# Basic properties of Expo-Rational B-splines and practical use in Computer Aided Geometric Design

Thesis by

**Arne Lakså**

Dissertation for the degree of dr.philos.
University of Oslo

*Series of dissertations submitted to the*
*Faculty of Mathematics and Natural Sciences, University of Oslo*
*Nr. 606*

ISSN 1501-7710



Cover: Inger Sandved Anfinsen.
Printed in Norway: AiT e-dit, Oslo, 2007.

Produced in co-operation with Unipub AS.
The thesis is produced by Unipub AS merely in connection with the
thesis defence. Kindly direct all inquiries regarding the thesis to the copyright
holder or the unit which grants the doctorate.

*Unipub AS is owned by*
*The University Foundation for Student Life (SiO)*

# Contents

# Preface

This doctoral thesis is submitted in partial fulfilment of the requirements for the degree of doctor philosophiae (dr. philos.) at University of Oslo (UIO). The work has been carried out at Narvik University College (NUC) in the R&D Simulations group. The main goal of this work has firstly been to clarify a theoretical fundament for a new type of splines, Expo-Rational B-splines, secondly, to develop algorithms to use as tools, thirdly to show results from test applications and generally enable an overall preparation for presenting this new type of splines for public use.

In the report there are a lot of figures and examples. There are, of cause, several reasons for this, for instance to show the potential of Expo-Rational B-splines, and how they can be used. For me, another reason has also been important. The figures have acted as a kind of quality control. This is because the equations and the algorithm in this report are the basis for the implementation of the test programmes used for making the figures.

This report concentrates on the practical use of Expo-Rational B-splines in geometric design, simulations and sophisticated shaping. This is very much affected by my experience in practical use of splines in industrial applications and in computer programming.

Expo-Rational B-splines have a lot of similarities with polynomial B-splines. I do not expect Expo-Rational B-splines to replace polynomial B-splines, but rather to complement them. B-splines/NURBS are today the, de facto, industry standard for the representation, design, computations etc. of curves and surfaces. They have been attracting my attention for the last 20 years. Therefore, a short résumé of me experience follows.

In the early 1980s I was engaged in ship construction as a mechanical engineer and constructor, and from that I got an interest for free-form geometry. Then I went back to university, this time to the University of Oslo, where I was introduced to splines in 1985, as a student of Tom Lyche. At the end of the 1980s and at the beginning of the 1990s I was the main contributor to the programming library "SISL", now SINTEF´s spline library (see [38]). In the first half of the 1990s I was, in cooperation with the research center of "Norsk Hydro", an oil company in Norway, implementing a system for geological modeling using B-splines (see [28]). In the period 1990-95 I had contact with several companies, helping them to improve product design and production, using splines. From 1995 I was involved in developing a master program in Engineering Design at Narvik University College, where geometric modeling and splines were one of the main parts of the program. Later, from 1998, I have been responsible for establishing a master program in Computer Simulations and Game Programming, where geometric modeling and

especially splines have been an important part. During this last period I have also been the main contributor in the development of "GM_lib" (see [37]), an in-house geometric modeling programming library developed at Narvik University College.

November 2006 Arne Lakså

# Acknowledgements

First I will express my deepest gratitude to Lubomir T. Dechevsky and Børre Bang. Lubomir T. Dechevsky was the one to introduce the new splines as a topic, and he has been the main responsible for raising this as a research area. He has also been a main consultants for me in my work on Expo-Rational B-splines. Børre Bang has been the main collaborator and discussion partner in the implementation and programming.

Next I am especially grateful to Tom Lyche and Knut Mørken for very valuable discussions and useful advice, and that they are inspiring and uniting the growing Norwegian part of the world spline community.

I am also grateful to Tor Dokken and Bernt Bremdal for unloading me regarding supervising students, and to Arnt Kristoffersen for helping me with GM_lib documentation and other practical issues, and to Susan Jane Berntsen for the language consultations.

I also want to send collective thanks to my other colleagues at Narvik University College for their support, helpfulness and for making an excellent working environment, and also to Narvik University College as a hole for giving the chance to do this work.

Finally, I would like to give big, unreserved, thanks to my family and friends for bearing with me through all this years. Special thanks goes to my wife, Marit, for her patience and support during this period.

Narvik, November 21, 2006.

Arne Lakså

# Chapter 1

# Introduction

"Expo-Rational B-splines" is an entirely new topic in the spline world. Both the analysis of the new spline (which is not the main topic of this work) and the practical use of them, has up to now not been explored in a greater extent. The purpose of this report is to improve this, and to show that Expo-Rational B-splines offer a lot of possibilities.

Perhaps the most promising possibilities is the surface approximation on triangulations (section 6.7), which is opening for the construction of a smooth surface (or a surface with a controlled smoothness), that is built up by a connected set of triangular surfaces. In Figure 1.1 an example is given. The example is a surface made by a connected set of 8 triangular surfaces (the properties and the construction of these triangular surfaces are described further in section 6.7). The surface is first made as an approximation of a sphere, and then edited to the present geometric shape.

Another new and promising feature is "the affine transformation of local functions" (see section 4.5 and 5.4 and several other places). This is opening for creative "geometric editing" and simulations, of which there are a lot of examples in several of the following chapters. The properties and the process is described further in section 5.4. In Figure 1.2 there is an example of the "geometric editing". The surface is a tensor product Expo-Rational B-spline surface, made by first interpolating a torus. Then the surface is edited to its present geometric shape. The blue cubes, which can be seen in the figure, are the editing points which are used when the "geometric editing" is done (this is described further in sections 1.3 and 5.4).

A third feature that is worth mentioning is the strong Hermite interpolation property (described in section 2.3 and in several examples all over). This makes Expo-Rational B-splines very easy to use in modeling. In Figure 1.3 there is an example. The surface in the figure is Hermite interpolating a surface called "Trianguloid Trefoil" (see [1]). at $5 \times 5$ points. At each point the position and a total of 8 partial derivatives are used.

Obviously, this report does not complete the work on Expo-Rational B-splines, it is hopefully only initiating it and, thus, there are many possibilities for further development of this new type of splines. There are a lot of "loose ends" in this report, and several of them are mentioned in the different chapters. It is hoped that these and other topics will be

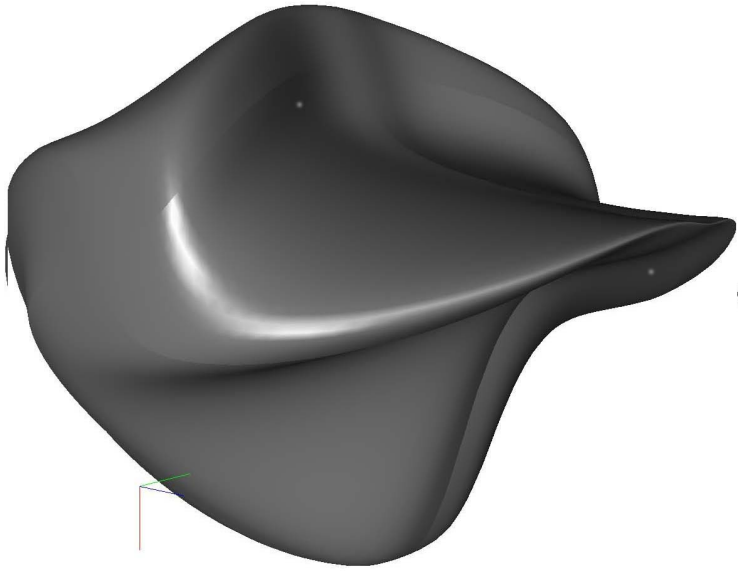

***Figure 1.1:*** *The surface is a connected set of eight triangular Expo-Rational B-spline surfaces. This surface is made by first approximating a sphere. Then the surface is deformed by an editing process.*

taken up by other researchers. It is our sincere wish to develop a growing interest for the Expo-Rational B-spline, because it is our firm belief that Expo-Rational B-splines have a great potential.

The name "Expo-Rational B-splines" refers to the derivative of the Expo-Rational B-spline basis function in the construction (equation 2.4), which is (in most cases) an exponential function with a rational exponent, and which in addition also has a connection to polynomial B-splines. In Expo-Rational B-splines there are several characteristics that we also will find in linear polynomial B-splines. This will clearly be demonstrated later in this report.

In the remaining part of this introduction there will be a review of the development of the Expo-Rational B-splines so far. Then challenges in implementation will be described. There will also be a description of the algorithmic language used in this report. Then there will be a short review of the benefit of using Expo-Rational B-splines in possible new applications, and finally there will be a short overview of the rest of the report.

## 1.1 Historical notes

The development of Expo-Rational B-splines has been one of the main working areas of the R&D group for Mathematical Modeling, Numerical Simulation and Computer Visual-

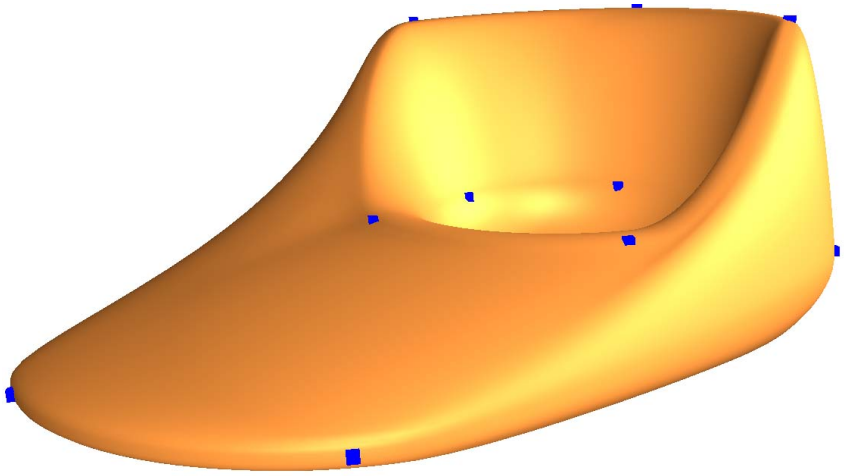

***Figure 1.2:*** *This tensor product Expo-Rational B-spline surface is made by first Hermite-interpolating a torus at $5 \times 5$ points. The positions of the interpolating points are seen as cubes. The surface is then edited by moving the blue cubes, so the surface finally is getting into its present shape.*

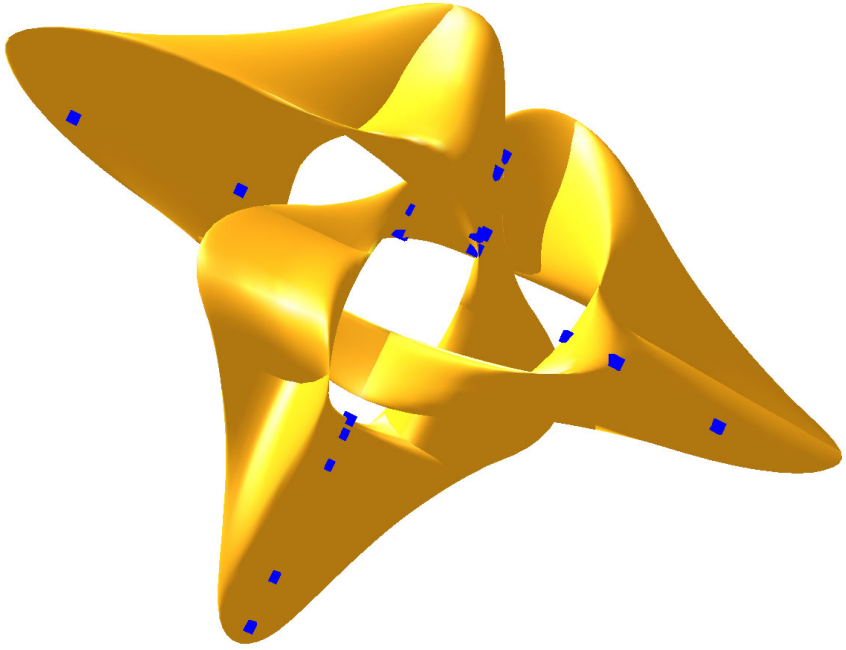

***Figure 1.3:*** *This tensor product Expo-Rational B-spline surface is made by Hermite interpolating a "Trianguloid Trefoil" (see [1]) at $5 \times 5$ points. The positions of the interpolating points are seen as blue cubes.*

ization (shortly the Simulations R&D group) at Narvik University College. The research topic began in the summer of 2003 when the leader of the Simulation R&D group professor Lubomir Dechevsky gave a definition of the new B-splines, suggested the name "Expo-Rational" for it, and proposed that the development of the theory and applications of this new B-splines become a strategic priority of the Simulations R&D group.

Lubomir Dechevsky refers to the definition of the Expo-Rational B-splines as a process of "gradual ripening of the concept" which has continued for more than 10 years, since the beginning of the 1990s till the actual process. Dechevsky has been inspired by a number of heuristic predecessors of the ERBS constructions arising in approximation theory, functional analysis, differential geometry and operator theory, which are all described in Section 3 of [18]. As described in [16] an important step has been also the meeting of Dechevsky with Charles Micchelli at a conference in 1997 and the continuing discussion between them after this meeting, which lasted from 1997 to 1999.

The theoretical fundament, and an overview of relevant research topic and applications of the Expo-Rational B-splines, are given in the main theoretical paper [16]. This paper was written by Lubomir Dechevsky, with a major contribution by this author (Arne Lakså), especially in the triangular (simplex) case, and graphic visualization provided by Børre Bang. This theoretical paper were preceded by [36] which gave the first practical orientation into the design and use of splines. This paper was written by this author, with contribution from Børre Bang and Lubomir Dechevesky. Together with [19], these were the three initial paper which introduced Expo-Rational B-splines to the public.

The story began at the 6th International Conference on Mathematical Methods for Curves and Surfaces in Tromsø, on 5th July 2004. Three talks were given;

- "Expo-Rational B-splines" by Lubomir Dechevesky,
- "Exploring Expo-Rational B-splines for Curves and Surfaces" by Arne Lakså,
- "Expo-Rational Spline Multiwavelets" by Niklas Grip.

The first published article is in the proceedings of the conference [36]. The main article on Expo-Rational B-splines was first a preprint [18], but can now be found as a published article in [16].

Then on 12th October 2004 at the "eVITA" workshop in Oslo another talk was given,

- The "NUERBS form of Expo-Rational B-splines", by Arne Lakså.

This talk also resulted first in a preprint [19], and later in a published article [17] (see also section 7.1 about NUERBS). All published articles and preprints so far are the result of joint work by Lubomir Dechevsky, Børre Bang and Arne Lakså (this author).

The R&D group for Mathematical Modeling, Numerical Simulation and Computer Visualization (shortly The R&D "Simulations" group) at Narvik University College collaborates closely with "SINTEF Applied Mathematics" in Oslo, where Tor Dokken, is a part-time member of the group. The group is also an associated member of the "Center of Mathematics for Applications", CMA, at the University of Oslo, and has, therefore, discussed the explorations of "Expo-Rational B-splines" with Tom Lyche and Knut Mørken.

## 1.2 Motivation and possible applications

The purpose of Expo-Rational B-splines is to use them as basis functions in a compound spline function. Therefore, in Expo-Rational B-splines we have, analogous to polynomial B-splines, introduced knot vectors in the definition of the basis functions. The main difference is that, instead of an ordinary coefficient vector, there is a vector of coefficient functions also called local functions (see chapter 2 for definitions). There are, of course, many other differences between Expo-Rational B-splines and polynomial B-splines which are further discussed in the next chapter. These differences and the properties they give are of course the motivation for introducing this new type of B-spline. Some of the reasons for introducing Expo-Rational B-splines are, thus:

- The $C^\infty$-smoothness on $\mathbb{R}$ property of the Expo-Rational B-splines and the implied property for a compound spline function.
- The special Hermite interpolation property that makes it very easy to approximate a geometric object.
- The minimal local support of the basis functions, over two knot-intervals as in linear polynomial B-splines.
- The easy and flexible "geometric editing" possibilities.
- The possibilities for dynamic shape transformation by simple affine transformations (rotation, scaling, translating) of local functions.
- The great potential for creating triangular surfaces with arbitrary smoothness.

The shapes one can construct using Expo-Rational B-splines are much more "sophisticated" than what it is currently possible to do with B-splines, NURBS or other spline types. Some examples of this can be seen in the chapters 4, 5 and 6.

"Advanced shape modeling" is defined to be complex curve/surface/object modeling using essentially combinations and compositions of free-form geometry, and where one might request control of smoothness, fast changing directions, topological consistence, etc. (for example, class A curves and surfaces, see [24]).

Using this definition, it is clear that advanced shape modeling using the modeling tools that are available today is not at all easy. In modeling for the virtual world (computer games, movies etc.), there is a desire to get more useful tools. Product design is also drawn towards more sophisticated shapes.

Therefore, there are two types of applications that were in mind for use in the future, and also for use in the present implementation and testing of Expo-Rational B-splines,

1. Shape modeling tools according to the description given above.
2. Simulation tools for interactive moving geometry. Examples are given in figures 4.23 and 4.24, and in section 7.2.

## 1.3 Challenges regarding implementation

Many theoretical and practical problems arise when implementing Expo-Rational B-splines. The following is a list of some of them:

- Simplifying the complexities regarding the basis functions and their derivatives.
- Making a reliable, precise and fast evaluator.
- Simplifying the complexity regarding a complete evaluator.
- Speeding up the algorithm by using pre-evaluated basis functions.
- Implementing the Hermite interpolation of local functions,

There are of course several other challenges in the implementation work. First there is a need in the implementation environment for an API to simplify the user-interface, a graphic environment (OpenGL and GUI), tools for linear algebra (matrix computations etc.). In some test examples there might be a need for a FFT routine (example in [6]), simple ODE/PDE solvers (example in [46]). For speeding up the computations it could also be useful to exploit GPGPU programming (see [21]). This is only a short list gathered from our experience.

During the development of a test environment we have extended the inhouse C++ programming libraries "GM_lib" (described in [37]) and "GM_wave" (described in [45]), developed at Narvik University College, to also include state-of-the-art classes for Expo-Rational B-splines. We have also introduced a mechanism for geometric editing and simulations. In addition we have made a special class for the fast evaluation of Expo-Rational B-spline basis functions (see section 3.4). This class is independent of other libraries. The source code for this evaluation class can be found at the project homepage: "`http://www.hin.no/simulations/ERBS`".

We have also developed a test program for constructing Expo-Rational B-splines by Hermite interpolation of given curves/surfaces at sets of given points. Because of this it has been necessary to introduce several types of local curves and surfaces. In addition we have implemented an editing functionality by introducing a "representative" (a cube which visually replaces a local curve/surface/etc. on the screen), on which one can interactively do local affine operations like rotations, translations and scalings. In the examples (figures), the "representatives" can, be seen as blue cubes. We have also introduced "selector" (a visual cube representing a control point in the control polygon for a local Bézier curve/surface on the screen). Selectors can only be translated interactively. In the figures, "selectors" can be seen as green cubes.

The "selector" and an appurtenant "selectorgrid" represents the control polygon of a B-spline/Bezier curve/surface, and the "selector" thus makes it possible to edit the control polygon. The "representative" is the Expo-Rational B-spline analogy of the control polygon, and can be seen in a lot of examples. It is important that the "representative" is a cube so that the orientation and not only the position can be seen.

## 1.4 Algorithmic language

By “algorithmic language” we mean the language used to describe the algorithms that are developed in this report. All programming is done in C++. This is naturally also reflected in the algorithmic language. One important factor in the definition of the syntax is that it should be compact and at the same time clear and easy to understand. The aim of an algorithmic description is, therefore, to be a recipe for an implementation, and to explain and make it easier to understand how the algorithm works.

The items that describe the algorithmic language are

- The notation in the algorithm is a simplification and essentially a mix of C++ and a mathematical notation.
- C++ template notation is used. There are also types from the C++ standard template library - stl (example, vector<double>, a vector of numbers in double precision).
- To make the notation more compact, begin “{” and end “}” are only marked by indentation.
- A routine might only be notated as a set, as example $vector\langle T\rangle\ C\ =\ \{D^j c(t)\}_{j=0}^{n}$. But it will always be followed by an explaining comment.
- Ordinary C++ standard is used for comments.

Below is an example, an implementation of a function $f_2(t)$ defined in equation (2.38). The C++ code will first be shown, then the description by the algorithmic language.

```
double f2(double t)
{
    if(t==lambda || t<2.3e-308 || t==1.0)    return 0.0;

     double h = (t-1/(1+gamma))*abs(t-lambda)/(t*(1-t));
     if(t<lambda)      h -= 1.0;
     else              h += 1.0;
     h *= -_sk*alpha*beta*
                  (1+gamma)/pow(t*pow(1-t,gamma),alpha);

     if((1+gamma)*alpha < 1)
     {
         double g = pow(abs(t-lambda),1-(1+gamma)*alpha)/h;
         if(g < 2.3e-308)     return 0.0;
         else                 return 1/g;
     }
     else if((1+gamma)*alpha > 1)
         return  h * pow(abs(t-lambda),(1+gamma)*alpha-1);
     else
         return  h;
}
```

```
double f2 ( double t )
    if ( t < 2.3e-308 || t == λ || t == 1 )      // See (2.38), upper part.
        return 0.0;
    double h = (t-1/(1+γ)) / (t(1-t)) |t-λ|;     // First part of the second factor from (3.3).
    if ( t < λ )   h -= 1;                       // Last part of second factor (3.3).
    else           h += 1;
    h *= -S_k αβ(γ+1) / (t(1-t)^γ)^α;            // Inserting S_k and the first factor of (3.3).
    if ( (1+γ)α < 1 )                            // Asymptote at t = λ is present.
        double g = |t-λ|^(1-(1+γ)α) / h;         // The inverse of (3.3).
        if ( g ≠ normal value )  return 0;
        else                     return 1/g;
    else if ( (1+γ)α > 1 )                       // Ordinary solution.
        return h |t-λ|^((1+γ)α-1);
    else                                         // Discontinuity at t = λ.
        return h;
```

This example can be found in section 3.2, algorithm 3. The C++ code in this example can be found in the evaluator class used in our test program (see section 1.3). The function is a member function of this evaluator class, the variables; _sk, alpha, beta, gamma and lambda are all members of the class, and thus initiated and ready for use.

The differences between the C++ code and the algorithm are not very big. We can see that the main difference is that the formula and the comments and especially the references to the equations in the report, make the algorithm easier to understand. It is actually only a formalization of the text in the report.

## 1.5 Overview of the report

There are a total of 7 chapters in this report, and their purpose is to introduce the Expo-Rational B-spline and to show that these new splines can be used in geometric modeling, design and simulation. Here is a short summary of the remaining chapters.

Chapter 2 defines the Expo-Rational B-spline, “ERBS” . The basic properties are stated and proved. A subset called the scalable subset of ERBS, and the so-called default set are also introduced. These sets are introduced because they are more convenient to use in geometric design. In addition to introducing the basis function, the composite ERBS function, including knot vectors and interpolation properties, is also introduced.

Chapter 3 concentrates on the ERBS evaluator. This is essentially used to develop a reliable, precise and efficient evaluator for the ERBS basis function. The chapter discusses reliability with regards to the IEEE standard for binary floating-point arithmetic. It also introduces algorithms for evaluation of ERBS, including their derivatives. Tests are made

with respect to both precision and efficiency. At the end of the chapter a special reliable and fast evaluator, wrapped into a C++ class, is introduced.

ERBS curves are the topic of chapter 4. The definition is given and practical aspects concerning implementation are discussed. A complete curve evaluator including derivatives is developed, and two different types of local curves are introduced. The construction, and the use of ERBS curves are discussed, and a lot of examples are given.

Chapter 5 introduces tensor product ERBS surfaces. Both definitions and aspects concerning implementations are discussed. In addition complete evaluators are introduced, as are Bézier patches as local patches. Hermite interpolation is discussed, and free form sculpturing using affine transformation of local patches is also considered.

Chapter 6 deals with triangular surfaces. First there is a short repetition/definition of homogeneous barycentric coordinates and Bézier triangles. Then Expo-Rational B-spline basis functions in homogeneous barycentric coordinates are introduced, and the basic properties are discussed. Then the ERBS-triangle is introduced, including two types of local triangles, the Bézier triangle and the sub-triangle in general parameterized surfaces. The last one leads to surface approximations on triangulations. Both chapter 5 and 6 contain many examples.

The last chapter, 7, sums up the report, as well as it includes two additional sections. These sections briefly summarize two subjects, namely NUERBS and three-variate tensor product ERBS. The first of these refers to an earlier published article, and the second one partly discusses the Master thesis of a student at Narvik University College (NUC).

# Chapter 2

# "ERBS" – definitions and basic properties

The purpose of this chapter is to introduce the new type of B-splines, Expo-Rational B-splines (abbreviated ERBS), to give the definitions, show the basic properties, to explore some of these properties, to study some of the possibilities for applications of these properties, and to give some examples. The chapter has the following organization,

- A simple version of the Expo-Rational B-splines will first be introduced, followed by a short description of their basic properties (section 2.1).
- Next there follows a general definition of Expo-Rational B-splines (section 2.2). Compared to the simple version in section 2.1, this definition introduces higher degrees of freedom as it involves 5 intrinsic parameters. The reason for this is to raise the possibilities for use in applications, such as approximating solution spaces for PDE/ODE´s, modeling complex phenomena in nature and experimental simulations.
- In section 2.3 are presented the basic properties of the general definition of Expo-Rational B-splines. We also give the range of the parameters in which these properties hold.
- The "scalable subset" is defined in section 2.4. It defines a limited subrange of the intrinsic parameters of Expo-Rational B-splines better fitted for use in interactive visualization, modeling and simulation. Within the range of the scalable subset it is possible to speed up the computations tremendously.
- In section 2.5 particular attention is given to the default intrinsic set and a study of the derivatives of the ERBS for these intrinsic parameters. The default set of Expo-Rational B-splines is within the scalable subset, and is used in most of the examples in this report.
- In section 2.6 we define a general ERBS-based function as a linear combination of Expo-Rational B-splines, with coefficients which may be functions (not necessarily scalars/points). These "coefficient functions" will be called "local functions".

- In section 2.7 we discuss the multiplicities in a knot vector and the continuity properties of a compound spline function which these multiplicities generate.
- The Hermite interpolation property of fitting the "global" function to the "local" functions in the knots is investigated and discussed in section 2.8. A short recipe of how to make an Expo-Rational B-spline function interpolating a known function is given.
- In the last section of this chapter, 2.9, examples are given of the effect of separately varying each of the intrinsic parameters. The section will give some ideas about the diversity of possible shapes of the Expo-Rational B-splines.

**Remark 1.** *The simple version described in section 2.1, is actually the same as the default set described in section 2.5, but the default set is, according to the computation, further simplified.*

## 2.1 A simple version

Let $t_k \in \mathbb{R}$ and $t_k < t_{k+1}$ for $k = 0,1,2,...,n,$ i.e. we consider a strictly increasing knot vector $\{t_k\}_{k=0}^{n+1}$. A simple version of an Expo-Rational B-spline (which is the same as the "default set" in section 2.5) is then defined by the following.

**Definition 2.1.** *The simple version of an Expo-Rational B-splines (ERBS) associated with the (strictly increasing) knots $t_{k-1}$, $t_k$ and $t_{k+1}$ is as follows:*

$$B_k(t) = \begin{cases} \dfrac{\displaystyle\int_{t_{k-1}}^{t} \psi_{k-1}(s)ds}{\displaystyle\int_{t_{k-1}}^{t_k} \psi_{k-1}(s)ds}, & \text{if } t_{k-1} < t \le t_k \\ \dfrac{\displaystyle\int_{t}^{t_{k+1}} \psi_k(s)ds}{\displaystyle\int_{t_k}^{t_{k+1}} \psi_k(s)ds}, & \text{if } t_k < t < t_{k+1} \\ 0, & \text{otherwise} \end{cases} \tag{2.1}$$

*with*

$$\psi_k(t) = e^{-\frac{\left(t-\frac{t_k+t_{k+1}}{2}\right)^2}{(t-t_k)(t_{k+1}-t)}} \tag{2.2}$$

This means that $B_k(t)$ is defined on $\mathbb{R}$, and its support is $[t_{k-1}, t_{k+1}]$, which is the minimal possible support for continuous B-splines to satisfy partition of unity (see basic property **P2** below, section 2.3 and the text about the partition of unity in e.g [13], [48] and [23]).

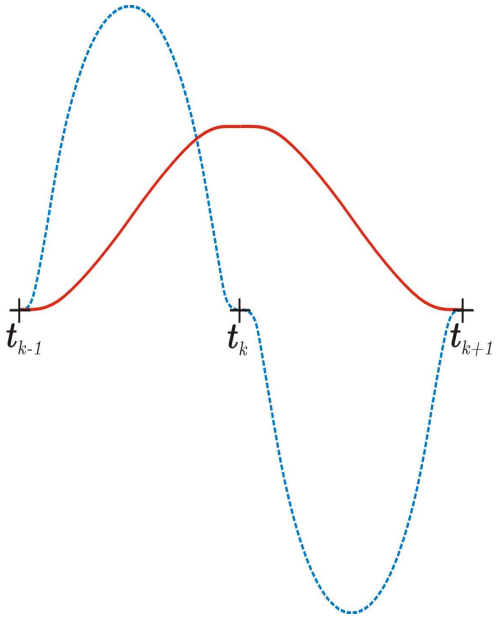


***Figure 2.1:*** *A graph of $B_k(t)$ (solid red) and its first derivative (dotted blue). The knots $t_{k-1}, t_k$ and $t_{k+1}$ are also marked on the plot.*

An example of the plot of $B_k(t)$, $\psi_{k-1}(t)$ and $-\psi_k(t)$ is illustrated in Figure 2.1. In this figure $\psi_{k-1}(t)$ and $-\psi_k(t)$ are scaled so that the integral on their respective knot interval is 1 for each of them.

A list of the basic properties of $B_k(t)$ follows. The three first properties are also properties of linear B-splines. The two last properties are unique among spline basis functions, and are influencing very much the way we can use Expo-Rational B-splines.

**P1.** Every basis functions $B_k(t)$ is positive on $(t_{k-1}, t_{k+1})$ and zero otherwise.

**P2.** The set of basis functions $B_k(t)$ for $k = 1, 2, ..., n$ form a partition of unity on $(t_1, t_n]$. It follows that if $t_k < t \leq t_{k+1}$ then $B_k(t) + B_{k+1}(t) = 1$.

**P3.** For $k = 1, 2, ..., n$ $B_k(t_k) = 1$ holds, which is also a property of the Lagrange form of polynomials.

**P4.** Every basis function $B_k(t)$ is $C^\infty$-smooth on $\mathbb{R}$.

**P5.** All derivatives of all basis functions are zero at their interior knot $t_k$. In fact they are zero at every knots.

In section 2.3 there is a more general version of these properties and the proof of the properties are given for this more general version. Property **P4** is in section 2.3 divided into 2 properties, 4 and 5. It, thus, follows that **P5** goes to property 6 in section 2.3.

## 2.2 Generalized definition

Let $t_k \in \mathbb{R}$ and $t_k \le t_{k+1}$ for $k = 0,1,2,...,n$, i.e., we consider an increasing knot vector $\{t_k\}_{k=0}^{n+1}$. The generalized definition of an Expo-Rational B-spline (ERBS) is based on an increasing knot vector, and for each knot interval, a set of 5 different intrinsic parameters (see [36], [18] and [19]). The definition is according these previous articles (and was first proposed by Lubomir Dechevsky).

**Definition 2.2.** *An Expo-Rational B-spline, associated with three increasing knots $t_{k-1}$, $t_k$ and $t_{k+1}$, $B_k(t) = B_k(\alpha_{k-1},\beta_{k-1},\gamma_{k-1},\lambda_{k-1},\sigma_{k-1},\alpha_k,\beta_k,\gamma_k,\lambda_k,\sigma_k;t)$ is defined by*

$$B_k(t) = \begin{cases} \dfrac{\int_{t_{k-1}}^{t} \psi_{k-1}(s)ds}{\int_{t_{k-1}}^{t_k} \psi_{k-1}(s)ds}, & \text{if} \quad t_{k-1} < t \le t_k, \\ \dfrac{\int_{t}^{t_{k+1}} \psi_k(s)ds}{\int_{t_k}^{t_{k+1}} \psi_k(s)ds}, & \text{if} \quad t_k < t < t_{k+1}, \\ 0, & \text{otherwise}, \end{cases} \tag{2.3}$$

*with*

$$\psi_k(t) = e^{-\beta_k \frac{|t-((1-\lambda_k)t_k+\lambda_k t_{k+1})|^{2\sigma_k}}{\left((t-t_k)(t_{k+1}-t)^{\gamma_k}\right)^{\alpha_k}}}, \tag{2.4}$$

*where*

$$\alpha_k > 0, \quad \beta_k > 0, \quad \gamma_k > 0, \quad 0 \le \lambda_k \le 1, \quad \sigma_k \ge 0.$$

**Remark 2.** *Equation (2.3) allows the use of multiple knots. As with polynomial splines, here multiple knots are also a tool for making discontinuities.*

Multiple knots are discussed further in the sections 2.3 and 2.7.

To simplify the equations and to prepare for their practical use, we make here some preliminary preparations. First a scaling factor is introduced according to the normalizing factors in definition 2.2.

**Definition 2.3.** *An ERBS scaling factor is a constant defined as follows,*

$$S_k = \frac{1}{\int_{t_k}^{t_{k+1}} \psi_k(t)dt} \quad \text{when} \quad t_k < t_{k+1}.$$

To study the derivatives of the ERBS defined in definition 2.2 we need some additional notations. If the respective limits exist, the right derivative will be,

$$D_+ f(x) = \lim_{h \to 0_+} \frac{f(x+h) - f(x)}{h},$$

and the left derivative

$$D_- f(x) = \lim_{h \to 0_+} \frac{f(x) - f(x-h)}{h}.$$

If both right and left derivatives exist and are equal, there is

$$Df(x) = D_+ f(x) = D_- f(x).$$

If only one of them exists, we denote

$$Df(x) = \begin{cases} D_+ f(x), & \text{if } D_- f(x) \text{ do not exist,} \\ D_- f(x), & \text{if } D_+ f(x) \text{ do not exist.} \end{cases}$$

We now introduce a more detailed version of equation 2.3, also including the derivatives,

$$B_k(t) = \begin{cases} S_{k-1} \displaystyle\int_{t_{k-1}}^{t} \psi_{k-1}(s)ds, & \text{if } t_{k-1} < t \leq t_k, \\ S_k \displaystyle\int_{t}^{t_{k+1}} \psi_k(s)ds, & \text{if } t_k < t < t_{k+1}, \\ 0, & \text{otherwise,} \end{cases} \tag{2.5}$$

and where the derivative of an Expo-Rational B-spline is

$$DB_k(t) = \begin{cases} S_{k-1}\, \psi_{k-1}(t), & \text{if } t_{k-1} < t \leq t_k, \\ -S_k\, \psi_k(t), & \text{if } t_k < t < t_{k+1}, \\ 0, & \text{otherwise.} \end{cases} \tag{2.6}$$

For all derivatives ($j > 0$) we get the following general equation,

$$D^j B_k(t) = \begin{cases} f_{j,k-1}(t)\, \psi_{k-1}(t), & \text{if } t_{k-1} < t \leq t_k, \\ -f_{j,k}(t)\, \psi_k(t), & \text{if } t_k < t < t_{k+1}, \\ 0, & \text{otherwise,} \end{cases} \tag{2.7}$$

where $f_{1,k}(t) = S_k$. In Figure 2.1, there is an example of an Expo-Rational B-spline basis and its first derivative. The next derivatives involve $f_{j,k},\ j = 2,3,\ldots$ that are functions computed recursively, as follows below.

Due to the complementary integration limits $[t_{k-1}, t_k]$ and $[t_k, t_{k+1}]$, the equation (2.5-2.7) reflect the symmetry around the interior knot $t_k$. It can clearly be seen that, because of the scaling on $[t_{k-1}, t_k]$ and $[t_k, t_{k+1}]$, $B_k(t)$ in (2.5) is continuous on $(t_{k-1}, t_{k+1})$.

The structure of the equation 2.7 for the derivatives of higher order is simplified by using a factor $f_{j,k}(t)$. This factor will be discussed below, and in section 2.5 will be given the seven first factors for the default set of intrinsic parameters. These factors will be plotted in Figure 2.8 (separated in numerator and denominator). One important property of the derivatives is that all derivatives for all ERBS basis functions, under some constraints on the intrinsic parameters, are zero at all knots. The proof of this is given in Theorem 2.1. The consequences of this are very important for the use of ERBS.

To take a closer look at the derivatives, and thus $f_{j,k}(t)$, we need some definitions and simplifications. First we consider the linear function in the numerator in the exponent of $\psi_k(t)$ in equation 2.4, which will be denoted by

$$c_k(t) = t - ((1-\lambda_k)\,t_k + \lambda_k\,t_{k+1})\,. \tag{2.8}$$

Using (2.8), we define $g_k(t)$ to be the derivative with respect to $t$ of the exponent in (2.4). Thus the extra factor generated when computing $D\psi_k(t)$ is

$$g_k(t) = \beta_k \left( \alpha_k \left( \frac{1}{t-t_k} - \frac{\gamma_k}{t_{k+1}-t} \right) c_k(t) - 2\sigma_k \right) \frac{c_k(t)\left(c_k(t)^2\right)^{\sigma_k - 1}}{\left((t-t_k)\,(t_{k+1}-t)^{\gamma_k}\right)^{\alpha_k}}. \tag{2.9}$$

Computing the first derivative of $g_k(t)$ according to $t$, we get

$$\begin{aligned} g_k'(t) = \beta_k \Big( & 2\sigma_k \left( \alpha_k \left( \tfrac{1}{t-t_k} - \tfrac{\gamma_k}{t_{k+1}-t} \right) c_k(t) - 1 \right) - \alpha_k \left( \tfrac{1}{(t-t_k)^2} - \tfrac{\gamma_k}{(t_{k+1}-t)^2} \right) c_k(t)^2 + \\ & \left( 2\sigma_k - \alpha_k \left( \tfrac{1}{t-t_k} - \tfrac{\gamma_k}{t_{k+1}-t} \right) c_k(t) \right) \left( \tfrac{\alpha_k (t_{k+1} - t(1+\gamma_k) + t_k\gamma_k) c_k(t)}{(t-t_k)(t_{k+1}-t)} \right) \Big) \frac{\left(c_k(t)^2\right)^{\sigma_k - 1}}{\left((t-t_k)(t_{k+1}-t)^{\gamma_k}\right)^{\alpha_k}}. \end{aligned} \tag{2.10}$$

We now investigate the formulas for $f_{j,k}(t)$, $j = 1,2,3,....$ Using the general rules for derivations of products we get, after computations, the following recursive definition

$$f_{j,k}(t) = f_{j-1,k}'(t) + f_{j-1,k}(t)\; g_k(t), \quad j > 1, \tag{2.11}$$

where $f_{j,k}(t)$, $j > 1$, are rational functions at least when $\alpha_k$, $\gamma_k$ and $\frac{\sigma_k}{2}$ are nonnegative integers. Starting with $f_{1,k}(t) = S_k$ we can expand definition (2.11) to get the next functions. For $j = 2,3,4,5$ we get

$$\begin{aligned} f_{1,k}(t) &= S_k, \\ f_{2,k}(t) &= S_k\; g_k(t), \\ f_{3,k}(t) &= S_k \left( g_k'(t) + g_k^2(t) \right), \\ f_{4,k}(t) &= S_k \left( g_k''(t) + 3g_k(t)g_k'(t) + g_k^3(t) \right), \\ f_{5,k}(t) &= S_k \left( g_k'''(t) + 3g_k'^2(t) + 4g_k(t)g_k''(t) + 6g_k^2(t)g_k'(t) + g_k^4(t) \right). \end{aligned} \tag{2.12}$$

where $g_k''$ and $g_k'''$ are computed recursively from 2.10.

The question if $D^j B_k(t)$ is continuous or at least is defined everywhere on the two open intervals $(t_{k-1}, t_k)$ and $(t_k, t_{k+1})$ is being investigated in Theorem 2.1, and in the proof on page 23 and in section 2.4.

## 2.3 Basic properties

In this section, six basic properties for the Expo-Rational B-splines (ERBS) will be presented. For the last three properties which concern the value of derivatives at the knots and the continuity of all derivatives in the case of simple knots, there are some restrictions on the intrinsic parameters.

**Theorem 2.1.** *There are six basic properties of the basis function* $B_k(t),\ k = 1,...,n$. *These six properties are:*

*Properties* **P1**–**P3** *are shared by both ERBS and linear B-splines.*

**P1** *is the nonnegativity and the local support, i.e.,*

$$B_k(t) \begin{cases} > 0, & \text{if} \quad t_{k-1} < t < t_{k+1}, \\ = 0, & \text{otherwise}. \end{cases}$$

**P2** *is that the set of basis functions forms a partition of unity (an affine combination), i.e.,*

$\sum_{k=1}^{n} B_k(t) = 1$, *for every* $t \in (t_1, t_n]$,

*it follows that* $B_k(t) + B_{k+1}(t) = 1$ *if* $t \in (t_k, t_{k+1}]$.

**P3** *is about the value at the interior knot* $t_k$ *and the continuity at the interior knot* $t_k$ *for simple knots, i.e.,*

$B_k(t_k) = 1$ *if* $t_{k-1} < t_k$, *and*

$\lim_{t \to t_{k+}} B_k(t) = 1$ *if* $t_k < t_{k+1}$.

*The next three important properties* **P4**–**P6** *holds only for ERBS but depend on the following three additional restrictions on the intrinsic parameters,*

1. $\sigma \in \{0\} \cup \mathbb{N}$, *for property* **P4** *below (at both intervals* $k-1$ *and* $k$*),*
2. *either* $\lambda > 0$, *or* $\lambda = 0$ *and* $2\sigma < \alpha$, *for property* **P4** *(at interval* $k$*) and* **P5** *(at interval* $k-1$*) below,*
3. *either* $\lambda < 1$, *or* $\lambda = 1$ *and* $2\sigma < \gamma\alpha$, *for property* **P4** *(at interval* $k-1$*),* **P5** *(at interval* $k$*) and* **P6** *(at interval* $k-1$*) below.*

**P4** *is that every basis function is in* $C^{\infty}(t_{k-1}, t_{k+1})$.

**P5** *is that in the case of simple knots, the basis functions are* $C^{\infty}$*-smooth at the respective exterior knot,* $t_{k-1}$ *or* $t_{k+1}$*, i.e.,*

$\lim_{t \to t_{k-1+}} D^j B_k(t) = 0$ *for* $j = 0, 1, 2, \ldots,$ *if* $t_{k-1} < t_k$ *and*

$\lim_{t \to t_{k+1-}} D^j B_k(t) = 0$ *for* $j = 0, 1, 2, \ldots,$ *if* $t_k < t_{k+1}$.

**P6** *is that all derivatives are zero at their interior knot, i.e.,*

$D^j B_k(t_k) = 0$ *for* $j = 1, 2, ...,$

*and they are actually zero at every knot.*

Before the proof follows there will first be two remarks and a list of consequences of the properties. Besides this list, consequences from the properties will be illustrated with examples in the Following sections and chapters.

**Remark 3.** *The name Expo-Rational B-splines refers to that the exponent of $\psi_k$ in (2.4), is a rational function. Because the intrinsic parameters $\sigma$, $\alpha$ and $\gamma$ are not only positive integers, the exponent in $\psi_k$ are actually extended to general functions. The exponent of $\psi_k$ will not be a true rational polynomial function if $\sigma = \frac{n}{2}$, for $n = 1, 2, ...$ and $\alpha$ and $\gamma$ are positive integers. This is because of the absolute value used in the numerator of the exponent. This function can also create discontinuities in the derivatives of the ERBS (in the interior of knot intervals). A true Rational polynomial function where $\sigma$, $\alpha$ and $\gamma$ all are positive integers is entirely inside the restrictions on the intrinsic parameters in Theorem 2.1, and they will ensure that the ERBS will have all properties from Theorem 2.1.*

**Remark 4.** *From Theorem 2.1, it is clear that $B_k(t)$, under the three additional restrictions, is $C^\infty$-smooth at the three knots $t_{k-1}$, $t_k$ and $t_{k+1}$ without being an analytic function in these knots. Since all derivatives of $B_k$ at any of these knots are zero (Property* **P6***), the Taylor series around these knots will converges to the constant $0$ at $t_{k-1}$ and $t_{k+1}$, and the constant $1$ at $t_k$ (which obviously diverges from $B_k(t)$ when $t \in (t_{k-1}, t_{k+1})$. As for the open intervals $(t_{k-1}, t_k)$ and $(t_k, t_{k+1})$, we can see that $B_k$ is analytic there, at least when the fraction in the exponent in (2.4) is a true rational polynomial function (see the previous remark). For all other cases inside the restrictions of the intrinsic parameters of ERBS , a separate investigation is needed to study the analyticity in the open intervals between the knots. This will not be done here.*

Here we provide a brief summary of consequences from the properties.

- Property **P1** shows that the local support is two knot interval, the same as linear B-splines.
- Property **P2** implies that Expo-Rational B-splines are invariant under affine maps, which also is the case for B-splines/Bernstein polynomials (explained in [23]).
- Properties **P1** and **P2** imply that the ERBS set forms not only an affine combination, but a convex combination (that is, forms a positive partition of unity).
- Property **P3** implies strong interpolation. It is interpolation of the Lagrange type.
- Properties **P3**, **P4** and **P5** together imply that the basis functions are $C^\infty$ on $\mathbb{R}$ when the knots are simple.
- Property **P6** extends the interpolation property to Hermite interpolation.

*Proof.* The proof (of Theorem 2.1) is first dealing with the three first properties **P1**, **P2** and **P3** in a numbered list.

1. Property **P1** follows directly from definition 2.2. Since $e^{-x} > 0$ for all $x$, it is certain that $\psi_{k-1}(t) > 0$ when $t \in (t_{k-1}, t_k)$ and $\psi_k(t) > 0$ when $t \in (t_k, t_{k+1})$. The fact that $B_k(t)$ is an integral of a strictly positive function when $t \in (t_{k-1}, t_k)$, and also when $t \in (t_k, t_{k+1})$ because we then integrate from $t$ to $t_{k+1}$, means that $B_k(t)$ is strictly positive on the open interval $(t_{k-1}, t_{k+1})$. It also follows that $B_k(t)$ is strictly increasing on $(t_{k-1}, t_k]$, and strictly decreasing on $(t_k, t_{k+1})$.

2. There are only two ERBS basis functions different from zero on a knot interval $(t_k, t_{k+1}]$, so proving the last part of property **P2** also proves the first part. Suppose that $t_k < t_{k+1}$. On the knot interval $(t_k, t_{k+1}]$ only $B_k(t)$ and $B_{k+1}(t)$ are different from zero (equation (2.5)). So we have

$$B_k(t) + B_{k+1}(t) = S_k \int_t^{t_{k+1}} \psi_k(s)ds + S_k \int_{t_k}^{t} \psi_k(s)ds = S_k \int_{t_k}^{t_{k+1}} \psi_k(s)ds, \quad t \in (t_k, t_{k+1}].$$

From definition 2.3 we know that $S_k$ is the inverse of the integral and therefore

$$B_k(t) + B_{k+1}(t) = 1, \quad \text{if} \quad t \in (t_k, t_{k+1}].$$

3. Both parts of property **P3** follow directly from the fact that the numerator and the denominator, in both parts of (2.3) become equal when $t = t_k$.

We organize the proof of properties **P4**, **P5** and **P6** by giving a list of items investigating the properties of $D^j B_k(t)$ and $D^j B_{k+1}(t)$ for $j = 0, 1, 2, ...$ on the knot interval, $[t_k, t_{k+1}]$, where $t_k < t_{k+1}$. Afterwards, we will connect this list of items to the properties **P4**, **P5** and **P6**.

Several places below we have to use a well known fact (proved among others in [47]),

$$\lim_{x \to 0+} \frac{e^{-\frac{a}{x}}}{bx^n} = 0, \quad \text{for any } n, \text{ and any } a > 0 \text{ and } b \neq 0, \tag{2.13}$$

i.e. $e^{-\frac{a}{x}}$ tends to zero "faster" than any polynomial of $x$ (in the denominator) when $x$ tends to zero. Notice the range of constants $a$ and $b$, which will be used below.

(a) $\lim_{t \to t_{k+1}-} B_k(t) = 0$. This follows directly from definition 2.2.

(b) $\lim_{t \to t_k+} B_{k+1}(t) = 0$. This also follows directly from definition 2.2.

(c) $DB_{k+1}(t_{k+1}) = 0$. The proof follows.

We start by investigating $\psi_k(t_{k+1})$. We shall see that under the additional restriction 3 in Theorem 2.1, we will have $\psi_k(t_{k+1}) = 0$. Here we look at the right hand side of the knot interval.

If $\lambda_k < 1$, it is obvious that $\psi_k(t_{k+1}) = 0$, because the exponent in (2.4) (named $\overline{\zeta}(t)$) goes towards $-\infty$ when $t \to t_{k+1}-$. If $\lambda_k = 1$, the exponent is,

$$\overline{\zeta}(t) = -\beta_k \frac{|t_{k+1} - t|^{2\sigma_k}}{(t - t_k)^{\alpha_k} (t_{k+1} - t)^{\gamma_k \alpha_k}}.$$

It then follows that

$$\lim_{t \to t_{k+1}-} \overline{\zeta}(t) = \begin{cases} 0, & \text{if} \quad 2\sigma_k > \gamma_k \alpha_k, \\ \frac{-\beta_k}{(t_{k+1} - t_k)^{\alpha_k}}, & \text{if} \quad 2\sigma_k = \gamma_{k-1} \alpha_k, \\ -\infty, & \text{if} \quad 2\sigma_k < \gamma_k \alpha_k. \end{cases}$$

Therefore, we might use (additional restriction 3) either the constraint

$$2\sigma_k < \gamma_k \alpha_k \quad \text{if} \quad \lambda_k = 1, \tag{2.14}$$

or the constraint

$$\lambda_k < 1, \tag{2.15}$$

to secure that $\lim_{t \to t_{k+1}-} \overline{\zeta}(t) = -\infty$ and thus $DB_{k+1}(t_{k+1}) = \psi_k(t_{k+1}) = 0$.

(d) $\lim_{t \to t_k+} DB_k(t) = 0$. The proof follows.

We shall see that under the additional restriction 2 in Theorem 2.1, we will have $\Psi_k(t_k) = 0$. This question is analogous to the proof of the previous item, only that this time we look at the left hand side of the knot interval.

If $\lambda_k > 0$, it is obvious that $\Psi_k(t_k) = 0$, because the exponent in (2.4) (named $\widehat{\zeta}(t)$) goes toward $-\infty$ when $t \to t_k+$. If $\lambda_k = 0$, the exponent is,

$$\widehat{\zeta}(t) = -\beta_k \frac{|t - t_k|^{2\sigma_k}}{(t - t_k)^{\alpha_k} (t_{k+1} - t)^{\gamma_k \alpha_k}}.$$

It thus follows that

$$\lim_{t \to t_k+} \widehat{\zeta}(t) = \begin{cases} 0, & \text{if} \quad 2\sigma_k > \alpha_k, \\ \frac{-\beta_k}{(t_{k+1} - t_k)^{\gamma_k \alpha_k}}, & \text{if} \quad 2\sigma_k = \alpha_k, \\ -\infty, & \text{if} \quad 2\sigma_k < \alpha_k. \end{cases}$$

Therefore, we might use (additional restriction 2) either the constraint

$$\alpha_k > 2\sigma_k \quad \text{if} \quad \lambda_k = 0,$$

or the constraint

$$\lambda_k > 0$$

to secure that $\lim_{t \to t_k+} \widehat{\zeta}(t) = -\infty$ and thus $\lim_{t \to t_k+} DB_k(t) = \psi_k(t_k) = 0$.

(e) $\lim_{t \to t_k+} DB_{k+1}(t) = 0$. The proof is the same as for (d) because $\lim_{t \to t_k+} DB_{k+1}(t) = \psi_k(t_k)$.

(f) $\lim_{t \to t_{k+1}-} DB_k(t) = 0$. The proof is the same as for (c) because $\lim_{t \to t_{k+1}-} DB_k(t) = \psi_k(t_{k+1})$.

(g) $D^j B_{k+1}(t_{k+1}) = 0 \;\; for \;\; j = 2, 3, \ldots$ . The proof follows.

We have to show that $\lim_{t \to t_{k+1}-} f_{j,k}(t)\ \psi_k(t) = 0$. The question is, therefore, whether $\psi_k(t)$ goes faster towards 0 than the denominator in $f_{j,k}(t)$ for $j = 2, 3, \ldots$ when $t \to t_{k+1}-$. Here we look at the right hand side of the knot interval.

In the following, we shall see that this will only be the case under the additional restriction 3 on the intrinsic parameters (in Theorem 2.1). Suppose first that $\lambda_k < 1$,

the limit when $t \to t_{k+1}-$ of the exponent in (2.4) (named $\zeta(t)$) can in this case be reformulated to

$$\lim_{t\to t_{k+1}-} \zeta(t) = \lim_{t\to t_{k+1}-} \left[ -\frac{a}{(t_{k+1}-t)^{\gamma_k\alpha_k}} \right], \tag{2.16}$$

where

$$a = \beta\,|1-\lambda_k|^{2\sigma_k}\,(t_{k+1}-t_k)^{2\sigma_k-\alpha_k}.$$

We can easily see that $a > 0$ because $\beta > 0$, $t_k < t_{k+1}$ and $\lambda_k < 1$.

Studying (2.9), (2.10) and (2.11), we observe that the contributing factors in the denominator are $(t-t_k)$ and $(t_{k+1}-t)$. When $t \to t_{k+1}-$, all factors (both in the numerator and the denominator) except $(t_{k+1}-t)$ will tend to a real number different from zero. It follows that for all $j > 1$ then,

$$\lim_{t\to t_{k+1}-} f_{j,k}(t) = \lim_{t\to t_{k+1}-} \left[ \frac{1}{b\,(t_{k+1}-t)^n} \right], \tag{2.17}$$

where $b \neq 0$ and $n > 0$.

If we combine (2.16) with (2.17) we get

$$\lim_{t\to t_{k+1}-} f_{j,k}(t)\psi_k(t) = \lim_{t\to t_{k+1}-} \frac{e^{-\frac{a}{(t_{k+1}-t)^{\gamma_k\alpha_k}}}}{b\,(t_{k+1}-t)^n}.$$

where $\gamma_k\alpha_k > 0$ (initial constraints). This expression is of the same type as (2.13) and, therefore, tends to zero when $t \to t_{k+1}-$.

If alternatively $\lambda_k = 1$ and $2\sigma_k < \gamma_k\alpha_k$, the limit when $t \to t_{k+1}-$ of the exponent in (2.4) can in this case be reformulated to

$$\lim_{t\to t_{k+1}-} \zeta(t) = \lim_{t\to t_{k+1}-} \left[ -\frac{a}{(t_{k+1}-t)^{\gamma_k\alpha_k-2\sigma_k}} \right], \tag{2.18}$$

where

$$a = \beta(t_{k+1}-t_k)^{-\alpha_k}.$$

We can easily see that $a > 0$ because $\beta > 0$ and $t_k < t_{k+1}$.

If we now combine (2.18) with (2.17) we get

$$\lim_{t\to t_{k+1}-} f_{j,k}(t)\psi_k(t) = \lim_{t\to t_{k+1}-} \frac{e^{-\frac{a}{(t_{k+1}-t)^{\gamma_k\alpha_k-2\sigma_k}}}}{b\,(t_{k+1}-t)^n}.$$

where $\gamma_k\alpha_k - 2\sigma_k > 0$. This is of course the same as $2\sigma_k < \gamma_k\alpha_k$ (additional restriction 3). This expression is also of the same type as (2.13) and therefor tends to zero when $t \to t_{k+1}-$.

This shows that if we use the additional restriction 3 in Theorem 2.1, then
$\lim_{t\to t_{k+1}-} D^j B_{k+1}(t) = \lim_{t\to t_{k+1}-} f_{j,k}(t)\psi_k(t) = 0$, for $j = 2,3,....$

(h) $\lim_{t\to t_k+} D^j B_k(t) = 0 \;\; for \;\; j = 2,3,\ldots$ . The proof follows.

We have to show that $\lim_{t\to t_k+} f_{j,k}(t)\psi_k(t) = 0$. The question is if $\psi_k(t)$ goes "faster" towards 0 than the denominator in $f_{j,k}(t)$, for $j = 2,3,\ldots$ when $t \to t_k+$. This question is analogous to the proof of the previous item, only that this time we look at the left hand side of the knot interval.

In the following, we shall see that this will only be the case under the additional restriction 2 on the intrinsic parameters (in Theorem 2.1). Suppose first that $\lambda_k > 0$, the limit when $t \to t_k+$ of the exponent in (2.4) (named $\zeta(t)$) can be reformulated to

$$\lim_{t\to t_k+} \zeta(t) = \lim_{t\to t_k+} \left[ -\frac{a}{(t-t_k)^{\alpha_k}} \right], \tag{2.19}$$

where

$$a = \beta_k \lambda_k^{2\sigma_k} (t_{k+1} - t_k)^{2\sigma_k - \gamma_k \alpha_k}.$$

We can easily see that $a > 0$ because $\beta > 0$, $t_k < t_{k+1}$ and $\lambda_k > 0$.

Studying (2.9), (2.10) and (2.11), we can clearly see that the contributing factors in the denominator are $(t - t_k)$ and $(t_{k+1} - t)$. When $t \to t_k+$, all factors (both in the numerator and the denominator) except $(t - t_k)$ will tend to a real number different from zero. It follows that for all $j > 1$, the limit when $t \to t_k+$, of $f_{j,k}(t)$ can be reformulated to,

$$\lim_{t\to t_k+} f_{j,k}(t) = \lim_{t\to t_k+} \left[ \frac{1}{b\,(t-t_k)^n} \right], \tag{2.20}$$

where $b \neq 0$ and $n > 0$.

If we combine (2.19) with (2.20) we get

$$\lim_{t\to t_k+} f_{j,k}(t)\psi_k(t) = \lim_{t\to t_k+} \frac{e^{-\frac{a}{(t-t_k)^{\alpha_k}}}}{b\,(t-t_k)^n}.$$

where $\alpha_k > 0$ (initial constraint). The expression is of the same type as (2.13), and therefore tends to zero when $t \to t_k+$.

If alternatively $\lambda_k = 1$ and $2\sigma_k < \gamma_k \alpha_k$, the limit when $t \to t_k+$ of the exponent in (2.4) (named $\zeta(t)$) can be reformulated to,

$$\lim_{t\to t_k+} \zeta(t) = \lim_{t\to t_k+} \left[ -\frac{a}{(t_k - t)^{\gamma_k \alpha_k - 2\sigma_k}} \right], \tag{2.21}$$

where

$$a = \beta (t_{k+1} - t_k)^{-\alpha_k}.$$

We can easily see that $a > 0$ because $\beta > 0$ and $t_k < t_{k+1}$.

If we now combine (2.21) with (2.20) we get

$$\lim_{t\to t_k+} f_{j,k}(t)\psi_k(t) = \lim_{t\to t_k+} \frac{e^{-\frac{a}{(t_k-t)^{\alpha_k - 2\sigma_k}}}}{b\,(t_k-t)^n}.$$

where $\alpha_k - 2\sigma_k > 0$ is a requirement. This is of course the same as $2\sigma_k < \alpha_k$ (additional restriction 2). The expression is also of the same type as (2.13) and, therefore, tends to zero when $t \to t_k+$.

This shows that if we use the additional restriction 2 in Theorem 2.1, then $\lim_{t\to t_k+} D^j B_k(t) = \lim_{t\to t_k+} f_{j,k}(t)\psi_k(t) = 0, \quad \text{for } j = 2,3,....$

(i) $\lim_{t\to t_k+} D^j B_{k+1}(t) = 0 \ \ for \ \ j = 2,3,....$

The proof is the same as for (h) because $\lim_{t\to t_k+} D^j B_{k+1}(t) = \lim_{t\to t_k+} f_{j,k}(t)\psi_k(t), \ \ \text{for } \ j = 2,3,....$

(j) $\lim_{t\to t_{k+1}-} D^j B_k(t) = 0 \ \ for \ \ j = 2,3,... \ .$

The proof is the same as for (g) because $\lim_{t\to t_{k+1}-} D^j B_k(t) = \lim_{t\to t_{k+1}-} f_{j,k}(t)\psi_k(t) \ \ for \ \ j = 2,3,....$

(k) $D^j B_k(t) \ \ for \ \ j = 2,3,...$ is in $C^\infty(t_k, t_{k+1})$.

The proof is discussing two problems, I: The numerator of $f_{j,k}(t)$ and the asymptotic problem it can cause inside the open segments $(t_k, t_{k+1})$. II: The numerator of $f_{j,k}(t)$ and the sign problem (because of the absolute value) and the discontinuity it can cause inside the open segments $(t_k, t_{k+1})$.

I The general idea of how to avoid the numerator creating asymptotes, is that the derivation of a polynomial reduces the degree of the polynomial with 1. If the initial degree is in $\mathbb{N}$, the polynomial will terminate as a zero after a certain number of differentiations. If the degree is not in $\mathbb{N}$, it will, after a certain number of differentiations, become negative and then create an asymptote. Thus from (2.8) it follows that

$$c_k(\widehat{t}) = 0, \quad \text{if} \quad \widehat{t} = (1-\lambda_k)\, t_k + \lambda_k\, t_{k+1}.$$

This value, $\widehat{t}$, is in the closed interval $[t_k, t_{k+1}]$, and if $0 < \lambda_k < 1$, it is clearly inside the open interval $(t_k, t_{k+1})$. Notice that $\widehat{t}$ is the position for a possible asymptote.

Inspecting (2.12) (remembering that $S_k$ is a constant) we see that for the five examples considered, $f_{j,k}(t)$, for all $j > 1$, will be a sum of polynomials where the most "critical" element consists of a $(j-2)$-nd derivative of $g_k(t)$. This also follows from the recursive definition $f_{j+1,k}(t) = f'_{j,k}(t) + f_{j,k}(t)g_k(t)$, defined in (2.11), because the derivative of the previous $f_{j-1,k}$ is one of the elements in the next $f_{j,k}$.

Studying $g_k(t)$, see (2.9), we can observe that it consists of a sum of two rational polynomials. Skipping sign and constant factors, we can see that the numerator of the rational function on the left hand side can be written $|c_k(t)|^{2\sigma_k}$, and on the right hand side $|c_k(t)|^{2\sigma_k - 1}$. If $\sigma = \frac{n}{2}$, $n \in \mathbb{N}$, then the element on the right will first terminate at given points, otherwise it will create an asymptote.

II Inspecting $g_k(t)$, and its two parts, we can see that the numerator on left hand side is always positive, but the element on the right hand side has sign depending on $t$. If we skip the constant factors (they are clearly not zero) the numerator on the right hand side will be

$$sign\,(c_k(t))\ |c_k(t)|^{2\sigma_k-1}\,.$$

Recall that when computing a derivative of an absolute value of the polygon, a $sign\,(c_k(t))$ is generated every time. The first derivative of this will cancel the sign which is already there, the next will introduce it again, and so on. This shows us that only the odd number of derivatives will be without $sign\,(c_k(t))$. It follows that, when the exponent is zero, then $|c_k(t)|^0 = 1$. If $sign\,(c_k(t))$ is present on this level (when the exponent is zero), there will be a discontinuity at $t = (1-\lambda_k)\,t_k + \lambda_k\,t_{k+1}$, because the sign then change, but the value is not zero. To avoid this, $\sigma = \frac{1}{2}, \frac{3}{2}, \frac{5}{2}, ...$ can not be used.

The conclusion is as follows:
to guarantee $D^jB_k(t)$, for $j = 2,3,...$, is in $C^\infty(t_k,t_{k+1})$ the additional restriction 1 in Theorem 2.1 must be fulfilled (in addition to restriction 2 and 3 shown by the other items).

(l) $D^jB_{k+1}(t) \quad for j = 2,3,...$ is in $C^\infty(t_k,t_{k+1})$. The proof is the same as for (k).

Summing up:

If the restrictions 1, 2 and 3 in Theorem 2.1 are all fulfilled, then **P4** is a property of ERBS because of:

item (d), item (c), item (h), item (g), item (k), item (l), and property **P3**.

———————————-

If the restrictions 2 and 3 in Theorem 2.1 are both fulfilled, then **P5** is a property of ERBS because of:

item (a), item (b), item (e), item (f), item (i) and item (j).

———————————-

If the restriction 3 in Theorem 2.1 is fulfilled, then **P6** is a property of ERBS because of:

item (c), item (d), item (g) and item (h).

That $D^jB_k(t)$ is zero at every knot follows from **P5** and **P6**.

———————————-

This completes the proof of Theorem 2.1. □

## 2.4 The Scalable subset

If we add one more new restriction on the constraints on the intrinsic parameters, there appears one new important specific property connected to $\psi(t)$ defined in definition 2.2, expression (2.4).

**Theorem 2.2.** *Let the intrinsic values be restricted according to definition 2.2, and let an additional restriction be* $2\sigma_k = (1+\gamma_k)\alpha_k$*, then ,generally,*

$$\int_{t_k}^{t} \psi_k(s)ds = (t_{k+1}-t_k)\int_0^{\frac{t-t_k}{t_{k+1}-t_k}} e^{-\beta_k \frac{|s-\lambda_k|^{2\sigma_k}}{\left(s(1-s)^{\gamma_k}\right)^{\alpha_k}}}\, ds, \quad \textit{if} \;\; t_k < t \le t_{k+1}. \tag{2.22}$$

*and, particularly, when* $t = t_{k+1}$*, it has the simpler form*

$$\int_{t_k}^{t_{k+1}} \psi_k(t)dt = (t_{k+1}-t_k)\int_0^{1} e^{-\beta_k \frac{|t-\lambda_k|^{2\sigma_k}}{\left(t(1-t)^{\gamma_k}\right)^{\alpha_k}}}\, dt. \tag{2.23}$$

*Proof.* Relation (2.23) follows directly from (2.22). To prove (2.22) it is only nessesary to prove that the exponent from (2.4),

$$-\beta_k \frac{|t-((1-\lambda_k)t_k+\lambda_k t_{k+1})|^{2\sigma_k}}{\left((t-t_k)\,(t_{k+1}-t)^{\gamma_k}\right)^{\alpha_k}}, \tag{2.24}$$

is preserved during translation and scaling of the domain and thus $t, t_k, t_{k+1}$. Translation with $-t_k$ changes (2.24) to

$$-\beta_k \frac{|\bar{t}-\lambda_k(t_{k+1}-t_k)|^{2\sigma_k}}{\left(\bar{t}\,(t_{k+1}-t_k-\bar{t})^{\gamma_k}\right)^{\alpha_k}}, \tag{2.25}$$

where $\bar{t} = t - t_k$, and where the value of the expression remains unchanged. If we multiply both the numerator and the denominator in the fraction with the same constant, $(\frac{1}{t_{k+1}-t_k})^{(1+\gamma_k)\alpha_k}$, (2.25) remains unchanged, and we get,

$$-\beta_k \frac{(\frac{1}{t_{k+1}-t_k})^{(1+\gamma_k)\alpha_k}\,|\bar{t}-\lambda_k(t_{k+1}-t_k)|^{2\sigma_k}}{(\frac{1}{t_{k+1}-t_k})^{(1+\gamma_k)\alpha_k}\left(\bar{t}\,(t_{k+1}-t_k-\bar{t})^{\gamma_k}\right)^{\alpha_k}}. \tag{2.26}$$

It follows that if $2\sigma_k = (1+\gamma_k)\,\alpha_k$ then (2.26) is the same as

$$-\beta_k \frac{\left|\widehat{t}-\lambda_k\right|^{2\sigma_k}}{\left(\widehat{t}\left(1-\widehat{t}\right)^{\gamma_k}\right)^{\alpha_k}},$$

where $\widehat{t} = \frac{t-t_k}{t_{k+1}-t_k}$, which completes the proof. □

The property in Theorem 2.2 leads us to a new definition. Assume the additional restriction in Theorem 2.2. If we reduce the number of intrinsic parameters from 5 to 4, we can define a subset of Expo-Rational B-splines where the integrals are independent of the knot vector itself, depending only on the intrinsic parameters in the knot-interval. Recall from Theorem 2.2 that both the "general" integral (2.22) and the "particular" integral (2.23) is scaled by the constant $(t_{k+1}-t_k)$. In the definition below this scaling is initially in both the numerator $\phi$ and the denominator of $S$ and is therefore canceled.

**Definition 2.4.** *The scalable subset $\mathfrak{B}$ of Expo-Rational B-splines contains elements defined by $B_k(t) = B_k(\alpha_{k-1},\beta_{k-1},\gamma_{k-1},\lambda_{k-1},\alpha_k,\beta_k,\gamma_k,\lambda_k;t)$, as follows,*

$$B_k(t) = \begin{cases} S(\alpha_{k-1},\beta_{k-1},\gamma_{k-1},\lambda_{k-1}) \int_0^{w_{k-1}(t)} \phi(\alpha_{k-1},\beta_{k-1},\gamma_{k-1},\lambda_{k-1};s)ds, & \text{if } t_{k-1} < t \le t_k, \\ S(\alpha_k,\beta_k,\gamma_k,\lambda_k) \int_{w_k(t)}^{1} \phi(\alpha_k,\beta_k,\gamma_k,\lambda_k;s)ds, & \text{if } t_k < t < t_{k+1}, \\ 0, & \text{otherwise} \end{cases} \tag{2.27}$$

*where*

$$w_k(t) = \frac{t-t_k}{t_{k+1}-t_k}, \tag{2.28}$$

*and*

$$\phi(\alpha,\beta,\gamma,\lambda;t) = e^{-\beta\frac{|t-\lambda|^{(1+\gamma)\alpha}}{(t(1-t)^\gamma)^\alpha}}, \tag{2.29}$$

*and the scaling factor is*

$$S(\alpha,\beta,\gamma,\lambda) = \frac{1}{\int_0^1 \phi(\alpha,\beta,\gamma,\lambda;t)dt}, \tag{2.30}$$

*where*

$$\alpha > 0, \quad \beta > 0, \quad \gamma > 0, \quad 0 \le \lambda \le 1.$$

In the following, $S_k := S(\alpha_k,\beta_k,\gamma_k,\lambda_k)$. This notation using an index k will be also applied for other dependencies on the k-th set of intrinsic parameter. It follows that the first derivative for the scalable subset is

$$DB_k(t) = \begin{cases} \frac{S_{k-1}}{t_k-t_{k-1}}\ \phi_{k-1}\circ w_{k-1}(t), & \text{if } t_{k-1} < t \le t_k, \\ -\frac{S_k}{t_{k+1}-t_k}\ \phi_k \circ w_k(t), & \text{if } t_k < t < t_{k+1}, \\ 0, & \text{otherwise.} \end{cases} \tag{2.31}$$

For all derivatives of order $j > 0$, we get the following general equations,

$$D^j B_k(t) = \begin{cases} \left(\frac{1}{t_k-t_{k-1}}\right)^j f_{j,k-1}(t)\ \phi_{k-1}\circ w_{k-1}(t), & \text{if } t_{k-1} < t \le t_k, \\ -\left(\frac{1}{t_{k+1}-t_k}\right)^j f_{j,k}(t)\ \phi_k \circ w_k(t), & \text{if } t_k < t < t_{k+1}, \\ 0, & \text{otherwise,} \end{cases} \tag{2.32}$$

where $f_{1,k}(t) = S_k$, and where $f_{j,k}(t)$, for $j > 1$, is a rational (or more general) function.

To fulfill all the properties in Theorem 2.1 for the scalable subset (and thus also that (2.31) and (2.32) is in $C^\infty(t_{k-1},t_{k+1})$), we have to look at the additional restrictions on

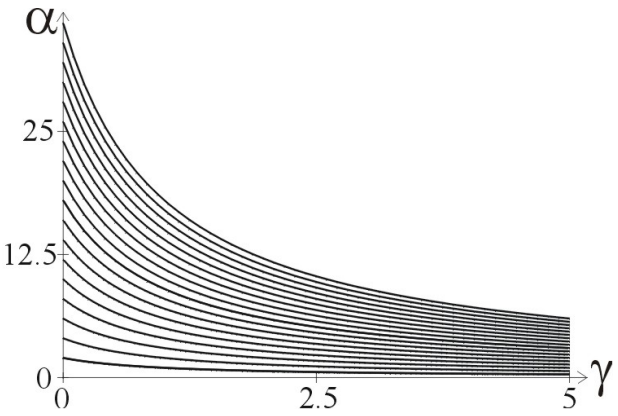


***Figure 2.2:*** *The graph shows the equation* $(1+\gamma)\alpha = 2n$, *for* $n = \{1,2,...,20\}$. $B_k(t)$ *is* $C^\infty(t_{k-1},t_{k+1})$ *when using the intrinsic parameters* $(\gamma,\alpha)$ *describing the curves in the plot, if* $0 < \lambda < 1$.

the constraints on the intrinsic parameters. If, according to the restriction on the intrinsic parameters in Theorem 2.2, we replace $\sigma$ with $\frac{(1+\gamma)\alpha}{2}$, the three additional constraints in Theorem 2.1 will also have to be changed. It follows that if we do this replacement, the first additional restriction in Theorem 2.1 will be,

$$\frac{(1+\gamma)\alpha}{2} > 0 \quad \text{if} \quad \gamma > 0 \quad \text{and} \quad \alpha > 0.$$

In Figure 2.2 there are 20 graphs of the equation $(1+\gamma)\alpha = 2n$, for $n = 1,2,...,20$. What we can see is the $\alpha$ and $\gamma$ values for which $B_k(t)$ is $C^\infty$-smooth on $\mathbb{R}$. For the second additional restriction, when we plug $\sigma = \frac{(1+\gamma)\alpha}{2}$, we get that

$$\gamma < 0 \quad \text{if} \quad \lambda = 0.$$

This is a contradiction to the initial constraint, $\gamma > 0$. For the same value of $\sigma$ we get for the third additional restriction

$$1 < 0 \quad \text{if} \quad \lambda = 1.$$

This is of course a contradiction. To fulfill all the properties in Theorem 2.1 we, therefore, get the following set of restrictions on the intrinsic parameters replacing the three items in Theorem 2.1 for the scalable subset.

1. $\frac{(1+\gamma)\alpha}{2} \in \mathbb{N}$, for property 4 in Theorem 2.1,
2. $\lambda > 0$, for property 4 and 5 in Theorem 2.1,
3. $\lambda < 1$, for property 4, 5 and 6 in Theorem 2.1,

Now follows a closer investigation of $f_{j,k}(t)$, for $j = 2,3$. Item 1 above will be investigated further studying also discontinuities and asymptotes, while item 2 and 3 will be fulfilled ($0 < \lambda < 1$). This will give us some practical experience to be used later in implementation.

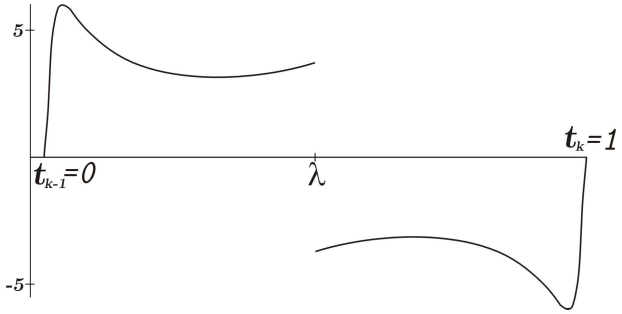


***Figure 2.3:*** *The second derivative $D^2B_k(t)$ where the intrinsic parameters are $\alpha = 0.5$, $\beta = 1$, $\gamma = 1$, $\lambda = 0.5$. Notice the discontinuity at $t = \lambda$.*

We first consider the exponent in (2.29)

$$\zeta(\alpha,\beta,\gamma,\lambda;t) = -\beta\frac{|t-\lambda|^{(1+\gamma)\alpha}}{\left(t\,(1-t)^{\gamma}\right)^{\alpha}}, \tag{2.33}$$

so that (2.29) can be rewritten as

$$\phi(\alpha,\beta,\gamma,\lambda;t) = e^{\zeta(\alpha,\beta,\gamma,\lambda;t)}.$$

The derivative of $\phi$ is thus

$$\phi_k'(t) = \zeta_k'(t)\;\phi_k(t). \tag{2.34}$$

Computing $\zeta_k'(t)$, reordering and factorizing the result, we get

$$\zeta_k'(t) = x_{2,k}(t)\;\zeta_k(t), \tag{2.35}$$

where

$$x_{2,k}(t) = \alpha_k\left(\frac{t\,(1+\gamma_k)-1}{t\,(1-t)} + \frac{\gamma_k+1}{t-\lambda_k}\right). \tag{2.36}$$

We can see that $x_{2,k}(t)$ in (2.36), and thus $\zeta_k'(t)$ in (2.35), is well defined on the two open segments $(0,\lambda_k)$ and $(\lambda_k,1)$. We can also see that $x_{2,k}(t)$ is a sum of two different parts. If we multiply the first part by $\zeta_k(t)$, it will obviously be 0 at $t = \lambda_k$. But if we multiply the second part by $\zeta_k(t)$, rearrange the result, and then study its behavior when $t \to t_k$ from both left and right, we will see that

$$\lim_{t\to\lambda_k\pm} x_{2,k}(t)\;\zeta_k(t) = \lim_{t\to\lambda_k\pm} \operatorname{sign}(\lambda_k - t)\alpha_k\beta_k\left(\frac{\gamma_k+1}{\left(t\,(1-t)^{\gamma_k}\right)^{\alpha_k}}\right)|t-\lambda_k|^{(1+\gamma_k)\alpha_k-1},$$

which, if $(1+\gamma_k)\,\alpha_k = 1$, obviously is not the same from below $(-)$ and above $(+)$, because the sign is changing at $t = \lambda_k$, but the value is obviously not 0 (illustrated in Figure 2.3). If $(1+\gamma_k)\,\alpha_k < 1$ there is an asymptotic behavior at $t = \lambda_k$ (illustrated in Figure 2.4). But if we restrict the intrinsic parameters $\alpha$ and $\gamma$ to

$$(1+\gamma)\,\alpha > 1, \tag{2.37}$$

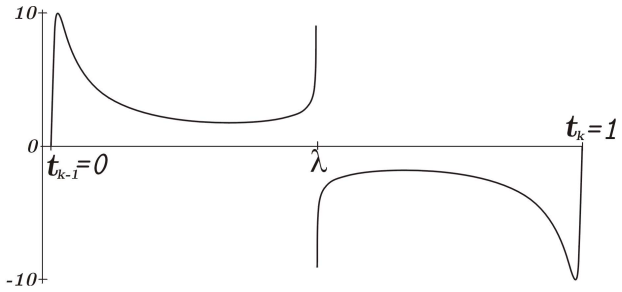


***Figure 2.4:*** *The second derivative $D^2B_k(t)$ where the intrinsic parameters are $\alpha = 0.4$, $\beta = 1$, $\gamma = 1$, $\lambda = 0.5$. Notice the asymptotic behavior at $t = \lambda$.*

we get a continuous function

$$f_{2,k}(t) = \begin{cases} 0, & \text{if } t = \lambda_k, \\ S_k\, x_{2,k}(t)\, \zeta_k(t), & \text{otherwise.} \end{cases} \tag{2.38}$$

Computing the derivative of $x_{2,k}(t)$ (2.36) we get

$$x'_{2,k}(t) = \alpha_k \left( \frac{t^2\,(1+\gamma_k) + 1 - 2t}{(t\,(1-t))^2} - \frac{\gamma_k + 1}{(t-\lambda_k)^2} \right).$$

Computing the second derivative of $\phi$, and thus the derivative of (2.34), we get

$$\begin{aligned} \phi''_k(t) &= \left(x_{2,k}(t)\, \zeta_k(t)\, \phi_k(t)\right)' \\ &= x'_{2,k}(t)\, \zeta_k(t)\, \phi_k(t) + x_{2,k}(t)\, \zeta'_k(t)\, \phi_k(t) + x_{2,k}(t)\, \zeta_k(t)\, \phi'_k(t) \\ &= \left(x'_{2,k}(t) + x_{2,k}(t)^2\,(1+\zeta_k(t))\right)\, \zeta_k(t)\, \phi_k(t). \end{aligned}$$

This can be reorganized into the following formula,

$$\phi''_k(t) = x_{3,k}\, \zeta_k(t)\, \phi_k(t),$$

where

$$\begin{aligned} x_{3,k}(t) = \alpha_k \Bigg( & \frac{t^2(\gamma_k+1) + 1 - 2t}{(t\,(t-1))^2} - \frac{\gamma_k + 1}{(t-\lambda_k)^2} + \\ & \alpha_k \left( \frac{(t(\gamma_k - \lambda_k - \lambda_k\gamma_k) + \lambda_k)^2}{(t\,(1-t)\,(t-\lambda_k))^2} \right) \left( 1 - \beta_k \frac{|t-\lambda_k|^{(1+\gamma_k)\alpha_k}}{\left(t\,(1-t)^{\gamma_k}\right)^{\alpha_k}} \right) \Bigg). \end{aligned} \tag{2.39}$$

Looking at $x_{3,k}(t)$ (2.39), one can see that it is divided into a sum of four parts. The first part will obviously be 0 at $t = \lambda_k$ when multiplied by $\zeta_k(t)$, and also the last part will be 0 at $t = \lambda_k$ if, in addition, restriction (2.37) is fulfilled. Multiplying the rest by $\zeta_k(t)$ gives

$$\left( \gamma_k + 1 - \alpha_k \left( \frac{t\,(\gamma_k - \lambda_k - \lambda_k\gamma_k) + \lambda_k}{t\,(1-t)} \right)^2 \right) \frac{\alpha_k\beta_k\,|t-\lambda_k|^{(1+\gamma_k)\alpha_k - 2}}{\left(t\,(1-t)^{\gamma_k}\right)^{\alpha_k}}, \tag{2.40}$$

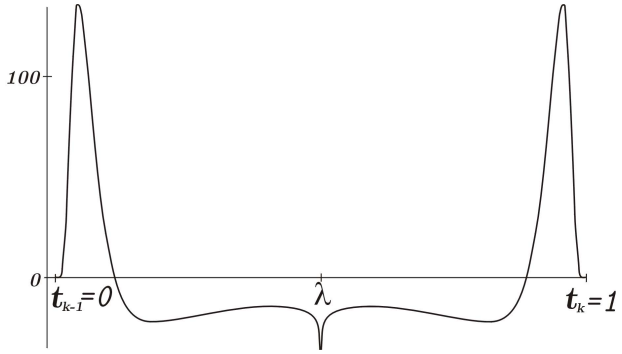


***Figure 2.5:*** *The third derivative $D^3B_k(t)$ where the intrinsic parameters are $\alpha = 0.9$, $\beta = 1$, $\gamma = 1$, $\lambda = 0.5$. At $t = \lambda$ there is a very steep asymptote (here truncated).*

which, if $(1+\gamma)\,\alpha < 2$, is obviously not defined at $t = \lambda_k$. Two figures shows examples of asymptotes. In Figure 2.5 $\alpha = 0.9$ and thus $(1+\gamma)\,\alpha = 1.8$, and in Figure 2.6 $\gamma = 0.8$ and thus also $(1+\gamma)\,\alpha = 1.8$. In both cases the asymptotic behavior is clearly illustrated. If we now use $t = \lambda_k$ in (2.40) (except for $|t-\lambda_k|$), we can see that

$$\lim_{t\to\lambda_k\pm} x_{3,k}(t)\;\zeta_k(t) = \lim_{t\to\lambda_k\pm} -\frac{\beta_k\alpha_k\,(\gamma_k+1)\,(\alpha_k\,(1+\gamma_k)-1)}{\left(\lambda_k\,(1-\lambda_k)^{\gamma_k}\right)^{\alpha_k}}\,|t-\lambda_k|^{(1+\gamma_k)\alpha_k-2}. \tag{2.41}$$

If we then restrict the intrinsic parameters $\alpha$ and $\gamma$ to

$$(1+\gamma)\,\alpha \geq 2, \tag{2.42}$$

we get a continuous function

$$f_{3,k}(t) = \begin{cases} 0, & \text{if } t=\lambda_k \text{ end } (1+\gamma_k)\,\alpha_k > 2, \\ \frac{-2\beta_k}{\lambda_k^{\alpha_k}(1-\lambda_k)^{2-\alpha_k}}, & \text{if } t=\lambda_k \text{ end } (1+\gamma_k)\,\alpha_k = 2, \\ S_k\;x_{3,k}(t)\;\zeta_k(t), & \text{otherwise.} \end{cases} \tag{2.43}$$

In section 2.9 below there is a study of the variation of intrinsic parameter values, providing examples of basis functions $B_k(t)$ where the intrinsic parameters are being modified one by one.

## 2.5 The default set of intrinsic parameters

The default values of the intrinsic parameters for the scalable subset are

$$\alpha = \beta = \gamma = 1 \quad \text{and} \quad \lambda = \frac{1}{2}.$$

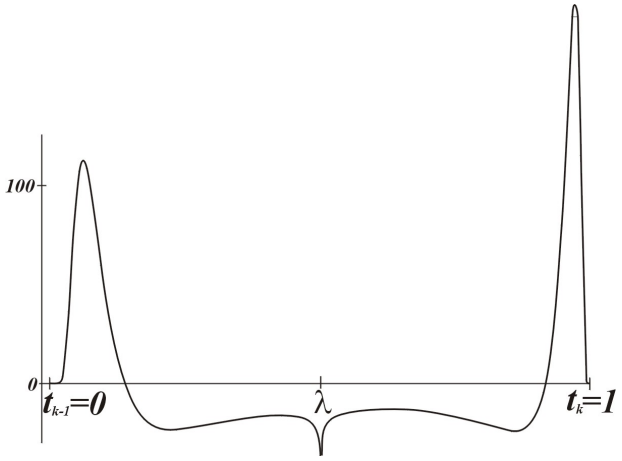


***Figure 2.6:*** *The third derivative $D^3B_k(t)$ where the intrinsic parameters are $\alpha = 1$, $\beta = 1$, $\gamma = 0.8$, $\lambda = 0.5$. At $t = \lambda$ there is a very steep asymptote (here truncated).*

This corresponds to $\sigma = 1$ in the general case. This set of values for the intrinsic parameters obviously meets all restrictions to fulfill all the basic properties including being $C^\infty$-smooyh. Using these intrinsic values, we get a simpler version of $\phi(t)$,

$$\phi(t) = e^{-\frac{\left(t-\frac{1}{2}\right)^2}{t(1-t)}}. \tag{2.44}$$

In Figure 2.1, there is a graph of $B_k(t)$ (2.27) and its first derivative (2.31), where the default values of the intrinsic parameters are used. The form is bell-shaped, and it spans over two knot intervals. The scaling of the first derivative is, as we can see in equation (2.31), dependent on the size of the knot intervals because the integral of the first derivative over the whole knot interval must be 1. In Figure 2.1 the two knot intervals have both length 1.

The scaling factor (definition 2.3) for the default set is the constant

$$S_d = 1.6571376797382103. \tag{2.45}$$

The rational function $f_{j,k}(t)$ is now independent of the knot $k$, because the intrinsic parameters are the same on all knot intervals. It will, therefore, be denoted by $f_{jd}(t)$. The

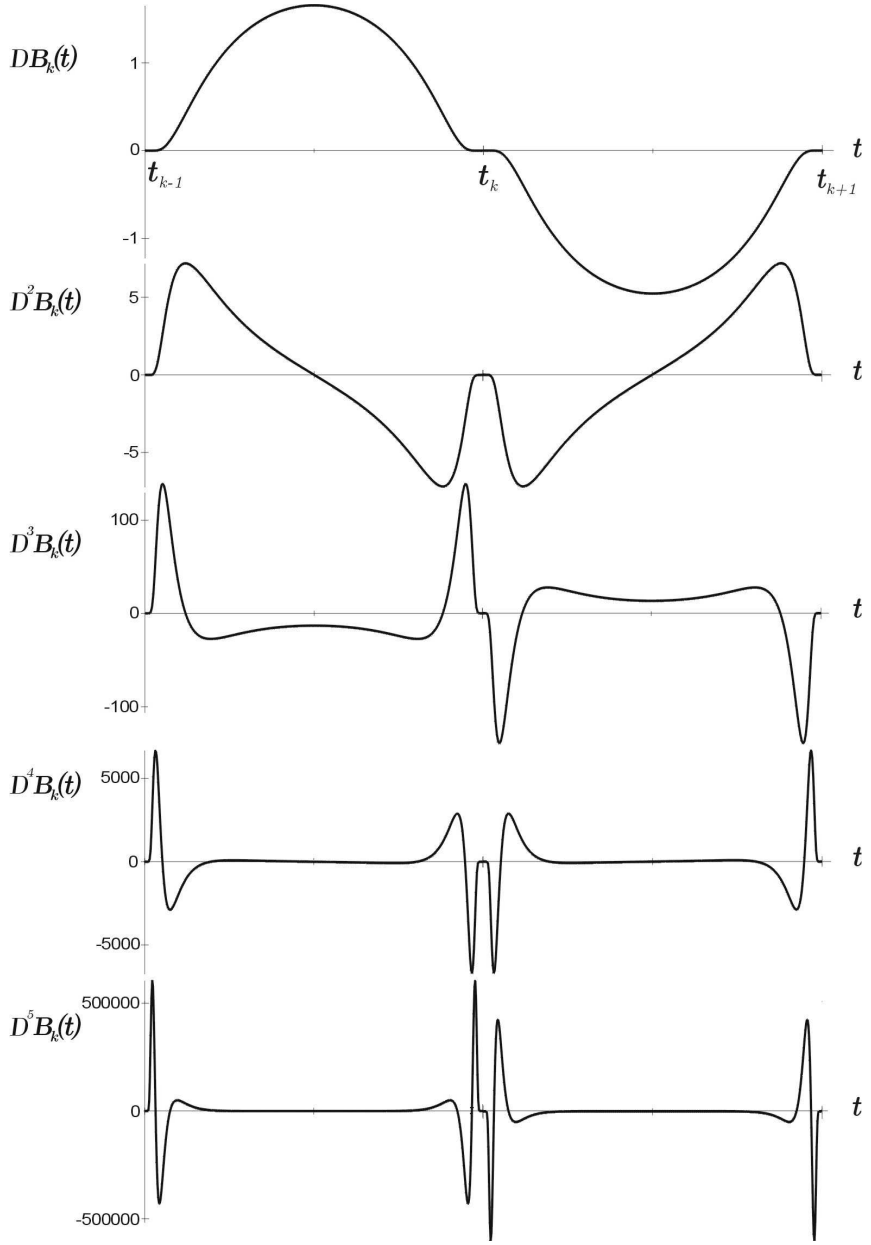


***Figure 2.7:*** *A graph of the first, second, third, fourth and fifth derivatives of $B_k(t)$ when the default values of the intrinsic parameters are used. The knots $t_{k-1}, t_k$ and $t_{k+1}$ are also given on the graph, and the function values in this example correspond to $t_{k+1} - t_k = t_k - t_{k-1} = 1$. Notice the symmetry/antisymmetry, and the fast growing absolute value of the extreme values, as well as the concentration of the rapid changes of the graph towards the knots.*

default set has the following functions $f_{jd}(t)$ for the first seven derivatives ($j = 1,2,\ldots,7$),

$$
\begin{aligned}
f_{1d}(t) &= S_d,\\
f_{2d}(t) &= S_d \tfrac{(1-2t)}{(2t(1-t))^2},\\
f_{3d}(t) &= S_d \tfrac{\frac{3}{2}(1-2t)^4-\frac{1}{2}}{(2t(1-t))^4},\\
f_{4d}(t) &= S_d \tfrac{\left(3(1-2t)^6+\frac{3}{2}(1-2t)^4-5(1-2t)^2+\frac{3}{2}\right)(1-2t)}{(2t(1-t))^6},\\
f_{5d}(t) &= S_d \tfrac{\frac{15}{2}(1-2t)^{10}+\frac{45}{4}(1-2t)^8-33(1-2t)^6+\frac{29}{2}(1-2t)^4+\frac{3}{2}(1-2t)^2-\frac{3}{4}}{(2t(1-t))^8},\\
f_{6d}(t) &= S_d \tfrac{\left(\frac{45}{2}(1-2t)^{12}+\frac{135}{2}(1-2t)^{10}-\frac{765}{4}(1-2t)^8+75(1-2t)^6+66(1-2t)^4-\frac{85}{2}(1-2t)^2+\frac{15}{4}\right)(1-2t)}{(2t(1-t))^{10}},\\
f_{7d}(t) &= S_d \tfrac{\frac{315}{4}(1-2t)^{16}+\frac{1575}{4}(-)^{14}-\frac{8505}{8}(-)^{12}+\frac{465}{4}(-)^{10}+\frac{9645}{8}(-)^8-\frac{3551}{4}(-)^6+\frac{1005}{8}(-)^4+\frac{135}{4}(-)^2-\frac{15}{8}}{(2t(1-t))^{12}}.
\end{aligned}
\tag{2.46}
$$

The expressions in (2.46) show us that $f_{jd}(t)$ is "symmetric" around $t = 0.5$ when $j$ is an odd number. This happens because the polynomial exponents in all factors are even numbers. On the other hand, $f_{jd}(t)$ is antisymmetric when $j$ is an even number. This is because there is a linear factor in the numerator, $(1-2t)$, without an exponent, and thus it is changing the sign at $t = \frac{1}{2}$ and is ensuring that the value is zero when the sign changes. All factors in the numerator are expansions in powers of $(1-2t)^2$. This coincides with the constraint $\sigma \in \mathbb{N}$ in Theorem 2.1, and fits with the proof of the same theorem, and thus guarantees that the continuity is not destroyed by a change of sign.

In Figure 2.7, the first, second, third, fourth and fifth derivatives of $B_k(t)$ are plotted using the default values of the intrinsic parameters. The graph is spanning over two knot intervals $[t_{k-1}, t_k]$ and $[t_k, t_{k+1}]$. The right hand side $[t_{k-1}, t_k]$ is mirroring the left hand side $[t_k, t_{k+1}]$ alternated around the t-axis at $(y = 0)$ and y-axis at $(t = t_k)$. The graph confirms the conclusion of Theorem 2.1, showing us that all derivatives are zero at all knots. In addition, it shows us that the extreme values are increasing very fast. For the fifth derivative, the extreme value (for this derivative it is a global maximum with positive value) is actually more than 500000. The occurrence of "roots" is: at $\frac{t_{k-1}+t_k}{2}$ for the second derivative (for the part on the left hand side), and then for increasing order of the derivatives, the positions of the roots tend to go towards the knots $t_{k-1}$ and $t_k$. For the part on the right hand side, we see the same behavior, the position of a root is starting at $\frac{t_k+t_{k+1}}{2}$, and then tends to go towards the knots $t_k$ and $t_{k+1}$ when the order of the derivatives increases.

In Figure 2.8, the numerators and the denominators in $f_{jd}(t)$, $j = 2,3,4,5$ are plotted separately. The extreme value of the denominator is a positive value rapidly decreasing towards zero, and for $j = 5$ the maximum value is $\left(\frac{1}{2}\right)^8 \approx 0.0039$. For $j \leq 5$, the numerator is bounded within $[-1,1]$. This is in general not a rule for higher derivatives, but it is a rule that it starts in 1 and ends in $-1$ for even $j$ values, and that it start in 1 and ends in 1 for odd $j$ values (provided that the denominator is expanded in the same way as in Figure 2.8 and in expression (2.46)).

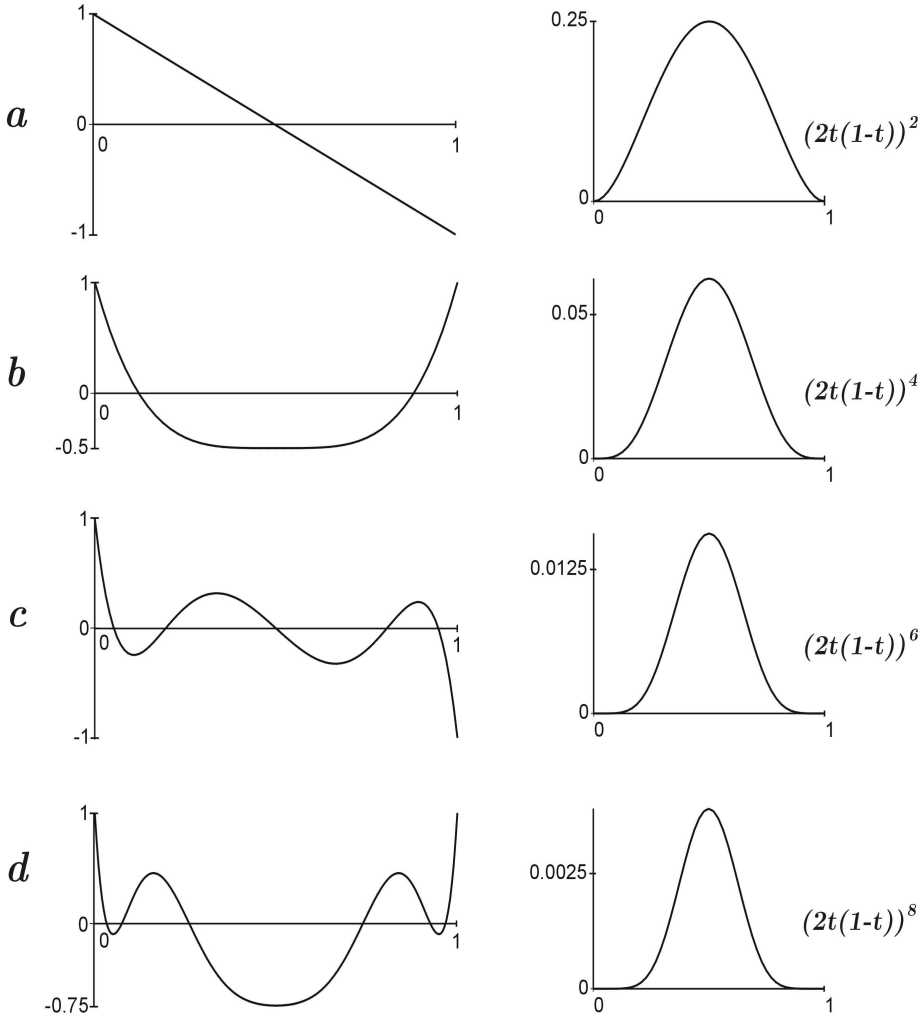


***Figure 2.8:*** *A graph of the numerator (on the left hand side) and the denominator (on the right hand side) of the fraction in $f_{j,k}(t)$, $j = 2,3,4,5$ (in the second, third, fourth and fifth derivatives of $B_k(t)$). The default set of the intrinsic parameters is used. The expressions for the four graphs on the left hand side are*

**a** $(1-2t)$,
**b** $\frac{3}{2}(1-2t)^4 - \frac{1}{2}$,
**c** $\left(3(1-2t)^6 + \frac{3}{2}(1-2t)^4 - 5(1-2t)^2 + \frac{3}{2}\right)(1-2t)$,
**d** $\frac{15}{2}(1-2t)^{10} + \frac{45}{4}(1-2t)^8 - 33(1-2t)^6 + \frac{29}{2}(1-2t)^4 + \frac{3}{2}(1-2t)^2 - \frac{3}{4}$.

*Notice that on $[0,1]$, the four numerators are bounded within $[-1,1]$, and the extreme values of the denominators are getting progresively smaller. The numerator is alternatingly symmetric and antisymmetric. The number of roots for the numerators is 1 for $j = 2$ and 2 for $j = 3$. It is then increasing by 4 from one even $j$ to the next, and from one odd $j$ to the next.*

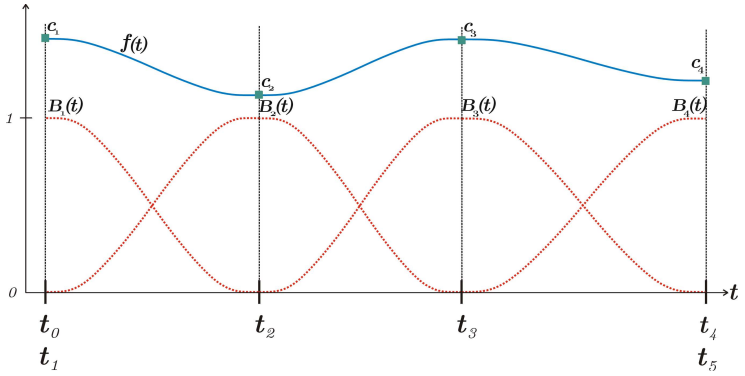


***Figure 2.9:*** *A "global" Expo-Rational B-spline function $f(t)$ (blue) with four scalar coefficients $\{c_i\}_{i=1}^{4}$ (green). The global function is a blending of the coefficients, with the Expo-Rational B-splines $\{B_i(t)\}_{i=1}^{4}$ (red) being the blending functions. The global function interpolates the coefficients, and all derivatives are zero at all knots. The knot vector $\{t_i\}_{i=0}^{5}$ is also marked, and there are multiple knots at both ends (discontinuity showed in section 2.7).*

## 2.6 Expo-Rational B-spline functions

The reason for developing the Expo-Rational B-splines is to use them as basis functions (blending functions) in a compound function, as usual in polynomial B-spline functions. In a polynomial B-spline function there is, between two knots, a polynomial of degree $d$ composed by a barycentric combination of $d+1$ coefficients, weighted by $d+1$ B-spline basis functions. In an ERBS function there are, inside one knot segment, only 2 basis functions that are different from zero. We shall consider first separately the case when all coefficients in the linear combination of the ERBS are constants (scalars, vectors or points) and we shall see that in this case some particular results are obtained. Recall from Theorem 2.1 that ERBS are interpolating all coefficients in the knots, and that all derivatives are zero in the knots. A scalar ERBS function will in each segment (between two knots) be a scaled and translated version (an affine mapping) of the ERBS basis function. It will interpolate the coefficients, and all derivatives will be zero at all knots. This behavior is illustrated in Figure 2.9, and in function (2.48) below. We, therefor, will have the following function inside a knot interval,

$$f(t) = c_i\, B_i(t) + c_{i+1}\, B_{i+1}(t), \quad \text{if} \quad t_i < t < t_{i+1}. \tag{2.47}$$

Recall the basic property 2 in Theorem 2.1. It follows that (2.47) can be reformulated to

$$f(t) = c_{i+1} + (c_i - c_{i+1})\, B_i(t), \quad \text{if} \quad t_i < t < t_{i+1}. \tag{2.48}$$

Secondly, we shall consider the general case of non-constant (scalars, vectors or points) local functions as coefficients (illustrated in Figure 2.10). The domain of these functions has to be the same as the domain of the respective ERBS basis function. It means that the

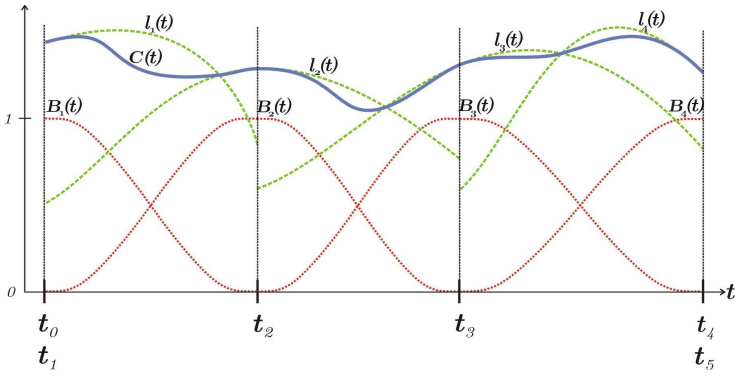


***Figure 2.10:*** *A "global" Expo-Rational B-spline function $f(t)$ (blue) with four local functions $\{l_i(t)\}_{i=1}^{4}$ (green). The global function is a blending of its local functions, with the Expo-Rational B-splines $\{B_i(t)\}_{i=1}^{4}$ (red) being the blending functions. The global function also completely interpolates all existing derivatives of each of the adjacent local functions at the respective knots. The knot vector $\{t_i\}_{i=0}^{5}$ is also marked, and there are multiple knots at both ends (discontinuity showed in section 2.7).*

local function with index $k$ is defined (and usually bounded) on the domain $(t_{k-1}, t_{k+1})$. A practical solution is to map the domains from the basis functions to the local functions (see definition 2.7 later in this section). Now follows a definition of a set of function spaces, defining functions that are proper to use as local functions because they ensure the fulfillment of the basic property 5 of the ERBS basis functions described in Theorem 2.1.

**Definition 2.5.** *To classify local functions $l_k(t)$, we define a set of function spaces,*

$$\mathfrak{F}(B_k) = \{l : D^j(l(t)B_k(t)) = 0 \quad for \quad j = 0, 1, \ldots \quad and \quad t = t_{k-1} \quad or \quad t = t_{k+1}\},$$

*where the intrinsic parameters on the segments $(t_{k-1}, t_k)$ and $(t_k, t_{k+1})$ support property 5 described in Theorem 2.1.*

This is a very weak restriction, and in normal practical use it is difficult to find a local function that is not in these sets of function spaces. Now the definition of an Expo-Rational B-spline function follows.

**Definition 2.6.** *An Expo-Rational B-spline (ERBS) function $f(t)$ (scalar or vector-valued or point-valued) is defined on the domain $(t_1, t_n]$ by*

$$f(t) = \sum_{k=1}^{n} l_k(t)B_k(t) \quad if \quad t_1 < t \leq t_n,$$

*where $l_k(t)$ are local (scalar or vector-valued) functions defined on $(t_{k-1}, t_{k+1})$, $k = 1, \ldots, n$, and $B_k(t)$, $k = 1, \ldots, n$ is defined in definition 2.2. It is assumed that $l_k(t) \in \mathfrak{F}(B_k)$.*

Just like Polynomial B-spline functions, Expo-Rational B-spline functions: have basis functions defined by a knot vector, they form a nonnegative partition of unity (which

also means that they are invariant under affine transformations), the basis functions have minimal local support (the same as 1st degree B-splines). In addition, for the typical case of simple knots $t_k < t_{k+1}$ properties 2-6, together with $l_k(t) \in \mathfrak{F}(B_k)$, imply Hermite interpolation (discussed in section 2.8). Figure 2.10 illustrates this clearly. In every knot the "global" function interpolates the local functions, not only the functional value, but also the derivatives. The figure also shows that the "global" function passes through all intersections between two neighboring local functions. This is because the "global" function is an affine combination of two and only two local functions for every value of the argument.

Using local functions offers a new possibility (or challenge, in practical implementation), namely, a domain mapping from segments in the "global" domain onto the "local" domains. It, therefore, follows that local functions must be split up into a function on the local domain and what we define to be the affine global/local mapping.

**Definition 2.7.** *We denote the local domain for the local function $l_k$ to be,*

$$I_k = (s_{k0}, s_{k1}) \subset \mathbb{R} \quad \textit{where } s_{k0} < s_{k1},$$

*and where $s_{k0}$ is the start parameter value, end $s_{k1}$ is the end parameter value of the local function $l_k$. The global/local mapping $\omega_k$ is said to be the mapping of a segment $(t_{k-1}, t_{k+1})$ in the "global" domain of the Expo-Rational B-spline function onto the domain $I_k$,*

$$\omega_k : (t_{k-1}, t_{k+1}) \subset \mathbb{R} \rightarrow I_k.$$

*$\omega_k$ is, therefore, the affine mapping*

$$\omega_k(t) = s_{k0} + \frac{t - t_{k-1}}{t_{k+1} - t_{k-1}}(s_{k1} - s_{k0}). \tag{2.49}$$

*The inverse mapping from $I_k$ to $(t_{k-1}, t_{k+1})$ is*

$$\omega_k^{-1}(s) = t_{k-1} + \frac{s - s_{k0}}{s_{k1} - s_{k0}}(t_{k+1} - t_{k-1}). \tag{2.50}$$

The possibilities in constructing a great diversity of functions depending of the variety of local functions gives ERBS a new "dimension", because by this one can customize the typical properties of the function. Some brief examples are that one can use trigonometric functions or circular arcs (in $\mathbb{R}^2$ or $\mathbb{R}^3$), Bézier functions, rational functions, special functions etc. The local function can even be an ERBS function itself, raising the possibilities for multilevel ERBS. There is nothing which says that all local functions must be of the same type, and there is a great potential of undetected possibilities and properties waiting to be descovered.

## 2.7 Knot vectors and continuity

In polynomial B-splines the knot vector is determining the parametrization and the continuity over knots by the multiplicity of the knots. The same is the case for Expo-Rational

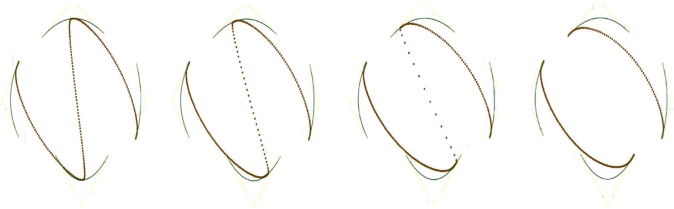

***Figure 2.11:*** *Four versions of a curve. In the left figure the knots are uniformly distributed except for the multiple knots at each end. In curve two from the left, the distance between the two middle simple knots are half of the original size. In the next curve the size is halved once more, and in the figure on the right hand side the two middle simple knots merge into one multiple knot. As a result, the latter curve is being split into two curves. The curves are drawn with points, to illustrate the speed of the parametrization (denser means slower).*

B-spline functions, but here the knot vector of ERBS is only determining if the function is $C^\infty$ or if it is discontinuous at a knot. When the knot is multiple ($t_k = t_{k+1}$), it can be seen that Definition 2.2 and Figure 2.1 shows that $B_k$ and $B_{k+1}$ have discontinuity at $t_k$. In this case, the continuity of the global B-spline function $f$ in definition 2.6 depends on the local curve, as shown in the following Theorem.

**Theorem 2.3.** *Provided that the intrinsic parameters are restricted so that all properties in Theorem 2.1 are fulfilled, an Expo-Rational B-spline function $f$ belongs to $C^\infty(t_1,t_n]$ if, for $k = 1,\ldots,n-1$, the knots is simple, i.e., $t_k < t_{k+1}$, and the local functions $l_k(t)$ are in $C^\infty(t_{k-1},t_{k+1})$ and $l_k(t) \in \mathfrak{F}(B_k)$. If $t_k = t_{k+1}$, then $f$ is only continuous if $l_k(t_k) = l_{k+1}(t_{k+1})$. Otherwise, $f$ is discontinuous at $t = t_k$.*

*Proof.* If $t_k < t_{k+1}, \quad k = 1,\ldots,n-1$, then $f$ is $C^\infty$-smooth because of properties 3,4,5 and definition 2.6. If $t_k = t_{k+1}$ then because of property 3

$$f(t_k) = l_k(t_k),$$

and

$$\lim_{t \to t_{k+1}+} f(t) = l_{k+1}(t_{k+1})$$

which shows that in the case of multiple knots, $f(t)$ is only continuous if $l_k(t_k) = l_{k+1}(t_{k+1})$. This completes the proof. □

Figure 2.11, where a curve in $\mathbb{R}^2$ is used as an example, illustrates the effect of moving two knots towards each other. One can see that the speed of the curve is increasing towards infinity until the curve suddenly (when the knots coincide) breaks into two parts.

## 2.8 Hermite Interpolation properties

Here we shall see that, under the general assumptions that the intrinsic parameters fulfill all properties in Theorem 2.1 and that for the local functions, $l_k(t) \in \mathfrak{F}(B_k)$ holds, Expo-Rational B-spline functions have an Hermite interpolation property; we also discuss how this can be used. The important underlying fact here is that

$$D^j B_k(t_k) = 0, \quad \text{if } \; k = 1,...,n \; \text{ and } \quad j = 1,2,...,$$

which is property 6 in Theorem 2.1. Let us now increase the generalization to a higher dimensional value of the function,

$$f : (t_1, t_n] \to \mathbb{R}^d, \quad \text{where } \; 1 \leq d < \infty,$$

and let us also consider the definition of the affine global/local mapping $\omega_i$ from definition 2.7, where $s_{i0}$ is the start parameter of the domain of the local function with index $i$ and $s_{i1}$ is the end parameter value. We then have the following Hermite interpolation property for an Expo-Rational B-spline function.

**Theorem 2.4.** *Let the sequense of functions* $c_i : [s_{i0}, s_{i1}] \subset \mathbb{R} \to \mathbb{R}^d$ *where* $1 \leq d < \infty$, *be* $\in \mathfrak{F}(B_k)$. *Let*

$$f(t) = \sum_{i=1}^{n} c_i \circ \omega_i(t) B_i(t) \tag{2.51}$$

*be the general "ERBS" d-dimensional vector function, defined by the knot vector* $\{t_i\}_{i=0}^{n+1}$ *and the local vector functions* $c_i(s), i = 1,...,n$. *Denote the global/local "scaling" factors by*

$$\delta_i = \frac{s_{i1} - s_{i0}}{t_{i+1} - t_{i-1}}, \quad \textit{for } \; i = 1,2,...,n.$$

*If the properties in Theorem 2.1 are fulfilled, then*

$$D^j f(t_i) = \delta_i^j \, D^j c_i \circ \omega_i(t_i), \quad \textit{for } \; j = 0,1,2,... \; \textit{ and } \; i = 1,..,n. \tag{2.52}$$

*Proof.* Deriving (2.51) according to $t$ we get

$$Df(t) = \sum_{i=1}^{n} \left( D(c_i \circ \omega_i)(t) B_i(t) + c_i \circ \omega_i(t) DB_i(t) \right).$$

Recalling that $B_i(t_i) = 1$ (property 3 in Theorem 2.1), $B_i(t_j) = 0$ when $j \neq i$ (property 1) and $DB_i(t_j) = 0$ for all $j$ (property 6), it follows that

$$\begin{aligned} Df(t_i) &= D(c_i \circ \omega_i)(t) \\ &= \delta_i \, Dc_i \circ \omega_i(t). \end{aligned}$$

Since

$$D(D^j c_i \circ \omega_i)(t) = \delta_i \, D^{j+1} c_i \circ \omega_i(t),$$

expression (2.52) follows. Which completes the proof. □

**Remark 5.** *One typical situation is when the domain of the local function is scaled to the standard "unit" domain. For example, Bézier curves have a domain where* $s_{i1} - s_{i0} = 1$. *Combined with a uniform knot vector where* $t_{i+1} - t_{i-1} = 1$ *for* $i = 1,...,n$, *the scaling factor* $\delta_i^j$ *for* $i = 1,...,n$ *and for* $j = 1,2,...$ *can be eliminated from consideration, which is very convenient practically. In all other cases it is important to remember these important scalings!*

The fact that an ERBS function completely interpolates the values and all existing derivatives of its local functions in their respective knots, raises the possibilities to approximate a function/curve/surface/ etc. by using local functions/... etc., respectively, which provides Hermite interpolates of the original function in the respective interior knot. Therefore, the Hermite interpolation property gives the following possibilities to construct an Expo-Rational B-spline function:

- Given a function $g(x)$ and a strictly increasing vector $\{x_i\}_{i=1}^n$ of parameter values, indicating the interpolation knots.
- We can now construct a knot vector $\{t_i\}_{i=0}^{n+1}$, where $t_i = x_i,\ i = 1,...,n$ and $t_0 = x_1$ and $t_{n+1} = x_n$ (if $g(t)$ is cyclic then $t_0 = x_1 - (x_n - x_{n-1})$ and $t_{n+1} = x_n + x_1 - x_0$).
- The next step is to decide the type of local functions, including the local domain, and the number of derivatives $d_i$ to use.
- Finally we can generate the local functions $c_i$ by Hermite interpolation,

$$D^j c_i \circ \omega_i(t_i) = \left(\frac{t_{i+1} - t_{i-1}}{s_{i1} - s_{i0}}\right)^j D^j g(x_i), \quad \text{for} \quad j = 0,1,...,d_i.$$

In the following, a short investigation of the convergence of an ERBS function towards an original function will be given, when the number of knots or/and the number of derivatives used in the Hermite interpolation increases. As a test function and, thus, an original function, we use

$$sin(t), \quad \text{for} \quad 0 \le t \le 2\pi,$$

and we use monomial basis functions in the Taylor series,

$$c_{j,i}(t) = f(t_i) + f'(t_i)(t - t_i) + \frac{f''(t_i)}{2!}(t - t_i)^2 + \frac{f'''(t_i)}{3!}(t - t_i)^3 + ..., \tag{2.53}$$

for $i = 1,2,...,n$, as local functions. The index $j$ is denoting the number of derivatives used, i.e. the number of terms in the equation (2.53) is $j+1$. It follows that the global/local scaling factors, connected to the respective local functions, are $\delta_i = 1$.

Table 2.1 illustrates the convergence of the approximation of the sine function by the ERBS function using Hermite interpolation. For a number $n$, denoting the number of interior knots in an ERBS function (the total number of knots is thus $n+2$), and a number $j$, denoting the number of derivatives in the Hermite interpolation used, we denote the error between the ERBS function and the sine function by

$$\varepsilon_{n,j} = \max_{t \in [0,2\pi]} |f_{n,j}(t) - sin(t)|,$$

| number | number of derivatives used | | | |
|---|---|---|---|---|
| of knots | 1 | 2 | 3 | 4 |
| 5 | $2.9e-01$ | $7.8e-02$ | $1.6e-02$ | $2.4e-03$ |
| 10 | $6.1e-02$ | $7.0e-03$ | $6.1e-04$ | $4.3e-05$ |
| 20 | $1.4e-02$ | $7.5e-04$ | $3.1e-05$ | $1.0e-06$ |
| 40 | $3.2e-03$ | $8.7e-05$ | $1.8e-06$ | $2.8e-08$ |
| 80 | $7.9e-04$ | $1.1e-05$ | $1.0e-07$ | $8.2e-10$ |
| 160 | $1.9e-04$ | $1.3e-06$ | $6.3e-09$ | $2.5e-11$ |

***Table 2.1:*** *The table shows the connection between the error (maximum deviation between the ERBS function and the sine function), the number of knots and the number of derivatives used in the Hermite interpolation.*

where

$$f_{n,j}(t) = \sum_{i=1}^{n} c_{j,i}(t) B_i(t).$$

are the ERBS functions. In Table 2.1 we can see $\varepsilon_{n,j}$, for $n = 5, 10, 20, 40, 80, 160$ and $j = 1, 2, 3, 4$.

In Figure 2.12 the "errors" shown in Table 2.1 are plotted as a surface. The vertical axis uses a logarithmic scale, and one of the horizontal axes (the one showing the number of knots) has a logarithmic scale with base 2. The surface is plotted with different colors, where each color represents one unit on the error axis. In the figure the minimal error (at $n = 160$ and $j = 4$) is $2.5e-11$. Outside the range of the figure, the surface continues in the same way, and at $n = 320$ and $j = 5$, the error is $1.5e-15$. Observing the surface, we can see that a curve along a constant number of knots is getting steeper when the number of knots increases. The same is also the case in the other direction, we can see that a curve along a constant number of derivatives is getting steeper when the number of derivatives increases (provided that the function approximated by this interpolation is sufficiently smooth).

Practical examples of how to use the Hermite interpolation properties to construct "ERBS" curves and surfaces will be given later in this report.

## 2.9 Influence of the intrinsic parameters

In this section we will see examples of the intrinsic parameters´ influence one by one, but restricted to the scalable subset $\mathfrak{B}$ (definition 2.4). The basis function $B_k(t)$ (see (2.27), (2.28), (2.29) and (2.30)) spans 3 knots, $t_{k-1}, t_k, t_{k+1}$, and thus 2 intervals, and the shape depends on the intrinsic parameters on each interval. We will, only consider examples where the intrinsic parameters are equal on both intervals. The domain of $B_k(t)$ in the examples will be $[0,1]$, and the knots will be uniformly distributed. Therefore, for the rest

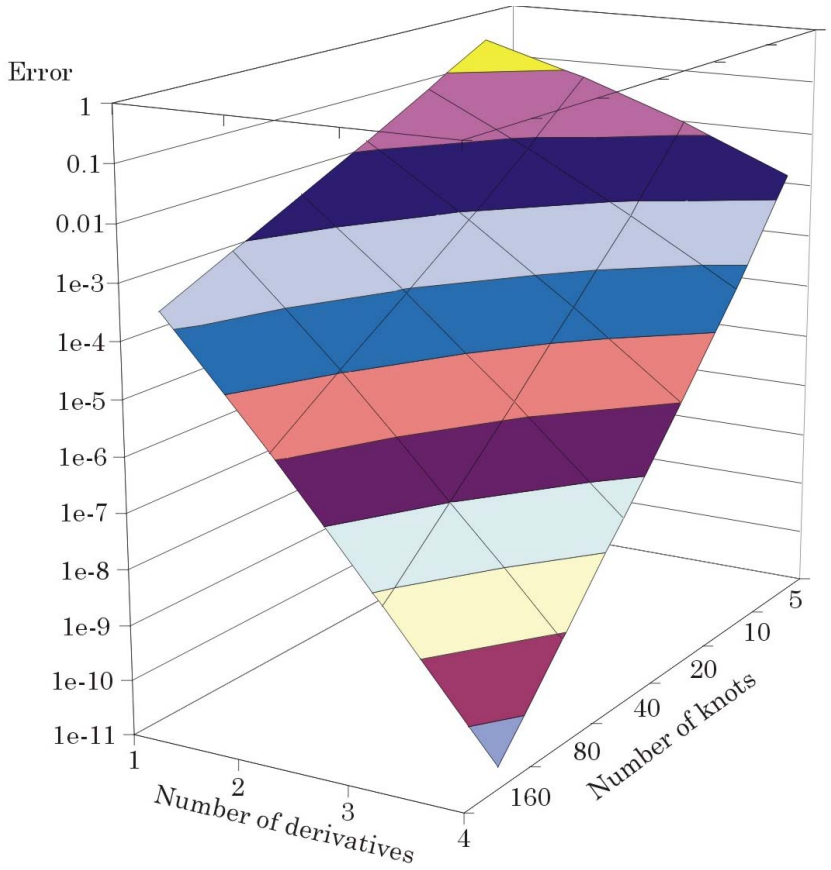


***Figure 2.12:*** *The surface shows the error (maximum deviation between the ERBS function and the sine function). Notice that the vertical axis uses a logarithmic scale. The horizontal axes show the number of derivatives and the number of knots, where the scale of the number of knots is logarithmic with 2 as the base. The numbers used to construct the surface can be found in table 2.1.*

of this section the basis function will be denoted

$$B(\alpha,\beta,\gamma,\lambda;t).$$

We will see four figures (2.13, 2.14, 2.15, 2.16), each with five plots of $B(\alpha,\beta,\gamma,\lambda;t)$, $t \in [0,1]$, and where only one of the intrinsic parameters vary, and the other parameters have default values. In the table below we can, for each of the four figures, see the intrinsic parameters that are not default, and the five different values they have.

| Figure | parameter | value 1 | value 2 | value 3 | value 4 | value 5 |
|---|---|---|---|---|---|---|
| 2.13 | $\alpha$ | 0.01 | 0.2 | 1 | 2 | 6 |
| 2.14 | $\beta$ | 0.01 | 0.2 | 1 | 6 | 100 |
| 2.15 | $\gamma$ | 0.001 | 0.1 | 1 | 2 | 10 |
| 2.16 | $\lambda$ | 0.01 | 0.3 | 0.5 | 0.7 | 0.99 |

In Figure 2.13, all intrinsic parameters have the default values except for $\alpha$. One can see that the shape of the basis function is close to a piecewise linear B-spline when $\alpha = 0.01$, and then getting smoother when $\alpha = 0.2$, but with a rather sharp top. Then it is rapidly getting smoother until $\alpha = 1$ (default value). It is then getting sharper when $\alpha = 2$ and until it looks more like piecewise linear trapezoid when $\alpha = 6$. This indicates that the shape of

$$\lim_{\alpha\to 0+} B(\alpha,1,1,0.5;t)$$

is a 1st degree B-spline basis. It also indicate that

$$\lim_{\alpha\to\infty} B(\alpha,1,1,0.5;t).$$

converges towards a piecewise linear function with breaks at approximately $\{0.075, 0.425, 0.575, 0.925\}$ (recalling that the domain of $B(t)$ is $[0,1]$). Notice that the two functions on the left hand side in Figure 2.13 are not in $C^\infty(0,1)$.

In Figure 2.14, all intrinsic parameters have the default values except for $\beta$. One can also see here that the shape of the basis function is close to a piecewise linear B-spline when $\beta = 0.01$, and now it is gradually getting smoother when $\beta = 0.2$ and $\beta = 1$. This also indicates that the shape of

$$\lim_{\beta\to 0+} B(1,\beta,1,0.5;t)$$

is converging towards a linear polynomial B-spline basis. When $\beta$ increases further to 6 and then to 100, one can clearly see that

$$\lim_{\beta\to\infty} B(1,\beta,1,0.5;t)$$

is converging towards a step function, which will approximate a zero degree B-spline basis. Notice that $B(1,\beta,1,0.5;t)$ is in $C^\infty(0,1)$ when $\beta$ obeys the restriction in definition 2.2 which, of course, is the case for all functions in Figure 2.14. Notice also that variation of these two parameters, $\alpha$ and $\beta$ preserves the symmetry, while varying of the other two parameters $\gamma$ and $\lambda$ does not preserve the symmetry.

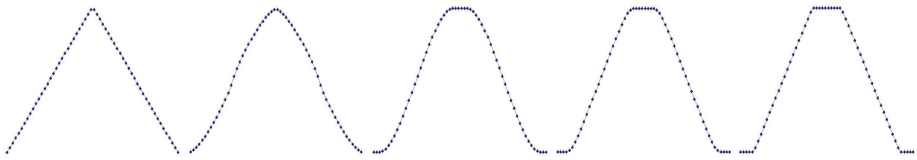

***Figure 2.13:*** *Five versions of* $B(\alpha,\beta,\gamma,\lambda;t)$ *where all intrinsic parameters have the default values, except for* $\alpha$*. The domains for each of the five functions is* $[0,1]$*. From left to right* $\alpha = \{0.01,\ 0.2,\ 1$ *(default value),* $2,\ 6\}$.

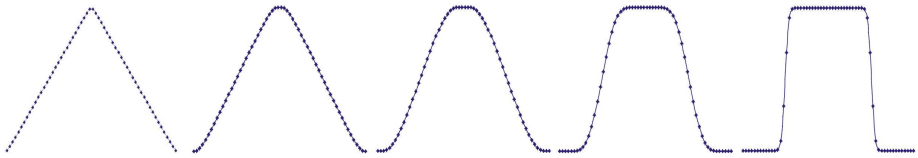

***Figure 2.14:*** *Five versions of* $B(\alpha,\beta,\gamma,\lambda;t)$ *where all intrinsic parameters have the default values, except for* $\beta$*. The domains for each of the five functions is* $[0,1]$*. From left to right* $\beta = \{0.01,\ 0.2,\ 1$ *(default value),* $6,\ 100\}$.

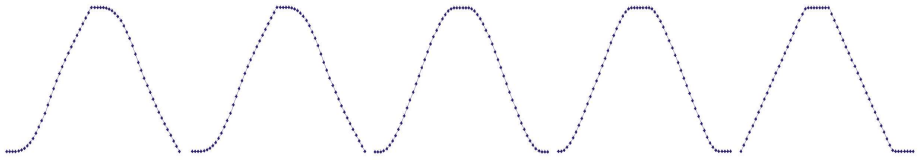

***Figure 2.15:*** *Five versions of* $B(\alpha,\beta,\gamma,\lambda;t)$ *where all intrinsic parameters have the default values, except for* $\gamma$*. The domains for each of the five functions is* $[0,1]$*. From left to right* $\gamma = \{0.001,\ 0.1,\ 1$ *(default value),* $2,\ 10\}$*. Notice that the difference between* $\gamma = 0.1$ *and* $\gamma = 0.001$ *is small*

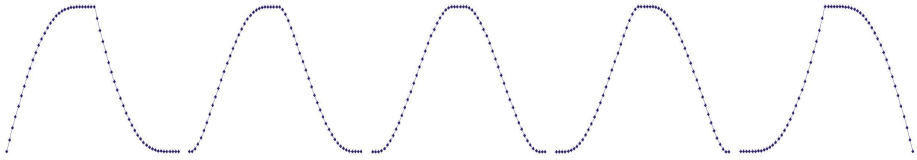

***Figure 2.16:*** *Five versions of* $B(\alpha,\beta,\gamma,\lambda;t)$ *where all intrinsic parameters have the default values, except for* $\lambda$*. The domains for each of the five functions is* $[0,1]$*. From left to right* $\lambda = \{0.01,\ 0.2,\ 0.5$ *(default value),* $0.7,\ 0.99\}$*. One can clearly see that the two functions on the left hand side is mirroring the two functions on the right hand side, i.e., mirror around* $\lambda = 0.5$.

In Figure 2.15, all intrinsic parameters have the default values, except for $\gamma$. One can clearly see that on the left hand side the two functions are close to equal, which indicates very fast convergence. The "breakpoint" at the top of these functions is at the interior knot (midpoint). The two halves are turned compared to each other (because they sum up to one). On the right hand side one can clearly see that

$$\lim_{\gamma \to \infty} B(1,1,\gamma,0.5;t)$$

is converging towards a piecewise linear function with breaks at approximately $\{0.38, 0.5, 0.88\}$ (the domain of $B(t)$ is $[0,1]$). Notice that the two functions on the left hand side, in Figure 2.15, are not in $C^{\infty}(0,1)$.

In Figure 2.16, all intrinsic parameters have the default values, except for $\lambda$. This parameter, restricted to $[0,1]$, is moving the "center of mass" towards the left hand side or the right hand side. The function on the left hand side is weighing the local function on the right hand side more than the local function on the left hand side. We can also see that $B(1,1,1,\lambda;t)$ is in $C^{\infty}(0,1)$ when $0 < \lambda < 1$, which is the case for all functions in Figure 2.16.

**Remark 6.** *Note that the four figures (2.13, 2.14, 2.15, 2.16) represent only a small part of the diversity of possible shapes of the Expo-Rational bell-function. By tuning the intrinsic parameters of the B-spline not only separately, but altogether or in groups, one can obtain fine control over an Expo-Rational B-spline curve/surface.*

# Chapter 3

# ERBS-evaluators, reliability, precision and efficiency

An Expo-Rational B-spline "evaluator" is a computation of $B_k(t)$ (and thus the integrals in definition 2.2 and the computation of the exponential function $\psi_k(t)$ in expression (2.4)). In addition, the computation of $D^j B_k(t)$ in (2.7) for $j = 1,2,...,d$ for some d is required, which includes computation of $f_{j,k}(t)$. (In practical computations, d ranges between 0 and 7). All this involves handling overflow and underflow and thus division by zero in the fractions, stable and precise numerical integrations and methods for speeding up the computations.

This chapter will only consider the scalable subset $\mathfrak{B}$ (see definition 2.4), where $B_k(t)$ is defined in (2.27), and $D^j B_k(t)$, $j = 1,2,...$ is defined in (2.32). We consider the scalable subset, and in most cases the default set, the most natural to use in geometric design. The scalability makes it possible to speed up the algorithm tremendously, and it makes the algorithms more stable. All algorithms in this chapter are either based on the scalable or the default set, and we will see that for the scalable subset $\mathfrak{B}$, there exist a reliable, precise and efficient evaluator for the basis function and some of the derivatives (for the default set, it should be possible to compute up to 49 derivatives).

This chapter considers the implementation, programming, and problems related to the number system of the computer. In the first section (section 3.1) we will investigate the requirements for a reliable algorithm. The questions here are overflow, underflow and division by zero, and we will investigate these using "IEEE binary floating point" standardized devices. The second section is treating algorithms for the derivatives. Then in the next section (section 3.3), we will investigate an algorithm for numerical integration of the expression in (2.27) and (2.28), namely the integral

$$\int_{s=0}^{w_k(t)} \phi(\alpha_k, \beta_k, \gamma_k, \lambda_k; s) ds, \quad \text{where} \quad 0 < w_k(t) \leq 1,$$

because

$$w_k(t) = \frac{t - t_k}{t_{k+1} - t_k} \quad \text{and} \quad t_k < t \leq t_{k+1}.$$

In [16], section 6, a sequence of numerical methods for computing ERBS were considered. In this report we study only the simplest of them, because we will use an algorithm with fine control of the precision and a simple implementation.

In the fourth section we will show an implementation and a test of a precise (with controllable precision) and extremely fast evaluator for practical use. The evaluator is based on preevaluation and Hermite interpolation.

## 3.1 Reliability in evaluations

Reliability of the algorithm depends on possibilities for overflow, underflow and division by zero. We, therefore, start by looking at what these three phenomena are, and when they occur. The IEEE standard for Binary Floating-Point Arithmetic [52] describes the formats of floating-point numbers (there is an ongoing revision called IEEE 754R [60]). According to the present standard, binary floating point numbers shall be of the form

$$(-1)^s \, 2^E \, (b_0.b_1b_2\ldots b_{p-1}), \tag{3.1}$$

where $p$ is the precision and,

$$\begin{array}{ll} s \in \{0,1\} & \text{(binary)}, \\ E \in \{E_{min},\ldots,E_{max}\} & \text{(signed integer)}, \\ b_i \in \{0,1\} & \text{(binaries)}. \end{array}$$

The following table describes how the bits in the numbers are distributed.

| type | sign | significant bits (precision) | bits for exponent | sum bits |
|---|---|---|---|---|
| single precision | 1 | 23 | 8 | 32 |
| double precision | 1 | 52 | 11 | 64 |

The form defined in (3.1) is defining the so-called normal values. In addition, the IEEE standard specifies the following special values for numbers; $\pm 0$ (signed zero), denormalized values (in 745R changed to subnormal), $\pm\infty$ and signaled and quiet NaN (Not a Number). Usually the first significant bit $b_0$ is 1, because if it is not, one can always, for normal values, obtain it by reducing the exponent $E$. For subnormal values, this is not the case, because we are now already using $E_{min}-1$. Therefore, for numbers with subnormal values, the first significant bit is always 0. This fact makes it possible to skip this first bit, and thus raise the precision (number of significant bits), because the separation between normal and subnormal values is well defined without the first significant bit (see the table below). The following table (see [31]) shows us how the 5 different types of values can be separated.

| Type | values | implemented exponent | implemented precicion |
|---|---|---|---|
| Special values | $\pm 0$ | $E = E_{min}-1$ | $b = 0$ |
| Subnormal values | $\pm 0.b \times 2^{E_{min}}$ | $E = E_{min}-1$ | $b \neq 0$ |
| Normal values | $\pm 1.b \times 2^E$ | $E_{min} \leq E \leq E_{max}$ | |
| Special values | $\pm\infty$ | $E = E_{max}+1$ | $b = 0$ |
| Special values | s/q NaN | $E = E_{max}+1$ | $b \neq 0$ |

Using this improved precision (skipping the first bit), we get the following number of significant digits in the decimal number system. For normal values, the biggest value for single precision (float) is $2^{24}-1 = 16777215$, i.e. more than 7 significant digits, and for double precision it is $2^{53}-1 = 9007199254740991$, i.e. close to 16 significant digits. For subnormal values the number of significant digits is reduced, and is gradually decreasing until there is only 1 binary digit (case discussed further below).

To describe overflow, and how it occurs, we first look at the maximum normal value. For single precision it is

$$1.11\ldots11\times 2^{2^{8-1}-1} = \left(2-2^{-23}\right)2^{127} \approx 3.4028237\, e+38.$$

For double precision it is

$$1.11\ldots11\times 2^{2^{11-1}-1} = \left(2-2^{-52}\right)2^{1023} \approx 1.7976931348623159\, e+308.$$

Numbers that becomes larger than this will be set to $\pm\infty$ depending on the sign bit (*overflow is actually assigning the special values $\pm\infty$ to a number*).

To describe underflow, and how it occurs, we first look at the minimum normal value. For single precision it is [1]

$$1.00\ldots00\times 2^{2-2^{8-1}} = 2^{-126} \approx 1.1754944\, e-38.$$

For double precision it is

$$1.00\ldots00\times 2^{2-2^{11-1}} = 2^{-1022} \approx 2.22507385850720138\, e-308.$$

Values smaller than this are subnormal values with lower precision. Notice that the significant bit will be $0.11\ldots11$ (in binary representation) for the first subnormal value, and $0.00\ldots01$ for the last subnormal value. Therefore, the minimum (unsigned) subnormal value is, for single precision,

$$2^{2^{-126-23}} \approx 1.4012984\, e-45,$$

and, for double precision,

$$2^{2^{-1022-52}} \approx 4.9406564584124654\, e-324.$$

Numbers smaller than this will be set to $\pm 0$ depending on the sign bit. Looking at the numbers above we can see that for both single and double precision then

$$\frac{1}{\text{min normal value}} < \text{max normal value}.$$

It follows from this that if the denominator in a fraction has a normal value, and the numerator is $\leq 1$ then the fraction will not produce an overflow.

For a closer study of how the number system affects algorithms, we recommend [31]. An edited reprint can be found at "http://docs.sun.com/source/806-3568/ncg_goldberg.html".

Summing up:

[1] The reason for using a total of 3 less numbers in the exponent than available is that one is used for zero and 2 are used for special and subnormal values as showed in the previous table.

- Overflow produces a signal and $\pm\infty$, which cannot be legally used further in most computations.
- Underflow first produces subnormal values and then $\pm 0$.
- Fractions might produce a signaled overflow.
  - Division by zero clearly produces a signaled overflow.
  - If the numerator in a fraction is $\leq 1$ there will never be an overflow if the denominator is a normal value.

Now let us go back to the search of a reliable algorithm for an ERBS evaluator. The first critical part is the computation of the fraction in the expression (2.29), namely,

$$-\beta\frac{|t-\lambda|^{(1+\gamma)\alpha}}{\left(t\,(1-t)^{\gamma}\right)^{\alpha}}, \quad \text{where } \alpha>0,\ \beta>0,\ \gamma>0,\ 0\leq\lambda\leq 1 \text{ and } t\in[0,1], \tag{3.2}$$

and later also the fraction $f_{j,k}(t)$, $j=2,3,...$, described in (2.32). When we look at the numerator of the fraction in (3.2), we can see that

$$|t-\lambda|\leq 1, \quad \text{if } 0\leq\lambda\leq 1 \text{ and } t\in[0,1],$$

and thus that

$$|t-\lambda|^{(1+\gamma)\alpha}\leq 1, \quad \text{if } \alpha>0, \quad \gamma>0, \quad 0\leq\lambda\leq 1 \text{ and } t\in[0,1].$$

We, therefore, get the following remark and, thus, reliable algorithm.

**Remark 7.** *It follows that it is not possible to get overflow even if the denominator is as small as possible, but still a normal number. The only critical part in the algorithm for computing $\phi(t)$ (expression (3.2), (2.29) and (2.44)) is the underflow of the denominator, i.e., that the number in the denominator is not a normal value.*

In the following we give an algorithm first for the general scalable set $\mathfrak{B}$, and then for the default set.

**Algorithm 1.** *(For notation, see section "Algorithmic Language", page 7.)*
*The algorithm computes $\phi(\alpha,\beta,\gamma,\lambda;t)$ from expression (2.29), where $\alpha$, $\beta$, $\gamma$ and $\lambda$ are present and, thus, states, and $t\in[0,1]$ is the input parameter. First follows a general function, then there is an optimized function for the default set of intrinsic parameters, (expression (2.44)).*

```
double φ ( double   t )
   double d = (t (1 − t)^γ)^α;
   if ( d ≠ normal value )  return 0.0;          // Test if underflow
   else                     return e^(−β |t−λ|^((1+γ)α) / d);

double φ ( double   t )
   t −= 0.5;
   t *= t;
```

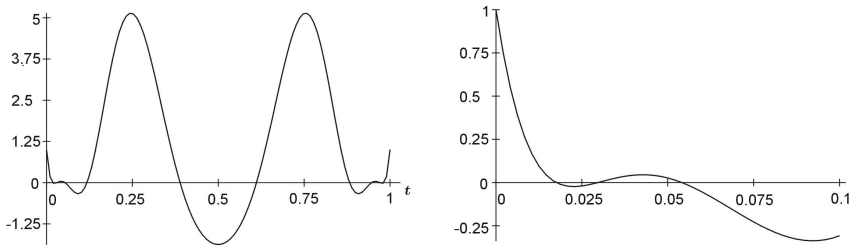


***Figure 3.1:*** *The numerator for $f_{7,d}(t)$. On the left hand side is a graph from 0 to 1, on the right hand side a "zoomed version" from 0 to 0.1. One can clearly see that $|f_{7,d}(t)| \leq 1$, $t \in [0,0.1]$.*

```
double  d = 0.25 − t;
if ( d ≠ normal value )  return  0.0;          // Test if underflow
else                     return  e^(−t/d);
```

Computing the second derivative and higher, we shall first look at the default set of intrinsic parameters. The exponential function $\phi(t)$, described in (2.44), and plotted in Figure 2.1, is symmetric about $t = 0.5$, and rapidly goes towards 0 when t goes towards 0 or 1. The number system on the computer, however, is, as we could see, only approximating real numbers, and it is ,therefore, necessary to clarify: when do we actually get underflow computing $\phi(t)$, and when do we get underflow computing the denominator in $f_{jd}(t)$. Computing on an Intel (Pentium 4 or Pentium Mobile) based computer and using double precision we find that for

$$\widehat{t} = 0.0003525365489384924,$$

we get

$$\phi(\widehat{t}) = e^{-708.39641853223657} = 2.225073858568479\, e-308,$$

which is the lowest value before underflow and thus giving a subnormal value. Recall that $\phi(t)$ is symmetric around $t = 0.5$ when using the default intrinsic parameters, and that $1-\widehat{t}$ is the limit on the right hand side.

In expression (2.46) and in Figure 2.8, one can see the expression of the numerators and the expression of denominators of $f_{j,d}(t)$, $j = 2,3,4,5,6,7$, and a plot of the same numerators and denominators for $j = 2,3,4,5$. From this it is clear that the denominator for $f_{j,d}(t)$ for all $j$ can be expressed in the following way

$$(2t(1-t))^{2(j-1)},$$

and that the absolute value of the numerator in $f_{j,d}(t)$ for the first 5 derivatives is less or equal to 1, and close to 0 or 1, all numerators go from 0 towards $\pm 1$. In Figure 3.1 one can clearly see that the numerator is less or equal to 1, in the area near the knots, also for the seventh derivative. Computing on an Intel-based computer using double precision and $t = \widehat{t}$ we get

$$(2\widehat{t}(1-\widehat{t}))^{2(49-1)} = 2.60482\, e-303.$$

For $j = 50$ we get underflow and thus a subnormal value.

**Remark 8.** *This shows that at least for the default set of intrinsic parameters, we can compute the seven first derivatives, and it indicates that we can compute 49 derivatives and only do a test if $\phi(t)$ is underflowing, and thus deliver zero, otherwise we can just compute and assemble the parts, always getting a valid and reliable result.*

## 3.2 ERBS-evaluator and derivatives

We are now ready to introduce a reliable algorithm for computing the derivatives for the Expo-Rational B-Spline using the default set of intrinsic parameters.

**Algorithm 2.** *(For notation, see section "Algorithmic Language", page 7.)*
*The algorithm computes $D^j B_k(t),\ j = 0,1,...,d,$ for the default set of intrinsic parameters. It is, thus, an implementation of (2.44–2.46). The algorithm assumes that there are algorithms to compute both $\phi(t)$, and $\int_{s=0}^{t}\phi(s)ds$, $t \in [0,1]$. The knot vector $\{t_i\}_{i=0}^{n+1}$ is supposed to be present and, thus, a state. The input variables are:* [2] *$t \in [t_1,t_n]$, $k$ for which $t_k < t \le t_{k+1}$, and $d \in \{0,1,\ldots,6,7\}$ (the number of derivatives to compute). The algorithm depends on $k$ being consistent according to $t$ and the knot vector. The return is a* "vector⟨double⟩"*, where the first element contains $B_k(t)$, and then $DB(t),\ldots,D^dB(t)$.*

vector⟨double⟩ *B (* double *t,* int *k,* int *d )*
$t = \frac{t-t_k}{t_{k+1}-t_k}$; *// Mapping from the global domain $[t_1,t_n]$ to the local domain $[0,1]$.*
vector⟨double⟩ $R(d+1) = 0.0$; *// Make a vector for return, size d+1, all elements 0.*
double $p = \phi(t)$; *// See algorithm 1, part 2.*
if *($p \neq$ normal value)* return *R; // Test if underflow.*
$R = S_d$; *// Each of the $d+1$ elements equal to $S_d$.*
double $\widetilde{t} = (1-2t)^2$;
switch *(d)*
case *7:* $R_7 * = \big(\big(\big(\big(\big(\big(\frac{315}{4}\widetilde{t}+\frac{1575}{4}\big)\widetilde{t}-\frac{8505}{8}\big)\widetilde{t}+\frac{465}{4}\big)\widetilde{t}+\frac{9645}{8}\big)\widetilde{t}+$ *... see (2.46)*
case *6:* $R_6 * = \big(\big(\big(\big(\big(\frac{45}{2}\widetilde{t}+\frac{135}{2}\big)\widetilde{t}-\frac{765}{4}\big)\widetilde{t}+75\big)\widetilde{t}+66\big)\widetilde{t}-\frac{85}{2}\big)\widetilde{t}+\frac{15}{4}$;
case *5:* $R_5 * = \big(\big(\big(\big(\frac{15}{2}\widetilde{t}+\frac{45}{4}\big)\widetilde{t}-33\big)\widetilde{t}+\frac{29}{2}\big)\widetilde{t}+\frac{3}{2}\big)\widetilde{t}-\frac{3}{4}$;
case *4:* $R_4 * = \big(\big(3\widetilde{t}+\frac{3}{2}\big)\widetilde{t}-5\big)\widetilde{t}+\frac{3}{2}$;
case *3:* $R_3 * = \frac{3}{2}\widetilde{t}^2-\frac{1}{2}$;
$R_0 * = \int_{s=0}^{t}\phi(s)ds$; *// See algorithm 6 in the next section.*
for *(* int *i=1; $i \le d$; i++ )* $R_i * = \frac{p}{(t_{k+1}-t_k)^i(2t(1-t))^{2(i-1)}}$;
for *(* int *i=2; $i \le d$; i+=2 )* $R_i * = (1-2t)$;
return *R;*

[2] It is preferable that the value of $k$ for which $t_k < t \le t_{k+1}$, see (2.27), is computed separately from the evaluation of the basis function, because this knowledge is also needed for the local functions, see (2.49). The question as to whether the local mapping, see (2.28), and scaling of the derivatives, see (2.32), should be done inside the evaluator (as in the first and third last line in Algorithm 2) might be an open question. The question is really if the evaluator should be independent of the knot vector, or be complete.

The algorithm for the scalable subset $\mathfrak{B}$ will in general be the same as for the default set of intrinsic parameters. The differences are mostly limited to computing $f_{j,k}(t)$ (2.11). Because of the complexity of the algorithms, $f_{j,k}(t)$ will be implemented in separate algorithms. The choice is that the algorithm should be able to deal with asymptotes, both for $f_{2,k}(t)$ in (2.36), (when $(1+\gamma)\alpha < 1$, see (2.37)), and for $f_{3,k}(t)$ in (2.39) (when $(1+\gamma)\alpha < 2$, see (2.42)). This is done under the additional restriction $0 < \lambda < 1$. The implementation of $f_{2,k}(t)$ in (2.38) leads to implementation of $\zeta_k'(t)$ in (2.35). Because $x_{2,k}(t)$ in (2.36) consists of two parts where one might have an asymptote, we start by multiplying and factorizing, and we get

$$x_{2,k}(t)\zeta_k(t) = \frac{-\alpha\beta(\gamma+1)}{\left(t\,(1-t)^{\gamma}\right)^{\alpha}} \left( \frac{t-\frac{1}{1+\gamma}}{t(1-t)} |t-\lambda| + \text{sign}(t-\lambda)1 \right) |t-\lambda|^{(1+\gamma)\alpha-1}. \qquad (3.3)$$

Now it is time to look at an algorithm computing $f_{2,k}(t)$.

**Algorithm 3.** *(For notation, see section "Algorithmic Language", page 7.)*
*The algorithm computes $f_{2,k}(t)$ for the scalable subset $\mathfrak{B}$ of the intrinsic parameters, where $\alpha_k$, $\beta_k$, $\gamma_k$ and $\lambda_k$ are supposed to be present and thus are states, $S_k = S(\alpha_k, \beta_k, \gamma_k, \lambda_k)$ is supposed to be pre-evaluated and, thus, a state. The algorithm is the implementation of (2.38) and, thus, uses (3.3). The input variable is supposed to be $t \in [0,1]$, and it is such that the second line in the algorithm shall guarantee that t in the computation of (3.3) has a normal value on the open segment $(0,1)$.*[3]

```
double f2 ( double t )
    if ( t < 2.3e-308 || t == λ || t == 1 )     // See (2.38), upper part.
        return 0.0;
    double h = (t - 1/(1+γ)) / (t(1-t)) |t-λ|;  // First part of the second factor from (3.3).
    if ( t < λ )    h -= 1;                     // Last part of second factor (3.3).
    else            h += 1;
    h *= -S_k αβ(γ+1) / (t(1-t)^γ)^α;           // Inserting S_k and the first factor of (3.3).
    if ( (1+γ)α < 1 )                           // Asymptote at t = λ is present.
        double g = |t-λ|^(1-(1+γ)α) / h;        // The inverse of (3.3).
        if ( g ≠ normal value )
            return 0.0;
        else
            return 1/g;
    else if ( (1+γ)α > 1 )                      // Ordinary solution.
        return h |t-λ|^((1+γ)α-1);
    else                                        // Discontinuity at t = λ.
        return h;
```

A comment to the foregoing algorithm; it is, as said abow, treating the have of possible asymptote $(1+\gamma)\alpha < 1$, see (2.37), for all $t$ values. The convention of what to do at $t = \lambda$ when there is no value is that it returns 0, as it will do for all cases when there is a value.

[3] The guarantee is that we have to introduce practical restrictions on the intrinsic parameters because of the binary number system. See Table 3.1 and the following comments to that table.

The implementation of $f_{3,k}(t)$ (2.43) leads to implementation of $x_{3,k}(t)\zeta_k(t)$ from (2.43). Because $x_{3,k}(t)$ (2.39) consists of four parts where two of the parts have an asymptote when $(1+\gamma)\alpha < 2$, and one of the parts in addition also has an asymptote when $(1+\gamma)\alpha < 1$, we start by multiplying and factorizing, and we then get

$$x_{3,k}(t)\zeta_k(t) = \frac{-\alpha\beta}{\left(t\,(1-t)^{\gamma}\right)^{\alpha}}\left(|t-\lambda|^2\,\frac{t^2\,(\gamma+1)-2t+1}{(t\,(t-1))^2}-(\gamma+1)+\right.$$
$$\left.\alpha\left(1-\frac{\beta\,|t-\lambda|^{(1+\gamma)\alpha}}{\left(t\,(1-t)^{\gamma}\right)^{\alpha}}\right)\left(\frac{t\,(\gamma-\lambda-\lambda\gamma)+\lambda}{t\,(1-t)}\right)^2\right)|t-\lambda|^{(1+\gamma)\alpha-2}. \tag{3.4}$$

We will now look at an algorithm computing $f_{3,k}(t)$.

**Algorithm 4.** *(For notation, see section "Algorithmic Language", page 7.)*
*The algorithm computes $f_{3,k}(t)$ for the scalable subset $\mathfrak{B}$ of the intrinsic parameters, where $\alpha_k$, $\beta_k$, $\gamma_k$ and $\lambda_k$ are supposed to be present and, thus, states, $S_k = S(\alpha_k,\beta_k,\gamma_k,\lambda_k)$ is supposed to be pre-evaluated and thus a state. The algorithm is the implementation of (2.43) and, thus, uses (3.4). The input variable is supposed to be $t \in [0,1]$, and it is such that the second line in the algorithm shall guarantee that t in the computation of (3.4) has a normal value on the open segment $(0,1)$.*

```
double f3 ( double t )
  if ( t == λ )                                   // See (2.43), upper part.
    if ( (1+γ)α > 2 )        return 0.0;          // "Ordinary" continuity.
    else if ( (1+γ)α == 2 )  return -2β/(λ^α(1-λ)^(2-α));   // "Standard" continuity.
    else if ( (1+γ)α > 1 )   return -"max normal value"; // Negative asymptote.
    else if ( (1+γ)α == 1 )  return 0.0;          // Continuity.[4]
    else if ( (1+γ)α < 1 )   return "max normal value"; // Positive asymptote.
  if ( t < 2.3e-308 || t == 1 ) return 0.0;
  double h = |t-λ|^2 (t^2(γ+1)-2t+1)/(t(1-t))^2 - (γ+1);   // First part of the second factor of (3.4).
  h += α ( 1 - β|t-λ|^(1+γ)α / (t(1-t)^γ)^α ) ( t(γ-λ-λγ)+λ / t(1-t) )^2;  // Second part of the second factor.
  h *= -S_k αβ / (t(1-t)^γ)^α;                    // Inserting S_k and the first factor.
  if ( (1+γ)α < 2 )                               // Asymptote at t = λ might be present.
    double g = |t-λ|^(2-(1+γ)α) / h;              // The inverse of (3.4).
    if ( g ≠ normal value )
      return "signed max normal value";
    else
      return 1/g;
  else if ( (1+γ)α > 2 )                          // "Ordinary" solution.
    return h |t-λ|^((1+γ)α-2);
  else                                            // "Standard" solution (e.g. default set).
    return h;
```

A comment to the foregoing algorithm; it is as said above, treating the asymptotic case $(1+\gamma)\alpha < 2$, see (2.37), for all $t$ values. The convention of what to do at $t = \lambda$, when a function value does not exist, is that it returns an approximation of $\lim_{t\to\lambda} f_3(t) = \mp\infty$ by "Normal values", where the sign is negative for $1 < (1+\gamma)\alpha < 2$, and positive for $0 < (1+\gamma)\alpha < 1$. The reason for the sign can clearly be seen in expression (2.41) , where the only factor that is not necessarily positive is $\alpha_k(1+\gamma_k) - 1$. This also shows that when $\alpha_k(1+\gamma_k) = 1$, the function becomes continuous.

There is one question in a footnote connected to Algorithm 3, and also connected to Algorithm 4: what are the practical restrictions on the intrinsic parameters that guarantee no underflow and, thus, no overflow by division? The question is then: what is the range of the intrinsic parameters that guarantee a reliable algorithm for computing the derivatives (for all $t$ in the open segment $(0,1)$) which have a normal value? (Note that subnormal values shall not be considered). It follows that what actually has to be investigated is the range of the intrinsic parameters to guarantee that:

$$\phi(t) < \frac{1}{|f_j(t)|} \quad \text{for} \quad j = 2,3 \quad \text{when} \quad t \to 0+ \quad \text{or} \quad t \to 1-, \quad \text{and where } t \text{ is a normal value.}$$

In the following, we will only consider changes of the intrinsic parameters one by one. In a finite binary number system, the range of an intrinsic parameter can actually be computed "exactly". Recall from section 3.1 that the maximum negative binary exponent in double precision is 1022 and that the number of significant bits is 52. The sum of this is 1074. A number between 0 and 1 can, therefore, be found "exactly" by binary search with depth 1074. If the range is bigger than this, we also have to consider the maximum positive binary exponent, and then the depth is 1074+1023=2097. The algorithm is a two-level binary search, the outer level being to find the intrinsic parameter, the inner level being to find the first $t$ where $\phi(t)$ does not have a normal value in order to see if $\frac{1}{|f_j(t)|}$ has still a normal value (e.g. $> \phi(t)$). The result of a search for all intrinsic parameters separately, for upper and lower bounds, for both when $t \to 0+$ and $t \to 1-$, and for second and third derivatives, can be seen in Table 3.1.

From Table 3.1 we can draw the following conclusion of the range of the intrinsic parameters which guarantees reliable algorithms for computing second and third derivatives:

| | | | |
|---|---|---|---|
| Lower and upper bounds of the intrinsic parameters which guarantee reliability when computing the second derivative. | $0.1857$ | $< \alpha <$ | $1075$ |
| | $3.146 \times 10^{-13}$ | $< \beta <$ | $+\infty$ |
| | $0.201$ | $< \gamma <$ | $534.37$ |
| | $6.72 \times 10^{-152}$ | $< \lambda <$ | $0.999999$ |
| Lower and upper bounds of the intrinsic parameters which guarantee reliability when computing the third derivative. | $0.1857$ | $< \alpha <$ | $465.97$ |
| | $3.146 \times 10^{-13}$ | $< \beta <$ | $1.436 \times 10^{34}$ |
| | $0.201$ | $< \gamma <$ | $496$ |
| | $1.337 \times 10^{-75}$ | $< \lambda <$ | $0.999999$ |

Let us recall that reliability means that we do not get overflow and, thus, we do not get numbers that are not usable for further computations. We can see from the range of the

[4] When $(1+\gamma)\alpha = 1$, all even derivatives will only be discontinuous, and all odd derivatives will be continuous (see proof of Theorem 2.1, item k-II on page 24).

| par. | der. | bound | $t \to 0$ | $t \to 1$ |
|---|---|---|---|---|
| $\alpha$ | 2 | lower | $9.294556548602447 \times 10^{-3}$ | 0.1856551589777102 |
| | | upper | 1075 | 1075 |
| | 3 | lower | 0.01864443248188489 | 0.1856551589777102 |
| | | upper | 466.889890736804 | 465.9770256841861 |
| $\beta$ | 2 | lower | $4.466312333786737 \times 10^{-302}$ | $3.145912057383679 \times 10^{-13}$ |
| | | upper | max normal value | max normal value |
| | 3 | lower | $1.124974606116484 \times 10^{-149}$ | $3.145912057383679 \times 10^{-13}$ |
| | | upper | $5.747194528456591 \times 10^{34}$ | $1.436798632114148 \times 10^{34}$ |
| $\gamma$ | 2 | lower | 0 | 0.2013156366896773 |
| | | upper | 1002.994782602334 | 534.3735944906107 |
| | 3 | lower | 0 | 0.2013156366896773 |
| | | upper | 496.0047816968047 | 534.3735944906107 |
| $\lambda$ | 2 | lower | $6.713476659990283 \times 10^{-152}$ | 0 |
| | | upper | 1 | 0.9999997195578465 |
| | 3 | lower | $1.336727340482124 \times 10^{-75}$ | 0 |
| | | upper | 1 | 0.9999997195578465 |

***Table 3.1:*** *The table shows the result of the computation (binary search) to find the bounds on the intrinsic parameters to ensure reliable algorithms for computing second and third derivatives. For each of the four intrinsic parameters there are 4 numbers for computing second derivatives and 4 numbers for computing third derivatives. Because the formulas are not symmetric, and the number system is not symmetric, the numbers are computed both for when $t \to 0+$ and when $t \to 1-$. To compute the lower bounds, the depth used in the binary search is 1074, and for the upper bounds, the depth used in the binary search is 2097. The number of digits used is 16, and the decimal numbers represent binary numbers that are the exact bounds.*

parameters that $\alpha$ and $\gamma$ are the most "influential" parameters, which is natural, because they are exponents. It is also remarkable that the range of $\beta$ is so large that we can, by only varying $\beta$, almost get a piecewise linear, or a step function, and still be able to evaluate second and third derivatives for all t values.

We have previously been able to see that for the default set of intrinsic parameters, we can compute at least 7, and probably up to 49, derivatives without overflow trouble. Now it is time to introduce an algorithm for an "evaluator" for the scalable subset $\mathfrak{B}$ computing up to 3 derivatives.

**Algorithm 5.** *(For notation, see section "Algorithmic Language", page 7.)*
*The algorithm computes $D^j B_k(t),\ j = 0,1,...,d$ for the scalable set of intrinsic parameters $\mathfrak{B}$. The algorithm assumes that there are algorithms present for computing both $\phi(t)$, and $\int_{s=0}^{t}\phi(s)ds$ for $0 \le t \le 1$. The knot vector $\{t_i\}_{i=0}^{n+1}$ is supposed to be present and, thus, a state. The input variables are: $t \in [t_1,t_n]$, $k$ for which $t_k < t \le t_{k+1}$, and $d \in \{0,1,2,3\}$ (the number of derivatives to compute). The algorithm depends on k being consistent with the choice of t within the knot vector. The return of the function is a* "vector⟨double⟩", *where the first element is $B_k(t)$, and the next elements are $DB(t),\ldots,D^dB(t)$.*

vector⟨double⟩ *B (* double *t,* int *k,* int *d )*
  $t = \frac{t-t_k}{t_{k+1}-t_k}$; *// Mapping from the global domain $[t_1,t_n]$ to the local domain $[0,1]$.*
  vector⟨double⟩ $R(d+1) = 0.0$; *// Make a vector for return, size d+1, all elements 0.*
  double $p = \phi(t)$; *// See algorithm 1, part 2.*
  if *($p \neq$ normal value)* return *R; // Test if underflow.*
  switch *(d)*
    case *3:* $R_3 = f_3(t)$;
    case *2:* $R_2 = f_2(t)$;
    case *1:* $R_1 = S_k$;
  $R_0 = S_k \int_{s=0}^{t} \phi(s)ds$; *// See algorithm 6 in the next section.*
  for *(* int *i=1; $i \le d$; i++ )* $R_i * = \frac{p}{(t_{k+1}-t_k)^i}$;
  return *R;*

## 3.3 ERBS-evaluator based on Romberg-integration

The principal part of the ERBS-evaluator $B_k(t)$ defined in Definition 2.2 is the integration of $\psi_k(t)$ defined in (2.4) or $\phi(t)$ defined in (2.29). In this section, only one of the possible reliable and controllable numerical integration will be investigated. The choice fell on Romberg integrations (see [9]), which is based on repeated Richardson extrapolation to eliminate error terms (see [34]). The background for the algorithm is the Euler-MacLaurin integration formula (see [54]). Given a function $f$ that is $C^\infty[a,b]$, then the error from a trapezoidal approximation $T_n(f)$ according to the integral $I(f)$ is

$$I(f) - T_n(f) = \frac{1}{n^2}\sum_{j=0}^{\infty} A_j^{(0)} \frac{1}{n^{2j}} = \frac{1}{n^2}A_0^{(0)} + \frac{1}{n^4}A_1^{(0)} + \frac{1}{n^6}A_2^{(0)} + \ldots, \tag{3.5}$$

where $A_j^{(0)}$ are constants. For example, the error formula for the Trapezoidal and Simpson method gives

$$A_0^{(0)} = \frac{-(b-a)^2}{12}(f'(b) - f'(a)),$$
$$A_1^{(0)} = \frac{(b-a)^4}{180}(f^{(3)}(b) - f^{(3)}(a)).$$

Richardson extrapolation can now be used in conjunction with (3.5) to iteratively eliminate terms in the error formula. Richardson extrapolation is in general a method to improve an approximation by combining two equations using different step sizes. If we make two simplified versions of (3.5), using step size h instead of the number of steps n, and using half step size (h/2) on the second one, we get

$$I(f) = T_h(f) + A_0 h^{2j} + O(h^{2j+1}),$$
$$I(f) = T_{h/2}(f) + A_0 \left(\tfrac{h}{2}\right)^{2j} + O(h^{2j+1}).$$

If we put this into order and subtract the second from the first equation we get

$$B(h) = \frac{2^{2j} T_{h/2}(f) - T_h(f)}{2^{2j} - 1}, \tag{3.6}$$

where

$$I(f) = B(h) + O(h^{2j+1}).$$

If we also compute $B(h/2)$ we can use $B(h)$ and $B(h/2)$ in a next step (in (3.6)) to reduce the error terms. This can be repeated in an iterative process until the "removed error term" is smaller than a given tolerance.

There are three questions appearing when using Romberg integration to integrate (numerically) $\phi(t)$ defined in (2.29),

1. reliability,
2. efficiency,
3. precision.

What follows there will not be any attempt to compare Romberg integration with other methods, but there will be an investigation of how suitable Romberg integration is for integrations of $\phi(t)$. There following some tests done for several sub-domains (integration intervals), to find out how fast the quadrature process is converging, the number of computations of $\phi(t)$ and the remaining errors. The result can be seen in both a table and a figure.

First the reliability depends not only on the fact that $\phi(t)$ is bounded on $[0,1]$, but also on the fact that we do not get an overflow when actually computing $\phi(t)$ using algorithm 1. (Remark 7 state this). Since also the range of the integration interval is within $[0,1)$, the computation of the integral will always give normal value, and is in this sense reliable. The degree of reliability, the error, will be discussed below.

In Table 3.2 and in Figure 3.2 one can see the connection between the number of steps (and evaluations) in the Romberg-integration and the "last removed error term". Six evaluations (and integrations) are computed. In the computation, the default set of intrinsic

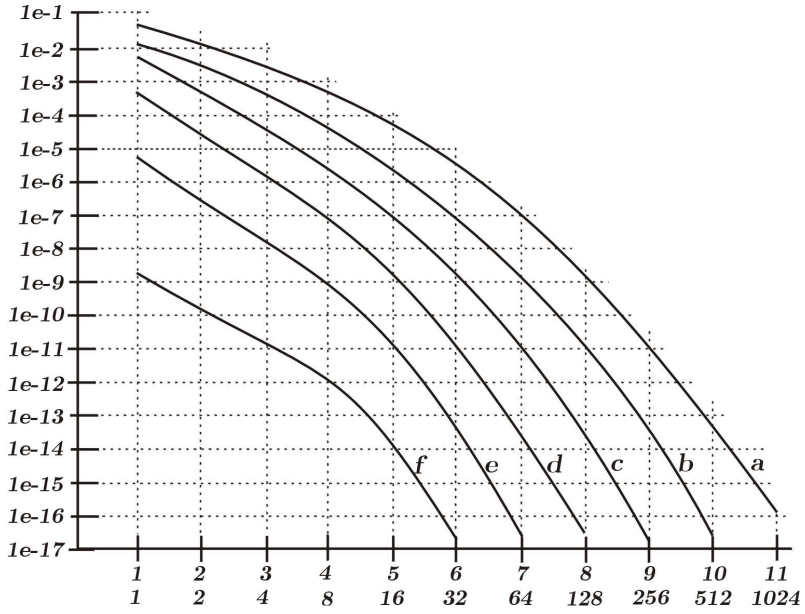


***Figure 3.2:*** *The figure shows the relationship between the precision and the number of iterations in the Romberg-integration algorithm for ERBS-evaluation when using the default intrinsic set of parameters. The upper scale on the horizontal axis is the number of iterations, and the lower scale is the new number of evaluations (i.e. at step 11 there are 1024 new computations of $\phi(t)$ and a total of 1+1+2+...+1024=2048 computations). The scale on the vertical axis is the difference in value between a step and the previous step in the iteration. Five graphs (a,b,c,d,e,f) are plotted from smoothed data in Table 3.2. Each graph represents an evaluation of a given t, i.e. $\int_{s=0}^{t} f(s)ds$. For graph* **f**, *$t = \frac{1}{64}$, for graph* **e**, *$t = \frac{1}{32}$, for graph* **d**, *$t = \frac{1}{16}$, for graph* **c**, *$t = \frac{1}{8}$, for graph* **b**, *$t = \frac{1}{4}$, and for graph* **a**, *$t = \frac{1}{2}$. One can clearly see that the number of iterations is related to the size of the integration interval (domain).*

| curve name | | **a** | **b** | **c** | **d** | **e** | **f** |
|---|---|---|---|---|---|---|---|
| upper limit | | 0.5 | 0.25 | 0.125 | 0.0625 | 0.03125 | 0.015625 |
| 1 | 1 | $7.2e-2$ | $1.4e-2$ | $8.3e-3$ | $7.7e-4$ | $7.3e-6$ | $1.2e-9$ |
| 2 | 2 | $2.0e-2$ | $2.6e-3$ | $7.9e-4$ | $1.1e-5$ | $1.4e-6$ | $3.2e-10$ |
| 3 | 4 | $1.3e-3$ | $9.3e-4$ | $7.0e-5$ | $1.6e-6$ | $1.8e-8$ | $8.4e-11$ |
| 4 | 8 | $9.5e-4$ | $8.5e-5$ | $2.9e-6$ | $6.9e-8$ | $1.6e-9$ | $4.0e-12$ |
| 5 | 16 | $8.9e-5$ | $3.3e-6$ | $8.0e-8$ | $1.7e-9$ | $1.9e-11$ | $1.6e-14$ |
| 6 | 32 | $3.4e-6$ | $8.3e-8$ | $1.8e-9$ | $2.0e-11$ | $4.8e-14$ | $2.2e-17$ |
| 7 | 64 | $8.4e-8$ | $1.8e-9$ | $2.0e-11$ | $5.3e-14$ | $2.8e-17$ | |
| 8 | 128 | $1.9e-9$ | $2.0e-11$ | $5.4e-14$ | $2.4e-17$ | | |
| 9 | 256 | $2.0e-11$ | $5.5e-14$ | $1.9e-17$ | | | |
| 10 | 512 | $5.5e-14$ | $2.8e-17$ | | | | |
| 11 | 1024 | $1.7e-16$ | | | | | |

***Table 3.2:*** *The table shows the connection between the number of steps/evaluations and the precision in the Romberg-integration algorithm. There are 6 "graphs" computed. Each graph is an integration from 0 to upper limit (second row). The column on the left hand side is showing the number of steps in the integration (up to 11), the next column is showing the new number of computations of $\phi(t)$ that have to be done on the current step. All the other numbers are the last correction, indicating the level of tolerance (remaining errors).*

parameters is used, and six different values are computed. The computations are of

$$\int_{s=0}^{t} \phi(s)ds, \quad \text{for} \quad t = \left\{ \frac{1}{2^k} \right\}_{k=1}^{6}.$$

Each computation is named by a letter, "a)" means that the integration interval is $\left[0, \frac{1}{2}\right]$. This is the computation over the biggest interval. The interval is then halved from computation to computation until the last computation, named "f)", where the integration interval is $\left[0, \frac{1}{64}\right]$. In Table 3.2 we will find the value of all "remaining errors" until they are "out of significant bits". The values are actually the differences between the result from this step in the iteration and the result from the previous step. As we can see in Table 3.2, a value in the table is clearly bigger than the sum of all values below in the same column. So the values are actually better than the "remaining errors". We can also see that the last values (on the bottom of the table) are numbers "outside the edge of significant bits", when the results are close to 1. In Figure 3.2 is the values from Table 3.2 plotted as 6 smooth graphs. On the horizontal axes is the number of new computations of $\phi(t)$ on a logarithmic scale. This tells us that the computational costs are doubled for each mark on the horizontal axes.

The depth of the iteration depends on the size of the integration interval. Evaluating an interval of 0.5 requires up to 11 iterations, which is the same as $2 \times 1024 = 2048$ computations of $\phi(t)$. This is indeed a time consuming process. In this case it is important to

be careful; one should not use higher tolerance than necessary. When evaluating numbers bigger than 0.5, one should use mirroring and then instead integrate on a subinterval of the right part of the domain, $\left[\frac{1}{2},1\right]$.

The following algorithm is only for integrating from 0 to $t$. To make an algorithm for the general case, integrating from $a$ to $b$, the change is to be done only in the third line where it must be $\frac{\phi(a)+\phi(b)}{2}$, and in line 9 where it must be $s$ += $\phi(a+j*t)$. In addition a new declaration introducing "double $t = b-a$;" must of course be included. The algorithm is optimal in the way that the evaluation of $\phi(t)$ is minimized. In some cases the evaluation on the boundaries are already known, in these cases the algorithm can easily be adapted either by using state variables or by using parameters.

**Algorithm 6.** *(For notation, see section "Algorithmic Language", page 7.)*
*The algorithm computes the integral* $\widetilde{B} = \int_{s=0}^{t} \phi(s)ds,\ t \in (0,1]$, *where* $\widetilde{B}$ *is inside the given tolerance* $\varepsilon$. *The value of the input "tolerance" variable* $\varepsilon$ *is recommended to be in* $[1e-2, 1e-15]$ *(according to Table 3.2).*

```
double integrate ( double t, double ε)
    double M[16][16];
    double sum = φ(t)/2;
    M_{0,0} = t * sum;
    for ( int i=1; i < 16; i++ )
        double s = 0;
        int k = 1 ≪ i;                          // N.B, using C++ bit shift operator ≪
        t /= 2;
        for ( int j=1; j < k; j+=2 )  s += φ(j*t);
        M_{0,i} = t * (sum += s);
        for ( int j=1; j ≤ i; j++ )
            double c = 4^j;                     // C++ implementation:  1 ≪ (j ≪ 1)
            M_{j,i-j} = (c*M_{j-1,i-j+1} - M_{j-1,i-j})/(c-1);   // Richardson extrapolation scheme
        if ( |M_{i,0} - M_{i-1,0}| < ε )  return M_{i,0};
    return M_{15,0};
```

## 3.4 Practical ERBS-evaluator using preevaluation and interpolation

To implement "ERBS" for use in geometric modeling and design, graphic computation, simulations and others, it is extremely important to have a precise and fast evaluator. In this section, we shall take a closer look at a method using preevaluation and Hermite interpolation.

In practical implementation the whole evaluation system should be wrapped in an "object" (C++ class or equivalent). The advantage of this is that it takes care of all states such as

the intrinsic parameters $\alpha$, $\beta$, $\gamma$, $\lambda$, the scaling factor $S(\alpha,\beta,\gamma,\lambda)$ (see (2.30)), and, of course, the number of samples (below denoted by $m$) and all sampled values (more than $6m$ in total, as will be described below). Therefore, in the following, an evaluation object type will be defined, (it will be called the "ERBS-evaluator") containing the following:

**"ERBS-evaluator"**

The following state variables are present:

| | |
|---|---|
| $\alpha$, $\beta$, $\gamma$, $\lambda$ | // Intrinsic parameters (the state identifiers, together with $m$). |
| $m$ | // The number of sample intervals, number of samples is $m+1$ |
| $\Delta t = \frac{1}{m}$ | // The interval between each sample (also the scaling factor), |
| | // the sampling vector $\{t_i\}_{i=0}^{m}$ defined via $t_i = i * \Delta t$. |
| $S = S(\alpha,\beta,\gamma,\lambda)$ | // Scaling factor (see (2.30)). |
| $\mathbf{b} = \left\{\int_0^{t_i} \phi(s)ds\right\}_{i=0}^{m}$ | // Vector storing of the integral $\int_0^{t_i} \phi(s)ds$ in the sample points. |
| $\mathbf{a}$ | // A matrix with dimension $m \times 5$ (actually $m$ vectors). |
| | // Each of the vectors$\{\mathbf{a}_i\}_{i=0}^{m}$ stores the Hermite coefficients |
| | // $a_0$, $a_1$, $a_2$, $a_3$ and $a_4$ for each sample interval (see 3.12). |

It will, of course, be necessary to have functions for construction, destruction, settings etc. They will not be handled here, but the important "public" functions are:

| | |
|---|---|
| $initiate(\alpha,\beta,\gamma,\lambda,m)$ | // Changing states and thus computing $S$, $\mathbf{b}$, $\mathbf{a}$ and $\Delta t$. |
| $B(t,k,d)$ | // For a given $t \in [0,1]$, k - knot interval, d - nr. of derivatives, |
| | // analogous to Algorithm 5, computing $D^j B(t),\ j = 0,\ldots,d$. |

For "internal" use (used only by initiate( )) we need the following functions (the three last functions are defined in previous sections):

| | |
|---|---|
| $interpolate(i,\phi_0,\phi_1,\phi_0',\phi_1')$ | // Computing $\mathbf{a}_{i,0}$, $\mathbf{a}_{i,1}$, $\mathbf{a}_{i,2}$, $\mathbf{a}_{i,3}$ and $\mathbf{a}_{i,4}$ (using (3.12)). |
| $\phi(t)$ | // Defined in Algorithm 1. |
| $f_2(t)$ | // Defined in Algorithm 3. |
| $integrate(t_0,t_1,\phi_0,\phi_1,\varepsilon)$ | // Modified version of Algorithm 6. This version uses |
| | // both the start and end value of the interval, and $\phi(start)$ |
| | // and $\phi(end)$, to minimize the number of evaluations. |

The question is, how can we use the "ERBS-evaluator"? A possible set of answers to this is listed below.

1. For a given set of intrinsic parameters and a given "acceptable tolerance", make an instance of an "ERBS-evaluator". All curves and surfaces using this set of parameters can now use this object for evaluation.

2. The parameters or the "acceptable tolerance" can, at any time, be changed for one or more specific curves/surfaces by making and using a new "ERBS-evaluator", or

by resetting the parameters in the "old" one.

3. It is possible to have more than one "ERBS-evaluator" available at the same time.
4. It is possible to have several instances of a curve/surface, the "ERBS-evaluator" needs only to be altered.

There is an obvious cost of using "ERBS-evaluators", related to the use of memory. This concerns, however, small numbers, from approximately 1.8 to 48 kbytes for each instance of an object, depending on the number of samples, and thus the required guarantee for the errors (the tolerance). The sample data consist of the vector **b** storing the incrementing integral samples, and the matrix **a** storing, in each line, the Hermite interpolation coefficients at each sampling interval. Only the first four values in each line, $\mathbf{a}_{i,0}$, $\mathbf{a}_{i,1}$, $\mathbf{a}_{i,2}$, $\mathbf{a}_{i,3}$, are actually the Hermite interpolation coefficients. The last one $\mathbf{a}_{i,4}$ is the integral from 0 to 1 of the Hermite interpolant. The reason for having this last element is to either integrate from 0 to $\leq 0.5$ or from $> 0.5$ to 1, so as to reduce the error. The equations for Hermite interpolation between 0 and 1 will now briefly be discussed (we will later look at the problems of the scaling of the derivatives because of domain scaling). We start with a general 3th degree polynomial equation,

$$f(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3, \tag{3.7}$$

and its first derivative,

$$f'(x) = a_1 + 2a_2 x + 3a_3 x^2. \tag{3.8}$$

Integrating (3.7) from 0 to a given value $\widehat{t} \in (0,1)$ gives the following result

$$\int_{x=0}^{\widehat{t}} f(x) = \widehat{t}\left(a_0 + \widehat{t}\left(\frac{a_1}{2} + \widehat{t}\left(\frac{a_2}{3} + \widehat{t}\left(\frac{a_3}{4}\right)\right)\right)\right), \tag{3.9}$$

and evaluating (3.7) at $\widehat{t}$ yields

$$f(\widehat{t}) = a_0 + \widehat{t}\left(a_1 + \widehat{t}\left(a_2 + \widehat{t}(a_3)\right)\right). \tag{3.10}$$

Then evaluating (3.8) gives

$$f'(\widehat{t}) = a_1 + \widehat{t}\left(2a_2 + \widehat{t}(3a_3)\right). \tag{3.11}$$

If we introduce $f(0)$, $f'(0)$, $f(1)$ and $f'(1)$ as known, we can solve the system according to $\{a_i\}_{i=0}^{3}$. If we, in addition, also include $a_4$ (the integral on the whole interval, i.e. $\int_0^1 f(x)dx$), the solution is

$$\begin{aligned} a_0 &= f(0), \\ a_1 &= f'(0), \\ a_2 &= 3\left(f(1) - f(0)\right) - f'(1) - 2f'(0), \\ a_3 &= -2\left(f(1) - f(0)\right) + f'(1) + f'(0), \\ a_4 &= \tfrac{f(0)+f(1)}{2} + \tfrac{f'(0)-f'(1)}{12}. \end{aligned} \tag{3.12}$$

The question now is: how do we use these four expressions (3.9), (3.10), (3.11) and (3.12) in the evaluator? Recall that scaling of the domain influences the derivatives and the antiderivatives (integrals). Suppose the real interval is $\Delta t$ (not 1). Because of the general rule about scaling of the domain the derivatives / antiderivatives must be scaled / inversely scaled respectively, and do to definition 2.7 (also discussed in [20]), we have to do the following three adjustments:

- The input derivatives $f'(0)$ and $f'(1)$ have both to be scaled by $\Delta t$,
- The output integral, $B(t)$, has to be scaled by $\Delta t$.
- The output derivative (actual second derivative $D^2B(t)$) has to be inversely scaled by $\Delta t$.

To complete the description of the "ERBS-evaluator" there are still three algorithms that have to be described. The first algorithm is initialization.

**Algorithm 7.** *(For notation, see section "Algorithmic Language", page 7.)*
*The algorithm computes the internal state variables* $\Delta t$, $S = S(\alpha,\beta,\gamma,\lambda)$, $\mathbf{b} = \{B(t_i)\}_{i=0}^{m}$ *and the matrix* $\mathbf{a}$ *(by using an interpolating function). The algorithm assumes that there are algorithms present to compute* $\phi(t)$, $\phi'(t)$ *(can use* $f_2(t)$ *if* $S = 1$*) and* $\int_{s=0}^{t}\phi(s)ds$, $0 \leq t \leq 1$. *The input variables are: The intrinsic parameters* $\alpha$, $\beta$, $\gamma$, $\lambda$, *and* $m$ *(the number of sample intervals). There are no return values.*

```
void initiate ( double α, double β, double γ, double λ, int m )
    double φ0, φ1;                          // Value at the start and end of each interval
    double f0, f1;                          // Derivative at start and end of each interval
    set(α,β,γ,λ,m);                         // Store intrinsic parameters and m in object
    Allocate memory for b, a;               // Set size = m+1 for b, and m × 4 for a
    Δt = 1/m;                               // Set interval size
    S = 1;                                  // Temporary, to use φ'(t) = f2(t)
    b0 = φ0 = f0 = 0.0;                     // Defined to be zero (basic properties)
    for ( int i=1; i < m; i++ )             // For each sampling interval
        double t = i * Δt;                  // The sampling vector marked t_i in definition
        φ1 = φ(t);                          // Algorithm 1
        f1 = φ1 f2(t);                      // Algorithm 3
        bi = integrate(t − Δt, t, φ0, φ1, 1 × 10^−16);  // Algorithm 6 (using max tolerance)
        interpolate(i − 1, φ0, φ1, Δt f0, Δt f1); // Updating Hermite coefficients {a_{i−1,k}}_{k=0}^3
        φ0 = φ1;                            // Preparing for the next step
        f0 = f1;
    bm = integrate(Δt(m − 1), 1, φ0, 0, 1 × 10^−16); // Algorithm 6 (using max tolerance)
    interpolate(m − 1, φ0, 0, Δt f0, 0);    // Updating Hermite coefficients {a_{m−1,k}}_{k=0}^3
    for ( int i=1; i ≤ m; i≪=1 )            // These three following lines are introduced
        for ( int j=m; j ≥ i; j− = 1 )      // because there in line 13/17 is not used + =
            b_j + = b_{j−i};                // See below for explanation
    S = 1/b_m;
```

In line 13 and 17 **b** += *integrate*$(\ldots)$, incremental adding, should have been used. But then there will have been loss of at least one significant bit because of adding a small number to a bigger (and growing) number. To avoid this, the next three last lines are summing up in a binary way. The algorithm is summing neighbors that are nearly equal. The algorithm is, therefore, constructed to optimize the precision. In the algorithm one can see that both integrate() and interpolate() are called twice. The reason for this, is to avoid calling $\phi(t)$ and $f_2(t)$ for $t = 0$ and $t = 1$, because then they are defined to be 0. In addition, let us recall the scaling of the derivatives: they are all, according to the scaling

rules, scaled by $\Delta t$ in the input of the interpolate() function in lines 14 and 18.

The second algorithm, the interpolation, is a very simple algorithm implementing computation of the Hermite interpolation coefficients (see (3.12)).

**Algorithm 8.** *(For notation, see section "Algorithmic Language", page 7.)*
*The algorithm computes the Hermite interpolation coefficients* $\mathbf{a}_{i,0}$, $\mathbf{a}_{i,1}$, $\mathbf{a}_{i,2}$, $\mathbf{a}_{i,3}$ *and* $\mathbf{a}_{i,4}$, *and is an implementation of (3.12).*

void *interpolate (* int *i,* double $f_0$, double $f_1$, double $f'_0$, double $f'_1$ *)*
$\mathbf{a}_{i,0} = f_0$;
$\mathbf{a}_{i,1} = f'_0$;
$\mathbf{a}_{i,2} = 3(f_1 - f_0) - f'_1 - 2f'_0$;
$\mathbf{a}_{i,3} = -2(f_1 - f_0) + f'_1 + f'_0$;
$\mathbf{a}_{i,4} = \frac{f(0)+f(1)}{2} + \frac{f'(0)-f'(1)}{12}$;

The third and last function is the main "evaluation function" B(), analogous to Algorithm 5. The knot vector is generally not a part of the object, because the object is then associated with a curve/surface. The knot vector is, however, necessary for computing the complete values. The solution is to give a temporary reference to the knot vector before the computation.

**Algorithm 9.** *(For notation, see section "Algorithmic Language", page 7.)*
*The algorithm computes* $D^j B(t), j = 0,1,2$ *for the set of intrinsic parameters that are initialized in the object. The algorithm assumes that initializing is done. The knot vector* $\{t_i\}_{i=0}^{n+1}$ *is also supposed to be present. The input variables are:* $t \in [t_1, t_n]$, $k | t_k < t \le t_{k+1}$, *and* $d \in \{0,1,2\}$ *(the number of derivatives to compute). The algorithm depends on k being consistent with respect to t and the knot vector. The return is a* "vector⟨double⟩"*, where the first element contains* $B_k(t)$*, and then ,optional by,* $DB_k(t)$ *and* $D^2B_k(t)$.

vector⟨double⟩ *B (* double *t,* int *k,* int *d )*
$t = \frac{t-t_k}{t_{k+1}-t_k}$; *// Mapping from* $[t_1, t_n]$ *to the local domain* $[0,1]$
vector⟨double⟩ $R(d+1) = S$; *// Make a vector for return, size d+1, all elements S*
int $j = \min(\text{int}(t*m), m-1)$; *// j not equal m, due to the "mirroring" around 0.5*
double $dt = \frac{t-j*\Delta t}{\Delta t}$; *// Mapping from total* $[0,1]$ *to* $[0,1]$ *on sample*
switch *(d)*
case *2:* $R_2 * = \frac{\mathbf{a}_{j,1} + dt\left(2\mathbf{a}_{j,2} + dt\ 3\mathbf{a}_{j,3}\right)}{\Delta t}$; *// Using (3.11), scaled by* $\frac{1}{\Delta t}$
case *1:* $R_1 * = \mathbf{a}_{j,0} + dt\left(\mathbf{a}_{j,1} + dt\left(\mathbf{a}_{j,2} + dt\ \mathbf{a}_{j,3}\right)\right)$; *// Using (3.10), no scaling*
if $(dt > 0.5)$ *// Integrating: dt - 1 (3.12)*
$R_0 * = b_{j+1} - \Delta t\left(\mathbf{a}_{j,4} - dt\left(\mathbf{a}_{j,0} + dt\left(\frac{\mathbf{a}_{j,1}}{2} + dt\left(\frac{\mathbf{a}_{j,2}}{3} + dt\frac{\mathbf{a}_{j,3}}{4}\right)\right)\right)\right)$; *// Scaled by* $\Delta t$
else *// Integrating: 1 - dt (3.12)*
$R_0 * = b_j + \Delta t\ dt\left(\mathbf{a}_{j,0} + dt\left(\frac{\mathbf{a}_{j,1}}{2} + dt\left(\frac{\mathbf{a}_{j,2}}{3} + dt\frac{\mathbf{a}_{j,3}}{4}\right)\right)\right)$; *// Scaled by* $\Delta t$
for *(* int *i=1; i* $\le d$*; i++ )* $R_i\ /= (t_{k+1} - t_k)^i$; *// Due to (2.32)*
return *R;*

The next question is the "evaluation" of the "ERBS-evaluator" itself. There are two different evaluations to be done: evaluation of the efficiency, and evaluation of the precision.

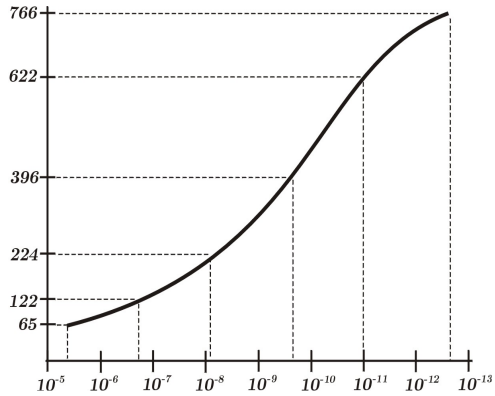


***Figure 3.3:*** *The relationship between the precision (tolerance) and the speed of the evaluation compared between the use of the B() function in the "ERBE-evaluator" and the use of the B() function in Algorithm 2 from section 3.2. For a sample rate of 1024, which gives a precision of 3.3e-13, the "ERBE-evaluator" is 766 times faster than the old one (computing function value and two derivatives).*

The efficiency is clearly the reason for making the whole system. Due to the Hermite interpolation and the integration, with regarding to time consumption, the whole system is identical to evaluating a 4th degree polynomial function, including the derivatives. This is in some sense a kind of optimal solution. In Figure 3.3 there is a graph of the relationship between the use of Algorithm 2 (with two derivatives) and Algorithm 9, the "ERBS-evaluator". Algorithm 2 uses the default set of intrinsic parameters, and is, therefore, relatively fast. The speed of the B() in the "ERBS-evaluator" is itself independent of the sample rate, but the precision is highly dependent on the sample rate. The evaluator B() in Algorithm 2 is very dependent on the precision, especially the integration part using Romberg integration. This can clearly be seen in Figure 3.3. On the horizontal axis you can see the precision (tolerance) of the function value, $B(t)$. On the vertical axis you can see the relation between the speed of the "ERBE-evaluator" B() and the "old" B(). One can clearly see that the difference in speed is tremendous, i.e. it takes up to 766 *times* longer to use B() in Algorithm 2 than B() in the "ERBS-evaluator". The figure also indicates that as we pass the tolerance $10^{-13}$ we approach the maximum of the possible acceleration of the algorithm by replacing with Algorithm 9.

The precision is a more complex problem to deal with, especially since the precision is getting worse with the increase of the order of derivatives. The reason for this is actually easy to see because $B(t)$ is the integral of $DB(t)$, and also because $DB(t)$ is the integral of $D^2B(t)$. Observe that a simplified computation of a maximal error of an integral of a function in a sample interval, is approximately: $\frac{2}{3} \times$ max error of the function $\times \frac{1}{m}$ (the sample interval). The difference in error between an approximation of a function and an approximation of the derivative of the function is, therefor, close to $10^3$ for number of samples m=1024. There are, of course, methods to improve this, but they might decrease the speed or the flexibility (this will be discussed further later).

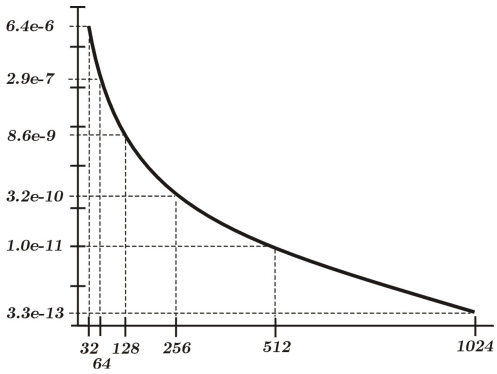


***Figure 3.4:*** *The relationship between the number of samples and the precision (max norm) of $B(t)$ (function value) using the "ERBS-evaluator". For a sample rate of 32, the precision is 6.4 e-6. For a sample rate of 1024, the precision is 3.3 e-13.*

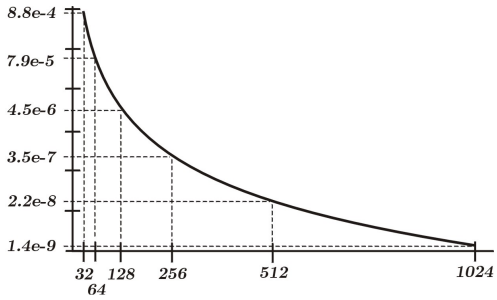


***Figure 3.5:*** *The relationship between the number of samples and the precision (max norm) of $DB(t)$ (first derivatives) using the "ERBS-evaluator". For a sample rate of 32, the precision is 8.8 e-4. For a sample rate of 1024, the precision is 1.4 e-9.*

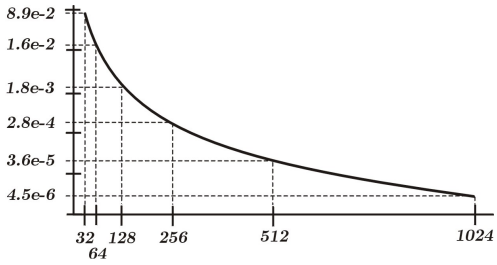


***Figure 3.6:*** *The relationship between the number of samples and the precision (max norm) of $D^2B(t)$ (second derivatives) using the "ERBS-evaluator". For a sample rate of 32, the precision is 8.9 e-2. For a sample rate of 1024, the precision is 4.5 e-6.*

| samples interval | | 1024 | 512 | 256 | 128 | 64 | 32 |
|---|---|---|---|---|---|---|---|
| $B(t)$ | $L^1[0,1]$ | $7.6e-15$ | $2.4e-13$ | $7.6e-12$ | $7.7e-10$ | $8.5e-9$ | $2.7e-7$ |
| | $L^2[0,1]$ | $2.9e-14$ | $9.5e-13$ | $3.0e-11$ | $9.6e-10$ | $3.3e-8$ | $9.7e-7$ |
| | $L^\infty[0,1]$ | $3.3e-13$ | $1.0e-11$ | $3.2e-10$ | $8.6e-9$ | $2.9e-7$ | $6.4e-6$ |
| $DB(t)$ | $L^1[0,1]$ | $4.9e-11$ | $7.8e-10$ | $1.3e-8$ | $2.0e-7$ | $3.5e-6$ | $5.6e-5$ |
| | $L^2[0,1]$ | $1.7e-10$ | $2.7e-9$ | $4.3e-8$ | $6.7e-7$ | $1.2e-5$ | $1.7e-4$ |
| | $L^\infty[0,1]$ | $1.4e-9$ | $2.2e-8$ | $3.5e-7$ | $4.5e-6$ | $7.9e-5$ | $8.8e-4$ |
| $D^2B(t)$ | $L^1[0,1]$ | $1.9e-7$ | $1.5e-6$ | $1.2e-5$ | $9.7e-5$ | $8.4e-4$ | $6.5e-3$ |
| | $L^2[0,1]$ | $5.9e-7$ | $4.7e-6$ | $3.8e-5$ | $2.9e-4$ | $2.6e-3$ | $1.9e-2$ |
| | $L^\infty[0,1]$ | $4.5e-6$ | $3.6e-5$ | $2.8e-4$ | $1.8e-3$ | $1.6e-2$ | $8.9e-2$ |

***Table 3.3:*** *The table shows the connection between the number of sample intervals and the error for the three functions $B(t)$, $DB(t)$ and $D^2B(t)$, using three different norms, $L^1[0,1]$, $L^2[0,1]$ and $L^\infty[0,1]$.*

Looking at the precision of the "ERBS-evaluator", Table 3.3 shows the connection between the number of sample intervals and the error for the ERBS basis function $B(t)$, and its derivatives $DB(t)$ and $D^2B(t)$, using three different norms. These norms are $L^1[0,1]$ representing an arithmetic mean value, $L^2[0,1]$ representing a geometric mean value, and $L^\infty[0,1]$, a max norm, giving us the guaranteed tolerance. The three figures 3.4, 3.5 and 3.6, show graphs of the maximum error of the "ERBS-evaluator" produced according to the number of samples. In Figure 3.4 the error for the function value $B(t)$, are plotted. It ranges from $6.4 \times 10^{-6}$ for 32 samples, to $3.3 \times 10^{-13}$ for 1024 samples. This can be regarded as quite a good result. In Figure 3.5 the error for the first derivative $DB(t)$ is plotted. It ranges from $8.8 \times 10^{-4}$ for 32 samples, to $1.4 \times 10^{-9}$ for 1024 samples. This can in many cases still be regarded as an acceptable result. In Figure 3.6 the errors for the second derivative $D^2B(t)$ are plotted. It ranges from $8.9 \times 10^{-2}$ for 32 samples, to $4.5 \times 10^{-6}$ for 1024 samples. This result is clearly on the edge of what is acceptable.

The conclusion is, therefore, that for the value $B(t)$, and the first derivative $DB(t)$ the tolerances are acceptable. But the tolerance for the second derivative can be improved by an enhancement. This will bring us to an implementation of B() in the "ERBS-evaluator" using parts of Algorithm 2 or Algorithm 5. All derivatives will then get the same error level as $DB(t)$. In this case it is obvious that we have to split the algorithm into two different functions/algorithms:

1. For the default set of intrinsic parameters,
   i) Remove the components of B() that treat the 2nd derivative.
   ii) Insert components of all elements treating $D^jB(t),\ j = 2,3,...,7$ from algorithm 2.

*The advantages:* This will not effect the speed of computing up to 2 derivatives at all, and we can in addition compute up to 7 derivatives.

*The disadvantages:* The flexibility is suffering because, if we want to change the intrinsic parameters, we have to change the function, or make a new object.

2. For the scalable set of intrinsic parameters,
   i) Remove the components of B() that treat the 2nd derivative.
   ii) Insert components of all elements treating $D^jB(t)$ for $j = 2,3$ from algorithm 5.

*The advantages:* The flexibility is kept, and we can in addition compute up to 3 derivatives.

*The disadvantages:* This will result in approximately twice the time needed to compute up to 2 derivatives compared to using only the "ERBS-evaluator".

---

Using C++ , the natural choice is to use inheritance and polymorphism:

| | | |
|---|---|---|
| The base class | "ERBS-evaluator" | clean Algorithm 9. |
| Inherited class 1 | "ERBS-default" | combine Algorithm 9 and 2. |
| Inherited class 2 | "ERBS-scalable" | combine Algorithm 9 and 5. |

In this case all three alternatives are available, and the flexibility is still present.

# Chapter 4

# Curves

The general formula of an Expo-Rational B-spline curve is,

$$f(t) = \sum_{i=1}^{n} c_i(t) B_i(t), \tag{4.1}$$

where $c_i(t),\ i = 1,...,n,$ are local curves and $B_i(t),\ i = 1,...,n$, are the ERBS basis functions. As can be seen, the formula resembles the usual polynomial B-spline curve formula, except that $c_i(t)$ are not points, but curves. The Expo-Rational B-splines can, therefore, be viewed as a blending of local curves. Figure 4.1 shows an example of a curve and its local blending curves. The curve has 4 local curves (green). The knot vector is $\{t_i\}_{i=0}^{5}$, where $t_0 = t_1$ and $t_4 = t_5$ are the multiple start and end knots, and where $t_2$ and $t_3$ are simple (non multiple) inner knots. Each basis function and, thus, local curve, spans a two knot interval, and the local curve interpolates the global curve at the middle knot of its span. This is the reason why the first local curve, spanning $[t_0, t_2]$, interpolates at the start of the global curve, because the middle knot of this interval, $t_1$ is at the start. This is, of course, also the reason why the last curve interpolates at the end of the global curve. The curve in Figure 4.1 can be divided into three parts, and the "dividing points" are the

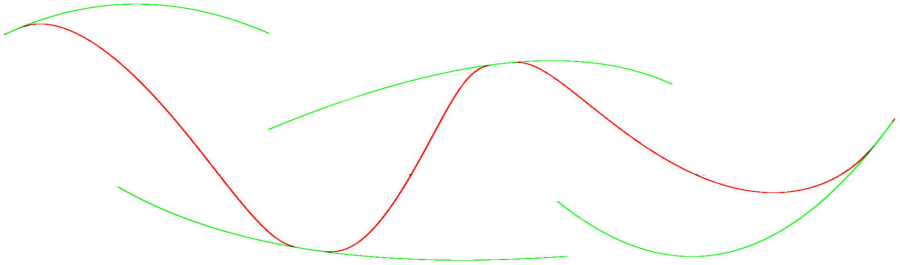

***Figure 4.1:*** *A (global) Expo-Rational B-spline curve (red) with four local curves (green). The global curve is a blending of its local curves, with the Expo-Rational B-splines being the blending functions. The global curve also completely interpolates all existing derivatives of each of the adjacent local curves at the "middle" knot.*

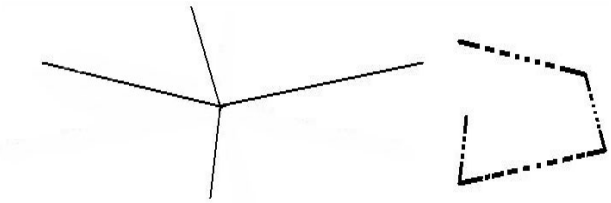

***Figure 4.2:*** *The local curves are replaced by points. The complete curve is on the right hand side. The curve is drawn with points to illustrate the "speed" (denser is slower). The star, on the left hand side, is the plot of the derivative centered at the origin.*

ones where the local curves touch the global curve, i.e. the inner knots $t_2$ and $t_3$. Each part of the curve is a blending of parts of two local curves. The first part is a blending of the whole first local curve and the first half (until knot $t_2$) of the second local curve. The second part is a blending of the second part (from knot $t_2$) of the second local curve and the first part (until knot $t_3$) of the third local curve. The third part is a blending of the second part (from knot $t_3$) of the third local curve and the whole fourth local curve. This shows the extreme local support, i.e. if we change the first local curve, only the first third of the global curve will be changed.

Before we investigate different types of local curves, a comment is required on what it will look like using points as coefficients, as in ordinary B-splines.

**Remark 9.** *If we replace the local curves with points, the complete curve will geometrically be a piecewise linear curve with an infinitely smooth parametrization (on a strictly increasing knot vector) and $D^j B_k(t_i) = 0$ (basic property 6, see Theorem 2.1) implies that all derivatives of $f$ must be zero at every knot: $D^j f(t_i) = 0,\ j = 1,2,...,\ i = 1,...,n$ (this is illustrated on the right hand side in Figure 4.2). It follows that all higher derivatives (vectors) must be parallel to the curve (line) and, thus, to the first derivative. Thus, every derivative of $f$ will geometrically form a 'star' concentrated at the origin (see the left hand side of Figure 4.2).*

In definition 2.5 the classification of local function to support the basic properties, i.e. the $C^\infty$ property, is given. For a vector-valued function (parametrized curve), the constraints on the local function can be summed up in the following remark.

**Remark 10.** *In principle, the only constraint on the choice of the local curve $c_i(t)$ is that $|c_i(t)| \in \mathfrak{F}(B_i)$ (definition 2.5).*

Important instances of admissible local curves are curves based on algebraic and trigonometric polynomials, rational functions, circular arcs, etc. It is possible to use curves of different types as local functions, and it is of particular interest to consider "multilevel" Expo-Rational B-splines. In Figure 4.3, there is an example where two parameterized circles are local curves. The example shows that two closed curves can act as local curves for an open curve. The definition and structures of open/closed curves are perhaps not obvious, and will, therefore, be discussed in the first of the following sections.

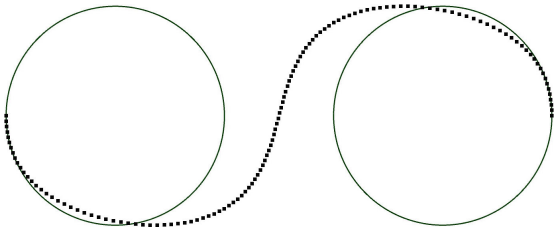

***Figure 4.3:*** *Two circles as local curves. The complete curve is the dotted curve. At the ends, the global curve interpolates the local curves with the derivatives of every order.*

In the remainder of this chapter the following subjects will be discussed. In section 4.2, the ERBS curve evaluation, including derivatives, will be discussed. Then, in section 4.3 Bézier curves as local curves will be investigated, including Hermite interpolation, and numerous examples. In section 4.4 circular arcs as local curves will be discussed. In this case it is necessary to use a modified Hermite interpolation. In the following reparametrization using approximative curve length parametrization will be discussed, and lot of examples connected to this will be given, including the general circular arc's case. Then in the last section in this chapter affine transformation on local curves will be discussed. Examples of this will be shown although the examples really should have been animations. Also computational aspects concerning both Bézier curves and Arc curves will be discussed in the chapter.

## 4.1 Definition/implementation of "open/closed" curves

We regard an ERBS curve to be a "1-dimensional object", i.e., a parametrized differentiable curve, allowing self intersection, singularities and other irregularities. Although an ERBS curve almost always can be regarded as a differentiable manifold, we will nevertheless keep to the concept of non-manifold geometry.

Therefor, the definition of a curve is as following:

**Definition 4.1.** *A parametrized differentiable curve is a differentiable map* $\alpha : I \to R^n$ *of an open interval* $I = (s,e) \subset \mathbb{R}$ *into* $\mathbb{R}^n$ *for* $n = 2,3,\ldots$ .

With this as a background the next definitions are introduced:

**Definition 4.2.** *A "standard", "open" and "closed" ERBS curve is defined as:*

*i) A "standard" ERBS curve is defined on a half open interval* $(a,b]$*, which is a restriction of a differentiable function on an open interval containing the half open interval* $(a,b]$.

*ii) By exclusively adding* $B_1(t_1) = 1$ *to definition 2.2, an "open" ERBS curve* $\alpha$ *can be defined on a closed interval* $[a,b]$*, which is a restriction of a differentiable function on an open interval containing the closed interval* $[a,b]$.

*iii) A "closed" ERBS curve* $\alpha$ *is apparently also defined on the half open interval* $(a,b]$.

*But because a and b are defined to be identical, and* $lim_{t\to a+}D^j\alpha(t) = D^j\alpha(b)$, $j = 0,1,2,\ldots$ *it, therefore, follows that the domain is actually open.*

A "standard" ERBS curve is defined following the definition for an ERBS function given in definition 2.6, and is very suitable for spliced curves. An ERBS curve with multiple knots at the start and end, and defined to be an "open" curve, is defined on a closed domain because the multiple knots are actually closing the interval/domain. A circle, an ellipse are examples of "closed" curves (which also are called cyclic). The only special feature of "closed" curves is that the open domain implies that there is no start and end.

An ERBS curve is, in addition to the intrinsic parameters, determined by a knot vector, defining the basis functions, and the local curves. Since a basis function is defined over two knot intervals, and there must be at least two basis functions to fulfill partition of unity, the minimum number of knots for a "standard" and an "open" ERBS curve must be 4. In practical implementations, local curves are usually implemented as objects (C++ instance of a class), and there are references (or pointers) to these objects. In general a "standard/open" ERBS curve should have the following relationship between the number of knots, local curves, and also references to local curves, and the indexing. The number of ERBS basis functions and thus local curves are denoted $n$ in the following.

| | | |
|---|---|---|
| Minimum number of local curves is: | $n = 2$ | "standard/open" ERBS curves |
| References to basis-curves/local-functions: | $n$ | indexing from: 1 to $n$ |
| Number of knots: | $n+2$ | indexing from: 0 to $n+1$ |

Using this convention, the indexing of the knots and the functions is actually connected and makes implementation easy.

For a "closed" ERBS curve, it is basically only necessary to have the same number of knots as the number of knot intervals and the number of basis functions/local curves. But practically, to describe $n$ knot intervals we usually need at least $n+1$ knots. A suggestion for a practical set is to increase both the number of knots and basis functions/local curves. The suggestion is not to increase the actual number of local curves, but to introduce an extra reference. The result is that we get the following numbers of references to local curves, and numbers of knots, for "closed" ERBS curves. Note that this time the number of ERBS basis functions and, thus, **references** to local functions, is denoted $n$.

| | | |
|---|---|---|
| Minimum number of real local curves is: | $n-1 = 1$ | for "closed" ERBS curves |
| References to basis-functions/local-curves: | $n$ | indexing from: 1 to $n$ |
| Number of knots: | $n+2$ | indexing from: 0 to $n+1$ |

Note that even if the main cycle of the domain is $(t_1,t_n]$, the real number of local curves is $n-1$. There are two invariants connected to this implementation of "closed" curves:

i) The knot intervals between $\{t_0,t_1,t_2\}$ are just a reflection of the knot intervals between $\{t_{n-1},t_n,t_{n+1}\}$, i.e. $t_0 = t_1 - (t_n - t_{n-1})$ and $t_{n+1} = t_n + (t_2 - t_1)$ . It is an invariant that the two first and the two last knot intervals have to be equal, and that this must be kept up if a knot value is changed.

ii) For the local curves, the reference with index $n$ is the same as the reference with index 1. This is to be seen as an invariant: if one of the references is changed, then

the other has to be changed in the same way.

For a "closed" ERBS curve, the basis functions just have to be computed in the same way as for open curves, with the same indices as for the references to the local curves. As one can see from the table, it is actually possible to use only 1 local curve because we then only get 1 knot interval, and the first half of the local curves is blended with the second half. The reason for using multiple representations is that, in the global/local mapping at both ends, we will need the access of the two knot intervals, and in the evaluator (see the next sections) we will need the access of the two local curves in the computations. By introducing this multiplicity in the structure, there will be no need for testing and special computations at the start and end of the main cycle of the domain.

## 4.2 Evaluation, value and derivatives

The general equation for an ERBS curve (4.1) and its derivatives are straightforward to compute,

$$\begin{aligned}
f(t) &= \sum_{i=1}^{n} c_i(t)B_i(t), \\
Df(t) &= \sum_{i=1}^{n} \left(Dc_i(t)B_i(t) + c_i(t)DB_i(t)\right), \\
D^2f(t) &= \sum_{i=1}^{n} \left(D^2c_i(t)B_i(t) + 2Dc_i(t)DB_i(t) + c_i(t)D^2B_i(t)\right), \\
D^3f(t) &= \sum_{i=1}^{n} \left(D^3c_i(t)B_i(t) + 3D^2c_i(t)DB_i(t) + 3Dc_i(t)D^2B_i(t) + c_i(t)D^3B_i(t)\right).
\end{aligned} \tag{4.2}$$

The equation shows that the constants resemble "Pascals triangle" as expected, and that the equations in a way have the same template as a general sum of the Bernstein polynomials in Bézier curves, where the polynomial degrees are substituted by derivational order.

There are, however, possibilities for simplifications. Remember that there are only two basis functions different from zero in the interior of a knot interval, and that they sum up to 1. At a knot value there is actually only 1 basis function different from zero. Analogical to equation (2.48) we can simplify the first part of (4.2) to

$$f(t) = \begin{cases} c_k(t), & \text{if } \ t = t_k \\ c_{k+1}(t) + (c_k(t) - c_{k+1}(t))\,B_k(t), & \text{if } \ t_k < t < t_{k+1} \end{cases} \tag{4.3}$$

Computing the derivatives of (4.3) we can see that at $t = t_k,\ k = 1,...,n$, all derivatives of the ERBS curve are equal to the respective derivatives of the local curve, i.e.

$$D^j f(t_k) = D^j c_k(t_k), \quad \text{for} \quad k = 1,...,n \quad \text{and} \quad j = 0,1,2,... \tag{4.4}$$

To simplify the equations for all other $t$ values in $[t_1, t_{n+1}]$, we first define

$$\widehat{c}_k(t) = c_k(t) - c_{k+1}(t), \ \text{ if } \ t_k < t < t_{k+1}, \tag{4.5}$$

then for $t_k < t < t_{k+1}$ we get the following equation for the function value and the derivatives

$$\begin{aligned} f(t) &= c_{k+1}(t) + \widehat{c}_k(t)B_k(t), \\ Df(t) &= Dc_{k+1}(t) + \widehat{c}_k(t)DB_k(t) + D\widehat{c}_k(t)B_k(t), \\ D^2f(t) &= D^2c_{k+1}(t) + \widehat{c}_k(t)D^2B_k(t) + 2D\widehat{c}_k(t)DB_k(t) + D^2\widehat{c}_k(t)B_k(t), \\ D^3f(t) &= D^3c_{k+1}(t) + \widehat{c}_k(t)D^3B_k(t) + 3D\widehat{c}_k(t)D^2B_k(t) + 3D^2\widehat{c}_k(t)DB_k(t) + D^3\widehat{c}_k(t)B_k(t) \end{aligned} \tag{4.6}$$

These expressions are similar to (4.2), but there is an extra term $D^j c_{k+1}(t)$ for $j = 0,1,2,3,...$, and there is $\widehat{c}_k(t)$ instead of $c_k(t)$. The biggest difference is, that the sum is removed and there is only a single line to compute for each of the derivatives.

There is, however, one important fact to be aware of. The algorithms made for computing the function value $B_k(t)$ and the derivatives $D^jB_k(t),\ j = 1,2,...$, i.e. algorithm 2, algorithm 5 and algorithm 9, are actually computing the values for the next basis function $B_{k+1}(t)$ on the current interval $[t_k, t_{k+1}]$; this means that they are computing the left side of the basis functions. Taking this into consideration, we have to decide whether to use an "overloaded ERBS evaluator", to adjust the evaluators to treat the right side of the basis functions $B_k(t)$.

**Remark 11.** *A postprocessing of the ERBS evaluator cannot just shift the index value (k), because algorithm 2, algorithm 5 and algorithm 9 can only compute the basis function $B_{k+1}(t)$ on the interval $[t_k, t_{k+1}]$. Therefore, another "shift" has to be taken into consideration, and executed in the following way:*

*i)* $B_k(t) = 1 - B_{k+1}(t)$ *for* $t_k < t < t_{k+1}$.
*ii)* $D^jB_k(t) = -D^jB_{k+1}(t)$ *for* $t_k < t < t_{k+1}$ *and* $j = 1,2,....$

*Both i) and ii) follow directly from property 2 in Theorem 2.1.*

We, therefore, introduce the following postprocessing "overloaded" algorithm for ERBS evaluation to be used in a curve evaluator.

**Algorithm 10.** *(For notation, see section "Algorithmic Language", page 7.)*
*The algorithm computes $D^jB_k(t)$, for $j = 0,1,...,d$. It is assumed that either algorithm 2, algorithm 5 or algorithm 9 is used as the basic algorithm. The input variables are: $t \in [t_1, t_n]$, $k | t_k < t \le t_{k+1}$, and $d \in \{0,1,2,...\}$ (the number of derivatives to compute). The algorithm depends on k being consistent according to t and the knot vector. The return is a* "vector⟨double⟩"*, where the first element contains $B_k(t)$ , and then $D^1B_k(t)$ and where the last element is $D^dB_k(t)$.*

```
vector⟨double⟩ B̃ ( double t, int k, int d )
    vector⟨double⟩ B̃ = B(t,k,d);          // Result evaluating ERBS-basis, (Alg. 9).
    B̃_0 = 1 − B̃_0;                       // Remark 11, point i.
    for ( int j=1; j ≤ d; j++ )
        B̃_j = −B̃_j;                      // Remark 11, point ii.
    return B̃;
```

To make the finite algorithm for an ERBS curve evaluator we have to recall the equations in (4.3–4.6). We first notice the special case where $t$ is equal to a knot value. We then notice that, for the elements in each line, the order of the derivative of $\widehat{c}_k(t)$ is "turned" compared with the order of the derivative of $B_k(t)$. In the computation of (4.6), we, therefore, must turn the vector from the evaluator of the basis function, algorithm 10. We also have to insert "Pascals triangle numbers" $a_{d,j} = \binom{d}{j}$, where $d$ is the line number and j is the element number. Following this, the algorithm will be simple to implement, and reliable. The reliability, however, will not only depend on a reliable ERBS evaluator (algorithm 10), but also on reliable evaluators for the local curves. Evaluators for Bézier curves and Arc curves will be discussed later in this chapter.

We assume that the curves are embedded in an Euclidian space, but where the dimension might differ. So to make it more general, the vector type is denoted T (template type in C++ ), usually T will be vector⟨double⟩(3), a vector with dimension 3. The global/local affine mapping, definition 2.7, is not considered in the following algorithm, but it must be inserted in the local curve evaluators.

**Algorithm 11.** *(For notation, see section "Algorithmic Language", page 7.)*
*The algorithm computes $\{D^j f(t)\}_{j=0}^d$ for an ERBS curve. The algorithm assumes that evaluators for the local curves and the ERBS basis function are present. The knot vector $\{t_i\}_{i=0}^{n+1}$ is also supposed to be present. The input variables are: $t \in [t_1, t_n]$ and $d \in \{0,1,2,...,p\}$ (the number of derivatives to compute, where p depends on the ERBS evaluator). The return is a* "vector⟨T⟩"*, where T is a n-dimensional vector matching $f(t)$, and where the first element contains $f(t)$, and then $Df(t),...,D^d f(t)$.*

vector⟨T⟩ *eval (* double *t,* int *d )*
int $k = k :\backslash;\ t_k \le t < t_{k+1}$; // *Index for the current knot-interval.*
vector⟨T⟩ $c_0 = \{D^j c_k(t)\}_{j=0}^d$; // *Result evaluating local curve - index k.*
if $(t == t_k)$ return $c_0$; // *Return only local curve - k, see (4.3).*
vector⟨T⟩ $c_1 = \{D^j c_{k+1}(t)\}_{j=0}^d$; // *Result evaluating local curve - k+1.*
vector⟨double⟩ $a(d+1)$; // *Vector to store "Pascals triangles nr".*
vector⟨double⟩ $B = \overline{B}(t,k,d)$; // *Result evaluating ERBS-basis, (Alg. 10).*
$c_0 -= c_1$; // *$c_0$ is now $\widehat{c}_0$, see (4.4).*
for *(* int *i=0; $i \le d$; i++ )*
$a_i = 1$;
for *(* int *j=i-1; $j > 0$; $j--$ )*
$a_j += a_{j-1}$; // *Computing "Pascals triangle"-numbers.*
for *(* int *j=0; $j \le i$; j++ )*
$c_{1,i} += (a_j B_j) c_{0,i-j}$; // *Computing (4.6), "T += scalar*T".*
return $c_1$;

The computational cost of computing the function value and d derivatives is (computing the value and d derivatives for 1 ERBS basis function $B_i(t)$ and 2 local curves, $c_i(t)$ and $c_{i+1}(t)$) a total of $3(d+1)$ values and derivatives, and $\approx d^2$ extra multiplications.

**Remark 12.** *In total the computational cost is less than 3 times the "low degree" Bézier Curve, but the design possibilities and properties are much better than a higher degree*

*Bézier/B-spline which has a computational cost that is higher then the ERBS curve.*

## 4.3 Bézier curves as local curves

In general, Bézier curves are very convenient to use as local curves. Recall that Bézier curves are defined by

$$c(t) = \sum_{i=0}^{d} c_i \, b_{d,i}(t), \quad \text{if } \; 0 \le t \le 1, \tag{4.7}$$

where the basis functions are the Bernstein polynomials

$$b_{d,i}(t) = \binom{d}{i} t^i \, (1-t)^{d-i},$$

and where $c_i \in \mathbb{R}^n, \; i = 0, 1, ...d$, are the coefficients and, thus, the control polygon, d is the polynomial degree and where n usually is 2 or 3 (but can be any positive integer).

There are three different types of evaluators (computations of (4.7) used for Bézier curves,

i) de Casteljau algorithm,
ii) a Cox/de'Boor type algorithm,
iii) computing the Bernstein polynomials directly.

Hard-coded Bernstein polynomial gives the fastest algorithm for specific degrees, but for general degrees a Cox/de Boor version of an algorithm is the most flexible and also quite a fast algorithm. We will not, however, be focused on a general evaluator for Bézier curves, because this is well known and discussed in many places. We shall, on the other hand, look in sufficient detail at a specific evaluator for use in both preevaluations and in the Hermite interpolation for generating local curves, when the local curves are Bézier curves.

The Generalized Bernstein/Hermite matrix will be introduced (interpolation matrices are discussed amongst others in [48]) for use in both Hermite interpolation and in preevaluation (both will be discussed later)

$$\mathbf{B}_d(t,\delta) = \begin{pmatrix} b_{d,0}(t) & b_{d,1}(t) & \cdots & b_{d,d}(t) \\ \delta \, D b_{d,0}(t) & \delta \, D b_{d,1}(t) & \cdots & \delta \, D b_{d,d}(t) \\ \vdots & \vdots & \ddots & \vdots \\ \delta^d D^d b_{d,0}(t) & \delta^d D^d b_{d,1}(t) & \cdots & \delta^d D^d b_{d,d}(t) \end{pmatrix}. \tag{4.8}$$

The special issue about this matrix (4.8) is the scaling $\delta^j$, where j, the power exponent, is the row number (the first row is numbered 0). The reason for this scaling will be explained later. However, in general Hermite interpolation $\delta = 1$. in the following, we shall introduce an algorithm to compute this generalized version of the matrix. Later, we shall see how to use this matrix.

First we have to look at another matrix; $\mathbf{T}(t) : \mathbb{R}^k \to \mathbb{R}^{k-1}$, defining the de Casteljau algorithm. This matrix is a band-limited matrix with bandwidth two, and with the elements

$1-t$ and $t$ on the nonzero band. We can now look at a matrix version of a Bézier curve. In the following equations we will see a 3rd degree Bézier curve and its three derivatives in matrix form,

$$\begin{aligned} c(t) &= T_1(t)T_2(t)T_3(t)\,C, \\ c'(t) &= 3\,T_1(t)T_2(t)\,T_3'\,C, \\ c''(t) &= 6\,T_1(t)\,T_2'T_3'\,C, \\ c'''(t) &= 6\,T_1'T_2'T_3'\,C, \end{aligned} \tag{4.9}$$

where the indices denotes the number of rows in the matrix. The derivative of the matrix $T(t)$, denoted $T'$ is a matrix independent of $t$, a band-limited matrix with bandwidth two, and with the elements $-1$ and 1 on the band. If we expand (4.9), we get the following equations,

$$\begin{aligned} c(t) &= \begin{pmatrix} 1-t & t \end{pmatrix} \begin{pmatrix} 1-t & t & 0 \\ 0 & 1-t & t \end{pmatrix} \begin{pmatrix} 1-t & t & 0 & 0 \\ 0 & 1-t & t & 0 \\ 0 & 0 & 1-t & t \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{pmatrix}, \\ c'(t) &= 3\begin{pmatrix} 1-t & t \end{pmatrix} \begin{pmatrix} 1-t & t & 0 \\ 0 & 1-t & t \end{pmatrix} \begin{pmatrix} -1 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & -1 & 1 \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{pmatrix}, \\ c''(t) &= 6\begin{pmatrix} 1-t & t \end{pmatrix} \begin{pmatrix} -1 & 1 & 0 \\ 0 & -1 & 1 \end{pmatrix} \begin{pmatrix} -1 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & -1 & 1 \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{pmatrix}, \\ c'''(t) &= 6\begin{pmatrix} -1 & 1 \end{pmatrix} \begin{pmatrix} -1 & 1 & 0 \\ 0 & -1 & 1 \end{pmatrix} \begin{pmatrix} -1 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & -1 & 1 \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{pmatrix}. \end{aligned} \tag{4.10}$$

Now we can clearly see that:

i) If we compute from the right hand side (skipping the zeros), we get the de Casteljau algorithm.

ii) If we compute from the left hand side, we get an algorithm of Cox/de Boor type.

iii) If we multiply the matrices from the left hand side, without the coefficient vector on the right hand side, we will get the Bernstein polynomials and their derivatives. [1]

In section 2.8, the Hermite interpolation properties are discussed. (On page 63 there is also a comment on Hermite interpolation and derivatives). If the domain of the Bézier curve is scaled, as is the norm, because of the global/local affine mapping (see (2.49) in definition 2.7), then in order to compute, for instance, the local Bézier curve $c_i(t)$, the $j$th

[1] As a digression, one can use the same matrix notation on B-splines. This easily gives us both the Cox/de Boor algorithm, a geometric de Casteljau version for B-splines, and if we multiply the matrices, a fast expanded version of an evaluator. Matrix notation on B-splines is discussed in [41].

derivatives actually have to be scaled by the global/local "scaling factor" $\delta_i^j$ where

$$\delta_i = \frac{1}{t_{i+1} - t_{i-1}}, \tag{4.11}$$

as described in Theorem 2.4. The numerator in the fraction is 1 because the domain of Bézier curves is $[0,1]$. Because the matrix (4.8) is supposed to be used both in Hermite interpolation and in the evaluation of local curves, this matrix has to include the scaling, as described earlier. It now follows that we get the following algorithm to make the matrix $\mathbf{B}_d(t,\delta)$ described in (4.8).

**Algorithm 12.** *(For notation, see section "Algorithmic Language", page 7.)*
*The algorithm computes the extended square matrix $\mathbf{B}_d(t,\delta) \in \mathbb{R}^{d+1\times d+1}$, contained in the first row, the values of the $d+1$ Bernstein polynomials $\{b_{d,i}(t)\}_{i=0}^d$, and, in the following rows values for each of the d derivatives, $\{D^j b_{d,i}(t)\}_{i=0}^d, \quad j = 1,2,\ldots,d$, respectively, in each of the j following rows. In addition, all rows where the number is $j > 0$ (the rows are numbered from 0 to d), are multiplied by $\delta^j$. If the matrix is supposed to be used in a general evaluator, then $\delta = 1$, and if the matrix is supposed to be used in Hermite interpolation and evaluation of local curves, then we have to use $\delta_i$, see (4.11), where i is the index of the local curve. The input variables are: the degree d of the Bernstein polynomials, the parameter value $t \in [0,1]$, and the scaling factor $\delta$.*

```
Matrix⟨double⟩ mat ( int d, double t, double δ )
    Matrix⟨double⟩ B(d+1,d+1);        // The return matrix, dimension (d+1)×(d+1).
    B_{d-1,0} = 1 − t;
    B_{d-1,1} = t;                    // The general Cox/deBoor like algorithm for
    for ( int i=d-2; i ≥ 0; i−− )     // - Bézier/Bernstein computing the triangle
        B_{i,0} = (1−t) B_{i+1,0};    // - of all values from "Bernstein" polynomial
        for ( int j=1; j < d−i; j++ ) // - of degree 1 to d, respectively in each row.
            B_{i,j)} = t B_{i+1,j−1} + (1−t) B_{i+1,j};
        B_{i,d−i} = t B_{i+1,d−i−1};

    B_{d,0} = −δ;
    B_{d,1} = δ;                      // Multiply all rows except the upper one
    for ( int k=2; k ≤ d; k++ )       // - with the derivative matrices in the
        double s = k δ;               // - expression (4.10), and the scalings,
        for ( int i = d; i > d−k; i−− ) // - so every row extends the number
            B_{i,k} = s B_{i,k−1};    // - of elements to d.
            for ( int j = k−1; j > 0; j−− )
                B_{i,j} = s (B_{i,j−1} − B_{i,j}) ;
            B_{i,0} = −s B_{i,0};
    return B;
```

The order of this algorithm can clearly be seen to be an improved $O(n^3)$ , and the following table shows the number of multiplications depending on d, for d up to 10.

| $d$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| multiplications | 0 | 11 | 30 | 59 | 100 | 155 | 226 | 315 | 424 | 555 |

The speed of the algorithm is not assential because the algorithm is only supposed to be executed when the curves are made, or when the sampling is changed (preevaluation). But for small $d$ values ($d < 4$) the algorithm is fairly fast.

In the next three subsections we will see how to make local curves using Hermite-interpolation, how to use sampling and preevaluations in dynamically deforming curves and, finally, a lot of examples will be given.

### 4.3.1 Local Bézier curves and Hermite interpolation

We start by recalling the settings from section 2.8, and adapting them to ERBS curves.

- Given is a curve $g(t)$, $g : [t_s, t_e] \subset \mathbb{R} \to \mathbb{R}^n$, where we might have $n = 1, 2, 3, ...,$
- given is a number of samples $m > 1$, and the number of derivatives $\{d_i\}_{i=1}^m > 0$ in each of the sampling points, to be used in the interpolation.
- Generate a knot vector by:
  - first set $t_1 = t_s$,
  - then set $t_m = t_e$.
  - Then for $i = 2, 3, ..., m-1$ generate $t_i$ so that $t_{i-1} < t_i$, and where $t_{m-1} < t_m$.
  - Finally, $t_0$ and $t_{m+1}$ must be set according to the rules for "open/closed" curves.
- Make an ERBS curve using the knot vector $\{t_i\}_{i=0}^{m+1}$, and generate local curves, in such a way that the ERBS curve is interpolating $\{D^s g(t_i)\}_{s=0}^{d_i}$, for $i = 1, ..., m$.

This looks like an adjustment of a general Hermite interpolation method used for generating an approximation of a curve. What is specific is the generation of the local curves. First recall that the domain of a Bézier curve is $[0, 1]$. Then note from Theorem 2.4 that (adjusted with the local domain for Bézier curves)

$$D^j f(t_i) = \delta_i^j \, D^j c_i \circ \omega_i(t_i), \quad \text{for } j = 0, 1, 2, ... \text{ and } i = 1, .., m,$$

where $f(t)$ is the ERBS curve, $c_i(t)$ now are Bézier curves, and

$$\omega_i(t_i) = \frac{t_i - t_{i-1}}{t_{i+1} - t_{i-1}}$$

is the affine global/local mapping from definition 2.7, and

$$\delta_i = \frac{1}{t_{i+1} - t_{i-1}}, \quad \text{for } i = 1, 2, ..., m,$$

is the global/local scaling factor of the domain defined in Theorem 2.4. It shows that the ERBS curve is, adjusted by the domain scaling factor, interpolating the local curve $c_i(t)$ for all derivatives at the knot $t_i$ for $i = 1, ..., m$. We now get the equation for the Hermite interpolations for a local Bézier curve with index i,

$$D^s g(t_i) = D^s f(t_i) = \delta_i^s \, D^s c_i \circ \omega_i(t_i) = \delta_i^s \sum_{j=0}^{d_i} c_{i,j} D^s b_{d_i,j} \circ \omega_i(t_i), \quad \text{for } s = 0, ..., d_i$$

This can be formulated in a vector/matrix form,

$$\mathbf{B}_{d_i}(\omega_i(t_i),\delta_i)\ \mathbf{c}_i = \widehat{\mathbf{g}}_i \tag{4.12}$$

where $\mathbf{B}_{d_i}(\omega_i(t_i),\delta_i)$ is the Bernstein/Hermite matrix described in equation (4.8) and computed in algorithm 12, and where

$$\mathbf{c}_i = \begin{pmatrix} c_{i,0} \\ \vdots \\ c_{i,d} \end{pmatrix}$$

are the coefficients of the local Bézier curve with index $i$, and where

$$\widehat{\mathbf{g}}_i = \begin{pmatrix} D^0 g(t_i) \\ \vdots \\ D_i^d g(t_i) \end{pmatrix}$$

is a vector that is a typical result of an evaluator for a general parametrized curve.

The final step in the generation of the local Bézier curves is thus solving equation (4.12) according to the Bézier coefficients $\mathbf{c}_i$,

$$\mathbf{c}_i = \mathbf{B}_{d_i}(\omega_i(t_i),\delta_i)^{-1}\ \widehat{\mathbf{g}}_i. \tag{4.13}$$

The conclusion is that, in order to compute the coefficient to the local Bézier curves (4.13), one has to compute the expanded Bernstein/Hermite matrix using algorithm 12, and then invert this matrix, and multiply the inverted matrix with the "evaluation"-vector from the original curve. The matrix inversion will not be discussed further here, but there are many programming libraries available, including optimized algorithms for matrix inversions, see, e.g., [59].

For several reasons, it is advantageous to translate all coefficients after computing (4.13), so that the interpolation point is "in the local origin". It follows then that we have to subtract $g(t_i)$ from all the coefficients in the control polygon $\mathbf{c}_i$ of the local Bézier curve, and that we have to cancel this by inserting the opposite movement to the graphical homogeneous matrix system[2]. The premise is, of course, that this homogeneous matrix system is involved in the total evaluator. This is discussed further in section 4.5.

### 4.3.2 Sampling and preevaluation

Sampling, and piecewise linear approximation of a curve is the most common method used in graphic display. If the curve is dynamic deforming like a "moving" rope, then preevaluation of the basis functions is a very useful method for speeding up the computations.

Preevaluation can of course be used both on the the global ERBS curve and on the local curves. In principle there are two different ways of sampling.

[2]By "homogeneous matrix" we understand that the coordinates given in the matrix are homogeneous

i) Non uniform sampling based on a given tolerance ε, and where the curvature or constant linear deviation is used to minimize the number of samples.
ii) Uniform sampling based on a given number of samples, usually called $m$.

Uniform sampling is the natural choice for use in dynamic deformation, especially if preevaluation is an option. In the following, the uniform sampling including preevaluation, will be discussed further.

Given is a number of samples $m$, and an ERBS curve

$$f(t) = \sum_{i=1}^{n} c_i(t) B_i(t)$$

where $c_i(t),\ i = 1,2,...,n$ are local Bézier curves, and $\{t_i\}_{i=0}^{n+1}$ is the ERBS knot vector. To make a uniform sampling vector, we first define the uniform sample step based on a given number of samples $m$,

$$\Delta x = \frac{t_n - t_1}{m-1},$$

so that we can make a total sampling vector for the ERBS curve, based on the uniform sampling,

$$x_j = t_1 + j\Delta x, \quad \text{for} \quad j = 0,1,2,...,m-1. \tag{4.14}$$

To distribute this sampling vector (4.14) to the local curves, we have to define the index of the first sample position touching the local domain of a local curve

$$s_i = min\{j : x_j \geq t_{i-1},\}, \quad \text{for} \quad i = 1,2,...,n,$$

and the last sample position touching the local domain of a local curve

$$e_i = max\{j : x_j \leq t_{i+1},\}, \quad \text{for} \quad i = 1,2,...,n.$$

Now we have the following three parameters to distribute to each of the local curves for local sampling and preevaluations ($\omega_i$ defined in definition 2.7):

1) The number of local sample interval, $e_i - s_i$.
2) The local parameter value for the first sample, $\omega_i(x_{s_i})$.
3) The local parameter value for the last sample, $\omega_i(x_{e_i})$.

The purpose of introducing preevaluations is to implement dynamic shape varying. There are, of course, several possible levels in this change. The two most obvious ways of doing this are

i) affine transformations of local curves,
ii) affine transformations of individual coefficient points of local curves.

Taking this into consideration, there are two levels of possible local preevaluations.

i) In each local curve, store the values of the curve in each of the local sample points. These can be stored in the graphical system of the local curve, using, i.e., vertex arrays in OpenGL, or their equivalent in other systems. These methods are only possible to use for affine transformations of local curves, and it is a requirement that the affine transformation is done by homogeneous matrices, typically used in graphical systems.

ii) In each local curve, store the values of all basis functions (Bernstein polynomials) in each of the local sample points. The most complete local sampling is: for each of the local sample points, store not only the values, but all derivatives, i.e., the Bernstein/Hermite matrix described in equation (4.8) and in algorithm 12. This means that for a local curve, indexed by $i$, the total local preevaluation is computing a vector of the matrices,

$$\{\mathbf{B}_{d_i}(\omega_i(x_j),\delta_i)\}_{j=s_i}^{e_i}.$$

### 4.3.3 Examples

In this subsection we will see examples of Hermite-interpolation using three different original curves. These three curves are approximated by ERBS curves using local Bézier curves of different degrees. The purpose is to show some of the properties and possibilities of ERBS curves using local Bézier curves. Most of the examples are "closed" curves, but there is also one example of an "open" curve. The examples clearly show that Hermite-interpolation is not an "optimal" approximation (a well known fact), and that it is possible to improve the solution by scaling the local curves, or by scaling the local domain in the input of the interpolation process.

The first example is a so called "Rose-curve" described in [58], and defined by the formula,

$$g(t) = \begin{pmatrix} \cos t \cos\left(\frac{7}{4}t\right) \\ \sin t \cos\left(\frac{7}{4}t\right) \\ 0 \end{pmatrix} \quad \text{for } t \in [0, 8\pi). \tag{4.15}$$

A plot of this formula will look like a rose with 14 petals. The number of petals is actually 2 times the numerator in the fraction in the cosine in equation (4.15). The speed of the "Rose-curve" is oscillating between 1 and 1.75, with the slowest speed at the center of the rose and the fastest speed at the tip of each petal.

In Figure 4.4, the "Rose-curve" is interpolated by an ERBS curve using 4 interpolation points on each petal. There are a total of 4*14=56 interpolation points uniformly distributed on the used parameter interval. The position and the first derivative in each of the interpolation points are used, and one can easily see the effect of using only one derivative. In this case the approximation regards the second derivatives and, thus, the curvature, to be zero at all intersection points. Since the curvature always is positive (greater than zero), the result is that the curve is getting too long. This can clearly be seen in Figure 4.4 where, compared to the original curve, the approximating curve is buckling between two intersection points. In Figure 4.5, the local curves are plotted. Because all local curves are of degree 1, they are straight lines. The length of the lines, and the fact that they intersect each other, also indicates that the resulting curve is going to be too long and thus will buckle.

In Figure 4.6, the same interpolation as in the previous example (Figure 4.4) is done, but the resulting local curves are now scaled by 0.5, as one can see in Figure 4.7. When scaling, it is important that the interpolation point is the origin of the local coordinate system

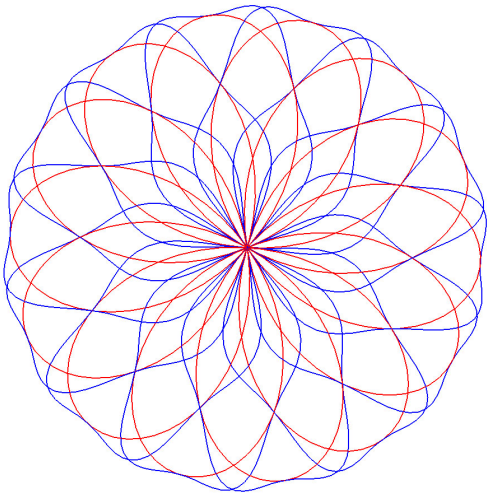

***Figure 4.4:*** *An ERBS curve (blue) approximating a "Rose-curve" (red) by using 56 interpolation points; only the position and the first derivative in each point are used. This approximation seems to make the curve too long, so that the curve is buckling between the interpolation points.*

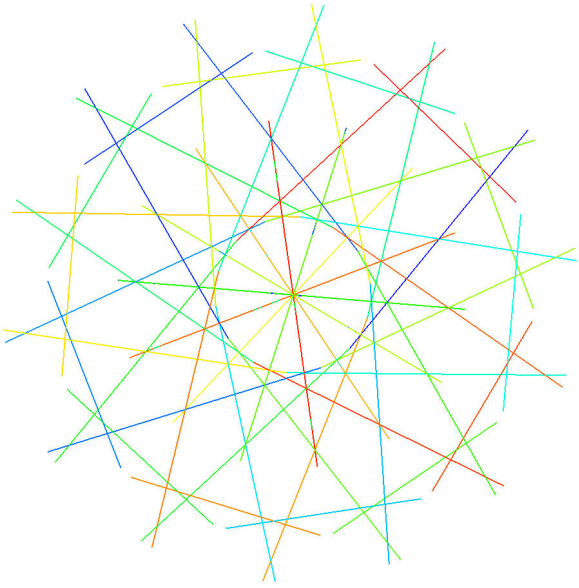

***Figure 4.5:*** *The 56 local Bézier curves (lines) of the ERBS curve from Figure 4.4, plotted in different colors (all 1st degree). Because the "Rose-curve" has 14 petals, and there are 14*4=56 interpolation points, the local curves (lines) are "symmetrical around the rose center". In addition, every pair of subsequent lines intersect.*

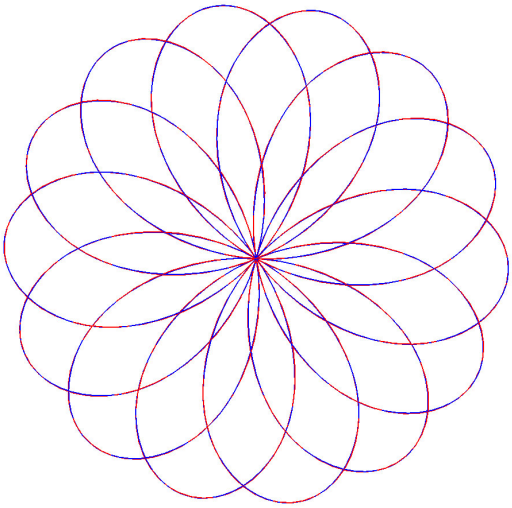

***Figure 4.6:*** *An ERBS curve (blue) approximating a "Rose-curve" (red) by using 56 interpolation points; only the position and the first derivative in each point are used, but the local curves are scaled by 0.5 after the interpolation. This approximation seems to be, geometrically, very close to the original curve.*

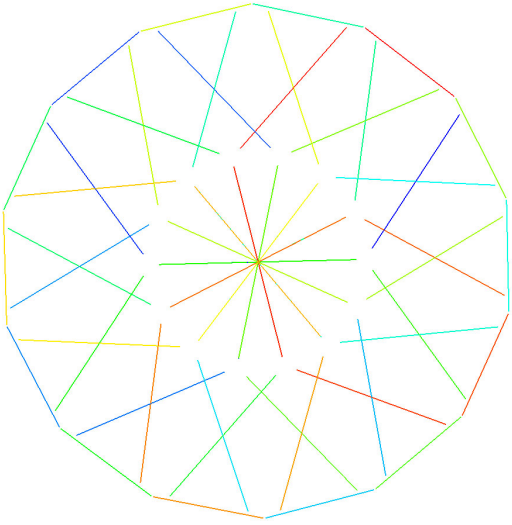

***Figure 4.7:*** *The 56 local Bézier curves (lines) of the ERBS curve (all 1st degree) from Figure 4.6 plotted in different colors. Because the "Rose-curve" has 14 petals, and there are 14*4=56 interpolation points, the local lines are "symmetrical around the rose center". In addition, each line does not intersect with the previous and next line in the sequence.*

of each local curve, so that the interpolation points are not moving. The resulting curve is geometrically very similar to the original "Rose-curve". The speed however will oscillate at a greater rate. It is possible to get the same result using another method. One can scale the input derivative in the interpolation process instead of scaling the resulting curve, and still get the same result. As can be seen in this example, the result is geometrically very good, but there is still a potential for improving the result further by changing the scaling factor. The next level, however, which has an even greater potential for improving the result, is if the scaling factor differs from local curve to local curve.

In Figure 4.8, the "Rose-curve" is, as in the two previous examples, interpolated by an ERBS curve using a total of 56 interpolation points, but now the position and two derivatives in each interpolating point are used. The result is quite good, but not as geometrically good as in the previous example. The speed however is quite equal to the original curve, and is thus much better than in the last example (where the local curves are scaled). In Figure 4.5, the local curves are plotted. They are all of degree 2, and "symmetrical", in the sense that on every petal there is a set of local curves that have the same form on each petal. One can also noticeably improve the result by changing the local curves in this example, using a similar argument as in the previous example. To make changes in this example, however, is more complex, but the possibilities are even bigger than in the previous example. Besides scaling, there are a lot of other "editing" possibilities on the local curves.

In Table 4.1 and Table 4.2 can be found two different "evaluation methods" of the quality of the approximation in the three previous examples: one using a "mathematical deviation", and another using "geometrical deviation" as a measure.

The next curve example is a so called "Cardioid curve" described in [32], and defined by the formula,

$$g(t) = \begin{pmatrix} 2\cos t\,(1+\cos t) \\ 2\sin t\,(1+\cos t) \\ 0 \end{pmatrix}, \quad \text{for } t \in [0, 2\pi). \tag{4.16}$$

A plot of (4.16) will look like an apple, with a cusp at the top. In Figure 4.10, the "Cardioid curve" is interpolated by an ERBS curve using a total of 7 interpolating points uniformly distributed on the parameter interval. In each of the interpolating points are the positions, the first derivatives, and the second derivatives used in the Hermite interpolation process. No special effort is done to reshape the cusp. There is one interpolation point at the bottom, and three on each side, where the distance (in the plane) is getting shorter on the way up from the bottom intersection point. The third point on each side is relatively close to the cusp.

In Figure 4.11, all seven local curves are plotted in different colors. They are all 3rd degree Bézier curves, modeling the shape of the original curve quite well, but they seem to be too long because not only do they overlap half of the next curve, but they also overlap some of the following curve. The logical reason for this is, of course, the same as for the previous example, that the algorithm in this case assumes that the fourth derivatives are zero.

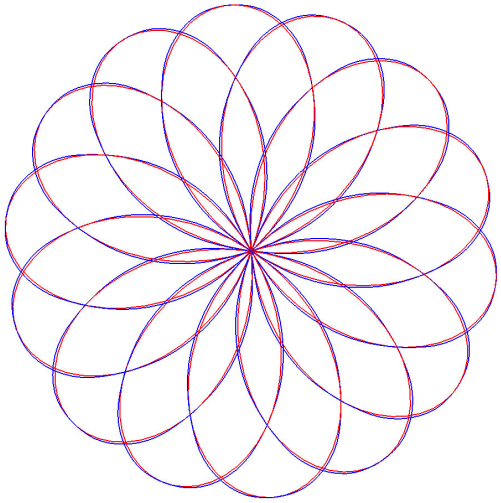

***Figure 4.8:*** *Two curves partly covering each other. The red curve is the original "Rose-curve", the blue curve is the approximating ERBS curve. The approximation is made by 56 interpolation points, the position and the first and second derivatives in each point are used.*

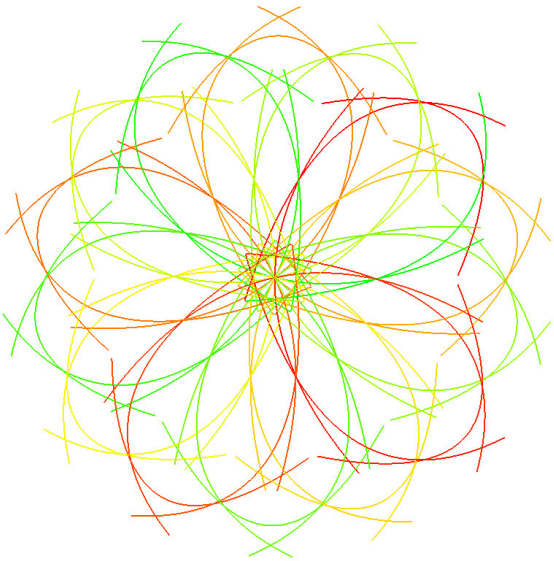

***Figure 4.9:*** *The 56 local Bézier curves of the ERBS curve from Figure 4.8 plotted gradually from green to red. They are all 2nd degree Bézier curves, forming the original curve around each interpolating point. Because the "Rose-curve" has 14 petals, and we are using 14*4=56 interpolation points, the local curves are "symmetrical around the rose center".*

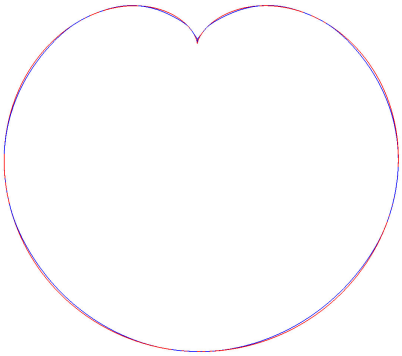

***Figure 4.10:*** *Two curves partly covering each other. The red curve is the original "Cardioid curve" . The blue curve is the approximating ERBS curve. The approximation is made by 7 interpolation points; the position and the three first derivatives in each point are used.*

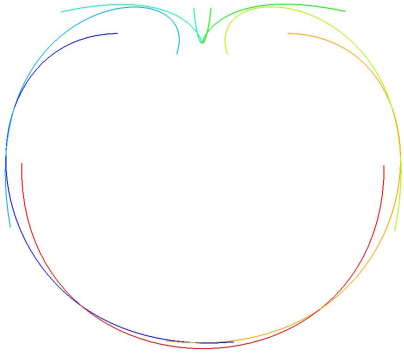

***Figure 4.11:*** *The seven local Bézier curves of the ERBS curve, from Figure 4.10, plotted in different colors. All seven curves are 3rd degree Bézier curves with 4 control points.*

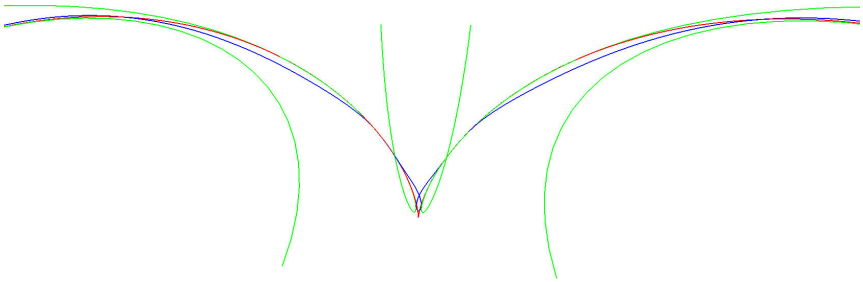

***Figure 4.12:*** *A zoomed picture of both the original "Cardioid curve" (red), the approximating ERBS curve (blue), and four of the local Bézier curves (green). The picture shows how the cusp is approximated (even when using uniform sampling).*

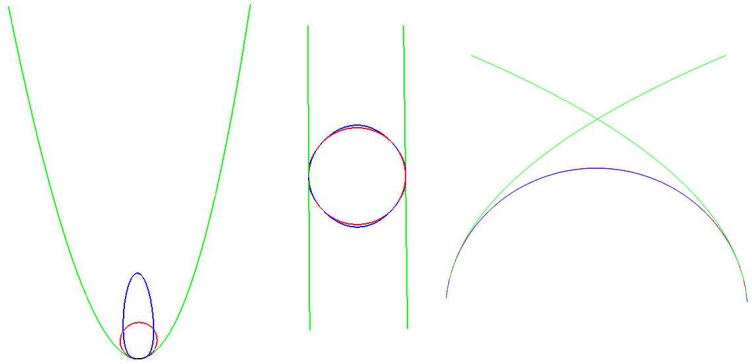

***Figure 4.13:*** *Three examples of approximations of circles/circular arcs by ERBS curves using local Bézier curves. The original curves are red, the ERBS curves are blue, and the local Bézier curves are green. On the left hand side only one local curve is used. In the middle, two local curves of 1st degree (lines) are used. On the right hand side there is a circular arc approximated by an ERBS curve using two 2nd degrees Bézier-curves as local curves.*

In Figure 4.12, the area around the cusp is plotted separately, including the original curve in red, the ERBS curve in blue, and 4 of the local Bézier curves in green. The result is close to being a cusp, but not an exact one. If the local curves are slightly changed, the result can be more accurate, and it is actually possible to get a real cusp.

The last example shows three different approximations of circles/circular-arcs. On the left hand side of Figure 4.13 there is an approximation of a circle using only one interpolating point, with the position and the first and second derivatives. The local curve is, therefore, only one 2nd degree Bézier curve. In the figure the original circle is red, the approximating ERBS curve is blue, and the local Bézier curve is green. The result can be improved by scaling or editing the local curve. This type of curve was also mentioned on page 75.

In the middle of Figure 4.13, a circle is approximated by only two interpolating points, using only the position and the first derivatives in each of the two points. The local lines are scaled by 0.92 after the interpolation. The result is visually quite good.

On the right hand side of Figure 4.13, an arc slightly smaller than a half circle is approximated by an ERBS curve using two interpolating points including the position, first and second derivatives. In this case, another type of correction is showed, the domain interval is scaled by 0.635. The result is very accurate. One can see almost only the blue ERBS curve which is covering the red original circular arc.

In any of these examples, the results are not “optimal” with respect to any kind of optimal criterion, and one can only see that there are possibilities for improvements, using simple scaling, or more complex editing of the local curves.

## 4.4 Circular arcs as local curves

Beside B-splines and NURBS, Arc-splines have been one of the successful spline types in practical use (see [8], or [44]). The arc-splines are in general not $C^2$ (or even $G^2$), because the curvature is typically piecewise constant (has the staircase form). Using circular arcs as local curves in an ERBS curve, however, gives an $C^\infty$ curve, because it is a blending of arcs, using the ERBS basis function as the blending factor. In this section we will, therefore, look at circular arcs as local curves. Another motive for investigating circular arcs as local curves is the possible relationship to nonlinear splines described by Even Mehlum in [42] and [43].[3]

An applicable function for a circular arc is $\widetilde{c}\colon [\alpha_0, \alpha_1] \subset \mathbb{R} \to \mathbb{R}^2$ with the expression

$$\widetilde{c}(\alpha) = \frac{1}{\varkappa} \begin{pmatrix} \sin(\varkappa\delta\alpha) \\ 1 - \cos(\varkappa\delta\alpha) \end{pmatrix}, \quad \text{if} \quad \varkappa > 0. \tag{4.17}$$

Equation (4.17) has the following fine properties for the use as local curves:

i) $\widetilde{c}(0)$ is in the local origin.
ii) $\widetilde{c}'(0)$ is parallel with the x-axis, and the length (speed) is $\delta$.
iii) $\widetilde{c}''(0)$ is parallel with the y-axis, and the length is equal to the curvature $\varkappa$.

Recall that a straight line is also an arc, with infinite radius (or curvature $\varkappa = 0$). A general formula for an arc must take this into consideration. We therefor get the following general formula for an arc-curve to use as local curves,

$$c(\alpha) = \begin{cases} \begin{pmatrix} \delta\alpha \\ 0 \end{pmatrix}, & \text{if} \quad \varkappa = 0, \\ \frac{1}{\varkappa} \begin{pmatrix} \sin(\varkappa\delta\alpha) \\ 1 - \cos(\varkappa\delta\alpha) \end{pmatrix}, & \text{if} \quad \varkappa > 0. \end{cases} \tag{4.18}$$

This function (4.18) still has kept all the properties mentioned above. The function is also easy to expand to $\mathbb{R}^3$ (or a higher dimension), by just regarding it to be in the xy-plane, i.e. all other coordinates are zero. All derivatives are also straight forward to compute, but this will not be done here. Note that if the curve has a curve length parametrization, then the factor (and, thus, speed) is $\delta = 1$.

### 4.4.1 Local Arc curves and modified Hermite interpolation

It is, of course, not possible to make a general Hermite interpolation of just any curve by arc-curves. It is, however, possible to interpolate position, first derivative and the curvature at given points, and this is the intention of the modified Hermite interpolation.

For local Arc curves we get the following settings:

[3] A relationship to nonlinear splines will not be discussed further here, but it is a topic for later investigations, especially because Even Mehlum, in the late nineties, asked if I could look at nonlinear splines. Unfortunately other things prevented me from doing so.

- Given is a curve $g(t)$, $g : [t_s, t_e] \subset \mathbb{R} \to \mathbb{R}^n$, where we might have $n = 1, 2, 3, ...,$
- given is the number of samples, $m > 1$.
- Generate a knot vector by:
  - first set $t_1 = t_s$,
  - then set $t_m = t_e$.
  - Then for $i = 2, 3, ..., m-1$, generate $t_i$ so that $t_i > t_{i-1}$, but where $t_{m-1} < t_m$.
  - Finally, $t_0$ and $t_{m+1}$ must be set according to the rules for "open/closed" curves.
- Make an ERBS curve using the knot vector $\{t_i\}_{i=0}^{m+1}$, and generate local curves, in such a way that the ERBS curve is interpolating the position $g(t_i)$, the first derivative $Dg(t_i)$, and the curvature $\varkappa(t_i)$, for $i = 1, ..., m$.

This is not an approximation using a general Hermite interpolation method. Note that the approximation is only using the curvatures, and it does not consider the change of speed. Later on in section 4.4.3, we will arrive at a better approximation method, a method which also makes a reparametrization of the curve to approximate curve length parametrization.

A modification of Theorem 2.4 implies that we have the following requirements for the interpolations. For all interior knots $t_i$, $i = 1, .., m$,

$$\begin{aligned} g(t_i) &= f(t_i) = c_i \circ \omega_i(t_i), \\ Dg(t_i) &= Df(t_i) = \delta_i \, Dc_i \circ \omega_i(t_i), \\ \varkappa_g(t_i) &= \varkappa_f(t_i) = \varkappa_{c_i}(t_i), \end{aligned}$$

where $g(t)$ is the original curve, $f(t)$ is the ERBS curve, and $c_i(t)$, $i = 1, .., m$ are the local arc-curves. Because we choose the arc-curves, $c_i(t)$, to have a curve length parametrization, and we want the interpolation point to be in the local origin, it follows that $\omega_i(t)$ from definition 2.7 is the mapping

$$\omega_i(t) : [t_{i-1}, t_{i+1}] \to [(t_{i-1} - t_i)\,|Dg(t_i)|, (t_{i+1} - t_i)\,|Dg(t_i)|] \tag{4.19}$$

defined by the affine mapping

$$\omega_i(t) = (t - t_i)\,|Dg(t_i)|. \tag{4.20}$$

From this it follows that

$$\omega_i(t_i) = 0,$$

which fulfills the request for the interpolation points to be in the local origins. The first derivative of $\omega_i(t)$ gives us the domain scaling factor defined in Theorem 2.4, which is then

$$\delta_i = |Dg(t_i)|, \tag{4.21}$$

The curvature is, however, unchanged by the affine domain mapping, because it is an intrinsic (geometric) property.

The recipe for the interpolation is now quite simple, and a short description follows. For each interior knot $t_i$, $i \in \{1, 2, ..., m\}$ it is given

$$D^j g(t_i), \quad \text{for } j = 0, 1, 2, \text{ and the knot value } t_i.$$

By (4.21), the scaling factor $\delta_i$ follows directly from the input. The knot value $t_i$, together with $\delta_i$, is needed to determine $\omega_i(t)$. What remains is to compute the curvature $\varkappa_i$, and the position/orientation based on the Frenet frame. We start by computing the vectors

$$\mathbf{x} = \frac{Dg(t_i)}{\delta_i}$$

then,

$$\widehat{\mathbf{y}} = \frac{D^2 g(t_i) - \langle \mathbf{x}, D^2 g(t_i) \rangle \mathbf{x}}{\delta_i^2}$$

and it follows now that

$$\varkappa_i = |\widehat{\mathbf{y}}|.$$

We now have two possibilities concerning equation 4.18, namely that $\varkappa_i > 0$ or $\varkappa_i = 0$.

- If $\varkappa_i > 0$ we can normalize $\widehat{\mathbf{y}}$, and then we get the vector

  $$\mathbf{y} = \frac{\widehat{\mathbf{y}}}{\varkappa_i}.$$

  In the general case of $\mathbb{R}^n$, $n = 2, 3, ...$, we have to make the rest of the orthogonal vectors match the axis in the coordinate system. E.g., if $n = 3$, and we use a left hand coordinate system (as in OpenGL) we get,

  $$\mathbf{z} = \mathbf{y} \wedge \mathbf{x}.$$

  The homogeneous matrix for adjusting the position and orientation is then

  $$M = \begin{bmatrix} \mathbf{x}_x & \mathbf{y}_x & \mathbf{z}_x & g(t_i)_x \\ \mathbf{x}_y & \mathbf{y}_y & \mathbf{z}_y & g(t_i)_y \\ \mathbf{x}_z & \mathbf{y}_z & \mathbf{z}_z & g(t_i)_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{if } \varkappa > 0, \tag{4.22}$$

  where the first three columns are the three orthogonal vectors defining the orientation, using 0 as the homogeneous coordinate, and the last column is the position using 1 as the homogeneous coordinate.

  We now have to insert homogeneous coordinates to the result of the evaluator and its equivalent derivatives, 1 to the value, and 0 to all derivatives. The final result of the evaluation is the primary evaluation multiplied by the homogeneous matrix $M$.

- if $\varkappa_i = 0$ we do not get any vector $\mathbf{y}$. It is, of course, possible to generate a set of orthonormal vectors, but it might not be necessary, as in this case only the first coordinate is different from zero, and the homogeneous matrix can, therefor, be simplified

  $$M = \begin{bmatrix} \mathbf{x}_x & 0 & 0 & g(t_i)_x \\ \mathbf{x}_y & 0 & 0 & g(t_i)_y \\ \mathbf{x}_z & 0 & 0 & g(t_i)_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{if } \varkappa = 0.$$

  If, however, the matrix is supposed to be used in a graphical system, we have to be sure that the matrix is invertible, and that the column vectors are linearly independent, and thus generate vectors different from zero.

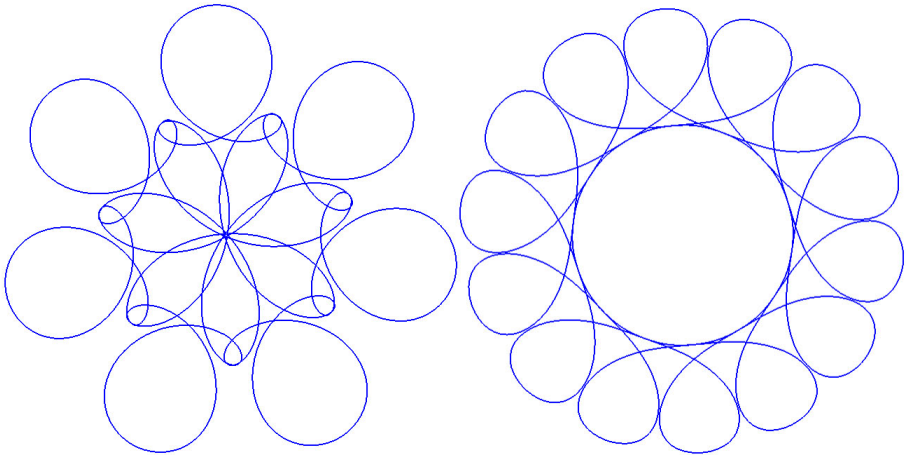

***Figure 4.14:*** *Two examples of approximations of the "Rose-curve" by ERBS curves using Circular arcs as local curves. On the left hand side only seven local curves are used. The "Rose-curve" is not completely reconstructed, only half of the "petals" are present. On the right hand side is an ERBS curve approximating the "Rose-curve" using 14 circular arcs as local curves.*

### 4.4.2 Examples

In this subsection, we will consider four examples. The two first examples show modeling subsets, i.e. using a subset of the available information and, thus, getting a result reflecting this. The next example is an approximation of a helical curve over a logarithmic spiral. This curve has two different ("opposite") properties, and is not easy to approximate. The last example is the "Rose-curve" interpolated at 56 points. This is a comparison between using local Bézier curves or local Arc curves, and it leads us into another approximation method (next subsection).

The first example is another approximation of the "Rose-curve". The same "Rose-curve" is used in several examples with ERBS curves using local Bézier curves, see Figure 4.4, Figure 4.6, and Figure 4.8. But now the approximation is different from the previous ones. In Figure 4.14, there are two plots. On the left hand side there is a plot of an ERBS curve interpolating a "Rose-curve" at only 7 points, using the positions, the first derivatives, and the curvatures. All interpolation points are at the tip of the "petals", and the local curves are circular arcs. Remember that there are a total of 14 "petals", so we are only reconstructing half of them, using only the information at one point for each of the 7 "petals". On the right hand side there is a plot of an ERBS curve interpolating the "Rose-curve" at 14 points, using the positions, the first derivatives, and the curvatures. Also in this case there are only interpolation points at the tip of the "petals" used, and the local curves are circular arcs. These examples show us that it is possible to make quite a good reconstructions of a curve based on incomplete information.

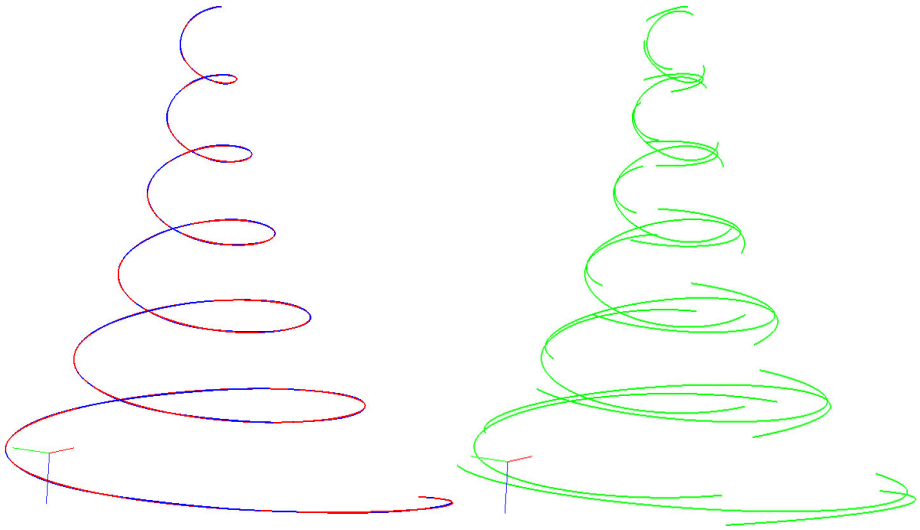

***Figure 4.15:*** *An example of approximation of a Helical curve over a logarithmic spiral by an "ERBS" curve using local Arc curves. The original curve is red, the "ERBS" curve is blue and they are both plotted on the left hand side. The local Arc curves are plotted on the right hand side, and are colored green. The number of interpolation points and, thus, local Arc curves, is 20.*

The second example is a Helical curve over a logarithmic spiral. In Figure 4.15, there is a plot of a Helical curve over a logarithmic spiral, and an approximation of this Helical curve by an ERBS curve using local Arc curves. The original curve is described in [32], and the formula is

$$c(t) = \begin{pmatrix} e^{\frac{t}{20}} \cos t \\ e^{\frac{t}{20}} \sin t \\ \frac{t}{2} \end{pmatrix} \quad \text{for } t \in [0, 12\pi).$$

In the figure the original Helical curve is plotted in red on the left hand side, and the approximating ERBS curve is plotted together with the original curve, but in blue. The plot is done in 3D, using depth buffer. The result is that we can see which curve is closest to the "camera" by its color. As one can see, the two curves are very close to being equal. 20 interpolation points are used, where the position, the first derivatives, and the curvatures are used in each point. The interpolation points are equally distributed along the parameter interval. The local curves are circular arc-curves, and on the right hand side it is these 20 local curves which are plotted in green. The length of these local curves indicates the "speed" of the "global" curve, and it is easy to see that the speed is highest at the bottom. Another observation one can do when studying the local curves in Figure 4.15, is that they fit very well, in the sense that the end of one local curve is very close to the start of the one after the next local curve. This indicates that the approximation is rather good.

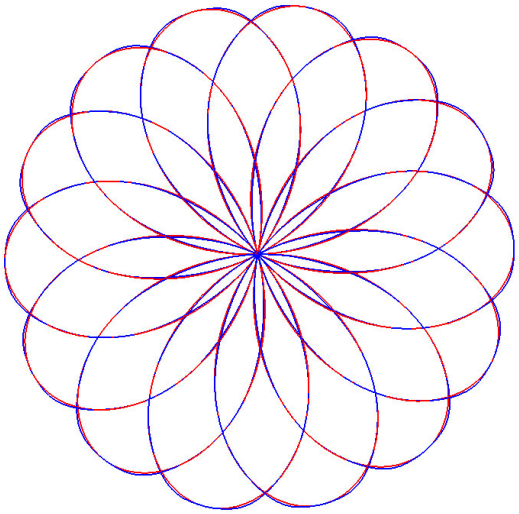

***Figure 4.16:*** *Two curves partly covering each other. The red curve is the original "Rose-curve", the blue curve is the approximating ERBS curve using local Arc curves. The approximation is made by 56 interpolation points, using the position, the first derivative and the curvature at each of the points.*

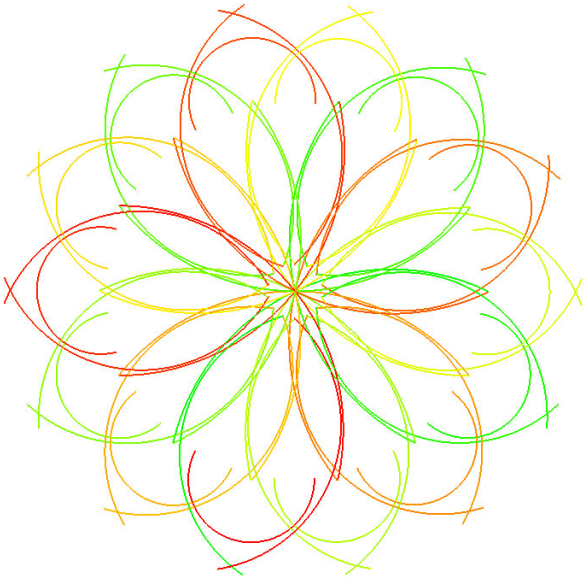

***Figure 4.17:*** *The 56 local Arc curves of the ERBS curve from Figure 4.16 plotted in colors gradually changing from green to red. They are forming the original curve locally around the interpolating points. One can see that they are quite similar to the Bézier curves at figure 4.9.*

| Local Curves | degree | scaling/interval | $L^\infty(f-g)$ | $L^2(f-g)$ |
|---|---|---|---|---|
| Bezier-curves | deg. 1 | parameter-interval | 0.091 | 0.053 |
| | | $0.5 \times$ parameter-interval | 0.049 | 0.028 |
| | deg. 2 | parameter-interval | 0.0092 | 0.0065 |
| Arc curves | Circular arc | speed $\times$ parameter-interval | 0.021 | 0.012 |
| | | curve length | 0.0110 | 0.0060 |

***Table 4.1:*** *The table shows the result of measuring the deviation of the approximation of the "Rose-curve" using the $L^\infty(f-g)$ and the $L^2(f-g)$ norms (see (4.23) and (4.24)). The computations are done by using numerical integration (algorithm 6) when necessary. The table shows the result of 5 different approximation curves, 3 using local Bézier curves, and 2 using local Arc curves. All examples are shown in this and the previous section. Recall that the diameter of the "Rose-curve" in the examples is 2.*

The last example is the by now well known "Rose-curve". The reason for using this example once more is to compare the result with the previous example shown in Figure 4.8 and Figure 4.9. The two figures, 4.16 and 4.17, show us the same type of approximation as in figures 4.8 and 4.9. On the two figures is the "Rose-curve", an approximation of the "Rose-curve" and its local curves, which are here circular arcs. The approximation is made by using 56 interpolation points, where the position, the first derivative, and the curvature at each point are used as in the Bézier example, except that in the Bézier example the second derivative is used instead of the curvature. The result is clearly comparable to the result using 2nd degree Bézier curves as local curves. It is of special interest to compare the local curves in this example (Figure 4.17) with the previous example (Figure 4.9). An observation is that the local Arc curves in this example are closer to the total curve than the local Bézier curves in the previous example.

It is, of course, of interest to compare the result of the approximation by a measure. We can measure the deviation using norms. We, therefore, first define a max norm

$$L^\infty(f-g) = \max_{t\in[t_1,t_n+1]} |f(t)-g(t)|, \tag{4.23}$$

to investigate the guarantied maximum deviation, and then an $L^2$ norm,

$$L^2(f-g) = \sqrt{\frac{1}{t_{n+1}-t_1}\int_{t=t_1}^{t_{n+1}} |f(t)-g(t)|^2 dt}, \tag{4.24}$$

to investigate the "quadratic" mean deviation. Note that these two norms also take the parametrization into consideration, and that they are measuring the "mathematical" deviation and not the "purely geometric" deviation. In Table 4.1 we can see the approximate result of the computation of the examples from this and the previous section. Recall that the diameter of the "Rose-curve" is 2.0, and the numbers in Table 4.1 are computed from this diameter. One can clearly see that using 2nd degree local Bézier curves gives the best result, according to the $L^\infty$ norm, and is almost best for the $L^2$ norm. The reason for this is the parametrization, because one can clearly see that some of the other examples

fit better geometrically. The local Arc curves are however parametrized with a constant speed, while for the local Bézier curves there is a smooth change of speed. If we take this into consideration and make a modification of the scaling factor, we can see some new results and possibilities.

The last approximation type in Table 4.1 is a modification of the affine mapping $\omega_i$, defined in (4.20). The modification is

$$\omega_i(t) = \frac{t - t_i}{t_{i+1} - t_{i-1}} \int_{t=t_{i-1}}^{t_{i+1}} |Dg(t)|dt.$$

We actually scale by using the curve length of the original curve $g(t)$ at the respective knot intervals instead of using the speed of the curve. Notice that in this example the scaling factor $\delta_i$, defined in (4.21) is not changed. The result is noticeably improved, and is nearly equal to the result for the local Bézier curves. In the next subsection, we will study reparametrization further, concentrated on approximative curve length parametrization.

### 4.4.3 Reparametrization and using an approximative curve length parametrization

Circular Arc curves as described in equation (4.18) have curve length parametrization if $\delta = 1$. If we regenerate the knot vector to reflect the length of the curve intervals, we might get an approximative "curve length" parametrization of a parametrized curve by an ERBS curve using local "Arc curves". This is described in the following sequence.

- Given a curve $g(t) \in \mathbb{R}^n$, where we might have $n = 1, 2, 3, ...,$
- and given a sample number $m > 1$.
- Generate a knot vector by first setting $t_1$ equal to the start of the domain of $g$ which is going to be interpolated, and then by setting

  $$t_i = t_{i-1} + \int_{t=x_{i-1}}^{x_i} |Dg(t)|dt, \quad \text{for} \quad i = 2, 3, ..., m,$$

  and finally by setting $t_0$ and $t_{m+1}$ according to the rules for "open/closed" curves.
- Make an ERBS curve by using the knot vector $\{t_i\}_{i=0}^{m+1}$, and by generating local curves in such a way that the ERBS curve is interpolating the positions $g(t_i)$, the normalized derivatives $\frac{Dg(t_i)}{|Dg(t_i)|}$, and the curvatures $\varkappa_g(t_i)$, all for $i = 1, ..., m$.

The consequences are that we have to change (4.19), (4.20), and (4.21). Since the knot vector already represents the curve length, $\omega_i$ must not scale the domain, but only translate it. Thus we get

$$\omega_i(t) : [t_{i-1}, t_{i+1}] \rightarrow [(t_{i-1} - t_i), (t_{i+1} - t_i)]$$

defined by the affine mapping

$$\omega_i(t) = (t - t_i).$$

It also follows that,

$$\delta_i = 1.$$

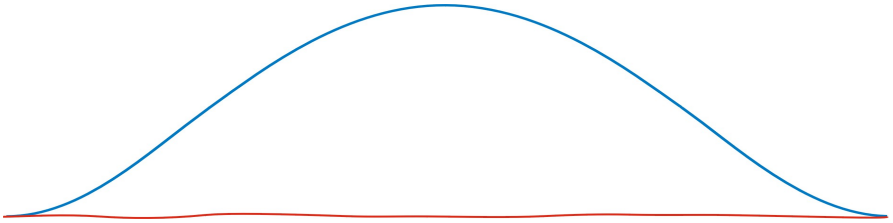

***Figure 4.18:*** *The blue curve is the plot of the function describing the speed of the original "Rose-curve", but for only one of the fourteen petals. The value at the start and end is 1.0, and the top value in the middle is at 1.75. The red curve is the plot of the function describing the speed of the approximating ERBS curve. The value is close to 1.0 throughout.*

Doing this we get an approximative curve length parametrization of a curve. If we look at the previous "Rose-curve" example, and use this new method, we get a curve very close to being curve length parametrized. In Figure 4.18, there is a plot of the speed of the original "Rose-curve", and the new ERBS curve. The plot is only done for one petal, i.e. $\frac{1}{14}$ of the curves. The speed of the original curve, plotted in blue, is deviating between 1 and 1.75. The speed of the new ERBS curve, plotted in red, is very close to 1.0 throughout. The maximum deviation is actually 0.01. The table below shows us the deviations of the speed to the new curve measured in three different norms.

| Norm type $L^\infty(\lvert Df\rvert - 1)$ | Norm type $L^2(\lvert Df\rvert - 1)$ | Norm type $L^1(\lvert Df\rvert - 1)$ |
|---|---|---|
| 0.01 | 0.003 | 0.002 |

To see how well the new curve approximates the curve length parametrization we can actually look at the curve length of the original and the ERBS curves. The curve length of the original "Rose-curve", and the new ERBS curve are,

| | | |
|---|---|---|
| The original "Rose-curve" | curve length: | 35.20 |
| The new "regular ERBS curve" | curve length: | 35.15 |

The reparametrization is not an affine mapping. We, therefore need a new error measure which only refers to the geometric shape, and not to the speed of the parametrization. A typical such measure is the Hausdorff distance between the two curves. We, therefore, start by defining the closest point from a point to a curve. (Efficient algorithm for closest points can be found in [35].)

**Definition 4.3.** *Given is a point* $\mathbf{p} \in \mathbb{R}^n$*, and a curve* $f \subset \mathbb{R}^n$*, with a map* $\mathbf{f} : I \subset \mathbb{R} \to f$*. We define*

$$C_f(\mathbf{p}) : \mathbb{R}^n \to f,$$

*to be the closest point on a curve* $f$ *from a point* $\mathbf{p}$*. By closest point we mean that the distance (metric)*

$$d(\mathbf{p}, C_f(\mathbf{p})) < d(\mathbf{p}, \mathbf{q}), \quad \textit{for all} \quad \mathbf{q} \in O(C_f(\mathbf{p}))$$

*where* $O(\mathbf{p}_f)$ *is a neighborhood, on the curve* $f$*, around the point* $\mathbf{p}_f \in f$*.*

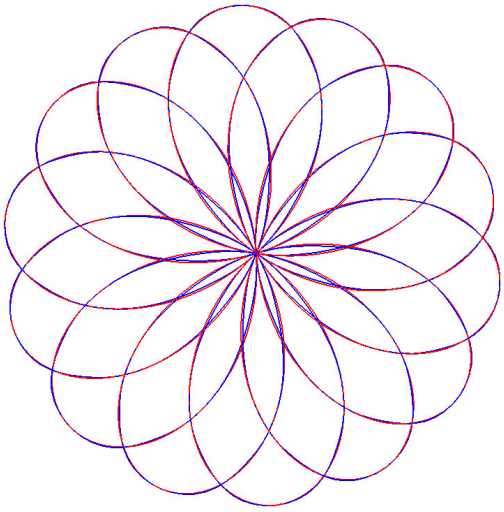

***Figure 4.19:*** *Two curves partly covering each other. The red curve is the original "Rose-curve", the blue curve is the approximating ERBS curve using local Arc curves. The approximation is a reparametrization to curve length parametrization.*

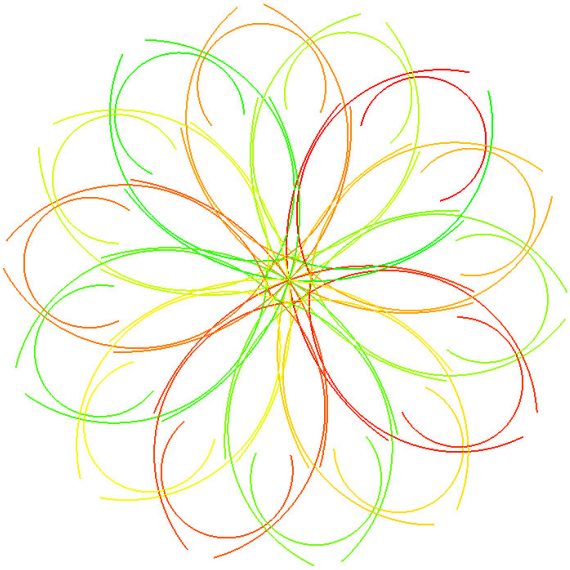

***Figure 4.20:*** *The 56 local Arc curves of the ERBS curve from Figure 4.19 plotted in colors gradually changing from green to red. They are forming the original curve locally around the interpolating points. One can see that the curves on the sides of each petal are shorter than the ones in Figure 4.17, and the curves at the tip of the petals are longer than the ones in Figure 4.17.*

| Local Curves | degree/parametriztion | scaling/interval | $L_G^\infty(f,g)$ | $L_G^2(f,g)$ |
|---|---|---|---|---|
| Bezier-curves | deg. 1/unchanged par. | parameter-interval | 0.089 | 0.049 |
| | | $0.5 \times$ par.-interval | 0.0030 | 0.0016 |
| | deg. 2/unchanged par. | parameter-interval | 0.0089 | 0.0059 |
| Arc curves | Unchanged param. | speed $\times$ par.-interval | 0.0086 | 0.0037 |
| | | curve length | 0.0067 | 0.0032 |
| | Curve length param. | curve length | 0.0078 | 0.0036 |

***Table 4.2:*** *The table shows the result of measuring the geometric deviation of the approximation of the "Rose-curve" using the $L_G^\infty(f,g)$ in (4.25) and the $L_G^2(f,g)$ in (4.26) metrics. The computations are done by using numerical integration (algorithm 6). The table shows the result of 6 different approximation curves, 3 using local Bézier curves, and 3 using local Arc curves (all examples showed in this and the previous section). Only the last example uses reparametrization to curve length parametrization, all the others have the original parametrization. Reall that the diameter of the "Rose-curve" is 2.0, and the numbers in the table correspond to this diameter.*

When comparing a curve and an approximating curve, we can assume that the two curves are "close" and in a way have the same shape. We can, therefore, propose a non-symmetric version of the Hausdorff distance for measuring the result, using a geometric version of a metric related to a max norm ,

$$L_G^\infty(f,g) = \max_{t\in[t_1,t_n+1]} |f(t) - C_g(f(t))|. \tag{4.25}$$

Notice that, although this metric is not "symmetric" in the sense that $L_G^\infty(f,g)$ is not necessarily equal to $L_G^\infty(g,f)$, it gives useful information for investigating the maximum geometric deviation. To construct a metric related to the $L^2$ norm, we can do the following,

$$L_G^2(f,g) = \sqrt{\frac{\int_{t_1}^{t_{n+1}} \left|f(t) - C_g(f(t))\right|^2 |Df(t)|\, dt}{\int_{t_1}^{t_{n+1}} |Df(t)|\, dt}}, \tag{4.26}$$

which actually is not so far away from being the area of the square of the distance between the two curves divided by the curve length. Note that the two curves are supposed to have the same curve length, and are supposed to be "close to equal".

In Table 4.2 we can see the result of using these measuring methods in the 6 different examples of approximating the "Rose-curve". The best result actually uses the 1st degree local Bézier curves, where the parameter interval is scaled by $\frac{1}{2}$. This could also be observed in Figure 4.6. The scaling factor $\frac{1}{2}$ is, however, not a computed optimized value, but rather a "shot in the dark".

We can also clearly see that geometrically the local arc-curves are better than the 2nd degree local Bézier curves, and that using reparametrization to approximative curve length

parametrization actually gives a good result. To confirm the measuring method we can look at the result for the 2nd degree local Bézier curve, and we will see that it is only slightly improved compared to the previous mathematical measurement (from Table 4.1 we get $L^{\infty}(f-g) \approx 0.0092$ and from Table 4.2, $L^{\infty}_{G}(f,g) \approx 0.0089$). This is a very likely result because the local Bézier curves are also approximating the parametrization.

In Figure 4.19 we can see the original "Rose-curve" (red) and the new "curve length parametrizised" ERBS curve (in blue), and in Figure 4.20 the belonging arc-curves are plotted in colors gradually changing from green to red.

## 4.5 Affine transformation on local curves

At the end of subsection 4.3.1 there is a suggestion of translating the coefficients of the local Bézier curve to ensure that the interpolation point is in the "local origin", and in section 4.4 a homogeneous matrix is introduced for positioning and orienting the local curves. The suggestion for the implementation of ERBS curves is as follows.

- The most important local variables for an ERBS curve are:
  - i) A state telling if the curve is "open/closed", definition 4.2.
  - ii) A vector of references to $n$ local curves.
  - iii) A knot vector of size $n+2$.
  - iv) An "ERBS-evaluator" object, defined on page 62.
- For each of the local curves in the ERBS curve there are:
  - i) The local data describing the curve; coefficient vector for Bézier curves, or velocity, curvature and domain mapping for Arc curves.
  - ii) The local data must be translated and rotated so that the interpolation point is in the local origin, the first derivative is on the local x-axis, the second derivative is in the local xy-plane etc.
  - iii) An homogeneous matrix $M \in \mathbb{R}^{d+1\times d+1}$ must be present (where d is the dimension of the space where the ERBS curve is imbedded). This matrix $M$ must initially reflect the translation and the rotation (positioning) based on the method used in (4.22). This matrix $M$ must then be used in the evaluator, multiplying the value (point) and the derivatives (vectors), where the value point must have the homogeneous coordinate 1, and all derivative vectors must have the homogeneous coordinate 0.

This way of implementing the ERBS curves will allow us to do affine transformations on the local curves, just by updating the homogeneous matrix $M$. We shall now see some examples of how to use this feature, and see some of the possibilities this gives.

The first example is based on the previous example of the "Rose-curve" approximation, where 56 interpolation points and, thus, as many local Arc curves, are used. After the approximation, all local arc curves are rotated 90° around their local x-axis, which are actually the first derivatives of the original "Rose-curve" at the interpolation points. The

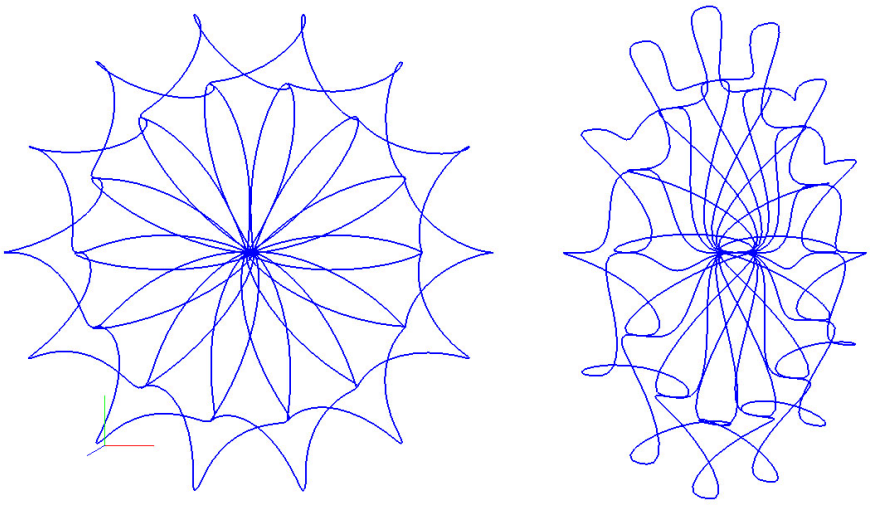

***Figure 4.21:*** *The "Rose-curve" is approximated by an ERBS curve using 56 interpolation points and local arc-curves, as in the previous examples. But in this case all local Arc curves are rotated* $90^\circ$ *around their local x-axis, which are the first derivatives at the interpolation points. In the figure, the "rotated Rose-curve" are shown from two different angles. On the right hand side we can see the curve from the side, showing a kind of depth. One can clearly see that the curve has become 3D, it is no longer flat.*

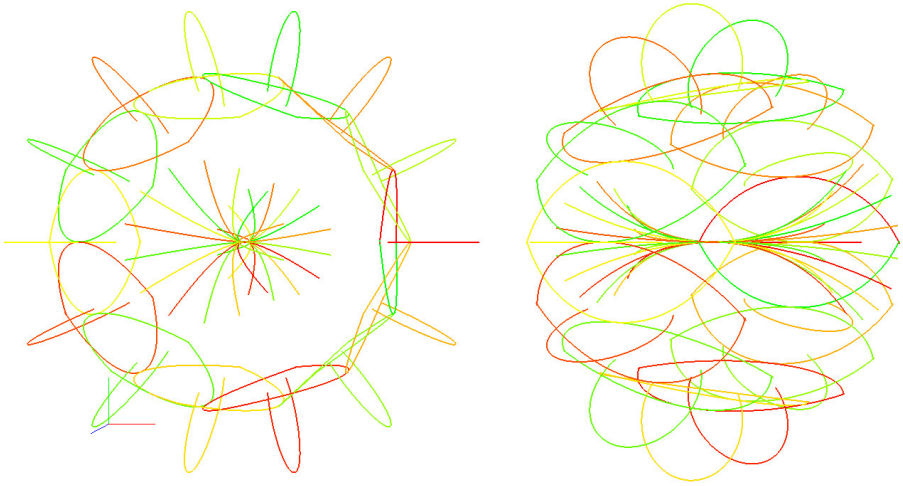

***Figure 4.22:*** *The local curves of the ERBS curve from Figure 4.21 can be seen from the same angles as in Figure 4.21. They are plotted in colors gradually changing from green to red so they can be separated.*

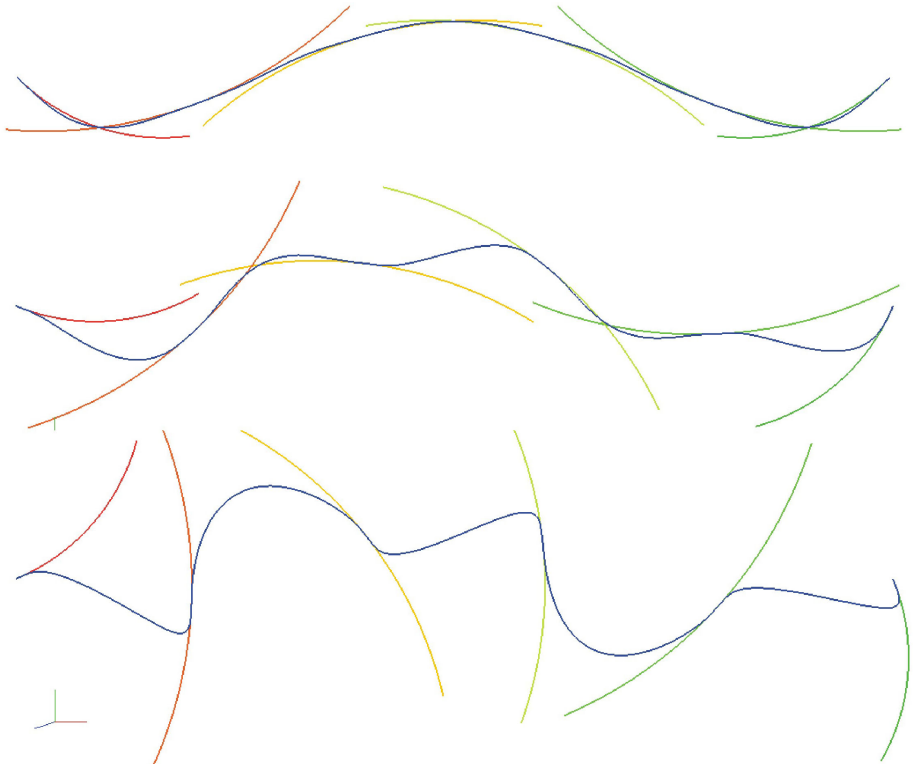

***Figure 4.23:*** *An example of a dynamical moving rope. The initial curve is a 4th degree Bézier curve, and the approximating ERBS curve uses 6 local Arc curves. First the curve is at the initial state, then the local curves start to rotating around their local z-axis. the second and the last plot shows snapshots after the rotation has started.*

result can be seen in Figure 4.21. The ERBS curve has now really become a 3D-curve, it is no longer flat. The curve has still kept the "symmetrical" property, in the way that all petals are equal. In Figure 4.22, the local Arc curves can be seen.

The obvious field where there is an opportunity to use the affine transformation of local curves, is to simulate curves that change the form dynamically, like ropes, webs, beams, etc. In the following examples we will see "snapshots" from two different examples. The first example starts with a 4th degree Bézier curve, approximated by an ERBS curve using 6 Arc curves as local curves. In Figure 4.23, there are 3 plots of this curve changing shape. Each plot includes the ERBS curve in blue, and the 6 local Arc curves with colors gradually changing from red to green. The upper plot shows the initial ERBS curve, and the next two curves show two snapshots after the rotation has started. In this example the local curves are rotating around the local z-axis (the axis "out of the paper" located at the interior knot of the ERBS associated to the local Arc curve). This example is taken from some testing done for making special effects for use in game programming.

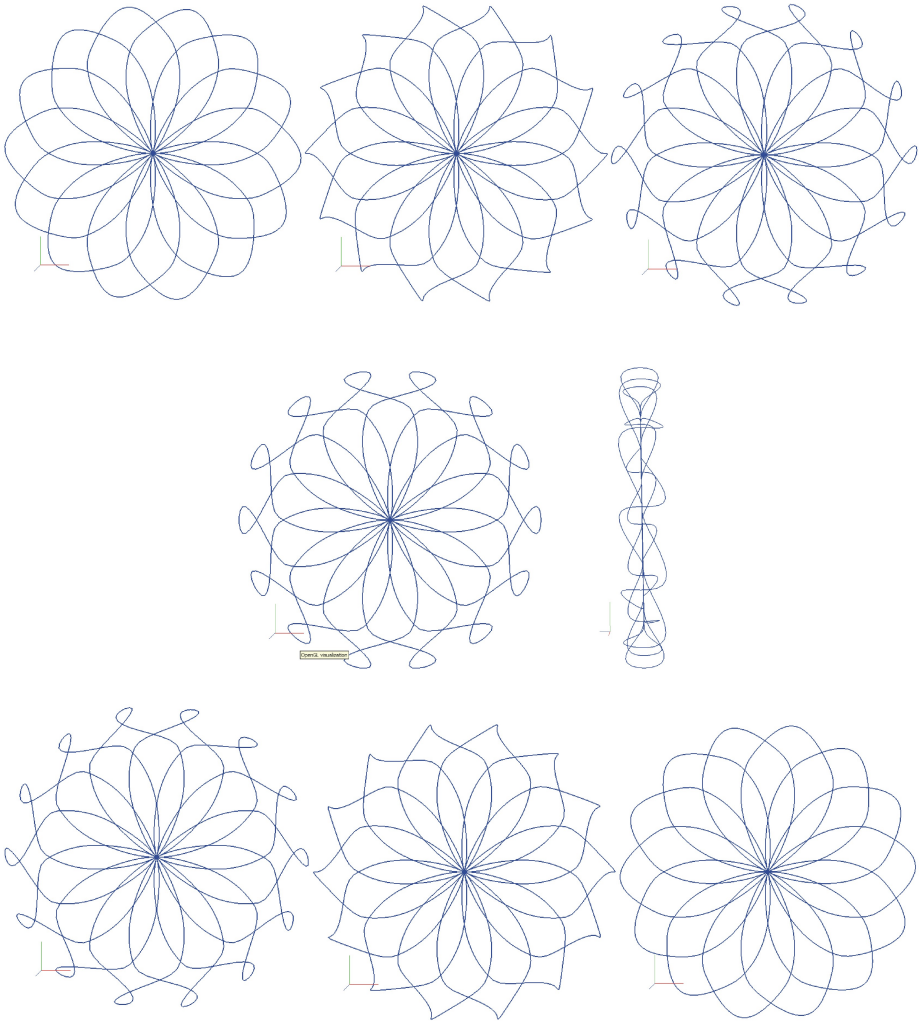

***Figure 4.24:*** *The "Rose-curve" approximated by an ERBS curve using 56 interpolation points and local arc-curves, as in the previous example. But in this case only the local Arc curves at the tip of the petals are rotated (actually, 7 times around their local y-axis which coincide with the direction of the second derivative at the interpolation point). In the figure, a total of 7 "rotated" "Rose-curves" are plotted, each rotated* $22.5°$ *more than the previous one. The fourth curve, where the local curves at the tip of the petals are rotated a total of* $90°$*, is also plotted from the side.*

The last example is the, by now well known, “Rose-curve” example, where the “Rose-curve” is interpolated by an ERBS curve using 56 local Arc curves as earlier. Only one of the local curves at each petal is rotating in this example, namely the one at the tip of each petal. In Figure 4.24, there are 7 plots showing how the curve changes during this rotation. From one plot until the next, the local curves have rotated $22.5^\circ$ around their local y-axis (the axis going through the center of the rose and the interior knot of the ERBS associated to the local Arc curve that is going to be rotated). The fourth plot also shows the curve, as seen from the side.

This is only a short description of some of the possibilities in interactive dynamic changing curves. There are examples using scaling of local curves, translations of the local curves, and combinations of two or three of the types of affine transformations.

# Chapter 5

# Tensor Product Surfaces

For tensor product surfaces we have the following general formula using Expo-Rational B-splines (ERBS),

$$S(u,v) = \sum_{i=1}^{n_u} \sum_{j=1}^{n_v} s_{ij}(u,v) B_i(u) B_j(v), \tag{5.1}$$

where $s_{ij}(u,v), \quad i = 1,..,n_u, \ j = 1,..n_v,$ are $n_u \times n_v$ local patches, and $B_i(u)$, $B_j(v)$ are the respective ERBS basis functions. As can be seen, the formula resembles the usual B-spline tensor product surface, except that $s_{ij}(u,v)$ are not points, but surfaces. An ERBS tensor product surface can therefor be regarded as a blending of local patches. An ERBS surface with $n_u \times n_v$ local patches can be divided into $(n_u - 1) \times (n_v - 1)$ "quadrilateral" parts, where each of them is a blending of parts of 4 local patches.

On the left hand side in Figure 5.1, there is a plot of a tensor product ERBS-surface with 9 local 2nd degree Bézier patches. The interpolation points, the points where the ERBS surface (later also called the global surface) completely interpolates (with all derivatives) the local patches, are marked as blue cubes. All local patches are planar and parallel, and on the right hand side in Figure 5.1, three of the local patches are plotted. The red

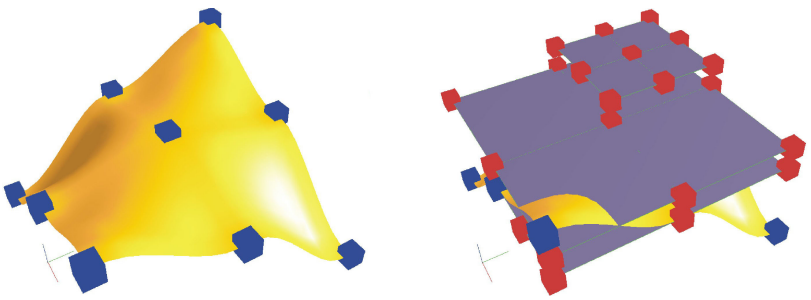

***Figure 5.1:*** *A global tensor product ERBS-surface with 9 local surfaces (interpolation points marked with blue cubes). All local surfaces are 2nd degree Bézier patches that are planar, some of them are viewed on the left hand side (the red cubes are control points).*

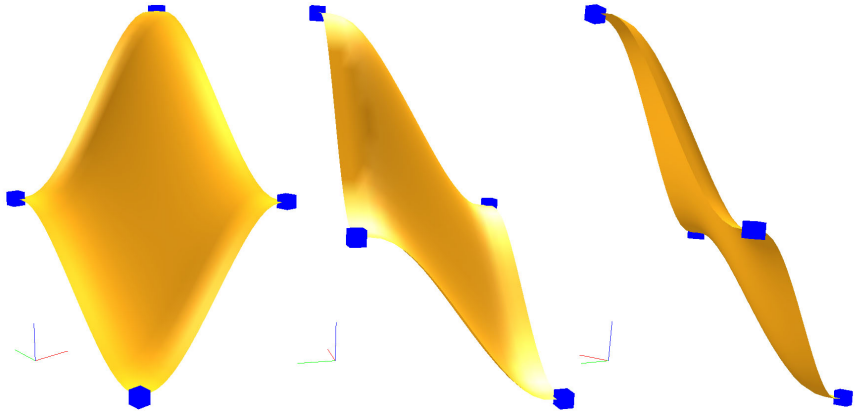

***Figure 5.2:*** *A global tensor product ERBS-surface with four local surfaces (interpolation points marked with blue cubes). All local surfaces are planar Bézier patches which are equal and also parallel. The surface is viewed from three different angles, and on the left hand side of each view, there is an RGB-frame showing the orientations.*

cubes represent the control points of the Bézier patches. The knot vector in $u$ direction is $\{u_i\}_{i=0}^4$, where $u_0 = u_1$ and $u_3 = u_4$ are the multiple start and end knots, and where $u_2$ is the non multiple inner knot. The knot vector in $v$ direction is $\{v_i\}_{i=0}^4$, where $v_0 = v_1$ and $v_3 = v_4$ are the multiple start and end knots, and where $v_2$ is the non multiple inner knot. As can be seen on the left hand side in Figure 5.1, the local Bézier patches in the corner are only covering $\frac{1}{4}$ of the global surface. One can also see that, for this example, the local Bézier patches connected to the points on the sides are covering half of the global surface, and one can clearly see that the Bézier patch connected to the point in the middle of the global surface is covering the whole global surface. In Figure 5.2 there is an ERBS surface with only 4 local surfaces. It follows that all local surfaces are covering the whole global ERBS surface. In the figure, the surface is viewed from three different angles. Note that in this case all the local patches are planar and parallel.

In definition 2.5 the classification of local function supporting the basic properties, i.e. the $C^\infty$ property, is done. For a tensor product surface we can sum up the following.

**Remark 13.** *In principle, the only constraint on the choice of the local surfaces $s_{i,j}(u,v)$ is that $|s_{i,j}(u,\bar{v})| \in \mathfrak{F}(B_i(u))$ for all $\bar{v} \in [v_1, v_{n_v}]$, and that $|s_{i,j}(\bar{u},v)| \in \mathfrak{F}(B_i(v))$ for all $\bar{u} \in [u_1, u_{n_u}]$.*

Important instances of admissible local surfaces are surfaces based on algebraic and trigonometric polynomials, rational functions, parts of a torus, etc. It is possible to use local surfaces of different types as local surfaces, and it is of particular interest to consider "multilevel" Expo-Rational B-splines. (For more information see [18]).

In the remainder of this chapter the following subjects will be discussed. In section 5.1, there are definitions of "open/closed" surfaces. Then, in section 5.2, surface evaluation including derivatives is discussed, and the section shows some algorithms for use in im-

plementations. In section 5.3 Bézier patches are discussed as local surfaces. This incudes Hermite interpolation of a given surface by a tensor product ERBS-surface using Bézier patches as local surfaces. Many examples are given. In the next section, 5.4, affine transformations on local surfaces are discussed, and how this affects the tensor product ERBS-surface. Some examples are quite spectacular. In the last section, 5.5, computational aspects concerning ERBS tensor product surfaces are discussed.

## 5.1 Definition/implementation of "open/closed" Surfaces

We regard a tensor product ERBS surface to be a "2-dimensional object", i.e. a parametrized differentiable surface, allowing self intersection, singularities and other irregularities. Although a tensor product ERBS surface almost always can be regarded as a differentiable manifold, we will nevertheless keep to the concept of non-manifold geometry.

Therefor, the definition of a surface is as following:

**Definition 5.1.** *A parametrized differentiable surface is a differentiable map* $S : \Omega \to \mathbb{R}^n$ *of an open set* $\Omega \subset \mathbb{R}^2$ *into* $\mathbb{R}^n$ *for* $n = 2, 3, \ldots$ .

With this as a background the next definitions are introduced:

**Definition 5.2.** *"Standard", "open" and "closed" are regarded as states for the two parameters for a tensor product surface. A "Standard", "open", or a "closed" state for a parameter for a tensor product surface is defined by the following restrictions:*

*i) A "standard" parameter for a tensor product surface is defined on a half open interval* $(a,b]$*, which is a restriction of a differentiable function on an open interval containing the half open interval* $(a,b]$*.*

*ii) Exclusively adding* $B_1(t_1) = 1$ *to definition 2.2, an "open" parameter for a tensor product surface is a restriction of a differentiable function on an open interval containing the closed interval* $[a,b]$*.*

*iii) A "closed" parameter is apparently also defined on the half open interval* $(a,b]$*. But because a and b are defined to be identical, then*

$$\lim_{t \to a+} D_u^j S(t,v) = D_u^j S(b,v), \quad \textit{for} \quad j = 0,1,2,\ldots \quad \textit{and} \quad v \in [v_1, v_{n_v}]$$

*or*

$$\lim_{t \to a+} D_v^j S(u,t) = D_v^j S(u,b), \quad \textit{for} \quad j = 0,1,2,\ldots \quad \textit{and} \quad u \in [u_1, u_{n_u}]$$

*follows, and the current parameter interval is actually open.*

with "standard" parameters, tensor product ERBS surfaces will be suitable as spliced surfaces. A tensor product ERBS surface with multiple knots at the start and end is an "open" surface in the appurtenant parameter direction, because the multiple knots at the start and at the end are, as for curves, closing the interval. A torus is an example of a surface where both parameters are "closed", while a cylinder is an example of a surface with only one "closed" parameter.

A special example is a sphere. A sphere is actually a cylinder with a contraction in both of the "open" ends, and it is, therefore, degenerated at the poles. It is actually possible to introduce "contraction" as a fourth state, to handle degenerated poles, prevent holes and secure a kind of "continuous" unit normal (a single map for a sphere is, as known, not possible without degenerations). "Contraction" is actually not a state for a parameter direction, but a "contraction" of all function values, including derivatives, along a straight line in the parameter plane. The straight line is restricted to be parallel with one of the coordinate axes in the parameter plane. For a sphere there typically is a "contraction" at the start value and end value of one of the parameters ($u$ or $v$), while the other parameter is "closed".

A tensor product ERBS surface is, in addition to the ERBS intrinsic parameters, determined by two knot vectors defining the two sets of basis functions, and a matrix (a 2D grid) of local surfaces. Implementing "standard/open/closed" parameters for tensor product ERBS surface is essentially the same as the implementation proposed for ERBS curves. A short repetition of the rules: if the surface has an "open" parameter, then, since a basis function is defined over two knot intervals, and there must be at least two basis functions to fulfill "the partition of unity", the minimum number of knots must be 4. Recall that in practice local patches are usually implemented as objects (C++ instance of a class), and there are references (or pointers) to these objects. In general an "open" parameter on a tensor product ERBS surface should have the following relationship between the number of knots, knot intervals, local patches, and the indexing (the proposal is identical to the one proposed for curves). The number of local patches and thus local patches in one of the parameter directions is n (actually $n_u$ or $n_v$ depending on the parameter direction), and the total number of local patches is $n_u \times n_v$.

| | | |
|---|---|---|
| Minimum number of local surfaces is: | $n = 2$ | "standard/open" parameters |
| References to basis-functions/local-surfaces: | $n$ | indexing from: 1 to $n$ |
| Number of knots: | $n+2$ | indexing from: 0 to $n+1$ |

Using this convention, the indexing of the knots and the functions are connected, which makes it easier to implement.

For a "closed" parameter in an ERBS surface, it is basically only necessary to have the same number of knots as the number of knot intervals, which equals the number of basis functions/local patches. But practically, to describe $n$ knot interval we need at least $n+1$ knots. A suggestion for a practical set is to increase both the number of knots and basis functions/local patches. The suggestion is not to increase the actual number of local patches, but to introduce an extra set of references (an extra row or column in the matrix depending on the direction). The result is that we get the following number of references (rows or columns) to local patches, and numbers of knots for a "closed" parameter for an ERBS surface. Note that this time, the number of ERBS basis functions and thus **references** to local patches in the current direction is denoted by n (actually it is $n_u$ or $n_v$).

| | | |
|---|---|---|
| Minimum number of real local surfaces is: | $n-1=1$ | for "closed" parameters |
| References to basis-functions/local-surfaces: | $n$ | indexing from: 1 to $n$ |
| Number of knots: | $n+2$ | indexing from: 0 to $n+1$ |

Note that even if the main cycle of the parameter domain is $[u_1, u_{n_u}]$, alternatively, $[v_1, v_{n_v}]$, the real number of local surfaces is $(n_u - 1) \times (n_v - 1)$.

There are two invariants connected to this implementation of"closed" parameters:

i) If u is "closed", the knot intervals between $\{u_0, u_1, u_2\}$ is just a reflection of the knot intervals between $\{u_{n_u-1}, u_{n_u}, u_{n_u+1}\}$, i.e. $u_0 = u_1 - (u_{n_u} - u_{n_u-1})$ and $u_{n_u+1} = u_{n_u} + (u_2 - u_1)$. It is an invariant that the two first knot intervals have to be equal to the two last knot intervals, and that this must be kept up if a knot value is changed. This is alternatively also valid for the **v** knot vector.

ii) For the local patches, the reference with index $n$ is the same as the reference with index 1. This is to be seen as an invariant, if one of the references is changed then the other has to be changed in the same way.

For a "closed" parameter, the basis functions just have to be computed in the same way as "open" parameters, with the same indices as the respective references to the local patches. As one can see from the table, it is actually possible to use only 1 local patch because we then only need 1 knot interval in both directions. Then, in both directions, the first half of the local function is blended with the second half.

## 5.2 Evaluation, value and derivatives

The general expression for a tensor product "ERBS surface" (5.1) and some of its partial derivatives can be computed as follows,

$$\begin{aligned}
S(u,v) &= \sum_{j=1}^{n_v}\sum_{i=1}^{n_u} s_{i,j}(u,v)B_i(u)B_j(v),\\
D_uS(u,v) &= \sum_{j=1}^{n_v}\sum_{i=1}^{n_u} \left(D_us_{i,j}(u,v)B_i(u)B_j(v) + s_{i,j}(u,v)DB_i(u)B_j(v)\right),\\
D_vS(u,v) &= \sum_{j=1}^{n_v}\sum_{i=1}^{n_u} \left(D_vs_{i,j}(u,v)B_i(u)B_j(v) + s_{i,j}(u,v)B_i(u)DB_j(v)\right),\\
D_uD_vS(u,v)) &= \sum_{j=1}^{n_v}\sum_{i=1}^{n_u} \left(D_uD_vs_{i,j}(u,v)B_i(u)B_j(v) + D_us_{i,j}(u,v)B_i(u)DB_j(v) + \right.\\
&\qquad \left. D_vs_{i,j}(u,v)DB_i(u)B_j(v) + s_{i,j}(u,v)DB_i(u)DB_j(v)\right),
\end{aligned} \tag{5.2}$$

where the notation, according to the usual notation for partial derivatives, is; $D_u = \frac{\partial}{\partial u}$, $D_v = \frac{\partial}{\partial v}$, and $D_uD_v = \frac{\partial^2}{\partial u \partial v}$. Note that $s_{ij}(u,v)$, and all respective derivative vectors have in expression (5.2) a dimension equal to the space in which the surface is embedded, while $B_i(u)$ and $B_j(v)$ and all their derivatives are scalars.

To make a generalized algorithm, however, we have to simplify. Because of the double sum, the natural choice is first to split the function into an inner and an outer loop. The inner loop is

$$c_j(u,v) = \sum_{i=1}^{n_u} s_{i,j}(u,v)B_i(u), \quad \text{for} \quad j = 1,2,...,n_v. \tag{5.3}$$

It now follows that the first line of equation (5.2) can be reformulated by using (5.3). We therefor get a new version of the first line of (5.2),

$$S(u,v) = \sum_{j=1}^{n_v} c_j(u,v)B_j(v). \tag{5.4}$$

Both equations, (5.3) and (5.4) are comparable with the curve formula (4.2). When developing an algorithm, we can develop, with some modifications, similar techniques to the one used for curves.

As in the curve example there are possibilities for simplifications. Recall that there are only two basis functions that are different from zero in the interior of a knot interval, and that these two basis functions sum up to 1. At a knot value there is actually only 1 basis function which is different from zero. Therefore, analogically to formula 2.48 we can simplify the first part of formula (5.3). For all knot intervals $j = 1,2,...,n_v - 1$,

$$c_j(u,v) = \begin{cases} s_{i,j}(u,v), & \text{if } \ u = u_i \\ s_{i+1,j}(u,v) + \big(s_{i,j}(u,v) - s_{i+1,j}(u,v)\big)\,B_i(u), & \text{if } \ u_i < u < u_{i+1} \end{cases} \tag{5.5}$$

Computing the partial derivatives of (5.5) only according to u, we can see that for $u = u_i,\ i = 1,...,n_u$, all partial derivatives with respect to $u$ are equal to the respective derivatives of the local surface, i.e.

$$D_u^{d_u} c_j(u,v) = D_u^{d_u} s_{i,j}(u,v), \quad \text{for } j = 1,...,n_u, \ \text{ and } \ d_u = 0,1,2,.... \tag{5.6}$$

To simplify for all other $u \in [u_1, u_{n_u}]$, we first define

$$\widehat{s}_{i,j}(u,v) = s_{i,j}(u,v) - s_{i+1,j}(u,v), \quad \text{if } \ u_i < u < u_{i+1}, \tag{5.7}$$

then for $u_i < u < u_{i+1}$, we get the following equation for the function value and the two first partial derivatives in the $u$ direction,

$$\begin{aligned} c_j(u,v) &= s_{i+1,j}(u,v) + \widehat{s}_{i,j}(u,v)B_i(u), \\ D_u c_j(u,v) &= D_u s_{i+1,j}(u,v) + D_u\widehat{s}_{i,j}(u,v)B_i(u) + \widehat{s}_{i,j}(u,v)DB_i(u), \\ D_u^2 c_j(u,v) &= D_u^2 s_{i+1,j}(u,v) + D_u^2\widehat{s}_{i,j}(u,v)B_i(u) + 2D_u\widehat{s}_{i,j}(u,v)DB_i(u) + \widehat{s}_{i,j}(u,v)D^2B_i(u). \end{aligned} \tag{5.8}$$

The expressions are close to being the same as what we used in the curve case (4.6). But we note that this is only a part of the computation of the inner loop. We also have to compute all the partial derivatives/mixed derivatives for $c_j(u,v)$ in the $v$ direction. But the derivatives in the $v$ direction are actually straightforward to compute, because the ERBS basis function in (5.8) is independent of $v$, and therefore, only one of the factors in all terms is dependent on $v$. The derivatives of the total lines are thus only the derivatives of each term. This can, as we will see, be used to expand formula (5.8) to a vector of vectors instead of only vectors. To expand formula (5.8) to also include derivatives in the $v$ direction we have to first look at the first line of (5.8). It follows that

$$D_v^d c_j(u,v) = D_v^d s_{i+1,j}(u,v) + D_v^d\widehat{s}_{i,j}(u,v)B_i(u), \quad \text{for } \ d = 1,2,...,$$

and that in general for computing

$$D_u^{d_u} D_v^{d_v} c_j(u,v), \quad \text{for } d_u = 1,2,... \text{ and } d_v = 1,2,...,$$

on the right hand side in expression 5.8, we just have to replace

$$\begin{aligned} &D_u^{d_u} s_{i+1,j}(u,v) \quad \text{with} \quad D_u^{d_u} D_v^{d_v} s_{i+1,j}(u,v) \quad \text{and} \\ &D_u^{d_u} \widehat{s}_{i,j}(u,v) \quad \text{with} \quad D_u^{d_u} D_v^{d_v} \widehat{s}_{i,j}(u,v). \end{aligned}$$

Actually, the total result of the inner loop must be a matrix of vectors, where both the matrix and the vectors must have the same dimension as the evaluators from the local surfaces returns, and the matrix must contain the value and all partial derivatives. We, therefore, clarify this by introducing the notation of the inner loop matrix,

$$\mathbf{C}_{j,d_u,d_v}(u,v), \tag{5.9}$$

where each element of the matrix is a vector $\in \mathbb{R}^n$, where n is the dimension of the Euclidian space the surface is imbedded in. The index $j$ is related to the knot interval, and $d_u$ denotes the number of derivatives there are in the $u$ direction, and $d_v$ denotes the number of derivatives there are in the $v$ direction. An example of this matrix, where $d_u = 2$ and $d_v = 2$, is

$$\mathbf{C}_{j,2,2}(u,v) = \begin{bmatrix} c_j(u,v) & D_v c_j(u,v) & D_v^2 c_j(u,v) \\ D_u c_j(u,v) & D_u D_v c_j(u,v) & D_u D_v^2 c_j(u,v) \\ D_u^2 c_j(u,v) & D_u^2 D_v c_j(u,v) & D_u^2 D_v^2 c_j(u,v) \end{bmatrix}. \tag{5.10}$$

The notation of the equivalent matrix from the local patches is

$$\widetilde{\mathbf{S}}_{i,j,d_u,d_v}(u,v), \tag{5.11}$$

where $i$ and $j$ are in the indices of the local patch, and $d_u$ and $d_v$ denote the number of derivatives there are in the respective $u$ and $v$ directions. These matrices are also organized as (5.10). An example of this matrix, where $d_u = 2$ and $d_v = 2$, is

$$\widetilde{\mathbf{S}}_{i,j,d_u,d_v}(u,v) = \begin{bmatrix} s_{i,j}(u,v) & D_v s_{i,j}(u,v) & D_v^2 s_{i,j}(u,v) \\ D_u s_{i,j}(u,v) & D_u D_v s_{i,j}(u,v) & D_u D_v^2 s_{i,j}(u,v) \\ D_u^2 s_{i,j}(u,v) & D_u^2 D_v s_{i,j}(u,v) & D_u^2 D_v^2 s_{i,j}(u,v) \end{bmatrix}. \tag{5.12}$$

To sum up before constructing an algorithm: The first column in (5.9–5.10) is equal to the left hand side of (5.8), and only elements from the first column in (5.11–5.12) are used on the right hand side of (5.8). To compute the other columns we have to replace elements from the first column in (5.11) with respective elements from the other columns. The conclusion is, therefore, that we only have to replace individual elements from the first column of (5.11), that is, on right hand side of (5.8), with the respective rows of (5.11), to expand (5.8) to return (5.9).

**Remark 14.** *It is, of course, possible and usual, to construct an evaluator which only returns the upper left half of the matrix (5.9). The advantage of a construction like this is to optimize speed, because one often only needs the upper left part of the matrix. The algorithm in this section, however, will focus on computing the whole matrix, but it is fairly simple to modify the algorithm to only compute the upper left part of the matrix.*

The algorithm now becomes quite similar to algorithm 11 used for curves. It depends on the ERBS evaluator and on reliable evaluators for local surfaces (evaluator for Bezier surfaces will be discussed later in this chapter). As for curves, we assume that the surfaces are embedded in an Euclidian space, where the dimension normally is 3, but it could also be something else, so the vector type is, therefore, denoted T (typical template type in C++ ). The global/local affine mapping, definition 2.7, is not used in the following algorithm, but it must be used in the local patch evaluators (see algorithm 9). The big difference from curves is that the algorithm returns a matrix (of vectors) instead of a vector (of vectors), and that the evaluator for local surfaces also returns matrices.

See remark 11 in connection with the ERBS-curve evaluator. This remark is, of course, also valid for a tensor product ERBS-surface evaluator. It follows that algorithm 10 also has to be used in the tensor product ERBS-surface evaluator. We can now introduce an algorithm for the inner loop of the tensor product surface evaluator:

**Algorithm 13.** *(For notation, see section "Algorithmic Language", page 7.)*
*The algorithm computes the matrix $\mathbf{C}_{j,d_u,d_v}(u,v)$ defined in (5.10). It is assumed that evaluators for the ERBS basis function, and the local patches are present, and that these evaluators return a matrix $\widetilde{\mathbf{S}}_{i,j,d_u,d_v}(u,v)$ defined in (5.11) and analogous to (5.10). The knot vectors $\{u_i\}_{i=0}^{n_u+1}$ and $\{v_i\}_{i=0}^{n_v+1}$ are also supposed to be present. The input variables are: $u \in [u_1, u_{n_u}]$, $v \in [v_1, v_{n_v}]$, and $k_u$ :\; $u_{k_u} \le u < u_{k_u+1}$, $k_v$ :\; $v_{k_v} \le v < v_{k_v+1}$, and $d_u \in \{0,1,2,...,p\}$ (the number of derivatives in u direction) and $d_v \in \{0,1,2,...,p\}$ (the number of derivatives in v direction), where p depends on the ERBS-evaluator. The return is a* "Matrix⟨T⟩"*, where T is a n-dimensional vector matching $c_j(u,v)$ (see 5.8).*

```
Matrix⟨T⟩ C ( double u, double v, int k_u, int k_v, int d_u, int d_v )
    Matrix⟨T⟩ C_0 = S̃_{k_u,k_v,d_u,d_v}(u,v);      // Result evaluating local patch - (k_u,k_v)
    if (u == u_{k_u})    return C_0;               // Return only local patch, see (5.5)
    Matrix⟨T⟩ C_1 = S̃_{k_u+1,k_v,d_u,d_v}(u,v);    // Result evaluating local patch - (k_u+1,k_v)
    vector⟨double⟩ a(d_u+1);                       // For numbers - "Pascals triangle"
    vector⟨double⟩ B = B̄(u,k_u,d_u);               // Result evaluating ERBS-basis, (Alg. 10)
    C_0 -= C_1;                                    // The matrix c_0 is now ĉ_0, expanded (5.7)
    for ( int i=0; i ≤ d; i++ )
        a_i = 1;
        for ( int j=i-1; j > 0; j-- )
            a_j += a_{j-1};                        // Computing "Pascals triangle"-numbers
        for ( int j=0; j ≤ i; j++ )
            C_{1,i} += (a_j B_j) C_{0,i-j};        // "row += scalar*row", (5.8)
    return C_1;
```

Note that the algorithm is updating whole rows of the matrix $C_1$ by summing up a row

with scaled rows of matrix $\widehat{C}_0$. This is directly followed by the fact that equation (5.8) is expanded to compute not only the first column of (5.10), but all columns, and that the ERBS basis function is only dependent on $u$.

We can now look at the outer loop. The main tensor product surface evaluator is, as we can see, an expanded version of equation (5.4). This equation is equivalent to equation (5.3), therefore, analogical to formula 2.48. We can simplify the first part of formula (5.4). It follows that for all knot intervals $j = 1, 2, ..., n_v - 1$,

$$S(u,v) = \begin{cases} c_j(u,v), & \text{if } \ v = v_j, \\ c_{j+1}(u,v) + \big(c_j(u,v) - c_{j+1}(u,v)\big)\, B_j(v), & \text{if } \ v_j < v < v_{j+1}. \end{cases} \tag{5.13}$$

Computing the partial derivatives of (5.13) only according to v, we can see that for $v = v_j,\ j = 1, ..., n_v$, all partial derivatives of the ERBS-surface with respect to v are equal to the respective derivatives from the inner loop, i.e.

$$D_v^{d_v} S(u,v) = D_v^{d_v} c_j(u,v), \qquad \text{for } j = 1, ..., n_u, \ \text{ and } \ d_v = 0, 1, 2, ....$$

To simplify for all other $v$ values in $[v_1, v_{n_v}]$, we first define

$$\widehat{c}_j(u,v) = c_j(u,v) - c_{j+1}(u,v), \quad \text{if } \ v_j < v < v_{j+1},$$

then for $v_j < v < v_{j+1}$ we get the following equation for the function value and the derivatives in the v direction,

$$\begin{aligned} S(u,v) &= c_{j+1}(u,v) + \widehat{c}_j(u,v) B_i(v) \\ D_v S(u,v) &= D_v c_{j+1}(u,v) + \widehat{c}_j(u,v) DB_j(v) + D_v \widehat{c}_j(u,v) B_j(v) \\ D_v^2 S(u,v) &= D_v^2 c_{j+1}(u,v) + \widehat{c}_j(u,v) D^2 B_j(v) + 2 D_v \widehat{c}_j(u,v) DB_j(v) + D_v^2 \widehat{c}_j(u,v) B_j(v). \end{aligned} \tag{5.14}$$

Studying (5.14) we can see that it has the same structure as (5.8). But as for the inner loop, the total result of the outer loop must be a matrix of vectors, where both the matrix and the vectors must have the same dimension as the algorithm for computing the inner loop returns, and the matrix must contain the value and all partial derivatives. We, therefore, clarify this by introducing the notation of the matrix of the surface evaluator,

$$\mathbf{S}_{d_u, d_v}(u,v), \tag{5.15}$$

where each element of the matrix is a vector $\in \mathbb{R}^n$, where n is the dimension of the Euclidian space the surface is imbedded in. The index $d_u$ denotes the number of derivatives there are in $u$ direction, and $d_v$ denotes the number of derivatives there are in $v$ direction. An example of this matrix, where $d_u = 2$ and $d_v = 2$, is:

$$\mathbf{S}_{2,2}(u,v) = \begin{bmatrix} S(u,v) & D_v S(u,v) & D_v^2 S(u,v) \\ D_u S(u,v) & D_u D_v S(u,v) & D_u D_v^2 S(u,v) \\ D_u^2 S(u,v) & D_u^2 D_v S(u,v) & D_u^2 D_v^2 S(u,v) \end{bmatrix}. \tag{5.16}$$

Finally, when preparing for the main algorithm of the tensor product ERBS surface evaluation, and thus the outer loop, note first that on the left hand side of (5.14) we do not have the first column in the resulting matrix (5.15–5.16), but we have the first *row*. In addition, on the right hand side of (5.14) we only find elements from the first *row* of the matrix $C_{j,d_u,dv}(u,v)$ (5.9–5.10). The algorithm becomes quite similar to algorithm 11 used for curves, and to algorithm 13 for the inner loop. It depends on the ERBS evaluator and on the inner loop. As it was for the inner loop algorithm, we assume that the surfaces are embedded in an Euclidian space, where the dimension normally is 3, but it could also be something else, so the vector type is, therefore, denoted T (typical template type in C++ ). The big difference from the inner loop algorithm is that we, in the next to the last line of the algorithm, have to sum up matrix columns instead of rows.

**Algorithm 14.** *(For notation, see section "Algorithmic Language", page 7.)*
*The algorithm computes the matrix* $\mathbf{S}(u,v)$ *equivalent to (5.10) and described in (5.2). The algorithm assumes that evaluators for the local patches and the ERBS basis function are present, The evaluators for the local patches must return a matrix* $\widetilde{\mathbf{S}}_{i,j}(u,v)$ *described in (5.11). The knot vectors* $\{u_i\}_{i=0}^{n_u+1}$ *and* $\{v_i\}_{i=0}^{n_v+1}$ *are supposed to be present. The input variables are:* $u \in [u_1, u_{n_u}]$, $v \in [v_1, v_{n_v}]$, *and* $d_u \in \{0,1,2,...,p\}$ *(the number of derivatives in u direction), and* $d_v \in \{0,1,2,...,p\}$ *(the number of derivatives in v direction), where p depends on the ERBS-evaluator. The return is a* "Matrix⟨T⟩"*, where T is an n-dimensional vector matching* $S(u,v)$*, and where the elements in the matrix are matching the elements in the matrix* $C_{j,d_u,d_v}(u,v)$ *described in (5.10).*

Matrix⟨T⟩ *eval (* double *u,* double *v,* int $d_u$, int $d_v$ *)*
- int $k_u = k_u :\backslash;\ u_{k_u} \le u < u_{k_u+1};$ // *Index for the current knot-interval for u.*
- int $k_v = k_v :\backslash;\ v_{k_v} \le v < v_{k_v+1};$ // *Index for the current knot-interval for v.*
- Matrix⟨T⟩ $S_0 = \mathbf{C}_{k_u,k_v,d_u,d_v}(u,v);$ // *Result from inner loop - $k_v$.*
- if $(v == v_{k_v})$ return $S_0;$ // *Return only inner loop, see (5.13).*
- Matrix⟨T⟩ $S_1 = \mathbf{C}_{k_u,k_v+1,d_u,d_v}(u,v);$ // *Result from inner loop - $k_v+1$.*
- vector⟨double⟩ *a(d+1);* // *For numbers - "Pascals triangle".*
- vector⟨double⟩ $B = \overline{B}(t,k,d);$ // *Result evaluating ERBS-basis, (Alg. 10).*
- $S_0 -= S_1;$ // *$C_0$ is now $\widehat{C}_0$, the whole matrix, see (5.6).*
- for *(* int *i=0; $i \le d$; i++ )*
  - $a_i = 1;$
  - for *(* int *j=i-1; $j > 0$; $j--$ )*
    - $a_j += a_{j-1};$ // *Computing "Pascals triangle"-numbers.*
  - for *(* int *j=0; $j \le i$; j++ )*
    - $(S_1^T)_i += (a_j\, B_j)(S_0^T)_{i-j};$ // *"column += scalar×column", (5.14).*
- return $S_1;$

The computational cost of evaluating a tensor product ERBS-surface is as we can see, evaluating two ERBS basis functions, four local surfaces, and passing a total of three times through the summing loop in the last half of both the inner and outer loop. The most expensive computational part is to evaluate the four local surfaces. This can take more than $\frac{9}{10}$ of the time, depending on the type of local surface.

## 5.3 Bézier patches as local patches

In general, Bézier patches are very convenient to use as local patches. Recall that Bézier patches are defined by

$$s(u,v) = \sum_{i=0}^{d_u} \sum_{j=0}^{d_v} c_{i,j}\, b_{d_u,i}(u) b_{d_v,j}(v) \quad \text{for } 0 \le u \le 1 \text{ and } 0 \le v \le 1, \tag{5.17}$$

where the basis functions are the Bernstein polynomials

$$b_{d,i}(x) = \left( \begin{array}{c} d \\ i \end{array} \right) x^i (1-x)^{d-i},$$

and where $c_{i,j} \in \mathbb{R}^n$, are the coefficients where $n > 0$ usually is 3.

All types of evaluators used for Bézier curves are of course also available for Bézier tensor product surfaces. In equation (4.8) the generalized Bernstein/Hermite matrix was introduced, and in algorithm 12 an algorithm to make the same matrix is developed and described. In the generalized Bernstein/Hermite matrix, each row is scaled by $\delta^j$, where $j$ is the row number (starting with 0). The matrix looks like this:

$$\mathbf{B}_d(t,\delta) = \left( \begin{array}{ccc} \delta^0 D^0 b_{d,0}(t) & \dots & \delta^0 D^0 b_{d,d}(t) \\ \vdots & \ddots & \vdots \\ \delta^d D^d b_{d,0}(t) & \cdots & \delta^d D^d b_{d,d}(t) \end{array} \right). \tag{5.18}$$

In the curve case, the matrix was used both for evaluation (preevaluation) and Hermite interpolation. It is also natural to do this for tensor product ERBS surfaces.

**Remark 15.** *There is now (year 2006) a new "reality" appearing when it comes to efficient programming codes, because of the introduction of dual/quatro/... cores processors, streaming technology, and general new processor architectures. Parallelizing/streaming is particularly easy and natural to do in Matrix computation, and thus, the optimization follows. Therefore, using Matrix computations in evaluators might seem to be a "slow overkill", but in reality, with the new architecture, it absolutely is not.*

Hence, using the matrix, (5.18), we can first make an expanded matrix version of equation (5.17), not only returning the value, but also all the derivatives. We now get:

$$\widetilde{\mathbf{S}}_{d_u,d_v}(u,v) = \mathbf{B}_{d_u}(u,\delta_u)\, C\, \mathbf{B}_{d_v}(v,\delta_v)^T \quad \text{for } 0 \le u \le 1 \text{ and } 0 \le v \le 1, \tag{5.19}$$

where $C$ is the control polygon (matrix), and $\delta_u$ and $\delta_v$ are the scaling as a result of the affine global/local mapping (definition 2.7). If $d_u = d_v = 2$ the matrix $\widetilde{\mathbf{S}}_{d_u,d_v}(u,v)$ will be

$$\widetilde{\mathbf{S}}_{2,2}(u,v) = \left[ \begin{array}{ccc} s(u,v) & D_v s(u,v) & D_v^2 s(u,v) \\ D_u s(u,v) & D_u D_v s(u,v) & D_u D_v^2 s(u,v) \\ D_u^2 s(u,v) & D_u^2 D_v s(u,v) & D_u^2 D_v^2 s(u,v) \end{array} \right]. \tag{5.20}$$

As one can see, this matrix (5.20) contains the position and all derivatives that are not 0 at the parameter value (u,v) on the surface. It follows that this matrix, together with the scaling factors $\delta_u$ and $\delta_v$, completely describes the patch.

### 5.3.1 Local Bézier patches and Hermite interpolation

We start by recalling the settings from section 2.8, and adapting them to tensor product ERBS surfaces.

- Given is a surface $g(u,v)$, $g:\Omega\subset\mathbb{R}^2\to\mathbb{R}^n$, where $\Omega=[u_s,u_e]\times[v_s,v_e]$, and where we might have $n=1,2,3,...,$
- given is the numbers of samples $m_u>1$ and $m_v>1$ and the number of derivatives $\{d_{u,i}\}_{i=1}^{m_u}>0$, and $\{d_{v,i}\}_{i=1}^{m_v}>0$ in each of the sampling points, to be used in the interpolation.
- Generate a knot vector by:
  - first setting $u_1=u_s$ and $v_1=v_s$, i.e. the start of the domain of $g$ in both $u$ and $v$ direction,
  - then setting $u_{m_u}=u_e$ and $v_{m_v}=v_e$, i.e. the end of the domain of $g$ in both $u$ and $v$ direction.
  - Then for $i=2,3,...,m_u-1$, generate $u_i$ so that $u_{i-1}<u_i$ and where $u_{m_u-1}<u_{m_u}$, and for $j=1,...,m_v$, generate $v_j$ so that $v_{j-1}<v_j$ and where $v_{m_v-1}<v_{m_v}$.
  - Finally, $u_0$ and $u_{m_u+1}$ and also $v_0$ and $v_{m_v+1}$ must be set according to the rules for "open/closed" parameters for tensor product surfaces.
- Make an ERBS-surface $S(u,v)$ using the knot vectors $\{u_i\}_{i=0}^{m_u+1}$ and $\{v_i\}_{i=0}^{m_v+1}$, and generate local patches, in such a way that the ERBS-surface is interpolating the local patches, so that, for $i=1,...,m_u$ and $j=1,...,m_v$,

$$D_u^{s_u}D_v^{s_v}S(u_i,v_j)=D_u^{s_u}D_v^{s_v}g(u_i,v_j),\quad \text{for } s_u=0,...,d_{u,i} \text{ and } s_v=0,...,d_{v,j}.$$

This looks like a general Hermite interpolation method used for generating an approximation of a parametric surface. The specific thing is the generation of the local patches. First recall that the domain of a Bézier patch is $[0,1]\times[0,1]$. Then invoke Theorem 2.4, but adjust it with the local domain for Bézier patches. We now get, for $i=1,...,m_u$ and $j=1,...,m_v$,

$$D_u^{s_u}D_v^{s_v}S(u_i,v_j)=\delta_{u,i}^{s_u}\delta_{v,j}^{s_v}\,D_u^{s_u}D_v^{s_v}s_{i,j}\circ(\omega_i(u_i),\omega_j(v_i)),\ \text{for } s_u=0,...,d_{u,i} \text{ and } s_v=0,...,d_{v,j},$$

where $S(u,v)$ is the tensor product ERBS-surface, $s_{i,j}(u,v)$ now are Bézier patches, and

$$\omega_i(u_i)=\frac{u_i-u_{i-1}}{u_{i+1}-u_{i-1}},$$

and

$$\omega_j(v_j)=\frac{v_j-v_{j-1}}{v_{j+1}-v_{j-1}},$$

are the affine global/local mappings from definition 2.7, and

$$\delta_{u,i}=\frac{1}{u_{i+1}-u_{i-1}}\quad \text{for } i=1,2,...,m_u,$$

and

$$\delta_{v,j} = \frac{1}{v_{j+1} - v_{j-1}} \quad \text{for} \quad j = 1,2,...,m_v,$$

are the domain scaling factors defined in Theorem 2.4. All this shows that the tensor product ERBS-surface is adjusted by the domain scaling factor, interpolating the local patches $s_{i,j}(u,v)$ for all derivatives at the knot nodes $(u_i, v_j)$ for $i = 1,...,m_u$ and $j = 1,...,m_v$. We look at the equation for the Hermite interpolations for a local Bézier patch with the pair of indices (i,j),

$$\begin{aligned} D_u^{s_u} D_v^{s_v} g(u_i, v_j) &= D_u^{s_u} D_v^{s_v} S(u_i, v_j) \\ &= \delta_{u,i}^{s_u} \delta_{v,j}^{s_v} \, D_u^{s_u} D_v^{s_v} s_{i,j} \circ (\omega_i(u_i), \omega_j(v_i)) \\ &= \sum_{r=0}^{d_1} \sum_{s=0}^{d_2} c_{i,j,r,s} \, \delta_{u,i}^{s_u} \, D^{s_u} b_{d_u,r} \circ \omega_i(u_i) \delta_{v,j}^{s_v} \, D^{s_v} b_{d_v,s} \circ \omega_j(v_j) \\ &\qquad\qquad \text{for} \;\; s_u = 0,...,d_u, \;\text{and}\; s_v = 0,...,d_v. \end{aligned}$$

This is nearly the same as the formulation in (5.19), but we now have included the global/local mapping. The matrix form now is, for $i = 1,...,m_u$, and $j = 1,...,m_v$,

$$\mathbf{g}_{d_u,d_v}(u_i, v_j) = \mathbf{B}_{d_u}(\omega_i(u_i), \delta_{u,i}) \, C_{i,j} \, \mathbf{B}_{d_v}(\omega_j(v_j), \delta_{v,j})^T, \tag{5.21}$$

where

$$\mathbf{g}_{d_u,d_v}(u_i, v_j) = \begin{pmatrix} g(u_i, v_j) & \dots & D_v^{d_v} g(u_i, v_j) \\ \vdots & \ddots & \vdots \\ D_u^{d_u} g(u_i, v_j) & \cdots & D_u^{d_u} D_v^{d_v} g(u_i, v_j) \end{pmatrix}.$$

Then the final step to generate the local Bézier patches is to solve equation 5.21 according to the Bézier coefficients (control polygon) $\mathbf{C}_{i,j}$,

$$C_{i,j} = \mathbf{B}_{d_u}(\omega_i(u_i), \delta_{u,i})^{-1} \, \mathbf{g}_{d_u,d_v}(u_i, v_j) \, \mathbf{B}_{d_v}(\omega_j(v_j), \delta_{v,j})^{-T}.$$

The conclusion is that, in order to compute the coefficient to the local Bézier-patches, one has to compute the expanded Bernstein/Hermite matrix using algorithm 12, and then invert this matrix and multiply the inverted matrix with the "evaluation"-matrix from the original surface. The matrix inversion will not be dealt with futher here, but there are a lot of available programming libraries including optimized algorithms for matrix inversions, see, e.g., [59].

**Remark 16.** *Note that the three matrices on the left hand side in 5.21 are not of the same "type". The middle one, $C_{i,j}$ is a matrix of points in $\mathbb{R}^n$, where n usually is 3. The Bernstein/Hermite matrix is a standard matrix where each element is a scalar. For both a surface evaluator and the Hermite interpolation, it is ,therefore, of great interest to implement a matrix template type that has overloaded matrix multiplication including multiplication between a matrix of scalars and a matrix of vectors.*

As in the curve case, there are several reasons why it is advantageous to translate all coefficients so that the interpolation point is in the local origin. Then it follows that we have to subtract the point $g(u_i, v_j)$ from all the coefficient vectors in the control polygon $C_{i,j}$ of the local Bézier-patches, and that we have to cancel this by inserting the opposite movement to the graphical homogeneous matrix system. The premise is, of course, that this homogeneous matrix system is involved in the total evaluator.

### 5.3.2 Examples of Hermite interpolations

The purpose of this subsection is to give an idea as to how this unique Hermite interpolation property works, and some of the possibilities it gives. It is not possible to give a very comprehensive view of this, because the possibilities are not really explored in the depth yet, but the following examples will hopefully give some ideas.

In this subsection we will see examples of Hermite interpolation of well-known parametric surfaces by tensor product ERBS surfaces with local Bézier patches. In some of the examples we will see some of the local Bézier patches, and take a closer look at how they are constructed by the Hermite interpolation.

The first example is based on a surface called "Trianguloid Trefoil". This surface was constructed by Roger Bagula and can be found at [1]. The formula is

$$s(u,v) = \begin{pmatrix} 2\frac{\sin(3u)}{2+\cos v} \\ 2\frac{\sin u+2\sin(2u)}{2+\cos\left(v+\frac{2}{3}\pi\right)} \\ \frac{(\cos u-2\cos(2u))(2+\cos v)\left(2+\cos\left(v+\frac{2}{3}\pi\right)\right)}{4} \end{pmatrix} \quad \begin{array}{l} \text{for} \quad u \in (-\pi,\pi], \\ \text{and} \quad v \in (-\pi,\pi]. \end{array} \tag{5.22}$$

In Figure 5.3 there is a plot of a tensor product ERBS surface interpolating a "Trianguloid Trefoil" surface (5.22) at $5 \times 5$ points. At each point the position and a total of 8 partial derivatives are used, i.e. the matrix $\mathbf{g}_{d_u,d_v}(u_i,v_j)$ defined in (5.18) has the dimension $3 \times 3$. In the Figure 5.3 the interpolation points are marked as blue cubes, and most of them can be clearly seen. The surface is "closed" in both parameters, and it is quite complex in shape, but the reconstruction has kept the structure and form faerly well. It is quite remarkable to reconstruct a complex surface as this one by only using 25 positions (including derivatives)!

The second example is based on a surface called "Bent Horns", also constructed by Roger Bagula and can be found at [2]. The formula for this surface is

$$s(u,v) = \begin{pmatrix} (2+\cos u)\left(\frac{v}{3}-\sin v\right) \\ \left(2+\cos\left(u-\frac{2}{3}\pi\right)\right)(\cos v-1) \\ \left(2+\cos\left(u+\frac{2}{3}\pi\right)\right)(\cos v-1) \end{pmatrix} \quad \begin{array}{l} \text{for} \quad u \in (-\pi,\pi], \\ \text{and} \quad v \in (-2\pi,2\pi]. \end{array} \tag{5.23}$$

In Figure 5.4 there is a plot of a tensor product ERBS surface interpolating a 'Bent Horns" surface (5.23) at $5 \times 5$ points. Also in this example, the position and a total of 8 partial derivatives are used at each point, i.e. the matrix $\mathbf{g}_{d_u,d_v}(u_i,v_j)$ defined in (5.18) has the dimension $3 \times 3$. In the Figure 5.4 the interpolation points are marked as blue cubes, and some of them can be clearly seen. The surface is "closed" in one parameter, but "open" in the other parameter. This cannot actually be seen, because the two "open" ends are squeezed into two separate edges. They can be seen at the front on either side, and are each marked by three blue cubes. These cubes are, in fact 5 cubes (one on the tip towards the center, and two coincident cubes on each of the other two). The surface is also irregular in the center, where it collapses to a point. The ERBS surface has managed

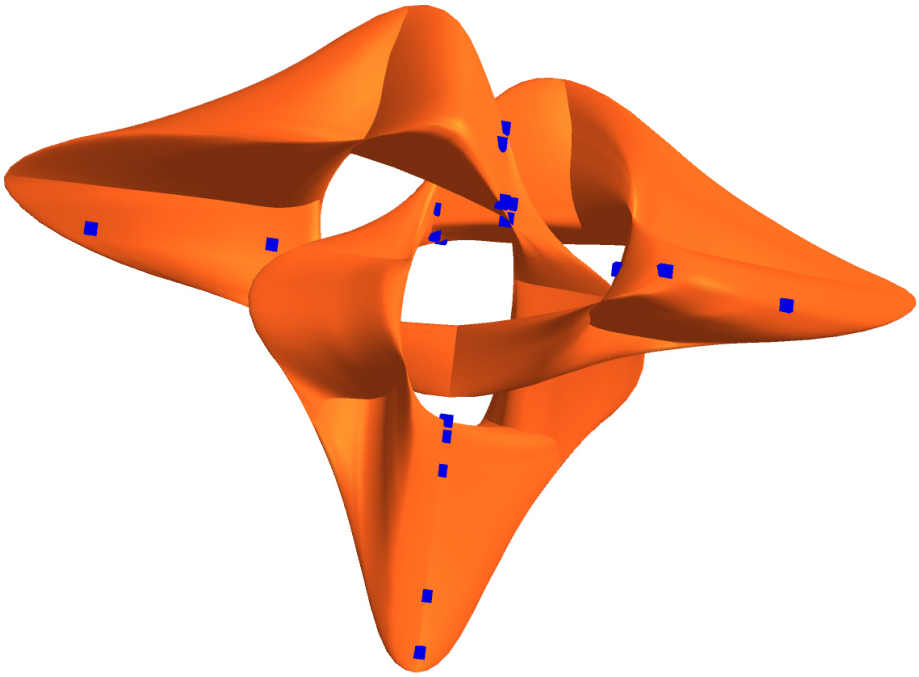

***Figure 5.3:*** *This Expo-Rational B-spline tensor product surface is made by Hermite interpolation of a "Trianguloid Trefoil" surface [1] at* $5 \times 5$ *points. The positions of the interpolating points are seen as cubes.*

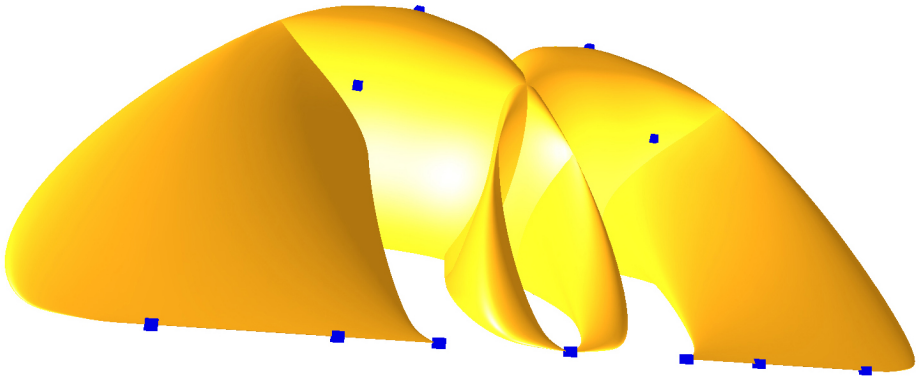

***Figure 5.4:*** *This Expo-Rational B-spline tensor product surface is made by interpolating (position, first and second derivative) a "Bent Horns" surface [2] at* $5 \times 5$ *points. The positions of the interpolating points are seen as cubes.*

to keep this irregularity intact because there is an interpolation point (actually, 5 points) at this point.

The third example is based on a sphere, where the formula is as follows,

$$s(u,v) = \begin{pmatrix} r\cos u\ \cos v \\ r\sin u\ \cos v \\ r\sin v \end{pmatrix} \quad \begin{array}{l} \text{for} \quad u \in (0, 2\pi], \\ \text{and} \quad v \in (-\frac{\pi}{2}, \frac{\pi}{2}]. \end{array} \tag{5.24}$$

In the equation $r$ is the radius of the sphere. Figure 5.5 is a plot of a tensor product ERBS surface interpolating a sphere (5.24) at $4 \times 4$ points. Also in this example, the position and a total of 8 partial derivatives are used at each point, i.e., the matrix $\mathbf{g}_{d_u,d_v}(u_i, v_j)$ defined in (5.18) has the dimension $3 \times 3$. In Figure 5.5 the interpolation points are marked as blue cubes. At the top we can see one cube, but there are actually 4 cubes in the same position. The surface is, therefore, irregular, and actually collapses to a point at both poles. In Figure 5.6 there is a plot of the same tensor product ERBS surface as in Figure 5.5, but now the plot includes one of the local Bézier surfaces located at the "north pole". There is one shaded picture on the left hand side, and one wireframe picture on the right hand side. As we can see, especially in the wireframe picture, the local surface only has two corners. The other two corners have collapsed to one point, and are lying on the apparently "smooth" edge at the "north pole". Also the edge between the two corners has completely collapsed to the same point. The ERBS surface has 8 local Bézier patches, which can be found at the two poles. They are all equal in form compared to the Bézier patch shown in the figure (but rotated and/or translated). In Figure 5.7 there is another plot of the same tensor product ERBS surface as in the figures 5.5 and 5.6, this time including one of the local Bézier surfaces that is not located at the poles. The figure is rotated to the left, so that the "north pole" is on the left hand side. The local surface looks quite complex, and probably unlike what can be expected. There are a total of 8 local Bézier surfaces at the tensor product ERBS surface which have the same shape as this local surface.

The fourth example is based on a Torus, where the equation is

$$s(u,v) = \begin{pmatrix} \cos u(R + r\cos v) \\ \sin u(R + r\cos v) \\ r\sin v \end{pmatrix} \quad \begin{array}{l} \text{for} \quad u \in (0, 2\pi], \\ \text{and} \quad v \in (0, 2\pi]. \end{array} \tag{5.25}$$

Here $r$ is the small radius, the radius in the tube, and $R$ is the big radius, the radius of the tube. In Figure 5.8 there is a plot of a tensor product ERBS surface interpolating a torus (5.25) at $5 \times 5$ points. As in the earlier examples, the position and a total of 8 partial derivatives are used at each point, i.e. the matrix $\mathbf{g}_{d_u,d_v}(u_i, v_j)$ defined in (5.18) has the dimension $3 \times 3$. In Figure 5.8 the interpolation points are marked as blue cubes, and some of them can clearly be seen. In Figure 5.9, one of the local Bézier surfaces is also plotted together with the ERBS surface. This surface models a part of a torus quite well. In Figure 5.9 the local Bézier patch is moved slightly upwards, so we can see it more clearly. In the figure the surfaces are plotted in two positions, so it is possible for us to

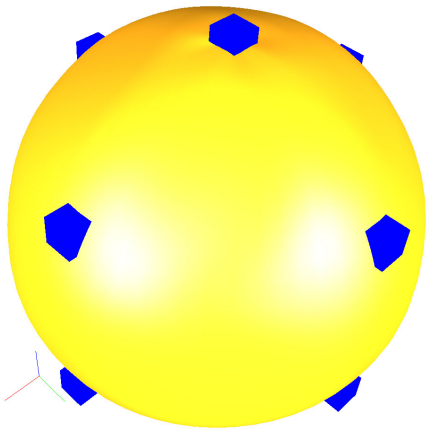

***Figure 5.5:*** *This Expo-Rational B-spline tensor product surface is made by interpolating a sphere at $4 \times 4$ points (position, first and second derivative). The positions of the interpolating points are seen as blue cubes.*

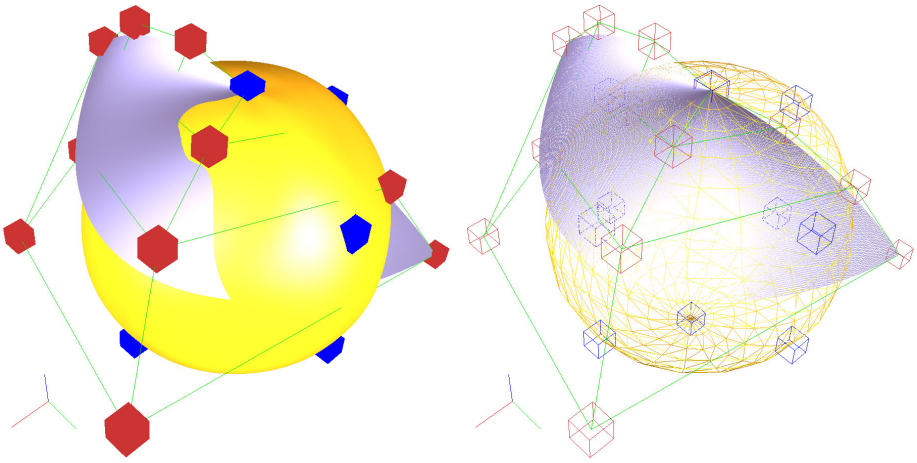

***Figure 5.6:*** *A plot of the approximated sphere from Figure 5.5, with one of the local Bézier patches located at the "north pole". The control polygon of the Bézier patch is marked in green, and the nodes in the polygon are marked as red cubes. On the right hand side there is a wireframe version of the figures that we can see on the left hand side.*

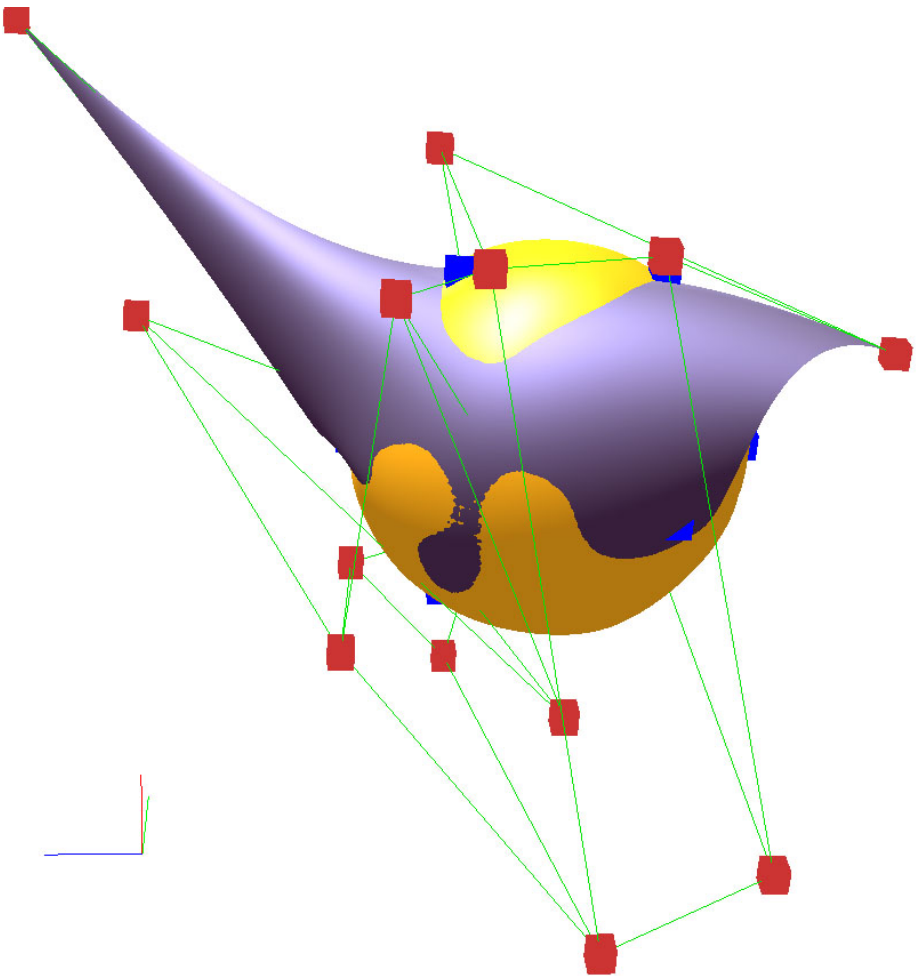

***Figure 5.7:*** *A plot of the approximated sphere from Figure 5.5, where one of the local Bezier patches is shown. The control polygon of the Bezier patch is marked in green, and the nodes in the polygon are marked as red cubes. Note that the figures are rotated, as can be seen on the RGB-frame in the bottom left hand corner. The "north pole", in both this figure and the previous figures, is in the direction of the color blue (B).*

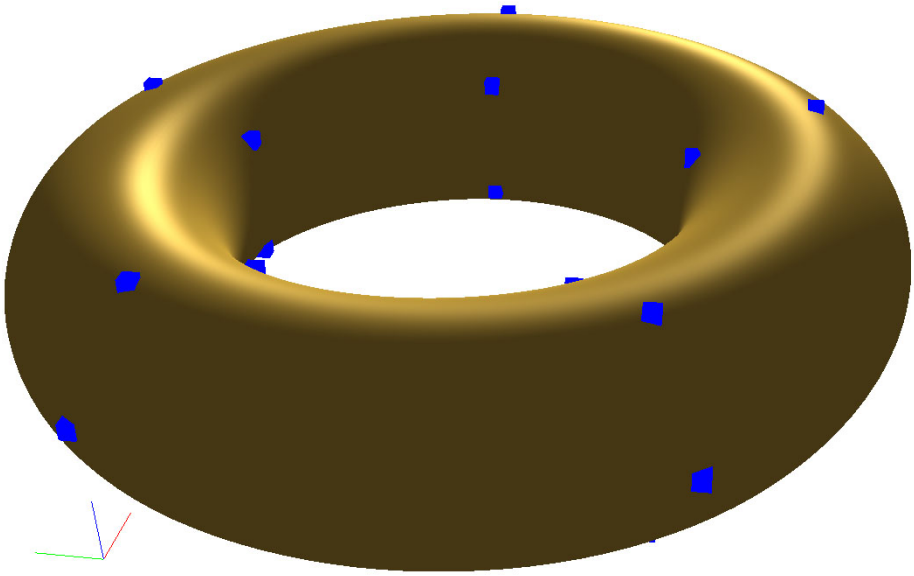

***Figure 5.8:*** *This Expo-Rational B-spline tensor product surface is made by interpolating a torus at $5 \times 5$ points (position, first and second derivative). The positions of the interpolating points are seen as blue cubes.*

see this local surface better. As we can see in the figure on the left hand side, the ERBS surface is actually following the local surface when it is moved, and thus changing shape. This will be discussed further in the next section.

In the last example, we can see a surface called "Sea Shell", described in several places, amongst other also described by Paul Bourke at [5]. The formula for this surface is

$$s(u,v) = \begin{pmatrix} \cos v + \frac{v}{10}\left(\cos u \cos v + a \cos u \sin v\right) \\ \sin v + \frac{v}{10}\left(\cos u \sin v - a \cos u \cos v\right) \\ \left(b \sin u + \frac{6}{10}\right) v \end{pmatrix} \quad \begin{array}{l} \text{for} \quad u \in (0, 2\pi], \\ \text{and} \quad v \in (\frac{\pi}{4}, 5\pi]. \end{array} \tag{5.26}$$

In Figure 5.11 there is a plot of an tensor product ERBS surface interpolating the "Sea Shell" surface (5.26) at $4 \times 8$ points. In this example, the position and a total of 8 partial derivatives are also used at each point, i.e. the matrix $\mathbf{g}_{d_u,d_v}(u_i, v_j)$ defined in (5.18) has the dimension $3 \times 3$. In Figure 5.11 the interpolation points are also marked as blue cubes, and most of them are visible. As can be seen, the surface is quite complex. It is "closed" in one parameter, but "open" in the other parameter. In the figure the surface can be seen from two different angles. In the view on the right hand side the blue cubes are hidden.

In the next figure, Figure 5.12, there are two plots of the tensor product ERBS surface from Figure 5.11, seen from two different angles. In both plots one of the local Bézier patches is also plotted, and it can be seen colored with violet. The control polygons of

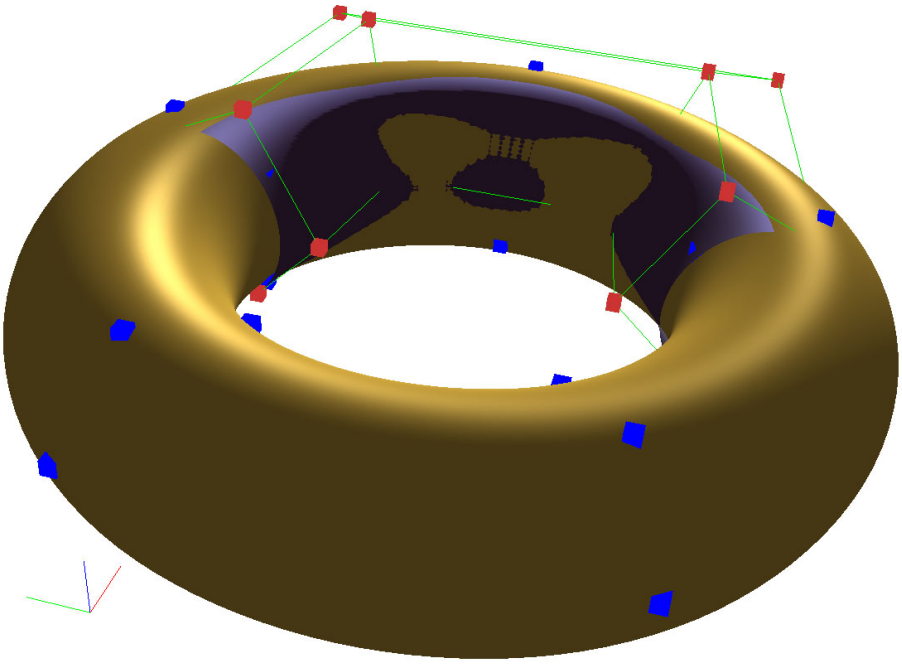

***Figure 5.9:*** *A plot of the approximated torus from Figure 5.8, where one of the local Bézier patches is also plotted. Also the control polygon of the local Bézier patch is plotted in green, while the nodes are plotted as red cubes.*

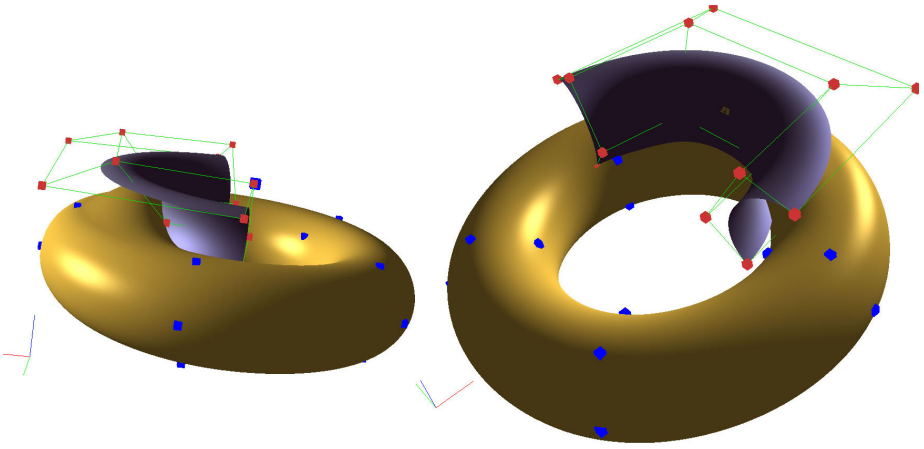

***Figure 5.10:*** *Two plots of the approximated torus from Figure 5.8, where one of the local Bézier patches is also plotted. In these plots the local patch is moved slightly upwards. In the plot on the left hand side one can clearly see that the ERBS surface has followed the movement. One can clearly see what the local Bézier patch looks like.*

the Bézier patches are also plotted. They can be seen in green and their nodes are marked with red cubes.

Interpolations does not generally preserve smoothness well. To see how well Hermite interpolation of a surface by ERBS tensor product surfaces preserve smoothness, we have computed the color of the ERBS surfaces using curvature. There are two figures (set of plots) of ERBS surfaces interpolating a "Sea Shell" surface. The first figure, 5.13, is using Gaussian curvature as the basis for the surface color. Here red represents the biggest positive value while blue represents the biggest negative value. All the colors in between are computed using linear interpolation in the HSV color system. The surface is shown from four different angles. The next figure, 5.14, uses the mean curvature as basis for the surface color. Also here red represents the biggest positive value while blue represents the biggest negative value, and the colors in-between are computed using linear interpolation in the HSV color system. The two plots show that the smoothness is very well preserved by the ERBS surface.

In all these examples "errors" have not been considered in the approximation. This is because numbers on their own make no sense if they cannot be compared with numbers from comparable methods from other geometry representations.

**Remark 17.** *Higher order Hermite interpolation for tensor product surfaces using B-splines or NURBS is not straightforward. It initially implies multiple knots and that the degree of the result is consistent with the number of partial derivatives in the interpolation. There are currently (year 2006) no available implementations of this among any of the "best known" products.*

Therefor, at the moment we have not been able to find available implementations of higher order Hermite interpolation for B-splines to use in comparing polynomial B-splines with ERBS. The comparison must, therefore, be the object of later explorations.

## 5.4 Free form sculpturing using tensor product ERBS surfaces

In a Computer Aided Geometric Design surface/object sculpturing is, of course, a big issue both for constructions for the real world (products for material productions), and for constructions for virtual worlds (virtual products: movies, computer games etc.). This has, thus, been one of the main goals for the research in Computer Aided Design. B-splines and NURBS are today the de facto standard for the surface representations in sculpturing tools, even though there is a series of other spline types (and other representations) with other nice properties available. As regards sculpturing, Barr [3] introduced, as early as in 1984, operations for twisting, stretching, bending, and tapering surfaces around a central axis. This was followed by Sederberg and Perry who in 1996 [51] introduced a more general technique, called the Free Form deformation method (FFD). This method embeds objects to be deformed in the 3D lattice of control points that define a trivariate Bézier volume. Deformations can, thus, be done by deforming this 3D control polygon

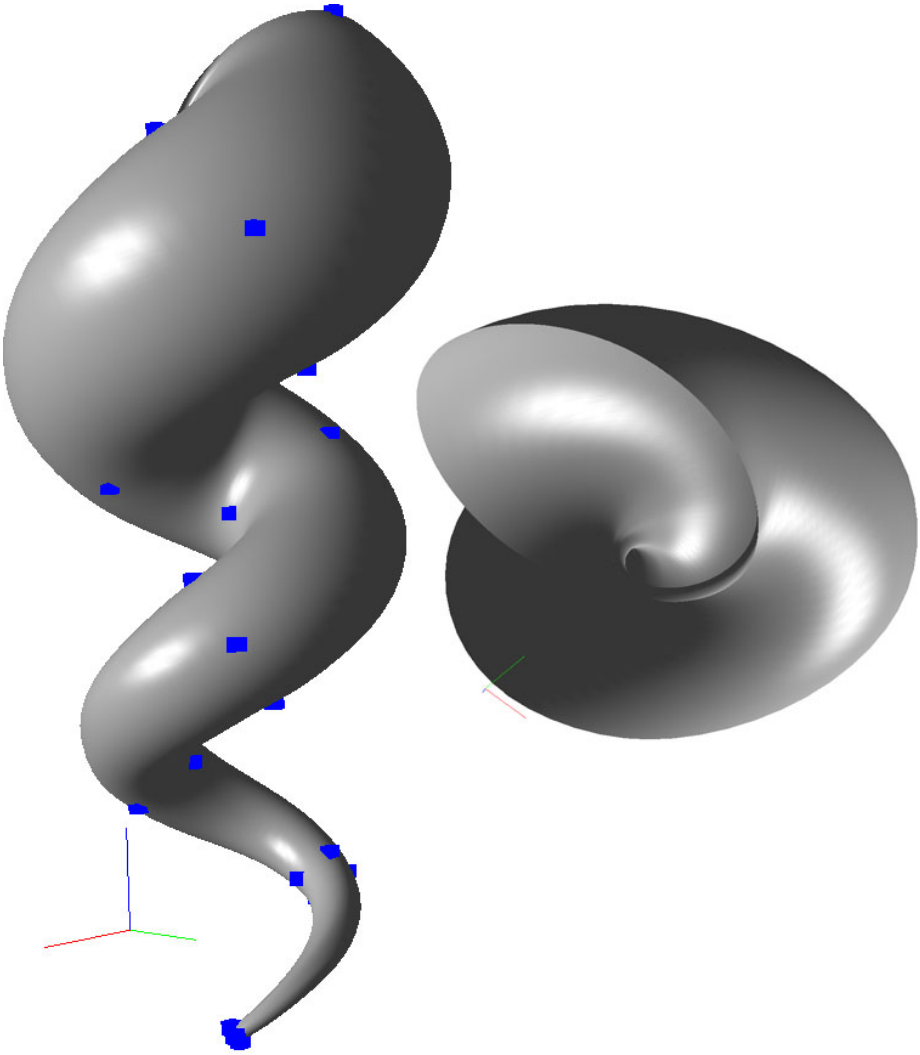

***Figure 5.11:*** *This Expo-Rational B-spline tensor product surface is made by interpolating (position, first and second derivative) a "Sea Shell" surface [5] at* $4 \times 8$ *points. The positions of the interpolating points are seen as cubes. The surface is plotted from two different angles. On the right hand side the interpolation points are hidden.*

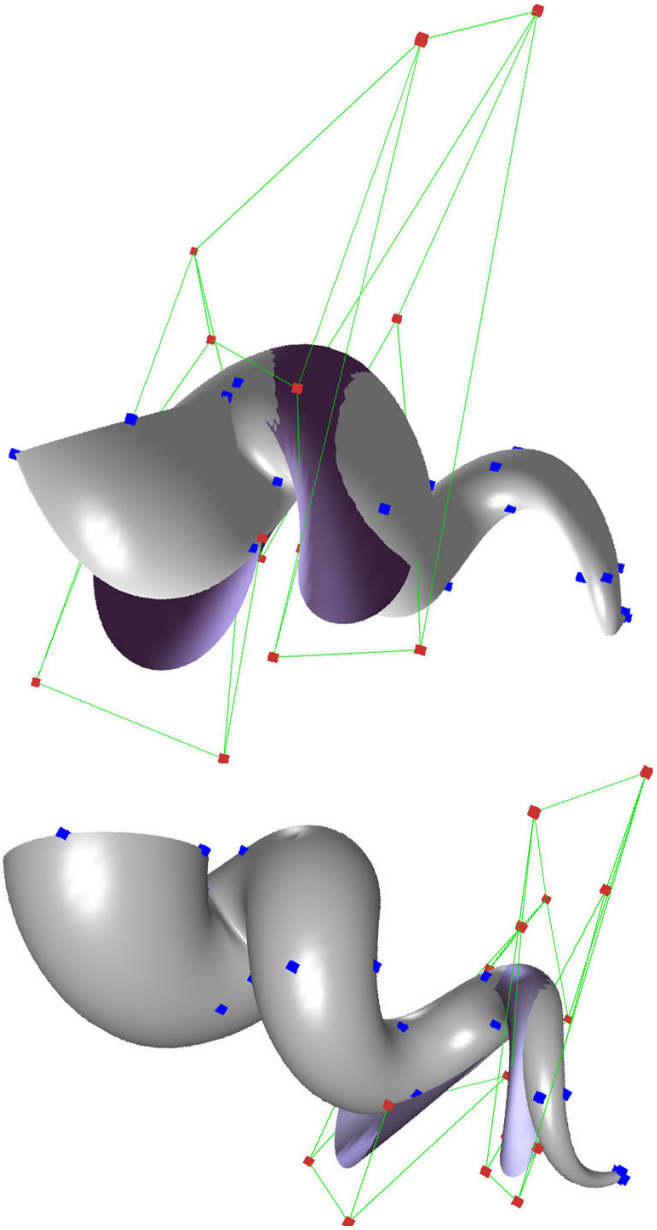

***Figure 5.12:*** *Two plots of the Expo-Rational B-Spline tensor product surface made by interpolating a "Sea Shell" surface plotted in Figure 5.11. The two plots are seen from different angles. In each of the two plots one of the local Bézier patches is also plotted. In the upper plot one of the bigger local patches is plotted. In the lower one a smaller patch is plotted. We can also see the control polygon for each of the local Bézier surfaces. The local patches model the global ERBS surface quite well locally.*

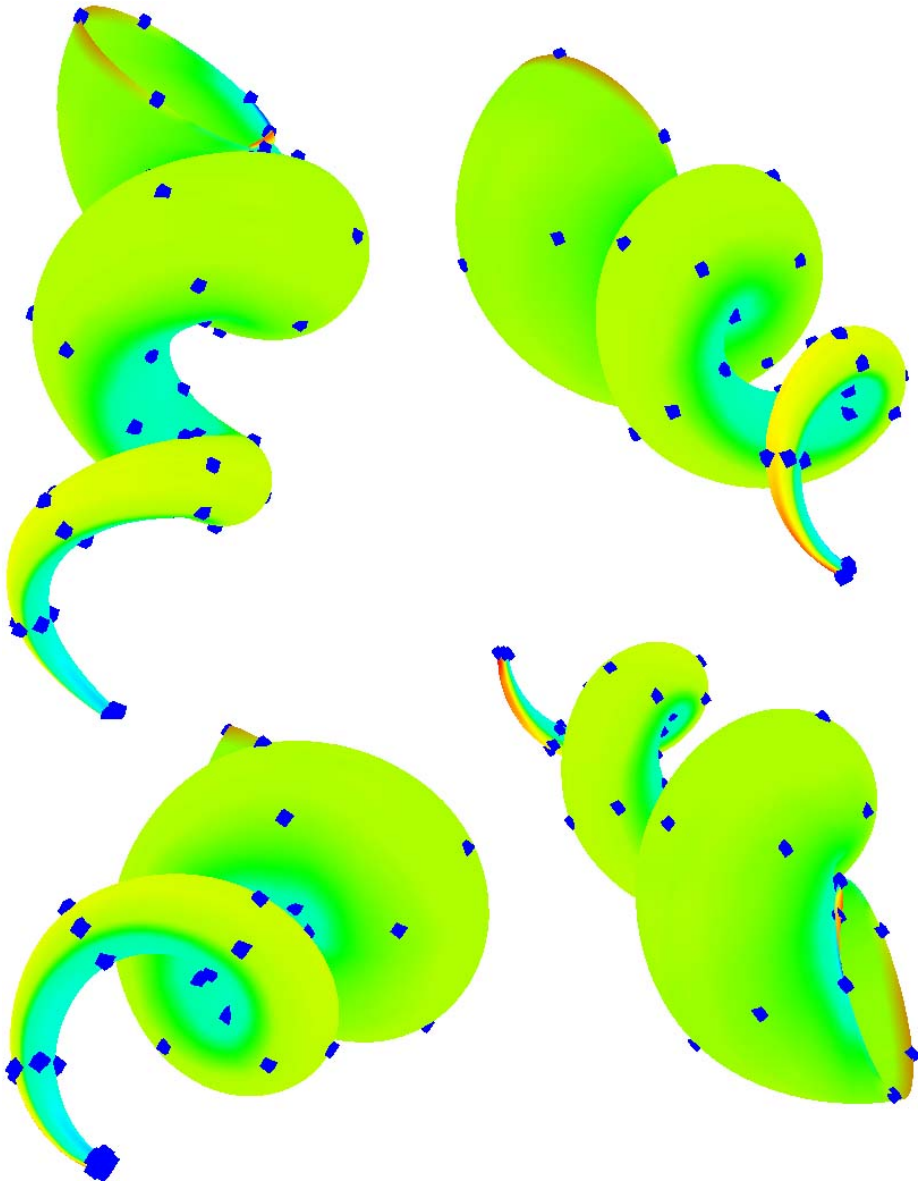

***Figure 5.13:*** *This Expo-Rational B-spline tensor product surface is made by interpolating (position, first and second derivative) a "Sea Shell" surface [5] at* $5 \times 12$ *points. The positions of the interpolating points are seen as blue cubes. The surface is plotted from four different angles. The color is based on the Gaussian curvature of the plotted ERBS tensor product surface.*

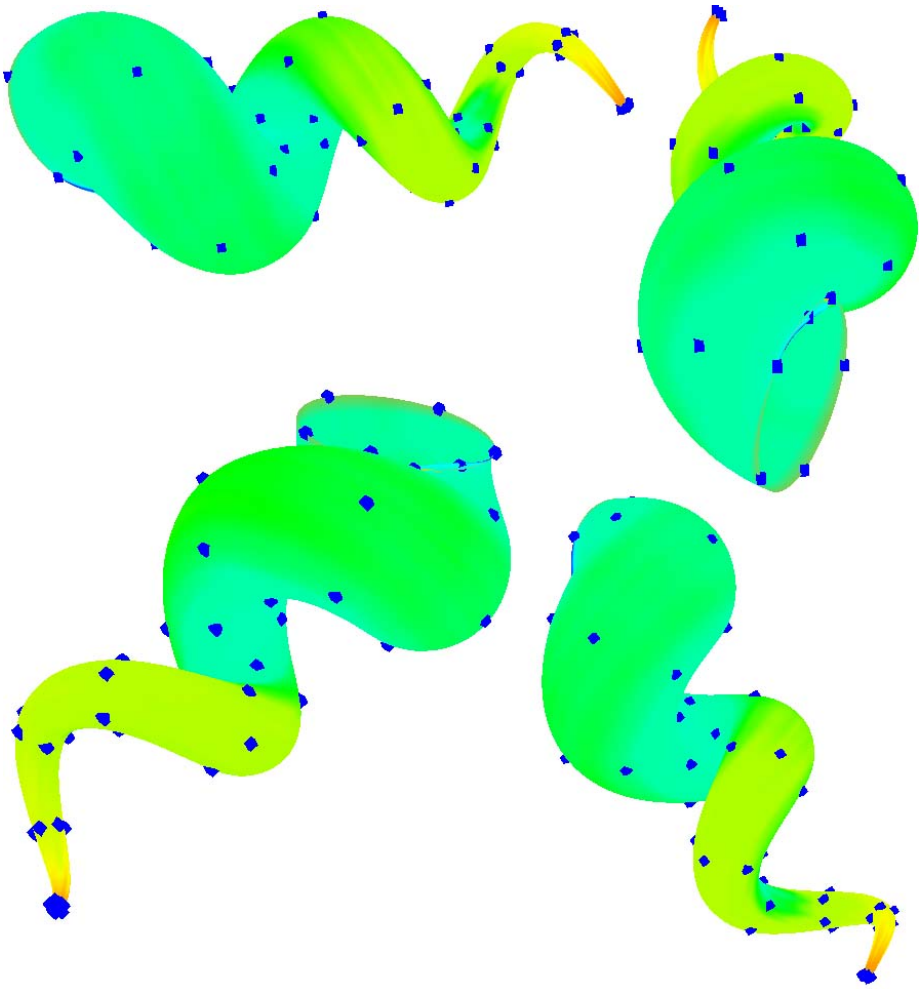

***Figure 5.14:*** *This Expo-Rational B-spline tensor product surface is made by interpolating (position, first and second derivative) a "Sea Shell" surface [5] at* $5 \times 12$ *points. The positions of the interpolating points are seen as blue cubes. The surface is plotted from four different angles. The color is based on the Mean curvature of the plotted ERBS tensor product surface.*

and evaluating the Bézier volume to find the new position for the embedded objects. In the following years a tremendous amount of work has been done in this area, using mechanical technics, multilevel representations etc. (Examples of articles are [10] [4] [33] [61]). Generally, all works so far have struggled with the geometric representation. What we will see in this section is what ERBS, and its representation, can offer. The general idea of this section is, therefor, that computer aided free-form sculpturing can be done in an equivalent way to how a sculptor works. The procedure is:

i) We start with an object "copied" from a well known surface/object.
ii) We then sculpture by editing the tensor product ERBS surface copy.

This is because the properties of tensor product ERBS surfaces are actually very convenient for surface design, especially as a sculpturing tool for free form deformations. For tensor product ERBS surfaces with local Bézier patches there are two notable types of editing possibilities in surface design:

i) The affine transformation of local surfaces, which is also possible to use for all kind of local surfaces.
ii) The editing of the control polygons of the local Bézier patches.

There are also, of course, other editing possibilities, such as changing intrinsic parameters or changing the knot vector. This will not be dealt with in this section.

In the following there are 6 examples of edited tensor product Expo-Rational B-spline surfaces where the editing is done only by affine transformations of local Bézier patches. In one additional example the control polygon of the local Bézier patches is also edited. Scaling of local patches will not be shown in these examples, but it is also an important editing tool.

The first three examples are based on a torus as the original surface. The first of these examples is a very simple surface editing example. It is a tensor product ERBS "torus" interpolating a torus at $5 \times 5$ points. The position and a total of 8 partial derivatives are used at each of the points, i.e., the matrix $\mathbf{g}_{d_u,d_v}(u_i, v_j)$ defined in (5.18) has the dimension $3 \times 3$. After the interpolation, two of the points marked as yellow cubes in Figure 5.15 are moved upwards.

The next example is also a tensor product ERBS "torus" interpolating a torus with the same number of points, and respective derivatives, as in the previous example. After the interpolation, the upper five points on the "inside" of the ERBS torus are moved upwards, while the five lower points on the "inside" of the ERBS torus are moved downwards. In Figure 5.16 the result can be seen from two different angles. The result is a surface which looks like a ´´flange".

The third torus example also starts with a tensor product ERBS "torus" interpolating a torus, this time at $8 \times 4$ points. As in the previous examples, the position and a total of 8 partial derivatives are used at each point, i.e., the matrix $\mathbf{g}_{d_u,d_v}(u_i, v_j)$ defined in (5.18) has the dimension $3 \times 3$. After the interpolation, half of the interpolating points at the top, every second one are slightly rotated around the "blue" axis (see the RGB frame down on left side of the figure). The result can be seen in Figure 5.17 as a deformed donut.

The next two examples are based on interpolating a sphere. The first example is based on constructing an ERBS surface by Hermite interpolation of a sphere at $4 \times 4$ points. The number of partial derivatives used results in the matrix $\mathbf{g}_{d_u,d_v}(u_i, v_j)$ defined in (5.18) which has the dimension $3 \times 3$ at each point. After the interpolation, the points at the north pole are moved towards the south pole together with the four points at the polar circle in the southern hemisphere. Remember that the north (and also south) pole actually consist of four points, and thus four local patches, even if it looks like it is only one point. The four points at the polar circle at the northern hemisphere are then moved upwards and also away from each other. Note that all partial derivatives, and thus the curvature, are kept at all interpolation points. This can clearly be seen in the bottom of the "mortar" on the right hand side in Figure 5.18.

The second sphere example can be seen in Figure 5.19. This example is based on constructing an ERBS surface by Hermit interpolation of a sphere at $6 \times 6$ points. Also this time the number of partial derivatives used results in the matrix $\mathbf{g}_{d_u,d_v}(u_i, v_j)$ defined in (5.18) which has the dimension $3 \times 3$ at each point. After the interpolation, three of the points are moved slightly downwards, the one on the "forehead", the one on the "nose", and the one on the "chin". The points at the "nose" and the points at the "chin" are also moved forwards. Then the point on the "forehead" is rotated forwards (around the green-axis in the RGB frame), and the points on the "eyes" are rotated towards the "nose" (around the blue-axis). Note that the surface is actually $C^\infty$ (but the normal is not defined at the poles, even if it is also "geometrically infinitely smooth" there).

The last two examples are based on an ERBS surface interpolating a planar surface. In Figure 5.20 we can see one of the special editing possibilities that ERBS surfaces offer. The surface started as an ERBS surface interpolating a planar rectangle with $3 \times 3$ interpolation points and where position and one partial derivative in each direction and the twist derivative are used at each point, i.e. the matrix $\mathbf{g}_{d_u,d_v}(u_i, v_j)$ defined in (5.18) has the dimension $2 \times 2$. In this example the result would have been the same even if the number of derivatives used in the Hermit interpolation was greater. As we can see, the central interpolation point is first moved upwards and is then rotated approximately $140^\circ$ around the blue axis of the RGB frame. The result can be seen in Figure 5.20, a kind of a twisted surface.

The last example is the "airplane". This example can be seen in Figure 5.21. The "airplane" construction also starts as an Hermite interpolation of a planar rectangle with $3 \times 3$ interpolation points, where position and two derivatives in each direction plus all the twist derivatives are used at each point, i.e., the matrix $\mathbf{g}_{d_u,d_v}(u_i, v_j)$ defined in (5.18) has the dimension $3 \times 3$. In the figure, 7 of the total 9 control polygons of the local Bézier patches are clearly visible. As we can see, they are not planar any more, because the local control points have been moved in an editing process. The interpolation points are actually moved and rotated first, and only some of the local control points are moved. The "airplane" was made by Børre Bang, who used less than 2 minutes on the process.

In connection with sculptured surfaces, it is of interest to look at the amount of data that is required. In 1987 knot removal for B-splines was introduced by Tom Lyche and Knut Mørken [40]. One of the motives for the knot removal was to reduce the amount of data.

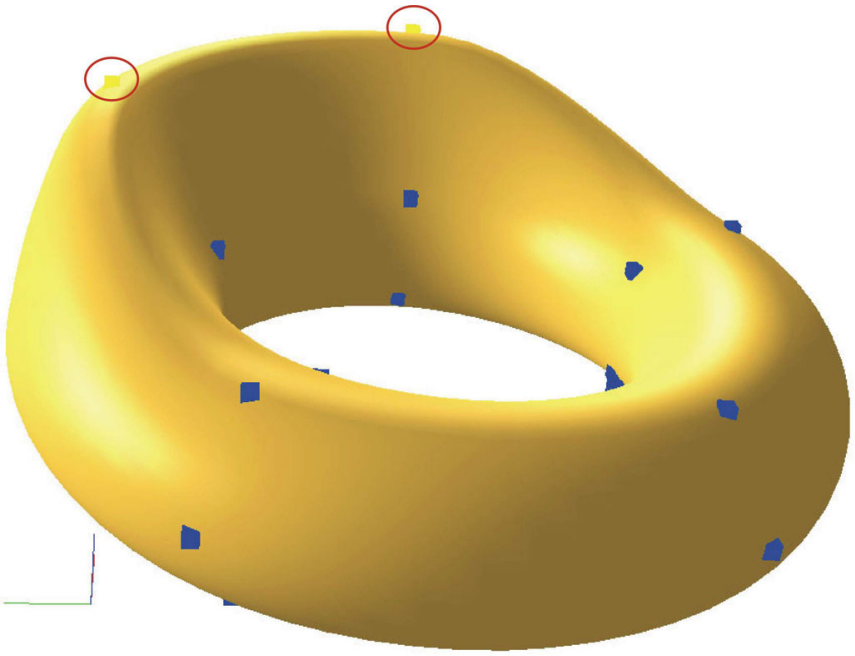

***Figure 5.15:*** *The approximated ERBS torus from Figure 5.8 is edited by moving two of the local Bézier patches upwards. The two Bézier patches are marked by yellow cubes (in red circles) at the interpolation points, while the other interpolation points are marked by blue cubes.*

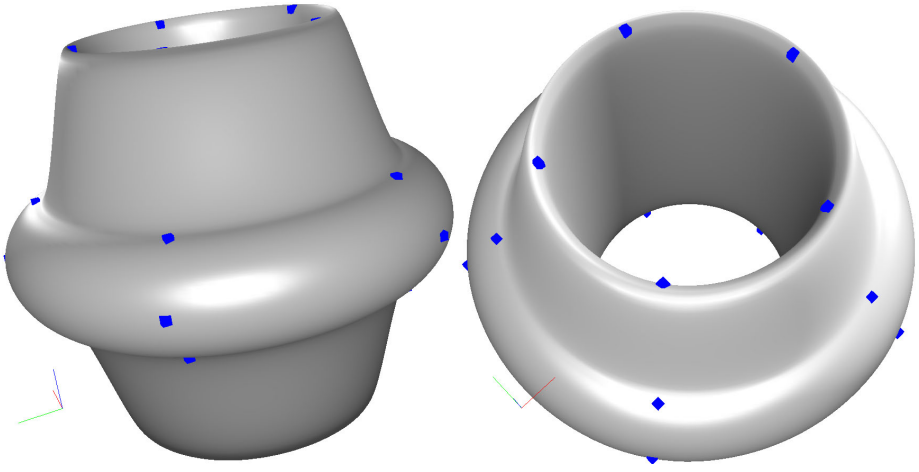

***Figure 5.16:*** *The approximated ERBS torus from Figure 5.8 is edited by moving five Bézier patches (interpolation points) on either side of the ERBS torus in opposite directions. In the figure there are two plots of the same object shown from different angles.*

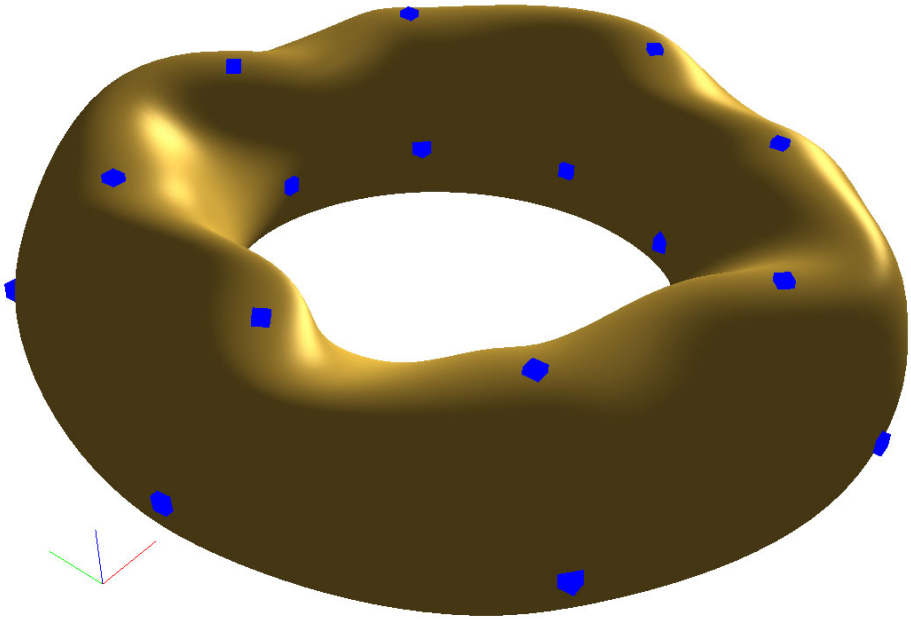

***Figure 5.17:*** *The approximated ERBS torus from Figure 5.8 is edited by rotating half of the interpolation points at the top around the z-axis. Note that none of the interpolation points are actually translated (only rotated).*

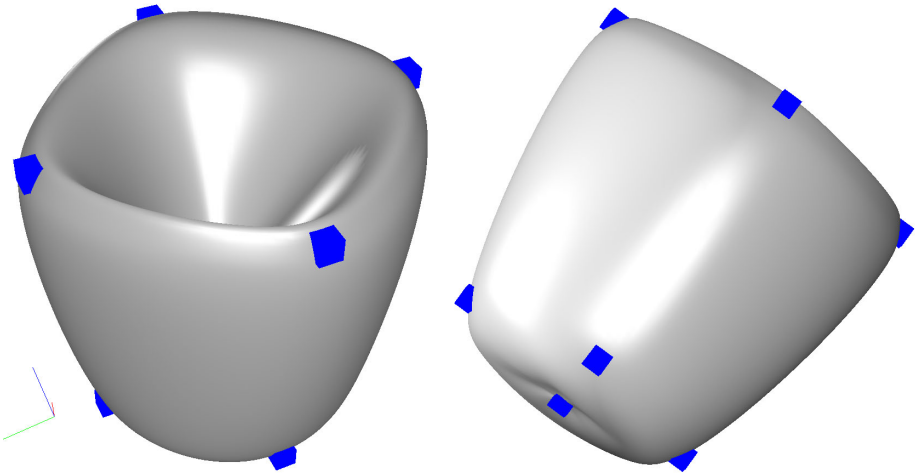

***Figure 5.18:*** *The approximated ERBS sphere from Figure 5.5 is edited by moving four interpolation points upwards and apart, and four points downwards. The four top points that coincide are also moved downwards. The result is a "mortar", seen from two angles.*

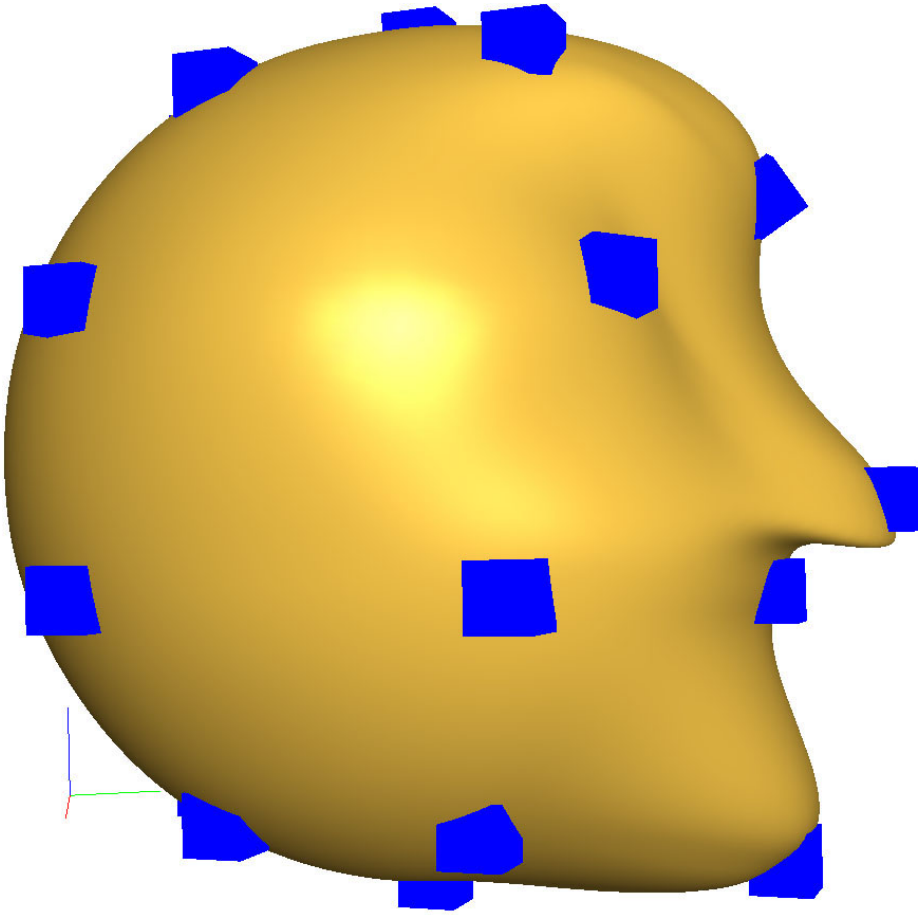

***Figure 5.19:*** *The figure, "head", shows one Expo-Rational B-spline tensor product surface made by first interpolating (position, first and second derivative) a sphere at* $6 \times 6$ *points. Then some of the interpolation points (five of the blue cubes) are moved, and some of them, three points at the eyes and nose, are also rotated.*

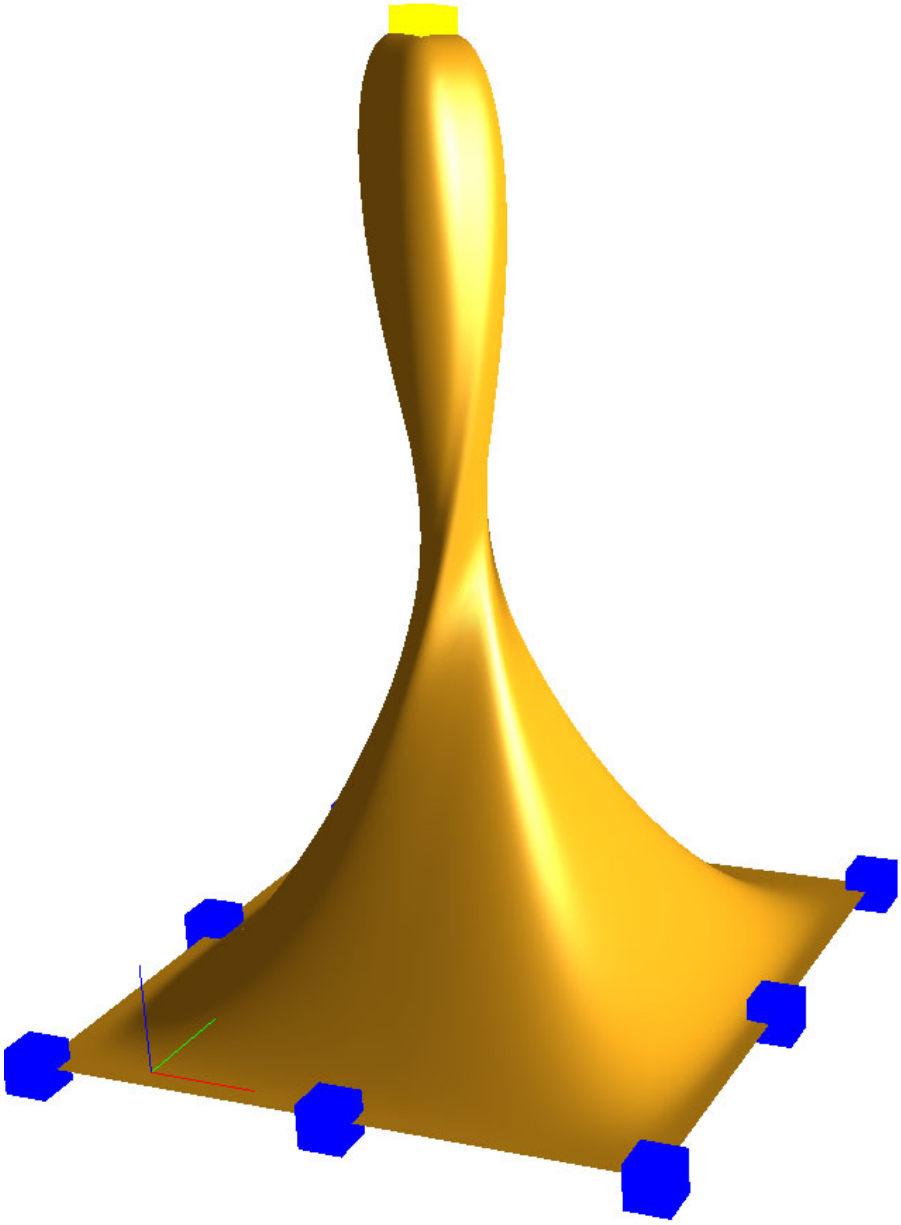

***Figure 5.20:*** *This Expo-Rational B-spline tensor product surface is first made by interpolating (position, first and second derivative) a plane at* $3 \times 3$ *points. The positions of the interpolating points are seen as cubes. The interpolation point in the center is moved upwards. Finally the center point is twisted, i.e. rotated approximately* $140^{\circ}$ *around the z-axis.*

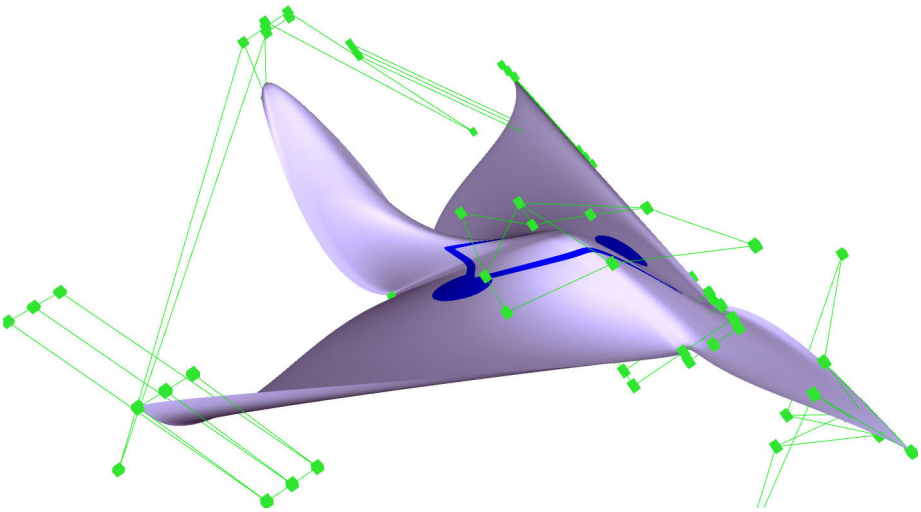

***Figure 5.21:*** *This Expo-Rational B-spline tensor product surface is made by first interpolating (position, first and second derivative) a plane at $3 \times 3$ points. Then the interpolation points are moved, and finally the control polygon on all of the local Bézier patches is edited. In the figure, some of the control polygons for the local Bézier patches are shown. (The surface is designed by Børre Bang.)*

One problem, however, was that local details in a small area made it impossible to reduce data, not only in the current area, but in a "big" area that incudes one of the parameter values that defined the current area. Later, some attentions were made to solve this problem by using (partial) hierarchical B-splines [39]. Even later, T-splines were introduced by Thomas Sederberg et all. [50], and a conversion from NURBS to T-spline was presented in [49]. Recently, a conversion between hierarchical B-splines and T-splines has also been introduced [56]. ERBS can already offer one solution to this problem, because it is possible to use local patches of different degrees or types and, thus, different detailing levels. The local surfaces can even be tensor product ERBS surfaces themselves (multi-level ERBS). But it is of course of great interest to look at ERBS with T-junctions. This will, however, not be discussed here, but left as a possible topic for investigation.

## 5.5 Computational and implementational aspects for tensor product ERBS surfaces

One small problem in the process of "editing" tensor product Expo Rational B-spline surfaces is the evaluation algorithm, which is computationally more expensive than the available algorithms used for B-splines/NURBS. Evaluating an ERBS surface $S(u,v)$ involves, $u_i < u \leq u_{i+1}$, $v_j < v \leq v_{j+1}$, computations of the basis functions $B_{i+1}(u)$ and $B_{j+1}(v)$, and the local patches $s_{i,j}(u,v)$, $s_{i+1,j}(u,v)$, $s_{i,j+1}(u,v)$ and $s_{i+1,j+1}(u,v)$.

Summing up, to evaluate a tensor product Expo Rational B-spline surface we have to compute:

- 2 basis functions and 4 local (Bézier) patches.

This is of course about 4 times more expensive than the available algorithms used for B-splines/NURBS. To compensate for the greater computational complexity, pre-evaluation on a given sampling grid can be used. This means that the basis function and the respective derivatives can be pre-evaluated in the sampling grid. From the point of view of object-oriented programming, all local patches 'can pre-evaluate' in their part of the sampling grid. This is, as we will see below, possible to do at two levels. It is also possible to only compute the sample points are be changed because of the editing process. This, of course, requires a more sophisticated organization in the programs. Implementing all this will result in editing programs running smoothly even on small laptops. Summing up the overview of the pre-evaluations we get the following:

1. We first decide the sampling number in both directions, $m_u$ and $m_v$. It is possible to resample whenever it is desirable. But then everything has to be recomputed.
2. We now pre-evaluate the basis functions and their respective derivatives in both parameter directions at all sampling points. The upper limit of the derivational degree of the derivatives is equal the maximum degree of one of the two local Bézier patches in the appurtenant parameter direction.
3. For each local Bézier patch, we compute, at every sampling point in its part, the matrix $\mathbf{S}_{r,s,d_u,d_v}(u_i, v_j)$ defined in (5.19), where $(r,s)$ is the index of the local patch, $d_u, d_v$ is the number of derivatives in the respective $u$ and $v$ directions, and $(i,j)$ is the index of the sampling point.
4. For each local Bézier patch, we compute, at every sampling point in its part, the two matrices $\mathbf{B}_{d_u}(u_i, \delta_{u,r})$ and $\mathbf{B}_{d_v}(v_j, \delta_{v,s})$ defined in (5.18), where $(r,s)$ is the index of the local patch, $d_u, d_v$ is the number of derivatives in the respective $u$ and $v$ directions, and $(i,j)$ is the index of the sampling point.

It follows that, if the sampling grid is changed, all pre-evaluations have to be recomputed, and if the evaluation is done outside the sampling grid, all parts of the evaluator have to be computed. If, however, the editing is using only affine transformation of local patches, then points 1, 2 and 3 in the previous list ensure that we do the minimal calculations to update the necessary sampling points. If we, however, also edit the control polygon of the local Bézier patches, we need all of the above-said 4 points to do the minimal calculations when updating the sampling points that will be changed.

In most of the figures of tensor product ERBS surfaces, we can see blue or green cubes. In our test program, these cubes are "selectors" or "representatives" implemented for editing purposes. The blue cubes represent a local Bézier patch and can be translated, rotated and scaled (scaling is not used in the previous examples). The green cubes represent control points for the local Bézier patch, and can, therefore, only be translated.

# Chapter 6

# Triangular Surfaces

Triangular surfaces (an example can be seen in Figure 6.1) have long been a focus of attention in Computer Aided Geometric Design and in particular for the Computer Graphics community. The reasons for this are that, tessellations of geometric faces of 3D objects are usually done by triangulations, and that triangular representation is more flexible than tensor product surfaces, especially because local refinements are easy to do. In addition the surface (boundary) of a 3D object can always be triangulated. These surfaces are bounded compact and connected surfaces of different possible topological genus $g$ (number of wholes/handles). A short investigation of the triangulation of these surfaces gives us the following connections. The Euler-Poincaré characteristic for a compact and connected surface $S$ is the well known

$$\varkappa(S) = F - E + V,$$

where $F$ is the number of triangles, $E$ is the number of edges, and $V$ is the number of vertices of a triangulation of the surface $S$. It is actually a number independent of a particular triangulation, and will be the same for all triangulations of the given surface $S$. It is also connected to the genus $g$ of the surface in the following way,

$$\varkappa(S) = 2(1-g).$$

A closer study of this can be found in connection with a study of the Gauss-Bonnet Theorem in [20].

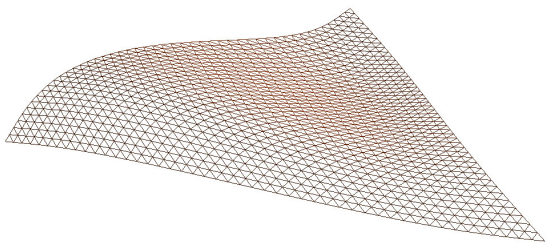

***Figure 6.1:*** *A smooth, but non-planar triangular surface in* $\mathbb{R}^3$.

The first part of this chapter will concentrate on basic properties of the simplex, single triangular surfaces, and then we introduce the Expo-Rational B-spline triangle. Finally, at the end of this chapter, ERBS partition of unity over triangulations will be introduced.

The best known curved triangular surface is the Bézier triangle. Bézier triangles were (according to Farin [23]) first introduced early in the 1960s by de Casteljau in two internal Citroën technical reports [14] and [15]. The basic constructions of the Bézier triangle will shortly be repeated in section 6.2.

However, the main goal of the first part of this chapter is the introduction, and a closer investigation of an ERBS analog to the Bézier triangle, also defined on a domain described by homogeneous barycentric coordinates. Therefor, the definition of the homogeneous barycentric coordinates will now shortly be presented.

## 6.1 Homogenous barycentric coordinates for simplices

Barycentric coordinates were introduced by Möbius as early as in 1827, see [25]. These coordinates can be used both to express a point inside a simplex (line segment, triangle, tetrahedron, etc.) as a convex combination of the n vertices in the simplex, and to linearly interpolate data given at the vertices. A lot of work has been done to investigate these coordinates and their use. Amongst others is Warrens, in [57] and in later publications, and Michael Floater and his colleagues investigating mean value/barycentric coordinates in [27] and [26].

In this section we will only repeat some main facts about homogeneous barycentric coordinates (contrary to barycentric coordinates, homogeneous barycentric coordinates are normalized and thus unique). We start first by looking at the triangle case (a surface). Consider an ordinary surface domain $\Omega \subset \mathbb{R}^2$ with an ordinary orthogonal coordinate system $u$ and $v$. Consider that this is the unit square, as in the tensor product Bézier surface case. In Figure 6.2 this unit square is expanded to a unit cube using a coordinate $w$ orthogonal to the other coordinates. If we intersect this cube with a plane that intersects the three corners where only one of the coordinates is 1 (and all the other coordinates are 0), we will get what we can call the "main diagonal" in the unit cube. In Figure 6.2 this intersection (main diagonal) can be seen as a triangle marked with thicker lines. This triangle is actually a domain described in homogeneous barycentric coordinates. Remember that the surface intersecting the cube has the following implicit formula,

$$u+v+w=1.$$

If we in addition want to restrict the domain to be in the interior or boundary of the triangle in Figure 6.2, the restriction on the parameters is,

$$u,v,w \geq 0.$$

The name "barycenter" is the Latin word for for the center of gravity. In fact it is refereing to the coordinates of the "mass center" which is always decided via a convex combination of the extreme points of a material system of points.

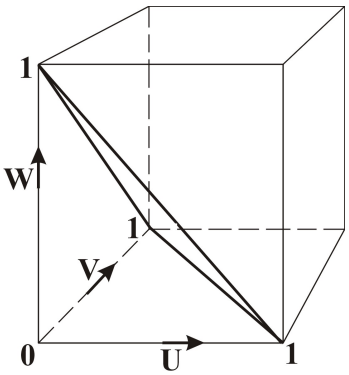


***Figure 6.2:*** *A parametric unit cube described by the coordinates u, v and w. The "main diagonal" is marked with thicker lines, and can be seen as a triangle in the figure.*

Now follows definition of homogeneous barycentric coordinates of a point.

**Definition 6.1.** *We first denote the convex set of points on the "main diagonal" of a unit (n+1)-dimensional hypercube as $\Xi_n$. The definition of points in homogeneous barycentric coordinates $\mathbf{p} \in \Xi_n$ is*

$$\mathbf{p} = \{u_i\}_{i=1}^{n+1}, \quad \textit{where} \quad \sum_{i=1}^{n+1} u_i = 1,$$

*fulfilling a convexity property when*

$$u_i \geq 0 \quad \textit{for} \quad i = 1,2,...,n+1.$$

A point $\mathbf{p} \in \Xi_2$ (a point on the triangle in Figure 6.2) is defined in the following way,

$$\mathbf{p} = (u,v,w), \quad \text{where} \quad u+v+w = 1 \quad \text{and} \quad u,v,w \geq 0.$$

We now extend the definition of homogeneous barycentric coordinates to also including vectors. We look at the plane in Figure 6.2, which intersects the unit cube and creates the triangle on the "main diagonal". We insert a new plane, which is parallel to this plane, but which now intersects the origin. This new plane can be seen as a 2D vector space in barycentric coordinates. The implicit formula for this plane is

$$u+v+w = 0.$$

Now follows the definition of homogeneous barycentric coordinates of a vector.

**Definition 6.2.** *We first denote the n-dimensional vectors space parallel to the "main diagonal" of an unit (n+1)-dimensional hypercube as $\Upsilon_n$. The definition of a vector in homogeneous barycentric coordinates $\mathbf{d} \in \Upsilon_n$ is*

$$\mathbf{d} = \mathbf{p}_2 - \mathbf{p}_1 = \{r_i\}_{i=1}^{n+1}, \quad \textit{where} \quad \sum_{i=1}^{n+1} r_i = 0 \quad \textit{and} \quad \mathbf{p}_1, \mathbf{p}_2 \in \Xi_n.$$

For a triangle (2-dimensional simplex), we then see that a vector $d \in \Upsilon_2$ is defined in the following way,

$$\mathbf{d} = (r,s,t), \quad \text{where} \quad r+s+t = 0.$$

## 6.2 Bézier triangles

A convex set defined by homogeneous barycentric coordinates defines the domain for the Bézier triangle, where we can construct basis functions of arbitrary degrees in the following way,

$$(u+v+w)^d = 1, \quad \text{where} \quad u,v,w \geq 0 \quad \text{and} \quad d > 0.$$

The expression of these basis functions for Bézier triangles are thus the Bernstein polynomials of degree $d$, where for $u+v+w=1$,

$$b_{d,i,j,k}(u,v,w) = \binom{d}{ijk} u^i v^j w^k, \quad \text{where} \quad i+j+k = d \quad \text{and} \quad i,j,k \geq 0.$$

For Bézier triangular surfaces we, therefor, have the following general formula,

$$S(u,v,w) = \sum_{\substack{i+j+k=d,\\ i,j,k\geq 0}} c_{i,j,k}\, b_{d,i,j,k}(u,v,w) \quad \text{for} \quad u+v+w = 1,$$

where $c_{i,j,k} \in \mathbb{R}^n$, are the coefficients where n is usually 2 or 3.

In homogeneous barycentric coordinates, we have both partial and directional derivatives. The appropriate derivatives are direction derivatives, where the vector $\mathbf{d} = p_1 - p_2$ (the distance vector between two points) is defined by a vector in homogeneous barycentric coordinates $\mathbf{d} = (r,s,t)$, where $r+s+t = 0$.

The directional derivative is thus,

$$D_{\mathbf{d}} S(u,v,w) = \sum_{\substack{i+j+k=d,\\ i,j,k\geq 0}} c_{i,j,k}\, D_{\mathbf{d}} b_{d,i,j,k}(u,v,w) \quad \text{for} \quad u+v+w = 1, \tag{6.1}$$

where

$$D_{\mathbf{d}} b_{d,i,j,k}(u,v,w) = rD_u b_{d,i,j,k}(u,v,w) + sD_v b_{d,i,j,k}(u,v,w) + tD_w b_{d,i,j,k}(u,v,w). \tag{6.2}$$

Cross derivatives are defined in the same way,

$$D_{\mathbf{d_2}} D_{\mathbf{d_1}} S(u,v,w) = \sum_{\substack{i+j+k=d,\\ i,j,k\geq 0}} c_{i,j,k}\, D_{\mathbf{d_2}} D_{\mathbf{d_1}} b_{d,i,j,k}(u,v,w) \quad \text{for} \quad u+v+w = 1,$$

where

$$D_{\mathbf{d_2}} D_{\mathbf{d_1}} b_{d,i,j,k}(u,v,w) = r_2\, D_u D_{\mathbf{d_1}} b_{d,i,j,k}(u,v,w) + s_2\, D_v D_{\mathbf{d_1}} b_{d,i,j,k}(u,v,w) + t_2\, D_w D_{\mathbf{d_1}} b_{d,i,j,k}(u,v,w).$$

More about the general theory of Bernstein-Bézier triangles, including the algorithm for evaluation and derivations, can be found in [22] or [23].

In the rest of this chapter we shall look at ERBS basis function in homogeneous coordinates, the definition of an ERBS triangle, Hermite interpolation by ERBS triangles, after which ERBS over triangulations are introduced, and several examples are given.

## 6.3 The general set of Expo-Rational B-spline basis-function in homogeneous barycentric coordinates

In this section we will introduce the definition of the basis function for Expo-Rational B-spline triangles. The definition will, however, be expanded and generalized so that it will be valid for simplices of any dimention. The properties will be stated, and will of course also be generally valid. The examples, however, will only show triangles.

The definition is divided into two parts.
**1.** The first part is an introduction of an underlying basic ERBS basis function, $B(u)$. This univariate ERBS basis function is, unlike the general ERBS in Definition 2.2, not defined on a general knot vector. It is only defined on the (knot) interval $[0,1]$. It thus is combining the properties, and the intrinsic parameters, of the general set, definition (2.2), and the functionality of the scalable set, definition (2.4).

We, therefore, first introduce the underlying basic ERBS basis function $B(u)$ for homogeneous barycentric coordinates.

**Definition 6.3.** *The underlying basic Expo-Rational B-splines for homogeneous barycentric coordinates is defined by $B(t) = B(\alpha,\beta,\gamma,\lambda,\sigma;t)$ as follows,*

$$B(t) = \begin{cases} S(\alpha,\beta,\gamma,\lambda,\sigma)\displaystyle\int_0^t \phi(\alpha,\beta,\gamma,\lambda,\sigma;s)ds, & \text{if} \quad 0 < t \le 1, \\ 0, & \text{otherwise} \end{cases}$$

*where*

$$\phi(\alpha,\beta,\gamma,\lambda,\sigma;t) = e^{-\beta\frac{|t-\lambda|^{2\sigma}}{(t(1-t)^{\gamma})^{\alpha}}},$$

*and the scaling factor*

$$S(\alpha,\beta,\gamma,\lambda,\sigma) = \frac{1}{\displaystyle\int_0^1 \phi(\alpha,\beta,\gamma,\lambda,\sigma;t)dt},$$

*where*

$$\alpha > 0, \quad \beta > 0, \quad \gamma > 0, \quad 0 \le \lambda \le 1, \quad \sigma \ge 0.$$

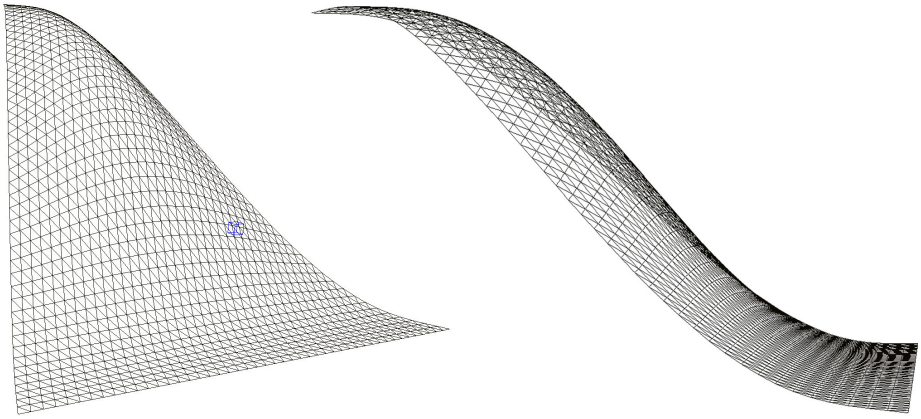

***Figure 6.3:*** *The Expo-Rational B-spline basis functions $\mathcal{B}_{k,i}(\mathbf{u})$ in homogeneous barycentric coordinates for triangles shown from two different angles.*

**2.** The second part is the introduction and thus definition of the sets of Expo-Rational B-spline basis functions in homogeneous barycentric coordinates for general degrees.

**Definition 6.4.** *Given is a point $\mathbf{u} = (u_1, u_2, ..., u_k) \in \Xi_{k-1}$ (in homogeneous barycentric coordinates, fulfilling the convexity property, see Definition 6.1). The definition of a set of Expo-Rational B-spline basis function in homogeneous barycentric coordinates is then defined by $\mathcal{B}_{k,i}(\mathbf{u}) = \mathcal{B}_{k,i}(\alpha, \beta, \gamma, \lambda, \sigma; \mathbf{u})$ as follows,*

$$\mathcal{B}_{k,i}(\mathbf{u}) = \frac{B(u_i)}{\sum_{j=1}^{k} B(u_j)} \quad \textit{for} \quad k > 1 \quad \textit{and} \quad i = 1, 2, ..., k, \tag{6.3}$$

*and where $B(u_i),\ i = 1, 2, ..., k$, is defined in definition 6.3.*

For $k = 3$, i.e. Expo-Rational B-spline triangles, the basis functions will be,

$$\mathcal{B}_{3,i}(\mathbf{u}) = \frac{B(u_i)}{B(u_1) + B(u_2) + B(u_3)} \quad \text{for} \quad i = 1, 2, 3.$$

The basic properties of this set of Expo-Rational B-spline basis functions will be further investigated at the end of this section. To get a better overview of $\mathcal{B}_{k,i}(\mathbf{u})$ there is, in Figure 6.3, a plot of one of three basis functions for triangular surfaces, plotted over a equilateral triangle, and seen from two different angles. The other two basis functions have the same shape, but they are rotated so that their "top-points" are in the other two corners.

As for the Bézier triangle, the derivatives have to be defined for specific directions $\mathbf{d} \in \Upsilon_{k-1}$. Thus, the partial derivatives are necessary to compute as components. Therefore, for $k > 1$, and for $i = 1, 2, ..., k$ we have for the following six equations, first for the first-order partial derivatives, where

$$D_{u_i}\mathcal{B}_{k,i}(\mathbf{u}) = DB(u_i)\frac{\sum_{j=1}^{k} B(u_j) - B(u_i)}{\left(\sum_{j=1}^{k} B(u_j)\right)^2}, \tag{6.4}$$

and, for $j = 1,2,...,k$, but where $j \neq i$,

$$D_{u_j}\mathcal{B}_{k,i}(\mathbf{u}) = DB(u_j)\frac{-B(u_i)}{\left(\sum_{j=1}^{k} B(u_j)\right)^2}. \tag{6.5}$$

For second order partial derivatives we have,

$$D^2_{u_i}\mathcal{B}_{k,i}(\mathbf{u}) = \left(D^2B(u_i) - \frac{2\left(DB(u_i)\right)^2}{\sum_{j=1}^{k} B(u_j)}\right)\frac{\sum_{j=1}^{k} B(u_j) - B(u_i)}{\left(\sum_{j=1}^{k} B(u_j)\right)^2}, \tag{6.6}$$

and for $j = 1,2,...,k$ but where $j \neq i$ for both

$$D^2_{u_j}\mathcal{B}_{k,i}(\mathbf{u}) = \left(D^2B(u_j) - \frac{2\left(DB(u_j)\right)^2}{\sum_{j=1}^{k} B(u_j)}\right)\frac{-B(u_i)}{\left(\sum_{j=1}^{k} B(u_j)\right)^2}, \tag{6.7}$$

and

$$D_{u_i}D_{u_j}\mathcal{B}_{k,i}(\mathbf{u}) = DB(u_i)\ DB(u_j)\frac{2B(u_i) - \sum_{j=1}^{k} B(u_j)}{\left(\sum_{j=1}^{k} B(u_j)\right)^3}. \tag{6.8}$$

And finally, for $j = 1,2,...,k$, but where $j \neq i$, and for $h = 1,2,...,k$, but where $h \neq i, j$, we have

$$D_{u_h}D_{u_j}\mathcal{B}_{k,i}(\mathbf{u}) = DB(u_h)\ DB(u_j)\frac{2B(u_i)}{\left(\sum_{j=1}^{k} B(u_j)\right)^3}. \tag{6.9}$$

Given is a vector $\mathbf{d} \in \Upsilon_{k-1}$, i.e.,

$$\mathbf{d} = \mathbf{u_1} - \mathbf{u_2} = (d_1, d_2, ..., d_k), \quad \text{where} \quad \mathbf{u_1} \text{ and } \mathbf{u_2} \in \Xi_{k-1}.$$

We can now define the directional derivatives for the ERBS basis function in homogeneous barycentric coordinates. We now get

$$D_{\mathbf{d}}\mathcal{B}_{k,i}(\mathbf{u}) = \sum_{j=1}^{k} d_j\ D_{u_j}\mathcal{B}_{k,i}(\mathbf{u}). \tag{6.10}$$

In Figure 6.4 there is a plot of a shaded ERBS basis function for homogeneous barycentric coordinates for triangles. The basis functions are not vector valued functions, but scalar functions. The normal used in the shading is, therefore, computed by a cross product of two "tangent vectors", where the derivatives of an ERBS basis function in the directions, $(-1,1,0)$ and $(-1,0,1)$ is used as a z-value for these tangent vectors, and where the x-values and y-values appear by computing the barycentric coordinates of the direction vectors with the respective points in the plane (i.e., 2 coordinates) and summing them up.

For practical use it is convenient to define some main directions for derivatives. For triangles we might use the following choice,

$$\begin{aligned} \mathbf{d}_1 &= (-1,1,0), \\ \mathbf{d}_2 &= (-1,0,1). \end{aligned} \tag{6.11}$$

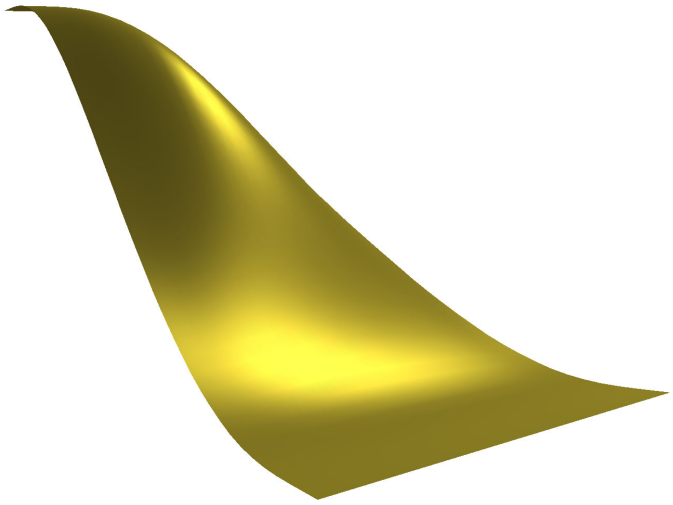

***Figure 6.4:*** *A shaded version of the Expo-Rational B-spline basis functions $\mathcal{B}_{k,i}(\mathbf{u})$ in homogeneous barycentric coordinates for triangles.*

In these main directions, for the ERBS basis functions in homogeneous barycentric coordinates for triangles, are $\mathcal{B}_{2,i}(u,v,w)$ for $i = 1,2,3$, the derivatives up to second order as follows,

$$\begin{aligned}
D_{\mathbf{d}_1}\mathcal{B}_{2,i}(u,v,w) &= D_v\mathcal{B}_{2,i}(u,v,w) - D_u\mathcal{B}_{2,i}(u,v,w),\\
D_{\mathbf{d}_2}\mathcal{B}_{2,i}(u,v,w) &= D_w\mathcal{B}_{2,i}(u,v,w) - D_u\mathcal{B}_{2,i}(u,v,w),\\
D^2_{\mathbf{d}_1}\mathcal{B}_{2,i}(u,v,w) &= D^2_v\mathcal{B}_{2,i}(u,v,w) - D_uD_v\mathcal{B}_{2,i}(u,v,w) + D^2_u\mathcal{B}_{2,i}(u,v,w),\\
D^2_{\mathbf{d}_2}\mathcal{B}_{2,i}(u,v,w) &= D^2_w\mathcal{B}_{2,i}(u,v,w) - D_uD_w\mathcal{B}_{2,i}(u,v,w) + D^2_u\mathcal{B}_{2,i}(u,v,w),\\
D_{\mathbf{d}_1}D_{\mathbf{d}_2}\mathcal{B}_{2,i}(u,v,w) &= D_vD_w\mathcal{B}_{2,i}(u,v,w) - D_uD_v\mathcal{B}_{2,i}(u,v,w)\\
&\quad - D_uD_w\mathcal{B}_{2,i}(u,v,w) + D^2_u\mathcal{B}_{2,i}(u,v,w).
\end{aligned}$$

It is clear that the properties for the generalized ERBS defined in Theorem 2.1 is also valid for the underlying basic ERBS for homogeneous barycentric coordinates in (def. 6.3).

Before looking at the properties of the sets of ERBS basis functions in homogeneous barycentric coordinates (def. 6.4), we will look at the following definition of corner points and points on an "edge", in homogeneous barycentric coordinates.

**Definition 6.5.** *The corner points, denoted $\widehat{\mathbf{u}}_i$, for $i = 1,2,\ldots,k$, are defined as follows,*

$$\widehat{\mathbf{u}}_i \in \Xi_{k-1}, \quad \textit{where the coordinate } u_i = 1,$$

*and it follows that the rest of the coordinates $u_j = 0$, $j = 1,2,\ldots,k$ but where $j \neq i$.*

*The edge points , denoted $\overline{\mathbf{u}}_i$, for $i = 1,2,\ldots,k$, are defined as follows,*

$$\overline{\mathbf{u}}_i \in \Xi_{k-1}, \quad \textit{where the coordinate } u_i = 0.$$

*The edge points are the points on the opposite edge/triangle/... of a corner point.*

We shall now look at the most important properties of the sets of ERBS basis functions in homogeneous barycentric coordinates, $\mathcal{B}_{k,i}(\mathbf{u})$, definition 6.4. The properties will be described in the following Theorem.

**Theorem 6.1.** *There are five important properties for the sets of the k ERBS basis-functions, $\mathcal{B}_{k,i}(\mathbf{u})$, $i = 1,2,...,k$, in homogeneous barycentric coordinates, where $k = 2,3,....$ For these properties it is assumed that $\mathbf{u} \in \Xi_{k-1}$ (def. 6.1), and $\widehat{\mathbf{u}}_i$, for $i = 1,2,...,k$ are corner points, and $\overline{\mathbf{u}}_i$, for $i = 1,2,...,k$ are edge points (def. 6.5). These properties are:*

**P1.** $\mathcal{B}_{k,i}(\mathbf{u}) \begin{cases} > 0, & \text{if } u_i > 0, \\ = 0, & \text{otherwise}, \end{cases}$

**P2.** $\sum_{i=1}^{k} \mathcal{B}_{k,i}(\mathbf{u}) = 1,$

**P3.** $\mathcal{B}_{k,i}(\widehat{\mathbf{u}}_i) = 1,$

**P4.** $D^r_{\mathbf{d}_1} D^s_{\mathbf{d}_2} \mathcal{B}_{k,i}(\widehat{\mathbf{u}}_j) = 0$ *for* $j = 1,2,...,k$, *where* $\mathbf{d}_1, \mathbf{d}_2 \in \Upsilon_{k-1}$, *and* $r+s > 0$,

**P5.** $D^r_{\mathbf{d}_1} D^s_{\mathbf{d}_2} \mathcal{B}_{k,i}(\overline{\mathbf{u}}_i) = 0$ *where* $\mathbf{d}_1, \mathbf{d}_2 \in \Upsilon_{k-1}$, *and* $r+s \geq 0$.

*Proof.* The proof is organized in a numbered list, where each number matches the number of the respective property.

1. Property 1 follows from definition 6.4. Because of property 1 in Theorem 2.1, the denominator will always be $> 0$. The numerator however is $> 0$ when $u_i > 0$, and 0, if $u_i = 0$.
2. Property 2 means, fulfilling the partition of unity, and it follows from
$$\sum_{i=1}^{k} \mathcal{B}_{k,i}(\mathbf{u}) = \frac{\sum_{i=1}^{k} B(u_i)}{\sum_{j=1}^{k} B(u_j)} = 1, \quad \text{if} \quad \mathbf{u} \in \Xi_{k-1}.$$
3. Property 3 defines the "top" corner, and it follows from Theorem 2.1, property 2, because the denominator in equation (6.3) also becomes $B(u_i)$.
4. This property says that all derivatives in all directions are zero at all corners. This follows from the fact that all partial derivatives are zero at the corners because all basic functions (definition 6.3) are zero at $u = 0$ (which follows from property 1 in Theorem 2.1), and all their derivatives are zero at $u = 0$ and $u = 1$ (which follows from property 6 in Theorem 2.1). This can clearly be seen in the equations from (6.4) to (6.9). And it is also clear that this will be the case for higher derivatives.
5. Property 5 says that both the value and all derivatives in all directions are zero at the edge/triangle/etc., opposite to the "top" corner (where the value is 1). This follows from the fact that all partial derivatives are zero at that edge/triangle/etc., because, as we can see, the equations from (6.4) to (6.9) all have either a factor $B(u_i)$ or $D^j B(u_i)$ that is zero because $u_i = 0$ when $\mathbf{u} = \overline{\mathbf{u}}_i$ . It is also clear that this will be the case also for higher derivatives.

□

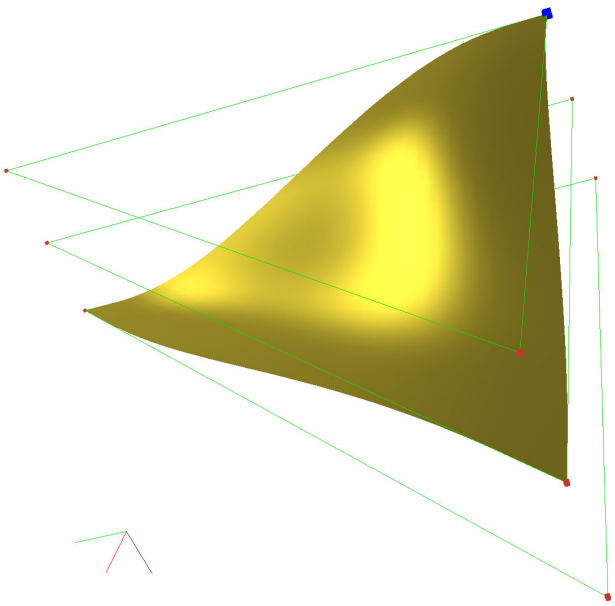

***Figure 6.5:*** *An Expo-Rational B-spline triangle surface with its three local triangles. The boundary of the local triangles are marked with green lines. As can be seen, the local triangles are all 1st degree Bézier triangular patches, and they are parallel to each other.*

## 6.4 ERBS triangles, definition and evaluators

The general formula for the ERBS triangle is,

$$S(u,v,w) = \sum_{i=1}^{3} s_i(u,v,w)\ \mathcal{B}_{3,i}(u,v,w), \quad \text{where } u+v+w=1 \text{ and } u,v,w \geq 0, \qquad (6.12)$$

where $s_i(u,v,w),\ i = 1,2,3$ are the local triangles. The construction is quite simple and unlike the other ERBS constructions earlier in this report, the knot vector is only the interval $[0,1]$ deduced from the homogeneous barycentric coordinates. We do not have to consider the usual global/local affine mapping (definition 2.7). In figures 6.5, 6.6 and 6.7 there are three plots of ERBS triangles and their 3 local triangles. All the local triangles are actually planar in all the figures, and we can see their boundary/control polygon marked as green lines. In the first figure we can see that they are also parallel. We can also see that the global ERBS triangle patches are interpolating each of the local triangle patches in the corners, and we know from the ERBS properties (Theorem 6.1) that the interpolation not only involves the position, it actually includes interpolation of all available derivatives.

It is preferable for several reasons that the local surfaces are oriented in such a way that the full support of the first parameter (i.e. $u = 1$) is in the interpolation point to the global ERBS triangle patch. This simplifies the construction and organization of the local

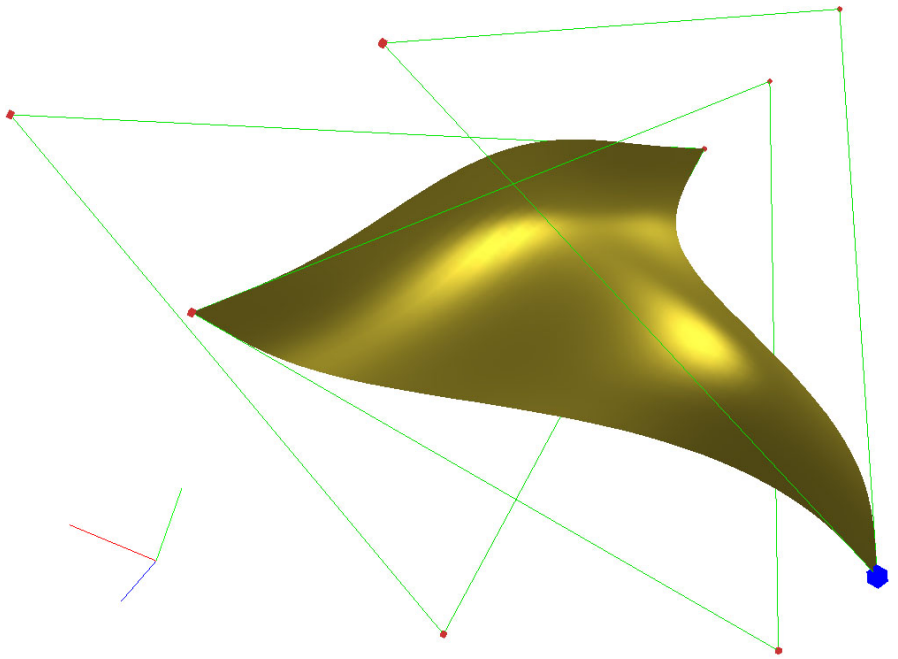

***Figure 6.6:*** *An Expo-Rational B-spline triangular surface where the local triangles initially were the same as in Figure 6.5, but are now translated and rotated. The blue cube shows us the interpolation point of the first local triangle.*

triangle patches, but it introduces the need for a new type of global/local mapping of the parameters between the global triangle patch and the local triangle patches for each of the vertices. This mapping is actually only a cyclic use of the parameters. This cyclic mapping is implemented in the following example,

$$\begin{aligned} S(u,v,w) = & \ \overline{s}_1(u,v,w)\ \mathcal{B}_{3,1}(u,v,w) + \\ & \ \overline{s}_2(v,w,u)\ \mathcal{B}_{3,2}(u,v,w) + \\ & \ \overline{s}_3(w,u,v)\ \mathcal{B}_{3,3}(u,v,w), \end{aligned} \tag{6.13}$$

where $u+v+w=1$ and $u,v,w \geq 0$, and where $\overline{s}_i,\ i=1,2,3$ are the local triangles, and are orientated (as described above) in such a way that the full support of the first parameter (i.e. the "local $u=1$") is in the interpolation point to the global ERBS triangle. This expression is straightforward to implement.

The general computation of a directional derivative for a given $\mathbf{d} \in \Upsilon_2$ is,

$$\begin{aligned} D_{\mathbf{d}}\, S(u,v,w) = & \ D_{\mathbf{d}}\, \overline{s}_1(u,v,w)\ \mathcal{B}_{3,1}(u,v,w) + \overline{s}_1(u,v,w)\ D_{\mathbf{d}}\, \mathcal{B}_{3,1}(u,v,w) + \\ & \ D_{\mathbf{d}}\, \overline{s}_2(v,w,u)\ \mathcal{B}_{3,2}(u,v,w) + \overline{s}_2(v,w,u)\ D_{\mathbf{d}}\, \mathcal{B}_{3,2}(u,v,w) + \\ & \ D_{\mathbf{d}}\, \overline{s}_3(w,u,v)\ \mathcal{B}_{3,3}(u,v,w) + \overline{s}_3(w,u,v)\ D_{\mathbf{d}}\, \mathcal{B}_{3,3}(u,v,w), \end{aligned} \tag{6.14}$$

where $D_{\mathbf{d}}\ \mathcal{B}_{3,i}(u,v,w),\ i=1,2,3$ is described in (6.10). If the local triangles $\overline{s}_i(u,v,w)$, $i=1,2,3$ are Bézier triangles, then the directional derivatives can be found in (6.1).

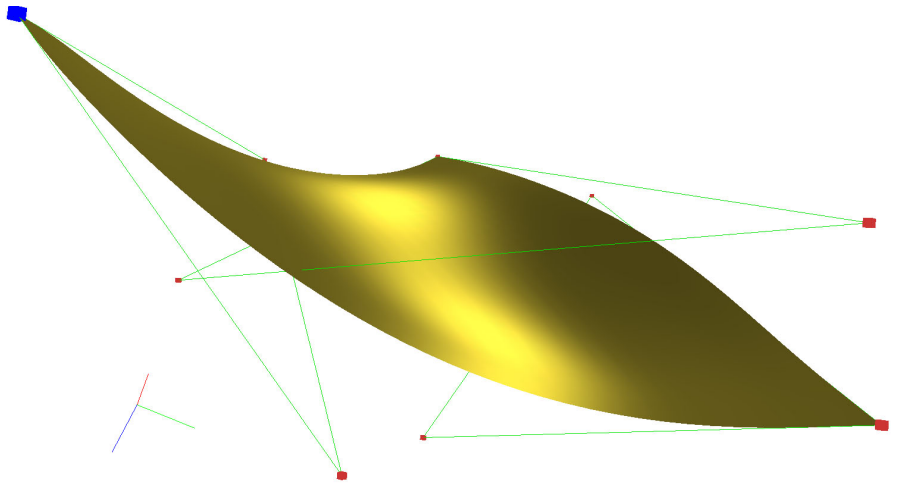

***Figure 6.7:*** *An Expo-Rational B-spline triangular surface where also the local triangles are all 1st degree Bézier triangular patches as in Figure 6.5 and in Figure 6.6. The shape can vary tremendous only by affine transformations of the local patches.*

Recall the definition of the two main directions for derivatives in (6.11), $\mathbf{d}_1 = (-1,1,0)$ and $\mathbf{d}_2 = (-1,0,1)$. These main directions can be used for computing normals or gaussian/mean/principal curvatures. Because of the cycling of the parameters the partial derivatives for the local triangles in the following equations will be denoted; $D_1$ for the partial derivatives as regards the first parameter, $D_2$ for the partial derivatives as regards the second parameter and $D_3$ for the partial derivatives as regards the third and last parameters. The formula for the derivatives in the first main direction is then,

$$\begin{aligned} D_{\mathbf{d}_1} S(u,v,w) = &\left(D_2\, \bar{s}_1(u,v,w) - D_1\, \bar{s}_1(u,v,w)\right) \mathcal{B}_{3,1}(u,v,w) \\ &+\bar{s}_1(u,v,w)\left(D_v\, \mathcal{B}_{3,1}(u,v,w) - D_u\, \mathcal{B}_{3,1}(u,v,w)\right) \\ &+\left(D_2\, \bar{s}_2(v,w,u) - D_3\, \bar{s}_2(v,w,u)\right) \mathcal{B}_{3,2}(u,v,w) \\ &+\bar{s}_2(v,w,u)\left(D_v\, \mathcal{B}_{3,2}(u,v,w) - D_u\, \mathcal{B}_{3,2}(u,v,w)\right) \\ &+\left(D_3\, \bar{s}_3(w,u,v) - D_1\, \bar{s}_3(w,u,v)\right) \mathcal{B}_{3,3}(u,v,w) \\ &+\bar{s}_3(w,u,v)\left(D_v\, \mathcal{B}_{3,3}(u,v,w) - D_u\, \mathcal{B}_{3,3}(u,v,w)\right). \end{aligned} \tag{6.15}$$

For the derivatives in the second main direction the formula is,

$$\begin{aligned} D_{\mathbf{d}_2} S(u,v,w) = &\left(D_3\, \bar{s}_1(u,v,w) - D_2\, \bar{s}_1(u,v,w)\right) \mathcal{B}_{3,1}(u,v,w) \\ &+\bar{s}_1(u,v,w)\left(D_w\, \mathcal{B}_{3,1}(u,v,w) - D_u\, \mathcal{B}_{3,1}(u,v,w)\right) \\ &+\left(D_1\, \bar{s}_2(v,w,u) - D_3\, \bar{s}_2(v,w,u)\right) \mathcal{B}_{3,2}(u,v,w) \\ &+\bar{s}_2(v,w,u)\left(D_w\, \mathcal{B}_{3,2}(u,v,w) - D_u\, \mathcal{B}_{3,2}(u,v,w)\right) \\ &+\left(D_2\, \bar{s}_3(w,u,v) - D_1\, \bar{s}_3(w,u,v)\right) \mathcal{B}_{3,3}(u,v,w) \\ &+\bar{s}_3(w,u,v)\left(D_w\, \mathcal{B}_{3,3}(u,v,w) - D_u\, \mathcal{B}_{3,3}(u,v,w)\right). \end{aligned} \tag{6.16}$$

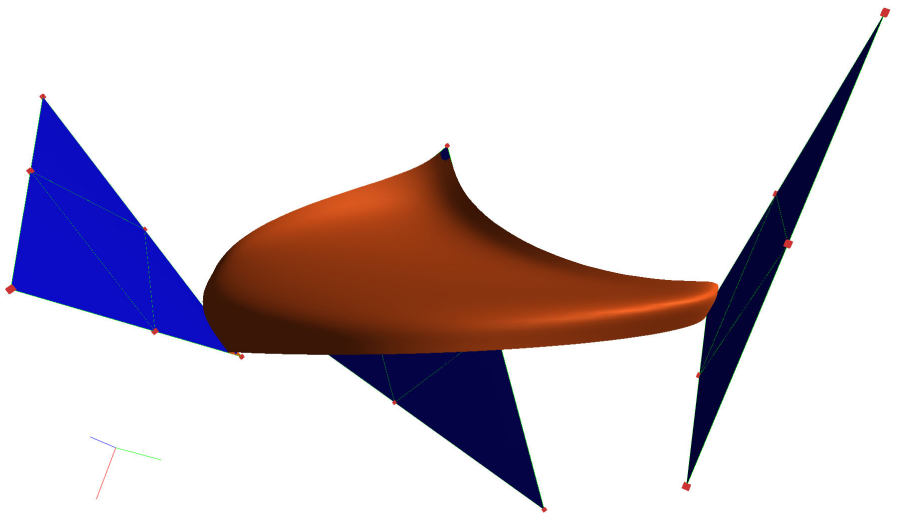

***Figure 6.8:*** *An Expo-Rational B-spline triangular surface is plotted. The local triangles are 2nd degree Bézier triangles (blue), but still planar. The control polygons of the Bézier triangles are shown as green lines, and the control points as red cubes.*

For the second order derivatives, the process has to be repeated ones more. We then get,

$$\begin{aligned}
D^2_{\mathbf{d}_1} S(u,v,w) = &\left(D^2_2\,\bar{s}_1(u,v,w) - 2D_2D_1\,\bar{s}_1(u,v,w) + D^2_1\,\bar{s}_1(u,v,w)\right)\mathcal{B}_{3,1}(u,v,w)\\
&+2\left(D_2\,\bar{s}_1(u,v,w) - D_1\,\bar{s}_1(u,v,w)\right)\left(D_v\,\mathcal{B}_{3,1}(u,v,w) - D_u\,\mathcal{B}_{3,1}(u,v,w)\right)\\
&+\bar{s}_1(u,v,w)\left(D^2_v\,\mathcal{B}_{3,1}(u,v,w) - 2D_uD_v\,\mathcal{B}_{3,1}(u,v,w) + D^2_u\,\mathcal{B}_{3,1}(u,v,w)\right)\\
&+\left(D^2_2\,\bar{s}_2(v,w,u) - 2D_2D_3\,\bar{s}_2(v,w,u) + D^2_3\,\bar{s}_2(v,w,u)\right)\mathcal{B}_{3,2}(u,v,w)\\
&+2\left(D_2\,\bar{s}_2(v,w,u) - D_1\,\bar{s}_2(v,w,u)\right)\left(D_v\,\mathcal{B}_{3,2}(u,v,w) - D_u\,\mathcal{B}_{3,2}(u,v,w)\right)\\
&+\bar{s}_2(v,w,u)\left(D^2_v\,\mathcal{B}_{3,2}(u,v,w) - 2D_uD_v\,\mathcal{B}_{3,2}(u,v,w) + D^2_u\,\mathcal{B}_{3,2}(u,v,w)\right)\\
&+\left(D^2_3\,\bar{s}_3(w,u,v) - 2D_1D_3\,\bar{s}_3(w,u,v) + D^2_1\,\bar{s}_3(w,u,v)\right)\mathcal{B}_{3,3}(u,v,w)\\
&+2\left(D_3\,\bar{s}_3(w,u,v) - D_1\,\bar{s}_3(w,u,v)\right)\left(D_v\,\mathcal{B}_{3,3}(u,v,w) - D_u\,\mathcal{B}_{3,3}(u,v,w)\right)\\
&+\bar{s}_3(w,u,v)\left(D^2_v\,\mathcal{B}_{3,3}(u,v,w) - 2D_uD_v\,\mathcal{B}_{3,3}(u,v,w) + D^2_u\,\mathcal{B}_{3,3}(u,v,w)\right).
\end{aligned} \tag{6.17}$$

The other second order derivatives, $D^2_{\mathbf{d}_2}\,S(u,v,w)$ and $D_{\mathbf{d}_1}D_{\mathbf{d}_2}\,S(u,v,w)$ can be computed in the same way, using the same template. They will however not be computed here, but it is easy to compute them when this is required.

In Figure 6.8 there is an Expo-Rational B-spline triangular surface using 2nd degree Bézier triangles as local triangles. One can see that the local triangles are still planar. This means that the number of coefficients (red cubes in the figures) is six for each local triangular Bézier patch. This six cubes are now in such a position that the respective local Bézier triangular patch is linear (compare with the previous figures 6.5, 6.6 and 6.7, where the local patches were 1st degree triangular Bézier patches (linear by default), only with

three coefficients each (red cubes in the figures)). The difference now is that by clicking on one of the respective six cubes in Figure 6.8, it is possible to edit the local Bezier triangle into a "true" quadratic patch, which was impossible to do in the case of figures 6.5, 6.6 and 6.7. This is, actually what we have done in the next figure.

In Figure 6.9 there are three different ERBS triangular surfaces, where the local triangles are not planar. These figures give a small insight into the possibilities of designing by using ERBS triangular surfaces.

## 6.5 Local Bézier triangles and Hermite interpolation

We start by recalling the settings from section 2.8, and adapting parts of these to ERBS triangles. At first, we will only interpolate a triangular part of a given surface with one ERBS triangle. The setting is:

- Given is a surface $g : \Omega_g \subset \mathbb{R}^2 \to \mathbb{R}^n$, where we might have $n = 1, 2, 3, ...,$
- Given are three points in the domain of $g$ i.e. $\Omega_g$. These points are denoted by $p_k$, $k = 1, 2, 3$ (they can be defined by either ordinary coordinates $(\mu, \nu)$ or by barycentric coordinates $(u, v, w)$). Together with each of these points, a respective number $d_k$ for $k = 1, 2, 3$, must be given. This numbers, $d_k$, gives us the degree of the local Bézier triangles associated with the respective point $p_k$ defining a triangle in the domain $\Omega_g$
- Make an ERBS triangle $S(u, v, w)$ by using the three points $p_i \in \Omega_g$ and their respective degrees $d_k$. This must be done by generating the three local triangles $\bar{s}_i(u, v, w)$, $i = 1, 2, 3$ (see equation 6.13 and the description below 6.13) in such a way that the ERBS triangle is Hermite interpolating the local triangles in each corner, so that: at the first corner

$$D^i_{(-1,1,0)} D^j_{(-1,0,1)} \bar{s}_1(1,0,0) = D^i_{(-1,1,0)} D^j_{(-1,0,1)} S(1,0,0) = D^i_{p_2-p_1} D^j_{p_3-p_1} g(p_1), \tag{6.18}$$

at the second corner:

$$D^i_{(-1,1,0)} D^j_{(-1,0,1)} \bar{s}_2(1,0,0) = D^i_{(0,-1,1)} D^j_{(1,-1,0)} S(0,1,0) = D^i_{p_3-p_2} D^j_{p_1-p_2} g(p_2), \tag{6.19}$$

and at the third corner:

$$D^i_{(-1,1,0)} D^j_{(-1,0,1)} \bar{s}_3(1,0,0) = D^i_{(1,0,-1)} D^j_{(0,1,-1)} S(0,0,1) = D^i_{p_1-p_3} D^j_{p_2-p_3} g(p_3), \tag{6.20}$$

is fulfilled, where for the respective three equations, it holds

$$0 \leq (i + j) \leq d_k, \quad \text{for } k = 1, 2, 3. \tag{6.21}$$

The right hand side of (6.18–6.20) looks like the ordinary Hermite interpolation method used for generating an approximation of a triangular part of a parametric surface. The

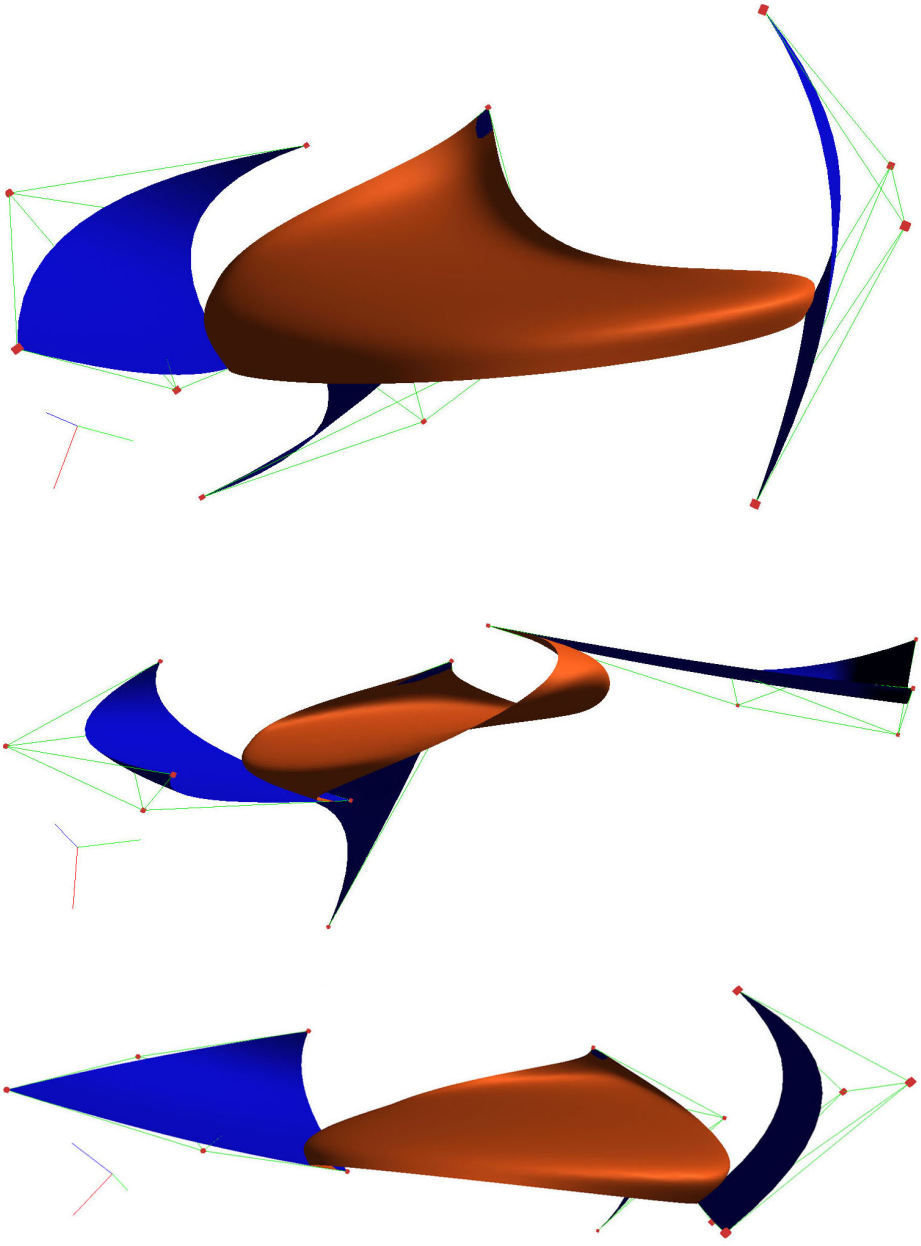

***Figure 6.9:*** *Three different Expo-Rational B-spline triangular surfaces are plotted. The local triangles are 2nd degree Bézier triangles (not planar any more), and we can see them as blue triangles, the control polygons of the local Bézier triangles as green lines, and their control points as red cubes.*

specific part is, however, the generation of the local triangles on the left hand side of (6.18–6.20), and thus the rotational global/local mapping. Remember from Theorem 6.1 that an ERBS triangular surface completely interpolates (position and all derivatives) its local triangles at the corners because of the properties P3 and P4 in Theorem 6.1. This is, of course, the basis for the equations (6.18–6.21), and it can also clearly be seen in figures 6.5–6.9. The consequence of this is that the Hermite interpolation by the local surfaces must only be done at one point for each local triangular surface, where the position and the derivatives from the original surface $g$ uniquely determine the local surface.

Using Bézier triangles as local triangles will imply that we have to look at Hermite interpolation by Bézier triangles. Let us start by making a generalized version of the leftmost side of the equations (6.18–6.20), together with the rightmost side of the equations (6.18–6.20). We then get,

$$D^i_{(-1,1,0)} D^j_{(-1,0,1)} s(1,0,0) = D^i_{\mathbf{d}_1} D^j_{\mathbf{d}_2} g(\mathbf{p}), \quad \text{where} \quad 0 \leq (i+j) \leq d, \tag{6.22}$$

where $d$ is the degree of the Bézier triangle, $\mathbf{p}$ is a point in $\Omega_g$, and $\mathbf{d}_1, \mathbf{d}_2$ are vectors $\in \mathbb{R}^2$ at the point $\mathbf{p}$. Recall that the number of equations in (6.22) is

$$n_b = \sum_{i=1}^{d+1} i,$$

which follows from the fact that the sum of the possible variants, for $i+j=d$ is $d+1$ when $i, j \geq 0$. For $d=1$, we get $n_b = 3$, for $d=2$, we get $n_b = 6$, and for $d=3$, we get $n_b = 10$. As we can see, this matches the relationship between the degree and the number of coefficients/basis functions of a Bézier triangle. Equation (6.22) is, therefore, uniquely determining the Bézier triangle from the position and derivatives of an original surface.

To compute the equations for the Hermite interpolation we need the formula for the directional derivatives for the Bézier triangle. The first (directional) derivatives are given in (6.1) and (6.2). In the following three formulas there are equations for one, two and three directional derivatives. They are general, so that they can be used for second, third, or various cross derivatives,

$$D_{\mathbf{p}} s(\mathbf{u}) = p_1 \, D_u s(\mathbf{u}) + p_2 \, D_v s(\mathbf{u}) + p_3 \, D_w s(\mathbf{u}), \tag{6.23}$$

$$\begin{aligned} D_{\mathbf{q}} D_{\mathbf{p}} s(\mathbf{u}) = {} & q_1 p_1 D_u^2 s(\mathbf{u}) + q_1 p_2 \, D_u D_v s(\mathbf{u}) + q_1 p_3 \, D_u D_w s(\mathbf{u}) \\ & + v q_2 p_1 \, D_u D_v s(\mathbf{u}) + q_2 p_2 \, D_v^2 s(\mathbf{u}) + q_2 p_3 \, D_v D_w s(\mathbf{u}) \\ & + q_3 p_1 \, D_u D_w s(\mathbf{u}) + q_3 p_2 \, D_v D_w s(\mathbf{u}) + q_3 p_3 \, D_w^2 s(\mathbf{u}), \end{aligned} \tag{6.24}$$

$$\begin{aligned}
D_{\mathbf{h}}D_{\mathbf{q}}D_{\mathbf{p}}s(\mathbf{u}) = {} & h_1q_1p_1\ D_u^3s(\mathbf{u}) \\
& +(h_1q_1p_2+h_1q_2p_1+h_2q_1p_1)\ D_u^2D_vs(\mathbf{u}) \\
& +(h_1q_2p_2+h_2q_1p_2+h_2q_2p_1)\ D_uD_v^2s(\mathbf{u}) \\
& +(h_1q_1p_3+h_1q_3p_1+h_3q_1p_1)\ D_u^2D_ws(\mathbf{u}) \\
& +(h_1q_3p_3+h_3q_1p_3+h_3q_3p_1)\ D_uD_w^2s(\mathbf{u}) \\
& +h_2q_2p_2\ D_v^3s(\mathbf{u}) \\
& +(h_2q_2p_3+h_2q_3p_2+h_3q_2p_2)\ D_v^2D_ws(\mathbf{u}) \\
& +(h_2q_3p_3+h_3q_2p_3+h_3q_3p_2)\ D_vD_w^2s(\mathbf{u}) \\
& +h_3q_3p_3\ D_w^3s(\mathbf{u}).
\end{aligned} \tag{6.25}$$

We will now look at Hermite interpolation by a 1st degree Bézier triangle. For the 1st degree Bézier triangle,

$$s(u,v,w) = u\mathbf{c}_1 + v\mathbf{c}_2 + w\mathbf{c}_3.$$

The partial derivatives are

$$\begin{aligned}
D_us(u,v,w) &= \mathbf{c}_1, \\
D_vs(u,v,w) &= \mathbf{c}_2, \\
D_ws(u,v,w) &= \mathbf{c}_3,
\end{aligned}$$

and using the equation and the partial derivatives together with equation (6.23), we get

$$\begin{aligned}
s(1,0,0) &= \mathbf{c}_1, \\
D_{(-1,1,0)}s(1,0,0) &= \mathbf{c}_2-\mathbf{c}_1, \\
D_{(-1,0,1)}s(1,0,0) &= \mathbf{c}_3-\mathbf{c}_1.
\end{aligned}$$

Reorganizing this, the formula for the Hermite interpolation of a parametric surface by the 1st degree Bézier triangle becomes

$$\begin{aligned}
\mathbf{c}_1 &= g(\mathbf{p}), \\
\mathbf{c}_2 &= \mathbf{c}_1 + D_{\mathbf{d}_1}g(\mathbf{p}), \\
\mathbf{c}_3 &= \mathbf{c}_1 + D_{\mathbf{d}_2}g(\mathbf{p}).
\end{aligned}$$

The Hermite interpolation of a 2nd degree Bézier triangle is a little bit more complex. We start with the equation for the 2nd degree Bézier triangle

$$s(u,v,w) = u^2\mathbf{c}_1 + 2uv\mathbf{c}_2 + 2uw\mathbf{c}_3 + v^2\mathbf{c}_4 + 2vw\mathbf{c}_5 + w^2\mathbf{c}_6.$$

The partial derivatives are

$$\begin{aligned}
D_u s(u,v,w) &= 2u\mathbf{c}_1 + 2v\mathbf{c}_2 + 2w\mathbf{c}_3,\\
D_v s(u,v,w) &= 2u\mathbf{c}_2 + 2v\mathbf{c}_4 + 2w\mathbf{c}_5,\\
D_w s(u,v,w) &= 2u\mathbf{c}_3 + 2v\mathbf{c}_5 + 2w\mathbf{c}_6,\\
D_u^2 s(u,v,w) &= 2\mathbf{c}_1,\\
D_v^2 s(u,v,w) &= 2\mathbf{c}_4,\\
D_w^2 s(u,v,w) &= 2\mathbf{c}_6,\\
D_u D_v s(u,v,w) &= 2\mathbf{c}_2,\\
D_u D_w s(u,v,w) &= 2\mathbf{c}_3,\\
D_v D_w s(u,v,w) &= 2\mathbf{c}_5,
\end{aligned}$$

and using the equation and the partial derivatives together with equation (6.24), we get

$$\begin{aligned}
s(1,0,0) &= \mathbf{c}_1,\\
D_{(-1,1,0)} s(1,0,0) &= 2(\mathbf{c}_2 - \mathbf{c}_1),\\
D_{(-1,0,1)} s(1,0,0) &= 2(\mathbf{c}_3 - \mathbf{c}_1),\\
D^2_{(-1,1,0)} s(1,0,0) &= 2(\mathbf{c}_1 - 2\mathbf{c}_2 + \mathbf{c}_4),\\
D^2_{(-1,0,1)} s(1,0,0) &= 2(\mathbf{c}_1 - 2\mathbf{c}_3 + \mathbf{c}_6),\\
D_{(-1,0,1)} D_{(-1,1,0)} s(1,0,0) &= 2(\mathbf{c}_1 - \mathbf{c}_2 - \mathbf{c}_3 + \mathbf{c}_5).
\end{aligned}$$

Reorganizing this, the formula for the Hermite interpolation of a parametric surface by the 2nd degree Bézier triangle becomes

$$\begin{aligned}
\mathbf{c}_1 &= g(\mathbf{p}),\\
\mathbf{c}_2 &= \mathbf{c}_1 + \frac{1}{2} D_{\mathbf{d}_1} g(\mathbf{p}),\\
\mathbf{c}_3 &= \mathbf{c}_1 + \frac{1}{2} D_{\mathbf{d}_2} g(\mathbf{p}),\\
\mathbf{c}_4 &= -\mathbf{c}_1 + 2\mathbf{c}_2 + \frac{1}{2} D^2_{\mathbf{d}_1} g(\mathbf{p}),\\
\mathbf{c}_5 &= -\mathbf{c}_1 + \mathbf{c}_2 + \mathbf{c}_3 + \frac{1}{2} D_{\mathbf{d}_1} D_{\mathbf{d}_2} g(\mathbf{p}),\\
\mathbf{c}_6 &= -\mathbf{c}_1 + 2\mathbf{c}_3 + \frac{1}{2} D^2_{\mathbf{d}_2} g(\mathbf{p}).
\end{aligned}$$

The Hermite interpolation of a 3rd degree Bézier triangle is, without a doubt, even more complex. We start with the equation for the 3rd degree Bézier triangle

$$\begin{aligned}
s(u,v,w) = {} & u^3\mathbf{c}_1 + 3u^2 v\mathbf{c}_2 + 3u^2 w\mathbf{c}_3 + 3uv^2\mathbf{c}_4 + 6uvw\mathbf{c}_5\\
& + 3uw^2\mathbf{c}_6 + v^3\mathbf{c}_7 + 3v^2 w\mathbf{c}_8 + 3vw^2\mathbf{c}_9 + w^3\mathbf{c}_{10}.
\end{aligned}$$

The partial derivatives are

$$
\begin{aligned}
D_u s(u,v,w) &= 3(u^2\mathbf{c}_1 + 2uv\mathbf{c}_2 + 2uw\mathbf{c}_3 + v^2\mathbf{c}_4 + 2vw\mathbf{c}_5 + w^2\mathbf{c}_6),\\
D_v s(u,v,w) &= 3(u^2\mathbf{c}_2 + 2uv\mathbf{c}_4 + 2uw\mathbf{c}_5 + v^2\mathbf{c}_7 + 2vw\mathbf{c}_8 + w^2\mathbf{c}_9),\\
D_w s(u,v,w) &= 3(u^2\mathbf{c}_3 + 2uv\mathbf{c}_5 + 2uw\mathbf{c}_6 + v^2\mathbf{c}_8 + 2vw\mathbf{c}_9 + w^2\mathbf{c}_{10}),\\
D_u^2 s(u,v,w) &= 6(u\mathbf{c}_1 + v\mathbf{c}_2 + w\mathbf{c}_3),\\
D_v^2 s(u,v,w) &= 6(u\mathbf{c}_4 + v\mathbf{c}_7 + w\mathbf{c}_8),\\
D_w^2 s(u,v,w) &= 6(u\mathbf{c}_6 + v\mathbf{c}_9 + w\mathbf{c}_{10}),\\
D_u D_v s(u,v,w) &= 6(u\mathbf{c}_2 + v\mathbf{c}_4 + w\mathbf{c}_5),\\
D_u D_w s(u,v,w) &= 6(u\mathbf{c}_3 + v\mathbf{c}_5 + w\mathbf{c}_6),\\
D_v D_w s(u,v,w) &= 6(u\mathbf{c}_5 + v\mathbf{c}_8 + w\mathbf{c}_9),\\
D_u^3 s(u,v,w) &= 6\mathbf{c}_1,\\
D_v^3 s(u,v,w) &= 6\mathbf{c}_7,\\
D_w^3 s(u,v,w) &= 6\mathbf{c}_{10},\\
D_v D_u^2 s(u,v,w) &= 6\mathbf{c}_2,\\
D_w D_u^2 s(u,v,w) &= 6\mathbf{c}_3,\\
D_u D_v^2 s(u,v,w) &= 6\mathbf{c}_4,\\
D_w D_v^2 s(u,v,w) &= 6\mathbf{c}_8,\\
D_u D_w^2 s(u,v,w) &= 6\mathbf{c}_6,\\
D_v D_w^2 s(u,v,w) &= 6\mathbf{c}_9,\\
D_u D_v D_w s(u,v,w) &= 6\mathbf{c}_5,
\end{aligned}
$$

and using the equation and the partial derivatives together with equation (6.25), we get

$$
\begin{aligned}
s(1,0,0) &= \mathbf{c}_1,\\
D_{(-1,1,0)} s(1,0,0) &= 3(\mathbf{c}_2 - \mathbf{c}_1),\\
D_{(-1,0,1)} s(1,0,0) &= 3(\mathbf{c}_3 - \mathbf{c}_1),\\
D^2_{(-1,1,0)} s(1,0,0) &= 6(\mathbf{c}_1 - 2\mathbf{c}_2 + \mathbf{c}_4),\\
D^2_{(-1,0,1)} s(1,0,0) &= 6(\mathbf{c}_1 - 2\mathbf{c}_3 + \mathbf{c}_6),\\
D_{(-1,0,1)} D_{(-1,1,0)} s(1,0,0) &= 6(\mathbf{c}_1 - \mathbf{c}_2 - \mathbf{c}_3 + \mathbf{c}_5),\\
D^3_{(-1,1,0)} s(1,0,0) &= 6(-\mathbf{c}_1 + 3\mathbf{c}_2 - 3\mathbf{c}_4 + \mathbf{c}_7),\\
D^3_{(-1,0,1)} s(1,0,0) &= 6(-\mathbf{c}_1 + 3\mathbf{c}_3 - 3\mathbf{c}_6 + \mathbf{c}_{10}),\\
D_{(-1,1,0)} D^2_{(-1,0,1)} s(1,0,0) &= 6(-\mathbf{c}_1 + \mathbf{c}_2 + 2\mathbf{c}_3 - 2\mathbf{c}_5 - \mathbf{c}_6 + \mathbf{c}_9),\\
D_{(-1,0,1)} D^2_{(-1,1,0)} s(1,0,0) &= 6(-\mathbf{c}_1 + 2\mathbf{c}_2 + \mathbf{c}_3 - \mathbf{c}_4 - 2\mathbf{c}_5 + \mathbf{c}_8).
\end{aligned}
$$

Reorganizing this, the formula for the Hermite interpolation of a parametric surface by the 3rd degree Bézier triangle becomes

$$
\begin{aligned}
\mathbf{c}_1 &= g(\mathbf{p}),\\
\mathbf{c}_2 &= \mathbf{c}_1 + \frac{1}{3} D_{\mathbf{d}_1} g(\mathbf{p}),\\
\mathbf{c}_3 &= \mathbf{c}_1 + \frac{1}{3} D_{\mathbf{d}_2} g(\mathbf{p}),\\
\mathbf{c}_4 &= -\mathbf{c}_1 + 2\mathbf{c}_2 + \frac{1}{6} D^2_{\mathbf{d}_1} g(\mathbf{p}),\\
\mathbf{c}_5 &= -\mathbf{c}_1 + \mathbf{c}_2 + \mathbf{c}_3 + \frac{1}{6} D_{\mathbf{d}_1} D_{\mathbf{d}_2} g(\mathbf{p}),\\
\mathbf{c}_6 &= -\mathbf{c}_1 + 2\mathbf{c}_3 + \frac{1}{6} D^2_{\mathbf{d}_2} g(\mathbf{p}),\\
\mathbf{c}_7 &= \mathbf{c}_1 - 3\mathbf{c}_2 + 3\mathbf{c}_4 + \frac{1}{6} D^3_{\mathbf{d}_1} g(\mathbf{p}),\\
\mathbf{c}_8 &= \mathbf{c}_1 - 2\mathbf{c}_2 - \mathbf{c}_3 + \mathbf{c}_4 + 2\mathbf{c}_5 + \frac{1}{6} D^2_{\mathbf{d}_1} D_{\mathbf{d}_2} g(\mathbf{p}),\\
\mathbf{c}_9 &= \mathbf{c}_1 - \mathbf{c}_2 - 2\mathbf{c}_3 + 2\mathbf{c}_5 + \mathbf{c}_6 + \frac{1}{6} D_{\mathbf{d}_1} D^2_{\mathbf{d}_1} g(\mathbf{p}),\\
\mathbf{c}_{10} &= \mathbf{c}_1 - 3\mathbf{c}_3 + 3\mathbf{c}_6 + \frac{1}{6} D^3_{\mathbf{d}_2} g(\mathbf{p}).
\end{aligned}
$$

As we can see above, to compute the coefficients for the Bézier triangles, we need to compute directional derivatives of the original surface $g$, both first, second, third, and the cross derivatives between two directions and so on, for higher-order Hermite interpolation. We must have a position $\mathbf{p}$ and two directions, $\mathbf{d}_1 = (u_1, v_1)$ and $\mathbf{d}_2 = (u_2, v_2)$, in the parameter plane of the original surface $g$. The first order derivatives in the two directions are then

$$
\begin{aligned}
D_{\mathbf{d}_1}\, g(\mathbf{p}) &= u_1 D_u g(\mathbf{p}) + v_1 D_v g(\mathbf{p}),\\
D_{\mathbf{d}_2}\, g(\mathbf{p}) &= u_2 D_u g(\mathbf{p}) + v_2 D_v g(\mathbf{p}).
\end{aligned}
$$

To compute the second order derivative we need to use two steps. The first step is

$$
\begin{aligned}
D_{\mathbf{d}_1}\, D_u g(\mathbf{p}) &= u_1 D_u D_u g(\mathbf{p}) + v_1 D_v D_u g(\mathbf{p}),\\
D_{\mathbf{d}_2}\, D_u g(\mathbf{p}) &= u_2 D_u D_u g(\mathbf{p}) + v_2 D_v D_u g(\mathbf{p}),\\
D_{\mathbf{d}_1}\, D_v g(\mathbf{p}) &= u_1 D_u D_v g(\mathbf{p}) + v_1 D_v D_v g(\mathbf{p}),\\
D_{\mathbf{d}_2}\, D_v g(\mathbf{p}) &= u_2 D_u D_v g(\mathbf{p}) + v_2 D_v D_v g(\mathbf{p}),
\end{aligned}
$$

and the second step gives us the second order derivatives in the two directions and the cross derivative, as follows:

$$
\begin{aligned}
D^2_{\mathbf{d}_1}\, g(\mathbf{p}) &= u_1 D_u D_{\mathbf{d}_1} g(\mathbf{p}) + v_1 D_v D_{\mathbf{d}_1} g(\mathbf{p}),\\
D^2_{\mathbf{d}_2}\, g(\mathbf{p}) &= u_2 D_u D_{\mathbf{d}_2} g(\mathbf{p}) + v_2 D_v D_{\mathbf{d}_2} g(\mathbf{p}),\\
D_{\mathbf{d}_1} D_{\mathbf{d}_2}\, g(\mathbf{p}) &= u_1 D_u D_{\mathbf{d}_2} g(\mathbf{p}) + v_1 D_v D_{\mathbf{d}_2} g(\mathbf{p}).
\end{aligned}
$$

To compute the third order derivative we need to use three steps. The first step is

$$\begin{aligned}
D_{\mathbf{d}_1}\, D_u^2 g(\mathbf{p}) &= u_1 D_u^3 g(\mathbf{p}) + v_1 D_v D_u^2 g(\mathbf{p}),\\
D_{\mathbf{d}_2}\, D_u^2 g(\mathbf{p}) &= u_2 D_u^3 g(\mathbf{p}) + v_2 D_v D_u^2 g(\mathbf{p}),\\
D_{\mathbf{d}_1}\, D_v^2 g(\mathbf{p}) &= u_1 D_u D_v^2 g(\mathbf{p}) + v_1 D_v^3 g(\mathbf{p}),\\
D_{\mathbf{d}_2}\, D_v^2 g(\mathbf{p}) &= u_2 D_u D_v^2 g(\mathbf{p}) + v_2 D_v^3 g(\mathbf{p}),\\
D_{\mathbf{d}_1}\, D_u D_v g(\mathbf{p}) &= u_1 D_u^2 D_v g(\mathbf{p}) + v_1 D_u D_v^2 g(\mathbf{p}),\\
D_{\mathbf{d}_2}\, D_u D_v g(\mathbf{p}) &= u_2 D_u^2 D_v g(\mathbf{p}) + v_2 D_u D_v^2 g(\mathbf{p}).
\end{aligned}$$

The second step is

$$\begin{aligned}
D_{\mathbf{d}_1}^2 D_u g(\mathbf{p}) &= u_1 D_{\mathbf{d}_1} D_u^2 g(\mathbf{p}) + v_1 D_{\mathbf{d}_1} D_u D_v g(\mathbf{p}),\\
D_{\mathbf{d}_2}^2 D_u g(\mathbf{p}) &= u_2 D_{\mathbf{d}_2} D_u^2 g(\mathbf{p}) + v_2 D_{\mathbf{d}_2} D_u D_v g(\mathbf{p}),\\
D_{\mathbf{d}_1}^2 D_v g(\mathbf{p}) &= u_1 D_{\mathbf{d}_1} D_u D_v g(\mathbf{p}) + v_1 D_{\mathbf{d}_1} D_v^2 g(\mathbf{p}),\\
D_{\mathbf{d}_2}^2 D_v g(\mathbf{p}) &= u_2 D_{\mathbf{d}_2} D_u D_v g(\mathbf{p}) + v_2 D_{\mathbf{d}_2} D_v^2 g(\mathbf{p}),\\
D_{\mathbf{d}_1} D_{\mathbf{d}_2} D_u g(\mathbf{p}) &= u_1 D_{\mathbf{d}_2} D_u^2 g(\mathbf{p}) + v_1 D_{\mathbf{d}_2} D_u D_v g(\mathbf{p}),\\
D_{\mathbf{d}_1} D_{\mathbf{d}_2} D_v g(\mathbf{p}) &= u_2 D_{\mathbf{d}_1} D_u D_v g(\mathbf{p}) + v_2 D_{\mathbf{d}_1} D_v^2 g(\mathbf{p}),
\end{aligned}$$

and the third step gives us the third order derivatives in the two directions, and the two cross derivatives, as follows:

$$\begin{aligned}
D_{\mathbf{d}_1}^3\, g(\mathbf{p}) &= u_1 D_{\mathbf{d}_1}^2 D_u g(\mathbf{p}) + v_1 D_{\mathbf{d}_1}^2 D_v g(\mathbf{p}),\\
D_{\mathbf{d}_2}^3\, g(\mathbf{p}) &= u_2 D_{\mathbf{d}_2}^2 D_u g(\mathbf{p}) + v_2 D_{\mathbf{d}_2}^2 D_v g(\mathbf{p}),\\
D_{\mathbf{d}_1} D_{\mathbf{d}_2}^2\, g(\mathbf{p}) &= u_1 D_{\mathbf{d}_2}^2 D_u g(\mathbf{p}) + v_1 D_{\mathbf{d}_2}^2 D_v g(\mathbf{p}),\\
D_{\mathbf{d}_1}^2 D_{\mathbf{d}_2}\, g(\mathbf{p}) &= u_2 D_{\mathbf{d}_1}^2 D_u g(\mathbf{p}) + v_2 D_{\mathbf{d}_1}^2 D_v g(\mathbf{p}).
\end{aligned}$$

Using these equations, we can interpolate parts of tensor product surfaces and other surfaces by ERBS triangles. In Figure 6.10, one ERBS triangle is made by interpolating a sphere at three points, using the position, two directional derivatives, two second order directional derivatives and a cross derivative, and two third order directional derivatives and two third order cross derivatives, at each of the points. In the figure, the ERBS triangle is slightly removed from the sphere so that we can see it better, otherwise we would not be able to see details because the ERBS triangle is "locally equal" to the sphere in the corners, and else also quite close. We can also see the three control polygons for the local Bézier triangles as green lines, and the control points as red cubes. The three interpolation points are all in the parameter plane of the sphere, and the directional derivatives are decided by straight lines in the parameter plane, between these points. The parametrization of the sphere affects the shape of the triangle. It can clearly be seen that two of the edges form a "slight S". One other comment is that the center points of the three control polygons almost coincides, and the three control polygons are quite equal.

The next example is based on a torus, equation (5.25). In Figure 6.11, we can see three different views of four ERBS triangles computed by Hermite interpolation of a torus,

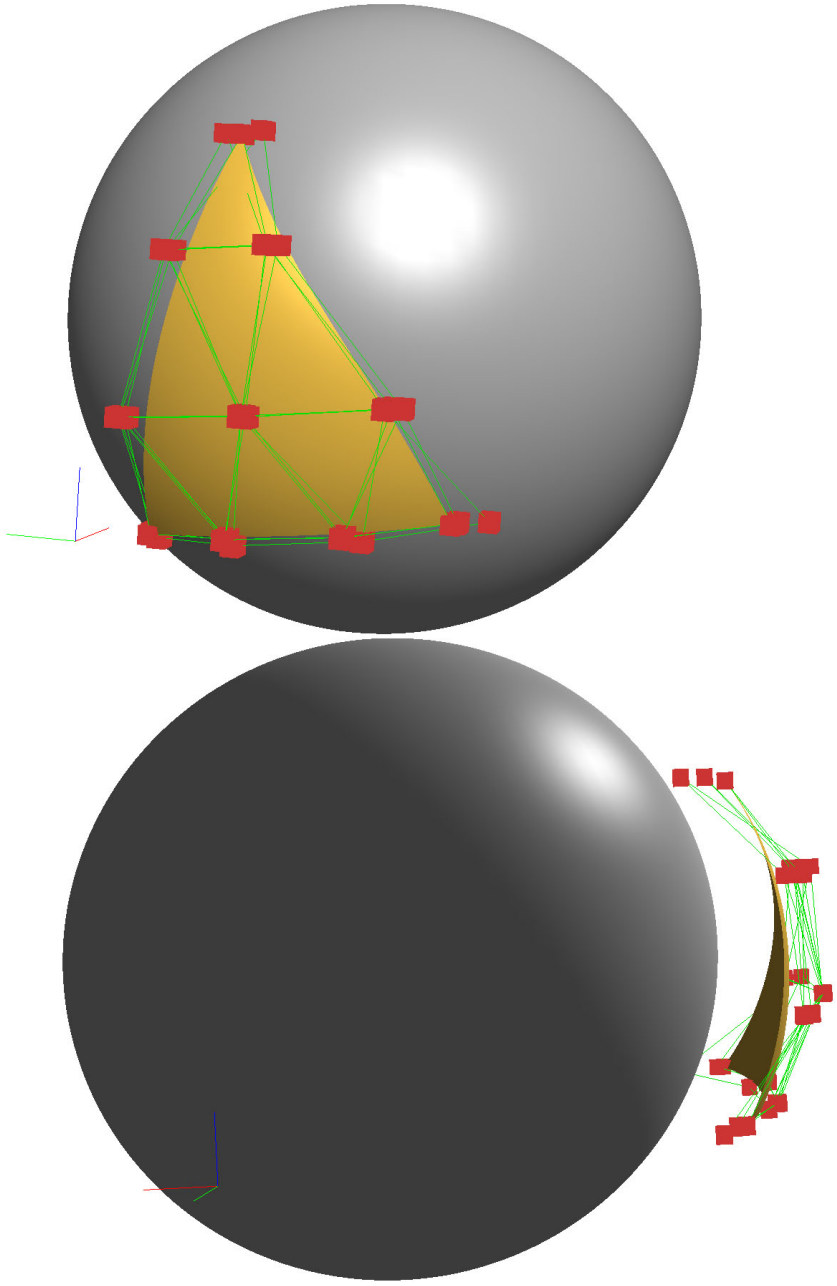

***Figure 6.10:*** *Two views of a sphere and one Expo-Rational B-spline triangular surface interpolating a part of the sphere. The ERBS triangle is slightly translated away from the sphere. The local triangles are 3rd degree Bézier triangles, and we can see all the control polygons of the local Bézier triangles as green lines, and the control points as red cubes.*

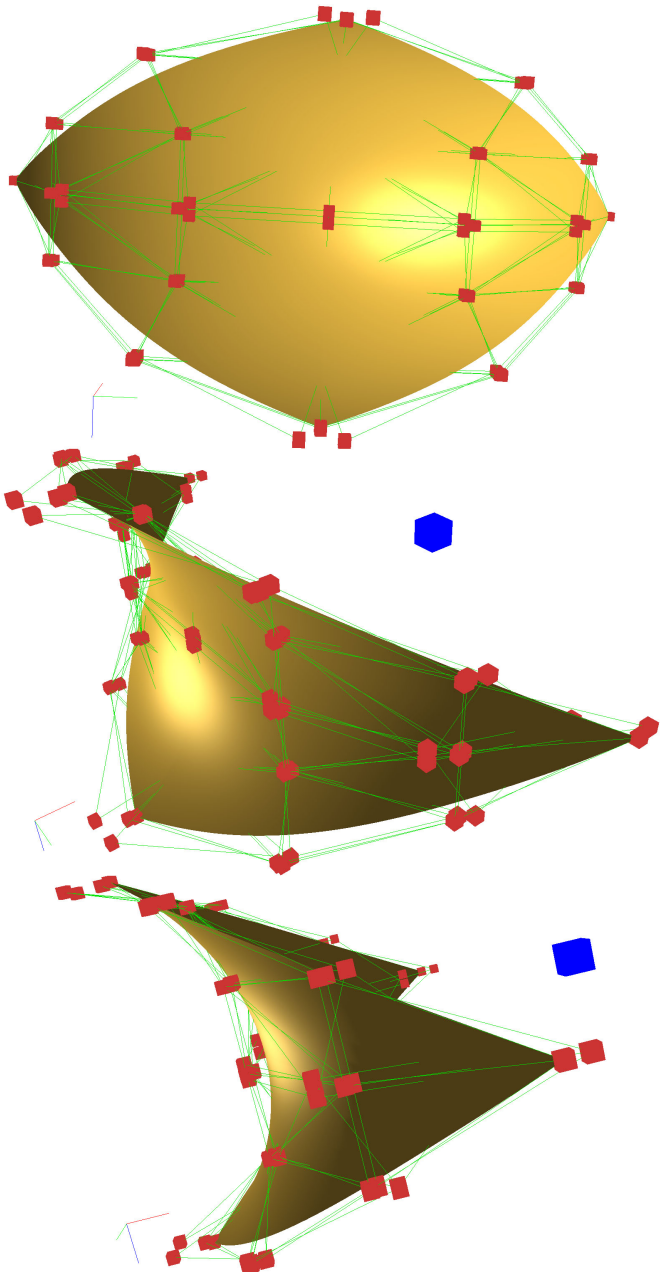

***Figure 6.11:*** *Three different views of a surface that is composed by four ERBS triangles Hermite interpolating a part of a torus, equation (5.25), at 5 points. The $4 \times 3$ local triangles are 3rd degree Bézier triangles; we can see their control polygons as green lines, and the control points as red cubes. The blue cube is the center of the original torus.*

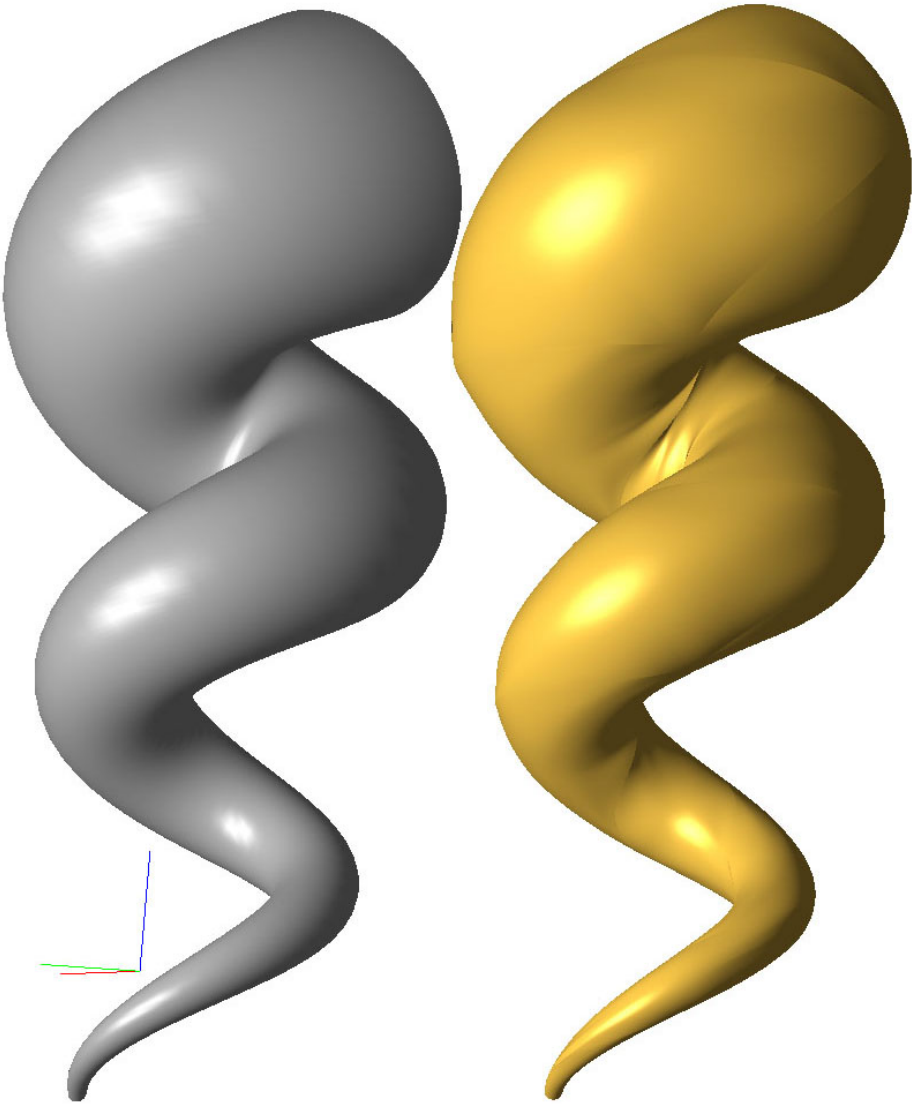

***Figure 6.12:*** *On the left hand side there is a plot of a Seashell, equation (5.26), and on the right hand side there is a plot of a set of 80 ERBS triangles, Hermite interpolating a Seashell at 44 different points. The 80 ERBS triangles are all independent of each other, but are "continuously connected"; this means that there are no holes in the composite surface after the interpolation.*

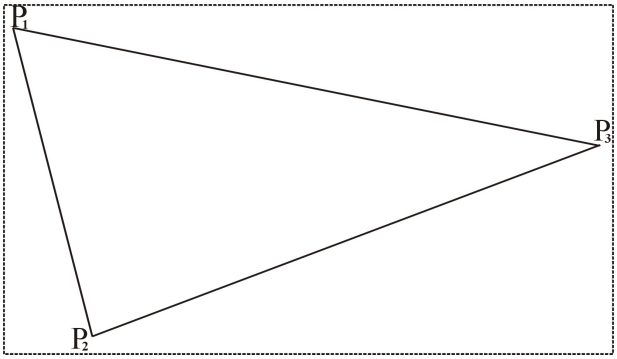


***Figure 6.13:*** *The parameter plane $\Omega$ of a surface $\overline{S}(\mathbf{u}) : \Omega \subset \mathbb{R}^2 \to \mathbb{R}^n$, $n > 0$. The three points $\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3 \in \Omega$ describe a triangle in the parameter plane.*

(5.25), at totally 5 different points. The composition of the four ERBS triangles is clearly continuous. Although this cannot be easely seen, the composition is actually based on four different triangles. But even if they together look as being $G^\infty$ (geometrically infinitely smooth), they are not. This can clearly be seen in the next example, which is a "Hermite" interpolation of a "Sea Shell" surface, equation (5.26). On the left hand side of Figure 6.12 there is a plot of a "Sea Shell" surface, (5.26), and on the right hand side there is a plot of 80 ERBS triangles interpolating the whole of the "Sea Shell" surface. One can clearly see that the composition is continuous, but it does not seem to be $G^\infty$. It is actually $G^\infty$ at all 44 interpolation points, but at the 124 edges it seems to be only continuous, $G^0$, although the result is quite good (more comments on the continuity can be found in section 6.8).

## 6.6 Sub-triangles from general parametrized surfaces as local triangles

It is possible to extract a triangular surface $S(u,v,w)$ from a general parametrized surface $\overline{S} : \Omega \subset \mathbb{R}^2 \to \mathbb{R}^n$, where n usually is 3, but can actually be any positive integer. $S(u,v,w)$ can, thus, be defined by the three points $\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3 \in \Omega$ (in the parameter plane of the general parametric surface $\overline{S}$, see Figure 6.13).

To clarify the notation, we have the surface,

$$\overline{S}(\mathbf{u}) = \overline{S}(u_1, u_2) \quad \in \mathbb{R}^n,$$

and the differential,

$$d\overline{S}_{\mathbf{u}} = \begin{bmatrix} D_{u_1}\overline{S}(\mathbf{u}) & D_{u_2}\overline{S}(\mathbf{u}) \end{bmatrix} \quad \in \mathbb{R}^{n\times 2}.$$

A triangular surface is then defined by,

$$S(u,v,w) = \overline{S}(u\mathbf{p}_1 + v\mathbf{p}_2 + w\mathbf{p}_3).$$

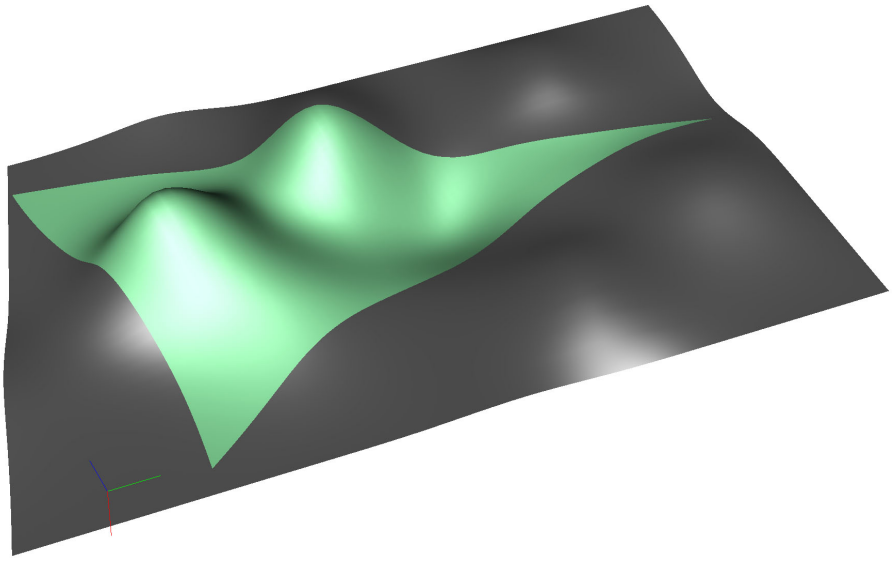

***Figure 6.14:*** *The grey surface is a B-spline tensor product surface. The triangular green surface is defined by three points in the parameter plane of the B-spline tensor product surface, and its domain is the minimum convex set including these three points (that is, a triangle in the parametric domain).*

If we denote

$$\widetilde{\mathbf{u}} = u\mathbf{p}_1 + v\mathbf{p}_2 + w\mathbf{p}_3,$$

where the three points are

$$\mathbf{p}_i = \begin{pmatrix} p_{i1} \\ p_{i2} \end{pmatrix} \quad \text{for} \quad i = 1,2,3.$$

We then get the first order partial derivatives,

$$\begin{aligned}
D_u S(u,v,w) &= d\overline{S}_{\widetilde{\mathbf{u}}}\,(\mathbf{p}_1) = D_{u_1}\overline{S}(\widetilde{\mathbf{u}})\; p_{11} + D_{u_2}\overline{S}(\widetilde{\mathbf{u}})\; p_{12}, \\
D_v S(u,v,w) &= d\overline{S}_{\widetilde{\mathbf{u}}}\,(\mathbf{p}_2) = D_{u_1}\overline{S}(\widetilde{\mathbf{u}})\; p_{21} + D_{u_2}\overline{S}(\widetilde{\mathbf{u}})\; p_{22}, \\
D_w S(u,v,w) &= d\overline{S}_{\widetilde{\mathbf{u}}}\,(\mathbf{p}_3) = D_{u_1}\overline{S}(\widetilde{\mathbf{u}})\; p_{31} + D_{u_2}\overline{S}(\widetilde{\mathbf{u}})\; p_{32},
\end{aligned}$$

and the second order partial derivatives,

$$\begin{aligned}
D_u^2 S(u,v,w) &= D_{u_1}^2\overline{S}(\widetilde{\mathbf{u}})\; p_{11}^2 + 2D_{u_1}D_{u_2}\overline{S}(\widetilde{\mathbf{u}})\; p_{11}p_{12} + D_{u_2}^2\overline{S}(\widetilde{\mathbf{u}})\; p_{12}^2, \\
D_v^2 S(u,v,w) &= D_{u_1}^2\overline{S}(\widetilde{\mathbf{u}})\; p_{21}^2 + 2D_{u_1}D_{u_2}\overline{S}(\widetilde{\mathbf{u}})\; p_{21}p_{22} + D_{u_2}^2\overline{S}(\widetilde{\mathbf{u}})\; p_{22}^2, \\
D_w^2 S(u,v,w) &= D_{u_1}^2\overline{S}(\widetilde{\mathbf{u}})\; p_{31}^2 + 2D_{u_1}D_{u_2}\overline{S}(\widetilde{\mathbf{u}})\; p_{31}p_{32} + D_{u_2}^2\overline{S}(\widetilde{\mathbf{u}})\; p_{32}^2, \\
D_u D_v S(u,v,w) &= D_{u_1}^2\overline{S}(\widetilde{\mathbf{u}})\; p_{11}p_{21} + D_{u_1}D_{u_2}\overline{S}(\widetilde{\mathbf{u}})(p_{11}p_{22} + p_{12}p_{21}) + D_{u_2}^2\overline{S}(\widetilde{\mathbf{u}})\; p_{12}p_{22}, \\
D_u D_w S(u,v,w) &= D_{u_1}^2\overline{S}(\widetilde{\mathbf{u}})\; p_{11}p_{31} + D_{u_1}D_{u_2}\overline{S}(\widetilde{\mathbf{u}})(p_{11}p_{32} + p_{12}p_{31}) + D_{u_2}^2\overline{S}(\widetilde{\mathbf{u}})\; p_{12}p_{32}, \\
D_v D_w S(u,v,w) &= D_{u_1}^2\overline{S}(\widetilde{\mathbf{u}})\; p_{21}p_{31} + D_{u_1}D_{u_2}\overline{S}(\widetilde{\mathbf{u}})(p_{21}p_{32} + p_{22}p_{31}) + D_{u_2}^2\overline{S}(\widetilde{\mathbf{u}})\; p_{22}p_{32}.
\end{aligned}$$

Notice that when using the two last sets of formulas, all formulas in section 6.4, the formulas (6.12) to (6.17), are valid.

In Figure 6.14 we can see a triangular surface extracted from a B-spline tensor product surface. The domain of the triangular surface is the triangle shown in Figure 6.13.

**Remark 18.** *Note that doing Hermite interpolation using sub-triangles in general parametrized surfaces as local triangles is just like doing Hermite interpolation using a general parametrized surface, for example, a tensor product Bézier surface, a torus, etc., and then calculating the three points in the local parameter plane.*

## 6.7 Surface approximation by triangulations.

In this section there is a brief description of an approximation of a surface by a set of ERBS triangles that are constructed to be continuous in the joints. This description concentrates on surfaces imbedded in $\mathbb{R}^3$.

Given is a compact continuous surface of any genus. The description of the approximation of this surface by a continuous and "editable" surface consisting of a set of ERBS triangles is, as follows:

- Given is a regular surface $G \subset \mathbb{R}^3$, where for each $p \in G$, there is a neighborhood $V$ in $\mathbb{R}^3$, and a map $\mathbf{x} : U \to V \cap G$ of an open set $U \subset \mathbb{R}^2$ onto $V \cap G \subset \mathbb{R}^3$. $\mathbf{x}$ is assumed to be a diffeomorphism. A local parametrization thus exists around any point $p \in G$.
- Assume that the surface $G$ is triangulated (see the beginning of this chapter, page 141, besides discussing triangulations is not among the purposes of this work). The triangulation introduces vertices, edges and triangles, and it is restricted by the requirement that in each vertex, there is a local map $\mathbf{x_i} : U_i \subset \mathbb{R}^2 \to \mathbb{R}^3$, where all the neighboring vertices also are in $U$. (Neighboring vertices are all vertices sharing an edge with the current vertex. Also, without loss of generality we assume that here $U$ can be taken to be a rectangle.)
- Compute the coordinates in $U_i$ for the vertex $\mathbf{p}_i$ and its neighboring vertices.
- For each vertex $\mathbf{p}_i$, make a tensor product Bézier surface $G_i$ by Hermite interpolations in the current vertex using the respective local map $\mathbf{x}_i$.
- For each vertex $\mathbf{p}_i$, organize all local triangles obtained in $G_i$ according to the coordinates (in the domain of $G_i$, that is $U_i$) for the vertex $\mathbf{p}_i$ and the neighboring vertices (see section 6.6, and Figure 6.15). Recall that by definition of $U_i$, all neighboring vertices of $\mathbf{p}_i$ are contained in $U_i$.
- For each triangle in the triangulation, make an ERBS triangle $S(u,v,w)$ by using, as local triangle in each of its vertices $\mathbf{p}_i$, the sub-triangles at $G_i$ that is "covering" the current triangle (the definition of a sub-triangle on a surface is given in Section 6.6).

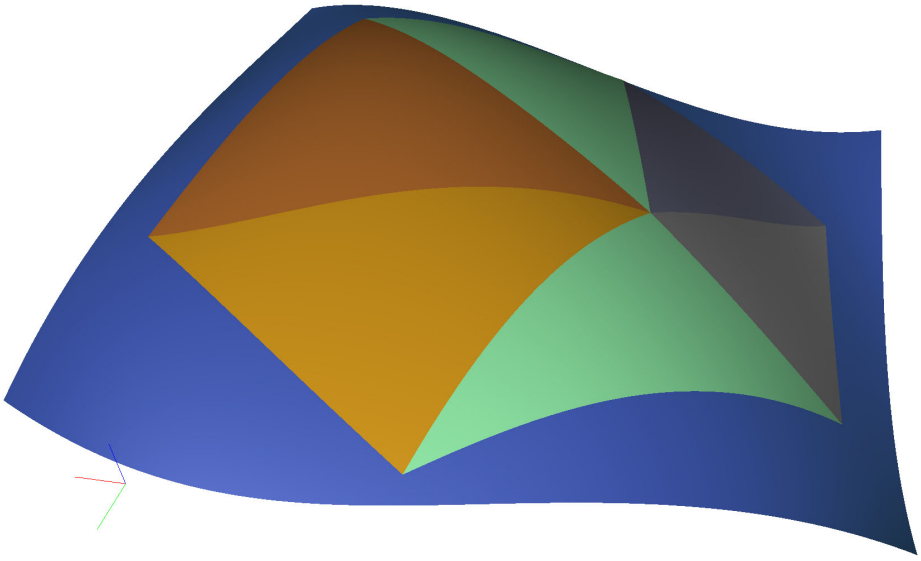

***Figure 6.15:*** *The blue surface is a Bézier tensor product surface. The different colored triangular surfaces are defined by a point (vertex) and 6 neighboring points (vertices) in the parameter plane of the Bézier tensor product surface. All triangular surfaces are, therefore, just sub-surfaces of the Bézier tensor product patch.*

The following example illustrates this procedure.

- We use a sphere with radius 1 as the surface $G$. This will clearly fulfill the first point in the previous description.
- We consider a triangulation of the sphere using 6 vertices, one vertex at each of the poles, and four vertices equally distributed along the equator line. This gives 8 equilateral triangles.
- We consider a local map at each vertex $\mathbf{p}_i$

$$\mathbf{x}_i(u,v) = \begin{pmatrix} \cos u \cos v \\ \sin u \cos v \\ \sin v \end{pmatrix}, \tag{6.26}$$

  such that the current vertex is in the local origin, i.e., $\mathbf{x}_i(0,0) = \mathbf{p}_i$.
- Now, we make a tensor product Bézier patch by Hermite interpolation as it is described in section 5.3.1. In addition, to evaluate in the vertices, using the local map, to get position and the partial derivatives, we need some more information about the position and mappings. We stated that the current vertex should be in the center of the Bézier patch. This implies that $\omega_u = \omega_v = \frac{1}{2}$ ($\omega_u$ and $\omega_v$ are described in section 5.3.1). To ensure that the four neighboring vertices are inside the domain of the Bézier patch we have to set the affine global/local mapping to $\pi$ (in the local

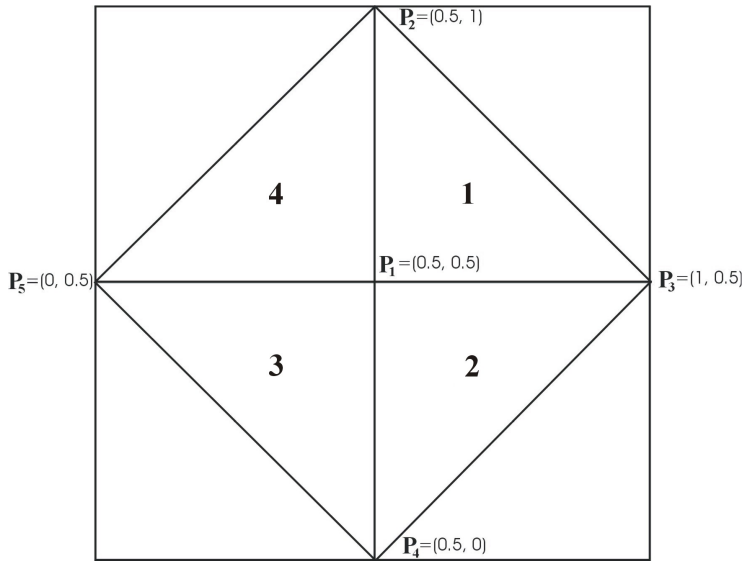


***Figure 6.16:*** *The parameter plane of a Bézier tensor product patch connected to one of the vertices in the triangulation of a sphere. The coordinates of the current vertex* $\mathbf{p}_1$*, and the four neighboring vertices* $\mathbf{p}_2, \mathbf{p}_3, \mathbf{p}_4, \mathbf{p}_5$ *are all marked in the figure. This parameter plane looks in the same way for each of the vertices.*

map the distance from a pole to equator is $\frac{\pi}{2}$). This implies that $\delta_u = \delta_v = \pi$ ($\delta_u$ and $\delta_v$ is described in section 5.3.1).

- The construction of the local triangles (sub-triangles of the Bézier patch) in the ERBS triangle is straightforward. The coordinates in the parameter plane of the Bézier patch can be seen in Figure 6.16. The "current vertex" in the center has the coordinates $\mathbf{p}_1 = (0.5, 0.5)$. The neighboring vertices are ordered clockwise, and their coordinates are $\mathbf{p}_2 = (0.5, 1)$, $\mathbf{p}_3 = (1, 0.5)$, $\mathbf{p}_4 = (0.5, 0)$ and $\mathbf{p}_5 = (0, 0.5)$. The result is the four triangles we can see in Figure 6.16, the first (numbered as 1 in the figure) is defined by the three points $\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3$, the second (numbered as 2 in the figure) is defined by the three points $\mathbf{p}_1, \mathbf{p}_3, \mathbf{p}_4$, and the third (numbered as 3 in the figure) is defined by the three points $\mathbf{p}_1, \mathbf{p}_4, \mathbf{p}_5$, and the last triangle (numbered as 4 in the figure) is defined by the three points $\mathbf{p}_1, \mathbf{p}_5, \mathbf{p}_2$.

- The construction of the 8 ERBS triangles is a question of putting everything into order. In Figure 6.16 the local sub-triangles are numbered from 1 to 4. Take first this patch to be the patch at the north pole. If we then rotate a copy of this patch 90° around a horizontal axis going through the sphere center and, from our view in Figure 6.16), passing through the vertices $\mathbf{p}_5$ and $\mathbf{p}_3$, we get the patch of one of the vertices on the equator line. The other three patches on the equator can be produced by rotating a copy of the previous patch 90° around the vertical axis going through the sphere center and the poles. The last patch is the result of rotating a copy of

the first patch 180° around the horizontal axis going through the sphere center (like the first rotation). If we use Figure 6.16 to keep track of positions assuming we are above the north pole, then up means at vertex $\mathbf{p}_2$ in the figure, left means at vertex $\mathbf{p}_5$ and so on. We can start ordering. We number the patches with no. 1 at the north pole, no. 2 the "upper" equator, no. 3 the "left" equator, no. 4 the "lower" equator, no. 5 at the "right" equator, and nr. 6 at the south pole. We also denote and number the local sub-triangles $t_{ij}$, where $i$ is the patch number and $j$ is the triangle number in the patch. We now get 8 ERBS triangles consisting of the following local sub-triangles:

1st ERBS triangle with the local triangles $t_{11}$, $t_{22}$, $t_{53}$,

2nd ERBS triangle with the local triangles $t_{12}$, $t_{52}$, $t_{43}$,

3rd ERBS triangle with the local triangles $t_{13}$, $t_{42}$, $t_{33}$,

4th ERBS triangle with the local triangles $t_{14}$, $t_{32}$, $t_{23}$,

5th ERBS triangle with the local triangles $t_{61}$, $t_{44}$, $t_{51}$,

6th ERBS triangle with the local triangles $t_{62}$, $t_{54}$, $t_{21}$,

7th ERBS triangle with the local triangles $t_{63}$, $t_{24}$, $t_{31}$,

8th ERBS triangle with the local triangles $t_{64}$, $t_{34}$, $t_{41}$.

In Figure 6.17 we can see parts of some of the Bézier patches and the local sub-triangles from the sphere example. In Figure 6.18 we can see both the original sphere (in brass on the right hand side), and the approximation (in silver on the left hand side). Remember that the approximation is only done with 8 triangles. The result is quite good, and we can see that it is geometrically smooth at the vertices, but it is clearly not smooth over the edges although it is continuous (more comments on the continuity and smoothness will be given in section 6.8).

A general algorithm for surface approximation by triangulation using ERBS consists of three main parts:

1. The algorithm for triangulation is separated for surfaces of different genus and for simply connected surfaces (genus 0), and must generate a local map around each vertex containing all of its neighboring vertices. The first part of the algorithm concerns the generation of the vertices. The second part of the algorithm concerns appropriate version of the Delaunay (or other) triangulations (see [12]). The triangulation algorithm itself can, of course, be tricky, but in itself it is not a part of the ERBS algorithm.

2. The local mapping must be automated, together with finding the coordinates for the neighboring vertices.

3. The local sub-triangles for use in the generation of the ERBS triangles must be automatically ordered, so that the right sub triangle is associated to the right triangle, and this sub triangle must be correctly orientated.

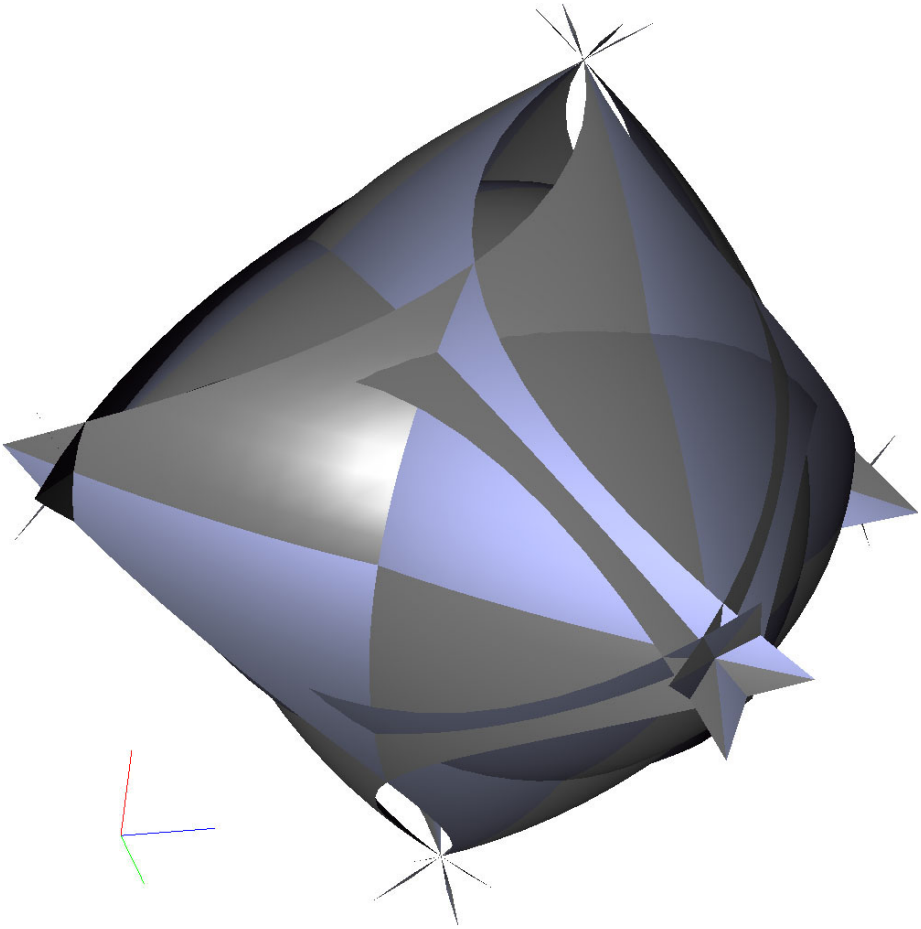

***Figure 6.17:*** *All 24 local triangles in the sphere example described in section 6.6 are displayed in the figure. We can clearly see some of them, and we can also see trimmed parts of the 6 Bézier patches connected to each of the 6 vertices. The center of each Bézier patch actually interpolates the original sphere in each of the vertices.*

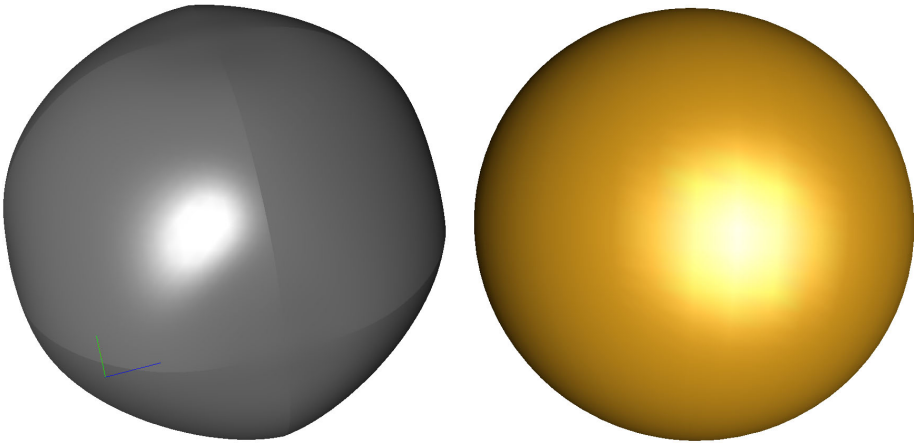

***Figure 6.18:*** *The brass sphere on the right hand side is the original sphere. The silver sphere on the left hand side is an ERBS approximation by triangulation. The approximation is done with only 8 triangles, and all the triangles are "equilateral", symmetric according to all three vertices in each triangle.*

One important feasible feature in surface approximation by triangulation using ERBS is the object editation feature. This feature is used in figures 6.19, 6.20, 6.21 and 6.22. In Figure 6.19 the Bézier patch connected to the vertex at the north pole is translated $1\frac{1}{2}$ to the north. The result is an object shaped like an egg. In Figure 6.20 the Bézier patch connected to the vertex at the north pole is translated $\frac{1}{2}$ to the south. The result is an object shaped more like a hazelnut. The third example of using the editing feature is shown in Figure 6.21. In this example all the Bézier patches (connected to all of the vertices) are translated away from the center of the sphere. The result is an eight sided cube where all the edges and corners are rounded off. All these examples only use translation of the local Bézier patches.

In the next examples, rotation is also used. In Figure 6.22 there are four objects in the same figure, and they are all a result of rotation of local Bézier patches. The object in the upper left hand corner was at first the same object as in Figure 6.20, but the Bézier patch at the top has been rotated by an additional 70° clockwise around a vertical axis. The object in the upper right hand corner was also at first the same as in Figure 6.20, but the Bézier patch at the top has been rotated by an additional 70° clockwise around a horizontal axis. The object in the lower left hand corner was also at first the same object as in Figure 6.20, but the Bézier patch on one of the sides (the one pointed towards us) has been rotated by an additional 70° clockwise around a vertical axis. The object in the lower right hand corner was also at first the same as in Figure 6.20, but two opposite Bézier patches at the side have been rotated by an additional 70° clockwise around a horizontal axis. These two vertices, which are connected to the patches that have been rotated, are the ones we see on either side of the object.

In the last figure, 6.23, there is one object plotted from four different angles. All local

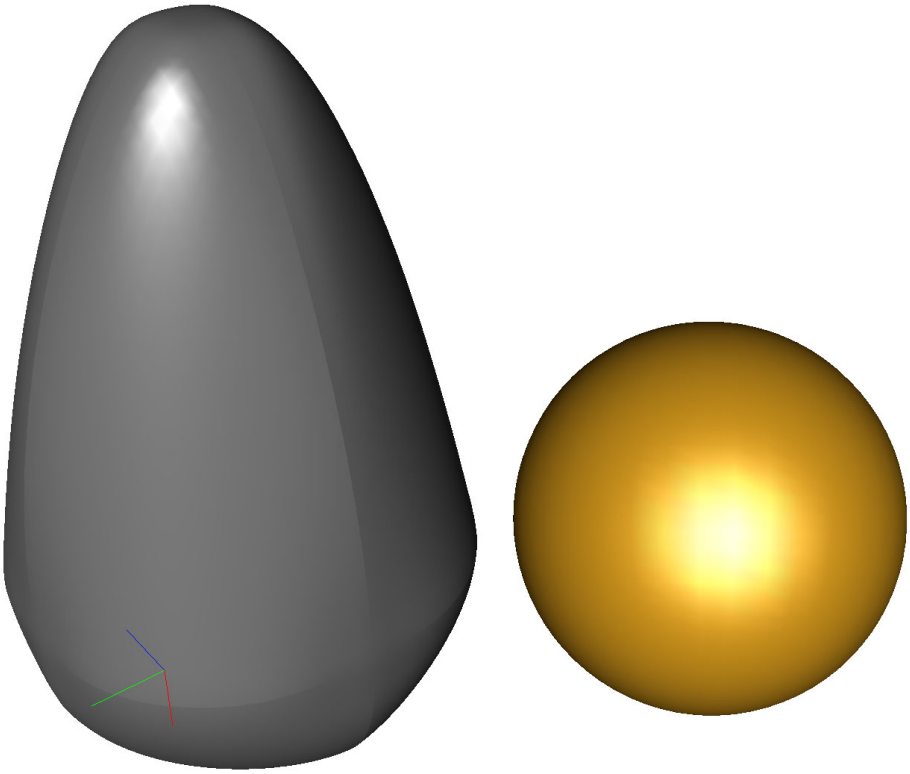

***Figure 6.19:*** *The brass sphere on the right hand side is the original sphere. The silver object on the left hand side is the ERBS approximation by triangulation, after the Bézier patch associated to the vertex at the north pole has been translated* $1\frac{1}{2}$ *to the north. The total length of the object (from south to north) is* $3\frac{1}{2}$ *(the diameter of the sphere is* 2*).*

Bézier patches in this object, except for the one at the south pole, are rotated 70° clockwise around a horizontal axis. The horizontal axis actually has the same direction as the one at the upper and right hand side in Figure 6.22. The red marks in the figure mark the position of four of the six vertices in the object. One can clearly see the smoothness in the vertices, $G^\infty$, which follows from using local triangles (which are sub-triangles on a common local smooth surface at each of the vertices) and from property 4 in Theorem 6.1. Notice also that the "edge" of the "bill" is not an edge between the ERBS triangles, but it is actually smooth. This is because the "edge" of the "bill" is in the interior of an ERBS triangle. This can clearly be seen on the two lower plots in Figure 6.23, where we also clearly can see where the real edges are located.

These examples only give a small insight into the possibilities of surface approximation by triangulation using ERBS. The problem is, of course, the continuity, which will be briefly discussed further in the next section.

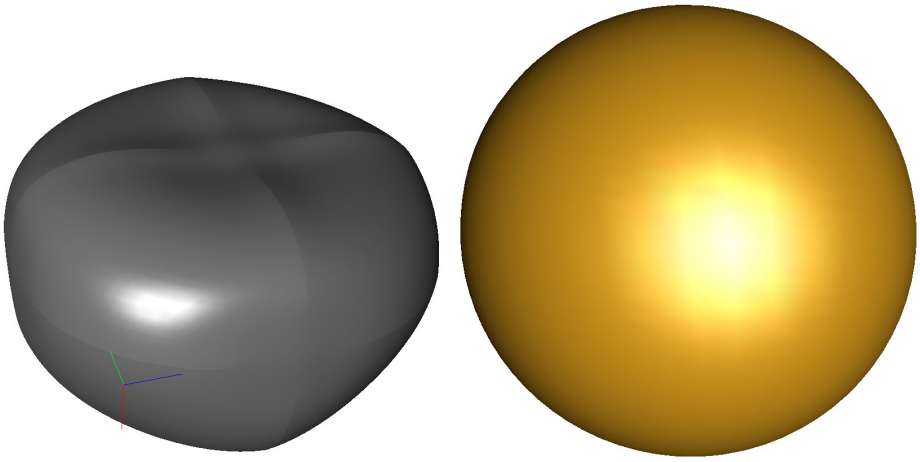

***Figure 6.20:*** *The brass sphere on the right hand side is the original unit sphere. The silver object on the left hand side is the ERBS approximation by triangulation, after the Bézier patch associated to the vertex at the north pole has been translated* $\frac{1}{2}$ *to the south. The total length of the object (from south to north) is* $1\frac{1}{2}$ *(the diameter of the sphere is* 2*).*

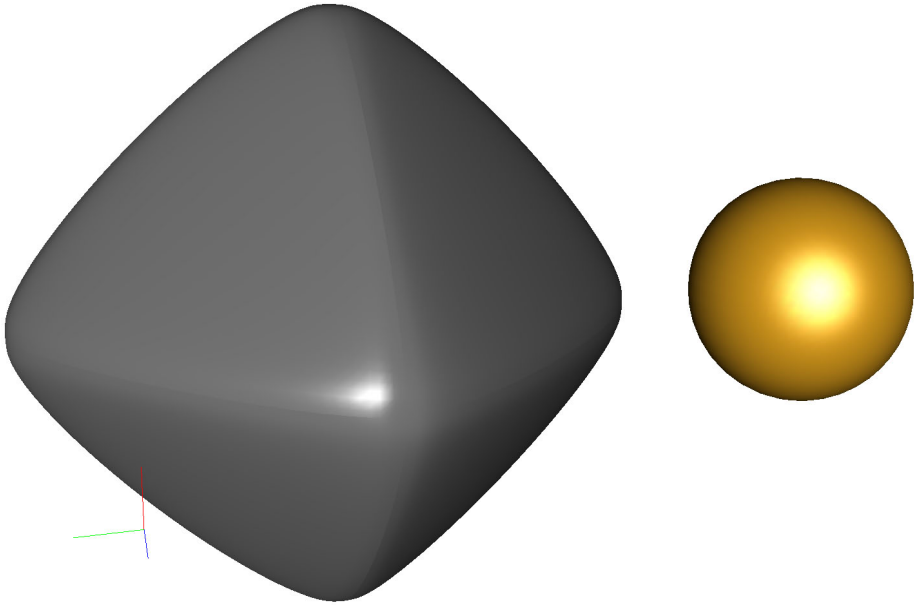

***Figure 6.21:*** *The brass sphere on the right hand side is the original unit sphere. The silver object on the left hand side is the ERBS approximation by triangulation, after the Bézier patch associated to each vertex has been translated* $1\frac{1}{2}$ *out from the center of the sphere. The total length of the object, from one vertex to the opposite vertex is* 5 *(the diameter of the sphere is* 2*).*

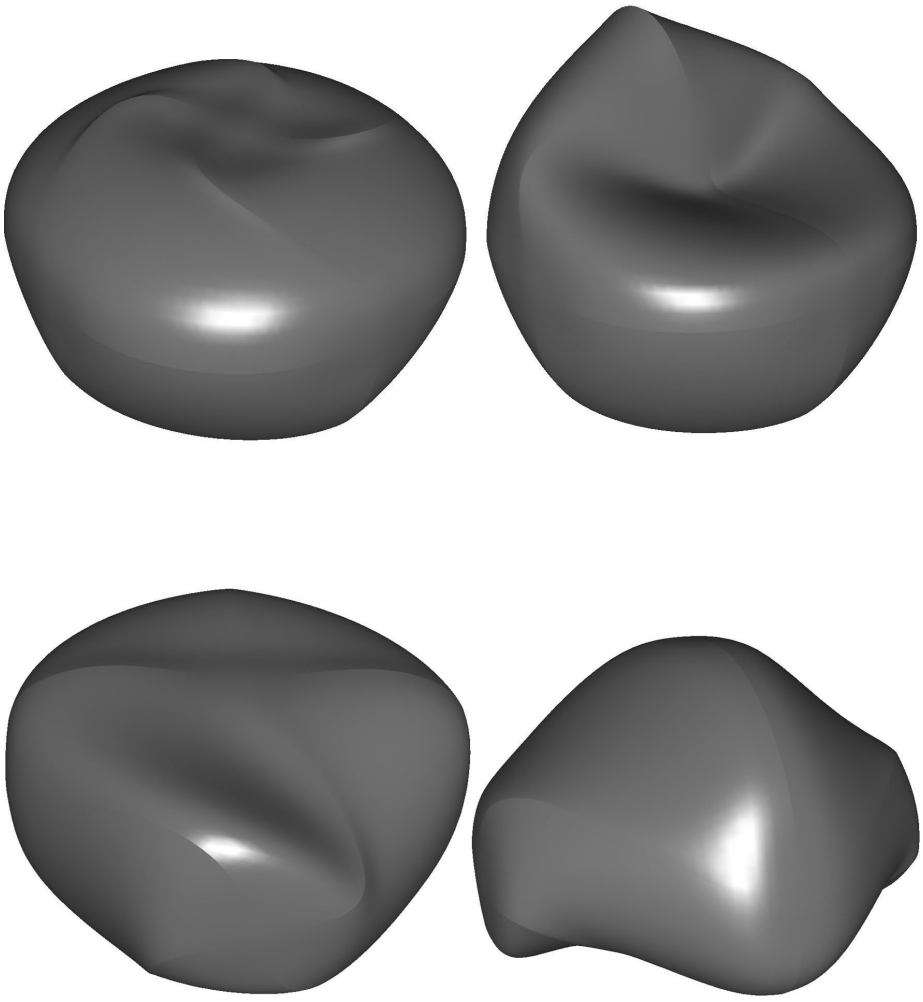

***Figure 6.22:*** *In this figure there are four objects obtained as a result of rotating local Bézier patches. All objects were initially the same object as the "silver object" in Figure 6.20. The object in the upper left hand corner has been further edited in a way that the Bézier patch at the top has been rotated an additional* $70^\circ$ *clockwise around a vertical axis going through the center of the original sphere approximation. The object in the upper right hand corner has been further edited in a way that the Bézier patch at the top has been rotated an additional* $70^\circ$ *clockwise around a horizontal axis. The object in the lower left hand corner has been further edited in a way that the Bézier patch on one of the sides has been rotated an additional* $70^\circ$ *clockwise around a vertical axis. The object in the lower right hand corner has been further edited in a way that two opposite Bézier patches on the sides have been rotated an additional* $70^\circ$ *clockwise around a horizontal axis.*

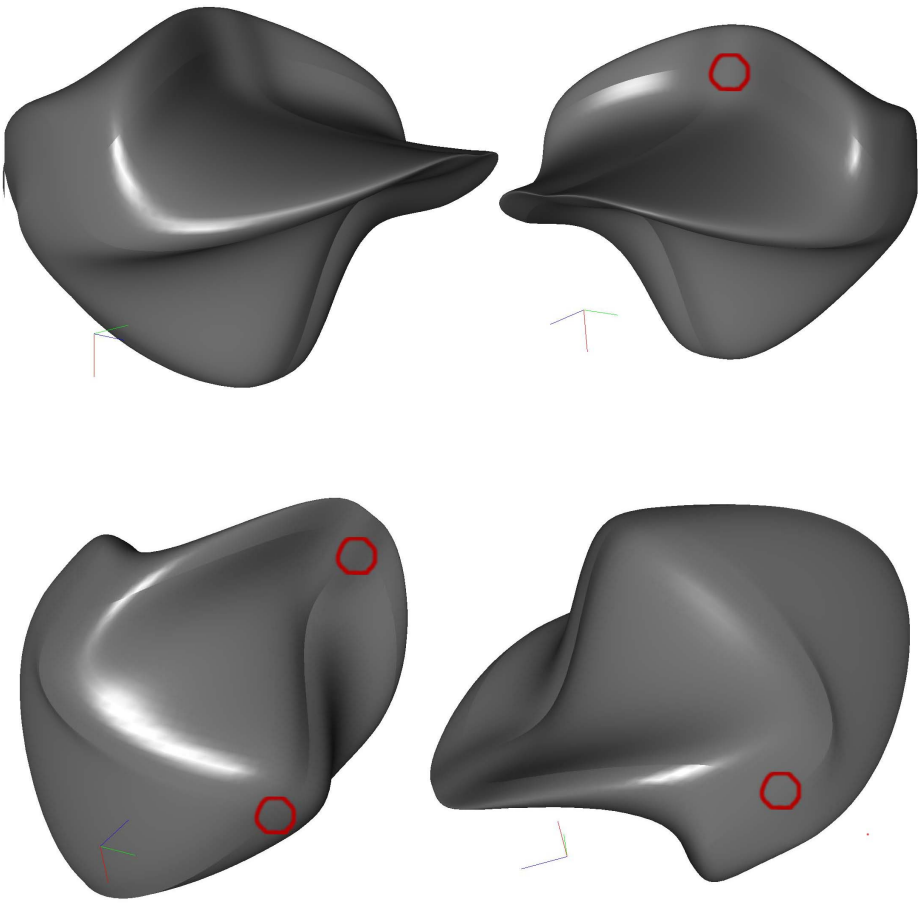

***Figure 6.23:*** *The four objects in this figure are actually four plots of the same object, only seen from a different angle. In this object all local Bézier patches, except for the one at the south pole, are rotated* $70^\circ$ *clockwise around the same horizontal axis. The horizontal axis has the same direction as the one in the upper right hand corner in Figure 6.22. The red marks on the figure mark the position of four of the six vertices in the figure. One can clearly see the smoothness in the vertices.*

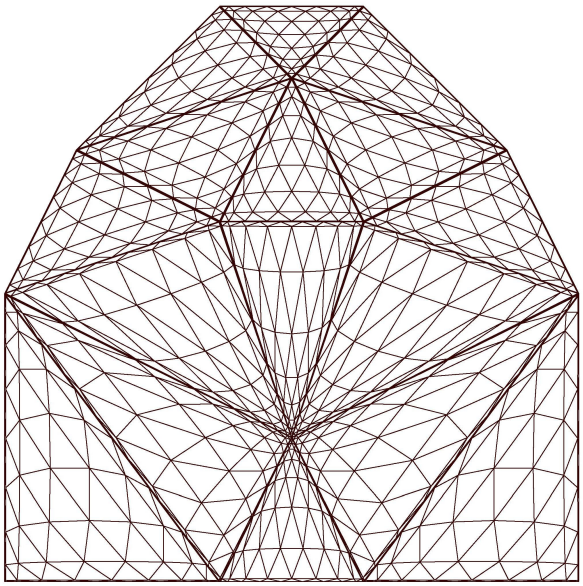

***Figure 6.24:*** *The figure shows a set of ERBS triangles where each is plotted with constant parameter lines: $line(v,w)$, where u is a constant, $line(w,u)$, where v is a constant and $line(u,v)$, where w is a constant (recall that the parameters are homogeneous barycentric coordinates also for the curves).*

## 6.8 Epilogue

ERBS have the potential to make most impact as a new approach in the case of surfaces on triangulations. It is clear from the properties defined in Theorem 6.1 that the continuity at the vertices of an ERBS triangulated surface is $G^\infty$ (actually also $C^\infty$). The continuity at the edges are clearly only continuous, although the parametrization seems to be smooth (see article [16]). One observation which was already made in the case of curves (see Remark 9) is that the Lagrange interpolation of curves (and surfaces) using ERBS, leads to the piecewise-linear ($G^0$) curve obtained when using linear B-splines, but with a $C^\infty$ parametrization on it. Recall that this is possible because this parametrization is not regular at the knots. When the constant values in the Lagrange interpolation are replaced by polynomial curves, we can see that they are already $G^k$ where k was the degree of the local polynomial curves used. Similar is the situation in the case of surfaces on triangulations. On Figure 6.24 is given an example of a Lagrange interpolant over a triangulation (the coefficients are numbers not local functions). The plot is that of the Lagrange interpolant by linear B-splines over the triangulation, but it can be shown that the parametrization induced by the use of ERBS leads to possible $C^\infty$ smoothness also on the edges of the triangles (where linear B-splines are only continuous but not smooth). The reason is again the absence of regularity of the surface on the edges. (although in the vertices the surface is also G-smooth). This can be judged from the fact that the constant parameter lines:

$line(v,w)$, where $u$ is a constant, $line(w,u)$, where $v$ is a constant and $line(u,v)$, where $w$ is a constant (recall that the parameters are homogeneous barycentric coordinates also for the curves) are actually tangential to the respective edge which they are supposed to intersect. So this intersection happens in a smooth way, with the visual effect that each lines smoothly continuous into the neighboring triangle. This behavior is due to Property **P5** in Theorem 6.1. The continuity will, however, not be discussed further here, but is an important topic for further investigations, on which we already have started.

For practical use in CAGD it will be necessary to develop a complete/controllable geometrically smooth version of the surface approximation by triangulation using ERBS. This might, however, not be very difficult because the critical part, the continuity over the vertices, is already a property in the approximation. This is, however, also a topic for further investigation although some tests have already been done. One alternative implementation is to introduce smoothness as a property for the edges, so one can choose if an edge shall be smooth or not.

# Chapter 7

# Summary and some other topics

Expo-Rational B-splines are slightly more computationally expensive than the present industrial standard NURBS/B-splines. But they have a lot of advantages compared to their counterparts. The following can be mentioned.

- They have a much better ability for modeling complex geometry.
- The Hermite interpolation property is much simpler to use and better in quality than what we find for other types of splines.
- The editing possibility is much better than for its competitors, due to the affine transformations of the local functions.
- The dynamic "simulation" possibilities are also much better, also due to the affine transformations of the local functions.
- The future perspectives are much greater, especially in the case of surface interpolation and approximation over a triangulation.

The previous chapters have shown us that this list is not an exaggeration. Especially all the examples in the figures show us that there are really a lot of possibilities.

Expo-Rational B-splines are just in the initial phase with regards to both development and practical use. So far, the most work has been done at Narvik University College (NUC), but there is hope that this will now change. Together with the people at the Simulations R&D group, who have designed and developed ERBS, a number of students have also been working with ERBS in their Master theses. Several students have used ERBS as a partial topic in their theses. The students who have been mainly studying ERBS in their theses are:

- Andreas Bredesen: Parameter study of ERBS curves with regard to data reduction and approximation/interpolation problems [6].
- Yan Wang: Parameter study of ERBS surfaces with regard to data reduction and approximation/interpolation problems [55].
- Einar Rasmussen: Improved visualization of scalar data using interpolation [46].

- Bjørn Anders Ulsund: Expo-Rational B-spline evaluator on GPU [53].
- Richard Dahl: Realistic simulation using Expo-Rational B-splines and GPU programming [11].
- Jie Gao: Expo-Rational B-splines to/from NURBS [29].

To have been the supervisor of these students has, for me, been very useful, and some of the results they have produced might also be of general interest. At the end of section 7.2, "Three-variate tensor product Expo-Rational B-spline", there is a short summary of one of these theses, namely "Improved visualization of scalar data using interpolation". This work was initiated by an industrial partner, "ComputIT", located in Trondheim, Norway. (The company is a specialist in fire simulations.)

There is also another section in this chapter, namely "NUERBS Form of Expo-Rational B-splines", section 7.1. The reason for having this section in the summary chapter is that it is just a short summary of an earlier publication [17], which was a common work of Lubomir Dechevsky, Børre Bang and me. Although NUERBS has a bigger potential, it has been necessary to concentrate on more basic aspect of ERBS needed earlier in the theory and application. Of course, we intend to return on the topic about NUERBS later.

## 7.1 NUERBS Form of Expo-Rational B-splines

This section contains a description of the rational forms of Expo-Rational B-splines. We shall call these "Non-Uniform Expo-Rational B-splines" (NUERBS), because of the close analogy with NURBS. This description will only be a short summary, but you can find a more complete description in [17]. In the following we will mainly concentrate on curves. The figures used in this section are made in cooperation with Børre Bang.
The general NUERBS form is:

$$f(t) = \frac{\sum_{i=1}^{n} c_i(t) W_i B_i(t)}{\sum_{i=1}^{n} W_i B_i(t)} \tag{7.1}$$

where $c_i(t), \quad i = 1,...,n$ are local curves, scalar or vector-valued. The formula resembles a usual rational Bezier/NURBS. The effect of the weights is also similar to what we expect from rational Bezier/NURBS, except that $c_i(t)$ is not a point but a curve. In Figure 7.1 the effect of increase of one weight is displayed: the global curve approximates better the corresponding local curve.

If we replace the local curves with points, which correspond to Lagrange interpolation by $B_i$ at the simple knot $t_i$, the total curve will geometrically be a piecewise linear curve. Since the curve is infinitely smooth, all derivatives must be zero at each knot. This can be seen in Figure 7.2 because the points are very dense at the knots (dense means slow speed). The effect of changing the weight is that the dense area is moving. The dense area is getting smaller at the knot with a small weight and bigger at a knot with a bigger

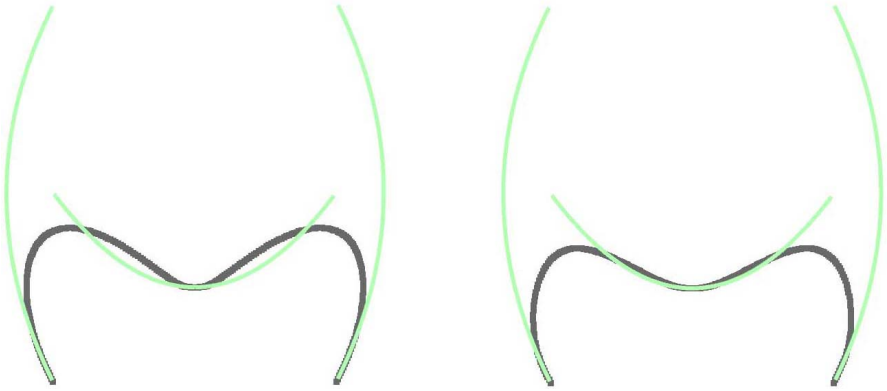

***Figure 7.1:*** *The two curves are equal except that all weights in the curve on the left hand side are 1, while the middle weight in the curve on the right hand side is 5. Observe that this causes the curve to be pulled towards the middle local curve.*

weight. In Figure 7.2 this can be seen on curves b and c. This phenomenon will be discussed in more detail in section 7.1.3.

**Remark 19.** *The general NUERBS form has variation-diminishing properties for a very broad range of classes of local curves. However, since these properties are a consequence of the variation-diminishing properties of the underlying Expo-Rational B-splines, it is more appropriate to consider variation diminishing as part of the theory of Expo-Rational B-splines (see [16] for a discussion of this topic).*

### 7.1.1 Hermite interpolation property of the general NUERBS form

Here we shall see that, under very general assumptions about the class of local curves in (7.1), the general NUERBS form, just like ERBS, has an Hermite interpolation property (regarding ERBS and Hermite interpolation property, see section 2.8 and Theorem 2.4). The important underlying fact here is property 4 in Theorem 2.1, that

$$D^j B_i(t_k) = 0, \quad i = 1,...,n, \quad k = 0,...,n+1, \quad j = 1,2,... \tag{7.2}$$

**Theorem 7.1.** *Let $c_i : [t_{i-1}, t_{i+1}] \subset \mathbb{R} \to \mathbb{R}^d$, $0 < d < \infty$, $i = 1,2,...,n$, be a sequence of $C^{m_i}$-smooth local functions on $[t_{i-1}, t_{i+1}]$, where $|c_i| \in \mathfrak{F}(B_i)$ (see definition 2.5), where $m_i \geq 0$ is an integer (in the case of $m_i = \infty$ we assume that $c_i$ is an entire analytic function on $[t_{i-1}, t_{i+1}]$). Let the weights $W_i \in (0,\infty)$, $i = 1,...,n$ be a sequence of scalars. Let*

$$f(t) = \frac{\sum_{i=1}^{n} c_i(t)\, W_i\, B_i(t)}{\sum_{i=1}^{n} W_i\, B_i(t)} \tag{7.3}$$

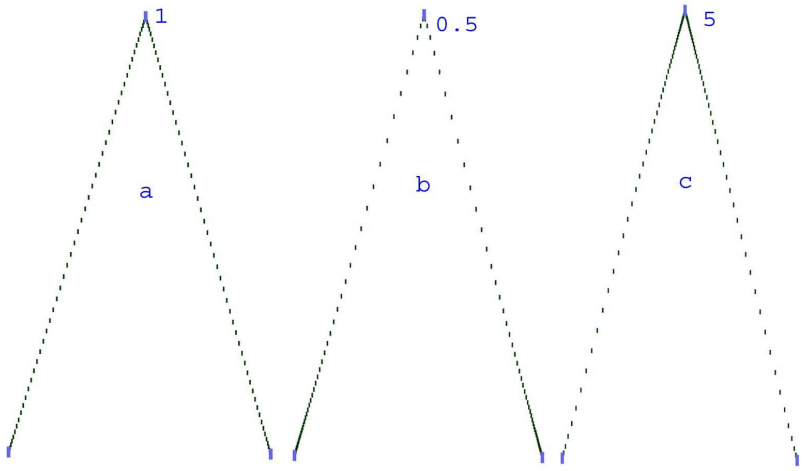


***Figure 7.2:*** *Three piecewise linear curves a, b and c. All weights for curve a are equal to 1, curve b has weight 0.5 in the middle and curve c has weight 5 in the middle. The curves are plotted with points, where the density of the points indicate the speed, and where dense means slow speed.*

*be the general NUERBS form (7.1) for* $c_i(t),\ i = 1,...,n$. *Then,*

$$f|_{(t_i,t_{i+1})} \in C^{\min(m_i,m_{i+1})}((t_i,t_{i+1}),\mathbb{R}^d), \quad i = 0,..,n \tag{7.4}$$

*and*

$$D^j\ f(t_i) = D^j\ c_i(t_i), \quad j = 0,..,n_i, \quad i = 1,..,n. \tag{7.5}$$

*Proof.* First (7.4) follows immediately from (7.3), in view of the properties of $B_i$ (Theorem 2.1) and the assumptions about $c_i$. To prove (7.5), we first write (7.3) in the form

$$(\sum_{i=1}^{n} W_k\ B_k(t))\ f(t) = \sum_{i=1}^{n} c_k(t)\ W_k\ B_k(t). \tag{7.6}$$

From property 2 and 3, Theorem 2.1, it follows that at the knot $t_i$ there is only one basis function $B_i(t)$ that is $\neq 0$. It follows that for $t = t_i$, (7.6) is reduced to

$$W_i\ f(t) = c_i(t)\ W_i.$$

Since $0 < W_i < \infty$ holds, we get (7.5) for $j = 0$. Next, we differentiate (7.6) in $t$ and apply (7.2) for $j = 1$. From here it is easy to obtain (7.5) for $j = 1$ by using the already established result (7.5) for $j = 0$. The general case of (7.5) is now obtained by induction, using (7.5) as induction hypothesis for every $j = 0,...,\nu \leq m_i - 1$, and then also using (7.2) on the induction step $j = \nu + 1$. □

We will now investigate in more detail the particular case of local polynomial-based curves.

### 7.1.2 The Hierarchy of general NUERBS forms of Bernstein-Bézier type

We shall consider polynomial-based local curves of Bernstein-Bézier type, where the general NUERBS form is:

$$f(t)=\frac{\sum_{i=1}^{n}\sum_{j=0}^{k_i}c_{ij}b_{k_i,j}(\frac{t-t_{i-1}}{t_{i+1}-t_{i-1}})W_iB_i(t)}{\sum_{i=1}^{n}W_iB_i(t)}, \tag{7.7}$$

where

$$b_{k_i,j}(x)=\left(\begin{array}{c}k_i\\ j\end{array}\right)x^j(1-x)^{k_i-j}. \tag{7.8}$$

If $W_i=1,\ i=1,2,...,n,$ this is identical to ERBS curves with local Bézier curves, as we can see in section 4.3. It is thus possible to use the Hermite interpolation as described for ERBS curves also to generate NUERBS curves.

When using polynomial-based local curves, it is possible, of course, to also use the rational form of these curves. In this case we have several alternative ways of expanding the Expo-Rational B-splines into NUERBS. We will now compare 3 different forms of NUERBS based on local rational Bézier curves (first proposed by Lubomir Dechevsky).

First we look at what we call the local NUERBS form:

$$f(t)=\frac{\sum_{i=1}^{n}\frac{\sum_{j=0}^{k_i}c_{ij}w_{ij}b_{k_i,j}(\frac{t-t_{i-1}}{t_{i+1}-t_{i-1}})}{\sum_{j=0}^{k_i}w_{ij}b_{k_i,j}(\frac{t-t_{i-1}}{t_{i+1}-t_{i-1}})}W_iB_i(t)}{\sum_{i=1}^{n}W_iB_i(t)}. \tag{7.9}$$

This form is identical with the general NUERBS form where the local curve is a rational Bezier curve.

The second one is the global NUERBS form:

$$f(t)=\frac{\sum_{i=1}^{n}\sum_{j=0}^{k_i}c_{ij}\vartheta_{ij}b_{k_i,j}(\frac{t-t_{i-1}}{t_{i+1}-t_{i-1}})W_iB_i(t)}{\sum_{i=1}^{n}\sum_{j=0}^{k_i}\vartheta_{ij}b_{k_i,j}(\frac{t-t_{i-1}}{t_{i+1}-t_{i-1}})W_iB_i(t)}. \tag{7.10}$$

The main difference between this and the local form is that we relate the denominators of the local and global parts, in other words, we consider the two-index basis $\{b_{k_i,j}B_i\}$ as a

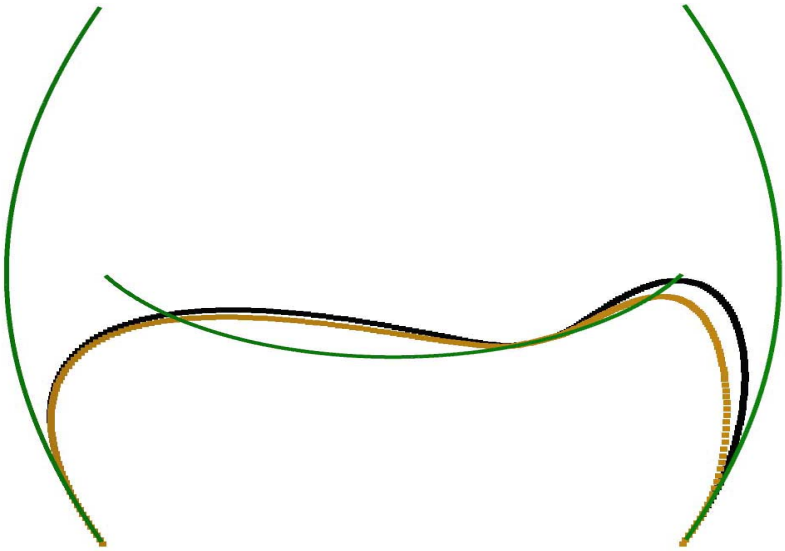

***Figure 7.3:*** *Two global curves based on the same local curves with all respective weights being the same. The black/outer one is defined via the global NUERBS form, the inner one is defined via the local NUERBS form (see section 7.1.2).*

global basis. The effect of this can be seen in Figure 7.3. In Figure 7.3 we have changed the weight on the right end point of the second curve to 4, all the other weights, both local and global, are 1. As seen, the global form is responding more to the increase in weight than the local form. If we change some of the global weights then both forms will respond equally.

Finally, we take a look at what we call the full NUERBS form:

$$f(t) = \frac{\displaystyle\sum_{i=1}^{n} \frac{\displaystyle\sum_{j=0}^{k_i} c_{ij}\mu_{ij}\vartheta_{ij}b_{k_i,j}(\frac{t-t_{i-1}}{t_{i+1}-t_{i-1}})}{\displaystyle\sum_{j=0}^{k_i} \vartheta_{ij}b_{k_i,j}(\frac{t-t_{i-1}}{t_{i+1}-t_{i-1}})} W_iB_i(t)}{\displaystyle\sum_{i=1}^{n} \frac{\displaystyle\sum_{j=0}^{k_i} \mu_{ij}\vartheta_{ij}b_{k_i,j}(\frac{t-t_{i-1}}{t_{i+1}-t_{i-1}})}{\displaystyle\sum_{j=0}^{k_i} \vartheta_{ij}b_{k_i,j}(\frac{t-t_{i-1}}{t_{i+1}-t_{i-1}})} W_iB_i(t)} \tag{7.11}$$

It can be seen that both the local and the global form are particular cases of the full form. Namely, the full form reduces to the local form when $\mu_{ij} \equiv 1$, and to the global form when $\vartheta_{ij} \equiv 1$; If $\mu_{ij} \equiv \vartheta_{ij} \equiv 1$, we get a particular instance of the general form.

**Remark 20.** *If in (7.9-7.11) we choose*

$$c_{ij} = \Phi((1-\frac{j}{k_i})t_{i-1} + \frac{j}{k_i}t_{i+1}) \tag{7.12}$$

*for some $C^0$-regular function $\Phi$ defined on $[t_1,t_n]$, we obtain Expo-Rational B-spline and NUERBS forms of all considered types for the classical Bernstein polynomial operator. Notice that this is also a first, particularly simple, instance of a variation diminishing operator in terms of Expo-Rational B-spline and NUERBS form (see also [17] and [16]).*

### 7.1.3 Basic differential geometry of NUERBS

Studying the intrinsic geometry (curvature and torsion) of parametric regular Bézier curves is particularly simple at the end points (see [30]). For intermediate values of the Bézier curve parameter, the intrinsic geometry of the curve can be considered as an interpolation of the geometry at the end points which depend on the control polygon. The situation is the same with the rational Bézier form (see [30]) where the weights of the rational form add additional fine control over the interpolation of the geometry between the end points. In the case of B-splines and NURBS, the situation is the same between each two (different) neighbouring knots; thus, the enhancement in the control of the intrinsic geometry of the B-spline and the NURBS curve compared to the Bézier and rational Bézier curve is due to the increase in the number of knots and the control of their position and multiplicity. This is an essential improvement in the control over the curve geometry, in view of the simple dependence of the curvature and torsion on the NURBS weights in the knots (see [30]).

As with B-splines and NURBS, the intrinsic geometry of Expo-Rational B-splines and NUERBS curves is simplest in the study of the knots (see [16]).

The first important new observation for regular NUERBS curves is that the intrinsic geometry of the global curve depends only on the intrinsic geometry of the local curve in the knots, i.e., it depends neither on the intrinsic parameters of the Expo-Rational B-spline basis function, nor on the global weights $W_i$ in (7.3). More precisely, the following result is true.

**Theorem 7.2.** *Let $t_i$ be one of the knots in (7.3) and assume that the respective local curve $c_i : [t_{i-1},t_{i+1}] \to \mathbb{R}^d$ satisfies the following:*

*(i) either $d = 1,2$ and $c_i \in C^2[t_{i-1},t_{i+1}]$, or $d = 3$ and $c_i \in C^3[t_{i-1},t_{i+1}]$;*

*(ii) $c_i$ is regular at $t_i$, i.e., $\dot{c}_i(t_i) \neq 0$.*
*Then, for the curvature $\varkappa$ of the (global) NUERBS curve (7.3),*

$$\varkappa(t_i) = \frac{|\dot{c}_i(t_i) \wedge \ddot{c}_i(t_i)|}{|\dot{c}_i(t_i)|^3} \tag{7.13}$$

*holds, and for the torsion $\tau$ of the (global) NUERBS curve*

$$\tau(t_i) = \begin{cases} 0, & d = 1,2 \\ \frac{[\dot{c}_i(t_i),\ddot{c}_i(t_i),\dddot{c}_i(t_i)]}{|\dot{c}_i(t_i)\wedge\ddot{c}_i(t_i)|^2}, & d = 3 \end{cases} \tag{7.14}$$

*is fulfilled, where* $\overrightarrow{a} \wedge \overrightarrow{b}$ *is the vector cross product of* $\overrightarrow{a}, \overrightarrow{b}$, *and* $[\overrightarrow{a}, \overrightarrow{b}, \overrightarrow{c}]$ *is the scalar triple product of* $\overrightarrow{a}, \overrightarrow{b}, \overrightarrow{c}$. *In particular,* $\varkappa(t_i)$ *and* $\tau(t_i)$ *depend neither on the weight* $W_i$, *nor on the intrinsic parameters of* $B_i$ *in (7.3).*

*Proof.* Follows from Theorem 7.1. □

**Remark 21.** *In the particular case of (7.11),* $\varkappa(t_i)$ *in (7.13) and* $\tau(t_i)$ *in (7.14) essentially depend, and can be controlled, by the weights* $\vartheta_{ij}$ *of the local rational Bezier curve.*

The second important observation about the geometry of NUERBS is that the much faster speeds and accelerations of the parametrization of NUERBS, and the fact that the Expo-Rational B-Splines are not regular curves at the knots, bring about new qualitative phenomena, as far as geometric continuity is concerned. A most conspicuous (and practically important) example in this respect is the case of Lagrange interpolation by $B_i$ in (7.1) and(7.3). This corresponds to $c_i(t) = c_i = const,\ t \in [t_{i-1}, t_{i+1}]$. In this case, $|\dot{c}_i(t)| \neq 0$ for every i, condition (*ii*) is violated and Theorem 7.2 does not hold. The effect is that here (7.1),(7.3) provide the geometry of a piecewise linear B-spline curve (see Figure 7.3) while retaining $C^\infty$-smooth parametrization. In this case, the NUERBS weights $W_i$ in (7.3) do not influence the geometric form but provide control over the speed of parametrization (see Figure 7.2).

## 7.2 Three-variate tensor product ERBS

In this section we will mainly look at the background, and a short summary of the result of the work done by Einar Rasmussen in his master theses [46]. He has also made all the three figures used in this section.

Often, the problem is too much data. The data comes from some kind of scanning, and the datasets becomes huge. Data reduction is then essential. This is the goal of Trond Brenna in [7]. If instead, the dataset comes from a computer simulation, the opposite problem appears. The dataset gives a coarse grid, and this is a problem in the visualization. To solve this, the ideas was to use the Hermite interpolation property of Expo-Rational B-splines to model 3D scalar data, and by this resample by evaluating the 3D model. In many cases is it possible to get additional information (partial derivatives) in the nodes. Using this will give a much better visual expression of the result of a simulation.

The general formula for a three-variate tensor product ERBS function, $f : \mathbb{R}^3 \to \mathbb{R}^n$, $n > 0$ is,

$$f(u,v,w) = \sum_{i=1}^{n_u} \sum_{j=1}^{n_v} \sum_{k=1}^{n_w} p_{ijk}(u,v,w)\, B_i(u)\, B_j(v)\, B_k(w), \tag{7.15}$$

where $p_{ijk}(u,v,w)$ are the $n_u \times n_v \times n_w$ local functions, and where the three sets of ERBS basis functions $B_i(u)$, $B_j(v)$ and $B_k(w)$ are defined by the three knot vectors, $\{u_i\}_{i=0}^{n_u+1}$, $\{v_j\}_{j=0}^{n_v+1}$ and $\{w_k\}_{k=0}^{n_w+1}$.

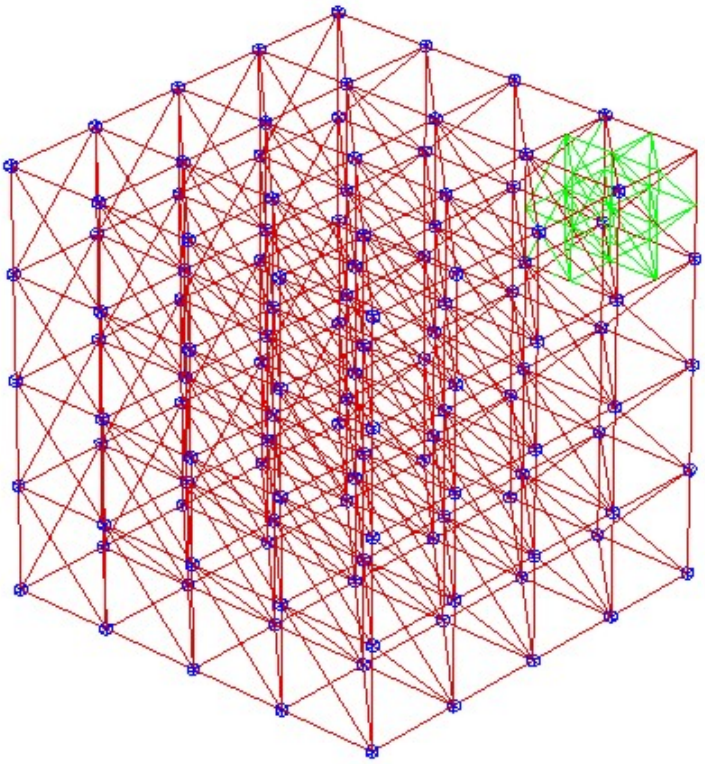

***Figure 7.4:*** *The figure shows a 3D "volume" object displayed as a wireframe. The object is a three-variate tensor product Expo Rational B-spline, where the local volumes are three-variate tensor product Bézier volumes. The local volume at one of the corners is shown in green, all the others are marked with blue cubes.*

Suppose that $n = 1$, i.e. we have a scalar field. This can be modeled by a three-tensor ERBS volume. In Figure 7.4, there is a plot of a wireframe volume object, where we have $n_u = n_v = n_w = 5$. We can see that the total volume (cube) is divided in $4 \times 4 \times 4 = 64$ small cubes. The local volume in the right hand corner (the upper one) is marked with green, and is filling one of the small cubes. If a local volume is on an edge but not in a corner it will fill two small cubes. If a local volume is on a side but not on an edge or in a corner it will fill four small cubes. And if a local volume is in the interior of the big volume it will fill eight small cubes.

We now assume that the local functions are three-variate tensor product Bézier functions, The formula for the three-variate tensor product Bézier functions $p : \mathbb{R}^3 \to \mathbb{R}^n$, $n > 0$, is:

$$p(u,v,w) = \sum_{i=1}^{n_u} \sum_{j=1}^{n_v} \sum_{k=1}^{n_w} c_{ijk} b_i(u) b_j(v) b_k(w), \tag{7.16}$$

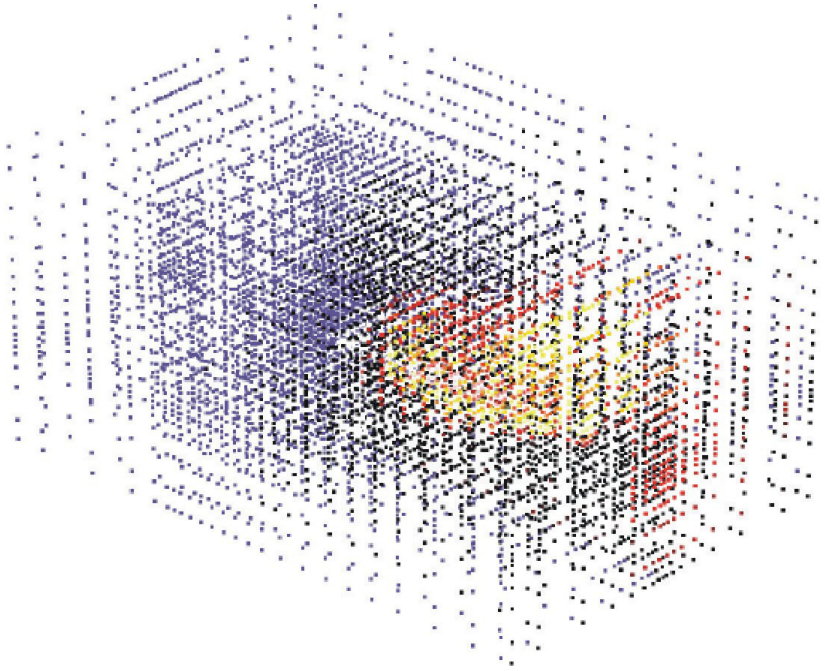

***Figure 7.5:*** *The figure shows the 3D flame data as a particle system. The points are the data given by ComputIT as the result from their simulation program "Kameleon FireEx".*

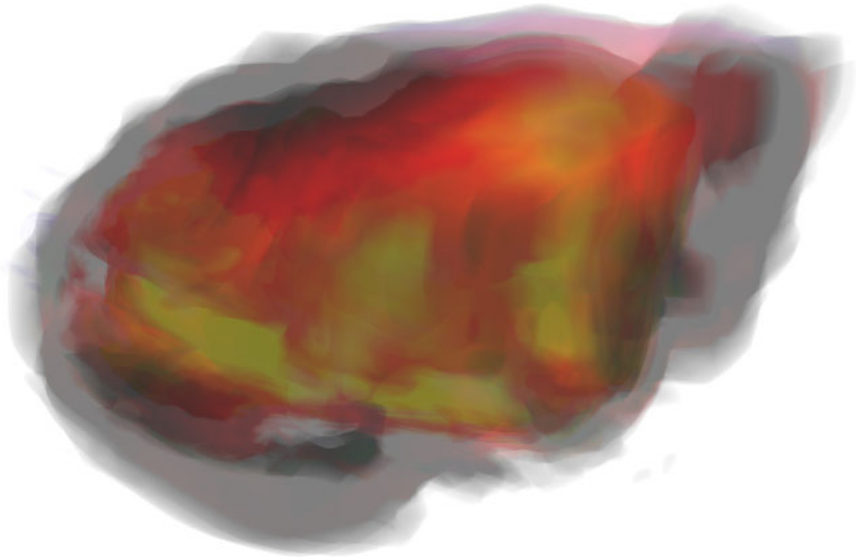

***Figure 7.6:*** *The figure shows a view of a 3D flame object. The object is a three-tensor Expo Rational B-spline volume, where the local volumes are three-tensor Bézier volumes.*

The Hermite interpolation is analogous to the process described in section 5.3.1. Thus we have to introduce 3D-matrices. We can now write (7.16) on the matrix format,

$$p(u,v,w) = \mathbf{C}\,\mathbf{B}_{d_w}(w,1)\,\mathbf{B}_{d_v}(v,1)\,\mathbf{B}_{d_u}(u,1). \tag{7.17}$$

where $\mathbf{C}$ is the 3D coefficient matrix, and $\mathbf{B}_{d_u}(u,1)$, $\mathbf{B}_{d_v}(v,1)$ and $\mathbf{B}_{d_w}(w,1)$ are the Bernstein/Hermite matrices defined in 4.8. The turning of the sequence of the Bernstein/Hermite matrices is due to the nature of the 3D-matrix (see [46]).

We now get the following formula for the Hermite interpolation in the knot $u_i$, $v_j$ and $w_k$,

$$\mathbf{g}_{d_u,d_v,d_w}(u_i,v_j,w_k) = \mathbf{C}_{i,j,k}\,\mathbf{B}_{d_w}(\omega_i(w_k),\delta_{w,k})\,\mathbf{B}_{d_v}(\omega_j(v_j),\delta_{v,j})\,\mathbf{B}_{d_u}(\omega_i(u_i),\delta_{u,i}), \tag{7.18}$$

where $\mathbf{g}_{d_u,d_v,d_w}(u_i,v_j,w_k)$ is thus a 3D analog to 5.18, $d_u$ is the number of partial derivatives in $u$, where $d_v$ is the number of partial derivatives in $v$ and where $d_w$ is the number of partial derivatives in $w$ that are known in the respective point. We also have,

$$\begin{aligned}
\omega_i(u_i) &= \tfrac{u_i-u_{i-1}}{u_{i+1}-u_{i-1}},\\
\omega_j(v_j) &= \tfrac{v_j-v_{j-1}}{v_{j+1}-v_{j-1}},\\
\omega_k(w_k) &= \tfrac{w_k-w_{k-1}}{w_{k+1}-w_{k-1}},
\end{aligned}$$

the affine global/local mappings from definition 2.7, and

$$\begin{aligned}
\delta_{u,i} &= \tfrac{1}{u_{i+1}-u_{i-1}},\\
\delta_{v,j} &= \tfrac{1}{v_{j+1}-v_{j-1}},\\
\delta_{w,k} &= \tfrac{1}{w_{k+1}-w_{k-1}},
\end{aligned}$$

the domain scaling factors defined in Theorem 2.4.

Now the final step for generating the local Bézier volumes is to solve equation 7.18 according to the Bézier coefficients (3D control polygon) $\mathbf{C}_{i,j,k}$,

$$\begin{aligned}
&\mathbf{C}_{i,j,k} =\\
&(((\mathbf{g}_{d_u,d_v,d_w}(u_i,v_j,w_k)\,\mathbf{B}_{d_w}(\omega_k(w_k),\delta_{w,k})^{-T})^T\mathbf{B}_{d_v}(\omega_j(v_j),\delta_{v,j})^{-T})^T\mathbf{B}_{d_u}(\omega_i(u_i),\delta_{u,i})^{-T})^T.
\end{aligned} \tag{7.19}$$

The following example is to build a 3D flame object using data from "ComputIT" (see www.computit.no), a company specialized on fire simulations. The data is generated in their own developed simulator program called "Kameleon FireEx", and is a $25 \times 17 \times 18$ grid, which make a total of 7650 points. In Figure 7.5 the data from "ComputIT" is plotted as a discrete particle system. In this example there is no information about partial derivatives, so divided differences are used to generate this information. In Figure 7.6 we can see a view of the 3D object that is built from the initial data, using Hermite interpolation and a three-variate tensor product ERBS volume with three-variate tensor product Bézier volume as local volumes. More details about the example can be found in [46]. One of the most promising features in the previous example is the possibility to make a dynamic flame by rotating and/or scaling the local Bézier volumes. Studies of the behavior compared to simulation data are, therefore, also a possible topic for further explorations.

# Bibliography

## References


[1] R. L. Bagula and P. Bourke. Trianguloid trefoil . http://astronomy.swin.edu.au/∼pbourke/surfaces/tranguloid/, 2002.

[2] R. L. Bagula and P. Bourke. Bent horns surface . http://astronomy.swin.edu.au/∼pbourke/surfaces/benthorns/, 2003.

[3] A. H. Barr. Global and local deformations of solid primitives. In *SIGGRAPH '84: Proceedings of the 11th annual Conference on Computer Graphics*, pages 21–30, 1984.

[4] D. Bechmann. Multidimensional Free-form Deformation Tools. In *Eurographics´98, State of the Art Report*, 1999.

[5] P. Bourke. Mathematical shell . http://astronomy.swin.edu.au/∼pbourke/surfaces/shell/, 1998.

[6] A. Bredesen. Parameter study of ERBS-curves with regard to data reduction and approximation/interpolation problems. Master theses, Narvik University College, Narvik, Norway, 2006.

[7] T. Brenna. *Data Reduction on Cubical and Spherical Domains Using High Dimensional Tensor Splines*. Dissertation for Degree of Dr. Scient. Unipub, Oslo, 2004.

[8] S. Bu-qing and L. Ding-yuan. *Computational Geometry - Curve and Surface Modeling*. Academic Press Inc. [Harcourt Brace Jovanovich Publishers], London, first edition, 1989.

[9] S. D. Conte and C. de Boor. *Elementary Numerical Analyses*. McGraw-Hill, Singapore, 1983.

[10] S. Coquillart. Extended Free-form deformation: a sculpturing tool for 3D geometric modeling. In *SIGGRAPH '90: Proceedings of the 17th annual Conference on Computer Graphics*, pages 187–196, 1990.

[11] R. Dahl. Realistic simulation using Expo-Rational B-splines and GPU programming. Master theses, Narvik University College, Narvik, Norway, 2005.

[12] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry*. Algorithms and Applications. Springer-Verlag, New York, 1997.

[13] C. de Boor. *A Practical Guide to Splines*, volume 27 of *Applied Mathematical Sciens*. Springer-Verlag, New York, 1978.

[14] P. de Casteljau. Outillages méthodes calcut. Technical report, A. Citroën, Paris, 1959.

[15] P. de Casteljau. Courbes et surfaces à pôles. Technical report, A. Citroën, Paris, 1963.

[16] L. T. Decevsky, A. Lakså, and B. Bang. Expo-Rational B-splines. *International Journal of Pure and Applied Mathematics*, 27(3):319–369, 2006.

[17] L. T. Decevsky, A. Lakså, and B. Bang. NUERBS form of Expo-Rational B-splines. *International Journal of Pure and Applied Mathematics*, 32(1):11–32, 2006.

[18] L. T. Dechevsky, A. Lakså, and B. Bang. Expo-Rational B-splines. Preprint 2/2004, Narvik University College, Narvik, Norway, 2004.

[19] L. T. Dechevsky, A. Lakså, and B. Bang. NUERBS form of Expo-Rational B-splines. Preprint 1/2004, Narvik University College, Narvik, Norway, 2004.

[20] M. P. do Carmo. *Differential geometry of curves and surfaces*. Prentic Hall, Inc., New Jersey, 1976.

[21] T. Dokken, T. R. Hagen, and J. M. Hjelmervik. The GPU as a high performance computational resource. In *Proceeding from Spring Conference on Computer Graphics, Budmerice Castle, Bratislave, Slovakia, May 12-14 2005*.

[22] G. Farin. Triangular Bernstein-Bézier patches. *Computer Aided Geometric Design*, 3(2):83–128, 1986.

[23] G. Farin. *Curves and Surfaces for CAGD*. Morgan Kaufmann, San Francisco, California, fifth edition, 2002.

[24] G. Farin. Class A Bezier curves. *Computer Aided Geometric Design*, 23(7):573–581, 2006.

[25] J. Fauvel, R. Wilson, and R.Flood (Eds.). *Möbius and his Band: Mathematics and Astronomy in Nineteenth-century Germany*. Oxford University Press, Oxford, England, fifth edition, 1993.

[26] M. S. Floater, K. Hormann, and G. Koz. A general construction of barycentric coordinates over convex polygons. *Advances in Computational Mathematics*, 24(1-4):311–331, 2006.

[27] M. S. Floater, G. Koz, and M. Reimers. Mean value coordinates in 3D. *Computer Aided Geometric Design*, 22(7):623–631, 2005.

[28] L. A. Frøyland, A. Lakså, and J. Pajchel. Modelling geological structures using splines. In T. Lyche and L. Schumaker, editors, *Mathematical Methods in Computer Aided Geometric Design II*, pages 287–296. Academic Press (New York), 1992.

[29] J. Gao. Expo-Rational B-splines to/from NURBS. Master theses, Narvik University College, Narvik, Norway, 2006.

[30] M.-S.Kim (Eds.) G.Farin, J.Hoschek. *Handbook of Computer Aided Geometric Design*. Elsevier, North Holland, 2002.

[31] D. Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys*, 23(1):5–48, 1991.

[32] A. Gray. *Modern Differentioal Geometry of Curves and Surfaces*. CRC Press, Inc., Boca Raton, Florida, first edition, 1993.

[33] S. Guillet and J. C. Leon. Parametrically deformed free form surfaces as a part of variational model. *Computer Aided Design*, 30(1), 1998.

[34] H. Jeffreys and B. S. Jeffreys. L. F. Richarddon's methode. *Methods of Mathematical Physics*, 3rd ed.:288, 1988.

[35] A. Lakså and B. Bang. Simulating rolling and colliding balls on freeform surfaces with friction. In J. Amundsen, H. I. Anderson, E. Celledoni, T. Gravdahl, F. A. Michelsen, H. R. Nagel, and T. Natvig, editors, *SIMS 2005*, pages 253–262. Tapir Academic Press, 2005.

[36] A. Lakså, B. Bang, and L. T. Dechevsky. Exploring expo-rational B-splines for curves and surfces. In M. Dæhlen, K. Mørken, and L. Schumaker, editors, *Mathematical methods for Curves and Surfaces*, pages 253–262. Nashboro Press, 2005.

[37] A. Lakså, B. Bang, and A. R. Kristoffersen. GM_lib, a C++ library for geometric modeling. Technical report, Narvik University College, Narvik, Norway, 2006.

[38] A. Lakså and M. Floater. Reference manual SI Spline Library. Technical report, Senter for industriforskning, Oslo, Norway, 1990.

[39] S. Lee, G. Wolberg, and S. Y. Shin. Scattered data interpolation with multilevel B-splines. *IEEE Transactions on Visualization and Computer Graphics*, 3(3):228–244, 1997.

[40] T. Lyche and K. Mørken. Knot removal for parametric B-spline curve and surfaces. *Computer Aided Geometric Design*, 4(3):217–230, 1987.

[41] T. Lyche and K. Strøm. Knot insertion for natural splines. *Annals of Numerical Mathematics*, (3):221–246, 1996.

[42] E. Mehlum. Nonlinear splines. *Computer Aided Geometric Design*, pages 173–207, 1974.

[43] E. Mehlum. Appell and apple (nonlinear splines in space). In Larry L. Schumaker Morten Dæhlen, Tom Lyche, editor, *Mathematical Methods for Curves and Surfaces*, pages 365–383. Vanderbilt University Press (Nashville & London), 1995.

[44] E. Mehlum and P. F. Sørensen. Example of an existing system in the ship building industry: the AUTOCON system. *Proc. Roy. Soc. London*, A 321:219–233, 1971.

[45] T. Moguchaya, J. Gundersen, N. Grip, L. T. Dechevsky, B. Bang, A. Lakså, E. Quak, and B. Tong. Curve and surface fitting by wavelet shrinkage using GM_Waves. In M. Dæhlen, K. Mørken, and L. Schumaker, editors, *Mathematical methods for Curves and Surfaces*, pages 263–274. Nashboro Press, 2005.

[46] E. Rasmussen. Improved visualization of scalar data using interpolation. Master theses, Narvik University College, Narvik, Norway, 2006.

[47] W. Rudin. *Principles of Mathematical Analysis. I.* McGraw-Hill Inc., Singapore, third edition, 1976. Functional analysis.

[48] L. L. Schumaker. *Spline Functions: Basic Theory*. A Wiley-interscience publication. John Wiley & Sons Inc., New York, 1981.

[49] T. W. Sederberg, D. L. Cardon, D. G. Finnigan, J. Zheng, and T. Lyche. T-spline Simplification and Local Refinement. *ACM Transactions on Graphics*, 23(2):276–283, 2004.

[50] T. W. Sederberg, J. Zheng, A. Bakenow, and A. Nasri. T-splines and T-nurces. *ACM Transactions on Graphics*, 22(3):477–484, 2003.

[51] T.W. Sederberg and S.R. Parry. Free-form deformation of solid geometric models. In *SIGGRAPH '86: Proceedings of the 13th annual Conference on Computer Graphics*, pages 151–160, 1986.

[52] ANSI/IEEE std. 754-1985. IEEE standard for binary floating-point arithmetic. Copyright standard, The Institute of Electrical and Electronical Engineers, Inc., New York, USA, 1985.

[53] B. A. Ulsund. Expo-Rational B-spline evaluator on GPU. Master theses, Narvik University College, Narvik, Norway, 2005.

[54] I. Vardi. The Euler-Maclaurin Formula. *Computational Recreation in Mathematica*, pages 159–163, 1991.

[55] Y. Wang. Parameter study of ERBS-surfaces with regard to data reduction and approximation/interpolation problems. Master theses, Narvik University College, Narvik, Norway, 2006.

[56] Y. Wang, J. Zheng, and H. S. Seah. Conversion between T-splines and Hierarchical B-splines. In *Proceedings of the 8th LASTED International Conference COMPUTER GRAPHICS AND IMAGING*, pages 8–13, August 2005.

[57] J. Warren. Barycentric coordinates for convex polytopes. *Advances in Computational Mathematics*, 6(2):97–108, 1996.

[58] E. W. Weisstein. Rose. http://mathworld.wolfram.com/Rose.html, 2006.

[59] Wikipedia. Basic linear algebra subprograms — wikipedia, the free encyclopedia, 2006. [Online; accessed 20-April-2006].

[60] Wikipedia. IEEE 754r, an ongoing revision of the ieee 754 floating point standard. http://en.wikipedia.org/wiki/IEEE_754r, 2006.

[61] K. C. Wu, T. Fernando, and H. Tawfik. Freesculptor: A computer-aided freeform design environment. *gmag*, 00:188, 2003.

# List of Acronyms

## Acronyms

| | | |
|---|---|---|
| 3D | – | Describes objects imbedded in $\mathbb{R}^3$ |
| API | – | Application Programming Interface |
| C++ | – | Object oriented programming language |
| CAGD | – | Computer Aided Geometric Design |
| CMA | – | Center of Mathematics for Applications, University of Oslo |
| ComputIT | – | Computational Industry Technologies AS – Norwegian fire-simulation company |
| ERBS | – | Expo-Rational B-splines |
| eVITA | – | e-Science, includes material from both Computational Science and Software Engineering as well as other topics such as graphics, virtual reality, general computer science |
| FFD | – | Free Form deformation method |
| GM_Lib | – | C++ library for geometric modeling and simulations, developed by the R&D Simulation Group at Narvik University College |
| GM_Wave | – | C++ wavelet library, developed by the R&D Simulation Group at Narvik University College |
| GPU | – | Graphic Processor Unit |
| GPGPU | – | GPU programming for General Purpose |
| GUI | – | Graphical User Interface |
| IEEE | – | The Institute of Electrical and Electronics Engineers, Inc. |
| Kameleon FireEx | – | A simulation program for fire simulation, developed by ComputIT |
| NaN | – | Not a number |
| NUC | – | Narvik University College |
| NURBS | – | Non-uniform rational B-splines |
| NUERBS | – | The ERBS analog to NURBS |
| ODE | – | Ordinary Differential Equation |
| OpenGL | – | Software interface to graphics hardware |
| PDE | – | Partial Differential Equation |

| | | |
|---|---|---|
| SINTEF | – | The Foundation for Scientific and Industrial Research at the Norwegian Institute of Technology |
| SISL | – | SINTEFs spline library |
| UIO | – | University of Oslo |

# Index

.

# List of Figures

.

# List of Tables