hardware directly supports rendering of spline objects, their use becomes more and more interesting.

In this thesis we examine the possibilities for replacing traditional representations of functions and graphs by spline representations. We describe the use of B-splines for interpolation and approximation, and discuss several approaches for generating parameterizations for these tasks. Finally we present some novel results regarding the use of rational splines for curve and surface fitting.

Acknowledgments

The work presented in this thesis would not have been possible without the help, supervision and encouragement so kindly offered by many people. In particular, I would like to thank my two supervisors Professor Richard Bartels and George Labahn for their constant support and guidance. I would also like to thank my two faculty readers Professor Hans-Peter Seidel and Professor Bruce Simpson, as well as my student reader Thomas Pflaum for their time and for all the useful remarks that improved the quality of this thesis.

I am grateful for the support and friendship from members of the Computer Graphics Lab that let me have such a great time in Waterloo. In particular I would like to thank Robert Berks, Fabrice Jaubert, Robert Kroeger, Sandra Loop, Andrew Park, Thomas Pflaum, Greg Veres and Julie Waterhouse for letting me feel at home in Waterloo.

Finally I thank my parents for their love, encouragement and understanding, without which I could not have done this.

Contents

List of Figures

Chapter 1

Introduction

Over the last few years the visualization of mathematical structures has evolved as an integral part of computer algebra systems. This development has been made possible by the introduction of inexpensive raster graphics hardware and graphical printing devices.

Current trends in printer technology allow for even higher resolutions and thus higher image quality. At the same time raster graphics hardware is becoming more sophisticated and instead of just storing pixel values, it now contains specialized hardware to render 2-dimensional and 3-dimensional geometric primitives itself. In many cases this results in drastically improved rendering performance and allows for more interactive applications.

Due to this development, the graphics subsystems of modern computer algebra systems need to support both high-quality, high-resolution graphics and high-performance, interactive graphics. Unfortunately, the traditional approach of representing graphs as piecewise lines and polygons is not well suited for this task, since the granularity of this approximation has to be adopted to the output device and the desired rendering quality and performance. Usually the user wants to perform other operations between the approximation step and the rendering of the graph. Unfortunately, these operations have to be repeated every time the quality parameters change.

Instead of using piecewise linear functions like lines and polygons, it would be better to represent the geometry with smooth higher order functions. A single representation in terms of smooth basis functions is then sufficient for rendering at a wide range of resolutions. The tradeoff between quality and performance can in this case be chosen at a later point in time by selecting an adequate precision for rendering the basis functions.

In this thesis we will describe the use of B-splines and rational B-splines (NURBS) in the computer algebra system MAPLE. We will describe the introduction of B-splines and NURBS as geometric primitives to the MAPLE plot system and its data structures, and will focus on their use for fitting discrete data points and representing arbitrary functions.

1.1 Splines and the MAPLE Plot System

One of the advantages of MAPLE over other computer algebra systems is the presence of a variety of efficiently implemented data structures, which results in a relatively high performance and low memory usage.

MAPLE is also a very flexible and expandable system due to its modular architecture. The core mathematical functionality is contained in the MAPLE *kernel*, a comparatively small program that is implemented in C. The kernel also contains the implementation of the data structures and an efficient interpreter for the MAPLE programming language [3].

This programming language has been used to implement the MAPLE library [4], which is actually a set of different packages that provide support for specific areas of mathematics. Packages for statistics, p-adic numbers and linear algebra exist, to name just a few. Most important for the computer graphics part of MAPLE are the `plots` and `plottools` packages, which provide functions for creating and manipulating 2- and 3-dimensional plots. It is these functions that create the piecewise linear representation of functions for rendering.

A process separate from the MAPLE kernel handles the user interface. This program, called the *Iris*, provides a level of abstraction between the platform-independent kernel and the user. It passes user input to the kernel and prints the results on the screen. If available, the Iris makes use of raster graphics for displaying formulas. However, it does not directly create 2D or 3D plots.

To create a plot with MAPLE the user typically uses one of the functions that are part of the library, most importantly `plot` and `plot3d`. These procedures store the approximation of the geometry in a data structure called the *plot data structure*. For the actual rendering, this data structure is handed over to the *plot driver*, which is a separate process in the MAPLE system. Each driver is capable of rendering a plot in a specific image format. Depending on the driver, this could either be for immediate display in a window system like X, or for storage in a specific file format if the image is to be displayed or printed later.

There are several reasons for keeping the kernel and the plot driver separated. Most importantly this approach allows adding new drivers for additional hardware or file formats without recompiling the kernel. The separation of kernel and driver also means that only those drivers that are actually in use occupy space in the main memory.

On the other hand, this separation makes it necessary to represent the geometry in terms of simple geometric primitives. Ideally, an exact representation of the function would be transmitted to the plot driver. The driver would then sample this function based on the hardware and on some quality requirements that the user provides. Unfortunately, an exact representation of arbitrary functions would require a lot of the symbolic functionality of the MAPLE kernel within the plot driver. This is not acceptable if we want to keep the kernel and the drivers separate.

The problem here is that the quality of the rendering is already determined in the MAPLE kernel, which has no information about the plot driver, the purpose of the plot, or potentially available hardware. It is, however, possible to approximate arbitrary functions with some smooth function that is easy to evaluate and is generated by a simple set of basis functions. If this approximation is good enough, the user can then customize the rendering quality by setting the sampling quality for the approximated function from within the driver.

An important class of functions with the properties mentioned above are piecewise polynomials. A well-known and frequently used basis for this class of functions are the so-called *basis splines* (B-splines). This idea can be extended to rational B-splines (NURBS) [1, 7, 9, 10]. These functions have mathematical properties that allow for efficient rendering and comparatively easy handling of the approximations.

One of the tasks that has to be done in order to implement spline support for MAPLE, is to add spline objects as geometric primitives to the plot system. Some drivers will then be able to render these objects using one of the existing direct rendering methods without further approximating the geometry. Other drivers will still sample these objects and render them as lines and polygons, but they will allow the user to adjust the number of sample-points to his needs, without having to quit the driver and recreate the geometry.

However, these changes are only a small part of what is necessary for full NURBS support. A more challenging problem is to re-implement the functions in the `plots` and `plottools` library packages so that they create a NURBS approximation of the geometry. In particular we require methods for approximating and interpolating discrete data points that are obtained by sampling smooth functions with integral and rational B-spline curves and surfaces.

1.2 Spline Interpolation and Approximation

The usual approach for interpolating a set of discrete points with B-splines involves two separate steps. Since B-spline curves and surfaces are parametric and defined over a certain parameter interval, the first step involves assigning an appropriate parameter value to every data point. This value determines the parameter at which the final B-spline curve will evaluate to the data point. The problem of finding appropriate values for a given set of data points is known as the "parameterization problem", and will be discussed in Chapter 4.

In the second step the actual interpolation problem is solved by creating a set of B-spline basis functions that is suitable for the data points and their parameter values, and then solving a linear equation system. As we will see in Chapter 3, the usual way of computing the basis function introduces additional degrees of freedom and results in an under-determined linear equation system. In order to make reasonable choices for these degrees of freedom, we can choose between a set of different "end conditions", which influence the shape of the curve or surface at the boundaries of the parameter domain.

If the data points are sampled from a relatively smooth function, it might not be necessary to compute a full interpolation solution. Instead, it is usually sufficient to generate an approximation of the data points with significantly less curve or surface segments than a full interpolation would require. This reduces the computation time and offers the possibility of smoothing the curve in cases where the data points are subject to sampling errors.

Like the interpolation algorithm, the approximation procedure also consists of two major parts. The first part, the parameterization problem, is identical to the interpolation case. In the second part, an approximation is obtained by first generating a set of basis functions, and then generating a spline curve or surface by minimizing the error between the spline and the data points according to some norm.

Traditionally the approximation problem for B-splines is solved using the least-squares (L_2) norm. However, in order to obtain specific approximation properties, it is reasonable to use different norms like L_1 and L_∞. In Chapter 5 we will describe how such minimizations can be implemented using a linear programming approach.

The vast majority of the literature about spline interpolation and approximation only considers integral B-splines as stated above. However, rational B-splines (NURBS) provide more degrees of freedom to control the shape of curves and surfaces, and could be used to further reduce the number of curve or surface segments or to increase the quality of a fit. In particular, NURBS can be used to create exact representations of conic sections.

The problem with using rational splines for approximation and interpolation tasks is to find a set of appropriate weights for the control points. A recent paper [17] describes how such weights can be obtained as a minimization process with respect to the L_2 error. The actual control points of the curve or surface can then be found using methods similar to those used in the integral case.

In Chapter 6 we will describe the original minimization process used for obtaining these weights, and propose similar methods for more efficient solutions using the L_1 and L_∞ norm. We will also describe how the well-known end conditions for integral spline interpolation can be used in the context of rational splines.

Finally, in Chapter 7 we will describe the implementation of these concepts in the environment of MAPLE, and present the achieved results. We will then conclude by giving pointers to future research on the topic.

Chapter 2

Preliminaries

Before we go into the details of spline interpolation and approximation, we will now define some terms and notations that will be used throughout the thesis.

Of particular interest are the *B-spline basis functions.* Given a degree d and a sequence of *knots* $T = (t_0, \ldots, t_{n+d+1})$ with $t_i \leq t_{i+1}$, we can recursively define a sequence of $n+1$ piecewise polynomial functions $N_i^d(u,T)$:

$$N_i^0(u,T) := \begin{cases} 1 & t_i \leq u < t_{i+1} \\ 0 & \text{otherwise} \end{cases} \quad ; \quad i = 0 \ldots n+d$$

$$N_i^k(u,T) := \frac{u - t_i}{t_{i+k} - t_i} N_i^{k-1}(u,T) + \frac{t_{i+k+1} - u}{t_{i+k+1} - t_{i+1}} N_{i+1}^{k-1}(u,T) \quad ; \quad i = 0 \ldots n+d-k$$

If the degree and the knot vector are clear from the context, we will usually use $B_i(u) := N_i^d(u,T)$ as a shorthand notation.

From the definition, it is clear that $N_i^d(u,T)$ is a piecewise polynomial of degree d, and has a support of $[t_i, t_{i+d+1})$. Other important properties, which can be found in the standard literature on splines (for example [1, 7, 9, 10]), are the fact that $N_i^d(u,T)$ is non-negative for all $u \in \mathbb{R}$, and that all functions sum up to 1 in the interval $[t_d, t_{n+1})$:

$$\sum_0^n N_i^d(u,T) = 1; \quad u \in [t_d, t_{n+1})$$

It can also be proven that in this interval the functions $N_i^d(u,T)$ are actually a basis for the function space of all piecewise functions that are polynomial of degree d between any two successive

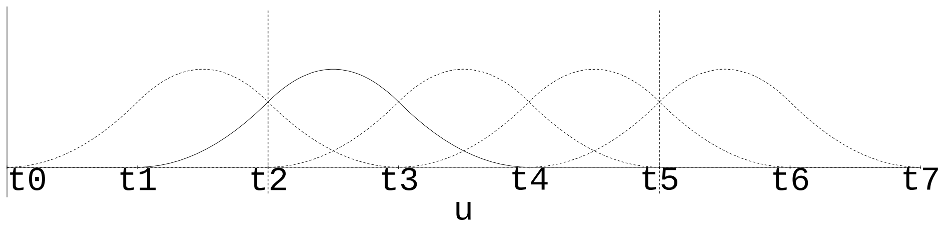

Figure 2.1: A sequence of 5 quadratic B-spline functions. The functions represent a basis for all piecewise quadratic functions that have a domain of $[t_2, t_5)$ and knots $(t_2, \ldots, t_5)$.

knots t_i, t_{i+1}. This important result has been found by Curry and Schoenberg in 1966 [5], and has led to the name "B-splines" as an abbreviation for "basis-splines" for these functions.

Figure 2.1 shows $n+1 = 5$ B-splines of degree $d = 2$ defined over a knot vector $T = (t_0, \ldots, t_7)$. Since each basis function has a support of $d+1 = 3$ intervals, we can see that at every parameter value u in the interval $[t_2, t_5)$ three of the basis functions are non-zero. The polynomials constituting these three basis functions at u are linearly independent. Hence they are capable of representing all quadratic polynomials at u.

2.1 Spline Curves and Surfaces

Based on these functions we can now define parametric curves and surfaces in spaces of arbitrary dimension. Given $n+1$ points $[c_0, \ldots, c_n]^T$ in this space, a B-spline curve can be defined as

$$F(u) := \sum_{i=0}^{n} c_i B_i(u) \tag{2.1}$$

The c_i are called "control points", because they are used to control the shape of the curve.

The following is a short list of those properties of B-spline curves that are most important to the topics covered in this thesis. Proofs for these properties can be found in the B-spline literature, including [1], [7] and [9].

- **Convex Hull Property.** For every parameter value u with $t_i \leq u < t_{i+1}$ the corresponding point on the curve $F(u)$ is in the convex hull of the control points $c_{i-d}, \dots, c_i$.

  This property leads to a method for rendering B-splines by iteratively refining the control polygon. It also helps us for generating reasonable bounding boxes for the curve.

- **Affine Invariance.** B-spline curves are invariant under affine transformations. This means, the image of a B-spline curve under an affine transformation is simply the B-spline curve generated by the images of the control points: $A\left(\sum_{i=0}^n c_i B_i(u)\right) = \sum_{i=0}^n A(c_i) B_i(u)$.

  This is important, because it allows us to fit data with B-splines without having to worry about the exact scaling and positioning of the final graph.

- **Continuity.** A B-spline curve is C^{d-1} continuous at every knot t_i with $t_{i-1} < t_i < t_{i+1}$. It is C^{d-l-1}-continuous at t_i with $t_{i-1} < t_i = t_{i+1} = \cdots = t_{i+l-1} < t_{i+l}$. In this case we say that the knot t_i has multiplicity l. Since B-splines are piecewise polygons, they are C^∞ continuous at every parametric position that is not a knot.

  Continuity is useful for generating optically smooth interpolations and approximations. Moreover we can model discontinuities by selecting an adequate knot sequence.

- **Local Control.** Since the basis functions have a support of $d+1$ intervals, every single point of the curve only depends on $d+1$ control points.

  For interpolations this means that a change of one of the data points will only affect the interpolant locally, segments that are far away from the point of change will not be affected. As we will see in Chapter 3, local control also means that the interpolation problem is local as well, and this leads to a well-conditioned, banded matrix for the interpolation problem.

The construction of surfaces on a rectangular parameter domain works similarly to the case of curves. We can define a set of basis functions using a *tensor-product* approach. For the two parametric directions u and v, we require separate degrees d_u, d_v and knot sequences T_u, T_v. With the shorthand notations $B_i^u(u) := N_i^{d_u}(u, T_u)$ and $B_j^v(v) := N_i^{d_v}(v, T_v)$ the basis is defined as follows:

$$B_{i,j}(u,v) := B_i^u(u)B_j^v(v)$$

Using these basis functions, we can then define a parametric surface $F(u,v)$ as

$$F(u,v) = \sum_{i=0}^{n_u}\sum_{j=0}^{n_v} c_{ij}B_{i,j}(u,v) = \sum_{i=0}^{n_u}\sum_{j=0}^{n_v} c_{ij}B_i^u(u)B_j^v(v) \tag{2.2}$$

Surfaces of this type are known as *tensor-product* B-spline surfaces. By construction it is clear that the properties for B-spline curves mentioned above also hold for tensor-product surfaces.

It should be mentioned at this point that surfaces with a rectangular parameter domain are not well suited for several applications like, for example, the interpolation of scattered data points. If such a surface is forced into a triangular shape, this will usually result in discontinuities. The development of spline surfaces over triangular parameter domains is a topic of ongoing research.

In the context of MAPLE and other computer algebra systems, however, tensor-product surfaces are usually sufficient, since these systems typically only evaluate functions over a rectangular domain anyway.

2.2 Rational Spline Curves and Surfaces

As we have stated in the previous section, B-spline curves and surfaces are invariant under affine transformations. Unfortunately, this is not in general true for arbitrary projective transformations, like the ones that are used to project 3-dimensional objects on a 2-dimensional image plane. However, it turns out that the image of every B-spline under such a transformation can be represented as a so-called *rational* B-spline [10].

The definition of a rational B-spline requires the introduction of a separate coefficient w_i, called *weight* for every control point. We can then write a rational B-spline curve as

$$F(u) = \frac{\sum_{i=0}^{n} w_i c_i B_i(u)}{\sum_{i=0}^{n} w_i B_i(u)} \tag{2.3}$$

In cases where the knot sequence is not restricted to be uniformly spaced, these curves are usually called "Non-Uniform, Rational B-Splines" (NURBS). If we have to distinguish between NURBS and B-splines as defined in the previous section, the latter will be called *integral* B-splines.

Every NURBS curve can be interpreted as the projection of a B-spline curve from a space with a higher dimensionality. For example, the 2-dimensional, rational B-spline curve with control points $c_i = (x_i, y_i)$ and weights w_i can be seen as the projective image of an integral B-spline curve in 3-dimensional space with control points $[w_i x_i, w_i y_i, w_i]$. This representation of the rational curve is also called the *homogeneous form.* We refer to [10] for an in-depth discussion of the relationship between the two representations.

An important property of NURBS, and the actual reason why we want to use them, is that they can exactly represent all conic sections, as well as all integral B-splines. They are therefore more powerful than integral B-splines and can be used to specify a larger variety of shapes.

The same principle that led to the definition of rational spline curves, can also be applied to tensor-product surfaces. In this case we require a whole mesh of weights $w_{i,j}$, one for each control point $c_{i,j}$. This leads to tensor-product NURBS surfaces:

$$F(u,v) = \frac{\sum_{i=0}^{n_u} \sum_{j=0}^{n_v} w_{ij} c_{ij} B_i^u(u) B_j^v(v)}{\sum_{i=0}^{n_u} \sum_{j=0}^{n_v} w_{ij} B_i^u(u) B_j^v(v)} \tag{2.4}$$

Chapter 3

B-Spline Interpolation

Using the definitions from the previous chapter, we can now formulate the interpolation problem for curves, first using integral B-splines.

Suppose we are given a set of $n+1$ data points $D = [d_0, \ldots, d_n]^T$. Let us further assume that we are also given a parameter value u_i for every data point d_i. If the parameter values are not provided with the data points, we can use heuristics to generate reasonable choices for the parameterization. This will be described in Chapter 4.

Given the points and parameter values, we are searching for a B-spline curve of degree d that interpolates every data point d_i at the corresponding parameter value u_i. Since the $n+1$ data points constitute an interpolation problem with $(n+1) \times e$ degrees of freedom, where e is the dimension of the space, in general we need a spline curve with exactly the same degrees of freedom. In other words, we need a B-spline curve with $n+1$ control points.

These control points correspond to a set of $n+1$ B-spline basis functions, which in turn are determined by a knot sequence T of length $n+d+2$. In Section 3.2 we will discuss how a knot sequence can be selected from the parameterization of the data points. For the moment we assume that both the knots and the basis functions $(B_0(u), \ldots, B_n(u))$ are already available.

As we have seen in the previous chapter, we can write any B-spline curve as

$$F(u) = \sum_{i=0}^{n} c_i B_i(u),$$

where the control points c_i are to be determined in such a way that the curve interpolates the data points. This means that the curve has to satisfy the conditions

$$\sum_{i=0}^{n} c_i B_i(u_k) = d_k \quad ; \quad k = 0 \dots n \quad . \tag{3.1}$$

Note that each of these conditions is actually a set of e independent equations, one for each vector component in the space.

We can rewrite the system of equations 3.1 to get the following matrix form:

$$\underbrace{\begin{bmatrix} B_0(u_0) & B_1(u_0) & \cdots & B_n(u_0) \\ B_0(u_1) & B_1(u_1) & \cdots & B_n(u_1) \\ \vdots & \vdots & \ddots & \vdots \\ B_0(u_n) & B_1(u_n) & \cdots & B_n(u_n) \end{bmatrix}}_{\mathbf{B}} \cdot \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} d_0 \\ d_1 \\ \vdots \\ d_n \end{bmatrix} \tag{3.2}$$

If we have selected the basis functions such that for all $i = 0 \dots n$ the parameter value u_i lies within the support of basis function $B_i(u)$, the square matrix $\mathbf{B} := (B_i(u_k))_{i,k}$ is of full rank. This result is known as the theorem from Schoenberg and Whitney ([7, 6]). Because of the local control property of B-spline curves, the matrix is also banded (the width of the band is $d + 1$, the number of basis functions that are non-zero at a parameter value) and well-conditioned.

Thus, the interpolation problem comes down to independently solving e linear equation systems of the form of 3.2, one for each vector component in the space. To obtain the solution, we can either use Gaussian elimination or some specialized algorithm that makes use of the special structure of $\mathbf{B}$.

3.1 Tensor-Product Surface Interpolation

With a similar approach it is possible to solve the interpolation problem for tensor-product surfaces. Assuming we are given a $(n_u + 1) \times (n_v + 1)$ grid of data points, and we have somehow found two sequences $(u_0, \dots, u_{n_u})$ and $(v_0, \dots, v_{n_v})$ so that every data point $d_{i,j}$ is assigned a parameter value (u_i, v_j). As in the case of curves, we will also assume that we have already found knot sequences T_u and T_v and the corresponding basis functions $B_i^u(u)$ and $B_j^v(v)$.

Using the formula for tensor-product B-spline surfaces

$$F(u, v) = \sum_{i=0}^{n_u} \sum_{j=0}^{n_v} c_{i,j} B_i^u(u) B_j^v(v)$$

we can set up a system of $(n_u+1)(n_v+1)$ equations with the same amount of unknowns.

$$\sum_{i=0}^{n_u}\sum_{j=0}^{n_v} c_{i,j} B_i^u(u_k) B_j^v(v_l) = d_{k,l}. \tag{3.3}$$

A matrix representation of this system can be generated using the following definitions

$$\mathbf{B} := (b_{i,j}) \quad \text{where} \quad b_{jn_u+i, ln_u+k} := \mathbf{B}_i^u(u_k)\mathbf{B}_j^v(v_l),$$

$$C := [c_{0,0}, \dots, c_{n_u,0}, \quad \dots \quad , c_{0,n_v}, \dots, c_{n_u,n_v}]^T$$

and

$$D := [d_{0,0}, \dots, d_{n_u,0}, \quad \dots \quad , d_{0,n_v}, \dots, d_{n_u,n_v}]^T.$$

The control points for an interpolating spline surface can then be obtained by solving

$$\mathbf{B} \cdot C = D. \tag{3.4}$$

Due to the local control property, the matrix $\mathbf{B}$ is relatively sparse. However, it is not banded as in the case of curves.

Although it is theoretically possible to obtain a solution by solving the above system, this is in practice not feasible for larger interpolation problems. For example, a problem with 100×100 data points would involve computing the inverse of a 10000×10000 matrix, which could take several hours on current workstations.

Fortunately, the special structure of tensor-product surfaces allows for a more compact representation of the problem. By re-positioning the terms in Equation 3.3, we can see that the interpolation can actually be split into two separate processes:

$$\sum_{i=0}^{n_u} B_i^u(u_k) \left[\sum_{j=0}^{n_v} c_{i,j} B_j^v(v_l) \right] = d_{k,l}$$

Note that this is only possible because for every row of points in the data grid there exists only a single u parameter value, and for every point in a column there is only a single v. In a more general setting every data point could have its own u and v coordinates $(u_{i,j}, v_{i,j})$. In this case the above restructuring is not possible.

In matrix form the above system can be written as

$$\mathbf{B}_u \cdot (\mathbf{C} \cdot \mathbf{B}_v^T) = \mathbf{D}, \tag{3.5}$$

where $\mathbf{B}_u := (B_i^u(u_k))$, $\mathbf{B}_v := (B_j^v(v_l))$, $\mathbf{C} := (c_{i,j})$ and $\mathbf{D} := (d_{k,l})$. A solution can be obtained by solving the two systems $\mathbf{B}_u\mathbf{X} = \mathbf{D}$ and $\mathbf{B}_v\mathbf{C}^T = \mathbf{X}^T$ of size $(n_u+1) \times (n_u+1)$ and $(n_v+1) \times (n_v+1)$. As in the case of curves, the matrices B_u and B_v are banded and well-conditioned if we have selected appropriate basis functions. It is therefore possible to use optimized solution methods.

3.2 Knot Selection and Basis Functions

So far we have seen that the interpolation problem reduces to solving a system of linear equations, once an appropriate knot sequence and the corresponding basis functions have been found. By "appropriate" we mean that the matrix $\mathbf{B}$ from Equation 3.2 as well as the matrices $\mathbf{B}_u$ and $\mathbf{B}_v$ from Equation 3.5 are non-singular.

As we have stated above, we can guarantee this by making sure that for every $i = 0 \ldots n$, the data point i is located in the support of basis function $B_i(u)$. For curves this means mathematically that every data point d_i has a parameter value that corresponds to a position within the support $[t_i, t_{i+d+1})$ of B_i.

In the case of tensor-product surfaces we have to select two knot sequences T_u and T_v, and the above property has to independently hold for both knot sequences.

The consequence of these observations is that in general we have to take the parameterization into account when deciding upon a knot sequence. Alternatively, we could create a knot sequence first, and then select a parameterization based on these knots. However, since there is a one-to-one correspondence between the data points and the parameter values, it is easier to come up with a reasonable parameterization first, and then select the knots afterwards.

Because the knot sequence has to depend on the parameterization, we can not simply choose B-splines with uniformly distributed knots. In addition, since the positioning of the knots has a substantial influence on the shape of the interpolant, it is important to use a method that produces "good" interpolations in most cases, or at least in the most common cases. Figure 3.1 shows how strong the influence of the knot selection algorithm on the curve shape can be.

Most of the existing literature about B-spline interpolation does not address this issue at all but rather quietly assumes that the knots coincide with the parameter values for the data points. This method seems to produce reasonable results for most interpolation problems. Since the knots in a B-spline are exactly those positions where the curve or surface are only C^{d-1} continuous, we can

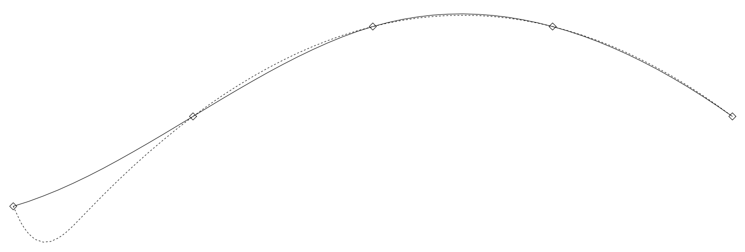

Figure 3.1: The effect of different knot sequences on solutions for an interpolation problem. Both curves are interpolants of the given data points, and use the same parameterization but different knot sequences.

interpret this method as giving the spline more flexibility around the points we want to interpolate.

When implementing this method, we have to pay attention at the borders of the domain. Suppose we have $m+1$ data points together with the corresponding parameter values. If we use these parameter values as knots, this will give us $m+1$ knots. However, in Chapter 2 we have stated, that the B-splines over a knot sequence $T = (t_0, \ldots, t_{n+d+1})$ only forms a basis for the interval $[t_d, t_{n+1})$.

So, in order to have a basis on the complete parameter interval $[u_0, u_m)$, we have to generate an additional d knots to the left of u_0 and d knots to the right of u_m. A good choice for these extra knots is usually to set all the knots on the left to u_0 and all knots on the right to u_m. The result is a B-spline that passes through its first and its last control point.

$$T := (\underbrace{u_0, \ldots, u_0}_{d+1}, u_1, \ldots, u_{m-1}, \underbrace{u_m, \ldots, u_m}_{d+1})$$

This leaves us with $(m+1)+2d$ knots. We also know that a knot sequence of length $(n+1)+d+1$ gives us $n+1$ basis functions, and hence allows for a B-spline curve with $n+1 = (m+1)+d-1$

control points.

Thus our B-spline curve has $d-1$ more control points than data points, and the interpolation problem becomes the following under-determined system of linear equations:

$$\begin{bmatrix} B_0(u_0) & B_1(u_0) & \cdots & B_n(u_0) \\ \vdots & \vdots & \ddots & \vdots \\ B_0(u_m) & B_1(u_m) & \cdots & B_n(u_m) \end{bmatrix} \cdot \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} d_0 \\ \vdots \\ d_m \end{bmatrix}$$

The extra degrees of freedom introduced by this approach can be used to enforce additional constraints on the curve. There are a variety of possibilities for setting these constraints, which are usually called "end conditions". The choice of a particular end condition depends on the specific application for which the interpolant is required.

Since for tensor-product surfaces the knot selection can be done separately in u and v direction, we can also directly use the parameterization as knots for the two parametric directions. The result is a surface with an extra $d_u - 1$ control points in u direction and an extra $d_v - 1$ control points in v direction.

3.3 End Conditions

The name "end condition" comes from the fact that the extra degrees of freedom are usually filled in at the borders of the parameter domain, that is, at the ends of the curve or surface. It would also be possible to specify additional conditions in the interior of the spline, as long as those conditions are linearly independent from the rows in the matrix $\mathbf{B}$. In practice, however, constraints at the borders of the domain are the most interesting, as they allow us to smoothly join different curves or surfaces.

In the case of curves, a simple way of generating a smooth transition between two separate splines is to specify the tangent vectors in the endpoints and set them to the same vector for both curves. Setting the extra degrees of freedom by specifying tangent vectors is called the *clamped end condition*. In the case where the spline curve is of degree $d = 2$, we have $d - 1 = 1$ extra control points available, and can therefore force the tangent on one side of the curve to a specific value. If we have a curve of degree 3, we can use the two extra control points to set the tangents on both sides of the curve. For even higher degrees we can use a generalization of the clamped end conditions which allows us to specify higher order derivatives on one or both sides of the curve.

In order to implement clamped end conditions, we need to compute the derivative of a B-spline curve at the endpoints, and choose the available control points to force this vector to a given tangent vector ξ. It is a well-known fact (see Appendix A) that the derivative of a B-spline curve $F(u)$ is given by

$$\frac{\mathrm{d}}{\mathrm{d}u}F(u) = d\sum_{i=0}^{n+1}\frac{(c_i - c_{i-1})}{(t_{i+d} - t_i)}N_i^{d-1}(u,T).$$

We can see that the derivative is a linear combination of the control points of the curve. If we re-group the terms to reflect this observation, and set the derivative at some parameter t to ξ, we get

$$\xi = \sum_{i=0}^{n}\underbrace{\left[\frac{d}{t_{i+d} - t_i}N_i^{d-1}(t,T) - \frac{d}{t_{i+d+1} - t_{i+1}}N_{i+1}^{d-1}(t,T)\right]}_{e_i}\cdot c_i.$$

This equation can be rewritten in matrix form, which yields

$$\begin{bmatrix} e_0 & e_1 & \dots & e_n \end{bmatrix}\cdot\begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{bmatrix} = \xi.$$

In order to fill in the extra degrees of freedom we have available, we add a number of equations of the above form to the matrix $\mathbf{B}$ from Equation 3.2. For degrees $d > 3$ we will have to move to higher order derivatives, but these can be fit into our linear equation system in a similar way. The result is an equation system of the form

$$\begin{bmatrix} \mathbf{E}_l \\ \hline \mathbf{B} \\ \hline \mathbf{E}_r \end{bmatrix}\cdot\begin{bmatrix} C \end{bmatrix} = \begin{bmatrix} \Xi_l \\ \hline D \\ \hline \Xi_r \end{bmatrix} \tag{3.6}$$

$\mathbf{E}_l$ is a matrix holding the derivatives for the left side of the curve, and Ξ_l is a vector of the corresponding tangents. Similarly $\mathbf{E}_r$ and Ξ_r describe the end conditions on the right side of the curve.

In the next section we discuss a set of end conditions that have often been used in the standard spline literature. All these constraints have the property of being linear in the control points, and can be represented in a matrix form similar to the way we have discussed above.

3.3.1 Bessel End Conditions

For many applications it is desirable to have the system spend the degrees of freedom automatically instead of letting the user supply additional information. This is especially true if the user does not know the data set well enough to make a good choice. The approach of *Bessel end conditions* is to generate a set of tangent vectors automatically, which can then be plugged in to the clamped end conditions described above.

More specifically, the vectors that are created are the tangents of the interpolating parabola through the first (or last) three data points. This (parametric) parabola is found by solving the system of equations

$$\begin{aligned} a \cdot u_0^2 + b \cdot u_0 + c &= d_0, \\ a \cdot u_1^2 + b \cdot u_1 + c &= d_1, \\ a \cdot u_2^2 + b \cdot u_2 + c &= d_2. \end{aligned}$$

The tangent vector ξ we are searching for is then given by the derivative of this parabola at parameter value u_0, that is,

$$\xi = 2a \cdot u_0 + b$$

3.3.2 Natural End Conditions

Another simple choice for end conditions of quadratic or cubic splines are the *natural end conditions*, which set the curvature in the endpoints of the curve to zero:

$$\ddot{F}(u_0) = \ddot{F}(u_n) = 0$$

In the case of a quadratic spline we have of course only enough degrees of freedom to set the curvature in one of the points.

This method involves computing the second derivative of a B-spline curve, which is again a linear combination of the control points. The name “natural” for these constraints originates from the field of ship construction. In this field the word "spline" is used to describe wooden planks that are used to determine the shape of a ship body. When such a plank is fixed to pass through a set of some specific points, it takes on a shape in which the bending energy in the two outermost points is zero. Mathematically this is corresponds to the curvature being zero.

3.3.3 Quadratic End Conditions

The next kind of end conditions we want to describe are called *quadratic end conditions*. Traditionally these have been used for degree 3 curves only, but generalizations to curves of other degrees are obvious. For a degree 3 curve we make sure that the second derivatives in the two leftmost data points match, as well as the second derivatives in the two rightmost data points.

$$\ddot{F}(u_0) = \ddot{F}(u_1) \quad \text{and} \quad \ddot{F}(u_{n-1}) = \ddot{F}(u_n)$$

Since we identify the knots with the parameter values of the data points as described in Section 3.2, the two parameter values on each side are also knots of the spline.

For cubic curves the second derivative is a piecewise linear, C^0 continuous function. Consequently, if we set $\ddot{F}(u_0) = \ddot{F}(u_1)$, we actually force $\ddot{F}(u)$ to be constant on the interval $[u_0, u_1]$. Therefore the spline $F(u)$ is actually quadratic instead of cubic on this segment, and hence the name of this method. The same applies similarly for the interval $[u_{n-1}, u_n]$.

Quadratic end conditions can again be described as a linear combination of the control points of the B-spline curve, and can therefore be written in matrix form.

3.3.4 Not-A-Knot End Conditions

A slightly more complicated set of constraints is represented by the so-called *not-a-knot end conditions*. Instead of spending the extra degrees of freedom on reducing the degree of a segment, this method merges two spline segments into a single one, thus increasing the continuity at the knot that separates the two segments.

Again we identify knots with parameter values and request that the d^{th} derivatives at certain parameter values u_i are continuous. In the cubic case we require two of these points, and typically choose u_1 and u_{n-1}. Now, since we are dealing with a spline of degree d, the d^{th} derivative $F^{(d)}(u)$ is a piecewise constant function. A simple condition for making this function continuous at a parameter value u_i is to request that the values at the centers of the two intervals $[u_{i-1}, u_i]$ and $[u_i, u_{i+1}]$ are the same.

$$F^{(d)}(\frac{u_{i-1}+u_i}{2}) = F^{(d)}(\frac{u_i+u_{i+1}}{2})$$

For a curve of degree d we have to remove $d-1$ knots in this way to get rid of all the extra degrees of freedom.

Thus we have again reduced the constraints to a condition on higher order derivatives, which are linear functions in the control points.

3.3.5 Closed End Conditions

Often we want to create closed spline curves from a set of data points. If those points are given such that $d_0 = d_n$, we can use *closed* end conditions to make the two ends of the curve join together smoothly. Since we have $d-1$ additional control points available, and an interpolating curve for data points with $d_0 = d_n$ is C^0 continuous automatically, we can use the extra degrees of freedom to achieve C^{d-1} continuity. This is also the degree of continuity we have at the internal knots of the curve. In other words, we can create a closed spline curve without a distinguished start and end point.

The conditions for C^{d-1} continuity at the joint are obviously that the first $d-1$ derivatives at the beginning and the end of the curve match:

$$F^{(k)}(u_0) = F^{(k)}(u_n) \quad ; \quad k = 1..d-1$$

3.3.6 Multiple Control Point End Conditions

For the sake of completeness, we shall finally mention a very simple end condition that can be used if no information about the data is available. The additional control points are set by letting multiple control points coincide. This is usually also done at the end points of the curve.

The resulting constraints are again linear in the control points, but they do not require the derivative of the B-spline. Thus this method is very efficient to implement, since every constraint is represented as a fixed vector that does not depend on the data values. For the more complex methods based on derivatives, on the other hand, each vector has to be created using both the formula for derivatives of B-splines, and the B-spline recursion formula.

3.4 End Conditions for Tensor-Product Surfaces

So far we have concentrated on end conditions for spline curves. As we have seen in Section 3.2, the situation for tensor-product surfaces requires additional constraints in a similar fashion. It turns out that all the end conditions from the previous section can also be applied in the case of tensor-product splines. However, some extra care has to be taken for the actual implementation of these constraints.

In the case of curves we were able add the end conditions directly to the interpolation matrix $\mathbf{B}$, and thus to rewrite the linear equation system in Equation 3.2 to the one in Equation 3.6. Our first attempt to handle end conditions for surfaces is to use a similar approach. The equation system we gain by rewriting Equation 3.5 is

$$\begin{array}{|c|}\hline \mathbf{E}_t \\ \hline \mathbf{B}_u \\ \hline \mathbf{E}_b \\ \hline\end{array} \cdot \begin{array}{|c|}\hline \mathbf{C} \\ \hline\end{array} \cdot \begin{array}{|c|c|c|}\hline \mathbf{E}_l^T & \mathbf{B}_v^T & \mathbf{E}_r^T \\ \hline\end{array} = \begin{array}{|c|c|c|}\hline & \Xi_t & \\ \hline \Xi_l^T & \mathbf{D} & \Xi_r^T \\ \hline & \Xi_b & \\ \hline\end{array}$$

Note that we now have end conditions for the left, right, top and bottom side of the surface.

Unfortunately it turns out that most of the end conditions we discussed in the previous section do not fit into this scheme. In particular, all the end conditions involving derivatives can not be handled. This leaves us with the rather simplistic multiple control point end conditions as the only alternative.

To understand why this is the case, consider the simple case of clamped end conditions. In the above scheme we have only a simple row vector E in the matrix $\mathbf{E}_t$ available to represent the derivative in a whole row of data points. However, the derivative of the surface is a different vector E in each of the data points. Thus the end condition can not be represented in this scheme. A similar argumentation shows that all other end conditions involving derivatives can not be represented in this scheme either.

Since the end conditions are linear in the control points even in the tensor-product case, we can still create a matrix representation of the constraints. For example the constraints at the top side of the surface can be written as

$$\mathbf{E}_t \cdot C = \Xi_t$$

where C is defined as

$$C := [c_{0,0}, \ldots, c_{n_u,0}, \quad \ldots \quad, c_{0,n_v}, \ldots, c_{n_u,n_v}]^T,$$

like in Equation 3.4. We thus discover that we can fit the end conditions into the scheme of Equation 3.4 as

$$\begin{bmatrix} \mathbf{E} \\ \hline \mathbf{B} \end{bmatrix} \cdot \begin{bmatrix} C \end{bmatrix} = \begin{bmatrix} \Xi \\ \hline D \end{bmatrix}$$

Here $\mathbf{B}$ and D are defined as in Equation 3.4, while $\mathbf{E}$ and Ξ contain all the end conditions for the left, right, top and bottom sides. We have already mentioned that the total size of this system is very big, and solving it is therefore a timeconsuming task.

In order to cut down on execution time, we can obtain the control points in two steps. First we solve the underdetermined equation system

$$\mathbf{B}_u \cdot (\mathbf{C}' \cdot \mathbf{B}_v^T) = \mathbf{D}.$$

this gives us a partial solution $\mathbf{C}'$ that still contains some undetermined variables. These can then be filled in by solving

$$\mathbf{E} \cdot C = \Xi$$

afterwards. Since the number of end conditions is usually small compared to the number of basis functions, this method is typically significantly faster than directly solving the big equation system.

In the case of a computer algebra system like MAPLE, this approach is also relatively easy to implement, since the symbolic engine of the computer algebra system can be used to obtain the partial solution $\mathbf{C}'$.

Chapter 4

Parameterization

When we formulated the interpolation problem in the previous chapter, we assumed that an appropriate parameterization is provided together with the data points. In the context of a computer algebra system, this is actually not a bad assumption, since most of the data sets consist of points that are sampled from mathematical functions. For these points the exact parameter value (i.e. the position at which the original function was evaluated to yield the data point) is available, and can be directly used for the fitting process. Consider following example using the MAPLE library function `spacecurve` from the `plots` package.

```
spacecurve( [10*cos(t/30), 10*sin(t/30), t/3], t=0..240 );
```

This line of MAPLE code creates a 3-dimensional plot of a helix by sampling the function at discrete parameter values and then connecting the sample points with line segments. A spline version of this function could also sample the function at discrete parameter values. Possibly these points would be uniformly distributed, but the selection of the sample points could also be more involved. The important part is that the function `spacecurve` knows the parameter at which the samples are located, and can consequently use them to do an interpolation as described in Chapter 3, or an approximation, as we will see in Chapter 5.

However, `spacecurve` also has a second mode, in which it is only passed a set of points instead of a function. Using this mode we can write the above example as

```
helixPoints:= [seq( [10*cos(r/30), 10*sin(r/30), r/3], r=0..240 )];
spacecurve( helixPoints );
```

While this version of `spacecurve` is generally not used as often as the first one, it has some important applications. Similarly, most other MAPLE library functions in the `plots` and `plottools` packages also have corresponding discrete versions in which only a set of sampled data points is passed to the function.

If we want to create a spline-based version of these discrete functions, we have to generate parameter values for the data points before we can apply an interpolation or approximation method. The exact choice we make for the parameter values has a strong influence on the shape of the resulting interpolant. In this chapter we will present several heuristic methods for determining parameterizations that produce a "good" interpolant. More examples comparing the different methods can then be found in Chapter 8. All methods for determining the parameterization where originally designed for curve interpolation and approximation. In Section 4.7 we will then describe how these methods can be applied to tensor-product surfaces.

4.1 Uniform Parameterization

The easiest way of choosing a parameterization is the *uniform parameterization*, which simply divides the parameter interval into equally spaced segments. Clearly it can not be expected that this simplistic method works well for arbitrary sets of data points, since the particular position of the points is not even considered. An example of a data set taken from [11] and interpolated using the uniform parameterization scheme is shown in Figure 4.1.

The biggest advantage of this method is that the interpolant is a uniform B-spline if we use the parameter values also as knots, as described in Section 3.2. This might be preferable in situations where B-splines with arbitrary knot sequences cannot be implemented for some reason.

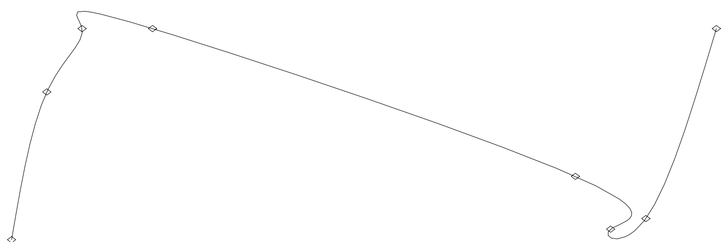

Figure 4.1: A data set interpolated with a cubic B-spline using the uniform parameterization. Note the artifacts between the 3rd and the 4th point as well as between the 5th and the 6th point.

4.2 Chord Length Parameterization

The so-called *cord length parameterization* is probably the easiest parameterization method that actually takes the data points into account. It tries to resemble the "arc length parameterization", that is, to choose the parameter values in such a way that they reflect the length of the curve segments of the interpolant between the data points. Since in general it is not possible, or at least infeasible, to find the exact arc length parameterization, the cord length parameterization approximates the arc length using the distance of the data points (the length of the "chords" connecting all the data points).

More formally, the distance between the knots u_{i-1} and u_i is made proportional to the Euclidean distance of the data points d_{i-1} and d_i:

$$u_i - u_{i-1} = k \cdot \|d_i - d_{i-1}\|_2 \quad ; \quad i = 1 \ldots n$$

The constant k can be chosen arbitrarily, as can the first parameter value u_0. If we set $u_0 := 0$ and

$$k := \frac{1}{\sum_{i=1}^{n} \|d_i - d_{i-1}\|_2},$$

the resulting parameters u_i will be in the interval $[0, 1]$.

In general the chord length parameterization produces better results than the uniform parameterization. It has been proven that the slope of an interpolating B-spline using the chord length

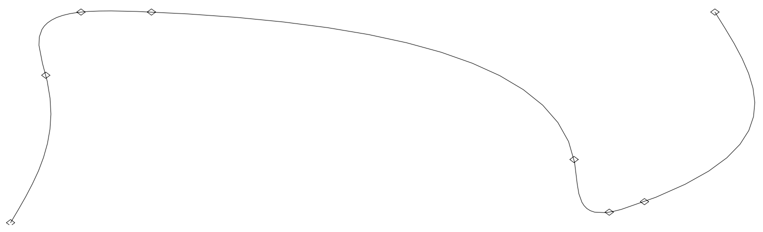

Figure 4.2: The chord length parameterization has been applied to the data set from Section 4.1. For this specific data set the results are in some sense even worse than with uniform parameterization.

parameterization varies continuously over the curve [8]. In fact this holds for all parameterization schemes that can be expressed as

$$u_i - u_{i-1} = D(d_i, d_{i-1}), \tag{4.1}$$

where $D(.,.)$ is some metric. A similar result for the uniform parameterization is not known.

There are, however, data sets for which the chord length parameterization actually performs worse than the uniform parameterization. One example is the data set from the previous section. Figure 4.2 shows an interpolation of the same data set using the chord length method. We can see that the interpolant is optically smoother, but has even larger wiggles than the curve generated with the uniform parameterization.

In the case of this particular data set, the shape of the curve can be slightly improved by using a different metric D for measuring the distance between the data points. A metric which on average makes the chord length algorithm perform better is described in Section 4.6. This does, however, not solve the general problem that parameterizations that are deemed superior can perform worse on specific data sets.

4.3 Centripetal Parameterization

The *centripetal parameterization* is a heuristic method which is very similar to the chord-length parameterization, but is inspired by a physical model. The author of [15], the original paper about the centripetal parameterization, challenges the idea of using an approximation of the arc length parameterization for interpolation. He uses the analogy of a car travelling on the interpolant to explain the reasons for his objections.

If we succeed in generating an interpolant with the exact arc length parameterization, this means that a car travelling on the curve will have constant speed, independent of the shape of the curve. The idea behind the centripetal parameterization is to slow the car down in areas where the curve has a high curvature. Based on some physical computations and the assumption that the centripetal force should be kept proportional to the angular change of the curve, the following formula can be derived for the parameter values:

$$u_i - u_{i-1} = k \cdot \|d_i - d_{i-1}\|_2^{1/2} \quad ; \quad i = 1 \ldots n.$$

In order to normalize the parameter values to the interval $[0, 1]$, we can set $u_0 := 0$ and

$$k := \frac{1}{\sum_{i=1}^{n} \|d_i - d_{i-1}\|_2^{1/2}}.$$

As we can see, the centripetal parameterization is very similar to the chord length parameterization. The only difference is that the square root of the Euclidean distance is used in the centripetal case. This suggests that it might be useful to explore different exponents as well. This results in

$$u_i - u_{i-1} = \frac{\|d_i - d_{i-1}\|_2^e}{\sum_{i=1}^{n} \|d_i - d_{i-1}\|_2^e} \quad ; \quad i = 1 \ldots n$$

where e is some constant exponent. The author of [15] writes that different values of e seem to be optimal for different sets of data points, but that an exponent of $1/2$ is in general better than an exponent of 1. So far, no method for automatically selecting an optimal e is known.

Figure 4.3 shows an interpolation of the data set using the centripetal parameterization with an exponent of $e = 1/2$. The resulting interpolant is significantly tighter and smoother than the two interpolants using uniform and chord length parameterizations.

Since $\|.\|_2^e$ is not a norm for $e \neq 1$ (the homogeneity criterion $\|\alpha \cdot x\| = |\alpha| \cdot \|x\|$ does not hold), we can not express the chord length parameterization as a parameterization based on a metric in

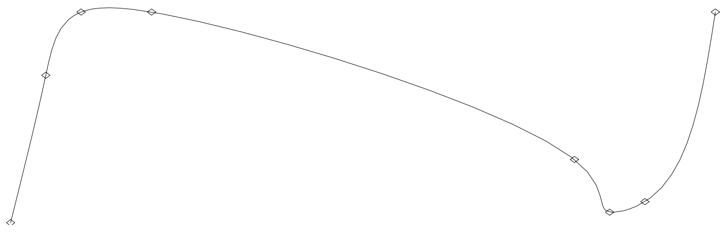

Figure 4.3: The centripetal parameterization yields better results than both the uniform and the chord length parameterizations.

the sense of Equation 4.1. Thus the proof in [8] can not be used to show that interpolants created with centripetal parameterization have a continuous slope. In practice, however, this seems to be the case anyway.

4.4 Angular Parameterization

Following the same idea that the parameter values should "slow down" in regions of high curvature, the authors of [11] introduce a purely heuristic method. The algorithm, which we will call *angular parameterization*, takes into account both the angle between the chords connecting the data points, and the length of these chords. We introduce the shorthand notations Θ_i for the external angle between the chords at data point d_i, and Δ_i for the length of the chord connecting d_i and d_{i+1}, as depicted in the following diagram:

We also define θ_i as an abbreviation for $\min(\Theta_i, \pi/2)$, and then the angular parameterization is given as

$$u_{i+1} - u_i = \Delta_i \left[1 + \frac{k \cdot \theta_i \cdot \Delta_{i-1}}{\Delta_{i-1} + \Delta_i} + \frac{k \cdot \theta_{i+1} \cdot \Delta_{i+1}}{\Delta_i + \Delta_{i+1}} \right] \quad ; \quad i = 0 \ldots n-1 \tag{4.2}$$

Note that this formula requires two extra chord lengths Δ_{-1} and Δ_n, as well as two extra angles Θ_0 and Θ_n. The authors of [11] do not specify how these parameters should be chosen. One possibility

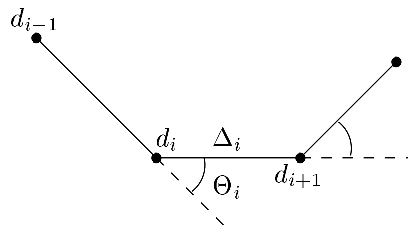

Figure 4.4: The configuration for the angular parameterization.

is to extrapolate the two outermost segments by setting $\Delta_{-1} = \Delta_0$, $\Delta_n = \Delta_{n-1}$ and $\Theta_0 = \Theta_n = 0$.

The distance between u_{i-1} and u_i is the larger the bigger the external angle Θ_i is, thus slowing the curve down, and giving it more time to turn. Please note that we use the angle θ_i instead of the real external angle. The upper limit of $\pi/2$ was determined empirically, and was introduced in order to prevent the curve from slowing down too much.

The influence of the chord lengths Δ_i is chosen such that the distance between two parameter values u_{i-1} and u_i is bigger the longer chord Δ_i is in comparison to the neighboring chords Δ_{i-1} and Δ_{i+1}.

The parameter k can either be used as a shape control in interactive applications, or should be set to the constant 1.5, which, according to the authors of [11] constitutes a good value for a large variety of data sets. This constant has again been chosen empirically.

Despite the fact that no theoretical foundation has been given for this algorithm, and some of the choices made seem quite arbitrary, the angular parameterization method in practice produces very good results. It usually outperforms both the chord length and the centripetal parameterization. The interpolant for our data set using the angular parameterization is shown in Figure 4.5. For this data set the result is actually very similar to the centripetal parameterization. Other examples, in which the differences between the parameterization methods are clearly visible, will be shown in Chapter 8.

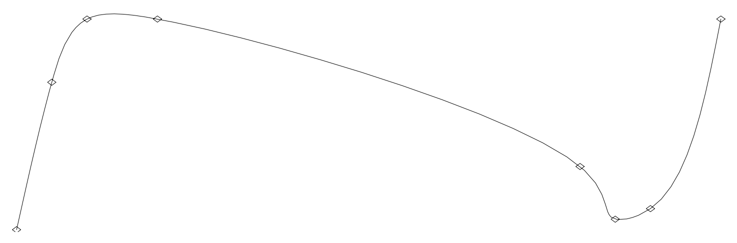

Figure 4.5: The resulting interpolant using the angular parameterization for this data set is comparable to the centripetal solution.

4.5 Area Based Parameterization

While the angular parameterization algorithm performs well in practice, it is somewhat unsatisfactory that no theoretical justification is given for this algorithm. We found it particularly disturbing that Formula 4.2 involves the multiplication of angles with distances, which makes it hard to come up with a geometric interpretation for the method.

We tried to overcome the aforementioned shortcomings by performing our own experiments, and to come up with a method that is based on principles similar to those of the angular parameterization, but with a reasonable geometric interpretation. The result is a method which we will call *area based parameterization*.

In analogy to the angular parameterization, we will consider two different contributing factors for selecting the parameter values: the distance Δ_i between two consecutive data points, and the degree to which the curve has to change direction while passing through these data points. The first contribution is simply given by the chord length Δ_i, which is normalized by dividing through the *average chord length* $\Delta := 1/n \sum_{i=0}^{n-1} \Delta_i$ of the data points:

$$h_i^1 := \frac{\Delta_i}{\Delta}$$

The normalization is necessary to keep the parameterization independent from the scaling of the data set.

For the contribution of the second part we consider the area of the parallelogram formed by two consecutive chords, as depicted in Figure 4.6.

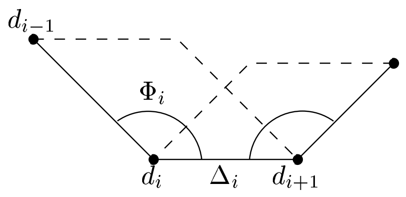

Figure 4.6: The configuration for the area based parameterization.

The area of such a parallelogram is given as $A_i = \sin\Phi_i \cdot \Delta_{i-1} \cdot \Delta_i$, and for reasons of symmetry we have to consider two of these areas, A_i and A_{i+1}, in order to calculate the value $u_{i+1} - u_i$. For similar reasons as in the angular parameterization, we restrict ourselves to internal angles larger than $\pi/2$, and define $\phi_i := \max(\Phi_i, \pi/2)$. We also normalize the term by dividing through the area of the rectangle with sides of length Δ_{i-1} and Δ_i. This results in the second contributing term for the parameterization:

$$h_i^2 := \frac{A_i + A_{i+1}}{(\Delta_{i-1} \cdot \Delta_i) + (\Delta_i \cdot \Delta_{i+1})} = \frac{\sin\phi_i \cdot \Delta_{i-1} + \sin\phi_{i+1} \cdot \Delta_{i+1}}{\Delta_{i-1} + \Delta_{i+1}}$$

Finally, we can express the parameterization as a weighted sum of the two contributing terms:

$$u_{i+1} - u_i = kh_i^1 + (1-k)h_i^2 = k\frac{\Delta_i}{\Delta} + (1-k)\frac{\sin\phi_i \cdot \Delta_{i-1} + \sin\phi_{i+1} \cdot \Delta_{i+1}}{\Delta_{i-1} + \Delta_{i+1}}$$

We have thus separated the two contributing factors for the parameterization into two different geometric terms that are weighted together. While the method is still purely heuristic, we think that the two contributing factors have clear geometrical interpretations. The factor k can again be used as a shape control parameter, if this is desired. In practice we found that a factor of $k = 2/3$ usually produced results that were comparable to the angular parameterization. In other words, the results were typically superior to the chord length and the centripetal parameterization. Figure 4.7 shows the result of the interpolation with the area based parameterization method. We will show more examples in Chapter 8.

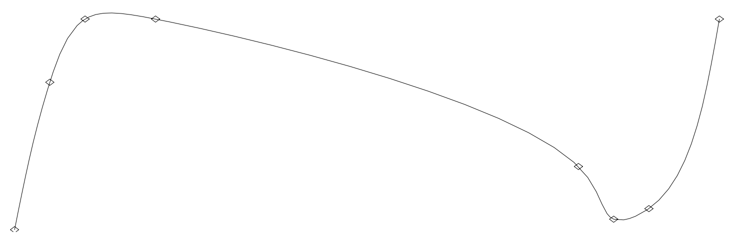

Figure 4.7: The area based parameterization typically produces results comparable to the angular parameterization.

4.6 An Affine Invariant Metric

Except for the uniform parameterization, all of the parameterization schemes we have presented used the Euclidean distance between the data point to determine the parameterization. These distances, and also the angles that are used for both the angular and the area based parameterization, are invariant under rigid body transformations such as rotations and translations.

The parameterizations are also invariant under scalings, provided that the same scaling factor is used for all directions. The area based scheme is invariant under these uniform scalings by construction. For all other methods a uniform scaling of the data set scales the distances between the resulting parameter values by a constant α:

$$t'_{i+1} - t'_i = \alpha(t_{i+1} - t_i)$$

This causes the knot vector to be scaled by the same factor α. As a consequence, the resulting interpolant will have a parameter domain that is also scaled by α, but the shape of the curve remains the same.

The situation is different for shearing transformations and for scalings in which the scaling factor is not uniform in every direction. In such cases the parameterization is not a multiple of the original one, and thus the shape of the resulting interpolant is different. Unfortunately, this means that the parameterization (and thus the resulting interpolant) is not invariant under affine transformations.

To overcome this problem, an *affine invariant metric* has been proposed in [11], [18] and [19]. The metric can be used to measure the "distance" of the data points in a way that is invariant under affine transformations, and especially under non-uniform scalings.

If we are given a set of data points as

$$\mathbf{V} = \begin{bmatrix} x_0 & y_0 & z_0 \\ x_1 & y_1 & z_1 \\ \vdots & \vdots & \vdots \\ x_n & y_n & z_n \end{bmatrix}$$

we call a metric $D_{\mathbf{V}}(u,v)$ *affine invariant* over the data set $\mathbf{V}$ if $D_{\mathbf{V}}(x,y) = D_{A\mathbf{V}+b}(Au+b, Av+b)$ for all affine transformations $f(u) = Au + b$.

Assuming that we are in a 3-dimensional space, such a metric can be defined using the norm

$$\left\| \begin{bmatrix} x & y & z \end{bmatrix}^T \right\|_V := \begin{bmatrix} x & y & z \end{bmatrix} \cdot n(\bar{\mathbf{V}}^T\bar{\mathbf{V}})^{-1} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

where

$$\bar{\mathbf{V}} = \begin{bmatrix} x_0 - \bar{x} & y_0 - \bar{y} & z_0 - \bar{z} \\ x_1 - \bar{x} & y_1 - \bar{y} & z_1 - \bar{z} \\ \vdots & \vdots & \vdots \\ x_n - \bar{x} & y_n - \bar{y} & z_n - \bar{z} \end{bmatrix}$$

and $\bar{x} := 1/(n+1)\sum_{i=0}^{n} x_i$, $\bar{y} := 1/(n+1)\sum_{i=0}^{n} y_i$ and $\bar{z} := 1/(n+1)\sum_{i=0}^{n} z_i$ are the medians of the components of the data points. Metrics for spaces of different dimensionality can be created in a similar fashion. Please note that the matrix $\bar{\mathbf{V}}$ has to be of full rank because otherwise the product $\bar{\mathbf{V}}^T\bar{\mathbf{V}}$ is singular, and the inverse can not be found. This means that, for a 3-dimensional interpolation problem, not all the data points must be in one plane, and for a 2-dimensional one not all must be on one line. If such a case happens the parameterization problem has to be solved in a lower dimension.

We refer the reader to [18] and [19] for a discussion of why $D_{\mathbf{V}}(u,v)$ is a norm, and why it leads to an affine invariant metric.

When we replace the Euclidean distance in the chord length parameterization and in the centripetal parameterization by this affine invariant metric, it is immediately clear that the resulting parameterization scheme is affine invariant. Since both schemes only depend on the distances of the data points, and these do not change under affine transformations, the parameterization does not change either. As a consequence, the knot sequence and the interpolant also do not change.

The situation for the angular and the area based parameterizations is slightly more difficult. These parameterization schemes depend not only on the distances but also on the angles between the chords. Thus a method has to be found to represent these angles in an affine invariant way. Using the Law of Cosines we can write the internal angle Φ_i between two chords as

$$\Phi_i = \arccos\left(\frac{D^2_{\mathbf{V}}(d_{i-1}, d_i) + D^2_{\mathbf{V}}(d_i, d_{i+1}) - D^2_{\mathbf{V}}(d_{i-1}, d_{i+1})}{2D_{\mathbf{V}}(d_{i-1}, d_i)D_{\mathbf{V}}(d_i, d_{i+1})}\right)$$

The external angle Θ_i can then be calculated as $\Theta_i = \pi - Phi_i$. Using this substitute for the angles together with the affine invariant metric for the distances allows us to have an affine invariant version of the angular and the area based parameterization. This is actually the form in which the angular parameterization has originally been presented in [11].

Figure 4.8 shows the chord length interpolant from Section 4.2 together with a new interpolant that uses the affine invariant metric together with the chord length parameterization. The new metric yields a slightly improved shape of the interpolant, although still not as good as the centripetal or angular parameterization. Since the interpolants for these two schemes were already very good, their results are not further improved by the use of the new metric.

4.7 Parameterizations for Tensor-Product Surfaces

In the case of tensor-product surfaces, some additional considerations are necessary. If we obtain a grid of data points by sampling a mathematical function ourselves, we can choose the sampling positions in such a way, that the parameter values form a rectangular grid in the parameter domain. All data points in a row of the grid then share the same u component, and all points in a column share the same v component. This means that the parameter of a data point $d_{i,j}$ can be written as (u_i, v_j).

A parameterization of this form is advantageous when we then want to fit a tensor-product B-spline surface through the grid of data points, since it allows us to break down the interpolation

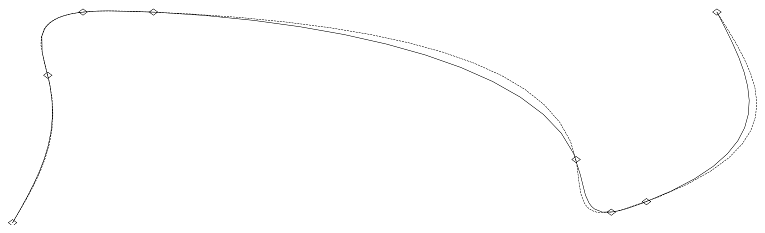

Figure 4.8: The use of the affine invariant metric improves the chord length parameterization slightly.

process into two smaller problems, as described in Section 3.1. If we had the more general case of a data grid in which each point can have its own parameter value $(u_{i,j}, v_{i,j})$ then this would not be possible, and we would have to solve a single, large system similar to Equation 3.3.

If we do not sample the function ourselves, but are rather given a set of discrete points without the parameterization, we first have to decide whether a single parameter value per row and per column is appropriate for the data set. If we think that such a parameterization is appropriate, we can simply use one of the algorithms described above. We separately compute the parameterization in u and v direction by applying one of the above algorithms to the averaged control points.

If we do not think that such a parameterization is appropriate for the data set, we can generate the parameter value $(u_{i,j}, v_{i,j})$ for each data point by applying one of the above algorithms separately to each row and column of the data grid. This will only work reasonably if the variations of the parameters in the rows and columns is not too high. If this does not hold, or if we don't even have gridded data points any more, the parameterization has to be generated with one of the methods from the field of scattered data interpolation.

Since in computer algebra systems we usually deal with gridded data, for which a rectangular grid of parameter values is a good assumption, we will not discuss the parameterization methods for scattered data interpolation at this place. We rather refer to [18], where the angular parameterization together with the affine invariant metric is applied to scattered data interpolation.

Chapter 5

B-Spline Approximation

The results of the previous two chapters show how an arbitrary set of discrete data points can be interpolated with B-splines. It has been shown how a parameterization for the data points can be generated, and how a set of B-spline basis functions can be selected. Given this information, a unique solution to the interpolation problem is obtained.

However, interpolation methods have a serious disadvantage if the goal is to represent more complex functions such as higher-order polynomials or transcendental functions, which can not be represented *exactly* as B-splines.

Since the interpolant is forced to go through the specified data points, the error in these points is zero, but the error between the points is relatively high. If too few data points are used for the interpolation, this tends to introduce bumps and other artifacts on the interpolant, which results in an unacceptable shape. Of course this problem can be avoided by using more sampling points, but then the size of the linear equation system grows rapidly, and obtaining a solution becomes expensive.

Figure 5.1 shows a quadratic interpolant for 11 data points sampled from a logarithm. The exact parameterization was used for the interpolation process, together with clamped end conditions specifying the original tangents in the end points. The interpolation process has introduced a bump near the second data point.

An alternative to B-spline interpolation is the use of B-spline *approximation* techniques. The idea behind approximation is to provide a large number of data points but to generate only a B-spline with a significantly lower number of control points. This means that there are fewer degrees of

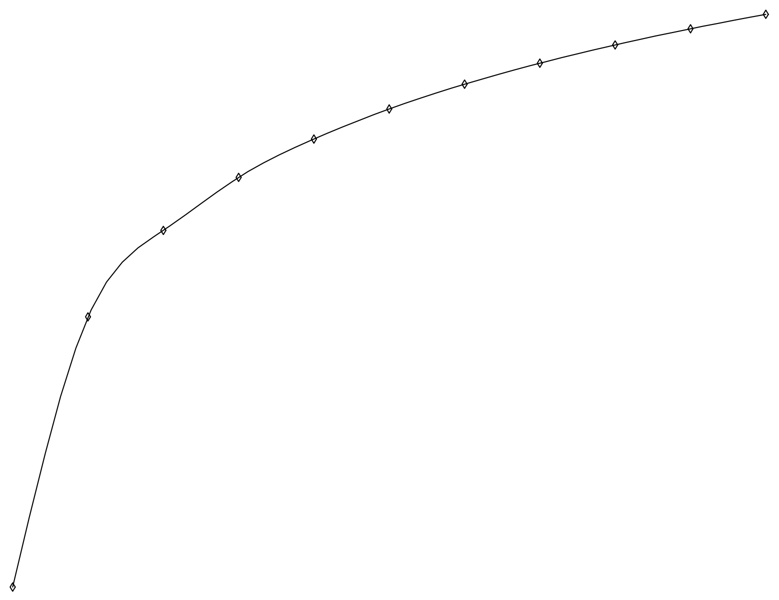

Figure 5.1: Interpolating sampled points from transcendental functions like logarithms sometimes introduces artifacts, like the bump in this example.

freedom than there are conditions, and that, in general, not all the conditions can be met. However, it is possible to minimize the error between the spline curve and the data points according to some norm.

If the right norm is chosen for this minimization process, the resulting curve can actually preserve the shape better than an interpolant through the same number of points, although the number of spline segments is lower in the case of an approximation. The idea is to give the spline more flexibility than in the interpolation case, and hopefully this results in a fit that does not pass exactly through all the data points, but is a better approximation to the shape of the original function.

In the case of curves, a set of $m+1$ data points $D = [d_0, \ldots, d_m]^T$ is provided for the approximation process. The parameterization of these data points is either provided as well, or it can be chosen using one of the methods described in Chapter 4. The approximation process determines the $n+1$ control points $C = [c_0, \ldots, c_n]^T$ of a B-spline curve $F(u) = \sum_{i=0}^{n} c_i B_i(u)$. Since the resulting linear equation system

$$\mathbf{B} \cdot C = D$$

is over-determined, such a solution does not exist in general. However, an approximation can be computed by minimizing $\|\mathbf{B} \cdot C - D\|$ according to a norm $\|.\|$. The choice of this norm has significant impact on the properties of the curve or surface, and can be chosen depending on the problem domain.

5.1 Least-Squares Fitting

Almost all of the literature on B-spline approximation uses the least-squares (L_2) norm for determining the "best" fit. The L_2 norm of a vector x is given as $\|x\|_2 = \sqrt{\sum_{i=0}^{m} x_i^2} = \sqrt{x^T \cdot x}$, where x_i is the i^{th} component of x.

The popularity of the L_2 norm for B-spline approximations is probably due to the simplicity with which this norm can be implemented. The following theorem describes how the least squares solution of an arbitrary over-determined linear equation system of the form $\mathbf{B} \cdot C - D$ can be obtained as the solution of a square linear equation system with full rank.

Theorem 5.1.1 (Least-squares solution for $\|\mathbf{B} \cdot C - D\|$)
For every $(n+1) \times (m+1)$ matrix $\mathbf{B}$ of full rank $n+1 \le m+1$ the least-squares solution of $\min \|\mathbf{B} \cdot C - D\|_2$ can be obtained by solving the (square) linear equation system

$$\mathbf{B}^T \mathbf{B} \cdot C = \mathbf{B}^T \cdot D$$

Proof: The least-squares solution $\min \|\mathbf{B} \cdot C - D\|_2$ is equivalent to the solution of $\min \|\mathbf{B} \cdot C - D\|_2^2$.

$$\begin{aligned}
\min \|\mathbf{B} \cdot C - D\|_2^2 &= \min \left[(\mathbf{B} \cdot C - D)^T (\mathbf{B} \cdot C - D) \right] = \\
&= \min \left[C^T \cdot \mathbf{B}^T - D^T)(\mathbf{B} \cdot C - D) \right] = \\
&= \min \left[C^T \cdot \mathbf{B}^T \mathbf{B} \cdot C - D^T \cdot \mathbf{B} \cdot C - C^T \cdot \mathbf{B}^T \cdot D \right] =
\end{aligned}$$

$$= \min \left[\sum_{i=0}^{m} \left(\sum_{j=0}^{n} b_{i,j} c_j \right)^2 - \sum_{i=0}^{m} d_i \sum_{j=0}^{n} b_{i,j} c_j - \sum_{j=0}^{n} c_j \sum_{i=0}^{m} b_{i,j} d_i + \sum_{i=0}^{m} d_i^2 \right] =$$

$$= \min \underbrace{\left[\sum_{i=0}^{m} \left(\sum_{j=0}^{n} b_{i,j} c_j \right)^2 - 2 \cdot \sum_{i=0}^{m} d_i \sum_{j=0}^{n} b_{i,j} c_j + \sum_{i=0}^{m} d_i^2 \right]}_{M}$$

In order to find this minimum, we compute the partial derivative of M with respect to all the control points c_k:

$$\begin{aligned} \frac{\partial M}{\partial c_k} &= \sum_{i=0}^{m} \left(2 \cdot \sum_{j=0}^{n} b_{i,j} c_j \right) a_{i,k} - 2 \cdot \sum_{i=0}^{m} d_i b_{i,k} \\ &= 2 \cdot \sum_{j=0}^{n} \left(\sum_{i=0}^{m} b_{i,k} b_{i,j} \right) c_j - 2 \cdot \sum_{i=0}^{m} d_i b_{i,k} \quad ; \quad k = 0 \ldots n \end{aligned}$$

Setting each of these derivatives to zero and writing the resulting equations in matrix form yields

$$2 \left(\mathbf{B}^T \mathbf{B} \cdot C - \mathbf{B}^T \cdot D \right) = 0$$

or

$$\mathbf{B}^T \mathbf{B} \cdot C = \mathbf{B}^T \cdot D.$$

Since it was assumed that $\mathbf{B}$ is of rank $n + 1 \leq m + 1$, the $(n+1) \times (n+1)$ matrix $\mathbf{B}^T\mathbf{B}$ is of full rank, and can be solved using standard techniques like Gaussian elimination. □

The importance of this result lies in the fact that we can compute the least-squares approximation by solving a linear equation system of size $(n+1) \times (n+1)$ instead of $(m+1) \times (m+1)$ in the interpolation case. Of course the equation system for the approximation has to be created before it can be solved. This involves the multiplication of two matrices $\mathbf{B}^T$ and $\mathbf{B}$ of size $(m+1) \times (n+1)$ and $(n+1) \times (m+1)$, respectively. The time complexity of these operations is $O(n^3 + mn^2)$, which compares to $O(m^3)$ for interpolation. This means that the approximation is faster than the interpolation if m is significantly larger than n.

Another efficient way of obtaining the least-squares solution is to compute the Q-R decomposition

$$\mathbf{B} = \mathbf{Q} \cdot \mathbf{R} = \mathbf{Q} \cdot \begin{bmatrix} \mathbf{R_1} \\ \mathbf{0} \end{bmatrix}$$

first. Here, $\mathbf{Q}$ is an orthogonal $m+1 \times m+1$ matrix, and R_1 is a upper triangular $n+1 \times n+1$ matrix. It can be shown that any $m+1 \times n+1$ matrix of full rank $n+1 \leq m+1$ can be written as a product of matrices of this form. Given this factorization of $\mathbf{B}$, it is easy to see that

$$\|\mathbf{B} \cdot C - D\|_2^2 = \|\mathbf{Q}^T \mathbf{B} \cdot C - \mathbf{Q}^T D\|_2^2 = \|\mathbf{R} \cdot C - \mathbf{Q}^T D\|_2^2 = \|\mathbf{R}_1 \cdot C - E_1\|_2^2 + \|E_2\|_2^2$$

where $\mathbf{Q}^T D = [E_1, E_2]^T$.

Therefore, the least-squares solution can be obtained from $\|\mathbf{R}_1 \cdot C - E_1\|_2^2$. Since $\mathbf{R}_1$ is a square matrix of full rank, the solution is identical to the solution of the linear equation system

$$\mathbf{R}_1 \cdot C = E_1,$$

which can be calculated easily due to the fact that $\mathbf{R}$ is an upper triangular matrix.

The complete algorithm takes $O(mn^2)$ operations for the Q-R decomposition, followed by (n^2) operations for actually obtaining the least-squares solution. This means that the asymptotic behavior of this algorithm is even better than the behavior of the first approximation algorithm and of B-spline interpolation.

Not only is approximation more efficient, it often also produces results that preserve the shape of the original curve better than interpolation does. Figure 5.2 shows the result of a least-squares approximation for the logarithm-data set shown before. The data set consists of the same 11 data points, but the approximation is done with a quadratic B-spline curve that has only 3 segments (6 control points). The resulting curve is much more pleasing, and visually closer to the original logarithm function.

5.2 Spline Fitting for Uncertain Data

An important field of applications for approximation techniques is the fitting of uncertain data that occurs for example in statistics or as a result of measurements in experiments. Approximation techniques in this context usually involve fitting a polynomial through a set of data points that are subject to some error. The error distribution depends on the specific application domain, and could for example be a uniformly distributed error on a few data points, or mostly correct data points with a few outliers.

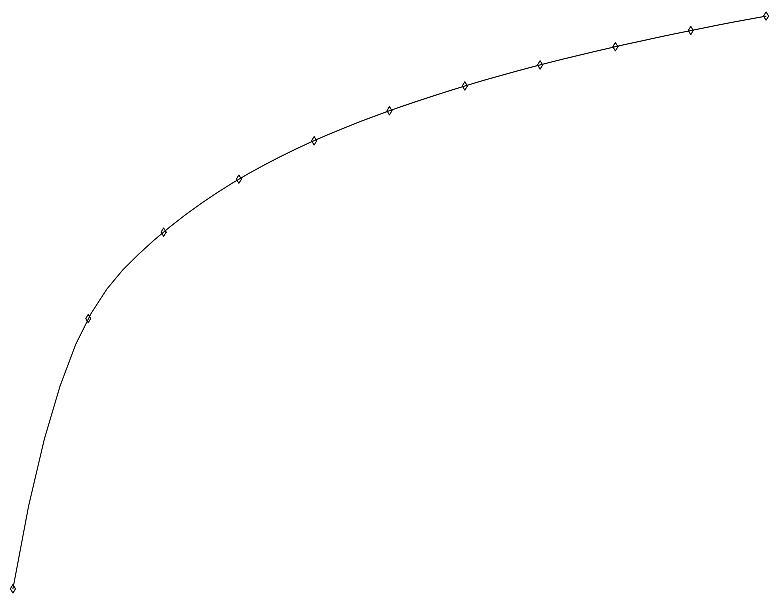

Figure 5.2: The least-squares approximation using 11 data points but only 6 control points yields a result that is superior to the interpolation solution, and visually significantly closer to the original logarithm function.

In most cases the approximation process is formulated as a *functional* problem. That is, given data points y_i that have been measured at specific positions x_i (for example points in time), a function $f(x)$ is desired, which minimizes $\|y_i - f(x_i)\|$. As mentioned above, usually a polynomial approximation is required, that is $f(x)$ is a polynomial function.

There are several differences between the scenario described above, and spline approximations. First of all, splines as described in this thesis define *parametric* instead of *functional* curves. However, since the B-spline approximation (like the interpolation) treats every component of the vector space separately, the parametric approximation can be seen as a composition of several independent functional approximations. Thus the only difference between a functional and a parametric approximation really is that in the functional case the exact parameterization is *always* known.

This is a significant advantage for the approximation of uncertain data, since it is not clear how an appropriate parameterization could be generated automatically. All the methods described in Chapter 4 have specifically been designed for generating curves that meet all points as close as possible. It can therefore not be expected that the removal of outliers is possible with parameterizations generated by these methods.

Another difference is that splines have different approximation properties than polynomials. Because of the local control property of B-splines, it is to be expected that a spline approximation locally changes to meet outliers better, and that the removal of outliers is therefore not as good as in the polynomial case. On the other hand with splines it should be possible to create good approximations to data sets with a more complex shape. The negative effects of the local control property can partly be compensated by increasing the degree of the curve, and thus extending the support of every basis function.

A final point to be aware of is that traditional fitting techniques for uncertain data involve the use of different norms for the minimization process, where a specific norm is selected according to the expected error distribution of the data set. In particular, the L_2 norm is typically used for data sets in which every data point is subject to a small, normally distributed error [20]. The L_1 norm is well-suited for removing a small set of outliers from a set of data points with otherwise high precision. Finally the L_∞ norm is appropriate if every single data point is very precise.

In the context of B-spline approximation, the L_2 norm is traditionally the only norm used. Consequently the form of B-spline approximation described in most of the literature is not well suited for handling uncertain data with outliers. Figure 5.3 shows a data set of 21 points sampled from a logarithm. One of these points has been moved in order to simulate an outlier. It is clearly visible that the approximating least-squares fit is trying to get close to the outlier, at the cost of introducing a larger error on the rest of the curve.

In the following we will describe how the L_1 and L_∞ norms can be applied to B-spline approximation problems using a linear programming approach. This allows the efficient combination of B-splines with standard techniques for fitting uncertain data.

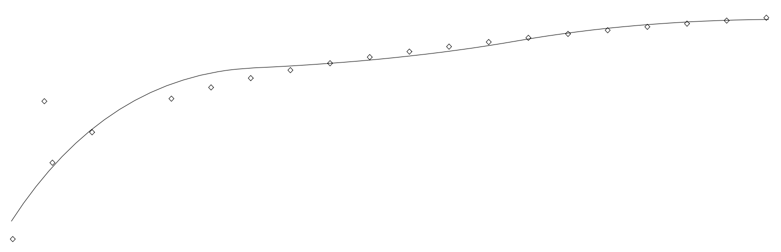

Figure 5.3: Quadratic least-squares approximation of a data set with a single outlier. Since the L_2 norm penalizes large distances from single points, the curve tries to get closer to the outlier, at the price of a larger error on the rest of the curve.

5.3 Fitting Using the L_1 Norm

The L_1 norm $\|x\|_1 = \sum_{i=0}^{m} |x_i|$ creates approximations that tend to ignore outliers. The reason is that the use of the absolute value penalizes a small error in a lot of points more than a large error in very few points. A curve created by an L_1 approximation therefore tries to be as close as possible to as many data points as possible, while at the same time accepting a large error in a very small number of points.

The L_1 B-spline approximation $\|\mathbf{B}\cdot C - D\|_1$ of a data set can be reduced to a *linear programming* problem (see Appendix B). Introducing B_i as a shorthand notation for the i^{th} row-vector of $\mathbf{B}$, the B-spline approximation using the L_1 norm can be written as $\min \sum_{i=0}^{m} |B_i^T C - d_i|$. The first step of transforming this to a linear programming problem is to make the function linear by removing the absolute value function. This can be achieved by introducing two vectors P and N of *slack variables* p_i and n_i. Slack variables are additional variables that are introduced to fit a linear programming problem into a specific form. This is a standard technique of linear programming.

Using these two vectors of variables, we can write

$$B_i^T C - d_i = p_i - n_i \quad ; \quad p_i \geq 0, \; n_i \geq 0 \tag{5.1}$$

The intention is to have

$$p_i = \begin{cases} B_i^T C - d_i & ; B_i^T C - d_i \geq 0 \\ 0 & ; \text{otherwise} \end{cases} \quad \text{and} \quad n_i = \begin{cases} 0 & ; B_i^T C - d_i \geq 0 \\ -B_i^T C + d_i & ; \text{otherwise} \end{cases}$$

Using these definitions, the expression

$$\sum_{i=0}^{m} p_i + n_i$$

becomes the function to minimize and Equation 5.1 becomes the constraint. We define $\mathbf{0}_n$ to be a vector of n zeroes, and $\mathbf{1}_n$ to be a vector of n ones. The linear programming problem for the L_1 approximation can now be written as

Minimize

$$\begin{bmatrix} \mathbf{0}_n & \mathbf{1}_m & \mathbf{1}_m \end{bmatrix} \cdot \begin{bmatrix} C \\ P \\ N \end{bmatrix}$$

subject to

$$\begin{bmatrix} \mathbf{B} & -\mathbf{Id}_m & \mathbf{Id}_m \end{bmatrix} \cdot \begin{bmatrix} C \\ P \\ N \end{bmatrix} = D \quad ; \quad p_i \geq 0, \; n_i \geq 0$$

This linear programming problem can be solved using the normal simplex method. An optimized implementation of the simplex method for this specific problem could be created, considering the special structure of the problem for the selection of pivot elements.

An L_1 fit using this method is shown in Figure 5.4. It contains the same data set as the least-squares fit shown in Figure 5.4 above. The L_1 fit does a much better job in ignoring the outlier and approximating the remainder of the curve. The figure also shows a problem that the L_1 norm sometimes has at the ends of the parameter interval. Since the leftmost point is relatively far away from the other points, it is treated as an outlier as well, although its position is actually correct.

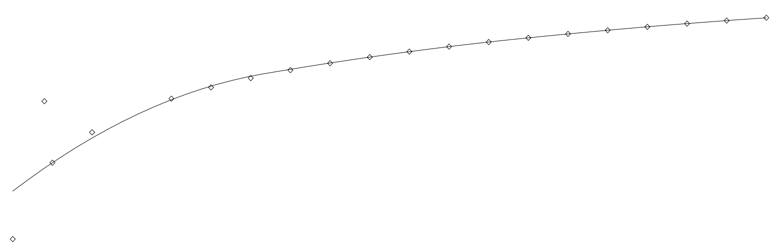

Figure 5.4: The quadratic L_1 approximation mostly ignores the outlier, and yields a good approximation to the rest of the curve.

5.4 Fitting Using the L_∞ Norm

As stated above, the L_∞ norm $\|x\|_\infty = \max_{0=1\ldots m} |x_i|$, also called *maximum-norm*, is particularly good for fitting curves to data points that are very exact. The reason is that this norm penalizes a large distance from a single point very strongly. The resulting curves consequently tend to distribute the error evenly across the whole curve.

As with the L_1 norm, it is also possible to express the approximation problem for the L_∞ norm as a linear programming problem. The L_∞ fit of some data points $D = [d_0, \ldots, d_m]^T$ is given as

$$\max_{i=0\ldots m} \left| d_i - B_i^T \cdot C \right| .$$

By defining $c_0 := \max_{i=0\ldots m} \left| d_i - B_i^T \cdot C \right|$, the approximation process can then be rewritten as a linear programming problem. The expression c_0 becomes the objective function, while the constraints are given as

$$\begin{aligned} &c_0 \geq 0 \\ &c_0 \geq d_i - B_i^T \cdot C \Leftrightarrow c_0 + B_i^T \cdot C \geq d_i \\ &c_0 \geq -d_i + B_i^T \cdot C \Leftrightarrow c_0 - B_i^T \cdot C \geq d_i \end{aligned}$$

In matrix form this results to

Minimize

$$\begin{bmatrix} \mathbf{0}_n & 1 \end{bmatrix} \cdot \begin{bmatrix} C \\ c_0 \end{bmatrix}$$

subject to

$$\begin{bmatrix} \mathbf{B} & \mathbf{1}_m^T \\ -\mathbf{B} & \mathbf{1}_m^T \end{bmatrix} \cdot \begin{bmatrix} C \\ c_0 \end{bmatrix} \geq \begin{bmatrix} D \\ -D \end{bmatrix} \quad ; \quad c_0 \geq 0$$

By introducing a vector S of slack variables the problem can be rewritten into the following form, which contains only equality and non-negativity constraints.

Minimize

$$\begin{bmatrix} \mathbf{0}_n & 1 & \mathbf{1}_m & \mathbf{1}_m \end{bmatrix} \cdot \begin{bmatrix} C \\ c_0 \\ S \\ T \end{bmatrix}$$

subject to

$$\begin{bmatrix} \mathbf{B} & \mathbf{1}_m^T & -\mathbf{Id}_m & \\ -\mathbf{B} & \mathbf{1}_m^T & & -\mathbf{Id}_m \end{bmatrix} \cdot \begin{bmatrix} C \\ c_0 \\ S \\ T \end{bmatrix} = \begin{bmatrix} D \\ -D \end{bmatrix} \quad ; \quad c_0 \geq 0,\ S_i \geq 0,\ T_i \geq 0$$

As in the case of the L_1 fit, the special structure of this problem could be used to implement an optimized version of the simplex algorithm.

For the sake of completeness we show the L_∞ approximation of the data set from the previous sections in Figure 5.5, although the L_∞ norm is inappropriate for data sets with outliers. We can

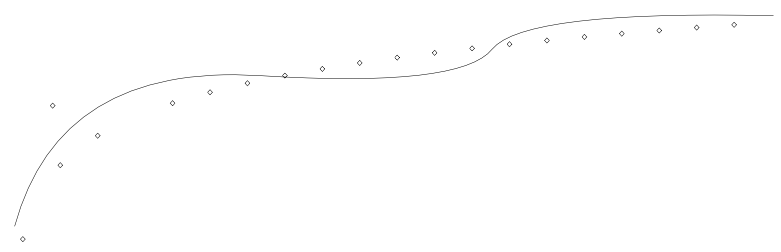

Figure 5.5: The L_∞ norm penalizes large errors in a single point even more than the L_2 norm, and thus the approximation comes even closer to the outlier, while at the same time sacrificing precision on the rest of the curve.

see that the shape of the curve is even worse than in the least-squares case, and that the error is distributed over the whole curve.

5.5 Reproducing B-Splines From Sampled Points

So far all arguments about the quality of an approximations have been very informal, mostly describing the resulting curve as being “good” or “pleasing” for certain applications, without a mathematical definition of what is “good” or “pleasing”. Unfortunately, there exist very few objective ways of measuring the quality of an approximation in a general setting.

The goal of any approximation method is that the shape of the resulting curve be similar to the shape of the function from which the data points were sampled. Thus, an important criterion for the quality of an approximation method is under which circumstances is the result an exact representation of the original function. In the case of B-spline approximation, we are interested in the conditions under which a B-spline curve can be exactly reproduced from a set of data points. The following theorem describes which information is required to make this possible.

Theorem 5.5.1 (Reproduction of B-splines with exact parameterization)
Suppose we are given $m+1 \geq n+1$ data points $D = [d_0, \ldots, d_m]^T$ that have been sampled from a

degree d B-spline curve $F(u) = \sum_{i=0}^{n} c_i B_i(u)$ at parameter values $(u_0, \ldots, u_m)$. Suppose these values have been chosen in such a way that the matrix $\mathbf{B}$ is of full rank. Then the degree d approximation process using either the least-squares (L_2), L_1 or L_∞ norm and the original parameter values as well as the original knot vector T yields the original control points $C = [c_0, \ldots, c_n]^T$.

Proof: Since the data points have been sampled from the B-spline curve $F(u)$, the matrix equation $\mathbf{B} \cdot C = D$ holds for the B-spline matrix $\mathbf{B} = (B_i(u_j))$ and the control points C. Because the original knot vector and parameterization is known, this matrix $\mathbf{B}$ is known as well, and can be used for the approximation process. Consequently, the approximation problem is solved by a minimization of $\|\mathbf{B} \cdot C' - D\|$ where $\|.\|$ may be any of the above norms, and C' is the vector of the resulting control points.

Using B_i as a shorthand notation for the i^{th} row vector of matrix B, the least-squares solution C' obtained by the approximation process is then characterized as

$$\min_{C'} \sum_{i=0}^{m} \left(B_i^T \cdot C' - d_i\right)^2 = \min_{C'} \sum_{i=0}^{m} \left(B_i^T \cdot C' - B_i^T \cdot C\right)^2$$

Because $M := \sum_{i=0}^{m} B_i^T \left(C' - C\right)$ is a sum of non-negative values, the minimal value that can be expected is $M = 0$. This value can actually be achieved by using the original control points $C' = C$. In other words, the original control points are *one* possible solution to the minimization problem. If we can prove that it is the *only* solution, we are done, because then it is clear that the original control points $C' = C$ will always be retrieved by the minimization process.

Since every term in the sum $M = \sum_{i=0}^{m} B_i^T \left(C' - C\right)$ is non-negative, $M = 0$ actually enforces that every single term $B_i^T \left(C' - C\right) = 0; \quad i = 0 \ldots m$. This gives us the linear equation system

$$\mathbf{B} \cdot \left(C' - C\right) = 0.$$

Since it was assumed that the matrix $\mathbf{B}$ has full rank $n + 1$, the only solution to this equation system is $C' - C = 0$.

A similar argument can be used to prove that the L_1 norm $\min \sum_{i=0}^{m} |B_i^T \cdot C - d_i|$ and the L_∞ norm $\min \left(\max_{i=0 \ldots m} |B_i^T \cdot C - d_i|\right)$ reproduce the original control points. $\square$

The consequence of this theorem is that B-spline curves with exact parameterization, knot sequence, and degree can always be reproduced exactly as long as the sample points are chosen in such a way that the matrix $\mathbf{B}$ is of full rank. It is important to note that the exact knot sequence is necessary and has to be provided. The knot vector represents the parametric values at which the polynomial segments of the spline join together. This information can not be generated from a set of discrete data points.

5.6 Surface Approximation Using Tensor-Product B-Splines

The methods for B-spline approximation described above can easily be extended to tensor-product B-spline surfaces. In analogy to the tensor-product B-spline interpolation, the approximation process can be described as minimizing the equation

$$\mathbf{B} \cdot C - D$$

where $\mathbf{B}$ is a matrix of dimension $(m_u+1)(m-v+1) \times (n_u+1)(n_v+1)$. Using a similar argumentation as in Section 3.1, this expression can be rewritten as the minimization of

$$\mathbf{B}_u \cdot (\mathbf{C} \cdot \mathbf{B}_v^T) - \mathbf{D}.$$

Thus a tensor-product approximation can be obtained by sequentially solving the two minimization problems

$$\mathbf{B}_u \cdot \mathbf{X} - \mathbf{D} \tag{5.2}$$

$$\mathbf{B}_v \cdot \mathbf{C}^T - \mathbf{X}^T \tag{5.3}$$

For the L_2 norm a solution can therefore be found by solving the equation

$$\mathbf{B}_u^T \mathbf{B}_u \cdot (\mathbf{C} \cdot \mathbf{B}_v^T \mathbf{B}_v) = \mathbf{B}_u^T \cdot \mathbf{D} \cdot \mathbf{B}_v.$$

In order to apply the L_1 and L_∞ norm to these two minimization problems, the norms have to be transformed into the form of linear programming problems defined in Appendix B. This can be achieved by rewriting the difference of matrices from Equation 5.2 into a difference of vectors in the following way:

$$\begin{bmatrix} \mathbf{B}_u & & & \\ & \mathbf{B}_u & & \\ & & \ddots & \\ & & & \mathbf{B}_u \end{bmatrix} \cdot \begin{bmatrix} x_{0,0} \\ x_{1,0} \\ \vdots \\ x_{n_u,n_v} \end{bmatrix} = \begin{bmatrix} d_{0,0} \\ d_{1,0} \\ \vdots \\ d_{m_u,m_v} \end{bmatrix}$$

The second minimization step $\mathbf{B}_v \cdot \mathbf{C}^T - \mathbf{X}^T$ can be rewritten in the same way. Please note that the resulting matrix is banded, and can therefore be handled efficiently.

Using these equations, the tensor-product approximation can be computed by separately applying the linear programming approach described in Sections 5.3 and 5.4 to the u and v direction.

As a consequence of this construction, the results from Theorem 5.5.1 directly translate to tensor-product approximation:

Corollary 1 (Reproduction of tensor-product B-spline surfaces)
Suppose we are given a grid of $(m_u + 1) \times (m_v + 1)$ data points $d_{i,j}$ that have been sampled from a degree $d_u \times d_v$ B-spline surface $F(u, v) = \sum_{i=0}^{n_u} \sum_{i=0}^{n_v} c_{i,j} B_i(u) B_j(v)$, where $m_u \geq n_u$ and $m_u \geq n_u$. Suppose the parameter values $(u_0, \ldots, u_{m_u})$ and $(v_0, \ldots, v_{m_v})$ have been chosen in such a way that the matrices $\mathbf{B}_u$ and $\mathbf{B}_v$ are of full rank. Then the $d_u \times d_v$ approximation process using either the least-squares (L_2), L_1 or L_∞ norm and the original parameter values as well as the original knot vectors T_u and T_v yields the original control points $c_{i,j}$. $\square$

Chapter 6

Rational Approximation Using NURBS

The vast majority of existing interpolation and approximation methods only produce integral B-spline curves and surfaces. However, rational B-splines provide more degrees of freedom for the shape of a curve or surface, and thus have the potential of producing better fits. NURBS are also easy to implement, and are frequently used for modeling purposes.

Unfortunately, the use of rational splines for approximation and interpolation is not well understood at this time. In [9], Farin suggests formulating the rational interpolation problem as follows: Given data points $D = [d_0, \ldots, d_m]^T$, weights $W = [w_0, \ldots, w_m]^T$ and parameter values $(u_0, \ldots, u_m)$, find the control points c_i of the NURBS curve that interpolates data points d_i and weights w_i at parameter values u_i. The approximation problem can be defined in a similar way. A solution for such a rational interpolation or approximation problem is easily obtained from the solution $[c_i w'_i, w'_i]^T$ of the integral problem for data points $[d_i w_i, w_i]^T$ in homogeneous coordinates.

The weights are usually not provided with the data set, however, and hence the weight for each data point has to be calculated first. This can be seen in analogy to finding a parameterization in cases where the exact parameterization is not known. Unfortunately, no algorithm for determining a set of weights for a given data set is known to date. Any algorithm for this task would have to determine the weights in such a way that the weights of all control points of the interpolant are non-negative. This is a hard task, since the interpolation process in homogeneous coordinates could set some of these weights to zero or to negative values, even if the weights in all data points are positive.

The lack of an algorithm for determining the weights practically renders the above approach for rational fitting useless for most applications.

In this situation, a recent paper by Ma and Kruth [17] describes a different approach for approximation with rational B-splines. Instead of trying to calculate the weights in the data points, they describe a way of directly determining the weights in the control points as a preprocessing step. The homogeneous form of the actual control points is then found by applying an integral approximation method to data points $w_i d_i$.

6.1 Observation Equations and Weights

Following the original paper [17], we now briefly describe how the weights in the control points of a curve can be obtained as the results of a minimization process. The arguments are be presented for approximations in a 3-dimensional vector space, but can easily be extended to other dimensions.

A NURBS curve $F(u)$ with $n+1$ control points c_i has the following form

$$F(u) = \frac{\sum_{i=0}^{n} w_i c_i B_i(u)}{\sum_{i=0}^{n} w_i B_i(u)}.$$

The evaluation of such a curve at $m+1$ parameter values $(u_0, \ldots, u_m)$ yields $m+1$ equations

$$\frac{\sum_{i=0}^{n} w_i c_i B_i(u_k)}{\sum_{i=0}^{n} w_i B_i(u_k)} = d_k; \quad k = 0 \ldots m.$$

Since the curve is assumed to be in 3-space, each of these equations actually denotes a set of three separate *observation equations*

$$\begin{aligned} d_k^x \sum_{i=0}^{n} w_i B_i(u_k) &= \sum_{i=0}^{n} w_i c_i^x B_i(u_k), \\ d_k^y \sum_{i=0}^{n} w_i B_i(u_k) &= \sum_{i=0}^{n} w_i c_i^y B_i(u_k), \\ d_k^z \sum_{i=0}^{n} w_i B_i(u_k) &= \sum_{i=0}^{n} w_i c_i^z B_i(u_k). \end{aligned}$$

We would like to write these equations in matrix form, and define $C_x := [c_0^x w_0, \ldots, c_n^x w_n]^T$,

$C_y := [c_0^y w_0, \ldots, c_n^y w_n]^T$ and $C_z := [c_0^z w_0, \ldots, c_n^z w_n]^T$ as well as

$$\mathbf{D}_x := \begin{bmatrix} d_0^x & & \\ & \ddots & \\ & & d_m^x \end{bmatrix} \quad ; \quad \mathbf{D}_y := \begin{bmatrix} d_0^y & & \\ & \ddots & \\ & & d_m^y \end{bmatrix} \quad ; \quad \mathbf{D}_z := \begin{bmatrix} d_0^z & & \\ & \ddots & \\ & & d_m^z \end{bmatrix}$$

The C_x, C_y and C_z are vectors containing the components of the control points in homogeneous form, while $\mathbf{D}_x$, $\mathbf{D}_y$ and $\mathbf{D}_z$ are diagonal matrices containing the components of the sampled points. Using these definitions, the resulting matrix form of the observation equations is

$$\begin{aligned} \mathbf{D}_x \cdot \mathbf{B} \cdot W &= \mathbf{B} \cdot C_x, \\ \mathbf{D}_y \cdot \mathbf{B} \cdot W &= \mathbf{B} \cdot C_y, \\ \mathbf{D}_z \cdot \mathbf{B} \cdot W &= \mathbf{B} \cdot C_z. \end{aligned} \tag{6.1}$$

These can further be rewritten into a single matrix system of dimension $3(m+1) \times 4(n+1)$,

$$\underbrace{\begin{bmatrix} \mathbf{B} & \mathbf{0} & \mathbf{0} & -\mathbf{D}_x\mathbf{B} \\ \mathbf{0} & \mathbf{B} & \mathbf{0} & -\mathbf{D}_y\mathbf{B} \\ \mathbf{0} & \mathbf{0} & \mathbf{B} & -\mathbf{D}_z\mathbf{B} \end{bmatrix}}_{\mathbf{A}} \cdot \begin{bmatrix} C_x \\ C_y \\ C_z \\ W \end{bmatrix} = \begin{bmatrix} \mathbf{0}_{m+1} \\ \mathbf{0}_{m+1} \\ \mathbf{0}_{m+1} \end{bmatrix}. \tag{6.2}$$

From these observation equations it follows that the approximation of $m+1$ data points with rational B-splines comes down to minimizing the left hand side of Equation 6.2 according to some norm. Unfortunately this equation system is rather large. It is, however, possible to manipulate the equation system in such a way, that the weights can be calculated separately by solving a smaller equation system. First, Equation 6.2 is multiplied by $\mathbf{A}^T$, yielding

$$\begin{bmatrix} \mathbf{B}^T\mathbf{B} & \mathbf{0} & \mathbf{0} & -\mathbf{B}^T\mathbf{D}_x\mathbf{B} \\ \mathbf{0} & \mathbf{B}^T\mathbf{B} & \mathbf{0} & -\mathbf{B}^T\mathbf{D}_y\mathbf{B} \\ \mathbf{0} & \mathbf{0} & \mathbf{B}^T\mathbf{B} & -\mathbf{B}^T\mathbf{D}_z\mathbf{B} \\ -\mathbf{B}^T\mathbf{D}_x\mathbf{B} & -\mathbf{B}^T\mathbf{D}_y\mathbf{B} & -\mathbf{B}^T\mathbf{D}_z\mathbf{B} & \mathbf{M}_0 \end{bmatrix} \cdot \begin{bmatrix} C_x \\ C_y \\ C_z \\ W \end{bmatrix} = \begin{bmatrix} \mathbf{0}_{n+1} \\ \mathbf{0}_{n+1} \\ \mathbf{0}_{n+1} \\ \mathbf{0}_{n+1} \end{bmatrix} \tag{6.3}$$

where $\mathbf{M}_0 = \mathbf{B}^T\mathbf{D}_x^2\mathbf{B} + \mathbf{B}^T\mathbf{D}_y^2\mathbf{B} + \mathbf{B}^T\mathbf{D}_z^2\mathbf{B}$. Note that the multiplication by $\mathbf{A}^T$ does not change the rank of the equation system. Eliminating the first elements of the last row results in

$$\begin{bmatrix} \mathbf{B}^T\mathbf{B} & \mathbf{0} & \mathbf{0} & -\mathbf{B}^T\mathbf{D}_x\mathbf{B} \\ \mathbf{0} & \mathbf{B}^T\mathbf{B} & \mathbf{0} & -\mathbf{B}^T\mathbf{D}_y\mathbf{B} \\ \mathbf{0} & \mathbf{0} & \mathbf{B}^T\mathbf{B} & -\mathbf{B}^T\mathbf{D}_z\mathbf{B} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{M} \end{bmatrix} \cdot \begin{bmatrix} C_x \\ C_y \\ C_z \\ W \end{bmatrix} = \begin{bmatrix} \mathbf{0}_{n+1} \\ \mathbf{0}_{n+1} \\ \mathbf{0}_{n+1} \\ \mathbf{0}_{n+1} \end{bmatrix} \tag{6.4}$$

with

$$\begin{aligned} \mathbf{M} = \mathbf{M}_0 - [&(\mathbf{B}^T\mathbf{D}_x\mathbf{B})(\mathbf{B}^T\mathbf{B})^{-1}(\mathbf{B}^T\mathbf{D}_x\mathbf{B}) + \\ &(\mathbf{B}^T\mathbf{D}_y\mathbf{B})(\mathbf{B}^T\mathbf{B})^{-1}(\mathbf{B}^T\mathbf{D}_y\mathbf{B}) + \\ &(\mathbf{B}^T\mathbf{D}_z\mathbf{B})(\mathbf{B}^T\mathbf{B})^{-1}(\mathbf{B}^T\mathbf{D}_z\mathbf{B})]. \end{aligned} \tag{6.5}$$

Note that $\mathbf{M}$ only depends on the data points and the B-spline basis functions. We have therefore managed to separate the weights from the control points, and can solve for them using the homogeneous equation system $\mathbf{M} \cdot W = \mathbf{0}_{n+1}$ of dimension $(n+1) \times (n+1)$.

Once a set of weights W has been found, the positions of the control points can be obtained by applying the minimization methods described in Chapter 5 to the observation equations (Equation 6.1). Figure 6.1 compares the integral approximation of a sine wave from 11 data points with the rational fit. Both approximations use produce a cubic curve with 3 intervals, and have been calculated using the L_2 norm.

6.2 Solving the Homogeneous Equation System

In order to obtain the weights W from Equation 6.4, a non-zero solution to the homogeneous equation system $\mathbf{M} \cdot W = \mathbf{0}_{n+1}$ has to be found. Actually, a positive solution (i.e. one with $w_i > 0$) is in general desirable, since negative weights, and especially the mixture of negative and positive weights in the same curve, may result in singularities, which is not acceptable.

For general sets of data points it might not be possible to obtain a non-negative solution, because the matrix $\mathbf{M}$ might well be non-singular. A solution only exists if the $(n+1) \times (n+1)$ matrix $\mathbf{M}$ has $rank(M) \leq n$. If the rank of M is exactly n, the solution is unique up to a scalar factor. Because scalar factors in the weights cancel out in the NURBS formula, the shape of the approximating curve is uniquely determined by such a system. If, on the other hand, the rank of

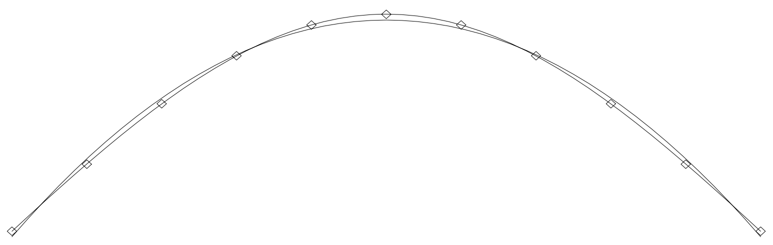

Figure 6.1: A comparison of a rational and an integral approximation. The rational approximation nearly interpolates the data points, while the integral one shows a significantly higher error.

$\mathbf{M}$ is smaller than n, the nullspace of M has a dimension larger than 1, and thus the solution is not unique.

Since it can not in general be assumed that the equation system actually has a solution, we are forced to use a minimization algorithm to find a set of weights. In cases where more than one solution is possible, the choice of the minimization algorithm determines which of the solutions is selected.

The authors of [17] propose two methods for determining a set of positive weights as the least-squares minimum of the equation system.

The first method is based on the singular value decomposition (SVD, see for example [12]). The SVD algorithm decomposes any matrix $\mathbf{M}$ into three matrices $\mathbf{M} = \mathbf{Q} \cdot \mathbf{D} \cdot \mathbf{P}^T$, such that $\mathbf{Q}$ and $\mathbf{P}$ are orthogonal, and $\mathbf{D}$ is a diagonal matrix with non-negative, decreasing entries. If the last p entries of $\mathbf{D}$ are zero or negligible, then the dimension of the nullspace of $\mathbf{M}$ is p. In this case the last p columns of $\mathbf{P}$ contain a basis for the nullspace.

If $\mathbf{M}$ is the matrix from Equation 6.5, any linear combination of these vectors can be chosen as a set of weights. If $p = 0$, that is, if the matrix $\mathbf{M}$ is not singular, simply the last vector of $\mathbf{P}$ is chosen. It can be proven that this is a least-squares solution of the equation system [12].

Another problem is that, even if $\mathbf{M}$ is singular, the obtained weights might be negative or zero. A set of positive weights might be obtained by choosing an appropriate linear combination of the basis vectors. If such a linear combination does not exist in the nullspace, it has been suggested that additional vectors from $\mathbf{P}$ be added until a positive solution is possible.

However, it is not clear, how the coefficients of a linear combination yielding positive weights can be found. The authors of [17] therefore suggest a second algorithm, based on *quadratic programming*, to find a set of positive weights. Quadratic programming is a method for solving minimization problems with linear constraints, but a quadratic objective function. One way of solving quadratic programming problems is to use an extension to the simplex algorithm for linear programming, as described in Appendix B. The details about quadratic programming can for example be found in [2] and [16].

Using quadratic programming, a set of positive weights can be obtained by minimizing $\|\mathbf{M}\cdot W\|_2^2$ subject to $w_i \geq \bar{w}$, where $\bar{w}$ is some positive value. A typical choice would be $\bar{w} = 1$, in which case the smallest weight is 1. Recall that a constant factor in the weights (and thus the value of $\bar{w}$) does not have an effect on the shape of the curve.

Unfortunately, the use of the general quadratic programming algorithm on this problem is rather inefficient. Stated as a quadratic programming problem, our minimization problem takes the form

Minimize

$$w^T \mathbf{M}^T \mathbf{M} w$$

subject to

$$w \geq 0$$

The inefficiencies arise from forming $\mathbf{M}^T\mathbf{M}$ and the fact that the constraints have a very special form. On the other hand, it is not clear why a least-squares solution for the weights should be preferable to the solution obtained using some other norm.

Obvious alternatives for the least-squares norm include the L_1 and L_∞ norms already described in Chapter 5. Using these norms, the weights can be obtained as the solution of a linear programming problem, which can in general be solved more efficiently than a quadratic programming problem of the same size.

Since the weights are required to be positive, additional constraints have to be added to the linear programming problems presented in Chapter 5. The minimization problem for obtaining L_1 weights can be stated as

---

Minimize

$$\begin{bmatrix} \mathbf{0}_n & \mathbf{1}_m & \mathbf{1}_m & \mathbf{1}_n \end{bmatrix} \cdot \begin{bmatrix} W \\ P \\ N \\ S \end{bmatrix}$$

subject to

$$\begin{bmatrix} \mathbf{B} & -\mathbf{Id}_m & \mathbf{Id}_m & \\ \mathbf{Id}_n & & & -\mathbf{Id}_n \end{bmatrix} \cdot \begin{bmatrix} W \\ P \\ N \\ S \end{bmatrix} = \begin{bmatrix} D \\ \mathbf{1}_n \end{bmatrix} \quad ; \quad p_i \geq 0,\ n_i \geq 0,\ s_i \geq 0$$

The additional slack variables s_i are used to ensure that the smallest weight has a value of 1 and is therefore positive. It should again be pointed out that the matrix of conditions, although larger than the matrix from Section 5.3, still has a special and very simple structure, which allows for optimized solution methods.

A comparable extension to the L_∞ problem presented in 5.4 can be stated in a similar fashion.

6.3 Analysis

It is clear that the evaluation of the quality of rational approximation methods is strongly dependent on the quality of the determined weights. The approximation of the control points using the observation equations 6.1 after the weights have been found, has properties similar to the integral approximation described in Chapter 5.

Since the weights are computed as the solutions of the homogeneous equation system $\mathbf{M}\cdot W = \mathbf{0}$, where $\mathbf{M}$ is the matrix from Equation 6.5, the rank of $\mathbf{M}$ plays an important role in the analysis of rational approximation. The following lemma provides an upper bound for this rank. It uses the dimension e of the space, and assumes that $m \geq n$, and that the matrix $\mathbf{B}$ is of full rank $n+1$. Since these two assumptions were already necessary in the case of integral approximation, it follows that they also have to hold for the rational case. This will be assumed throughout this section.

Lemma 1 (Rank of M)

An upper bound for the rank of $\mathbf{M}$ is given as

$$rank(\mathbf{M}) \leq \min(n+1, e(m-n))$$

Proof: In Equation 6.2 the observation equation for a 3-dimensional NURBS curve with $n+1$ control points, evaluated at $m+1$ data points was given as

$$\underbrace{\begin{bmatrix} \mathbf{B} & \mathbf{0} & \mathbf{0} & -\mathbf{D}_x\mathbf{B} \\ \mathbf{0} & \mathbf{B} & \mathbf{0} & -\mathbf{D}_y\mathbf{B} \\ \mathbf{0} & \mathbf{0} & \mathbf{B} & -\mathbf{D}_z\mathbf{B} \end{bmatrix}}_{\mathbf{A}} \cdot \begin{bmatrix} C_x \\ C_y \\ C_z \\ W \end{bmatrix} = \begin{bmatrix} \mathbf{0}_{m+1} \\ \mathbf{0}_{m+1} \\ \mathbf{0}_{m+1} \end{bmatrix}$$

This equation can be generalized to spaces of different dimensions e. The dimension of these generalized matrices $\mathbf{A}$ is $e(m+1) \times (e+1)(n+1)$ since $\mathbf{B}$ is of dimension $(m+1) \times (n+1)$. The rank of $\mathbf{A}$ is therefore limited by

$$rank(\mathbf{A}) \leq e(m+1)$$

for under-determined systems, and

$$rank(\mathbf{A}) \leq (e+1)(n+1)$$

for over-determined systems. This rank does not change under the matrix transformations that lead from Equation 6.2 over Equation 6.3 (multiplication by $\mathbf{A}^T$) to Equation 6.4 (elimination of row entries). The matrix in Equation 6.4 is given as

$$\left[\begin{array}{ccc|c} \mathbf{B}^T\mathbf{B} & \mathbf{0} & \mathbf{0} & -\mathbf{B}^T\mathbf{D}_x\mathbf{B} \\ \mathbf{0} & \mathbf{B}^T\mathbf{B} & \mathbf{0} & -\mathbf{B}^T\mathbf{D}_y\mathbf{B} \\ \mathbf{0} & \mathbf{0} & \mathbf{B}^T\mathbf{B} & -\mathbf{B}^T\mathbf{D}_z\mathbf{B} \\ \hline \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{M} \end{array}\right]$$

with the obvious modifications for spaces with dimension $e \neq 3$. Since it was assumed that $\mathbf{B}$ is of full rank $n+1 \leq m+1$, the $(n+1) \times (n+1)$ matrix $\mathbf{B}^T\mathbf{B}$ is also of full rank. Thus the upper part of the above matrix has full rank $e(n+1)$ as well. For under-determined systems this means that the rank of $\mathbf{M}$ is bounded by

$$rank(\mathbf{M}) \leq e(m+1) - e(n+1) = e(m-n),$$

while for over-determined systems

$$rank(\mathbf{M}) \leq (e+1)(n+1) - e(n+1) = n+1$$

holds. Since the matrix $\mathbf{A}$ is under-determined exactly in those cases where $e(m-n) \leq n+1$, these two results can be summarized as

$$rank(\mathbf{M}) \leq \min(n+1, e(m-n))$$

$\square$

When comparing rational and integral approximation, it is an interesting question how many control points are required in order to guarantee the interpolation of all data points. From Chapter 3, it is known that in the integral case interpolation is achieved with $n+1 = m+1$ control points. For the rational case, Lemma 1 helps proving the following theorem:

Theorem 6.3.1 (Interpolation of data points)
The rational approximation process with at least

$$n \geq \frac{em}{e+1}$$

control points yields a curve that interpolates all data points.

Proof: Interpolation is achieved if and only if all the observation equations from Equation 6.1 are fulfilled. This is equivalent to matrix $\mathbf{A}$ of Equation 6.2 being of less than full row rank, which in turn holds if and only if matrix $\mathbf{M}$ of Equation 6.4 is not of full rank $n+1$. From

$$n \geq \frac{em}{e+1} \Leftrightarrow n \geq e(m-n)$$

and Lemma 1 it follows that $rank(M) \leq n < n+1$ for $n \geq (em)/(e+1)$. $\square$

Obviously, the higher the value of n, the lower the rank of $\mathbf{M}$ will be. Consequently the dimension of the nullspace increases and the resulting weights are no longer unique. In these cases, it depends on the norm, which set of weights is generated by the minimization process.

The optimal case would be a rank of n, one lower than the full rank. In this case the nullspace has dimension 1, and the weights are determined uniquely up to a constant factor. This is of particular importance when a NURBS curve is to be reconstructed from discrete sample points.

It is easy to show that, if the data points are sampled from a NURBS curve, the original weights are always in the nullspace of $\mathbf{M}$, and are therefore always *one* possible solution, as long as the original parameterization of the original NURBS curve is used. The matrix $\mathbf{M}$ is a sum of terms of the form

$$\mathbf{B}^T \cdot \mathbf{D}_x^2 \cdot \mathbf{B} - (\mathbf{B}^T \cdot \mathbf{D}_x \cdot \mathbf{B}) \cdot (\mathbf{B}^T \cdot \mathbf{B})^{-1} \cdot (\mathbf{B}^T \cdot \mathbf{D}_x \cdot \mathbf{B})$$

Since the parameterization is known, the matrix $\mathbf{B}$ in the approximation process is identical to the matrix $\mathbf{B}$ in the observation equations. Thus the relationship $\mathbf{D}_x \cdot \mathbf{B} \cdot W = \mathbf{B} \cdot C_x$ from the observation equations holds. Multiplying the above matrix terms with W, and using this relationship yields

$$\mathbf{B}^T \cdot \mathbf{D}_x \cdot \mathbf{B} \cdot C_x - (\mathbf{B}^T \cdot \mathbf{D}_x \cdot \mathbf{B}) \cdot (\mathbf{B}^T \cdot \mathbf{B})^{-1} \cdot \mathbf{B}^T \cdot \mathbf{B} \cdot C_x \quad = \quad \mathbf{B}^T \cdot \mathbf{D}_x \cdot \mathbf{B} \cdot C_x - (\mathbf{B}^T \cdot \mathbf{D}_x \cdot \mathbf{B}) \cdot C_x \quad = \quad \mathbf{0}$$

Thus $\mathbf{M} \cdot W$ is a sum of zero vectors if W is the original vector of weights, or a multiple thereof.

Reconstruction of NURBS curves with exact parameterization is therefore guaranteed if the rank of $\mathbf{M}$ is n. A necessary, however not sufficient condition for this is given by

$$m \geq n + \frac{n}{e} \Leftrightarrow n \geq e(m-n) \geq rank(\mathbf{M})$$

In other words, if less than $n + n/e$ sample points are provided, the rank of $\mathbf{M}$ is always less than n, and a unique solution can not exist.

Since no non-trivial lower bound for the rank of $\mathbf{M}$ is known at this time, it is unfortunately not possible to state a sufficient condition that results in 1-dimensional nullspace of $\mathbf{M}$.

It is questionable, whether such a lower bound exists, since the exact rank of $\mathbf{M}$ depends on the components of the data points provided. For example consider a 2-dimensional curve in 3-dimensional space, say the $z = 1$ plane. The matrix $\mathbf{A}$ from Equation 6.2 then reduces to

$$\mathbf{A} = \left[\begin{array}{cccc} \mathbf{B} & \mathbf{0} & \mathbf{0} & -\mathbf{D}_x\mathbf{B} \\ \mathbf{0} & \mathbf{B} & \mathbf{0} & -\mathbf{D}_y\mathbf{B} \\ \hline \mathbf{0} & \mathbf{0} & \mathbf{B} & -\mathbf{B} \end{array}\right]$$

Clearly the last row of block matrices has only rank $n + 1$ instead of $m + 1$. Therefore the rank of $\mathbf{A}$ is not full, and the resulting rank of $\mathbf{M}$ is less than the value determined in Lemma 1. While this is an extreme example, it is clear that other data points can reduce the rank of the matrix as well.

6.4 Fitting With Tensor-Product NURBS

The algorithm for rational approximation presented above can easily be extended to tensor-product surfaces. Using the definitions

$$\mathbf{B} := (b_{i,j}) \quad \text{where} \quad b_{jn_u+i,ln_u+k} := \mathbf{B}_i^u(u_k)\mathbf{B}_j^v(v_l),$$

$$C_x := \left[\begin{array}{cccc} c_{0,0}^x & c_{1,0}^x & \dots & c_{n_u,n_v}^x \end{array}\right]^T,$$

$$W := \left[\begin{array}{cccc} w_{0,0} & w_{1,0} & \dots & w_{n_u,n_v} \end{array}\right]^T$$

and

$$\mathbf{D}_x := \left[\begin{array}{cccc} d_{0,0}^x & & & \\ & d_{1,0}^x & & \\ & & \ddots & \\ & & & d_{n_u,n_v}^x \end{array}\right],$$

the observation equations from Section 6.1 also hold for tensor-product NURBS. This leads to the known homogeneous equation system

$$\mathbf{M} \cdot W = \mathbf{0}$$

for the weights. The matrix $\mathbf{M}$ has dimension $(n_u + 1)(n_v + 1) \times (n_u + 1)(n_v + 1)$. Its generation involves a matrix product of the matrix $\mathbf{B}$ of dimension $(m_u+1)(m_v+1) \times (n_u+1)(n_v+1)$ with its transpose. Fortunately, $\mathbf{B}$ is sparse and is only multiplied by its transpose and by diagonal matrices, so that an efficient formula for each element of $\mathbf{M}$ can be found, and general matrix products need not to be used.

After the weights have been obtained we can once again take advantage of the special structure of tensor-product surfaces. We define $\mathbf{B}_u$ and $\mathbf{B}_v$ as in Chapter 3 and Chapter 5. Furthermore we define the $(n_u+1)\times(n_v+1)$ matrix $\mathbf{W}=(w_{i,j})$ and the $(m_u+1)\times(m_v+1)$ matrix $\mathbf{D}=(d_{i,j})$. Finally, we define $\mathbf{D}_H$ as the *component-wise* product of the two $(m_u+1)\times(m_v+1)$ matrices $\mathbf{D}$ and $(\mathbf{B}_u\cdot\mathbf{W}\cdot\mathbf{B}_v^T)$. This matrix $\mathbf{D}_H$ is the right hand side of the observation equation for tensor-product surfaces

$$\sum_{i=0}^{n_u}\sum_{j=0}^{n_v} w_{i,j}c_{i,j}B_i^u(u_k)B_j^v(v_l) = d_{k,l}\cdot\sum_{i=0}^{n_u}\sum_{j=0}^{n_v} w_{i,j}B_i^u(u_k)B_j^v(v_l) \quad ; k=0\dots m_u, l=0\dots m_v$$

With this definition the homogeneous form $C=(w_{i,j}c_{i,j})$ of the control points can be computed from

$$\mathbf{B}_u\cdot\mathbf{C}\cdot\mathbf{B}_v^T=\mathbf{D}_H.$$

Chapter 7

Implementation

Based on the concepts described in the previous chapters, a framework for integral and rational B-spline interpolation and approximation has been implemented in MAPLE. This package allows for the generation of B-spline based approximations of functions or discrete data points for the purpose of generating graphical plots of functions or sampled data.

The major design goal of this framework was a modular structure that allows for the easy exchange of the basic algorithms, in order to support experiments necessary for the research presented in this thesis. Thus modularity usually took precedence over efficiency where such a decision had to be made.

The implementation itself is structured into three parts. The first part is the addition of geometric primitives for B-splines and NURBS to the existing plot data structure of MAPLE. This allows to send spline data directly to the plot driver. That is, instead of approximate splines as a sequence of lines or polygons, these extensions allow to send the control points and the knot sequence to the plot driver, and thus provides the driver with an exact representation of the spline. The spline extensions were implemented for the existing OpenGL driver for MAPLE [14].

The second part consists of the actual framework for the implementation of the interpolation and approximation methods described in Chapters 3 through 6. This has been implemented as a MAPLE library package.

Finally, the third part involves replacing the functions in the MAPLE `plots` and `plottools` packages by spline based versions. This is the actual user-level programming interface for generating plots in MAPLE. This part is mostly to be seen as a proof of concept, since only the most important functions like `plot` and `plot3d` have been re-implemented based on splines.

In the following we briefly outline the basic design concepts behind the three parts of the implementation.

7.1 NURBS Extensions for the Plot Data Structure of MAPLE

User-level plot functions like `plot` and `plot3d` store the geometry they generate in a MAPLE data structure, called the *plot data structure*. Besides the actual geometry, this data structure contains all the rendering attributes, such as drawing mode, color, line style and so on. The plot data structure is accessible from the MAPLE programming language like every other data structure in MAPLE. An example of a plot data structure containing two red points is shown below.

```
PLOT( POINTS( [0,0], [1,1] ), COLOR( RGB, 1, 0, 0 ) );
```

Note the keyword `PLOT`, which denotes a data structure for 2-dimensional plots. A 3-dimensional plot would use the keyword `PLOT3D`.

For the actual rendering, the plot data structure is converted to a more compact binary representation as a C structure, and then transmitted to the plot driver. The driver then renders all the geometric primitives in the plot structure with the appropriate rendering parameters.

The existing MAPLE plot structure provides points, connected lines, polygons, rectangular polygon grids, and text as geometric primitives. In order to support NURBS curves and surfaces, two new primitives were added, `NURBS` and `TP_NURBS`. The `NURBS` primitive describes a potentially rational spline curve in two or three dimensions, while `TP_NURBS` describes a tensor-product surface in three dimensions only.

The arguments of the `NURBS` primitive are the degree of the curve, followed by the list of control points and an optional knot sequence. If no knot sequence is present, the sequence

$$T := (\underbrace{0,\ldots,0}_{d+1}, 1, \ldots, k-1, \underbrace{k,\ldots,k}_{d+1})$$

is assumed, where $k := n - d$ is the number of spline segments. This convention makes it easy to specify Bèzier-style NURBS curves, as the following example of a quarter circle demonstrates.

```
PLOT( NURBS( 2, [[1,0], [1,1,sqrt(.5)], [0,1]] ) );
```

The example also shows that the specification of weights is optional. A default weight of 1 is assumed if none is provided.

The general structure of the `TP_NURBS` primitive is similar to the case of curves. However, the control points for tensor-product surfaces form a grid, and are specified as a list of lists. Different degrees and knot sequences for the u and v direction of the surface are also supported. They can be specified as a list of degrees or knot sequences, respectively. An example for a tensor-product surface is shown below.

```
PLOT3D( TP_NURBS( [2,3],
        [ [[0,0,0], [1,1,0], [2,1,0], [3,0,0]],
          [[0,1,1], [1,2,1], [2,2,1], [3,1,1]],
          [[0,0,2], [1,1,2], [2,1,2], [3,0,2]] ] ) );
```

These two new drawing primitives have also been added to the C variant of the plot data structure, and are currently supported by the OpenGL plot driver [14]. The NURBS support in the OpenGL driver has been implemented using the NURBS facilities in OpenGL. The other MAPLE plot drivers do not support splines at this point, and simply ignore the additional drawing primitives. Spline support for these drivers could easily be implemented using subdivision or forward-differencing algorithms [1]. This is, however, beyond the scope of this thesis.

7.2 A Framework for Integral and Rational B-Spline Fitting

The actual spline fitting is implemented as a MAPLE package, and can be loaded like normal library packages. The interpolation code generates plot data structures, which normally contain the geometry directly represented in terms of the NURBS primitives defined in the previous section. Since these primitives are not yet supported by most drivers, it is alternatively possible to generate a plot data structure that contains only lines and polygon grids, and which can therefore be rendered

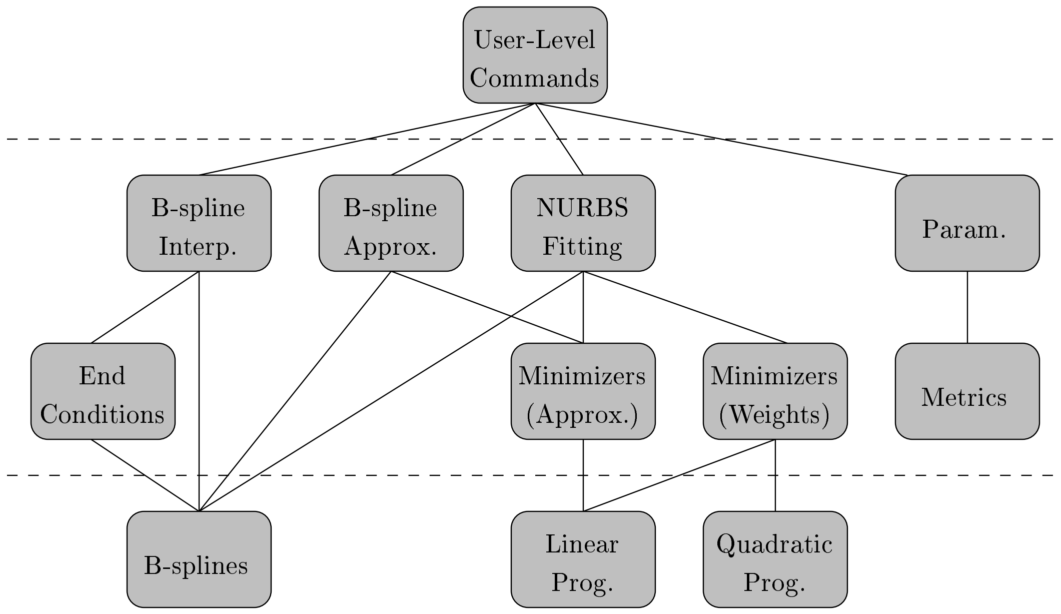

Figure 7.1: Dependency graph of the modules in the framework.

by all existing drivers. This feature was used to generate most of the figures in this thesis with the help of the PostScript driver.

The framework of modules making up the package is divided into three layers (see Figure 7.1). The lowest layer contains three modules of support functions: a B-spline package and one module for linear programming and quadratic programming, respectively. The module for linear programming contains an extended version of the standard MAPLE linear programming facilities. The B-spline module contains an implementation of the basis functions, and of integral as well as rational spline curves and surfaces, based on MAPLEs piecewise functions.

The intermediate layer is itself structured into a layer of support modules and the actual fitting modules. The former consists of a module of end conditions for interpolation, a module of minimizers for the approximation process, and another set of minimizers for determining the weights for NURBS fitting. It also provides a set of metrics for the parameterization algorithms. Each of these modules contains implementations for all algorithms described in the corresponding chapters.

The modules for the actual spline fitting are one module for B-spline interpolation, one for B-spline approximation and one for NURBS fitting. Each of these modules handles both curves and tensor-product surfaces. There also exists a module that contains all the parameterization algorithms described in Chapter 4.

The selection of the different algorithms is done with a scheme of callback functions. For example there is one callback that specifies which metric is to be used for generating the parameterization. This way, it is easy to exchange the Euclidean metric with the affine invariant one by simply changing the callback. A similar mechanism applies to the selection of a norm for minimization, a norm for determining the weights, and an end condition.

It is important to note that the spline fitting functions on this level do not directly use the parameterization methods themselves. Instead, they expect a vector of parameter values as one of their arguments. It is the user-level functions forming the third layer, which actually choose a parameterization, and pass it on to the second layer functions. This allows for the use of the exact parameterization where it is available.

7.3 Spline Replacements of MAPLE Library Functions

The user-level module contains a relatively small set of functions that are modeled after the most important existing MAPLE library functions. Most importantly, these include functions for generating both curves and surfaces, for both discrete data and continuous functions. The remaining library functions of the `plots` and `plottools` packages do either not generate geometry, or would not require techniques that are significantly different from the ones already used.

The functions that have been implemented support the full repertoire of options for 2-dimensional and 3-dimensional plots, as defined in [4]. In addition, a few new parameters have been defined, which influence the selection of algorithms and several other parameters that have been discussed in previous chapters. A complete list of the new options, as well as a description of the implemented functions can be found in Appendix C. The following is an example for generating a 2-dimensional plot showing a B-spline approximation of a sine curve on the interval $[0, \pi]$.

```
SplinePlot( sin(x), x=0..Pi, color=red, degree=2, segments=3, linestyle=2 );
```

The spline curve generated by this line will have 3 segments of degree 2 and will be rendered in red, using a dotted line style. Note, how the newly introduced options `degree` and `segments` are used together with the standard MAPLE options `color` and `linestyle`.

Chapter 8

Results

In Chapters 3 through 6 we have presented a set of algorithms for different parts of spline fitting. In this chapter we conclude by summarizing the effects of the different algorithms and showing results that have been obtained during experiments with the framework described in Chapter 7. Since at this stage every algorithm has only been considered in isolation, we also discuss how different parameters of the fitting process correlate.

8.1 Parameterization

We start by having a more detailed look at the different parameterization techniques. In Chapter 4 the different parameterization methods have been presented in the context of the interpolation problem. However, no statement has been made about the quality of these methods for spline approximation.

Fortunately, it turns out that the parameterization methods are relatively independent of the fitting algorithm. That is, a parameterization that works well for interpolating a specific set of data points usually also performs well for integral and rational approximation of the same data set. In other words, the angular and the area based parameterization techniques are usually preferable over other techniques like chord-length or centripetal parameterization. A comparison of the parameterization techniques applied to an approximation problem is shown in Figure 8.1.

An interesting data set for comparing different parameterization methods is shown in Figures 8.2, 8.3 and 8.4. It consists of 8 points, the first and the last 4 of which are collinear. The

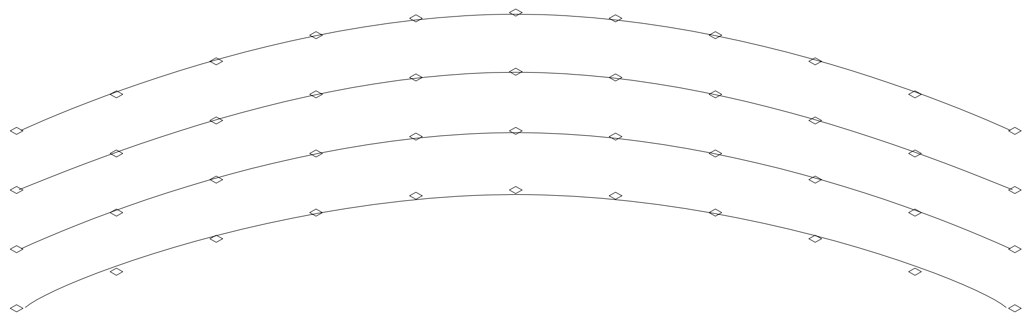

Figure 8.1: A comparison of the different parameterization schemes for approximation. From bottom to top: chord-length, centripetal, angular and area based parameterization.

two lines form a right angle, and the chord between the fourth and the fifth point is very short compared to the other chords.

The reason why this data set is so interesting, is that it is very ill-conditioned. On the one hand, the fourth and the fifth point are very close together in comparison to the other data points, and therefore the corresponding parameter values should be close together as well. On the other hand, a sharp turn occurs between these two points, so that the curve should slow down. This corresponds to parameter values that are further apart.

In Figure 8.2 the Euclidean metric has been used for all parameterizations. The centripetal method and the angular algorithm produce equally good parameterizations for this data set. The chord-length method does not take the sharp turn into account, and therefore places the fourth and the fifth parameter value too close together. This leads to the introduction of artifacts on the interpolant.

The uniform parameterization, on the other hand, places the two parameter values too far away from each other, and thereby introduces a loop in the curve, which can better be seen in the closeup in Figure 8.3. Also visible in the closeup is a sharp corner that is generated by the angular parameterization method.

Figure 8.4 depicts the results of using the affine invariant metric for all parameterization meth-

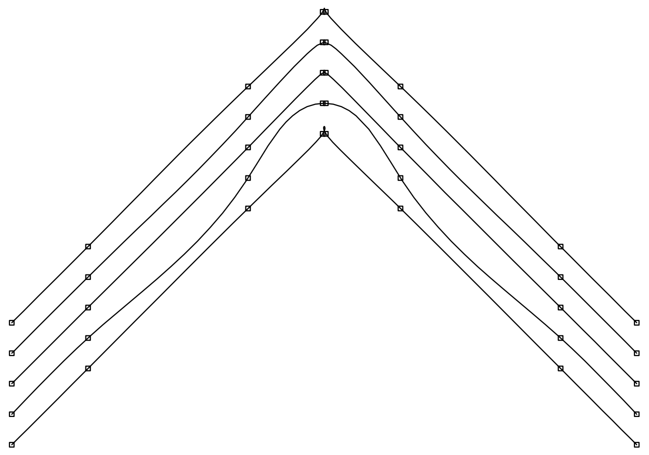

Figure 8.2: A comparison of the parameterization schemes using the Euclidean metric. The parameterization schemes (from bottom to top) are uniform, chord-length, centripetal, angular and area based parameterization.

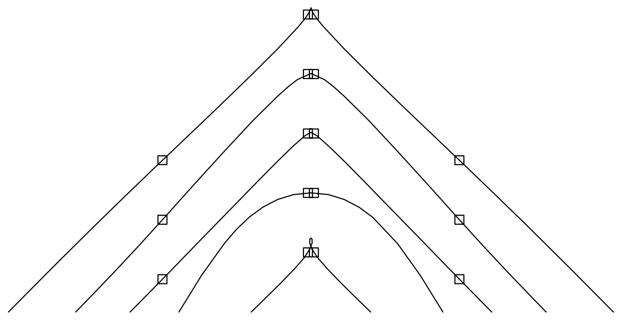

Figure 8.3: A closeup of Figure 8.2.

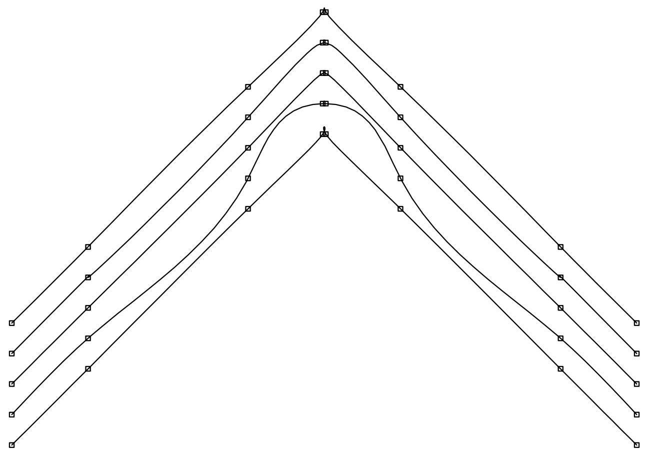

Figure 8.4: The same data set as used in Figure 8.2, now with the affine invariant metric.

ods. The result of the centripetal parameterization is further improved by this change, while the remaining parameterizations keep roughly the same characteristics.

An example where the affine invariant metric significantly improves the result is given in Figure 8.5. It contains data 9 points $d_i = [x_i, y_i]^T$ with $x_i = i$ and $y_i = 0$ except for y_4, which is 10. The interpolation of this data set using the angular parameterization and both the Euclidean and the affine invariant metric produces a thin peak. If the data is sheared and scaled so that $d_4 = [8, 6]^T$, the Euclidean metric causes a loop in the curve. The interpolation using the affine invariant metric on the same data set yields the sheared version of the original interpolant (for better visibility Figure 8.5 actually shows the symmetric problem $d_4 = [0, 6]^T$ for the affine invariant metric).

8.2 End Conditions and Degrees of Interpolants

End conditions have a very strong influence on the shape of an interpolant, and the wrong choice of conditions for a particular data set can cause unacceptable results. For even degrees, none of the end conditions presented in Chapter 3 seems to be able to produce good results. The reason is that for even degrees d, the number $d - 1$ of required end conditions is odd. This means that on one

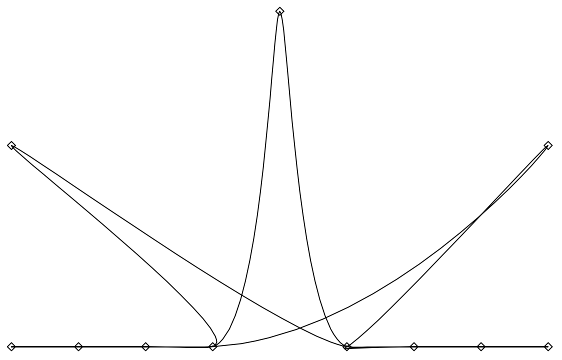

Figure 8.5: The effect of applying the affine invariant metric to sheared data. The central peak is generated using the original data set. If the Euclidean metric is used, a loop is generated for the sheared data set, while with the affine invariant metric the result is a shearing of the original curve.

side of the curve more end conditions have to be specified than on the other side. The result are unacceptable artifacts that are introduced on the side of the curve where less conditions have been specified. This is particularly disturbing if the original data set is symmetric. Such a situation is depicted in Figure 8.6.

The figure shows interpolants of different degrees for the data set from the previous section. The angular parameterization scheme has been used for all interpolations. For even degrees, one more end condition has been specified on the left side than on the right side. While the interpolants of odd degree are smooth and symmetric, those of even degree show large artifacts on the right side. For this particular figure, not-a-knot end conditions have been used, but similar results are achieved with the other end conditions.

In cases where the user has complete control over the interpolation process, the situation for even degrees can be improved, by applying the last "end condition" in the center of the curve. All methods described in Section 3.3 can be applied at an arbitrary parameter value. However, it is not clear how a good parameter value for the last end condition could be found automatically. The

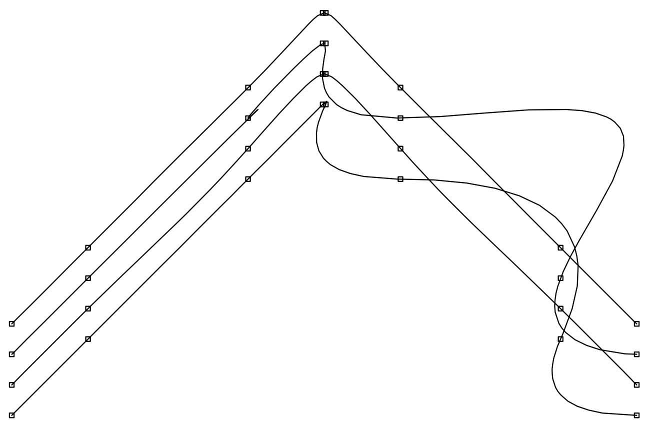

Figure 8.6: For even degrees the number of end conditions is odd. This leads to unacceptable artifacts on the side of the curve, on which less end conditions have been specified. (From bottom to top: degree 2 to 5.)

center of the parameter interval is a good choice if the data set is symmetric, but for asymmetric data it might be as bad of a choice as one of the ends of the curve.

8.3 Interpolation Versus Approximation

Because of these restrictions, it seems questionable whether B-spline interpolation can be used for representing arbitrary functions in mathematical plots. B-spline interpolation certainly has its place in applications where suitable end conditions can be specified by the user, or arise naturally from the problem domain.

However, the plot systems in mathematical applications like MAPLE require a robust algorithm which, without human intervention, produces good representations for a very large variety of functions. The problem of selecting good end conditions and the tendency of interpolation methods to introduce high frequencies between the data points, seriously restrict the use of B-spline interpolation for this task.

B-spline approximation, on the other hand, does not suffer from these deficiencies and is well-suited for representing smooth curves. However, care has to be taken in the selection of the number of sample points and the number of spline segments that are used for the approximation. If these numbers are too low, high-frequency detail of the original curve is lost.

Mathematical functions with discontinuities could be handled by finding the discontinuities first, and then generating a spline representation for each continuous piece of the curve. This removes too high frequencies which can not be handled by the approximation algorithm. The code for locating discontinuities is already available in the current MAPLE library [4], and can optionally be used to improve line-based approximations in the current plot system.

Using this strategy, B-spline approximation is a much more robust algorithm than interpolation. Especially in cases where the original parameterization of the data set is known, it constantly produces fits of high quality. Since it is also more efficient than B-spline interpolation, it is preferable for our purposes.

8.4 Integral Versus Rational Fitting

The already good results of integral B-spline approximation can further be improved using rational splines. In the experiments we have performed the rational approximation always produced at least as good results as the integral approximation. In rare cases all the generated weights were equal, so that the resulting curve was identical to the integral fit. This was mostly the case for data sets that had been sampled from an integral B-spline curve.

Of course the use of rational splines is particularly interesting for functions that have an exact representation as a NURBS curve but not as a B-spline curve, for example arbitrary conic sections. The quality of the rational fit depends on the parameterization that is used. Even rational fitting with exact parameterization can only generate the exact representation of the conic if the conic is parameterized the right way.

For example, a quarter circle in the first quadrant could either be given as the points $[\cos(\alpha), \sin(\alpha)]^T$, where $\alpha \in [0 \ldots \pi/2]$, or in the NURBS representation with control points $[1, 0]^T$, $[1, 1]^T$ and $[0, 1]^T$, as well as weights $(1, \sqrt{2}/2, 1)$, and a knot sequence of $(0, 0, 0, 1, 1, 1)$. The rational fitting process with exact parameterization only retrieves the exact quarter circle if the data points were sampled from the second representation.

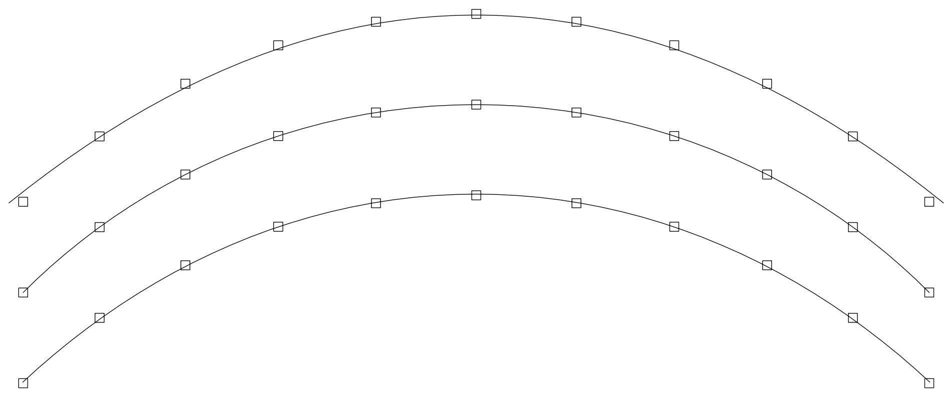
Figure 8.7: An approximation of a quarter circle with integral B-splines (top) and NURBS (bottom). The exact quarter circle is shown in the center.

If the first representation is used, the rational fitting algorithm determines the weight of the second control point as approximately 0.8 (as opposed to $\sqrt{2}/2 \approx 0.707$). This result is independent from the norm used for obtaining the weights (L_1, L_2 and L_∞). The resulting curve is shown in Figure 8.7 together with the integral approximation.

We found that, in practice, all conic sections that are parameterized as B-spline curves can be reconstructed using the exact parameterization and at least $m + 1 = 4 \geq n + \lceil n/e \rceil$ sample points. Note that this only holds for conics, that is, for quadratic NURBS curves with one segment.

As mentioned in Chapter 6, a lower bound for the rank of the matrix used to determine the weights is not known, and thus a sufficient condition for the reproduction of arbitrary NURBS can not be formulated.

Despite this restriction, rational fitting often significantly improves the shape of the curve. It is worth mentioning that throughout all our experiments not a single case was found, in which rational approximation actually produced worse results than integral approximation. Therefore, the decision between integral and rational approximation has to be made based on whether the improvement in quality is worth the performance penalty for determining the weights.

8.5 Performance

Due to the prototype-character of the implementation and some inefficiencies in parts of the MAPLE library that were heavily used through this project, it is hard to make a statement about the relative performance of the different algorithms. In particular, the implementation of B-splines relies on the facility for piecewise functions in MAPLE. Unfortunately, piecewise functions are very slow in the version of MAPLE that was used for the implementation. For example, the evaluation of a cubic B-spline curve with 10 segments at 50 points could take up to 20 seconds on a DEC Alpha system.

The reason for this slowness is that piecewise functions in the available version of MAPLE had been implemented on a high level in the maple library. In the mean time, piecewise functions have been re-implemented on a lower level, and make now use of different optimizations in the MAPLE kernel. These changes significantly improved the performance of piecewise functions. Unfortunately, they are incompatible to the older MAPLE kernel that was available for this thesis, and could therefore not be used.

Without these changes in the library, integral B-spline fitting of 50 data points with 10 B-spline segments of degree 3 takes approximately 8 seconds with the least-squares norm, and around 15 seconds for the L_1 and L_∞ norms. These times could be significantly improved by using specifically tailored algorithms for solving the linear equation systems and the linear programming problems. The combinations of these changes with the faster implementation of piecewise functions should allow for acceptable performance in generating mathematical plots.

The additional overhead for doing a rational approximation turns out to be roughly a factor of two for the L_1 and L_∞ norm, and a factor of three for the L_2 norm (using singular value-decomposition for determining the weights). A table of the measured times is shown below (times are rounded to seconds).

Norm	Points	Segments	Time (integral)	Time (rational)
L_1	50	10	15	27
L_2	50	10	8	23
L_∞	50	10	15	28

Keep in mind that the implementations of the minimization algorithms, that is, for the simplex algorithm and the singular value decomposition, have not been optimized at all. Significant performance improvements would be possible if these algorithms were implemented in such a way, that all computations are done in a purely numeric fashion.

8.6 Conclusions and Future Work

In this thesis we have discussed several ways of representing mathematical functions as integral or rational B-splines. A system has been implemented on top of the computer algebra system MAPLE, which uses these representations to generate resolution-independent mathematical plots, and thus provides a wide range of tradeoffs between image quality and performance in the actual rendering step.

In order to generate these representations, several algorithms including B-spline interpolation and approximation, as well as rational spline fitting have been evaluated.

B-spline interpolation (Chapter 3) is a widely used technique for fitting spline curves through a set of discrete data points. The specific way in which the B-spline knot sequence is selected for this task introduces additional degrees of freedom that can be specified using end conditions. While these end conditions can be useful in applications where they can be specified as additional constraints by the user, the sensibility of the interpolation algorithm to these constraints makes B-spline interpolation inappropriate for automated processes.

An alternative to interpolation is B-spline approximation (Chapter 5), where more data points are specified than control points are available. Since this leads to an over-determined equation system, which can not in general be solved, an approximation to the data has to be found by minimizing the error according to a norm. While almost all the available B-spline literature uses the least-squares (L_2) norm for this task, it has been shown, that the L_1 and L_∞ norm can also be useful, especially for fitting uncertain data. The minimization problem using either of these two norms can be formulated as a linear programming problem, and can thus be solved using the well-known simplex algorithm.

Finally, the possibility of using rational B-splines (NURBS) for the approximation process has been explored (Chapter 6). In this algorithm the weight of every control point is obtained from a constrained minimization process that takes place before the actual curve fitting. Possible norms for this constrained minimization are again L_1, L_2 and L_∞. L_1 and L_∞ minimizations can be achieved using linear programming, while the L_2 norm requires quadratic programming.

The analysis of the rational fitting algorithm yields sufficient conditions for obtaining interpolation of the data points, and necessary conditions for the reconstruction of NURBS curves from discrete sample points. Further research is necessary to find sufficient conditions for the reconstruction of NURBS and conic sections. Results in this area could possibly be achieved by considering the special structure of the B-spline coefficient matrix $\mathbf{B}$.

For interpolation and approximation problems in which the original parameterization is not known, a set of algorithms for determining the parameter value of each data point has been presented (Chapter 4). Unfortunately, the parameterization algorithms that work best (i.e. the angular and the centripetal parameterization) are mostly heuristic methods, and there is only a limited understanding on why they work well. An attempt has been made to modify the angular parameterization so that its contributing factors are geometrically meaningful and thus easier to understand.

Future research in the area of parameterization schemes could involve the development of more theoretically based parameterization methods. Another interesting research topic would be to try to develop a parameterization scheme that reproduces the exact parameterization for data that is sampled from integral or even rational B-splines. That is, given a set of data points, and assuming that these points have been sampled from a B-spline, the method would attempt to generate the parameter value that each point has on the spline.

In summary, we have presented a set of algorithms for representing mathematical functions as splines. We have also created a working prototype implementation of the algorithms described throughout the thesis. For the inclusion of these concepts in a commercial product, however, a substantial amount of optimizations is still necessary.

Appendix A

Derivatives of B-Splines

The derivatives of B-spline curves and surfaces are of importance for determining end conditions for curve and surface interpolation. Fortunately it turns out that these derivatives are linear functions in the control points, and can be interpreted as B-spline curves or surfaces of a lower degree. As a consequence, the computation of these derivatives is relatively simple and efficient. We sum up the most important facts about B-spline derivatives in the following.

From the construction of B-splines, it is clear that the derivative of the basis function plays an important role in determining the derivative of spline curves and surfaces. The following lemma describes the formula for the derivative of a basis function. The proof of this lemma is relatively straightforward and can be found in most of the B-spline literature, for example in [1].

Lemma 2 (Derivative of the B-spline basis functions)
The derivative of the B-spline basis functions $N_i^d(u,T)$ is given by

$$\frac{\mathrm{d}}{\mathrm{d}u} N_i^d(u,T) = \frac{d}{t_{i+d} - t_i} N_i^{d-1}(u,T) - \frac{d}{t_{i+d+1} - t_{i+1}} N_{i+1}^{d-1}(u,T)$$

Proof by induction over the B-spline recursion formula. □

The importance of this lemma lies in the fact that the derivative of a basis function of degree d is the linear combination of two basis functions of degree $d-1$. Recursive application of this formula yields higher order derivatives.

By applying this formula to the definition of B-spline curves, we can calculate the derivative of those curves.

Theorem A.0.1 (Derivative of a B-spline curve)

The derivative of a B-spline curve of degree d with knot vector $T = (t_0, \ldots, t_{n+d+1})$ is given by

$$\frac{\mathrm{d}}{\mathrm{d}u}F(u) = d\sum_{i=0}^{n+1} \frac{(c_i - c_{i-1})}{(t_{i+d} - t_i)} N_i^{d-1}(u, T)$$

where the extra control points $c_{-1} = c_{n+1} = 0$ have been introduced to simplify the result.

Proof:

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}u}F(u) &= \frac{\mathrm{d}}{\mathrm{d}u}\sum_{i=0}^{n} c_i N_i^d(u,T) = \sum_{i=0}^{n} c_i \frac{\mathrm{d}}{\mathrm{d}u} N_i^d(u,T) = \\ &= \sum_{i=0}^{n} c_i \left[\frac{d}{t_{i+d} - t_i} N_i^{d-1}(u,T) - \frac{d}{t_{i+d+1} - t_{i+1}} N_{i+1}^{d-1}(u,T)\right] = \\ &= d\left[\sum_{i=0}^{n} \frac{c_i}{t_{i+d} - t_i} N_i^{d-1}(u,T) - \sum_{i=0}^{n} \frac{c_i}{t_{i+d+1} - t_{i+1}} N_{i+1}^{d-1}(u,T)\right] = \\ &= d\left[\sum_{i=0}^{n} \frac{c_i}{t_{i+d} - t_i} N_i^{d-1}(u,T) - \sum_{i=1}^{n+1} \frac{c_{i-1}}{t_{i+d} - t_i} N_i^{d-1}(u,T)\right]\end{aligned}$$

Setting $c_{-1} = c_{n+1} = 0$ yields the result stated above. □

In a similar fashion we can derive the partial derivative of a tensor-product B-spline surface.

Corollary 2 (Derivative of a tensor-product B-spline surface)

The partial derivatives of a tensor-product B-spline surface of degree $d_u \times d_v$ with knot vectors $T_u = (t_0^u, \ldots, t_{n_u+d_u+1}^u)$ and $T_v = (t_0^v, \ldots, t_{n_v+d_v+1}^v)$ are given as follows

$$\frac{\partial}{\partial u}F(u,v) = d_u \sum_{i=0}^{n_u+1} \sum_{j=0}^{n_v} \frac{c_{i,j} - c_{i-1,j}}{t_{i+d_u}^u - t_i^u} N_i^{d_u-1}(u, T_u) N_j^{d_v}(v, T_v)$$

$$\frac{\partial}{\partial v}F(u,v) = d_v \sum_{i=0}^{n_u} \sum_{j=0}^{n_v+1} \frac{c_{i,j} - c_{i,j-1}}{t_{j+d_v}^v - t_j^v} N_i^{d_u-1}(u, T_u) N_j^{d_v}(v, T_v)$$

Since we can separate the sums for the u and the v direction, the proof is analogous to the curve case. □

Unfortunately, the situation is significantly harder in the case of rational splines. Since a rational spline is the projection of a integral spline from a higher dimension, rational splines can have discontinuities. Also, it is not possible to represent the derivative as a linear function in the control points. While a relatively simple closed form exists for the first derivative, a recursion formula has to be used for higher order derivatives [10].

Appendix B

Linear Programming

A linear programming problem [16] consists of an *objective function* that is linear in some variables, and a set of *equality and inequality constraints* that are linear in the same variables. The goal of linear programming is to determine values for the variables that minimize the objective function while at the same time fulfilling the conditions. This problem may be solved using the *simplex algorithm* or a modification thereof.

For the purposes of this thesis, only a specific form of linear programming problems will be considered, which can be directly used with MAPLE. Using $X := [x_0, \ldots, x_m]^T$ as the vector of variables, this particular form, called *standard form* [16], can be written as

---

Minimize

$$Q^T \cdot X - q_0$$

subject to

$$\mathbf{A} \cdot X = B$$

In addition some of the variables x_i may be restricted to non-negative values: $x_i \geq 0$.

Although this specific form is a restricted version of the more general linear programming problem, it is as powerful as the full version, since problems with arbitrary linear conditions can be rewritten into the above form by introducing additional variables, called *slack variables*.

An example for this kind of transformations is the replacement of inequalities in the side conditions. Every inequality $a \cdot x \geq b$ can be rewritten as $a \cdot x - s = b$ with the additional condition $s \geq 0$. Applying this trick to a linear programming problem of the form

---

Minimize

$$Q^T \cdot X - q_0$$

subject to

$$\mathbf{A} \cdot X \geq B$$

---

yields a new linear programming problem in standard form:

Minimize

$$\begin{bmatrix} Q^T & \mathbf{1} \end{bmatrix} \cdot \begin{bmatrix} X \\ S \end{bmatrix} - q_0$$

subject to

$$\begin{bmatrix} \mathbf{A} & -\mathbf{Id} \end{bmatrix} \cdot \begin{bmatrix} X \\ S \end{bmatrix} = B \quad ; s_i \geq 0$$

Appendix C

The User-Level Interface of the Spline Fitting Library

In the following we briefly describe the user-level functions that have been implemented. The additional options that have been introduced for these functions are then listed in Section C.2.

C.1 User-Level Functions

The user-level functions, which are described in the following, are mostly based on existing `plots` and `plottools` packages that are part of the MAPLE library.

- `SplinePlot`
  This is a replacement for the function `plot` and the continuous version of `spacecurve` from the MAPLE library. The function to be plotted can be specified in one of four ways:

```
SplinePlot( f(x), x=a..b );         # 2-dimensional only
SplinePlot( f, a..b );              # 2-dimensional only
SplinePlot( [x(t), y(t)], t=a..b ); # or the 3-dimensional equivalent
SplinePlot( [x, y], a..b );         # or the 3-dimensional equivalent
```

- `SplinePlot3d`

  This function replaces `plot3d` from the MAPLE library. It fits a tensor-product spline surface through a function that is defined over a rectangular parameter domain. The four variants are

```
SplinePlot3d( f(x,y), x=a..b, y=c..d );
SplinePlot3d( f, a..b, c..d );
SplinePlot3d( [x(s,t), y(s,t), z(s,t)], s=a..b, t=c..d );
SplinePlot3d( [x, y, z], a..b, c..d );
```

- `SplineCurve`

  This function fits a curve through a set of discrete data points that are supplied in a list. It is therefore comparable to the discrete version of the library function `spacecurve`, although it also handles 2-dimensional curves.

- `SplineSurface`

  This is the equivalent to `SplineCurve` for surfaces. Thus it is a replacement for `surfdata`.

- `SplineMatrix`

  This function takes a matrix, and interprets the values as heights over the $z = 0$ plane. It is comparable to `matrixplot`.

- `SplinePlotOptions`

  This function allows global changes to the default of the additional options described in the next section. In addition, it allows changes to the callbacks for the underlying algorithms. This is done using the additional parameter options

  - `parameterization`: the function that generates the parameterization. Currently available are `uniformParam`, `chordLengthParam`, `centripetalParam`, `angularParam` and `areaParam`.
  - `metric`: the metric used for some of the parameterizations. Possible values are `euclideanMetric` and `affineInvariantMetric`.

- `endcondition`: the algorithm for end conditions used in interpolations. Currently available are `naturalEndcond`, `quadraticEndcond`, `notAKnotEndcond`, `besselEndcond`, `closedEndcond` and `multiplePointEndcond`, as well as `clampedEndcond`, which has to be supplied with actual tangent vectors for the left and the right side of the curve.
- `weightMinimizer`: the minimizer used for the weights (`weightsL1`, `weightsL2` and `weightsLinf`).
- `fitMinimizer`: the minimizer used for the control points (`fitL1`, `fitL2` and `fitLinf`).

C.2 Additional Parameter Options

The following list describes the additional options that are available for the spline fitting functions. The original options from `plot` and `plot3d` (see [4]) are still available.

- `numpoints (integer)`

  In analogy to `plot`, specifies the number of sample points used to approximate a curve. The default is 20.

- `grid (integer list)`

  Specifies the number of sample points in u and v direction, similar to the use in `plot3d`. The default is 10×10.

- `degree (integer)`

  The degree of the B-spline or NURBS curve. The default is 3.

- `segments (integer)`

  The number of B-spline segments. The default is -1, which stands for interpolation ($n = m$).

- `rational (boolean)`

  Specifies whether integral or rational splines should be used. The default is `false`, for integral splines.

- `nurbsextension (boolean)`

  This option specifies whether the NURBS extensions to the plot data structure should be used or not. The default is `true`.

Bibliography

[1] Richard H. Bartels, John C. Beatty, and Brian A. Barsky. *An Introduction to Splines for use in Computer Graphics and Geometric Design.* Morgan Kaufmann, 1987.

[2] Johannes Cornelius Gerardus Boot. *Quadratic Programming: Algorithms, Anomalies and Applications.* North Holland Publishing Co., Amsterdam, 1964.

[3] Bruce W. Char, Keith O. Geddes, Gaston H. Gonnet, Benton L. Leong, Michael B. Monagan, and Stephen M. Watt. *Maple V Language Reference Manual.* Springer-Verlag, 1991.

[4] Bruce W. Char, Keith O. Geddes, Gaston H. Gonnet, Benton L. Leong, Michael B. Monagan, and Stephen M. Watt. *Maple V Library Reference Manual.* Springer-Verlag, 1991.

[5] H. B. Curry and I. J. Schoenberg. On polya frequency functions. iv: The fundamental spline functions and their limits. *J. d'Analyse Math.*, 17:71–107, 1966.

[6] Carl de Boor. Total positivity of the spline collocation matrix. Technical Report 25, Ind. Univ. J. Math., 1976.

[7] Carl de Boor. *A Practical Guide to Splines.* Number 27 in Applied Mathematical Sciences. Springer Verlag, 1978.

[8] M. P. Epstein. On the influence of parameterization in parametric interpolation. *SIAM Journal of Numerical Analysis*, 13(2):261–268, April 1976.

[9] Gerald Farin. *Curves and Surfaces for Computer Aided Geometric Design.* Academic Press, 1988.

[10] Gerald Farin. *NURB Curves and Surface: from Projective Geometry to Practical Use.* A K Peters, Inc., 1995.

[11] Thomas A. Foley and Gregory M. Nielson. Knot selection for parametric spline interpolation. *Mathematical Methods in Computer Aided Geometric Design*, pages 261–271, 1989.

[12] Gene H. Golub and Charles F. Van Loan. *Matrix Computations.* John Hopkins series in the mathematical sciences. John Hopkins University Press, 1983.

[13] Wolfgang Heidrich. Advanced computer graphics in a computer algebra environment. In *TRIO/ITRC Researcher Retreat 1995*, May 1995.

[14] Wolfgang Heidrich. An OpenGL driver for MAPLE. Master's thesis, Computer Graphics Group (IMMD IX), Friedrich-Alexander Universität Erlangen-Nürnberg, 1995.

[15] E. T. Y. Lee. Choosing nodes in parametric curve interpolation. *Computer Aided Design*, 21(6):363–370, July/August 1989.

[16] David G. Luenberger. *Introduction to Linear and Nonlinear Programming.* Addison Wesley, 1973.

[17] Weiyin Ma and Jean-Pierre Kruth. Mathematical modelling of free-form curves and surfaces from discrete points with NURBS. *Curves and Surfaces in Geometric Design*, pages 319–326, 1994.

[18] Gregory M. Nielson. Coordinate free scattered data interpolation. *Topics in Multivariate Approximation*, pages 175–184, 1987.

[19] Gregory M. Nielson and Thomas A. Foley. A survey of applications of an affine invariant norm. *Mathematical Methods in Computer Aided Geometric Design*, pages 445–467, 1989.

[20] G. A. Watson. *Approximation Theory and Numerical Methods.* John Wiley & Sons, 1980.