

Blending technics for Curve and Surface constructions

By

Arne Lakså

Preface

Geometry – from Ancient Greek, earth measurement – has been an important ingredient of the development of science and later also industry, design and production.

In modern time, the Geometric modeling community is established, and has now a 50-year-old history. The first pioneers were motivated by the introduction of computers in design, construction and manufacturing. The goal was basically to provide methods and algorithms for curve and surface representations and calculations on these, and to combine curve and surface methods in computer graphics and simulations. By doing this, they started the develop of a new discipline called geometric modeling, including computer aided geometric design, solid modeling, algebraic geometry, and computational geometry. Computer aided geometric design (CAGD) is the central part of the field. It started with the development of methods and algorithms for CAD/CAM. Today, however, is support for virtual reality/design and computer games, simulators and animations in movies and TV productions as important areas.

Curves and surfaces for computer aided geometric design started with classical geometric objects as line, arc, plane, sphere, cylinder etc. The next step in the development of free-form geometry was Bezier, Hermite, B-spline and rational variants as rational Bezier and NURBS and subdivision surfaces. The development began in earnest in the 1960s and went with full force until 1990. It is, however, still an ongoing development that includes T-splines and other improvement as Multivariate splines. New results also includes the introduction of generalized B-splines and thus it is still a long way to go. This way of constructing curves and surface is the most important today and will probably be it as far as we can see. It is the defacto industrial standard. What we can call a 2nd generation curve and surface construction started around year 2000, and it is just in its initial phase. It includes different types of curves and surfaces constructed by blending technics.

For me the work with blending technics accelerated one day the summer of 2003. I went to my college Børre Bang at his office, to discuss improvements of GM.lib , a C++ open source programming library for geometric modeling (see [93]). Lubomir Dechevsky was already there. He asked if we could look at a function he believed could be the limit of a polynomial B-spline when the degree (and, thus number of knots) tends to infinite (described in [43]). Since we then were working with the graphical part of a programming library GM.lib , we implemented this new basis function and tried it for curves in \mathbb{R}^3 . The result was a piecewise linear curve just like 1st degree B-spline curves, but there was, however, a big difference; this new curve was C^∞ -smooth but obviously only G^0 ,

i.e. piecewise linear. After a short discussion we replaced the coefficients of the curve with something we called “local curves”, and Expo-Rational B-spline (ERBS) curves were born. Later this has been expanded to a more general concept we call Generalized Expo-Rational B-spline.

This book was first planned to be about blending technics. However, because of input from both students, this manuscript is also influenced by the fact that parts of it have been used as compendiums in courses in a master program in Computer Science at Narvik University College (NUC), and industrial cooperation partners I decided to expand the book to include free-form geometry in general. Free-form geometry has attracted my attention for the more than 20 years both as a scientist, teacher and also for industrial implementations ([94], [60]).

Acknowledgements

First I will express my gratitude to my colleagues Lubomir T. Dechevsky and Børre Bang for a very fruitful cooperation. Next I am especially grateful to Tom Lyche, Knut Mørken for very valuable discussions and useful advice.

I am also grateful to Tor Dokken, Bernt Bremsdal, Joakim Gundersen, Arnt Kristoffersen, Jostein Bratlie and Rune Dalmo for discussions, helping with GM.lib, documentation and other practical issues, and to Susan Jane Berntsen for language consultations.

Finally, I would like to give big thanks to my wife Marit, for bearing with me through this work.

December 2012

Arne Lakså

Contents

Preface	v
1 Introduction	1
1.1 Historical notes about geometry	3
1.1.1 Modern geometry	7
1.1.2 Elementary geometry	9
1.1.3 Non-Euclidean geometry	10
1.1.4 Introduction of mathematical rigor	12
1.1.5 Algebraic geometry	12
1.1.6 New notes	14
1.2 Geometric modeling	16
1.3 Algorithmic language	18
1.4 Notations	20
1.5 Overview of this book	21
2 Mathematical spaces and notations	47
2.1 Euclidean spaces, cartesian coordinates and vector spaces	48
2.2 Homeomorphism, diffeomorphism and manifolds	49
2.2.1 Local/global parametrization, charts and atlas	50
2.3 Compact spaces	51
2.4 Affine space	52
2.5 Projective space and Grassmannien	55
2.6 Homogenous coordinates	56
3 GMlib, an Open Source programming library	57
3.1 The modules of GMlib	58
3.2 The core module	60
3.3 Scene hierarchy - GMlib:SH	60
3.4 Basic Linear Algebra Subprograms - BLAS	60
3.5 QT - a cross-platform application framework	66
I Curves	67
4 Parametric Curves	69

4.1	Differentiations	72
4.1.1	Regular curves - arc length parameterization	73
4.1.2	Reparameterization	73
4.1.3	Curvature	74
4.2	Basis functions	75
4.3	Hermite Curves	76
4.4	Bézier Curves	81
4.4.1	Bernstein polynomial	84
4.4.2	Factorization and de Casteljau's Corner cutting algorithm	88
4.4.3	The Bernstein/Hermite matrix	91
5	Spline Curves	95
5.1	History of interpolation	95
5.1.1	Divided differences and Newton polynomial	101
5.1.2	Lagrange polynomial	102
5.1.3	Hermite interpolation	104
5.2	Hermite spline	106
5.3	Cubic spline interpolation	107
5.4	History of B-Splines	110
5.5	Modern B-splines	114
5.5.1	B-spline curves	116
5.5.2	Bézier curves and de Casteljau algorithm	117
5.5.3	The B-spline factor matrix $T(t)$	119
5.5.4	Commutativity relations between $T(t)$ matrices and their derivatives	121
5.5.5	B-splines on Matrix notations	123
5.5.6	Examples of B-splines, matrix notation, de Casteljau's algorithm and derivatives	123
5.5.7	Hermite spline interpolation on B-spline form	127
5.5.8	Cubic spline interpolation on B-spline form	129
5.5.9	B-splines and knot insertion	131
5.5.10	NURBS	133
5.5.11	Important properties of spline functions on B-spline form	134
5.6	Blossoming	136
5.7	Use of blending in construction of geometry	136
5.8	Subdivision	138
5.8.1	Catmull-Rom Spline	138
6	Blending	141
6.1	B-functions	141
6.2	Blending of two functions	143
6.2.1	Examples, blending of order zero and order one	146
6.2.2	Examples, connecting two curves by using a B-function	148
6.3	Beta-functions, the group of polynomial B-functions	150
6.3.1	Beta-functions, differentiation	154
6.4	RB-functions, the group of rational B-functions	155
6.4.1	RB-functions, differentiation	158

6.4.2	RB-functions with a balance parameter	159
6.5	TB-functions, the series of trigonometric B-functions	160
6.5.1	TB-functions, differentiation	162
6.6	Expo-Rational B-function, a complete B-function	163
6.6.1	The slope parameter γ	165
6.6.2	The balance parameter μ	166
6.6.3	The asymmetric tightening parameters α and β	167
6.6.4	ERB-functions, differentiation	169
6.7	Other Expo-Rational B-functions	171
6.7.1	Logistic Expo-Rational B-function	173
6.8	Point-, Order- and Balance-symmetry of B-functions	173
6.9	Computing B-functions	177
7	Blending splines	179
7.1	B-splines with B-function	180
7.2	A simple version	183
7.3	Generalized definition	185
7.4	Basic properties	188
7.5	The Scalable subset	196
7.6	The default set of intrinsic parameters	202
7.7	Expo-Rational B-spline functions	206
7.8	Knot vectors and continuity	209
7.9	Hermite Interpolation properties	210
7.10	Influence of the intrinsic parameters	212
8	GERBS – Curves	217
8.1	Definition/implementation of “open/closed” curves	219
8.2	Evaluation, value and derivatives	221
8.3	Bézier curves as local curves	224
8.3.1	Local Bézier curves and Hermite interpolation	227
8.3.2	Sampling and preevaluation	228
8.3.3	Examples	230
8.4	Circular arcs as local curves	235
8.4.1	Local Arc curves and modified Hermite interpolation	239
8.4.2	Examples	241
8.4.3	Reparametrization and using an approximative curve length parametrization	246
8.5	Affine transformation on local curves	251
II	Surfaces	257
9	Parametric Surfaces	259
9.1	Differentiation	261
9.1.1	The differential dS_p	262
9.1.2	Curves on surfaces	262

9.1.3	The tangent plane $T_q(S)$	264
9.1.4	First fundamental form	265
9.1.5	Second fundamental form	267
9.2	Surface of revolution	269
9.3	Surface by sweeping	270
9.4	Surfaces based on Curve constructions	273
9.5	Boolean sum surface	274
9.5.1	Coons Patch, bilinear blending	275
9.5.2	Coons Patch, bicubically blending	276
9.5.3	Hermite blending surface	279
9.5.4	Curves on triangular surfaces	290
10	Tensor Product Surfaces	295
10.1	Definition/implementation of “open/closed” Surfaces	297
10.2	Evaluation, value and derivatives	299
10.3	Bézier patches as local patches	305
10.3.1	Local Bézier patches and Hermite interpolation	306
10.3.2	Examples of Hermite interpolations	308
10.4	Free form sculpturing using tensor product ERBS surfaces	320
10.5	Computational and implementational aspects for tensor product ERBS surfaces	324
11	Triangular Surfaces	329
11.1	Homogenous barycentric coordinates for simplices	330
11.2	Bézier triangles	332
11.3	The general set of Expo-Rational B-spline basis-function in homogeneous barycentric coordinates	333
11.4	ERBS triangles, definition and evaluators	338
11.5	Local Bézier triangles and Hermite interpolation	343
11.6	Sub-triangles from general parametrized surfaces as local triangles	354
11.7	Surface approximation by triangulations.	356
11.8	Epilogue	362
A	Evaluators, ERB-function, reliability, precision and efficiency	369
A.1	Reliability in evaluations	370
A.2	ERBS-evaluator and derivatives	374
A.3	ERBS-evaluator based on Romberg-integration	379
A.4	Practical ERBS-evaluator using preevaluation and interpolation	383
Bibliography		393
List of Acronyms		405
Index		407

Chapter 1

Introduction

The History of Geometry may be roughly divided into the five periods:

1. The synthetic geometry of the Greeks (600BC to 200BC);¹
2. The birth of analytic geometry (renaissance), in which the synthetic geometry merged into the coordinate geometry (1600 to 1650);
3. Application of the calculus to geometry (1650 to 1800);
4. The renaissance of pure geometry (the nineteenth century), characterized by the descriptive geometry, the new synthetic and analytic geometry, the non-Euclidean geometry, and the more elementary geometry of the triangle, and the merging of analytic and the synthetic geometry (1800 to 1920);
5. Industrial geometry, related to modern geometric modeling (Computer aided geometric design, computational geometry, CAD/CAM, and virtual worlds) and computations (finite element, finite difference etc.) (1920 - today).

This book is mostly about the last period, and especially the periods after world war II. The period of industrial geometry began early in the twentieth century, but the development of industrial geometry greatly accelerated when computers came into use after World War II, and especially in the 1960s.

The community of geometry started to expand in the late 1950s. The pioneers in the industrial period were often connected to ship-, airplane- or car-industry and motivated by the introduction of computers in design, construction and manufacturing. The goal was basically to provide methods and algorithms for curve and surface presentations and calculations on these, and to combine curve and surface methods in computer graphics and simulations. By doing this, they started the development of a new discipline called geometric modeling, including computer aided geometric design, solid modeling, algebraic geometry, and computational geometry. Computer aided geometric design (CAGD)

¹Greek geometry ended mainly by Archimedes. Beside or after the Greek area, geometry was in some sense developed in parallel or followed by Indian, Chinese and Arabic geometry. Greek geometry, however, was the direct predecessor of the analytical geometry of the point 2, because Euclides 13 books on geometry, titled “The Elements of Geometry” [50] was the direct predecessor.



Figure 1.1: Euclid of Alexandria, the “Father of Geometry”. A statue at Oxford University Museum of Natural History.

is the central part of the field. It started with the development of methods and algorithms for CAD/CAM. Today, however, is support for the development of virtual reality, virtual design, virtual prototyping, virtual engineering, computer games, simulations and animations in movies and TV productions an equally important area for CAGD and thus geometric modeling.

Curves and surfaces for computer aided geometric design started with classical geometry as line, arc, plane, sphere, cylinder etc. The next step in the development was free-form geometry represented with Bézier, Hermite, B-spline, rational Bézier, NURBS, subdivision surfaces etc. The development of this first part started in the 1960s and went with full force until 1990. It is however, still an ongoing development that now includes T-splines, LR-splines, Multivariate splines etc. New results also includes further development of generalized B-splines and therefore it is still a long way to go. This way of constructing curves and surface is the most important construction today and will probably be it as far as we can see. It is the defacto industrial standard.

What we can call a 2nd way of constructing “free-form” curves and surfaces also started in the 1960s but became more actual around year 2000 because of access to computational resources and it is therefore just in its initial phase. It includes different types of curves and surfaces constructed by blend technics.

This book will first give an introduction to the curves and surfaces that today are the industrial standard, and then give an introduction to the new generation of curves and surfaces constructed by blend technics. Occasionally we will go back to historical notes about geometry development.

In Figure 1.1 is there a statue of the father of geometry, Euclid. The 13 books on geometry “The Elements of Geometry” has been and is still important to give insight in geometry. The next section will give an overview of the history of geometry mostly before the last period.

1.1 Historical notes about geometry

This section is mostly for special interests. It gives no formulas or specific insight into geometry, but an overview of the different periods in the development of geometry from geometry articulated in words, drawings and postulates, via the introduction of coordinates and calculations, via the big leap through the nineteenth century, and until geometry finally became an important ingredient for industrial development.

We first follow the first four periods at a deeper way, and put names to the people who contributed² to the development in geometry, and we will also give a description of the development. The four periods can be described short as:

1. The synthetic geometry of the Greeks, practically closing with Archimedes and documented by Euclid in [50];
2. The birth of analytic geometry, in which the synthetic geometry of Guldin, Desargues, Kepler, and Roberval merged into the coordinate geometry of Descartes [45] and Fermat;
3. The period 1650 to 1800, characterized by the application of the calculus to geometry, and including the names of Newton, Leibnitz, the Bernoullis, Clairaut, Maclaurin, Euler, and Lagrange, each an analyst rather than a geometer;
4. The nineteenth century, the renaissance of pure geometry, characterized by the descriptive geometry of Monge, the modern synthetic of Poncelet, Steiner, von Staudt, and Cremona, the modern analytic founded by Plucker, the non-Euclidean hypothesis of Lobachevsky and Bolyai, and the more elementary geometry of the triangle founded by Lemoine. It is quite impossible to draw the line between the analytic and the synthetic geometry of the nineteenth century, in their historical development, and Arts.

Curves:

The Analytic geometry that Descartes gave to the world in 1637, see [45], was limited to planar curves ($\in \mathbb{R}^2$). The main characteristics that are common to all algebraic curves was shortly after this detection. To this theory, Newton contributed with three celebrated theorems on the *Enumeratio linearum tertii ordinis* (1706), see [112]. Others properties were discovered by Cotes (1722), Maclaurin, and Waring (1762, 1772, etc.). The scientific foundations of the theory of plane curves may be ascribed to Euler (1748) and Cramer (1750). Euler distinguished between algebraic and transcendent curves³, and he attempted a classification of the algebraic curves. Cramer is well known for the “paradox” which bears his name, an obstacle which Lame (1818) finally removed from the theory. Cramer also made an effort to introduce the theory of singularities of algebraic curves on a scientific foundation, although the theory was first really finalized by Poncelet.

²The text is based on, and partly quoted from two books, from 1906 “History of Modern Mathematics” written by David Eugene Smith, [132], and from 1961 “Introduction to Geometry” by Coxeter, [29].

³Algebraic curves are “level curves”, curves describes by an expression that is equal to zero, se page ..., transcendent curves are curves that are not algebraic.

Surfaces:

Meanwhile the study of surfaces had moved on. Descartes had suggested that his geometry could be extended to three-dimensional space, Wren (1669) had discovered the two systems of generating lines on the hyperboloid of one sheet, and Parent (1700) had referred a surface to three coordinate planes. The geometry of three dimensions began to assume definite shape, however, in a memoir of Clairaut's (1731), in which, at the age of sixteen, solved many of the problems relating to curves of "double curvature". Euler (1760) laid the foundations for the analytic theory of curvature of surfaces, attempting the classification of those of the second degree as the ancients had classified curves of the second order. Monge, Hachette, and other members of that school entered into the study of surfaces with great zeal. Monge introduced the notion of families of surfaces, and discovered the relation between the theory of surfaces and the integration of partial differential equations, enabling each to be advantageously viewed from the standpoint of the other. The theory of surfaces attracted a long list of contributors in the nineteenth century, including most of the geometers whose names are mentioned in the present section.

Möbius began his contributions to geometry in 1823, and four years later published his *Barycentrische Calcül* [109]. In this great work he introduced homogeneous coordinates with the attendant symmetry of geometric formulas, the scientific exposition of the principle of signs in geometry, and the establishment of the principle of geometric correspondence simple and multiple. He also (1852) summed up the classification of cubic curves, a service rendered by Zeuthen (1874) for quartics. To the period of Möbius also belong Bobillier (1827), who first used trilinear coordinates, and Bellavitis, whose contributions to analytic geometry were extensive. Gergonne's labors will be mentioned later.

Plücker, of all contributors to analytic geometry, he stands foremost. In 1828 he published the first volume of his "Analytisch-geometrische Entwickelungen", in which appeared the modern abridged notation, and which marks the beginning of a new era for analytic geometry. In the second volume (1831) he sets forth the present analytic form of the principle of duality. To him is due (1833) the general treatment of foci for curves of higher degree, and the complete classification of plane cubic curves (1835) which had been so frequently tried before him. He also gave (1839) an enumeration of plane curves of the fourth order, which Bragelogne and Euler had attempted. In 1842 he gave his celebrated "six equations" by which he showed that the characteristics of a curve (order, class, number of double points, number of cusps, number of double tangents, and number of inflections) are known when any three are given. To him is also due the first scientific dual definition of a curve, a system of tangential coordinates, and an investigation of the question of double tangents, a question further elaborated by Cayley (1847, 1858), Hesse (1847), Salmon (1858), and Dersch (1874). The theory of ruled surfaces, opened by Monge, was also extended by him. Possibly the greatest service rendered by Plucker was the introduction of the straight line as a space element, his first contribution (1865) being followed by his well-known treatise of the subject (1868-69). In this work he treats certain general properties of complexes, congruences, and ruled surfaces, as well as special properties of linear complexes and congruences, subjects also considered by Kummer and by Klein and others of the late 1800-school. It is not a little due to Plucker that the concept of 4- and hence n-dimensional space, already suggested by Lagrange and Gauss,

became the subject of later research. Riemann, Helmholtz, Lipschitz, Kronecker, Klein, Lie, Veronese, Cayley, d'Ovidio, and many others have elaborated the theory. The regular hypersolids in 4-dimensional space have been the subject of special study by Scheffler, Rudel, Hoppe, Schlegel, and Stringham.

Among Jacobi's contributions is the consideration (1836) of curves and groups of points resulting from the intersection of algebraic surfaces, a subject carried forward by Reye (1869). To Jacobi is also due the conformal representation of the ellipsoid on a plane, a treatment completed by Schering (1858). The number of examples of conformal representation of surfaces on planes or on spheres has been increased by Schwarz (1869) and Amstein (1872).

In 1844 Hesse, whose contributions to geometry in general are both numerous and valuable, gave the complete theory of inflections of a curve, and introduced the so-called Hessian curve as the first instance of a covariant of a ternary form. He also contributed to the theory of curves of the third order, and generalized the Pascal and Brianchon theorems on a spherical surface. Hesse's methods have recently been elaborated by Gundelfinger (1894).

Besides contributing extensively to synthetic geometry, Chasles added to the theory of curves of the third and fourth degrees. In the method of characteristics which he worked out may be found the first trace of the Abzählende Geometrie which has been developed by Jonquières, Halphen (1875), and Schubert (1876, 1879), and to which Clebsch, Lindemann, and Hurwitz have also contributed. The general theory of correspondence starts with Geometry, and Chasles (1864) undertook the first special researches on the correspondence of algebraic curves, limiting his investigations, however, to curves of deficiency zero. Cayley (1866) carried this theory to curves of higher deficiency, and Brill (from 1873) completed the theory.

Cayley's influence on geometry was very great. He early carried on Plucker's consideration of singularities of a curve, and showed (1864, 1866) that every singularity may be considered as compounded of ordinary singularities so that the "six equations" apply to a curve with any singularities whatsoever. He thus opened a field for the later investigations of Noether, Zeuthen, Halphen, and H. J. S. Smith. Cayley's theorems on the intersection of curves (1843) and the determination of self-corresponding points for algebraic correspondences of a simple kind are fundamental in the present theory, subjects to which Bacharach, Brill, and Noether have also contributed extensively. Cayley added much to the theories of rational transformation and correspondence, showing the distinction between the theory of transformation of spaces and that of correspondence of loci. His investigations on the bitangents of plane curves, and in particular on the twenty-eight bitangents of a non-singular quartic, his developments of Plucker's conception of foci, his discussion of the osculating conics of curves and of the sextactic points on a plane curve, the geometric theory of the invariants and covariants of plane curves, are all noteworthy. He was the first to announce (1849) the twenty-seven lines which lie on a cubic surface, he extended Salmon's theory of reciprocal surfaces, and treated (1869) the classification of cubic surfaces, a subject already discussed by Schläfli. He also contributed to the theory of scrolls (skew-ruled surfaces), orthogonal systems of surfaces, the wave surface, etc.,

and was the first to reach (1845) any very general results in the theory of curves of double curvature, a theory in which the next great advance was made (1882) by Halphen and Noether. Among Cayley's other contributions to geometry is his theory of the Absolute, a figure in connection with which all metrical properties of a figure are considered.

Clebsch was also prominent in the study of curves and surfaces. He first applied the algebra of linear transformation to geometry. He emphasized the idea of deficiency (*Geschlecht*) of a curve, a notion which dates back to Abel, and applied the theory of elliptic and Abelian functions to geometry, using it for the study of curves. Clebsch (1872) investigated the shapes of surfaces of the third order. Following him, Klein attacked the problem of determining all possible forms of such surfaces, and established the fact that by the principle of continuity all forms of real surfaces of the third order can be derived from the particular surface having four real conical points. Zeuthen (1874) has discussed the various forms of p plane curves of the fourth order, showing the relation between his results and those of Klein on cubic surfaces. Attempts have been made to extend the subject to curves of the n th order, but no general classification has been made. Quartic surfaces have been studied by Rohn (1887) but without a complete enumeration, and the same writer has contributed (1881) to the theory of Kummer surfaces.

Lie has adopted Plucker's generalized space element and extended the theory. His sphere geometry treats the subject from the higher standpoint of six homogeneous coordinates, as distinguished from the elementary sphere geometry with but five and characterized by the conformal group, a geometry studied by Darboux. Lie's theory of contact transformations, with its application to differential equations, his line and sphere complexes, and his work on minimum surfaces are all prominent.

Of great help in the study of curves and surfaces and of the theory of functions are the models prepared by Dyck, Brill, O. Henrici, Schwarz, Klein, Schönlies, Kummer, and others.

The Theory of Minimum Surfaces has been developed along with the analytic geometry in general. Lagrange (1760-61) gave the equation of the minimum surface through a given contour, and Meusnier (1776, published in 1785) also studied the question. But from this time on for half a century little that is noteworthy was done, save by Poisson (1813) as to certain imaginary surfaces. Monge (1784) and Legendre (1787) connected the study of surfaces with that of differential equations, but this did not immediately affect this question. Scherk (1835) added a number of important results, and first applied the labors of Monge and Legendre to the theory. Catalan (1842), Björling (1844), and Dini (1865) have added to the subject. But the most prominent contributors have been Bonnet, Schwarz, Darboux, and Weierstrass. Bonnet (from 1853) has set forth a new system of formulas relative to the general theory of surfaces, and completely solved the problem of determining the minimum surface through any curve and admitting in each point of this curve a given tangent plane. Weierstrass (1866) has contributed several fundamental theorems, has shown how to find all of the real algebraic minimum surfaces, and has shown the connection between the theory of functions of an imaginary variable and the theory of minimum surfaces.

1.1.1 Modern geometry

Descriptive, Projective, and Modern Synthetic Geometry are so interwoven in their historic development that it is even more difficult to separate them from one another than from the analytic geometry just mentioned. Monge had been in possession of his theory for over thirty years before the publication of his *Geometric Descriptive* (1800), a delay due to the jealous desire of the military authorities to keep the valuable secret. It is true that certain of its features can be traced back to Desargues, Taylor, Lambert, and Frezier, but it was Monge who worked it out in detail as a science, although Lacroix (1795), inspired by Monge's lectures in the École Polytechnique, published the first work on the subject. After Monge's work appeared, Hachette (1812, 1818, 1821) added materially to its symmetry, subsequent French contributors being Leroy (1842), Olivier (from 1845), de la Gournerie (from 1860), Vallée, de Fourcy, Adhémar, and others. In Germany leading contributors have been Ziegler (1843), Anger (1858), and especially Fiedler (3d edn. 1883-88) and Wiener (1884-87). At this period Monge by no means confined himself to the descriptive geometry. So marked were his labors in the analytic geometry that he has been called the father of the modern theory. He also set forth the fundamental theorem of reciprocal polars, though not in modern language, gave some treatment of ruled surfaces, and extended the theory of polars to quadrics.

Monge and his school concerned themselves especially with the relations of form, and particularly with those of surfaces and curves in a space of three dimensions. Inspired by the general activity of the period, but following rather the steps of Desargues and Pascal, Carnot treated chiefly the metrical relations of figures. In particular he investigated these relations as connected with the theory of transversals, a theory whose fundamental property of a four-rayed pencil goes back to Pappos, and which, though revived by Desargues, was set forth for the first time in its general form in Carnot's *Geometrie de Position* (1803), and supplemented in his *Theorie des Transversales* (1806). In these works he introduced negative magnitudes, the general quadrilateral and quadrangle, and numerous other generalizations of value to the elementary geometry of to-day. But although Carnot's work was important and many details are now commonplace, neither the name of the theory nor the method employed have endured. The present *Geometry of Position* (*Geometrie der Lage*) has little in common with Carnot's *Géométrie de Position*.

Projective Geometry had its origin somewhat later than the period of Monge and Carnot. Newton had discovered that all curves of the third order can be derived by central projection from five fundamental types. But in spite of this fact the theory attracted so little attention for over a century that its origin is generally ascribed to Poncelet. A prisoner in the Russian campaign, confined at Saratoff on the Volga (1812-14), "privé," as he says, "de toute espèce de livres et de secours, surtout distract par les malheurs de ma patrie et les miens propres," he still had the vigor of spirit and the leisure to conceive the great work which he published (1822) eight years later. In this work was first made prominent the power of central projection in demonstration and the power of the principle of continuity in research. His leading idea was the study of projective properties, and as a foundation principle he introduced the anharmonic ratio, a concept, however, which dates back to Pappos and which Desargues (1639) had also used. Möbius, following Pon-

celet, made much use of the anharmonic ratio in his *Barycentrische Calcül* (1827), but under the name “Doppelschnitt-Verhältniss” (ratio bisectionalis), a term now in common use under Steiner’s abbreviated form “Doppelverhältniss.” The name “anharmonic ratio” or “function” (rapport anharmonique, or fonction anharmonique) is due to Chasles, and “cross-ratio” was coined by Clifford. The anharmonic point and line properties of conics have been further elaborated by Brianchon, Chasles, Steiner, and von Staudt. To Poncelet is also due the theory of “figures homologiques,” the perspective axis and perspective center (called by Chasles the axis and center of homology), an extension of Carnot’s theory of transversals, and the “cordes ideates” of conics which Plücker applied to curves of all orders. He also discovered what Salmon has called “the circular points at infinity,” thus completing and establishing the first great principle of modern geometry, the principle of continuity. Brianchon (1806), through his application of Desargues’s theory of polars, completed the foundation which Monge had begun for Poncelet’s (1829) theory of reciprocal polars.

Among the most prominent geometers contemporary with Poncelet was Gergonne, who with more propriety might be ranked as an analytic geometer. He first (1813) used the term “polar” in its modern geometric sense, although Servois (1811) had used the expression “pole.” He was also the first (1825- 26) to grasp the idea that the parallelism which Maurolycus, Snell, and Viète had noticed is a fundamental principle. This principle he stated and to it he gave the name which it now bears, the Principle of Duality, the most important, after that of continuity, in modern geometry. This principle of geometric reciprocation, the discovery of which was also claimed by Poncelet, has been greatly elaborated and has found its way into modern algebra and elementary geometry, and has recently been extended to mechanics by Genese. Gergonne was the first to use the word “class” in describing a curve, explicitly defining class and degree (order) and showing the duality between the two. He and Chasles were among the first to study scientifically surfaces of higher order.

Steiner (1832) gave the first complete discussion of the projective relations between rows, pencils, etc., and laid the foundation for the subsequent development of pure geometry. He practically closed the theory of conic sections, of the corresponding figures in three-dimensional space and of surfaces of the second order, and hence with him opens the period of special study of curves and surfaces of higher order. His treatment of duality and his application of the theory of projective pencils to the generation of conics are masterpieces. The theory of polars of a point in regard to a curve had been studied by Bobillier and by Grassmann, but Steiner (1848) showed that this theory can serve as the foundation for the study of plane curves independently of the use of coordinates, and introduced those noteworthy curves covariant to a given curve which now bear the names of himself, Hesse, and Cayley. This whole subject has been extended by Grassmann, Chasles, Cremona, and Jonquières. Steiner was the first to make prominent (1832) an example of correspondence of a more complicated nature than that of Poncelet, Möbius, Magnus, and Chasles. His contributions, and those of Gudermann, to the geometry of the sphere were also noteworthy.

While Möbius, Plucker, and Steiner were at work in Germany, Chasles was closing the geometric era opened in France by Monge. His *Aperçu Historique* (1837) is a classic, and

did for France what Salmon's works did for algebra and geometry in England, popularizing the researches of earlier writers and contributing both to the theory and the nomenclature of the subject. To him is due the name "homographic" and the complete exposition of the principle as applied to plane and solid figures, a subject which has received attention in England at the hands of Salmon, Townsend, and H. J. S. Smith.

Von Staudt began his labors after Plücker, Steiner, and Chasles had made their greatest contributions, but in spite of this seeming disadvantage he surpassed them all. Joining the Steiner school, as opposed to that of Plücker, he became the greatest exponent of pure synthetic geometry of modern times. He set forth (1847. 1856-60) a complete, pure geometric system in which metrical geometry finds no place. Projective properties foreign to measurements are established independently of number relations, number being drawn from geometry instead of conversely, and imaginary elements being systematically introduced from the geometric side. A projective geometry based on the group containing all the real projective and dualistic transformations, is developed, imaginary transformations being also introduced. Largely through his influence pure geometry again became a fruitful field. Since his time the distinction between the metrical and projective theories has been to a great extent obliterated, the metrical properties being considered as projective relations to a fundamental configuration, the circle at infinity common for all spheres. Unfortunately von Staudt wrote in an unattractive style, and to Reye is due much of the popularity which now attends the subject.

1.1.2 Elementary geometry

Trigonometry and Elementary Geometry have also been affected by the general mathematical spirit of the century. In trigonometry the general substitution of ratios for lines in the definitions of functions has simplified the treatment, and certain formulas have been improved and others added. The convergence of trigonometric series, the introduction of the Fourier series, and the free use of the imaginary have already been mentioned. The definition of the sine and cosine by series, and the systematic development of the theory on this basis, have been set forth by Cauchy (1821), Lobachevsky (1833), and others. The hyperbolic trigonometry, already founded by Mayer and Lambert, has been popularized and further developed by Gudermann (1830), Hoüel, and Laisant (1871), and projective formulas and generalized figures have been introduced, notably by Gudermann, Möbius, Poncelet, and Steiner. Recently Study has investigated the formulas of spherical trigonometry from the standpoint of the modern theory of functions and theory of groups, and Macfarlane has generalized the fundamental theorem of trigonometry for three-dimensional space.

Elementary Geometry has been even more affected. Among the many contributions to the theory may be mentioned the following: That of Möbius on the opposite senses of lines, angles, surfaces, and solids; the principle of duality as given by Gergonne and Poncelet; the contributions of De Morgan to the logic of the subject; the theory of transversals as worked out by Monge, Brianchon, Servois, Carnot, Chasles, and others; the theory of the radical axis, a property discovered by the Arabs, but introduced as a definite concept

by Gaultier (1813) and used by Steiner under the name of “line of equal power”; the researches of Gauss concerning inscriptible polygons, adding the 17- and 257-gon to the list below the 1000-gon ; the theory of stellar polyhedra as worked out by Cauchy, Jacobi, Bertrand, Cayley, Möbius, Wiener, Hess, Hersel, and others, so that a whole series of bodies have been added to the four Kepler-Poinsot regular solids; and the researches of Muir on stellar polygons. These and many other improvements now find more or less place in the text-books of the day.

To these must be added the recent Geometry of the Triangle, now a prominent chapter in elementary mathematics. Crelle (1816) made some investigations in this line, Feuerbach (1822) soon after discovered the properties of the Nine-Point Circle, and Steiner also came across some of the properties of the triangle, but none of these followed up the investigation. Lemoine (1873) was the first to take up the subject in a systematic way, and he has contributed extensively to its development. His theory of “transformation continue” and his “géométregraphie ”should also be mentioned. Brocard’s contributions to the geometry of the triangle began in 1877. Other prominent writers have been Tucker, Neuberg, Vigarié, Emmerich, M’Cay, Longchamps, and H. M. Taylor. The theory is also greatly indebted to Miller’s work in *The Educational Times*, and to Hoffmann’s *Zeitschrift*.

The study of linkages was opened by Peaucellier (1864), who gave the first theoretically exact method for drawing a straight line. Kempe and Sylvester have elaborated the subject.

In recent years the ancient problems of trisecting an angle, doubling the cube, and squaring the circle have all been settled by the proof of their insolubility through the use of compasses and straight edge.

1.1.3 Non-Euclidean geometry

The Non-Euclidean Geometry is a natural result of the futile attempts which had been made from the time of Proklos to the opening of the nineteenth century to prove the fifth postulate (also called the twelfth axiom, and sometimes the eleventh or thirteenth) of Euclid. The first scientific investigation of this part of the foundation of geometry was made by Saccheri (1733), a work which was not looked upon as a precursor of Lobachevsky, however, until Beltrami (1889) called attention to the fact. Lambert was the next to question the validity of Euclid’s postulate, in his *Theorie der Parallellinien* (posthumous, 1786), the most important of many treatises on the subject between the publication of Saccheri’s work and those of Lobachevsky and Bolyai. Legendre also worked in the field, but failed to bring himself to view the matter outside the Euclidean limitations.

During the closing years of the eighteenth century Kant’s doctrine of absolute space, and his assertion of the necessary postulates of geometry, were the object of much scrutiny and attack. At the same time Gauss was giving attention to the fifth postulate, though on the side of proving it. It was at one time surmised that Gauss’ was the real founder of the non-Euclidean geometry, his influence being exerted on Lobachevsky through his friend Bartels, and on Johann Bolyai through the father Wolfgang, who was a fellow student

of Gauss's. But it is now certain that Gauss can lay no claim to priority of discovery, although the influence of himself and of Kant, in a general way, must have had its effect.

Bartels went to Kasan in 1807, and Lobachevsky was his pupil. The latter's lecture notes show that Bartels never mentioned the subject of the fifth postulate to him, so that his investigations, begun even before 1823, were made on his own motion and his results were wholly original. Early in 1826 he sent forth the principles of his famous doctrine of parallels, based on the assumption that through a given point more than one line can be drawn which shall never meet a given line coplanar with it. The theory was published in full in 1829-30, and he contributed to the subject, as well as to other branches of mathematics, until his death.

Johann Bolyai received through his father, Wolfgang, some of the inspiration to original research which the latter had received from Gauss. When only twenty-one he discovered, at about the same time as Lobachevsky, the principles of non-Euclidean geometry, and refers to them in a letter of November, 1823. They were committed to writing in 1825 and published in 1832. Gauss asserts in his correspondence with Schumacher (1831-32) that he had brought out a theory along the same lines as Lobachevsky and Bolyai, but the publication of their works seems to have put an end to his investigations. Schweikart was also an independent discoverer of the non-Euclidean geometry, as his recently recovered letters show, but he never published anything on the subject, his work on the theory of parallels (1807), like that of his nephew Taurinus (1825), showing no trace of the Lobachevsky-Bolyai idea.

The hypothesis was slowly accepted by the mathematical world. Indeed it was about forty years after its publication that it began to attract any considerable attention. Hoüel (1866) and Flye St. Marie (1871) in France, Riemann (1868), Helmholtz (1868), Frischauf (1872), and Baltzer (1877) in Germany, Beltrami (1872) in Italy, de Tilly (1879) in Belgium, Clifford in England, and Halsted (1878) in America, have been among the most active in making the subject popular. Since 1880 the theory may be said to have become generally understood and accepted as legitimate.

Of all these contributions the most noteworthy from the scientific standpoint is that of Riemann. In his *Habilitationsschrift* (1854) he applied the methods of analytic geometry to the theory, and suggested a surface of negative curvature, which Beltrami calls "pseudo-spherical," thus leaving Euclid's geometry on a surface of zero curvature midway between his own and Lohachevsky's. He thus set forth three kinds of . geometry, Bolyai having noted only two. These Klein (1871) has called the elliptic (Riemann's), parabolic (Euclid's), and hyperbolic (Lobachevsky's).

Starting from this broader point of view there have contributed to the subject many of the leading mathematicians of the last quarter of a century, including, besides those already named, Cayley, Lie, Klein, Newcomb, Pasch, C. S. Peirce, Killing, Fiedler, Mansion, and McClintock. Cayley's contribution of his "metrical geometry" was not at once seen to be identical with that of Lobachevsky and Bolyai. It remained for Klein (1871) to show this, thus simplifying Cayley's treatment and adding one of the most important results of the entire theory. Cayley's metrical formulas are, when the Absolute is real, identical with those of the hyperbolic geometry; when it is imaginary, with the elliptic; the limiting

case between the two gives the parabolic (Euclidean) geometry. The question raised by Cayley's memoir as to how far projective geometry can be defined in terms of space without the introduction of distance had already been discussed by von Staudt (1857) and has since been treated by Klein (1873) and by Lindemann (1876).

1.1.4 Introduction of mathematical rigor

All the work related to the Parallel Postulate revealed that it was quite difficult for a geometer to separate his logical reasoning from his intuitive understanding of physical space, and, moreover, revealed the critical importance of doing so. Careful examination had uncovered some logical inadequacies in Euclid's reasoning, and some unstated geometric principles to which Euclid sometimes appealed. This critique paralleled the crisis occurring in calculus and analysis regarding the meaning of infinite processes such as convergence and continuity. In geometry, there was a clear need for a new set of axioms, which would be complete, and which in no way relied on pictures we draw or on our intuition of space. Such axioms were given by David Hilbert in 1894 in his dissertation *Grundlagen der Geometrie* (Foundations of Geometry). Some other complete sets of axioms had been given a few years earlier, but did not match Hilbert's in economy, elegance, and similarity to Euclid's axioms.

1.1.5 Algebraic geometry

Developments in algebraic geometry included the study of curves and surfaces over finite fields as demonstrated by the works of among others André Weil, Alexander Grothendieck, and Jean-Pierre Serre as well as over the real or complex numbers. Finite geometry itself, the study of spaces with only finitely many points, found applications in coding theory and cryptography.

“Expo-Rational B-splines” abbreviated to “ERBS” is a relatively new topic in the spline world. In what ways are ERBS different from polynomial B-splines, and how and why should we use them? These are the questions addressed in this book. The construction of ERBS and their properties and also the analysis of their properties is, therefore, one of the main topics of this book. Although ERBS can be used in computations, for instance as basis functions for finite element method (FEM), this book will concentrate on their use in CAGD, i.e. on curves, tensor product surfaces and triangle based surfaces. Arguments for, and examples of the practical use of ERBS, will be given. In addition we will get practical advice, and tools to implement Expo-Rational B-splines, and also practical advices in using them as tools in geometric modeling. To introduce us to ERBS we will first look at three examples.

Perhaps the most promising possibility that ERBS offers is what is described in section 11.7, the surface approximation on triangulations, that which opens 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. An example is given in Figure 1.2. This example

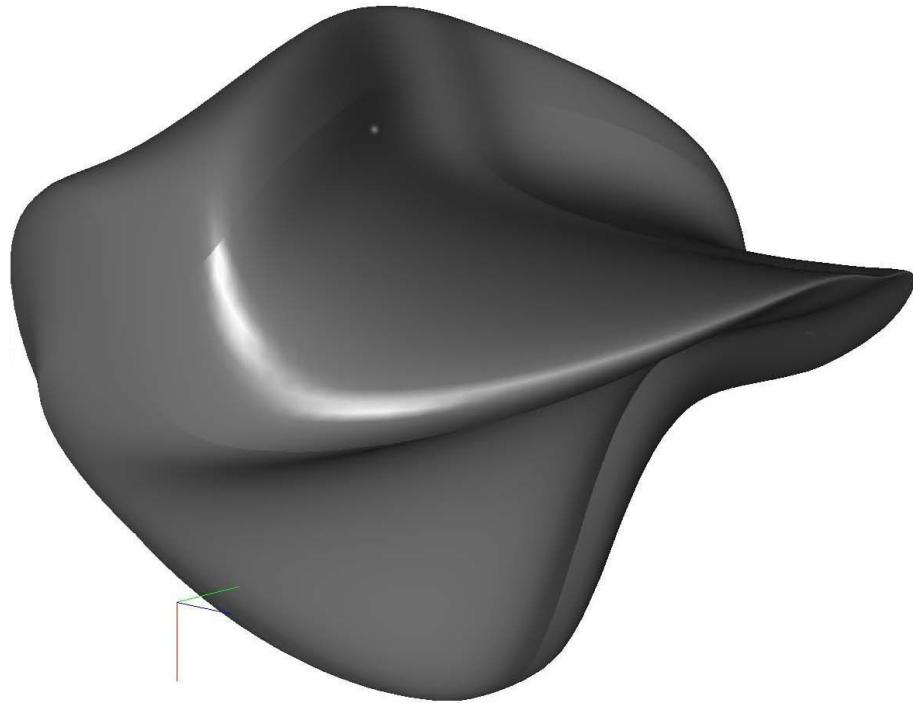


Figure 1.2: The surface is a connected set of eight triangular Expo-Rational B-spline surfaces. This surface is made by first approximating a sphere. Then the surface is deformed by an editing process.

shows a surface made by a connected set of 8 triangular surfaces, the properties and the construction of these triangular surfaces are further described in section 11.7. The surface is first made as an approximation of a sphere, and then changed by an editing process to the present geometric shape.

Another new and promising feature is “the affine transformation of local functions”, described in sections 8.5 and 10.4 and several other places. This opens 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 10.4. An example of the “geometric editing” is given in Figure 1.3. The surface is a tensor product Expo-Rational B-spline surface, made by first interpolating a torus. Then the surface is changed by an editing process to its present geometric shape. The blue cubes, which can be seen in the figure, are representing local surfaces and can be regarded as the editing points which are used in the “geometric editing” process. this technic is further described in section ?? and the implementation of the editing points is briefly discussed in section 10.4.

A third feature that is worth mentioning is the strong Hermite interpolation property, described in section 7.4 and in several examples throughout. This makes Expo-Rational B-splines very easy to use in modeling. An example is given in Figure 1.4. The surface in the figure is Hermite interpolating a surface called “Trianguloid Trefoil” (see [3]) at 5×5 points. At each point the position and a total of 8 partial derivatives are used.

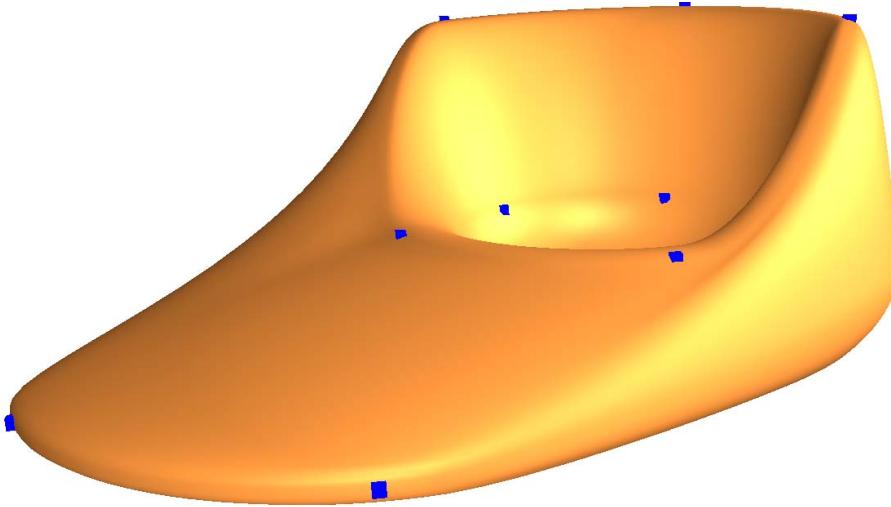


Figure 1.3: This tensor product Expo-Rational B-spline surface is made by first Hermite-interpolating a torus at 5×5 points. The positions of the interpolating points are seen as cubes. The surface is then edited by moving the blue cubes, so the surface finally gets its present shape.

Obviously, this book will not be a complete summary of Expo-Rational B-splines, it is only an introduction to these C^∞ -smooth B-splines. It is not an exaggeration to say that there are a lot of possibilities for the further development of this new type of splines, and there are a lot of “loose ends” in this book, and several of them are mentioned in the different chapters.

The name “Expo-Rational B-splines” refers to the derivative of the Expo-Rational B-spline basis function in the construction (equation 7.8), 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 will also find in linear polynomial B-splines. These will be clearly demonstrated later in this book.

The remaining part of this introduction is comprised of; a review of the development of Expo-Rational B-splines, a description of the challenges in implementations, a description of the algorithmic language used in this book and there will be a short review of the benefit of using Expo-Rational B-splines in possible new applications. Finally there will be a short overview of the rest of this book.

1.1.6 New notes

In this book you will find several places with historical remarks. Interpolation is the historical backbone to geometric modeling and is described in section 5.1. History of B-splines is discussed in section 5.4.

For Computer Aided Geometric Design, an important event was a conference held at

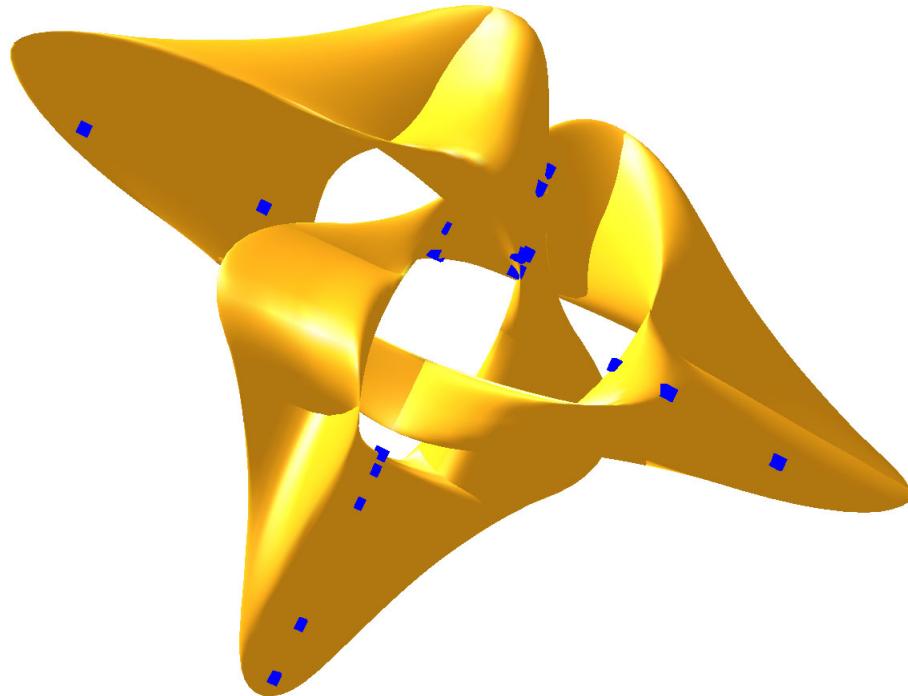


Figure 1.4: This tensor product Expo-Rational B-spline surface is made by Hermite interpolating a “Trianguloid Trefoil” (see [3]) at 5×5 points. The positions of the interpolating points are seen as blue cubes.

the University of Utah in 1974 organized by Barnhill and Riesenfeld [5]. Before this conference, we can say that the field is in its embryonic stage, after this conference, we can say that CAGD (Computer Aided Geometric Design) is born as a discipline.

Today CAGD is a mature discipline, where B-spline and spline methods are industrial standards and implemented in thousand of applications. However, the field is even a more exciting field than ever, because the importance of the field is expanding since the introduction of virtual worlds, computer games, animated movies, social worlds etc.

In this book we also introduce a new generation of curve/surface construction based on blending technics. Historical is this not new technics. Both Coons Patch [25] and different types of fillet constructions are using blending techniques, and Circle splines [148] is very close to the construction presented in different articles about what is called GERBS (Generalized Expo-rational B-splines), see [95, 43, 44, 89, 40, 91, 92].

The development of ERBS, Expo-Rational B-splines, and its generalization has been one of the main working areas of the R&D group for Mathematical Modeling, Numerical Simulation and Computer Visualization (shortly the Simulations R&D group) at Narvik University College. The research topic began in the summer of 2003.

The first presentations were given at the 6th International Conference on Mathematical Methods for Curves and Surfaces in Tromsø, on 5th July 2004. The first published article was in the proceedings of the conference [95]. It was followed by [43], [44] and [89],

The R&D group for Mathematical Modeling, Numerical Simulation and Computer Visualization at Narvik University College collaborates closely with “SINTEF Applied Mathematics” in Oslo, therefor, Tor Dokken was also a part-time member of the “Simulations” 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 Geometric modeling

The term geometric modeling first came into use during the 1960s, a time of rapidly developing computer graphics and computer aided design and manufacturing technologies. The discipline of geometric modeling is an interrelated, although somewhat loosely integrated, collection of mathematical methods that we use to describe the shape of an object or to express some physical process in terms of an appropriate geometric metaphor. These methods include computer aided geometric design, solid modeling, algebraic geometry, and computational geometry. Computer aided geometric design (CAGD) applies the mathematics of curves and surfaces to modeling, primarily using the parametric equations of differential geometry, supported by interpolation and approximation theory. It is here that we find the roots of contemporary geometric modeling. Solid modeling, usually encountered as constructive solid geometry (CSG), allows us to combine simple shapes to create complex solid models. CSG has its mathematical foundations in topology, algebraic geometry, and boolean algebra. Algebraic geometry is the contemporary extension of classical analytic geometry, including differential geometry. Computational geometry is concerned with design and analysis of geometric algorithms, and has strong ties to numerical methods, computation theory and complexity analysis. Computer-aided geometric design and constructive solid geometry are branches of geometric modeling, whereas algebraic and computational geometry tend to fields as well.

The shapes studied in geometric modeling are mostly two- or three-dimensional, although many of its tools and principles can be applied to sets of any finite dimension. Today geometric modeling is done with computers and for computer-based applications. Three-dimensional models are central to computer-aided design and manufacturing (CAD/CAM), and widely used in many applied technical fields such as civil and mechanical engineering, architecture, geology and medical image processing. A big new field is what we can call virtual world systems, computer games, VR systems like "Second life" and simulators.

The field Computer Aided Geometric Design can be of interest for a lot of people, among them researchers, scholars, designers and software developers dealing with mathematical and computational methods for the description of geometric objects as they arise in areas ranging from CAD/CAM, robotics, simulations and visualization. The primary objects of interest are curves, surfaces, and volumes such as splines (NURBS), meshes, subdivision surfaces as well as algorithms to generate, analyze, and manipulate them. This book will give an overview of the field and look on new developments in CAGD and its applications, including but not restricted to the following:

The aim is to collect and disseminate information on computer aided design. To provide the user community with methods and algorithms for representing curves and surfaces. To illustrate computer aided geometric design by means of interesting applications. To combine curve and surface methods with computer graphics. To explain scientific phenomena by means of computer graphics. To concentrate on the interaction between theory and application. To expose unsolved problems of the practice. To develop new methods in computer aided geometry.

Geometric modeling underlies applications from computer animation and special effects, to advanced modeling software for industrial design and architecture, to rapid prototyping machines that "print" 3-D objects in plastic, and many others. Geometric models represent the shapes and spatial relationships of the environment that is being studied, permitting a much deeper analysis than would be possible otherwise. How these models are encoded, and how the algorithms that utilize them are designed, comprise the field of computer-aided geometric design (CAGD), which is the subject of this course.

We will discuss a wide variety of techniques for representing and analyzing these models. Our emphasis will be on parametric curves and surfaces and on subdivision methods, but we will discuss less general approaches as well, particularly in relation to online and gaming 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 ?? 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^∞ -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 potential for creating triangular based 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 8, 10 and 11.

“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 [54]).

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 8.23 and 8.24, and in section ??.

1.3 Algorithmic language

By “algorithmic language” we mean the language used to describe the algorithms that are developed in this book. 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<T> 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 (7.42). The C++ code is first shown,

```

1 double f2( double t )
2 {
3     if( t < 2.3e-308 || t == lambda || t == 1.0)    return 0.0;
4

```

```

5   double h = (t - 1/(1+gamma))*abs(t-lambda)/(t*(1-t));
6   if(t<lambda)      h -= 1.0;
7   else              h += 1.0;
8   h *= -_sk*alpha*beta*(1+gamma)/pow(t*pow(1-t, gamma), alpha);
9   if((1+gamma)*alpha < 1)
10  {
11    double g = pow(abs(t-lambda), 1-(1+gamma)*alpha)/h;
12    if(g < 2.3e-308)    return 0.0;
13    else                  return 1/g;
14  }
15  else if ((1+gamma)*alpha > 1)
16    return h * pow(abs(t-lambda), (1+gamma)*alpha-1);
17  else
18    return h;
19 }
```

Then the description by the algorithmic language.

```

double f2 ( double t )
  if( t < 2.3e-308 || t == λ || t == 1 ) // See (7.42), upper part.
    return 0.0;
  double h =  $\frac{t-\frac{1}{1+\gamma}}{t(1-t)} |t - \lambda|$ ; // First part of the second factor from (A.3).
  if( t < λ )  h -= 1; // Last part of second factor (A.3).
  else          h += 1;
  h *=  $\frac{-S_k\alpha\beta(\gamma+1)}{(t(1-t)^\gamma)^\alpha}$ ; // Inserting  $S_k$  and the first factor of (A.3).
  if( (1+γ)α < 1 ) // Asymptote at  $t = \lambda$  is present.
    double g =  $\frac{|t-\lambda|^{1-(1+\gamma)\alpha}}{h}$ ; // The inverse of (A.3).
    if( g ≠ normal value ) return 0;
    else                  return  $\frac{1}{g}$ ;
  else if( (1+γ)α > 1 ) // Ordinary solution.
    return h |t - λ|(1+γ)α-1;
  else // Discontinuity at  $t = \lambda$ .
    return h;
```

This example can be found in section A.2, algorithm 10. The C++ code in this example can be found in the evaluator class used in our test program (see section ??). The function is a member function of this evaluator class, the variables; $_sk$, α , β , γ and λ 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 book, make the algorithm easier to understand. It is actually only a formalization of the text in this book.

1.4 Notations

The notation in this book is following standard commonly used. Because geometry is in most used embedded in the plane or in the 3-space the functions are commonly vector valued or point valued. A list explaining notations is:

- ✓ Euclidian spaces denotes \mathbb{R}^d , where d is the degree of the space, \mathbb{R}^2 is the plane and \mathbb{R}^3 is the 3D space.
- ✓ Cartesian coordinate system, is typical used for Euclidean spaces. It specifies each point uniquely in an Euclidian space by a set of numerical coordinates, which are the signed distances from the point to fixed perpendicular directed lines, measured in the same unit of length, i.e. $p = (x, y, z)$.
- ✓ Each reference line of a Cartesian coordinate system is called a coordinate axis or just axis of the system, and the point where they meet is its origin, $O = (0, 0, 0)$. The coordinates can also be defined as the positions of the perpendicular projections of the point onto the two axes, expressed as a signed distances from the origin.
- ✓ A vector or point is notated with a letter, Latin or Greek.

$$\mathbf{r} = (r_1, r_2, r_3) = \begin{pmatrix} r_1 \\ r_2 \\ r_3 \end{pmatrix} \in \mathbb{R}^3.$$

- ✓ An inner product between two vectors r and $s \in \mathbb{R}^d$, $d > 1$ is denoted

$$\langle r, s \rangle = (r_1 s_1 + r_2 s_2 + \dots + r_d s_d) \in \mathbb{R}.$$

An inner product between two vectors is the same as the scalar product.

- ✓ A vector product in \mathbb{R}^3 , also called wedge product, is; given the vectors $r = (r_1, r_2, r_3)$ and $s = (s_1, s_2, s_3)$. A vector (wedge) product is then

$$r \wedge s = \left(\begin{vmatrix} r_2 & r_3 \\ s_2 & s_3 \end{vmatrix}, \begin{vmatrix} r_1 & r_3 \\ s_1 & s_3 \end{vmatrix}, \begin{vmatrix} r_1 & r_2 \\ s_1 & s_2 \end{vmatrix} \right) = (r_2 s_3 - s_2 r_3, r_1 s_3 - s_1 r_3, r_1 s_2 - s_1 r_2)$$

It can be shown that the vector product is orthogonal to both of its vectors, i.e.

$$\langle r \wedge s, r \rangle = \langle r \wedge s, s \rangle = 0.$$

- ✓ A wedge product in \mathbb{R}^2 is, given the vectors $r = (r_1, r_2)$ and $s = (s_1, s_2)$. A wedge product is

$$r \wedge s = \begin{vmatrix} r_1 & r_2 \\ s_1 & s_2 \end{vmatrix} = r_1 s_2 - s_1 r_2$$

A wedge product in \mathbb{R}^2 is the same as the inner product

$$\langle r, s^L \rangle = r \wedge s,$$

where the vector s^L is vector s rotated 90° counter clockwise.

- ✓

1.5 Overview of this book

There are a total of 8 chapters in this book, their purpose is to introduce the Expo-Rational B-spline and to show how these new splines can be used in geometric modeling, design and simulation. Here is a short summary of the remaining chapters.

Chapter ?? gives some preliminary facts, a historical overview of interpolation, Hermite interpolation, cubic spline interpolation, a history of B-splines, modern B-splines, blending of geometry and charts and atlas. These topics are given because they are essential for ERBS, and can be a help for the reader to understand ERBS.

Chapter ?? 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, are also introduced.

Chapter A 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 as regards to the IEEE standard for binary floating-point arithmetic. It also introduces algorithms for the 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 8. 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 10 introduces tensor product ERBS surfaces. Both definitions and aspects concerning implementation 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 11 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. Chapters 10 and 11 contain many examples.

The last chapter, ??, gives a brief overview of work done on ERBS that is not covered by the seven previous chapters. Three of these subjects are briefly described in the three following sections in the chapter, “Three-variate tensor product ERBS”, “NUERBS Form of Expo-Rational B-splines” and “Beta-function B-splines, a polynomial analogy to ERBS”. Two of these are Master Theses done by students at Narvik University College (NUC). The third one is taken from an article based on work done by people at the R&D “Simu-

lations” group at NUC.

```

1 /*
2  ****
3 ** Copyright (C) 1994 Narvik University College
4 ** Contact: GMlib Online Portal at http://episteme.hin.no
5 **
6 ** This file is part of the Geometric Modeling Library, GMlib.
7 **
8 ** GMlib is free software: you can redistribute it and/or modify
9 ** it under the terms of the GNU Lesser General Public License as
10 ** published by
11 ** the Free Software Foundation, either version 3 of the License, or
12 ** (at your option) any later version.
13 **
14 ** GMlib is distributed in the hope that it will be useful,
15 ** but WITHOUT ANY WARRANTY; without even the implied warranty of
16 ** MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
17 ** GNU Lesser General Public License for more details.
18 **
19 ** You should have received a copy of the GNU Lesser General Public
20 ** License
21 ** along with GMlib. If not, see <http://www.gnu.org/licenses/>.
22 **
23 **
24 */
25 /*! \file gmpoint.h
26 *
27 * Interface classes for the Point/Vector/UnitVector/Arrow/ScalarPoint
28 * /Sphere/Box classes
29 */
30 #ifndef __gmPOINT_H__
31 #define __gmPOINT_H__
32
33
34 // **** GMlib ****
35 #include "gmangle.h"
36 #include "../static/gmstaticproc.h"
37 #include "../utils/gmstream.h"
38 #include "../containers/gmarray.h"
39
40
41 namespace GMlib {
42
43
44     template <typename T, int n>
45     class Point;
46
47     template <typename T, int n>
```

```
48 class Vector;
49
50 template <typename T, int n>
51 class UnitVector;
52
53 template <typename T, int n>
54 class Arrow;
55
56 template <typename T, int n>
57 class ScalarPoint;
58
59 template <typename T, int n>
60 class Sphere;
61
62 template <typename T, int n>
63 class Box;
64
65 template <typename T, int n>
66 class PlaneArrow;
67
68
69
70 // For following two classes are for internal use only
71 // They are either abstract base classes or help classes
72
73 template <typename T, int n>
74 class APoint;
75
76 template <typename T, int n, int m>
77 class M_I_;
78
79
80
81
82 /*! \class APoint gmpoint.h <gmapnt>
83 * \brief The Static APoint Class
84 *
85 * This class is just a placeholder, not to be used !!!!
86 *
87 * A template APoint, the APoint is static i.e. the dimentions
88 * can not be change. The template type must be clean, i.e. is not
89 * allocating
90 * memory and without any virtual functions
91 *
92 * The point is only using n values, but it is acting as it is in
93 * homogenous
94 * coordinates. The (n+1).nt element, which is not there, is
95 * actually implicite 1
96 */
97
98 template <typename T, int n>
99 class APoint {
  public:
    APoint();
    APoint( T t );
```

```

100    APoint( const T *t );
101    APoint( const APoint<T, n> &p );
102
103    Angle           getAngle(APoint<T, n> p) const;
104    T               getLength() const;
105    int              getMaxIndex() const;
106    int              getMaxAbsIndex() const;
107    T*
108    void             setTestType( int t, const APoint<T, n>& p,
109                                const Vector<T, n>& v = T
110                                (0) );
111
112    APoint<T, n>&   operator = ( const T t );
113    APoint<T, n>&   operator = ( const T *t );
114    APoint<T, n>&   operator = ( const APoint<T, n> &p );
115    APoint<T, n>&   operator += ( const APoint<T, n> &p );
116    APoint<T, n>&   operator -= ( const APoint<T, n> &p );
117    APoint<T, n>&   operator - () const;
118    APoint<T, n>&   operator + ( const APoint<T, n> &p ) const;
119    T               operator - ( const APoint<T, n> &p ) const;
120    T&
121    T const&        operator [] ( int i );
122
123 // Scaling
124 APoint<T, n>&   operator *= ( double d );
125 APoint<T, n>      operator * ( double d ) const;
126 APoint<T, n>&   operator %=( const APoint<T, n> &p );
127 APoint<T, n>&   operator % ( const APoint<T, n> &p ) const;
128
129 // Scaling: inverse
130 APoint<T, n>&   operator /= ( double d );
131 APoint<T, n>      operator / ( double d ) const;
132
133 // Boolean on equality
134 bool              operator == ( const APoint<T, n> &p ) const;
135 bool              operator != ( const APoint<T, n> &p ) const;
136
137 // Boolean on sorting
138 bool              operator < ( const APoint<T, n> &v ) const;
139 bool              operator > ( const APoint<T, n> &v ) const;
140 bool              operator <= ( const APoint<T, n> &v ) const;
141 bool              operator >= ( const APoint<T, n> &v ) const;
142
143 // Casting
144 template <typename G, int m>
145 operator APoint<G,m>& () const;
146
147 template <typename G, int m>
148 APoint<G,m>&      to() const;
149
150 template <typename G>
151 APoint<G,n>&      toType() const;
152
153

```

```
154
155     protected:
156         T           -pt[n];
157
158         void        _cpy( const APoint<T, n> &v );
159         void        _cpy( const T p[n] );
160         void        _cpy( const T &d );
161
162     private:
163
164         static Arrow<T, n>*   _arrow;      // Used for < and sorting see
165             setTestType();
166         static int            _sortType;
167
168     }; // END class APoint
169
170
171
172 // ****
173 // **      The Point class      **
174 // ****
175
176
177 /*! \class Point gmpoint.h <gmPoint>
178 * \brief The Static Point Class
179 *
180 * A template Point , the Point is static i.e. the dimentions
181 * can not be change. The template type must be clean , i.e. is not
182 * allocating
183 * memory and without any virtual functions
184 *
185 * The point is only using n values , but it is acting as it is in
186 * homogenous
187 * coordinates. The (n+1).nt element , which is not there , is
188 * actually implicite 1
189 */
190 template <typename T, int n>
191 class Point: public APoint<T,n> {
192     public:
193         Point():APoint<T,n>(){}
194         Point( T t ):APoint<T,n>(t){}
195         Point( const T *t ):APoint<T,n>(t){}
196         Point( const APoint<T, n> &p ):APoint<T,n>(p){}
197
198     template <typename T>
199     class Point<T,2>: public APoint<T,2> {
200         public:
201             Point<T,2>():APoint<T,2>(){}
202             Point<T,2>( T t ):APoint<T,2>(t){}
203             Point<T,2>( const T *t ):APoint<T,2>(t){}
204             Point<T,2>( const APoint<T,2> &p ):APoint<T,2>(p){}
```

```

205     Point<T,2>( const APoint<T,3> &p );
206     Point<T,2>( const T& x, const T& y );
207
208     APoint<T,2> getNormal(); // Return a vector 90 deg. to this.
209     int isInside( const APoint<T,2>& p1, const APoint<T,2>& p2,
210                   const APoint<T,2>& p3 ) const;
211     int isInside( const Array<Point<T,2>>& a ) const;
212     int isInsideCircle( const APoint<T,2>& p1, const APoint<T,
213                         ,2>& p2, const APoint<T,2>& p3 ) const;
214
215     T operator^( const APoint<T,2>& v ) const; // wedge
216     product.
217 };
218
219 template <typename T>
220 class Point<T,3>: public APoint<T,3> {
221 public:
222     Point<T,3>(): APoint<T,3>(){}
223     Point<T,3>( T t ): APoint<T,3>(t){}
224     Point<T,3>( const T *t ): APoint<T,3>(t){}
225     Point<T,3>( const APoint<T, 3> &p ): APoint<T,3>(p){}
226
227     Point<T,3>( const APoint<T,2> &p );
228     Point<T,3>( const T& x, const T& y, const T& z );
229
230     APoint<T,3> operator^( const APoint<T,3>& v ) const; // vector
231     product.
232 };
233 // END class Point
234
235
236 /*! \var static Arrow<T, n> *APoint<T, n>::_arrow
237  * \brief Used for < and sorting. See setTestType()
238  */
239
240 /*! \var static int _type; */
241
242 /*! APoint<T, n>& operator * ( double d, const APoint<T, n> &p )
243  * \brief Multiply a double with a APoint<T, n>
244  *
245  * Multiply a double with a APoint<T, n>.
246  * This function overloads the * operator of the double.
247  *
248  * \param[in] d The double
249  * \param[in] p The APoint<T,n>
250  */
251 template <typename T, int n>
252 inline
253 APoint<T, n> operator * ( double d, const APoint<T, n> &p ) {
254     return p*d;
255 }
```

```

256
257
258 /* ! bool Point<T,2>::convexHullFrom( Array<Point<T,2> >& a, const
259     Vector<T,2>& v)
260     * \brief Compute the convex hull of a point set
261     *
262     * This function compute the convex hull of a point
263     * set stored in the array a.
264     * The convex hull is also stored in the array a,
265     * and it is a counter clockwise oriented polygon.
266     * If the convex hull consist off all original points
267     * from a then false is retuned else true is returned.
268     */
269 template <typename T>
270 bool convexHullFrom( Array<Point<T,2> >& a, const Vector<T,2>& v) {
271
272     bool removed = false;
273     if (a.size() < 4) return removed;
274
275     int i,j,k;
276     Point<T,2> p;
277
278     for (i=0; i < a.size(); i++) p += a[i];
279
280     p.setTestType(3,p/a.size(),v);
281     a.sort();
282
283     for (i=0; i < a.size(); i++) {
284         if (i < a.size() - 2) { j=i+1; k=i+2; }
285         else if (i < a.size() - 1) { j=i+1; k=0; }
286         else { j=0; k=1; }
287         p = a[j] - a[i];
288         if ((p^(a[k]-a[j])) < 0) {
289             a.removeIndex(j);
290             removed = true;
291             if (i == a.size()) i -= 3;
292             else if (i > 0) i -= 2;
293             else i--;
294         }
295     }
296     return removed;
297
298
299
300
301 #ifdef GM_STREAM
302 /* ! T_Stream &operator << ( T_Stream &out, const APoint<T, n> &p )
303     * \brief Stream output operator
304     *
305     * Stream output operator, taking a APoint<T,n> as a second
306     * parameter.
307     *
308     * \param out The output stream
309     * \param p The APoint<T, n>

```

```
309     * \return The output stream
310     */
311     template <typename T_Stream, typename T, int n>
312     inline
313     T_Stream& operator << ( T_Stream &out, const APoint<T, n> &p ) {
314         for( int i = 0; i < n; i++ )
315             out << p(i) << GMseparator::Element;
316         return out;
317     }
318
319
320 /* ! T_Stream& operator << ( T_Stream &out, const APoint<T, n> *p )
321 * \brief Stream output operator
322 *
323 * Stream output operator, taking a APoint<T,n> pointer as a
324 * second parameter.
325 *
326 * \param[in, out] out The output stream
327 * \param[in] p The APoint<T, n> pointer
328 * \return The output stream
329 */
330     template <typename T_Stream, typename T, int n>
331     inline
332     T_Stream& operator << ( T_Stream &out, const APoint<T, n> *p ) {
333         for( int i=0;i<n;i++) out << (*p)(i) << GMseparator::Element;
334         return out;
335     }
336
337 /* ! T_Stream& operator >> ( T_Stream &in, APoint<T, n> &p )
338 * \brief Stream input operator
339 *
340 * Stream input operator, taking a APoint<T,n> as a second
341 * parameter.
342 *
343 * \param[in, out] in The input stream
344 * \param[in] p The APoint<T, n>
345 * \return The input stream
346 */
347     template <typename T_Stream, typename T, int n>
348     inline
349     T_Stream& operator >> ( T_Stream &in, APoint<T, n> &p ) {
350         Separator es(GMseparator::Element);
351         for( int i=0;i<n;i++) in >> p[i] >> es;
352         return in;
353     }
354
355 /* ! T_Stream& operator >> ( T_Stream &in, APoint<T, n> *p )
356 * \brief Stream input operator
357 *
358 * Stream input operator, taking a APoint<T,n> pointer as a second
359 * parameter.
360 *
361 * \param[in, out] in The input stream
362 * \param[in] p The APoint<T, n> pointer
```

```

361     * \return The input stream
362     */
363     template <typename T_Stream, typename T, int n>
364     inline
365     T_Stream& operator >> ( T_Stream &in , APoint<T, n> *p ) {
366         Separator es(GMseparator::Element);
367         for( int i=0;i<n; i++ ) in >> (*p)[ i ] >> es ;
368         return in ;
369     }
370 #endif
371
372
373
374 // ****
375 // **      The Vector class      **
376 // ****
377
378 /*! \class Vector gmpoint.h <gmPoint>
379  * \brief The Static Vector Class
380  *
381  * A template Vector, the Vector is static i.e. the dimentions
382  * can not be change. The template type must be clean, i.e. is not
383  * allocating
384  * memory and without any virtual functions.
385  *
386  * The vector is only using n values, but it is acting as it is in
387  * homogenous
388  * coordinates. The (n+1).nt element, which is not there, is
389  * actually implicite 0.
390  */
391 template <typename T, int n>
392 class Vector : public APoint<T,n> {
393 public:
394     Vector() : APoint<T,n>(){}
395     Vector( T t ) : APoint<T,n>(t){}
396     Vector( const T *t ) : APoint<T,n>(t){}
397     Vector( const APoint<T,n> &p ) : APoint<T,n>(p){}
398
399     APoint<T,n>    getNormalized() const;
400     Vector<T,n>    getLinIndVec() const;
401     APoint<T,n>&   normalize();
402     void            setLength( T length );
403 };
404
405 template <typename T>
406 class Vector<T,2> : public APoint<T,2> {
407 public:
408     Vector<T,2>() : APoint<T,2>(){}
409     Vector<T,2>( T t ) : APoint<T,2>(t){}
410     Vector<T,2>( const T *t ) : APoint<T,2>(t){}
411     Vector<T,2>( const APoint<T,2> &p ) : APoint<T,2>(p){}
412     Vector<T,2>( const APoint<T,3> &p );

```

```

413     Vector<T,2>( const APoint<T,4> &p );
414     Vector<T,2>( const T& x, const T& y );
415
416     APoint<T,2>      getNormalized() const;
417     Vector<T,2>      getLinIndVec() const;
418     APoint<T,2>&    normalize();
419     void              setLength( T length );
420
421     APoint<T,2>      getNormal() const; // Return a vector 90 deg. to
422                           this.
423
424     T                 operator^( const APoint<T,2>& v ) const; // wedge
425                           product.
426
427 template <typename T>
428 class Vector<T,3> : public APoint<T,3> {
429 public:
430     Vector<T,3>(): APoint<T,3>(){}
431     Vector<T,3>( T t ): APoint<T,3>(t){}
432     Vector<T,3>( const T *t ): APoint<T,3>(t){}
433     Vector<T,3>( const APoint<T,3> &p ): APoint<T,3>(p){}
434
435     Vector<T,3>( const APoint<T,2> &p );
436     Vector<T,3>( const APoint<T,4> &p );
437     Vector<T,3>( const T& x, const T& y, const T& z );
438
439     APoint<T,3>      getNormalized() const;
440     Vector<T,3>      getLinIndVec() const;
441     APoint<T,3>&    normalize();
442     void              setLength( T length );
443
444     APoint<T,3>      operator^( const APoint<T,3>& v ) const; // vector
445                           product.
446
447
448 template <typename T>
449 class Vector<T,4> : public APoint<T,4> {
450 public:
451     Vector<T,4>(): APoint<T,4>(){}
452     Vector<T,4>( T t ): APoint<T,4>(t){}
453     Vector<T,4>( const T *t ): APoint<T,4>(t){}
454     Vector<T,4>( const APoint<T,4> &p ): APoint<T,4>(p){}
455
456     Vector<T,4>( const APoint<T,2> &p );
457     Vector<T,4>( const APoint<T,3> &p );
458     Vector<T,4>( const T& x0, const T& x1, const T& x2, const T& x3 );
459
460     APoint<T,4>      getNormalized() const;
461     Vector<T,4>      getLinIndVec() const;
462     APoint<T,4>&    normalize();
463     void              setLength( T length );
464

```

```
465 // END class Vector
466
467
468
469
470 // *****
471 // Type Casting
472
473 /* ! Vector<T, n>& toVector( Point<T,n> &p )
474 * \brief Casts a Point<T,n> to a Vector<T,n>
475 *
476 * Casts a Point<T,n> to a Vector<T,n>
477 *
478 * \param[in] p The Point to be casted to Vector
479 * \return The casted vector
480 */
481 template <typename T, int n>
482 inline Vector<T, n>& toVector( Point<T,n> &p ) {
483     return static_cast<Vector<T,n>& >(p);
484 }
485
486 /* ! Point<T, n>& toPoint( Vector<T,n> &v )
487 * \brief Casts a Vector<T,n> to a Point<T,n>
488 *
489 * Casts a Vector<T,n> to a Point<T,n>
490 *
491 * \param[in] p The Vector to be casted to Point
492 * \return The casted point
493 */
494 template <typename T, int n>
495 Point<T, n>& toPoint( Vector<T,n> &v ) {
496     return *((Point<T,n>*)(&v));
497 }
498
499
500 // *****
501 // ** The UnitVector class **
502 // *****
503
504 /* ! \class UnitVector gmpoint.h <gmPoint>
505 * \brief The Static UnitVector class
506 *
507 * A template UnitVector, the UnitVector is static i.e. the
508 * dimentions
509 * can not be change. The template type must be clean, i.e. is not
510 * allocating
511 * memory and without any virtual functions.
512 *
513 * The UnitVector has, as the name indicate, always length 1. Be
514 * aware the you do not
515 * initiate it by a zero vector or by zero constant. In that case it
516 * will produce an overflow.
517 *
518 * The unit vector is only using n values, but it is acting as it is
519 * in homogenous
```

```

515     * coordinates. The (n+1).nt element (there is not there) is
516     actually implicite 0
517 */
518 template <typename T, int n>
519 class UnitVector : public Vector<T,n> {
520 public:
521     UnitVector( T t = 1 );
522     UnitVector( const T t[n] );
523     UnitVector( const APoint<T, n>& p );
524     UnitVector( const UnitVector<T,n>& uv );
525
526     APoint<T, n>& operator = ( const T t );
527     APoint<T, n>& operator = ( const T t[n] );
528     APoint<T, n>& operator = ( const APoint<T, n> &p );
529     APoint<T, n>& operator = ( const UnitVector<T, n>& uv );
530     const T& operator [] ( int i );
531     APoint<T, n>& operator += ( const APoint<T, n> &p );
532     APoint<T, n>& operator -= ( const APoint<T, n> &p );
533     APoint<T, n>& operator %= ( const APoint<T, n> &p );
534     APoint<T, n>& operator *= ( const double d );
535     APoint<T, n>& operator /= ( double d );
536 };
537 // END class UnitVector
538
539 template <typename T>
540 class UnitVector<T,2> : public Vector<T,2> {
541 public:
542     UnitVector<T,2>( T t = 1 );
543     UnitVector<T,2>( const T t[2] );
544     UnitVector<T,2>( const APoint<T,2>& p );
545     UnitVector<T,2>( const UnitVector<T,2>& uv );
546     UnitVector<T,2>( const T& x, const T& y );
547
548     APoint<T,2>& operator = ( const T t );
549     APoint<T,2>& operator = ( const T t[2] );
550     APoint<T,2>& operator = ( const APoint<T,2> &p );
551     APoint<T,2>& operator = ( const UnitVector<T,2>& uv );
552     const T& operator [] ( int i );
553     APoint<T,2>& operator += ( const APoint<T,2> &p );
554     APoint<T,2>& operator -= ( const APoint<T,2> &p );
555     APoint<T,2>& operator %= ( const APoint<T,2> &p );
556     APoint<T,2>& operator *= ( const double d );
557     APoint<T,2>& operator /= ( double d );
558 };
559 // END class UnitVector
560
561
562
563 template <typename T>
564 class UnitVector<T,3> : public Vector<T,3> {
565 public:
566     UnitVector<T,3>( T t = 1 );
567     UnitVector<T,3>( const T t[3] );
568     UnitVector<T,3>( const APoint<T,3>& p );

```

```

569     UnitVector<T,3>(& uv);
570     UnitVector<T,3>(& x, & y, & z);
571
572     APoint<T,3>& operator = (const T t);
573     APoint<T,3>& operator = (const T t[3]);
574     APoint<T,3>& operator = (const APoint<T,3> &p );
575     APoint<T,3>& operator = (const UnitVector<T,3>& uv );
576     const T& operator [] (int i);
577     APoint<T,3>& operator += (const APoint<T,3> &p );
578     APoint<T,3>& operator -= (const APoint<T,3> &p );
579     APoint<T,3>& operator %= (const APoint<T,3> &p );
580     APoint<T,3>& operator *= (const double d );
581     APoint<T,3>& operator /= (double d );
582
583 }; // END class UnitVector
584
585 #ifdef GM_STREAM
586 /*! T_Stream& operator >> ( T_Stream& in , UnitVector<T, n>& v )
587 * \brief Brief description
588 *
589 * Detailed Description
590 */
591 template <typename T_Stream, typename T, int n>
592 inline
593 T_Stream& operator >> ( T_Stream& in , UnitVector<T, n>& v ) {
594     Point<T, n> p;
595     in >> p;
596     v = p;
597     return in;
598 }
599
600 /*! T_Stream& operator >> ( T_Stream& in , UnitVector<T,n>* v )
601 * \brief Brief description
602 *
603 * Detailed Description
604 */
605 template <typename T_Stream, typename T, int n>
606 inline
607 T_Stream& operator >> ( T_Stream& in , UnitVector<T,n>* v ) {
608     Point<T,n> p;
609     in >> p;
610     (*v) = p;
611     return in;
612 }
613 #endif
614
615
616
617
618 // ****
619 // **      The Quaternion class      **
620 // ****
621
622 /*! \class Quaternion gmPoint.h <gmPoint>
```

```

624 * \brief The Static Quaternion class
625 */
626 template <typename T>
627 class Quaternion : public Vector<T,4> {
628 public:
629     Quaternion() : Vector<T,4>() {}
630     Quaternion( const T *t ):Vector<T,4>(t){}
631     Quaternion( const T& q0, const T& q1, const T& q2, const T& q3 ) :
632         Vector<T,4>(q0,q1,q2,q3) {}
633     Quaternion( const Quaternion& q ):Vector<T,4>(q){}
634     Quaternion( const APoint<T,4> &p ):Vector<T,4>(p){}
635 };
636
637
638
639 // ****
640 // **      The UnitQuaternion class      **
641 // ****
642
643
644 /*! \class UnitQuaternion gmpoint.h <gmPoint>
645 * \brief The Static UnitQuaternion class
646 */
647 template <typename T>
648 class UnitQuaternion : public Quaternion<T> {
649 public:
650     UnitQuaternion();
651     UnitQuaternion( const T t[4] );
652     UnitQuaternion( const T& q0, const T& q1, const T& q2, const T& q3 );
653     UnitQuaternion( const UnitQuaternion& uq );
654     UnitQuaternion( const APoint<T,4> &p );
655
656     APoint<T,4>& operator = ( const T t );
657     APoint<T,4>& operator = ( const T t[3] );
658     APoint<T,4>& operator = ( const APoint<T,3> &p );
659     APoint<T,4>& operator = ( const UnitQuaternion<T>& uv );
660     const T& operator [] ( int i );
661     APoint<T,4>& operator += ( const APoint<T,4> &p );
662     APoint<T,4>& operator -= ( const APoint<T,4> &p );
663     APoint<T,4>& operator %= ( const APoint<T,4> &p );
664     APoint<T,4>& operator *= ( const double d );
665     APoint<T,4>& operator /= ( double d );
666 };
667
668
669
670
671 // ****
672 // **      The Arrow class      **
673 // ****
674
675
676 /*! \class Arrow gmpoint.h <gmPoint>

```

```

677 * \brief The Static Arrow class
678 *
679 * A template Arrow, the Arrow is static i.e. the dimentions
680 * can not be change. The template type must be clean, i.e. is not
681 * allocating
682 * memory and without any virtual functions.
683 *
684 */
685 template <typename T, int n>
686 class Arrow : public Point<T,n> {
687 public:
688     Arrow();
689     Arrow(const APoint<T,n> &p);
690     Arrow(const APoint<T,n> &p, const Vector<T,n> &v );
691     Arrow(const Arrow<T,n> &a);
692
693     const Vector<T,n>& getDir() const;
694     const APoint<T,n>& getPos() const;
695     void setDir( const Vector<T,n> &v );
696     void setPos( const APoint<T,n> &p );
697
698     Arrow<T,n>& operator = ( const Arrow<T,n> &a );
699     Arrow<T,n>& operator += ( const Point<T,n> &p );
700     Arrow<T,n>& operator -= ( const Point<T,n> &p );
701     Arrow<T,n> operator + ( const Point<T,n> &p ) const;
702     Arrow<T,n> operator - ( const Point<T,n> &p ) const;
703     Arrow<T,n>& operator += ( const Vector<T,n> &v );
704     Arrow<T,n>& operator -= ( const Vector<T,n> &v );
705     Arrow<T,n> operator + ( const Vector<T,n> &v ) const;
706     Arrow<T,n> operator - ( const Vector<T,n> &v ) const;
707     Arrow<T,n> operator - () const;
708
709     template <typename G, int m>
710     operator Arrow<G, m>& () const;
711
712     Arrow<float ,n>& toFloat() const;
713     Arrow<double ,n>& toDouble() const;
714
715 protected:
716     Vector<T,n> _dir;
717
718 }; // END class Arrow
719
720
721
722 // ****
723 // IOSTREAM overloaded operators
724
725 #ifdef GM_STREAM
726 /* ! T_Stream &operator<<(T_Stream &out, const Arrow<T,n> &a)
727 * \brief Brief description
728 */

```

```
731 * Detailed Description
732 */
733 template <typename T_Stream, typename T, int n>
734 T_Stream &operator << ( T_Stream &out, const Arrow<T,n> &a ) {
735
736     out << a.getPos() << GMseparator::Element << a.getDir();
737     return out;
738 }
739
740
741 /* ! T_Stream& operator <<(T_Stream &out, const Arrow<T,n> *a)
742 * \brief Brief Description
743 *
744 * Detailed Description
745 */
746 template <typename T_Stream, typename T, int n>
747 T_Stream& operator << ( T_Stream &out, const Arrow<T,n> *a ) {
748
749     out << a->getPos() << GMseparator::Element << a->getDir();
750     return out;
751 }
752
753
754 /* ! T_Stream& operator >>(T_Stream &in, Arrow<T,n> &a)
755 * \brief Brief Description
756 *
757 * Detailed Description
758 */
759 template <typename T_Stream, typename T, int n>
760 T_Stream& operator >> ( T_Stream &in, Arrow<T,n> &a ) {
761
762     Separator es(GMseparator::Element);
763     Point<T,n> pt;
764     Vector<T,n> dir;
765
766     in >> pt >> es >> dir;
767     a.setPos(pt);
768     a.setDir(dir);
769
770     return in;
771 }
772
773
774 /* ! T_Stream& operator >>(T_Stream &in, Arrow<T,n> *a)
775 * \brief Brief Description
776 *
777 * Detailed Description
778 */
779 template <typename T_Stream, typename T, int n>
780 T_Stream& operator >> ( T_Stream &in, Arrow<T,n> *a ) {
781
782     Separator es(GMseparator::Element);
783     Point<T,n> pt;
784     Vector<T,n> dir;
```

```

786     in >> pt >> es >> dir;
787     a->setPos(pt);
788     a->setDir(dir);
789
790     return in;
791 }
792
793 #endif
794
795
796 // ****
797 // **      The ScalarPoint class      **
798 // ****
799
800 /*! \class ScalarPoint gmpoint.h <gmPoint>
801  * \brief The Static ScalarPoint class
802  * A template ScalarPoint, the ScalarPoint is static i.e. the
803  * dimentions
804  * can not be change. The template type must be clean, i.e. is not
805  * allocating
806  * memory and without any virtual functions.
807  *
808  * The ScalarPoint is a Point and a conected scalar value.
809 */
810 template <typename T, int n>
811 class ScalarPoint {
812 public:
813     ScalarPoint();
814     ScalarPoint( const APoint<T, n>& p, T v = T(0) );
815     ScalarPoint( const ScalarPoint<T, n>& s );
816
817     const APoint<T, n>&    getPos() const;
818     T*                      getPtr() const;
819     T                       getValue() const;
820     void                     reset( const APoint<T, n>& p, T v = T(0) );
821     void                     reset();
822     void                     resetValue( T t );
823     void                     resetPos( const APoint<T, n>& p );
824
825     ScalarPoint<T, n>&    operator += ( const APoint<T, n>& p );
826     ScalarPoint<T, n>        operator + ( const APoint<T, n>& p ) const;
827     ScalarPoint<T, n>&    operator += ( T p );
828     ScalarPoint<T, n>        operator + ( T p ) const;
829     ScalarPoint<T, n>&    operator += ( const ScalarPoint<T, n>& p );
830     ScalarPoint<T, n>        operator + ( const ScalarPoint<T, n>& p )
831                                         const;
832
833     ScalarPoint<T, n>&    operator *= ( double d );
834     ScalarPoint<T, n>        operator * ( double d ) const;
835
836     ScalarPoint<T, n>&    operator /= ( double d );
837     ScalarPoint<T, n>        operator / ( double d ) const;
838
839     ScalarPoint<T, n>&    operator %= ( const APoint<T, n>& p );
840     ScalarPoint<T, n>        operator % ( const APoint<T, n>& p ) const;

```

```

838 /* 
839     friend
840     ScalarPoint<T, n>      operator * ( double d, ScalarPoint<T, n> p )
841     { p*=d; return p; }
842 */
843 // Casting
844 template <typename G, int m>
845 operator ScalarPoint<G, m>& () const;
846
847 template <typename G, int m>
848 ScalarPoint<G,m>& to() const;
849
850 template <typename G>
851 ScalarPoint<G,n>& toType() const;
852
853
854 protected:
855     Point<T, n>           _pos;
856     T                      _value;
857
858 }; // END class ScalarPoint
859
860
861
862
863
864 #ifdef GM_STREAM
865
866 /* ! T_Stream &operator<<(T_Stream& os, const ScalarPoint<T,n>& s)
867  * \brief Brief Description
868  *
869  * Detailed Description
870  */
871 template <typename T_Stream, typename T, int n>
872 inline
873 T_Stream &operator<<(T_Stream& os, const ScalarPoint<T,n>& s) {
874     os << s .getPos() << GMseparator::Element << s .getValue();
875     return os;
876 }
877
878 /* ! T_Stream &operator<<(T_Stream& os, const ScalarPoint<T,n>* s)
879  * \brief Brief Description
880  *
881  * Detailed Description
882  */
883 template <typename T_Stream, typename T, int n>
884 inline
885 T_Stream &operator<<(T_Stream& os, const ScalarPoint<T,n>* s) {
886     os << s->getPos() << GMseparator::Element << s->getValue();
887     return os;
888 }
889
890 /* ! T_Stream &operator>>(T_Stream& is, ScalarPoint<T,n>& s)
891  * \brief Brief Description

```

```

892 *
893 *   Detailed Description
894 */
895 template <typename T_Stream, typename T, int n>
896 inline
897 T_Stream &operator >> ( T_Stream& is , ScalarPoint<T, n>& s ) {
898     Separator es(GMseparator::Element);
899     Point<T,n> p1;
900     T v;
901     is >> p1 >> es >> v;
902     s.reset(p1,v);
903     return is;
904 }
905
906 /* ! T_Stream &operator>>(T_Stream& is , ScalarPoint<T,n>* s)
907 *   \brief Brief Description
908 *
909 *   Detailed Description
910 */
911 template <typename T_Stream, typename T, int n>
912 inline
913 T_Stream &operator>>(T_Stream& is , ScalarPoint<T,n>* s) {
914     Separator es(GMseparator::Element);
915     Point<T,n> p1;
916     T v;
917     is >> p1 >> es >> v;
918     s->reset(p1,v);
919     return is;
920 }
921
922 #endif
923
924
925 // ****
926 // **      The Sphere class      **
927 // ****
928
929
930
931 /* ! \class Sphere gmpoint.h <gmPoint>
932 *   \brief The Static Sphere class
933 *   A template Sphere, the Sphere is static i.e. the dimentions
934 *   can not be change. The template type must be clean, i.e. is not
935 *   allocating
936 *   memory and without any virtual functions.
937 *
938 *   The Sphere is a centre Point and a conected scalar value, radius.
939 *   It also inclue a boolean telling if the Sphere is valid or not.
940 */
941 template <typename T, int n>
942 class Sphere : public ScalarPoint<T, n> {
943 public:
944     Sphere( bool sphere = false );
945     Sphere( const APoint<T, n>& p, T v = T(0) );
946     Sphere( const ScalarPoint<T, n>& s );

```

```

946     Sphere( const Sphere<T, n>& s );
947
948     T           getRadius() const;
949     bool        isValid() const;
950     bool        isIntersecting(const Sphere<T,n>& p) const;
951     void        resetPos( const APoint<T, n>& p );
952     void        resetRadius( T t );
953     void        reset();
954
955     Sphere<T, n>& operator += ( const APoint<T, n>& p );
956     Sphere<T, n>  operator + ( const APoint<T, n>& p ) const;
957     Sphere<T, n>& operator += ( const Sphere<T, n>& p );
958     Sphere<T, n>  operator + ( const Sphere<T, n>& p ) const;
959
960 // Casting
961 template <typename G, int m>
962 operator Sphere<G, m>& () const;
963
964 private:
965     bool _valid;
966
967 }; // END class Sphere
968
969
970
971
972
973
974
975
976
977
978 #ifdef GM_STREAM
979
980 /*! T_Stream &operator<<(T_Stream& os, const Sphere<T,n>& s)
981 * \brief Brief Description
982 *
983 * Detailed Description
984 */
985 template <typename T_Stream, typename T, int n>
986 inline
987 T_Stream &operator<<(T_Stream& os, const Sphere<T,n>& s) {
988     if(s.valid())
989         os << ScalarPoint<T,n>::s;
990     else
991         os << ScalarPoint<T,n>(Point<T,n>(),T(-1));
992     return os;
993 }
994
995 /*! T_Stream &operator<<(T_Stream& os, const Sphere<T,n>* s)
996 * \brief Brief Description
997 *
998 * Detailed Description
999 */
1000 template <typename T_Stream, typename T, int n>

```

```

1001     inline
1002     T_Stream &operator<<(T_Stream& os , const Sphere<T,n>* s) {
1003         if(s.isValid())
1004             os << reinterpret_cast< ScalarPoint<T,n> >(*s);
1005         else
1006             os << ScalarPoint<T,n>(Point<T,n>(),T(-1));
1007
1008         return os;
1009     }
1010
1011 /* ! T_Stream &operator>>(T_Stream& is , Sphere<T,n>& s)
1012 * \brief Brief Description
1013 *
1014 * Detailed Description
1015 */
1016 template <typename T_Stream , typename T, int n>
1017 inline
1018 T_Stream &operator>>(T_Stream& is , Sphere<T,n>& s) {
1019     ScalarPoint<T,n> ss;
1020     is >> ss;
1021     if(ss.getValue()<0)
1022         s.reset();
1023     else
1024         s = Sphere<T,n>(ss);
1025     return is;
1026 }
1027
1028 /* ! T_Stream &operator>>(T_Stream& is , Sphere<T,n>* s)
1029 * \brief Brief Description
1030 *
1031 * Detailed Description
1032 */
1033 template <typename T_Stream , typename T, int n>
1034 inline
1035 T_Stream &operator>>(T_Stream& is , Sphere<T,n>* s) {
1036     ScalarPoint<T,n> ss;
1037     is >> ss;
1038     if(ss.getValue()<0)
1039         (*s).reset();
1040     else
1041         (*s) = Sphere<T,n>(ss);
1042     return is;
1043 }
1044
1045 #endif
1046
1047
1048
1049 // ****
1050 // **      The Box class      **
1051 // ****
1052
1053
1054 /* ! \class Box gmpoint.h <gmPoint>
1055 * \brief The Static Box class

```

```

1056 * A template Box, the Box is static i.e. the dimentions
1057 * can not be change. The template type must be clean, i.e. is not
1058 * allocating
1059 * memory and without any virtual functions.
1060 *
1061 */
1062 template <typename T, int n>
1063 class Box {
1064 public:
1065     Box();
1066     Box( const APoint<T, n>& p );
1067     Box( const Box<T, n>& b );
1068     Box( const APoint<T, n>& p1, const APoint<T, n>& p2 );
1069     Box( const APoint<T, n>& p1, const APoint<T, n>& p2, const APoint<T
1070         ,n>& p3 );
1071     APoint<T, n> getPointMin() const;
1072     APoint<T, n> getPointMax() const;
1073     APoint<T, n> getPointCenter() const;
1074     Vector<T, n> getPointDelta() const;
1075     T*             getPtr() const;
1076
1077     T&             getValueAt( int i, int j );
1078     T               getValueMin( int i ) const;
1079     T               getValueMax( int i ) const;
1080     T               getValueCenter( int i ) const;
1081     T               getValueDelta( int i ) const;
1082     void            insert( const APoint<T, n>& );
1083     void            insert( const Box<T, n>& );
1084     bool            isIntersecting( const Box<T, n>& b ) const;
1085     bool            isSurrounding( const APoint<T, n>& p ) const;
1086     bool            isSurrounding( const Box<T, n>& b ) const;
1087     void            reset();
1088     void            reset( const APoint<T, n>& p );
1089
1090     Box<T, n>&      operator +=( const APoint<T, n>& p );
1091     Box<T, n>        operator +( const APoint<T, n>& p );
1092     Box<T, n>&      operator +=( const Box<T, n>& b );
1093     Box<T, n>        operator +( const Box<T, n>& b );
1094
1095
1096
1097 private:
1098     Point<T, n>    _min;
1099     Point<T, n>    _max;
1100
1101 }; // END class Box
1102
1103
1104
1105
1106
1107 #ifdef GM_STREAM

```

```
1108 /* ! T_Stream &operator<<(T_Stream& os , const Box<T,n>& b)
1109 * \brief Brief Description
1110 *
1111 * Detailed Description
1112 */
1113
1114 template<typename T_Stream , typename T, int n>
1115 T_Stream &operator<<(T_Stream& os , const Box<T,n>& b) {
1116     os << b.getPointMin() << GMseparator::Element << b.getPointMax();
1117     return os;
1118 }
1119
1120 /* ! T_Stream &operator<<(T_Stream& os , const Box<T,n>* b)
1121 * \brief Brief Description
1122 *
1123 * Detailed Description
1124 */
1125
1126 template<typename T_Stream , typename T, int n>
1127 T_Stream &operator<<(T_Stream& os , const Box<T,n>* b) {
1128     os << b->getPointMin() << GMseparator::Element << b->getPointMax()
1129         ();
1130     return os;
1131 }
1132
1133 /* ! T_Stream &operator>>(T_Stream& is , Box<T,n>& b)
1134 * \brief Brief Description
1135 *
1136 * Detailed Description
1137 */
1138
1139 template<typename T_Stream , typename T, int n>
1140 T_Stream &operator>>(T_Stream& is , Box<T,n>& b) {
1141     Separator es(GMseparator::Element);
1142     Point<T,n> p1,p2;
1143     is >> p1 >> es >> p2;
1144     b.reset(p1);
1145     b.insert(p2);
1146     return is;
1147 }
1148
1149 /* ! T_Stream &operator>>(T_Stream& is , Box<T,n>* b)
1150 * \brief Brief Description
1151 *
1152 * Detailed Description
1153 */
1154
1155 template<typename T_Stream , typename T, int n>
1156 T_Stream &operator>>(T_Stream& is , Box<T,n>* b) {
1157     Separator es(GMseparator::Element);
1158     Point<T,n> p1,p2;
1159     is >> p1 >> es >> p2;
1160     b->reset(p1);
1161     b->insert(p2);
1162     return is;
1163 }
1164
1165 #endif
```

```

1162
1163
1164
1165 // ****
1166 // **      The PlaneArrow class      **
1167 // ****
1168
1169
1170 /*! \class PlaneArrow gmpoint.h <gmPoint>
1171 *
1172 * \brief PlaneArrow class
1173 *
1174 * Detailed Description of class
1175 */
1176
1177 template <typename T, int n>
1178 class PlaneArrow : public Arrow<T,n> {
1179 public:
1180     PlaneArrow();
1181     PlaneArrow(const Point<T,n>& p);
1182     PlaneArrow(const Point<T,n>& p, const Vector<T,n>& v);
1183     PlaneArrow(const Arrow<T,n>& a);
1184
1185     void setNormal(const Vector<T,n>& v);
1186
1187     const Vector<T,n>& getNormal() const;
1188     APoint<T,n> getClosestPoint(const Point<T,n>& p) const;
1189     Vector<T,n> getDistanceVector(const Point<T,n>& p) const;
1190     T getDistanceTo(const Point<T,n>& p) const;
1191 };
1192 // END class PlaneArrow
1193
1194
1195
1196
1197 // ****
1198 // ***** The init of I-matrix and SubSpace *****
1199 // ***** NOT FOR EXTERNAL USE !!!!!!
1200 // ****
1201 /*! \class M_I_ gmpoint.h <gmPoint>
1202 * \brief The init of I-matrix and SubSpace
1203 *
1204 * NOT FOR EXTERNAL USE !!!!!!
1205 */
1206 template <typename T, int n, int m>
1207 class M_I_ {
1208 public:
1209     M_I_();
1210     T* getPtr() const;
1211
1212 private:
1213     T _p[n*m];
1214 };
1215 // END class M_I_
1216 } // END namespace GMlib

```

```
1217  
1218  
1219  
1220  
1221 // Include implementations  
1222 #include "gmpoint.c"  
1223  
1224  
1225  
1226 #endif // __gmPOINT_H__
```

[caption=test]

Chapter 2

Mathematical spaces and notations

In ancient time "space" was a geometric abstraction of the three-dimensional space observed in real life. In mathematics, Euclidean space is the two or three-dimensional space of Euclidean geometry, as well as the generalizations of these notions to higher dimensions. The term "Euclidean" distinguishes these spaces from the curved spaces of non-Euclidean geometry, and is named for the Greek mathematician Euclid of Alexandria (300 bc). It is common to define Euclidean space using Cartesian coordinates and is modelled by the real coordinate space that typically is described by a center point, origin, and n unit vectors that are orthogonal to each other. It is usual to denote the spaces \mathbb{E}^n if we wish to emphasize its Euclidean nature, but \mathbb{R}^n is used as well since the latter is assumed to have the standard Euclidean structure.

In modern mathematics, spaces are defined as sets with some added structure. They can also be described as different types of manifolds, which are spaces that are locally equivalent to Euclidean space, and where the properties are defined largely on local connectedness of points that lie on the manifold. Curves and Surfaces are manifolds if they are regular and not self intersecting.

There are however, many diverse mathematical objects that are called spaces. For example, vector spaces such as function spaces may have infinite numbers of independent dimensions and a notion of distance very different to Euclidean space, and topological spaces replace the concept of distance with a more abstract idea of nearness.

Mathematical spaces often form a hierarchy, i.e., one space may inherit all the characteristics of a parent space. For instance, all inner product spaces are also normed vector spaces, because the inner product induces a norm on the inner product space such that

$$\|s\| = \sqrt{\langle s, s \rangle}.$$

Beside Euclidean spaces, vector spaces and finite dimensional function spaces we shall in the following look at Compact spaces, Affine spaces, Projective spaces and Grassmannien, and we will look at some maps.

2.1 Euclidean spaces, cartesian coordinates and vector spaces

The notation in this book is following standard commonly used. Because geometry is in most used embedded in the plane or in the three dimensional space the functions are commonly vector valued or point valued (affine space will be described in section 2.4). A vector space is a mathematical structure formed by a collection of elements called vectors, which may be added together and multiplied (“scaled”) by numbers, called scalars in this context. Scalars are often taken to be real numbers, but they can also be something else.

A list explaining notations is:

- ✓ We denotes Euclidian spaces \mathbb{R}^d , where d is the degree of the space, \mathbb{R}^2 is the plane and \mathbb{R}^3 is the 3D space.
- ✓ Cartesian coordinate system, is typical used for Euclidean spaces. It specifies each point uniquely in an Euclidian space by a set of numerical coordinates, which are the signed distances from the point to fixed perpendicular directed lines, measured in the same unit of length, i.e. $p = (x, y, z)$.
- ✓ Each reference line of a Cartesian coordinate system is called a coordinate axis or just axis of the system, and the point where they meet is its origin, $O = (0, 0, 0)$. The coordinates can also be defined as the positions of the perpendicular projections of the point onto the two axes, expressed as a signed distances from the origin.
- ✓ A vector or point is notated with a letter, Latin or Greek. A vector can be expressed as either a row-vector or a column-vector,

$$\mathbf{r} = (r_1, r_2, r_3) = \begin{pmatrix} r_1 \\ r_2 \\ r_3 \end{pmatrix} \in \mathbb{R}^3.$$

- ✓ An inner product between two vectors r and $s \in \mathbb{R}^d$, $d > 1$ is denoted

$$\langle r, s \rangle = (r_1, \dots, r_d) \begin{pmatrix} s_1 \\ \vdots \\ s_d \end{pmatrix} = (r_1 s_1 + r_2 s_2 + \dots + r_d s_d) \in \mathbb{R}.$$

An inner product between two vectors is the same as the scalar product.

- ✓ A vector product in \mathbb{R}^3 (related to wedge product) is, given the vectors $r = (r_1, r_2, r_3)$ and $s = (s_1, s_2, s_3)$:

$$r \wedge s = \left(\begin{vmatrix} r_2 & r_3 \\ s_2 & s_3 \end{vmatrix}, \begin{vmatrix} r_1 & r_3 \\ s_1 & s_3 \end{vmatrix}, \begin{vmatrix} r_1 & r_2 \\ s_1 & s_2 \end{vmatrix} \right) = (r_2 s_3 - s_2 r_3, r_1 s_3 - s_1 r_3, r_1 s_2 - s_1 r_2)$$

It can be shown that the vector product is orthogonal to both of its vectors, i.e.

$$\langle r \wedge s, r \rangle = \langle r \wedge s, s \rangle = 0.$$

- ✓ A wedge product in \mathbb{R}^2 is, given the vectors $r = (r_1, r_2)$ and $s = (s_1, s_2)$. A wedge product is

$$r \wedge s = \begin{vmatrix} r_1 & r_2 \\ s_1 & s_2 \end{vmatrix} = r_1 s_2 - s_1 r_2$$

A wedge product in \mathbb{R}^2 is the same as the inner product

$$\langle r, s^L \rangle = r \wedge s,$$

where the vector s^L is vector s rotated 90° counter clockwise.

✓

2.2 Homeomorphism, diffeomorphism and manifolds

We will shortly (and superficially) from a mathematical point of view) look at the mathematical foundation of parametric curves and surfaces. To those readers who wish to study this more thoroughly and more seriously, we recommend reading Spivak [134] or DoCarmo [46, 47].

We start explaining homomorphism. From Greek, meaning something like “similar shape”.

Definition 2.1. A homeomorphism, also called a continuous transformation, is an equivalence relation and one-to-one correspondence between points in two geometric objects or topological spaces that is continuous in both directions. It is a map which preserves all the topological properties of a given space. Two objects/spaces with a homeomorphism between them are called homeomorphic, and from a topological viewpoint they are the same. A transformation is a homeomorphism if it:

- is a bijection, i.e. one to one and onto,
- is continuous,
- the inverse function is continuous.

A homeomorphism which also preserves distances is called an isometry. Affine transformations as rotation, scaling, translation, shearing are another type of common geometric homeomorphism.

To give the reader a kind of picture of this, think of homeomorphism as deforming clay where it is forbidden to make holes, cut and glue. An isometry is like bending a sheet into a part of a cylinder.

A related expression is diffeomorphism

Definition 2.2. A diffeomorphism is an isomorphism in the category of smooth manifolds. It is an invertible function that maps one differentiable manifold to another, such that both the function and its inverse are smooth. A function is a diffeomorphism if:

- the function is differentiable,
- the inverse function is differentiable,

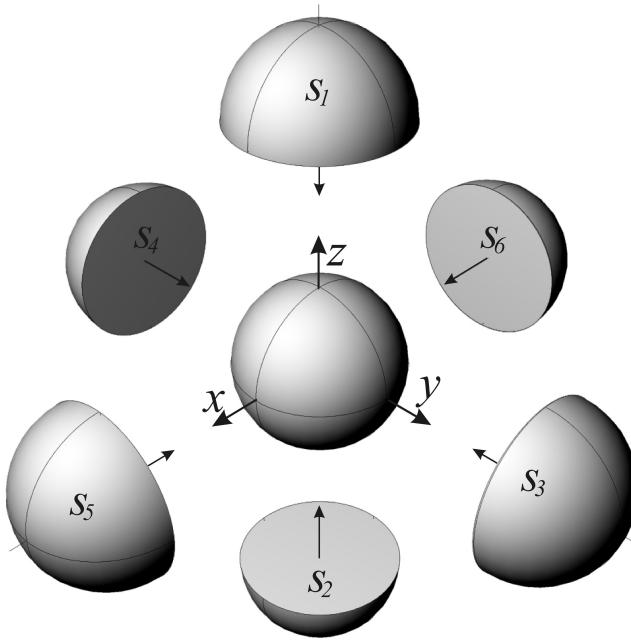


Figure 2.1: Example of Sphere that is covered by an atlas that has six charts/maps. The six maps are denoted S_1, S_2, S_3, S_4, S_5 and S_6 .

2.2.1 Local/global parametrization, charts and atlas

We first look at what we can call local curves and surfaces and their parametric form.

1. A local parametric curve is a curve defined by a parametric equation, involving one parameter, most commonly s or t . Typically they will be curves in \mathbb{R}^2 or \mathbb{R}^3 (and commonly denoted $c(t)$). More precisely, a parameterized differentiable curve is a differentiable map

$$c : I \subset \mathbb{R} \rightarrow \mathbb{R}^n, \quad n \in (\mathbb{Z}^+ \text{ but usually } \{2, 3\}),$$

i.e. from an open interval $I = (a, b)$ of the real line \mathbb{R} into $\mathbb{R}^n, n = 2, 3$.

2. A local parametric surface is a surface defined by a parametric equation, involving two parameters, most commonly (u, v) or (s, t) . Typically they will be surfaces in \mathbb{R}^3 (and commonly denoted $s(u, v)$). More precisely, a parameterized differentiable local surface is a differentiable map

$$s : U \subset \mathbb{R}^2 \rightarrow \mathbb{R}^n, \quad n \in (\mathbb{Z}^+ \text{ but generally } \{3\}).$$

i.e. from an open set $U \subset \mathbb{R}^2$ into \mathbb{R}^3 .

Both curves and surfaces are usually defined on closed domains \bar{I} or \bar{U} that are subsets of the open domains used in the definition.

Every curve can be described with one parametrization. This might not always be practical and it is actually possible to use several intersecting intervals (with nonempty intersections) as domains for several different parameterizations which together cover the whole curve.

For surfaces it is not actually possible to parameterize every surface using one parametrization. In particular it is generally not possible to parameterize bounded, compact and connected surfaces of different possible topological genus g (number of holes/handles) by using only one parametrization. One classical example is the sphere. There is no homeomorphism¹ between a sphere and \mathbb{R}^2 . But if a sphere is being punctured (one point taken away) then the rest of a sphere and \mathbb{R}^2 is homeomorphic.

In Figure 2.1 there are six parametrization that together cover a sphere,

$$\begin{aligned}s_1(u, v) &= (u, v, \sqrt{1 - (u^2 + v^2)}), \\s_2(u, v) &= (u, v, -\sqrt{1 - (u^2 + v^2)}), \\s_3(u, v) &= (u, \sqrt{1 - (u^2 + v^2)}, v), \\s_4(u, v) &= (u, -\sqrt{1 - (u^2 + v^2)}, v), \\s_5(u, v) &= (\sqrt{1 - (u^2 + v^2)}, u, v), \\s_6(u, v) &= (-\sqrt{1 - (u^2 + v^2)}, u, v),\end{aligned}$$

where for all six maps, $\{s_i\}_{i=1}^6$, the domain is the open disk $u^2 + v^2 < 1$.

If a curve or a surface is regular and not self intersecting, it can be considered as a manifold. This is a topological space that is locally homeomorphic to Euclidean space² by a collection (called an atlas) of homeomorphisms called charts. The composition of one chart with the inverse of another chart is a function called a transition map, and defines a homeomorphism of an open subset of Euclidean space onto another open subset of Euclidean space.³

Remark 1. As will be shown later, in the construction of Expo-Rational B-splines both local and global maps are used (in local and global geometry). This can be seen in both curves and tensor product surfaces. Triangular based surfaces are even more in accordance with the definition of manifolds, thus, there is no global parametrization. This opens for consistent constructions of i.g. bounded, compact and connected surfaces of different possible topological genus.

2.3 Compact spaces

Curves and surfaces or geometric objects in general are usually geometrically bounded or "geometrically not infinite", and they in general also includes their endpoints or edges.

They are denoted as compact objects (manifolds or spaces). These are curves including the start and end point, rectangles including the four edges, surfaces in general including their edges. a sphere, a torus, etc.

¹Two spaces with a homeomorphism between them are called homeomorphic. From a topological viewpoint they are the same. Roughly speaking, a topological space is a geometric object, and the homeomorphism is a continuous stretching and bending of the object into a new shape. Thus, a square and a circle are homeomorphic to each other, but a sphere and a donut are not.

²That is, around every point, there is a neighborhood that is topologically the same as the open unit ball in \mathbb{R}^2 or respective \mathbb{R}^3 .

³A closer study of manifolds can be found in [134] or [47].

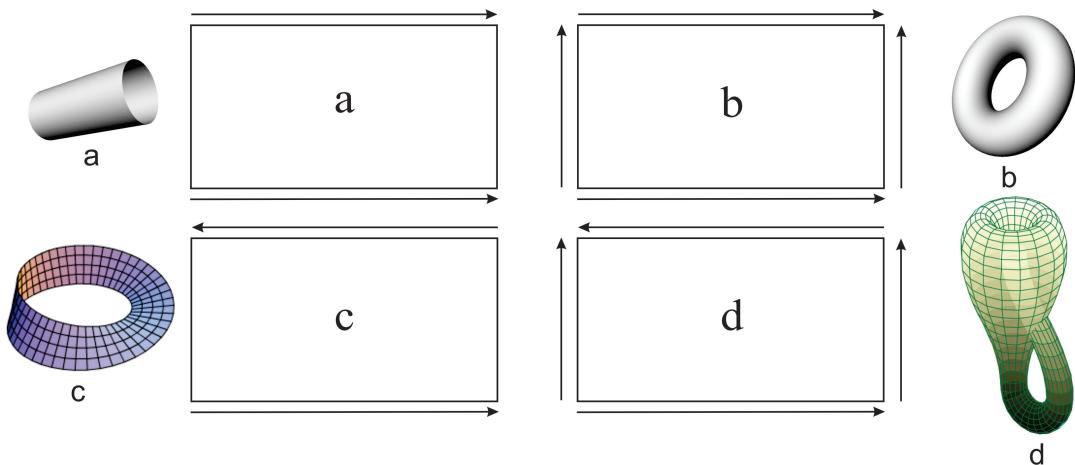


Figure 2.2: Four examples of 2D compact objects (sets), cylinder, torus, Möbius band and Klein bottle. A cylinder has two edges, a Möbius band has only one edge, and a torus and a Klein bottle have none edges.

Formally, a topological space X is called compact if each of its open covers (an infinite set of open sets covering the space totally) has a finite subcover (can be reduced to a finite set open sets covering the space totally). Otherwise it is called non-compact.

In Figure 2.2 there are four examples of compact objects.

- a) The rectangle a is bent so that two edges are glued together. The result is a cylinder.
- b) The rectangle b is first the same as a, then it is also bent in the other direction and the other two edges are also glued together. The result is a torus.
- c) The rectangle c is bent but now the two edges are turned before they are glued together. The result is a Möbius band.
- d) The rectangle d is bent but now the two edges are turned before they are glued together, then it is also bent in the other direction and the other two edges are also first turned and then glued together. The result is Klein bottle.

The Euclidean space and the Affine space described in the next section are not compact spaces, but the Projective and Grassmann spaces are compact spaces as we will see in section 2.5.

2.4 Affine space

A vector space is a set of elements we call vectors. A vector space is closed under finite vector addition and scalar multiplication. This means that a sum of two vectors also is a vector and to scale a vector by scalar multiplication also give a vector. This gives the following legal operations,

$$v = k_1 v_1 + k_2 v_2$$

where v , v_1 and v_2 are vectors and k_1 and k_2 are scalars (real numbers). It follows that you can sum many vectors into one if the number of vectors to sum is finite. In the previous section we have recognized that we do not only have vectors we sometimes also have points. Points acts different from vectors and do not fit into the concept of vector space.

We therefor introduce a new space.

The affine space

is a space of points, p , and associated vectors, v . The following two operations describes the connection between points and vectors and operations on vectors:

$$\begin{aligned} p &= p_1 + k v, && \text{connection between points and vectors,} \\ v &= k_1 v_1 + k_2 v_2, && \text{operations on vectors only,} \end{aligned} \quad (2.1)$$

where the k 's are scalars, i.e. real values.

- ◊ In addition there is one more legal operation. It is an operation on points only, and it is called the affine combination and will be further described below.

From the first line in (2.1), turning the expression we get

$$v = \tilde{k} (p - p_1), \quad \text{where } \tilde{k} = \frac{1}{k}. \quad (2.2)$$

Further, from a combination of all three expressions above (2.1 and 2.2) we get:

The affine combination

also called the barycentric combination. **This is the only legal operations on points only** and is formulated as follows (where p_i are points and k_i , $i = 0, \dots, n$, are scalars)

$$p = \sum_{i=0}^n k_i p_i, \quad \text{where } \sum_{i=0}^n k_i = 1. \quad (2.3)$$

- ◊ The name indicate to compute the barycenter. It is to sum up weighted points where the weights sum up to 1.
- ◊ If all weights k_i , $i = 0, \dots, n$, are nonnegative, $k_i \geq 0$, $i = 0, 1, \dots, n$, we call (2.3) for a **convex** affine combination.

The affine combination follows because (2.3) can be rewritten to fit the expression in (2.1),

$$p = p_0 + v, \quad \text{where } v = \sum_{i=1}^n k_i (p_i - p_0),$$

and it therefore follows that

$$k_0 = 1 - \sum_{i=1}^n k_i.$$

Recall from the Hermite basis functions (page 78) that the two basis functions blending points was summing up to 1. Also remember that the basis functions for Bezier curves,

the set of Bernstein polynomials that are only blending points sums up to 1 for all degrees (lemma 4.2 on page 87).

Later we will show that this is the case also for B-splines and NURBS.

The reason why this is so important is that to fulfill an affine combination are invariant under affine maps. These are the most used maps in computer graphics, CAD/CAM etc.

affine maps

are translation, scaling, rotation, shear and parallel projections, and are in general taken on the familiar form

$$\Theta p = A p + v$$

where p is a point and v is a associated vector in an affine space. If this is as usual \mathbb{R}^3 , then A is a 3×3 matrix.

Geometrically, an affine map (affine transformation) in Euclidean space is one that preserves

1. The collinearity relation between points; i.e., three points which lie on a line continue to be collinear after the transformation
2. Ratios of distances along a line; i.e., for distinct collinear points p_1, p_2, p_3 , the ratio $\frac{|p_2-p_1|}{|p_3-p_2|}$ is preserved

An affine map is invertible if and only if the matrix A is invertible. This is typically scaling, rotation, share and translation. Parallel projection is not invertible. A scaling matrix is a diagonal matrix where the inverse is a matrix where each number on the diagonal is inverted,

$$A = \begin{pmatrix} \alpha & 0 & 0 \\ 0 & \beta & 0 \\ 0 & 0 & \gamma \end{pmatrix}, \quad \text{and where } A^{-1} = \begin{pmatrix} \frac{1}{\alpha} & 0 & 0 \\ 0 & \frac{1}{\beta} & 0 \\ 0 & 0 & \frac{1}{\gamma} \end{pmatrix}.$$

This matrix is scaling a vector different in each coordinate.

A rotational matrix is an orthonormal matrix where the transposed matrix is the inverse,

$$A = \begin{pmatrix} \cos(\alpha) & -\sin(\alpha) & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{pmatrix},$$

rotating with the angle α around the z-axis, and where

$$A^{-1} = \begin{pmatrix} \cos(-\alpha) & -\sin(-\alpha) & 0 \\ \sin(-\alpha) & \cos(-\alpha) & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} \cos(\alpha) & \sin(\alpha) & 0 \\ -\sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{pmatrix} = A^T.$$

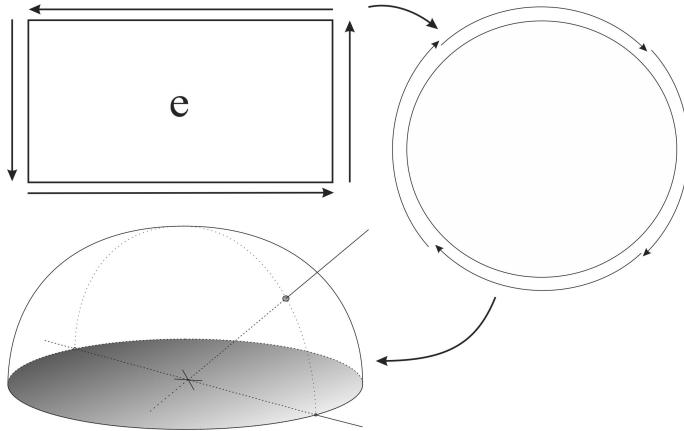


Figure 2.3: An illustration of a projective plane \mathbb{P}^2 . We start with a plane with four connected edges. Two and two edges are glued together. an edge and the edge on the opposite side are glued after one of them are turned. The plane is then deformed to a plate where the antipodal point on the edge are the same point. The plate is then deformed to a half sphere including the edge circle in the xy-plane. There is now a one to one map (homeomorphism) to the set of all lines through origin in \mathbb{R}^3 .

2.5 Projective space and Grassmannien

A projective space is the space of one-dimensional vector subspaces of a given vector space. \mathbb{P}^n denotes a projective space of dimension n . In general we assume it to be a projective space on real numbers. A more precise notation is \mathbb{RP}^n for real projective spaces, and \mathbb{CP}^n for complex projective spaces and so on.

\mathbb{P}^n can also be viewed as the set consisting of \mathbb{R}^n together with its points at infinity (in Figure 2.3 is this the edges glued with each other).

In Figure 2.3 is there an illustration of the projective plane, \mathbb{P}^2 . We start with a plane where the edges are glued together, one edge is glued with the edge on the opposite side, but turned before they are glued together. The figure shows us the maps, homeomorphism, from the initial plane via a plate where the antipodal points are the same point and then an half sphere including the edge circle in the xy-plane, to the set of all infinite lines through origin. This follows because there is a one to one map between the half sphere and the set of lines. This is obvious for all points on the half sphere except for the edge, but it is also true for the lines in the xy-plane because the antipodal points are the same point because of the initial gluing.

It follows that \mathbb{P}^2 is the set of all lines through origin in \mathbb{R}^3 , ie. a one-dimensional vector subspace. From the example in Figure 2.3 it is also clear that the projective plane \mathbb{P}^2 is compact.

A point in \mathbb{P}^n can be described by $n + 1$ cartesian coordinates but these coordinates can be scaled by any nonzero scalar. As an example, using this description it follows that $q \in \mathbb{P}^3$ can be expressed by

$$q = (kx, ky, kz, kw),$$

where q is independent of k , i.e. k can be any nonzero real.

There is a canonical injection of \mathbb{R}^n into \mathbb{P}^n . This means that an affine space \mathbb{R}^n can be embedded isomorphically in \mathbb{P}^n by the standard injection

$$(x_1, \dots, x_n) \mapsto (x_1, \dots, x_n, 1).$$

Affine points can be recovered from projective ones with the mapping

$$(x_1, \dots, x_n, x_{n+1}) \sim \left(\frac{x_1}{x_{n+1}}, \dots, \frac{x_n}{x_{n+1}}, 1 \right) \mapsto \left(\frac{x_1}{x_{n+1}}, \dots, \frac{x_n}{x_{n+1}} \right).$$

In general can \mathbb{P}^n be seen as the set of all lines through origin in \mathbb{R}^{n+1} . Contrary to Euclidean spaces \mathbb{R}^n is the projective spaces \mathbb{P}^n compact, as clearly can be seen in Figure 2.3. The points at infinite in \mathbb{R}^n is the same as the horizontal lines in \mathbb{P}^n , so the projective space also includes the points at infinite and is therefor compact.

Grassmannian is a generalization of the Projective spaces; it is the space of d -dimensional vector subspaces of a given n -dimensional vector space, $0 < d < n$, and is denoted $Gr(n, d)$. It follows that a projective space $\mathbb{P}^n = Gr(n, 1)$.

For example, $Gr(n, 2)$ can be the space of all planes through origin in an Euclidian space \mathbb{R}^n .

2.6 Homogenous coordinates

Homogeneous coordinates, introduced by August Ferdinand Möbius in 1827 (Der barycentrische Calcül), are a system of coordinates used in projective geometry much as Cartesian coordinates are used in Euclidean geometry. They have the advantage that the coordinates of points, including points at infinity, can be represented using finite coordinates. Formulas involving homogeneous coordinates are often simpler and more symmetric than their Cartesian counterparts. Homogeneous coordinates have a range of applications, including computer graphics and 3D computer vision, where they allow affine transformations and, in general, projective transformations to be easily represented by a matrix.

If the homogeneous coordinates of a point are multiplied by a non-zero scalar then the resulting coordinates represent the same point. An additional condition must be added on the coordinates to ensure that only one set of coordinates corresponds to a given point, so the number of coordinates required is, in general, one more than the dimension of the projective space being considered. For example, two homogeneous coordinates are required to specify a point on the projective line and three homogeneous coordinates are required to specify a point on the projective plane.

Chapter 3

GMlib, an Open Source programming library

Geometric modeling is fundamental in order to develop computer programs for design, modeling, simulations, calculations, ... It is the backbone of any "virtual reality based" implementation. It affects the structure, defines virtual objects and models, and make calculations. All the theory and algorithms in this book are therefore implemented and tested. These implementations has also been used to create all tables and figures in this book. The implementations are actually parts of an Open Source project called GMlib, that is free software, licensed under the GNU Lesser General Public License (LGPL). The project can be found on <http://episteme.hin.no/>.

GMlib was initially started and first developed by the staff at Narvik university college now joined with University of Tromsø to UIT- The Arctic University of Norway. The library is further partly based on student works done in the subjects "Geometric modeling", "Applied Geometry and special effects", "Graphics/Virtual Reality/Animation" and diploma works done by master students in computer science. The project "GMlib" started in 1995, and the first alpha version, written by the author of this book, was released in 1997. It consisted mainly of a template library of n-dimensional points, vectors, unit vectors, point with an associated scalar (spheres and scalar fields), point with an associated vector (arrows and vector fields), boxes, matrix, SqMatrix (squared matrices) and HqMatrix (Homogenius matrices). It also contained the derived 2D and 3D point/vector classes. In addition it was also implemented a set of dynamic container classes; Array, Array_T (tiny arrays), Array_LX (large extendibly arrays), DVector and DMatrix (dynamic vectors and matrices) with favorable properties for geometrical algorithms. Delaunay triangulation with order $O(n)$ and the accompanying classes was introduced in 1999. In 2000 a display system with Camera, Scene and an hierarchical object-tree based on OpenGL and GLUT was introduced. The following year the system was specialized to include time based simulation. The cameras become displayable objects with all the properties and privileges of those objects and the orientation of displayable objects (including cameras) could be locked to always point against an other object. During this period (2001-2003) the library was improved, bugs was removed and it compiled on both Windows and

LINUX platforms using different compilers. The parametric Surface- and Curve- base classes was introduced and a library of several curves and surfaces developed. In 2003 and 2004 new functionality was implemented, a window system was introduced and the library was tested to work together with GLUT and Qt.¹ In 2008 GM.lib was introduced as an Open Source project under GNU LGPL (Lesser General Public License). Today versions (January - 2016) is v1.1

3.1 The modules of GMlib

GMlib is a programming library developed in C++ . The main field for the programming library is geometry, computer graphic and simulation. Thus, it is assumed that the user has prior knowledge in C++ , Euclidian and affine geometry, homogenous coordinate systems and OpenGL.

GMlib consists of eight modules plus a simulation framework, and in addition there is a window extension:

Module 1 - Core: is a template library for elements of n-dimensional affine spaces, i.e. points, vectors, matrices, frames, simplices etc. defined by at least two template parameters $\langle\text{typename T, int n}\rangle$ where type T is typically float or double or ..., and the other template type *integer n* is the dimension (2D/3D/...).

Module 2 - OpenGL: contains the graphic interface including interfaces to GPU and shaders for graphic purposes.

Module 3 - Scene: is a module consisting of an hierarchical object-tree with a Scene as a root object and SceneObject as nodes. The scene has display, select, tracing, editing and simulation functionality. It also includes special SceneObject as different camera types, different light types etc.

Module 4 - Parametrics: has a template baseclass Parametric with the inherited PCurve, PSurf and PTriangle handling parametric curves, and surfaces as Bezier, Hermite, B-Splines, NURBS, Exponential B-Splines, etc.

Module 5 - Trianglesystem: is a set of template classes TriangleFacets, Triangle, Edge and Vertex useful for terrain modeling, and other type of "2D-based" (without overhang) triangle surfaces. A Delaunay triangulation also including constraints is implemented.

Module 6 - Stereolithography: handling stl-objects for stereo-lithography , object scanning, 3D-printing etc.

Module 7 - OpenCL: contains the interfaces to GPU to make general computation (not for graphic purposes).

¹Qt is a cross-platform application and UI framework for developers using C++ or QML, a CSS & JavaScript like language. Qt can be used under open source (LGPL) terms.

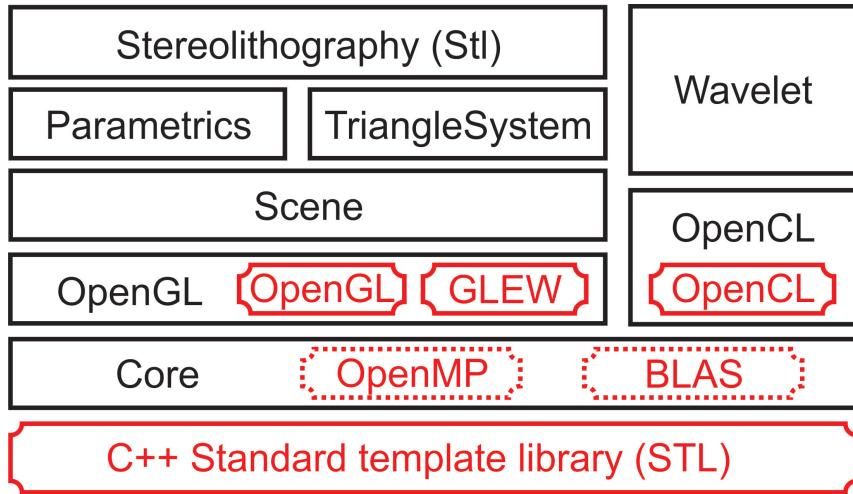


Figure 3.1: The figure shows an illustration of the dependency between different modules of GMlib and also to external libraries. Black means GMlib modules while red means external libraries. If the frame is dotted red it means that it is optional. The dependence goes from the bottom up.

Module 8 - Wavelet: is a module for developing wavelets. It includes several predefined types and is specialized for image compression and computations.

Simulations: is a framework for time based simulation including a controller example. The baseclass `DisplayObject` has a virtual function: `void localSimulate(double dt)` where `dt` is the time in second passed since last call given from the system.

Windows: is an extension to and thus not a part of GMlib. It is a framework for Windows and user interface in general and is using Qt.

The module dependencies are illustrated in Figure 3.1. It shows that:

- Core depends only on C++ Standard template library (STL),
 - optional is the core depending on OpenMP for parallelization,
 - optional is the core depending on BLAS for optimized matrix computations.
- The GMlib module OpenGL depends on Core,
 - it also depend on the system OpenGL library,
 - and on the system GLEW library.
- The GMlib module OpenCL depends on Core,
 - it also depend on the system OpenCL library.
- The module Scene depends on OpenGL and Core
- Parametric depends on Scene, OpenGL and Core
- Trianglesystem depends on Scene, OpenGL and Core

- Stereolithography depends on Parametric, Trianglesystem, Scene, OpenGL and Core.
- Wavelet depends on OpenCL and Core.

GM.lib can be connected to libraries for computations.

GM.lib has 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 red cubes.

The “selector” and an appurtenant “selectorgid” 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.

3.2 The core module

The core module can be divided in two sections, the *types* and the *containers*.

The *types* is a set of static objects, with no virtual functions, constructed for geometric modelling. The types is listed in Table 3.1. There is four groups, the affine objects point and vectors, the matrix, the simplex and the frame.

3.3 Scene hierarchy - GMlib:SH

The geometry-based main data structure in GMlib is the Scene-hierarchy. The scene hierarchy organizes the world. We can put object into it, and they can be organized hierarchically. The objects can be positioned and oriented in the scene. And the structure is especially adapted for time based simulations.

3.4 Basic Linear Algebra Subprograms - BLAS

Basic Linear Algebra Subprograms - BLAS is a de facto application programming interface standard for publishing libraries to perform basic linear algebra operations such as vector and matrix multiplication. They were first published in 1979, and are used to build

Point< T, n >	<i>The homogenous coordinate is implicit 1</i>
Vector< T, n >	<i>The homogenous coordinate is implicit 0</i>
UnitVector< T, n >	The length is always 1
Arrow< T, n >	Is a Point with an associated Vector
ScalarPoint< T, n >	Is a Point with an associated scalar
Sphere< T, n >	Is a ScalarPoint
Box< T, n >	Two Points, the lower left and upper right corners
Matrix< T, n, m >	<i>A Vector of Vectors</i>
SqMatrix< T, n >	A square Matrix, invertible if determinant $\neq 0$
HqMatrix< T, n >	A homogenous (square) Matrix
Simplex< T, n, m >	<i>A Vector of Points</i>
LineSegment< T, n >	A two-point Simplex
Triangle< T, n >	A three-point Simplex
Tetrahedron< T, n >	A four-point Simplex
KFrame< T, n, m >	<i>Is a Point with an associated Matrix</i>
Line< T, n >	A KFrame where the matrix is one Vector
Plane< T, n >	A KFrame where the matrix is two Vectors
Cube< T, n >	A KFrame where the matrix is three Vectors

Table 3.1: Types (Classes) defined in module **Core**, section types

Array< T >	A sortable dynamic container
ArrayLX< T >	Large extendable array
ArrayT< T >	Tiny array
DVector< T >	A dynamic (mathematical) vector
DMatrix< T >	A dynamic (mathematical) matrix
DVectorN< T, n, K >	An n-dimensional "matrix"

Table 3.2: Types (Classes) defined in module **Core**, section containers

GMlib:SH	The Scene hierarchy
Scene	The root object in the scene tree
SceneObject	The base class for a node in the Scene tree
DisplayObject	For visual objects in the Scene tree
Visualizer	Base class for any type of visualizing of a DisplayObject
Renderer	Base class for any type of rendering of a DisplayObject
Light	Base class for light objects
Event	Base class for events to be introduced in the scene
ScaleObject	Triangle visualizer
Frustum	Is the frustum of a Camera, the visual volume
Selector	Is a DisplayObject that is directly connected to a point in the scene

Table 3.3: From module **Parametric**, visualizers

Parametrics $\langle T, n \rangle$	Base class for the parametric objects below
PCurve $\langle T \rangle$	Parametric curves ($n = 1$), see table 3.6
PSurf $\langle T \rangle$	Parametric surfaces ($n = 2$), see table 3.7
PTriangle $\langle T \rangle$	Parametric triangular surfaces ($n = 2$), see table 3.8

Table 3.4: From module Parametric

GERBSEvaluator $\langle T \rangle$	Base class for the two GERBS evaluations below
ERBSEvaluato $\langle T \rangle$	ERBS - Expo rational B-spline evaluator
BFBSEvaluato $\langle T \rangle$	BFBS - Beta function B-spline evaluator
EvaluatorStatic $\langle T \rangle$	Matrix for Bernstein and Hermite evaluation

Table 3.5: From module Parametric, evaluators

PCurve $\langle T \rangle$	The Base class of the parametric curves below
PBezierCurve $\langle T \rangle$	A Bezier curve
PHermiteCurve $\langle T \rangle$	An Hermite curve
PHermiteCurve2 $\langle T \rangle$	An Hermite curve
PBSplineCurve $\langle T \rangle$	A B-spline curve
PGERBSCurve $\langle T \rangle$	A GERBS curve
PBasisCurve $\langle T \rangle$	Plotting a GERBS basis function
PBSplineBasisCurve $\langle T \rangle$	Plotting a B-spline basis function
PSubCurve $\langle T \rangle$	A sub curve (part of a curve)
PTCurve $\langle T \rangle$	A parametric transformed curve
PCircle $\langle T \rangle$	A parametric circle
PArc $\langle T \rangle$	A parametric arc
PHelicoid $\langle T \rangle$	A parametric helicoid
PButterfly $\langle T \rangle$	A parametric curve, imagine a butterfly
PRoseCurve $\langle T \rangle$	A parametric curve, imagine a rose
PChrysanthemumCurve $\langle T \rangle$	A parametric curve, imagine a chrysanthemum
PSurfCurve $\langle T \rangle$	A parametric curve on a surface
PTriangCurve $\langle T \rangle$	A parametric curve on a triangular surface

Table 3.6: From module Parametric, curves

PSurf< T >	The Base class of the parametric surfaces below
PRotationalSurf< T >	Making surface by rotating a curve
PSphere< T >	Parametric sphere (the poles are treated separately)
PThorus< T >	Parametric torus
PCone< T >	Parametric cone
PCylinder< T >	Parametric cylinder
PBezierSurf< T >	Bezier surface
PBSplineSurf< T >	B-spline surface
PHermiteSurf< T >	Hermite surface
PGERBSSurf< T >	GERBS surface
PSubSurf< T >	Sub surfaces, a part of another surface
PSweepSurf< T >	A sweep surface
PCoonsPatch< T >	Coons patch bilinear or bicubic
PCoonsNewType< T >	Blending surface
PHermiteCurveSurf< T >	Blending curves with Hermite basis functions
PBezierCurveSurf< T >	Blending curves with Bernstein basis functions
PSweepBaseSurf< T >	Parametric surfaces, imagine a horn
PSweepHermiteSurf< T >	Parametric surfaces, imagine a horn
PApple< T >	Parametric surfaces, imagine an apple
PApple2< T >	Parametric surfaces, imagine an apple
PAsteroidalSphere< T >	An asteroidal sphere surfaces
PBentHorns< T >	Parametric surfaces, imagine a horn
PBohemianDome< T >	Parametric surfaces, imagine a Bohemian dome
PBottle8< T >	Parametric surfaces, imagine a bottle
PBoysSurface< T >	Parametric surfaces
PDiniSurface< T >	Parametric surfaces
PEightSurface< T >	Parametric surfaces
PEnnepersSurface< T >	Parametric surfaces
PCrossCap< T >	Parametric surfaces
PInsideOutTorus< T >	Parametric surfaces
PKleinsBottle< T >	Kleins bottle
PKuenSurface< T >	Parametric surfaces
PMoebiusStrip< T >	A Möbius strip
PSeashell< T >	Parametric surfaces, imagine a sea shell
PSinSurface< T >	Parametric surfaces
PSlippersSurface< T >	Parametric surfaces, imagine a horn
PSteinerSurf< T >	Parametric surfaces
PTrianguloidTrefoil< T >	Parametric surfaces, imagine something
PWhitneyUmbrella< T >	Parametric surfaces, imagine something else

Table 3.7: From module Parametric, surfaces

PTriangle< T >	The Base class of the parametric triangular surfaces below
PBezierTriangle< T>	Bezier triangles
PERBSTriangle< T >	GERBS - triangles

Table 3.8: From module Parametric, triangular surfaces

Visualizer	The Base class of the visualizers below
PTriangleVisualizer< T>	Triangle visualizer
PTriangleDefaultVisualizer< T >	Triangle visualizer
PTriangleColorVisualizer< T >	Triangle visualizer
PTriangleColorPointVisualizer< T >	Triangle visualizer
PTriangleNormalsVisualizer< T >	Triangle visualizer
PSurfVisualizer< T >	Triangle visualizer
PSurfPointsVisualizer< T >	Triangle visualizer
PSurfNormalsVisualizer< T >	Triangle visualizer
PSurfDerivativesVisualizer < T >	Triangle visualizer
PSurfDefaultVisualizer< T >	Triangle visualizer
PSurfContoursVisualizer< T >	Triangle visualizer
PCurveVisualizer< T >	Triangle visualizer
PCurvePointsVisualizer< T >	Triangle visualizer
PCurveDerivativesVisualizer< T >	Triangle visualizer
PCurveDefaultVisualizer< T >	Triangle visualizer
PCurveContoursVisualizer< T >	Triangle visualizer

Table 3.9: From module Parametric, visualizers

larger packages such as LAPACK (see below). Heavily used in high-performance computing, highly optimized implementations of the BLAS interface have been developed by hardware vendors such as Intel, AMD for CPU and AMD and NVIDIA for GPU, as well as by other authors, e.g. Goto BLAS (OpenBLAS or SurviveGotoBLAS2) and ATLAS (a portable self-optimizing BLAS). The LINPACK and HPL benchmarks relies heavily on DGEMM, a BLAS subroutine, for its performance. BLAS homepage can be found at <http://www.netlib.orgblas/>.

Below follows a short (not complete) list of libraries that are implementation of the BLAS interface:

Intel MKL - the Intel - Math Kernel Library for Intel - CPU's
<http://software.intel.com/en-us/intel-mkl>

AMD ACML - the AMD - Core Math Library for AMD - CPU's
<http://developer.amd.com/>

OpenBLAS - is based on GotoBLAS2 1.13 BSD. OpenBLAS is an open source project supported by Lab of Parallel Software and Computational Science, ISCAS. It is licensed under BSD style - free software.
<http://xianyi.github.com/OpenBLAS/>

SurviveGotoBLAS2 - is also based on GotoBLAS2 1.13 BSD. SurviveGotoBLAS2 is licensed under AGPL-3.
<http://prs.ism.ac.jp/~nakama/SurviveGotoBLAS2/>

ATLAS - “Automatically Tuned Linear Algebra Software”. It is licensed under BSD style - free software.
<http://math-atlas.sourceforge.net/>

LAPACK - “Linear Algebra PACKage”. It is licensed under BSD style - free software.
<http://www.netlib.org/lapack/>

The following is a short list of libraries for programming of heterogeneous systems, GPU programming and multi-core CPU:

AMD APPML - AMD Accelerated Parallel Processing Math Libraries are software libraries containing FFT and BLAS functions written in OpenCL and designed to run on AMD GPUs. The libraries support running on CPU devices to facilitate debugging and multicore programming. Under AMD Open Source zone.
<http://developer.amd.com/>

NVIDIA CUBLAS - The NVIDIA CUDA Basic Linear Algebra Subroutines library is a GPU-accelerated version of the complete standard BLAS library that delivers a considerable better performance than the CPU versions.
<https://developer.nvidia.com/cublas>

ViennaCL - a free open-source linear algebra library for computations on GPUs and multi-core CPUs. The library is written in C++ and supports CUDA, OpenCL, and OpenMP. It contains core functionality and many other features including BLAS

level 1-3 support and iterative solvers.
<http://viennacl.sourceforge.net/>

OpenCL - The open standard for parallel programming of heterogeneous systems
<http://www.khronos.org/opencl/>

There is a lot of other implementation that can be found if you look on internet, see for example *Basic Linear Algebra Subprograms* at Wikipedia.

3.5 QT - a cross-platform application framework

Qt is a cross-platform application and UI framework for developers using C++ or QML, a CSS and JavaScript like language. Qt is available under GPL v3, LGPL v2 (see <http://www.gnu.org/licenses/old-licenses/lgpl-2.1.html>) and a commercial license. The Qt Project can be found at <http://qt-project.org/>.

Part I

Curves

Chapter 4

Parametric Curves

Imagine the real numbers \mathbb{R} as an infinite straight line. To make a curve we just pick a segment of this infinite straight line. Then we put it into the required space that may be a sheet (Euclidian – \mathbb{R}^2) or a 3 dimensional space, (Euclidian – \mathbb{R}^3), deform it by either stretching or pressing it and bending it in several ways. Then finally we put it in the desired position and orientation, and we have a curve. Note that to cut and glue is *not* an option here. Figure 4.1 give an example of this concept. We think about a curve as a 1-dimensional object. If we make some restrictions on a curve as none degenerations and self intersections, a curve can be called a 1-dimensional manifold (see [134] or [47]).

This is an attempt to give a picture of the concept of a parametric Curve. We can also think about a parametric curve as a track where positions of a moving object is determined by time schedules. To do this in a more mathematical way we first generalize the output not only to be in \mathbb{R}^2 or \mathbb{R}^3 but more general in \mathbb{R}^n , $n > 0$. A more formal definition is:

Definition 4.1. *A parameterized differentiable curve is a differentiable map $\alpha : I \longrightarrow \mathbb{R}^n$ of an open interval $I = (a, b)$ of the real line \mathbb{R} into \mathbb{R}^n . (Note that a half open or a closed interval is just a restriction of an open interval.)*

The first example is a circle in a plane, where the interval $I = (0, 2\pi]$ is half open. In the

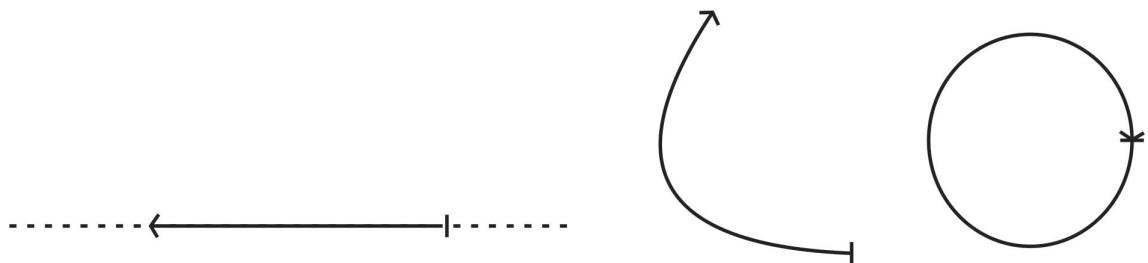


Figure 4.1: On left hand side is a part of the real numbers plotted as a dotted line, the half open interval $I = (0, 2\pi]$ is marked as solid. In the mid-figure is the interval bent and finally, on right hand side, is it bent into a circle. If the circle is a unit circle with radius $r = 1$ as in expression (4.1), then the length of the interval and the curve is the same, there is no stretching, while expression (4.2) is stretching to twice the length.

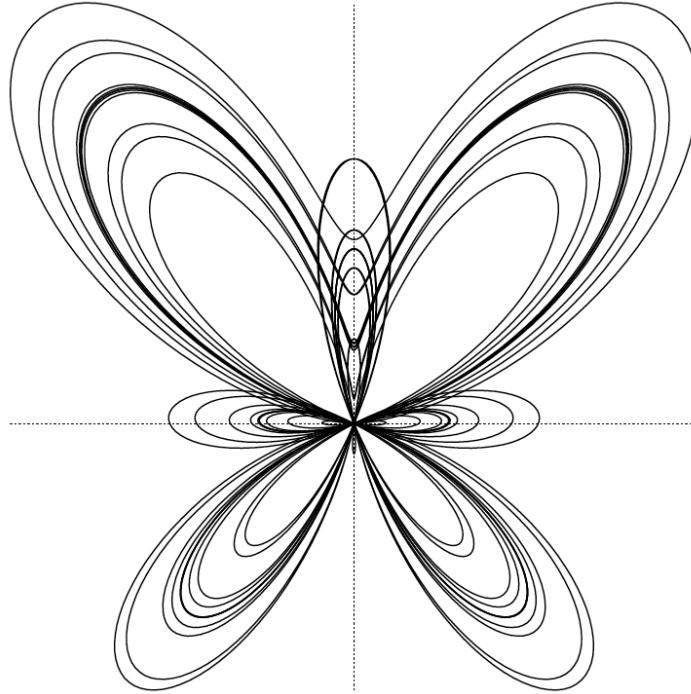


Figure 4.2: A plot of the butterfly curve following from expression (4.3).

expression is therefore t going from 0 (not included) to 2π (included). A specific t -value results in a vector with an x and y component, i.e. a vector in the Euclidean space \mathbb{R}^2 . Therefore, we call the following function (4.1) for vector-valued,

$$\alpha(t) = \begin{pmatrix} \cos t \\ \sin t \end{pmatrix}, \quad 0 < t \leq 2\pi. \quad (4.1)$$

The center of the circle in (4.1) is in origin and the radius is 1. If we want to change the radius of the circle to $r = 2$, and move the center of the circle to another position, for example to a point with coordinates $x = 1, y = 2$ we get:

$$\alpha(t) = 2 \begin{pmatrix} \cos t \\ \sin t \end{pmatrix} + \begin{pmatrix} 1 \\ 2 \end{pmatrix} = \begin{pmatrix} 2 \cos t + 1 \\ 2 \sin t + 2 \end{pmatrix}, \quad 0 < t \leq 2\pi. \quad (4.2)$$

A more sophisticated example is the butterfly, a curve made by Temple H. Fay in 1989 and published in [56]. The curve is plotted in Figure 4.2 and the equation is as following:

$$\alpha(t) = \begin{pmatrix} \left(e^{\cos t} - 2 \cos(4t) - \sin^5\left(\frac{t}{12}\right) \right) \sin t \\ \left(e^{\cos t} - 2 \cos(4t) - \sin^5\left(\frac{t}{12}\right) \right) \cos t \end{pmatrix}, \quad 0 < t \leq 24\pi. \quad (4.3)$$

This curve, figure 4.2, is 12 following circles deformed by cyclic changing and collapsing radius as can be seen from equation (4.3). Parametric curves can be made by all types of functions, trigonometric functions, logarithmic functions, exponential functions and of course polynomials that has been among the most popular choices. The next example is

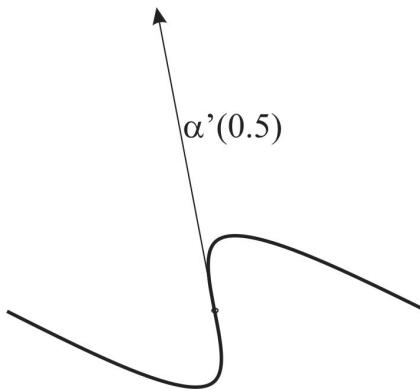


Figure 4.3: A plot of the polynomial based curve from expression (4.4). A derivative and tangent vector $\alpha'(\frac{1}{2})$ is also plotted.

therefore a 2D - vector valued parametric curve based on polynomials:

$$\alpha(t) = \begin{pmatrix} 6t - 9t^2 + 6t^3 \\ -3t + 9t^2 - 6t^3 \end{pmatrix}, \quad 0 \leq t \leq 1. \quad (4.4)$$

This curve, expression (4.4), is drawn in Figure 4.3 where the starting point of the curve is the origin of the plane. Note that $\alpha(t) = (x(t), y(t))$ for a given t is a point on the curve. The variable t is called the parameter of the curve. The image set $\alpha(I) \subset \mathbb{R}$ is called the trace of α . And the word differentiable means that derivative of $x(t)$ and $y(t)$ exist everywhere in the domain.

Vector spaces (Linear spaces)

Before we go further we shortly look at the concept of vectors and set of vectors called a vector space. In general we expect vector space, normed vector spaces and inner product spaces and their properties to be known by the reader.

Definition 4.2. A vector space is a set that is closed under finite vector addition and scalar multiplication (i.e the result of the operations is elements in the space). The basic example is n -dimensional Euclidean space notated \mathbb{R}^n , where every element is represented by a list of n real numbers, scalars are real numbers, addition is componentwise, and scalar multiplication is multiplication on each term separately.

In this book is a classic vector notated with a letter, for example v . If $v \in \mathbb{R}^2$ (element of a 2D plane) it will have 2 elements $v = (x, y)$ and can also be written transposed as $v = \begin{pmatrix} x \\ y \end{pmatrix}$. If $v_1 = (x_1, y_1)$ and $v_2 = (x_2, y_2)$ are vectors $\in \mathbb{R}^2$ and k is a scalar $\in \mathbb{R}$ then

$$v = v_1 + kv_2 = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} + k \begin{pmatrix} x_2 \\ y_2 \end{pmatrix} = \begin{pmatrix} x_1 + kx_2 \\ y_1 + ky_2 \end{pmatrix}$$

An inner product, which is a scalar, will have the following notation

$$\langle v_1, v_2 \rangle = x_1x_2 + y_1y_2$$

A norm (length) of a vector is defined by

$$|v| = \sqrt{\langle v, v \rangle} = \sqrt{x^2 + y^2}$$

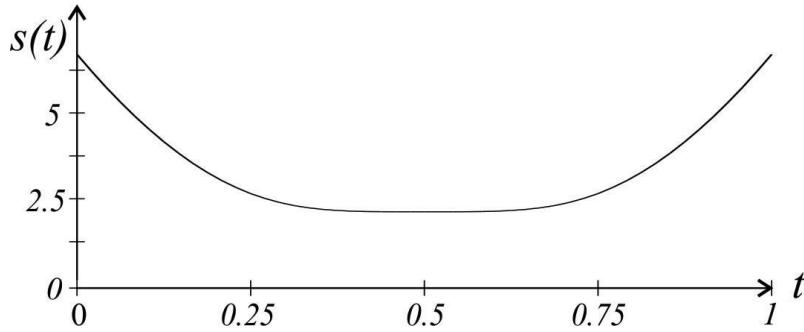


Figure 4.4: The figure shows the speed $s(t) = |\alpha'(t)|$ of the curve from expression (4.4).

4.1 Differentiations

We start with the first derivative of the curve.

- We denote the first derivative¹, $\frac{d\alpha}{dt}(t)$ as $\alpha'(t) = (x'(t), y'(t))$.
- $\alpha'(t)$ is a vector and is called the tangent or velocity vector of the curve in the point $\alpha(t)$.
- The length of the tangent/velocity vector $|\alpha'(t)|$ is called the speed at the point $\alpha(t)$.

In Figure 4.3 is the tangent vector at $t = 0.5$ plotted. We can clearly see that it is tangential to the curve. Because the parameter interval is 1 (from 0 to 1) will the average length of the vector be about as long as the curve (the arc length). The speed $s(t) = |\alpha'(t)|$ for $t \in [0, 1]$ of the curve defined in (4.4) is plotted in Figure 4.4. We can calculate the curve length from the average speed times the time used. In Figure 4.4 is this the same as the area under the function $s(t)$, $t \in [0, 1]$. In general, we get the following integral for computing the curve/arc length:

$$l(\alpha) = \int_a^b s(t) dt = \int_a^b |\alpha'(t)| dt = \int_a^b \sqrt{\langle \alpha'(t), \alpha'(t) \rangle} dt \quad (4.5)$$

Using the curve defined in (4.4) we get the speed function

$$\begin{aligned} s(t) &= \sqrt{(18t^2 - 18t + 6, -18t^2 + 18t - 3) \begin{pmatrix} 18t^2 - 18t + 6 \\ -18t^2 + 18t - 3 \end{pmatrix}} \\ &= 3\sqrt{72t^4 - 144t^3 + 108t^2 - 36t + 5}. \end{aligned}$$

And the curve/arc length is

$$l(\alpha) = \int_0^1 3\sqrt{72t^4 - 144t^3 + 108t^2 - 36t + 5} dt \approx 3.28.$$

¹In section 9.1.1, we will introduce a differential operator d . This is a generalization of differentiation to also include multivariate functions. With such a general description, can the first derivative of a curve $c(t)$ be described by the expression $d(c)_t(1)$, which means that we at a point $c(t)$ have a vector describing where we will be if we move, without changing direction and speed, during 1 time unit.

4.1.1 Regular curves - arc length parameterization

A regular curve is a curve, $\alpha(t), t \in I$, where the tangent/velocity vector does not vanish, i.e. $s(t) = |\alpha'(t)| \neq 0$ for all $t \in I$.

In order to emphasize some important properties, we will look into a special type of curves, curves that are arc length parameterized.

Given an arc length parameterized curve $\zeta(t)$, it follows that,

- $|\zeta'| = 1$, speed is 1 over the entire curve.
- $\langle \zeta'', \zeta' \rangle = 0$, the second derivative is always orthogonal to the first derivative, because it is only the direction of the first derivative that change, not the length (speed).
- $\kappa = |\zeta''|$, We call the curvature for κ . The curvature of a curve is defined to be the length of the second derivatives of an arc length parameterized curve.

For arc length parameterized curves embedded in \mathbb{R}^3 we have also the following properties

- Frenet frame (also called TNB frame) is a frame, three unit vectors, T , N , B , orthogonal to each other in a right hand system. $T = \zeta'$, $N = \frac{\zeta''}{\kappa}$ and $B = T \wedge N$ (the vector product). The existence of a TNB-frame at a point requires that $\kappa \neq 0$.
- τ is the torsion of a curve. It measures the speed of rotation of the binormal vector N at the given point, and is given by $\tau = -\langle N, B' \rangle$.

4.1.2 Reparameterization

Reparameterization has no influence on the shape of a curve. It change speed and parameter interval of a curve, but it does not affect any of the properties of a curve which we call the intrinsic properties (shape, curve length, curvature, torsion, etc.).

Given a curve $c(t)$, $t \in I \subset \mathbb{R}$. A curve,

$$\rho(t) = c(\omega(t)) = c \circ \omega(t),$$

is a reparameterization of $c(t)$ if

- $\omega(t)$ is differentiable for $t \in I$,
- if there exist an inverse ω^{-1} that also is differentiable for $t \in I$.

It follows that $\omega(t)$ must be a strongly monotone function.

The tangent/velocity vector of $\rho(t)$ is

$$\rho'(t) = \omega'(t)c' \circ \omega(t),$$

and the speed

$$s(t) = |\omega'(t)c' \circ \omega(t)| = |\omega'(t)| |c' \circ \omega(t)|.$$

4.1.3 Curvature

The second derivative of a curve $c(t)$ is denoted as $c''(t)$. The second derivative describe the linear change of the first derivative. The second derivative can be decomposed in, changing the speed and changing the direction.

It is obvious that not only the change of speed depends on the speed, but also changing of direction. To get the intrinsic values we can reparametrize a curve to be arc length parameterized, that the speed is 1 all over. Given a curve

$$\zeta(t) = c \circ \omega(t).$$

It follows that $\zeta(t)$ is arc length parameterized if $\omega'(t) = \frac{1}{|c'|}$.

To simplify we skip the parameter in the expression in the following. Using the kernel rule and the rule of derivation of a product, we get

$$\zeta' = \omega' c'$$

and

$$\zeta'' = \omega'' c' + (\omega')^2 c''.$$

To calculate the curvature and to avoid having to calculate ω'' we just use the vector product to find $|\zeta''|$. This can be done because the vector product of two parallel vectors is zero. Further, we know that the curvature, κ , is equal the length of the second derivative, $|\zeta''|$, and that ζ'' is normal to the first derivative ζ' and that $|\zeta'| = 1$. This knowledge will be used in the calculation. To simplify we first look at curves in \mathbb{R}^3 . We then get,

$$\begin{aligned}\kappa &= |\zeta''| = |\zeta' \wedge \zeta''| \\ &= |\omega''||\zeta' \wedge c'| + |\omega'|^2 |\zeta' \wedge c''| \\ &= 0 + |\omega'|^3 |c' \wedge c''|,\end{aligned}\tag{4.6}$$

which gives

Curvature for curves in \mathbb{R}^3 and \mathbb{R}^2

$$\kappa = \frac{|c' \wedge c''|}{|c'|^3}, \quad c \in \mathbb{R}^3.$$

where \wedge means the 3D vector (cross) product.

In \mathbb{R}^2 we can use the same formula but here we use the the wedge product giving a scalar, $a \wedge b = a_x b_y - a_y b_x$. This also open for signed curvatures,

$$\kappa = \frac{c' \wedge c''}{|c'|^3}, \quad c \in \mathbb{R}^2.$$

which give a positive curvature on left hand side and negative curvature on right hand side of the curve.

Related to the curvature is the radius of curvature that is:

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

4.2 Basis functions

The curve expression (4.4) can be reorganized in the following way

$$\alpha(t) = \begin{pmatrix} 6t - 9t^2 + 6t^3 \\ -3t + 9t^2 - 6t^3 \end{pmatrix} = \begin{pmatrix} 6 \\ -3 \end{pmatrix} t + \begin{pmatrix} -9 \\ 9 \end{pmatrix} t^2 + \begin{pmatrix} 6 \\ -6 \end{pmatrix} t^3, \quad 0 \leq t \leq 1. \quad (4.7)$$

This expression is on the general form

$$\alpha(t) = a_0 1 + a_1 t + a_2 t^2 + a_3 t^3, \quad 0 \leq t \leq 1. \quad (4.8)$$

where $a_0 = (0, 0)$, $a_1 = (6, -3)$, $a_2 = (-9, 9)$ and $a_3 = (6, -6)$ are coefficient vectors, and the basis functions are on the monomial form (the power basis) $\{t^i\}_{i=0}^3$.

Parametric Polynomial Curves

In general a polynomial parametric curve of degree d using the power basis will be on the following form

$$\alpha(t) = \sum_{i=0}^d a_i t^i, \quad t \in I \subset \mathbb{R} \quad (4.9)$$

where a_i , $i = 0, 1, \dots, d$ are vectors in the space where the curve is embedded.

One can easily see that the power basis functions can act in the same way as basis vectors in a $(d + 1)$ -dimensional Euclidean vector space. They are clearly linear independent because you can not get one of the basis functions from a linear combination of the others. We can therefore treat the set of all polynomial based functions of most degree d as a vector space where $1, t, t^2, \dots, t^d$ are basis vectors as in (4.9).

This leads to the general expression for curve constructions,

General parametric Curve construction

$$\alpha(t) = \sum_{i=1}^k c_i b_i(t), \quad t \in I \subset \mathbb{R} \quad (4.10)$$

where c_i , $i = 1, \dots, k$ are coefficient and vector/points in the space where the curve is embedded, and $b_i(t)$, $i = 1, \dots, k$ are a set of linearly independent functions spanning a given finite dimensional function space.

Notice that these are finite dimensional vector spaces, and the curves have clearly big restrictions in the shaping property. A second degree curve has no inflection points, a third degree curve has only one inflection point and so on. Imagine that you draw a curve on the free hand. To find an expression for that curve will require an infinite degree of the polynomial and, thus, an infinite dimensional vector space. Infinite dimensional vector spaces can be useful theoretically, but they are certainly not possible to implement for curves and surfaces in shaping applications on a computer.

Function space

is a very useful concept in constructing and analyzing curves for geometric modeling and design (a function space is a vector space).

Definition 4.3. A function space is a set of functions of a given kind from a set X to a set Y . It is in general a topological vector space. A typical example is the set of all polynomial of degree most 3 mapping the interval $I = [0, 1]$ to \mathbb{R}^n , $n > 0$ but finite. We denote it $\mathcal{P}_3(I)$

In the following sections we will show that in the polynomial case there are several set of basis functions for a given function space and thus to use for the same curves, all with valuable properties.

Infinite dimensional function spaces are also useful, especially theoretically. An infinite dimensional function space is a set $\mathcal{F}(I)$ that is the collection of all real-valued continuous functions defined on some interval I . $\mathcal{F}^{(n)}(I)$ is the collection of all functions $\in \mathcal{F}(I)$ with n continuous derivatives.

Example of an infinite dimensional function space is the Hilbert space L^2 , the set of all functions $f : \mathbb{R} \rightarrow \mathbb{R}$ such that the integral of $|f(x)|^2$ over the whole real line is finite. In this case, the inner product is

$$\langle f, g \rangle = \int_{-\infty}^{\infty} f(x)g(x)dx$$

4.3 Hermite Curves

We use the polynomial example first formulated in (4.4) and reformulated in (4.7).

$$\alpha(t) = \begin{pmatrix} 6 \\ -3 \end{pmatrix}t + \begin{pmatrix} -9 \\ 9 \end{pmatrix}t^2 + \begin{pmatrix} 6 \\ -6 \end{pmatrix}t^3 \quad 0 \leq t \leq 1.$$

We compute the first derivative

$$\alpha'(t) = \begin{pmatrix} 6 \\ -3 \end{pmatrix} + 2\begin{pmatrix} -9 \\ 9 \end{pmatrix}t + 3\begin{pmatrix} 6 \\ -6 \end{pmatrix}t^2, \quad 0 \leq t \leq 1.$$

We compute the expressions with the start parameter value $t = 0$ and the end parameter value $t = 1$

$$\alpha(0) = \begin{pmatrix} 6 \\ -3 \end{pmatrix}0 + \begin{pmatrix} -9 \\ 9 \end{pmatrix}0^2 + \begin{pmatrix} 6 \\ -6 \end{pmatrix}0^3 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

$$\alpha(1) = \begin{pmatrix} 6 \\ -3 \end{pmatrix}1 + \begin{pmatrix} -9 \\ 9 \end{pmatrix}1^2 + \begin{pmatrix} 6 \\ -6 \end{pmatrix}1^3 = \begin{pmatrix} 3 \\ 0 \end{pmatrix}$$

$$\alpha'(0) = \begin{pmatrix} 6 \\ -3 \end{pmatrix} + 2\begin{pmatrix} -9 \\ 9 \end{pmatrix}0 + 3\begin{pmatrix} 6 \\ -6 \end{pmatrix}0^2 = \begin{pmatrix} 6 \\ -3 \end{pmatrix}$$

$$\alpha'(1) = \begin{pmatrix} 6 \\ -3 \end{pmatrix} + 2\begin{pmatrix} -9 \\ 9 \end{pmatrix}1 + 3\begin{pmatrix} 6 \\ -6 \end{pmatrix}1^2 = \begin{pmatrix} 6 \\ -3 \end{pmatrix}$$

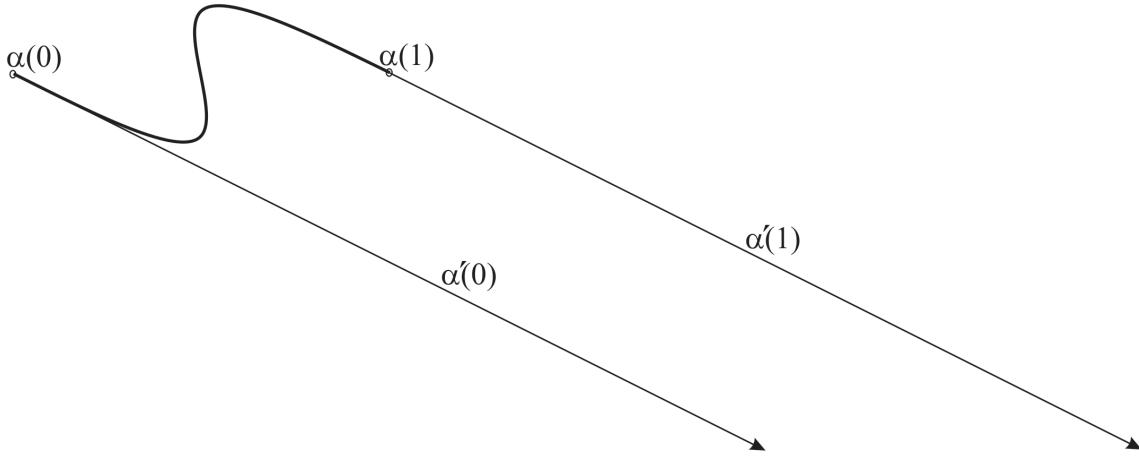


Figure 4.5: A plot of the curve from expression (4.4) and the four coefficients, the start point and respective first derivative and the end point and the respective first derivative.

In Figure 4.5 there is a plot of $\alpha(t)$. But now the start and the end point of the curve, $\alpha(0)$ and $\alpha(1)$ are marked with circles, and the respective tangent vectors at the starting point and the endpoint of the curve, $v_1 = \alpha'(0)$ and $v_2 = \alpha'(1)$ are plotted as arrows.

We now look at the general expression for polynomial based curves of degree 3 (equation (4.8) in section 4.2) and its first derivative,

$$\begin{aligned}\alpha(t) &= a_0 + a_1 t + a_2 t^2 + a_3 t^3 \\ \alpha'(t) &= a_1 + 2a_2 t + 3a_3 t^2.\end{aligned}\tag{4.11}$$

We start by computing from (4.11) the position in the starting point $\alpha(0)$, the position of the endpoint $\alpha(1)$, the first derivative (the tangent vector) in the starting point $\alpha'(0)$ and the first derivative of the endpoint $\alpha'(1)$;

$$\begin{aligned}\alpha(0) &= a_0 \\ \alpha(1) &= a_0 + a_1 + a_2 + a_3 \\ \alpha'(0) &= a_1 \\ \alpha'(1) &= a_1 + 2a_2 + 3a_3.\end{aligned}$$

We reorganize these equations in matrix notation and get

$$\begin{pmatrix} \alpha(0) \\ \alpha(1) \\ \alpha'(0) \\ \alpha'(1) \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 2 & 3 \end{bmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix}.\tag{4.12}$$

We invert the matrix and turn the total equation in (4.12) to

$$\begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -3 & 3 & -2 & -1 \\ 2 & -2 & 1 & 1 \end{bmatrix} \begin{pmatrix} \alpha(0) \\ \alpha(1) \\ \alpha'(0) \\ \alpha'(1) \end{pmatrix}.\tag{4.13}$$

We write the curve $\alpha(t)$ from (4.11) on vector notation using inner product,

$$\alpha(t) = (1, t, t^2, t^3) \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix}.$$

The next step is to replace the vector of coefficients $(a_i, i = 0, 1, 2, 3)$ on right hand side in the equation above with the expression on right hand side in (4.13)

$$\alpha(t) = (1, t, t^2, t^3) \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -3 & 3 & -2 & -1 \\ 2 & -2 & 1 & 1 \end{bmatrix} \begin{pmatrix} \alpha(0) \\ \alpha(1) \\ \alpha'(0) \\ \alpha'(1) \end{pmatrix}. \quad (4.14)$$

Finally we compute the vector with the power basis on left hand side with the matrix,

$$\alpha(t) = (1 - 3t^2 + 2t^3, \quad 3t^2 - 2t^3, \quad t - 2t^2 + t^3, \quad -t^2 + t^3) \begin{pmatrix} \alpha(0) \\ \alpha(1) \\ \alpha'(0) \\ \alpha'(1) \end{pmatrix}.$$

In the vector on left hand side we now have a new set of basis functions for a 3dt degree polynomial based curve.

Hermite Curves, 3th degree

is third degree polynomial curves on the following form:

$$\alpha(t) = \alpha(0) H_1(t) + \alpha(1) H_2(t) + \alpha'(0) H_3(t) + \alpha'(1) H_4(t), \quad t \in [0, 1], \quad (4.15)$$

where the coefficients are the position at start $\alpha(0)$ and end $\alpha(1)$, and the first derivative at start $\alpha'(0)$ and end $\alpha'(1)$. The four 3rd degree Hermite basis functions are

$$\begin{aligned} H_1(t) &= 1 - 3t^2 + 2t^3 = (2t + 1)(1 - t)^2, \\ H_2(t) &= 3t^2 - 2t^3 = (3 - 2t)t^2, \\ H_3(t) &= t - 2t^2 + t^3 = t(t - 1)^2, \\ H_4(t) &= -t^2 + t^3 = t^2(t - 1). \end{aligned} \quad (4.16)$$

The fact that the matrix in (4.12) is inverted in (4.13) and therefore obvious is invertible proves that the set of Hermite basis functions are linearly independent and thus is a basis set for 3dt degree polynomial curves. The four basis functions are plotted in Figure 4.6.

The special about Hermite curves is that the four coefficients are the starting point and it's related velocity vector (the first derivative) and the end point and it's related velocity vector. This is very practical for constructing curves. This also explains the name²;

²Hermite Curves and Hermite interpolation is named after Charles Hermite (1822 – 1901), a French mathematician who did research on number theory, quadratic forms, invariant theory, orthogonal polynomials, Abelian and elliptic functions, and algebra.

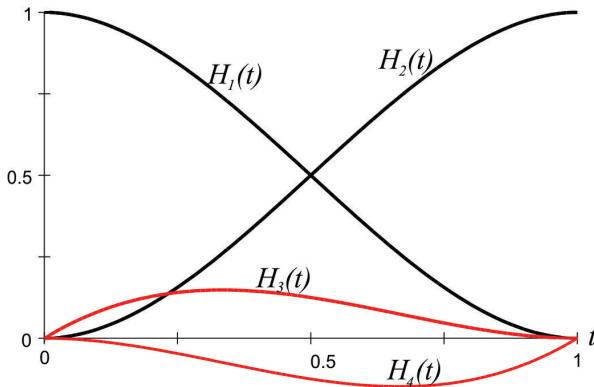


Figure 4.6: A plot of the four Hermite basis functions in (4.16). The functions connected to vectors are red. One can clearly see the symmetry in the pairs of functions.

Hermite interpolation is interpolating a point and the derivative vectors at the same point. This is also called osculatory interpolation (osculatory means kissing on Latin).

The property that is most important for the set of Hermite basis function of degree three, $\mathbf{H}_3(t) = [H_1(t), H_2(t), H_3(t), H_4(t)]$, is,

$$\begin{aligned}\mathbf{H}_3(0) &= [1, 0, 0, 0], \\ \mathbf{H}_3(1) &= [0, 1, 0, 0], \\ \mathbf{H}'_3(0) &= [0, 0, 1, 0], \\ \mathbf{H}'_3(1) &= [0, 0, 0, 1].\end{aligned}\tag{4.17}$$

Notice that $H_1(t) + H_2(t) = 1$. This is called fulfilling the property of partition of unity. These two basis functions, $H_1(t)$ and $H_2(t)$, are blending the points. The other two basis functions $H_3(t)$ and $H_4(t)$ are blending vectors and does not sum to 1. The importance of this was explained in the section of Affine spaces, see page 52. The use of a power basis set is sometimes called the algebraic form while using the Hermite basis set is called the geometric form.

Hermite interpolation can also be done for higher degree polynomials. We now look at the general expression for polynomial based curves of degree 5 and it's first and second derivative,

$$\begin{aligned}\alpha(t) &= a_0 + a_1 t + a_2 t^2 + a_3 t^3 + a_4 t^4 + a_5 t^5 \\ \alpha'(t) &= a_1 + 2a_2 t + 3a_3 t^2 + 4a_4 t^3 + 5a_5 t^4. \\ \alpha''(t) &= 2a_2 + 6a_3 t + 7a_4 t^2 + 20a_5 t^3.\end{aligned}\tag{4.18}$$

We start by computing from (4.18) the position in the starting point $\alpha(0)$, the position of the endpoint $\alpha(1)$, the first derivative (the tangent vector) in the starting point $\alpha'(0)$ and the first derivative of the endpoint $\alpha'(1)$, the second derivative in the starting point $\alpha''(0)$

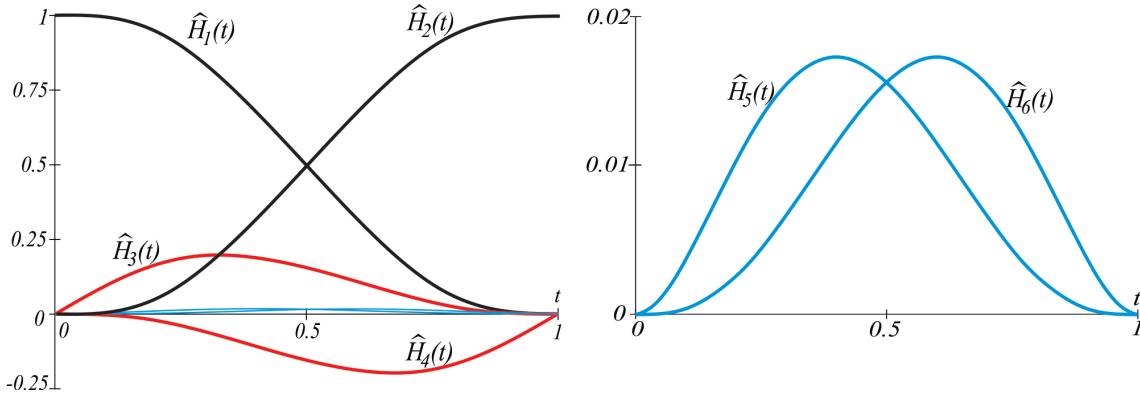


Figure 4.7: A plot of the six 5th degree Hermite basis functions defined in (4.23). The functions connected to the second derivatives are also plotted separately on right hand side, since the values are so small that we can nearly see them together with the other basis functions on the plot on left hand side.

and the second derivative of the endpoint $\alpha''(1)$;

$$\begin{aligned}\alpha(0) &= a_0 \\ \alpha(1) &= a_0 + a_1 + a_2 + a_3 + a_4 + a_5 \\ \alpha'(0) &= a_1 \\ \alpha'(1) &= a_1 + 2a_2 + 3a_3 + 4a_4 + 5a_5 \\ \alpha''(0) &= 2a_2 \\ \alpha''(1) &= 2a_2 + 6a_3 + 7a_4 + 20a_5.\end{aligned}$$

We reorganize the equations in matrix notation and get

$$\begin{pmatrix} \alpha(0) \\ \alpha(1) \\ \alpha'(0) \\ \alpha'(1) \\ \alpha''(0) \\ \alpha''(1) \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 3 & 4 & 5 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 2 & 6 & 12 & 20 \end{bmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{pmatrix}. \quad (4.19)$$

We invert the matrix and turn the total equation in (4.19) to

$$\begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & 0 \\ -10 & 10 & -6 & -4 & \frac{-3}{2} & \frac{1}{2} \\ 15 & -15 & 8 & 7 & \frac{3}{2} & -1 \\ -6 & 6 & -3 & -3 & \frac{-1}{2} & \frac{1}{2} \end{bmatrix} \begin{pmatrix} \alpha(0) \\ \alpha(1) \\ \alpha'(0) \\ \alpha'(1) \\ \alpha''(0) \\ \alpha''(1) \end{pmatrix}. \quad (4.20)$$

We write the curve $\alpha(t)$ on vector notation using inner product,

$$\alpha(t) = \begin{pmatrix} 1, t, t^2, t^3, t^4, t^5 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{pmatrix}.$$

The next step is to replace the vector of coefficients $(a_i, i = 0, 1, 2, 3, 4, 5)$ on right hand side in the equation above with the expression on right hand side in (4.20)

$$\alpha(t) = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & 0 \\ -10 & 10 & -6 & -4 & \frac{-3}{2} & \frac{1}{2} \\ 15 & -15 & 8 & 7 & \frac{3}{2} & -1 \\ -6 & 6 & -3 & -3 & \frac{-1}{2} & \frac{1}{2} \end{pmatrix} \begin{pmatrix} \alpha(0) \\ \alpha(1) \\ \alpha'(0) \\ \alpha'(1) \\ \alpha''(0) \\ \alpha''(1) \end{pmatrix}. \quad (4.21)$$

Finally we compute the vector with the power basis on left hand side with the matrix, and we get the new set of basis functions for a 5th degree polynomial based curve. The six basis functions are plotted in figure 4.7.

Hermite Curves, 5th degree

is a fifth degree polynomial curves on the following form:

$$\begin{aligned} \alpha(t) = & \alpha(0) \hat{H}_1(t) + \alpha(1) \hat{H}_2(t) + \alpha'(0) \hat{H}_3(t) + \\ & \alpha'(1) \hat{H}_4(t) + \alpha''(0) \hat{H}_5(t) + \alpha''(1) \hat{H}_6(t), \quad t \in [0, 1], \end{aligned} \quad (4.22)$$

where the coefficients are the position at start $\alpha(0)$ and end $\alpha(1)$, the first derivative at start $\alpha'(0)$ and end $\alpha'(1)$, and the second derivatives at start $\alpha''(0)$ and end $\alpha''(1)$. The six 5th degree Hermite basis functions are

$$\begin{aligned} \hat{H}_1(t) &= 1 - 10t^3 + 15t^4 - 6t^5 &= (6t^2 + 3t + 1)(1-t)^3 \\ \hat{H}_2(t) &= 10t^3 - 15t^4 + 6t^5 &= (6t^2 - 15t + 10)t^3 \\ \hat{H}_3(t) &= t - 6t^3 + 8t^4 - 3t^5 &= (3t + 1)(1-t)^3 t \\ \hat{H}_4(t) &= -4t^3 + 7t^4 - 3t^5 &= (3t - 4)(1-t)t^3 \\ \hat{H}_5(t) &= \frac{1}{2}t^2 - \frac{3}{2}t^3 + \frac{3}{2}t^4 - \frac{1}{2}t^5 &= \frac{1}{2}(1-t)^3 t^2 \\ \hat{H}_6(t) &= \frac{1}{2}t^3 - t^4 + \frac{1}{2}t^5 &= \frac{1}{2}(1-t)^2 t^3. \end{aligned} \quad (4.23)$$

4.4 Bézier Curves

In Figure 4.5 was the curve from (4.4) plotted together with the four Hermite coefficients, two points and two vectors. The next step is to replace the two vectors in the coefficients with point so we only have points as coefficients. Recall that the dimension of the polynomial function space is 4 and that v_i , thus, will have 4 points as coefficients. This means

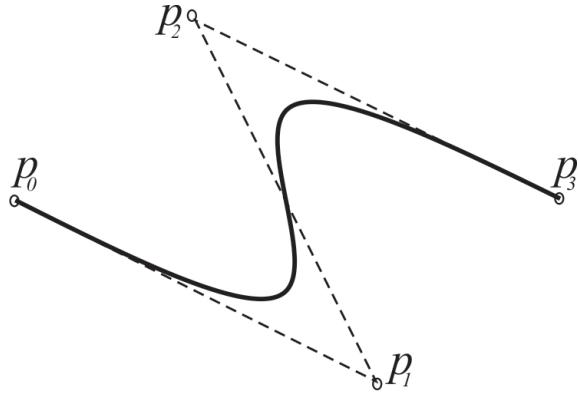


Figure 4.8: A plot of a curve first formulated in expression (4.4). The Bézier control polygon and the four control points (coefficients) are also plotted.

the points can be connected by 3 lines (see Figure 4.8). The length of the first derivative vector is the speed and since the parameter interval is 1 is the speed and the length in average equal. We, therefore, use 1/3 of the length of the vectors to find the points. Vi denote the points p_i , and we get

$$\begin{aligned} p_1 &= \alpha(0) \\ p_2 &= \alpha(0) + \frac{1}{3}\alpha'(0) \\ p_3 &= \alpha(1) - \frac{1}{3}\alpha'(1) \\ p_4 &= \alpha(1). \end{aligned}$$

Reorganizing this in matrix notation,

$$\begin{pmatrix} p_0 \\ p_1 \\ p_2 \\ p_3 \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & \frac{1}{3} & 0 \\ 0 & 1 & 0 & -\frac{1}{3} \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{pmatrix} \alpha(0) \\ \alpha(1) \\ \alpha'(0) \\ \alpha'(1) \end{pmatrix}, \quad (4.24)$$

inverting,

$$\begin{pmatrix} \alpha(0) \\ \alpha(1) \\ \alpha'(0) \\ \alpha'(1) \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ -3 & 3 & 0 & 0 \\ 0 & 0 & -3 & 3 \end{bmatrix} \begin{pmatrix} p_0 \\ p_1 \\ p_2 \\ p_3 \end{pmatrix}. \quad (4.25)$$

From expression (4.14) we have

$$\alpha(t) = (1, t, t^2, t^3) \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -3 & 3 & -2 & -1 \\ 2 & -2 & 1 & 1 \end{bmatrix} \begin{pmatrix} \alpha(0) \\ \alpha(1) \\ \alpha'(0) \\ \alpha'(1) \end{pmatrix}.$$

Replacing the Hermite coefficient with (4.25) we get

$$\alpha(t) = (1, t, t^2, t^3) \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -3 & 3 & -2 & -1 \\ 2 & -2 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ -3 & 3 & 0 & 0 \\ 0 & 0 & -3 & 3 \end{bmatrix} \begin{pmatrix} p_0 \\ p_1 \\ p_2 \\ p_3 \end{pmatrix}.$$

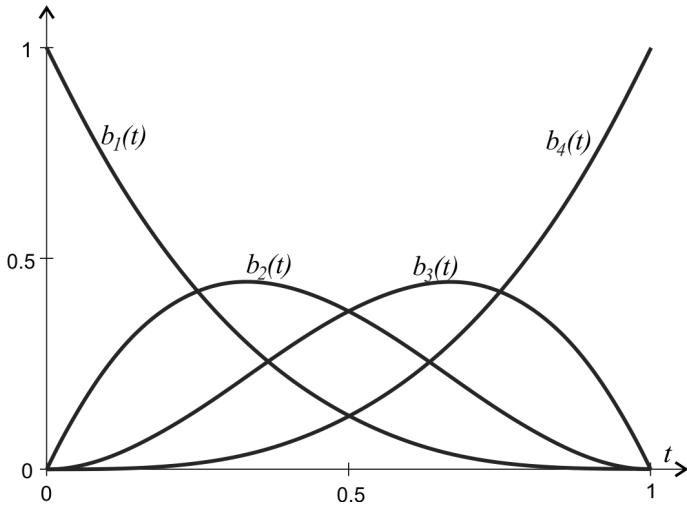


Figure 4.9: A plot of the four Bernstein polynomials of degree 3 (4.27) that forms the set of Bézier basis functions for 3th degree Bézier curves (4.26).

Finally, computing the vector on left hand side with the two matrices we get,

$$\begin{pmatrix} (1-t)^3, & 3t(1-t)^2, & 3t^2(1-t), & t^3 \end{pmatrix} \begin{pmatrix} p_0 \\ p_1 \\ p_2 \\ p_3 \end{pmatrix}.$$

We now have even a new set of basis functions for the curve, earlier described using a power basis in (4.8) and using an Hermite basis in (4.15).

Bézier Curves of degree 3

is uniquely defined by a set of four ordered points describing a control polygon (plotted as dotted lines in the example in figure 4.8),

$$\alpha(t) = p_0 b_{0,3}(t) + p_1 b_{1,3}(t) + p_2 b_{2,3}(t) + p_3 b_{3,3}(t) = \sum_{i=0}^3 p_i b_{i,3}(t), \quad t \in [0, 1], \quad (4.26)$$

where the 4 Bézier basis functions are the set of 3th degree Bernstein polynomials

$$\begin{aligned} b_{0,3}(t) &= (1-t)^3 \\ b_{1,3}(t) &= 3t(1-t)^2 \\ b_{2,3}(t) &= 3t^2(1-t) \\ b_{3,3}(t) &= t^3. \end{aligned} \quad (4.27)$$

The fact that the matrix in (4.24) is inverted in (4.25) proves that the set of 3th degree Bézier basis functions (the Bernstein polynomials) is linearly independent and thus is a basis set for 3th degree polynomial curves. The four basis functions for third degree Bezier curves are plotted in Figure 4.9.

The set of basis functions for 3th degree Bezier curves is called the Bernstein polynomial

set of the degree 3. There is a set of Bernstein polynomials for each degree. They will be described further in the following subsection, but every set are fulfilling the requirement to be a basis set for their respective degrees of polynomial based functions. Bézier curves are using a set of Bernstein polynomials as their blending functions. It follows that there are Bézier curves of all polynomial degrees d and that they are uniquely defined by an ordered set of $d + 1$ points.

Bézier Curves of degree d

is uniquely defined by $d + 1$ control points describing a control polygon. The general expression for a Bézier curve of degree d is,

$$\alpha(t) = \sum_{i=0}^d p_i b_{i,d}(t), \quad t \in [0, 1], \quad (4.28)$$

where the coefficients p_i , $i = 0, 1, \dots, d$ are the $d + 1$ points defining the control polygon of the Bézier curve. The set of the $d + 1$ basis functions $b_{i,d}(t)$, $i = 0, \dots, d$ are the Bernstein polynomials;

$$b_{i,d}(t) = \binom{d}{i} t^i (1-t)^{d-i}, \quad i = 0, 1, \dots, d.$$

The Bézier curve is starting at the first control point and ending at the last control point. The direction of the curve are equal the direction of the control polygon at the starting point and at the endpoint. The control polygon is modeling the curve in such a way that the variation of the curve are smaller than the variation of the control polygon (called the variation diminishing property). Examples can be seen in the figures 4.8 and 4.10.

Bézier curves³ are among the most popular curve descriptions. It is defined only by a point set we call a control polygon. As we can see in figure 4.10 the control polygon is modeling the curve. In the figure is a first degree, a second degree, a third degree and a fourth degree Bézier curve plotted together with their control polygons. The first degree curve and it's control polygon is actually coinciding. As we will see later Bézier curves are actually a special case of B-splines. Several algorithms for computing Bézier curves will therefore be shown later when B-splines are treated.

4.4.1 Bernstein polynomial

Here we will look at algorithm for computing Bernstein polynomials, show that they can be a basis for polynomial based curves, and that a set always sum up to 1.

³Bézier Curves are named after Pierre Bézier (1910 – 1999). He worked for Renault from 1933 – 1975, where he developed his UNISURF CAD-CAM system. He published and patented the results in 1962 although Paul de Casteljau (1930 –) already in 1959 had made an algorithm for computing Bézier Curves but only published this in an internal Citroën report.

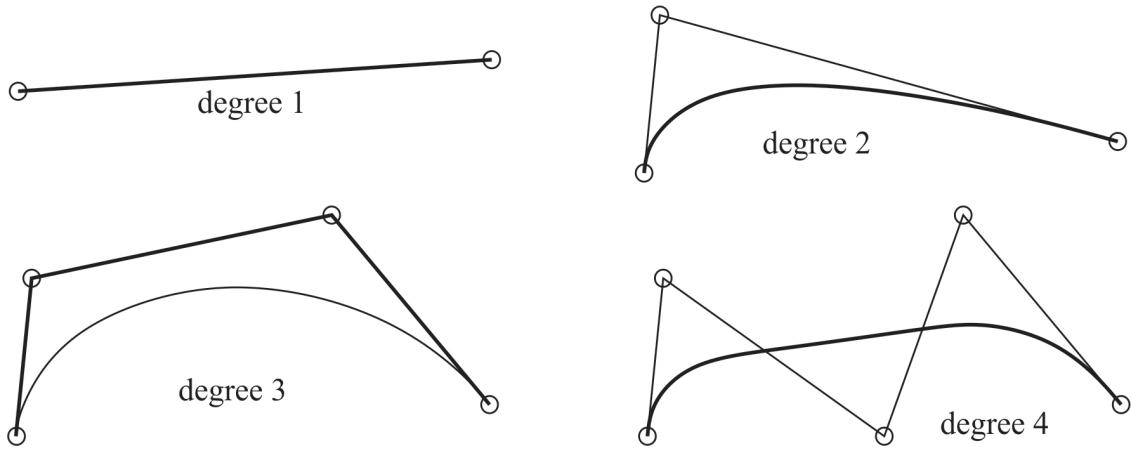


Figure 4.10: A plot of four Bézier curves. They are of degree 1, 2, 3 and 4. Also the control points (marked with circles) and the control polygons are plotted. The first degree curve and it's control polygon coincide.

The general expression for the Bernstein polynomial basis functions of degree d are

$$b_{i,d}(t) = \binom{d}{i} t^i (1-t)^{d-i}, \quad i = 0, 1, 2, \dots, d.$$

The name Bernstein polynomials were first used on a function were these polynomials were the basis functions.⁴ Today the set of basis functions are usually called the set of Bernstein polynomials. Below follows the 6 first set of Bernstein basis functions ($d = 0, 1, 2, 3, 4, 5$), one set on each row ($i = 0, \dots, d$),

$$\begin{array}{cccccc} 1 & & & & & \\ 1-t & & t & & & \\ (1-t)^2 & 2t(1-t) & t^2 & & & \\ (1-t)^3 & 3t(1-t)^2 & 3t^2(1-t) & t^3 & & \\ (1-t)^4 & 4t(1-t)^3 & 6t^2(1-t)^2 & 4t^3(1-t) & t^4 & \\ (1-t)^5 & 5t(1-t)^4 & 10t^2(1-t)^3 & 10t^3(1-t)^2 & 5t^4(1-t) & t^5 \end{array} \quad (4.29)$$

The notation is as following

$$\begin{aligned} & b_{0,0}(t) \\ & b_{0,1}(t) \quad b_{1,1}(t) \\ & b_{0,2}(t) \quad b_{1,2}(t) \quad b_{2,2}(t) \\ & b_{0,3}(t) \quad b_{1,3}(t) \quad b_{2,3}(t) \quad b_{3,3}(t) \end{aligned}$$

and so on

⁴The polynomial occurred as result of the work of Sergei Natanovich Bernstein, an Ukrainian mathematician (1880-1968). Bernstein introduced the polynomials in 1911 (published in [8]), using them for constructive proof of the Stone-Weierstrass approximation theorem. For a closer study of these polynomials see page 183–186 in [81] or page 108–126 in [33].

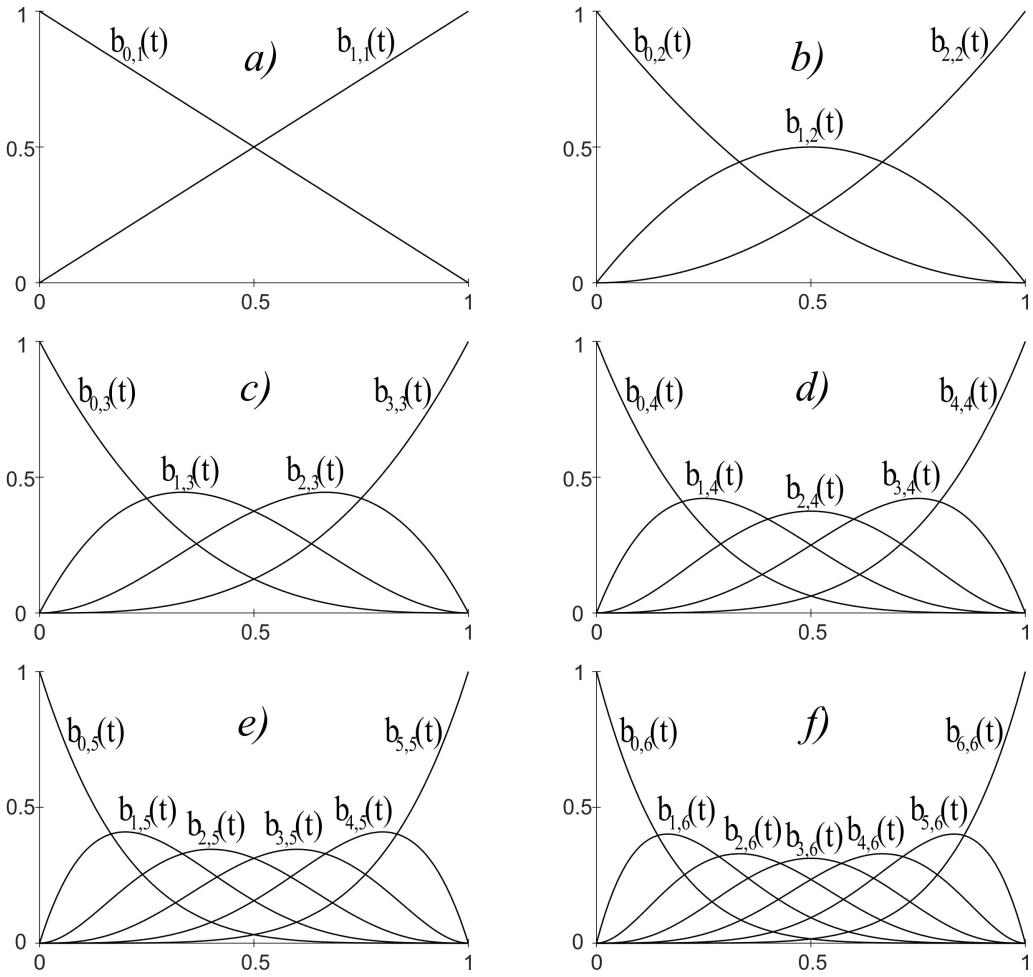


Figure 4.11: Six sets of Bernstein basis polynomials. **a)** is the set of degree one, forming first degree Bézier curves. **b)** is the set of degree two, **c)** is the set of degree three, **d)** is the set of degree four; **e)** is the set of degree five, **d)** is the set of degree six. All sets are basis functions for Bézier curves of the respective degree.

In figure 4.11 are the six sets of Bernstein polynomials of degree 1 to 6 plotted. These are the functions on each row of expression 4.29.

The algorithm for computing the sets of $d + 1$ Bernstein polynomials of degree d is recursive, and using the set of d Bernstein polynomials of degree $d - 1$.

1. First we compute the one on the left hand side, $b_{0,d}(t) = (1-t) b_{0,d-1}(t)$,
2. if $d > 0$ (we have more than one polynomial) we compute all internal polynomials in the row: for $i = 1, \dots, d - 1$, $b_{i,d}(t) = t b_{i-1,d-1}(t) + (1-t) b_{i,d-1}(t)$.
3. Finally we compute the one on the right hand side, $b_{d,d}(t) = t b_{d-1,d-1}(t)$.

We start with the function $b_{0,0}(t) = 1$ (a polynomial of degree 0). We then compute the next set, the degree $d = 1$ based on $b_{0,0}(t)$ and then the next, and so on. The algorithm is also called a pyramid algorithm (see [66]). Another view of the algorithm will be shown in the next section.

Algorithm 1. (For notation, see section “Algorithmic Language”, page 18.)

The algorithm computes a vector $\mathbf{b}_d(t) \in \mathbb{R}^{d+1}$, containing the values of the $d + 1$ Bernstein polynomials $\{b_{d,i}(t)\}_{i=0}^d$. The input variables are: the degree d of the Bernstein polynomials and the parameter value $t \in [0, 1]$.

```
Vector<double> bernstein( int d, double t )
    Vector<double> b(d+1);           // The return vector, dimension d + 1.
    b[0] = 1;                      // A general Cox/deBoor like algorithm for
    for ( int i=1; i <= d; i++ )    // - Bézier/Bernstein computing the set
        b[i] = t * b[i-1];          // - of “Bernstein” polynomial of degree d.
        for ( int j=i-1; j > 0; j-- )
            b[j] = t * b[j-1] + (1-t) * b[j];
        b[0] = (1-t) * b[0];
    return b;
```

Lemma 4.1. The Bernstein polynomials of degree d form a basis-set for polynomial functions of degree d and lower, on the domain $[0, 1]$.

Proof. This is the same as answering the following question; are the functions in the set linearly independent of each other or can we get one of them as a linear combination of the other?

The answer is yes, they are linearly independent. This can be shown in the following way, the argument is clearly illustrated in the pyramid (4.29):

- The function on the left hand side is the only one with a constant as a term. It is obvious linearly independent from the other.
- the next function from left hand side is beside the first one the only one with a term that is of first degree. It is obvious independent from the rest of the functions on right hand side.
- The same argument can be used for all other functions only raising the degree by 1 for each function.

□

Lemma 4.2. The set of Bernstein polynomials of degree d sums up to 1 for all d . We call this property; to form a partition of unity on $[0, 1]$, and it is expressed by

$$\sum_{i=0}^d b_{i,d}(t) = 1, \quad \text{for } d = 1, 2, 3, \dots$$

Proof. We start with the first function (on the first row) $b_{0,0}(t) = 1$ that obvious sum up to 1. The next row, $b_{0,1}(t) + b_{1,1}(t) = (1-t) + t = 1$ i.e. also sum up to 1. If we now look at the sum of a set (a row) of degree d defined by the recursive algorithm on the previous page, we get

$$s_d(t) = (1-t) b_{0,d-1}(t) + \sum_{i=1}^{d-1} (t b_{i-1,d-1}(t) + (1-t) b_{i,d-1}(t)) + t b_{d-1,d-1}(t).$$

If we reorganize this we get,

$$s_d(t) = \sum_{i=0}^{d-1} b_{i,d-1}(t) = s_{d-1}(t)$$

which shows that the sum of a row is equal the sum of the row above. By induction this shows that a set of Bernstein basis functions for all degrees sum up to 1. \square

4.4.2 Factorization and de Casteljau's Corner cutting algorithm

To investigate algorithm for Bernstein polynomials and Bézier curves closer, we start with linear interpolation between two points c_0 and c_1 (in the plane or in space).

$$c(t) = (1-t) c_0 + t c_1.$$

For $t \in [0, 1]$ we get the line segment between c_0 and c_1 .

Given a sequence of points c_0, c_1, c_2 and c_3 . If we for a given $t \in [0, 1]$ linear interpolate between pairs of two and two points, c_0 and c_1 , c_1 and c_2 , c_2 and c_3 , we will get three new points, which we call respectively c_{01} , c_{12} and c_{23} . In matrix, vector notation we have done the following,

$$\begin{pmatrix} c_{01} \\ c_{12} \\ c_{23} \end{pmatrix} = \begin{pmatrix} 1-t & t & 0 & 0 \\ 0 & 1-t & t & 0 \\ 0 & 0 & 1-t & t \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix}.$$

If we repeat the process with the same t we get

$$\begin{pmatrix} c_{012} \\ c_{123} \end{pmatrix} = \begin{pmatrix} 1-t & t & 0 \\ 0 & 1-t & t \end{pmatrix} \begin{pmatrix} c_{01} \\ c_{12} \\ c_{23} \end{pmatrix},$$

and finally

$$c_{0123} = (1-t) \begin{pmatrix} c_{012} \\ c_{123} \end{pmatrix}.$$

This process is called de Casteljau's⁵ (corner cuttings) algorithm for evaluating Bézier curves (see [38] and [39]), and it is illustrated in Figure 4.12.

If we connect these three multiplications, we get the following equation for a third degree Bézier curve,

$$c(t) = (1-t) \begin{pmatrix} 1-t & t & 0 \\ 0 & 1-t & t \end{pmatrix} \begin{pmatrix} 1-t & t & 0 & 0 \\ 0 & 1-t & t & 0 \\ 0 & 0 & 1-t & t \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix}. \quad (4.30)$$

⁵Paul de Casteljau (1930 -) is a French physicist and mathematician. In 1959, while working at Citroën, he developed an algorithm for evaluating calculations on what later was named Bézier curves because he was not permitted to publish his early work. We also call the multilinear polynomials "blossoming",

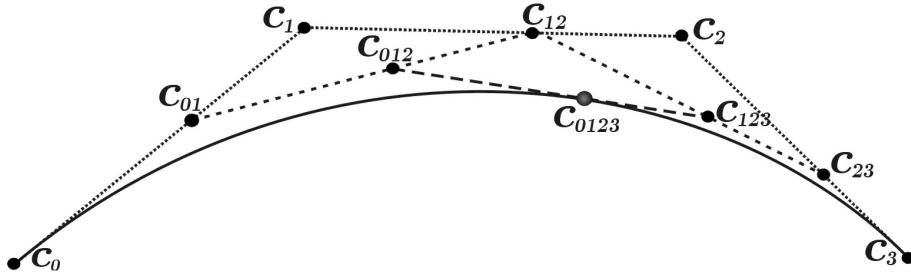


Figure 4.12: A Bézier curve (bold) and the points (bullets) and lines (dotted) illustrating the de Casteljau's algorithm for a third degree Bézier curve at $t = 0.6$.

These formulas, computed from the right side, are the de Casteljau (corner cuttings) algorithm. If we only compute the three matrices from the left we get a vector with the four Bernstein polynomials of degree three,

$$((1-t)^3, \quad 3(1-t)^2t, \quad 3(1-t)t^2, \quad t^3).$$

If we denote these Bernstein factor matrices for $T(t)$, and add an index d on it, for example, if $d = 2$, it is

$$T_2(t) = \begin{pmatrix} 1-t & t & 0 \\ 0 & 1-t & t \end{pmatrix}.$$

We now have the following general definition.

Definition 4.4. $T_d(t)$ is a matrix with dimension $d \times (d+1)$ where all lines numbered j

- i) only have element number j and $j+1$ different from zero.
- ii) where (in the Bézier case) element number j is $1-t$ and element $j+1$ is t .

Note that in the Bézier case we use t . Later we will see that in B-splines we will use a translation and scaling function $w_{d,i} : [t_i, t_{i+d}] \rightarrow [0, 1]$ instead of t , as described in chapter 5, expression (5.45) on page 119.

We also define the matrix notation of the set of Bernstein polynomials of degree d to be

$$B^d(t) = T_1(t)T_2(t)\cdots T_d(t).$$

It follows that equation (4.30) can be expressed as

$$c(t) = T_1(t)T_2(t)T_3(t) \mathbf{c} = B^3(t) \mathbf{c},$$

where $\mathbf{c} = (c_0, c_1, c_2, c_3)^T$ is a vector of points. In traditional notation this will be

$$c(t) = \sum_{i=0}^3 b_{i,d}(t) c_i.$$

Notice that the derivative of the matrix $T_d(t)$ is a matrix of constants. We, therefor, denote it T'_d . For $d = 1$ we get

$$T'_1 = (-1, \quad 1).$$

We can now look at a matrix version of a Bézier curve as described in expression (4.30). In the following equations we will see a 3rd degree Bézier curve and its three derivatives in matrix form.

$$\begin{aligned} c(t) &= B^3(t) C = T_1(t)T_2(t)T_3(t) C, \\ c'(t) &= 3 B^2(t)B' C = 3 T_1(t)T_2(t) T'_3 C, \\ c''(t) &= 6 B^1(t)B'^2 C = 6 T_1(t) T'_2 T'_3 C, \\ c'''(t) &= 6 B'^3 C = 6 T'_1 T'_2 T'_3 C. \end{aligned} \quad (4.31)$$

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 . The equations below also depends on the Commutativity relations between $T(t)$ matrices and their derivatives. It is that $T'_1 T_2(t) = T_1(t) T'_2$, and that $c'(t) = T'_1 T_2(t) T_3(t) + T_1(t) T'_2 T_3(t) + T_1(t) T_2(t) T'_3 C$. Expanding (4.31), we get,

$$\begin{aligned} c(t) &= \begin{pmatrix} 1-t & t \end{pmatrix} \begin{pmatrix} 1-t & t & 0 \\ 0 & 1-t & t \end{pmatrix} \begin{pmatrix} 1-t & t & 0 & 0 \\ 0 & 1-t & t & 0 \\ 0 & 0 & 1-t & t \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{pmatrix}, \\ c'(t) &= 3 \begin{pmatrix} 1-t & t \end{pmatrix} \begin{pmatrix} 1-t & t & 0 \\ 0 & 1-t & t \end{pmatrix} \begin{pmatrix} -1 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & -1 & 1 \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{pmatrix}, \\ c''(t) &= 6 \begin{pmatrix} 1-t & t \end{pmatrix} \begin{pmatrix} -1 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & -1 & 1 \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{pmatrix}, \\ c'''(t) &= 6 \begin{pmatrix} -1 & 1 & 0 \\ 0 & -1 & 1 \end{pmatrix} \begin{pmatrix} -1 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & -1 & 1 \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{pmatrix}. \end{aligned} \quad (4.32)$$

Now we can clearly see that:

- i) If we compute from the right hand side (skipping the zeros), we get the de Casteljau corner cutting 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.⁶

⁶In section 5.5, is the same matrix notation used on B-splines. This easily gives us both the Cox/de Boor algorithm for B-splines, 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 also discussed in [99].

4.4.3 The Bernstein/Hermite matrix

In general, Bézier curves are defined by

$$c(t) = \sum_{i=0}^d c_i b_{i,d}(t), \quad \text{if } 0 \leq t \leq 1, \quad (4.33)$$

where the basis functions are the Bernstein polynomials and where $c_i \in \mathbb{R}^n$, $i = 0, 1, \dots, d$, 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 (4.33) used for Bézier curves,

- i) de Casteljau algorithm,
- ii) a Cox-de'Boor recursion like 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 at a specific matrix for use in both evaluations and in Hermite interpolation for Bézier curves.

The Generalized Bernstein/Hermite matrix will be introduced (interpolation matrices are discussed amongst others in [124]) for use in both Hermite interpolation and in evaluation (both will be discussed later)

$$\mathbf{B}_d(t, \delta) = \begin{pmatrix} b_{0,d}(t) & b_{1,d}(t) & \dots & b_{d,d}(t) \\ \delta D b_{0,d}(t) & \delta D b_{1,d}(t) & \dots & \delta D b_{d,d}(t) \\ \vdots & \vdots & \ddots & \vdots \\ \delta^d D^d b_{0,d}(t) & \delta^d D^d b_{1,d}(t) & \dots & \delta^d D^d b_{d,d}(t) \end{pmatrix}. \quad (4.34)$$

$$\mathbf{B}_2(t) = \begin{pmatrix} b_{0,3}(t) & b_{1,3}(t) & b_{2,3}(t) & b_{3,3}(t) \\ D b_{0,3}(t) & D b_{1,3}(t) & D b_{2,3}(t) & D b_{3,3}(t) \\ D^2 b_{0,3}(t) & D^2 b_{1,3}(t) & D^2 b_{2,3}(t) & D^2 b_{3,3}(t) \\ D^3 b_{0,3}(t) & D^3 b_{1,3}(t) & D^3 b_{2,3}(t) & D^3 b_{3,3}(t) \end{pmatrix} = \begin{pmatrix} (1-t)^3 & 3t(1-t)^2 & 3t^2(1-t) \\ -3(1-t)^2 & 9t^2 - 12t + 3 & -3t(3t-2) \\ 6-6t & 18t-12 & 6-18t \\ -6 & 18 & -18 \end{pmatrix} \quad (4.35)$$

The special issue about this matrix (8.8) is the scaling δ^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.

In section 7.9, the Hermite interpolation properties are discussed. (On page 385 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 (7.53) in definition 7.7), then in order to compute, for instance, the local Bézier curve $c_i(t)$, the j th derivatives actually have to be scaled by the global/local “scaling factor” δ_i^j where

$$\delta_i = \frac{1}{t_{i+1} - t_{i-1}}, \quad (4.36)$$

as described in Theorem 7.4. The numerator in the fraction is 1 because the domain of Bézier curves is $[0, 1]$. Because the matrix (8.8) 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 (8.8).

Algorithm 2. (*For notation, see section “Algorithmic Language”, page 18.*)

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, $\{D^j b_{d,i}(t)\}_{i=0}^d$, $j = 1, 2, \dots, d$, 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 δ^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 δ_i , see (8.11), 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 δ .

```
Matrix<double> mat ( int d, double t, double δ )
    Matrix<double> B(d+1,d+1);      // The return matrix, dimension  $(d + 1) \times (d + 1)$ .
    Bd-1,0 = 1 - t;
    Bd-1,1 = t;                  // The general Cox/deBoor like algorithm for
    for ( int i=d-2; i ≥ 0; i -- ) // - Bézier/Bernstein computing the triangle
        Bi,0 = (1 - t) Bi+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.
            Bi,j = t Bi+1,j-1 + (1 - t) Bi+1,j;
        Bi,d-i = t Bi+1,d-i-1;

    Bd,0 = -δ;
    Bd,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 (8.10), and the scalings,
        for ( int i = d; i > d - k; i -- ) // - so every row extends the number
            Bi,k = s Bi,k-1;    // - of elements to d.
            for ( int j = k - 1; j > 0; j -- )
                Bi,j = s (Bi,j-1 - Bi,j);
        Bi,0 = -s Bi,0;
    return B;
```

The order of this algorithm can clearly be seen to be an improved $O(n^3)$, and the follow-

ing 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.

```

1 void BernsteinHermite(Matrix<double>& B, int d, double t, double scale)
2 {
3     if( d < 1 ) {
4         B.setDim( 1, 1 );
5         B[0][0] = 1;
6         return; // Escape in the case of degree zero
7     }
8     else
9         B.setDim( d+1, d+1 ); // Initiate the result matrix
10
11    // Compute all Bernstein polynomials up to degree d.
12    // Each degree at each row, starting from the bottom up.
13    B[d-1][0] = 1 - t;
14    B[d-1][1] = t;
15
16    for( int i = d-2; i >= 0; i-- ) {
17        B[i][0] = ( 1 - t ) * B[i+1][0];
18        for( int j = 1; j < d - i; j++ )
19            B[i][j] = t * B[i+1][j-1] + ( 1 - t ) * B[i+1][j];
20        B[i][d-i] = t * B[i+1][d-i-1];
21    }
22
23    // Compute all the deriatives that exist
24    B[d][0] = -scale;
25    B[d][1] = scale;
26
27    for( int k = 2; k <= d; k++ ) {
28        const double s = k * scale;
29        for( int i = d; i > d - k; i-- ) {
30            B[i][k] = s * B[i][k-1];
31            for( int j = k - 1; j > 0; j-- )
32                B[i][j] = s * ( B[i][j-1] - B[i][j] );
33            B[i][0] = - s * B[i][0];
34        }
35    }
36 }
```


Chapter 5

Spline Curves

The develop of splines has been the most important factor in computer aided geometric design. In this chapter we will look at the develop of splines from historical times to modern spline theory. The first section in this chapter will, therefore, give a short introduction to the early history of interpolation, and will then introduce Hermite interpolation, Hermite spline interpolation and cubic spline interpolation. The fourth section will give an historical introduction to B-splines, and introduce us to what we can call modern B-splines. One of the fundamental constructions in ERBS is blending geometry. The sixth section will, therefore, show some of the history of blending geometry and some consequences. The last section will focus on the concept of using maps and atlas. In ERBS it is the use of knot vector and global parameterizations to organize the “global/local mapping” (diffeomorphism) that blends (glues) the local geometry to get a global manifold, and this is one of the challenges in implementing ERBS. However, this is also one of the things that makes a closer connection between CAGD and classical Differential geometry, and it actually makes it possible to implement parameterized surfaces of different genus.

5.1 History of interpolation

In his book on interpolation, Thiele characterized the subject (interpolation) as *the art of reading between lines in a numerical table* (see [139]). Until the introduction of Computers and especially Computer graphics this was a very precise formulation. For a long time the driving force in developing interpolation technics was Astronomy.

We start in Mesopotamia and Babylon.¹ Astronomy was about time keeping and making predictions about astronomical events. This, of course, served practical needs for living, for example, farmers would base their strategies on these predictions. To this end it was

¹This section, 5.1 (and parts of 5.4) is mainly based on (and partly quoted from) an article by Erik Meijering “A Chronology of Interpolation” (see [108]) supported by information from the Chinese Electronic Periodical service <http://www.ceps.com.tw/ec/echome.aspx> and “A Concise History of Mathematics” by Dirk J. Struik [136]. Some of the references to the source material can be found here in this book, but a lot more references can be found in the article by Erik Meijering.

of great importance to keep up to date lists (so-called ephemerides) of the positions of the sun, moon, and the known planets for regular time intervals. Obviously these lists would contain gaps, due to either atmospherical conditions hampering observation or the fact that celestial bodies may not be visible during certain periods. The historian-mathematician Neugebauer (see [111]) has made a study of ephemerides found on ancient astronomical cuneiform tablets originating from Uruk and Babylon in the Seleucid period (the last three centuries BC). He concluded that interpolation was used in order to fill these gaps. Linear interpolation has been in use, but the tablets also revealed the use of more complex interpolation methods. Precise formulations of the latter methods have, unfortunately, not survived.

In the same period we can find an early example of the use of interpolation methods in ancient Greece. Toomer (see [140]) believes that Hipparchus of Rhodes (190–120 BC) used linear interpolation in the construction of tables of the so-called “chord function” (related to the sine function) for the purpose of computing the positions of celestial bodies. We can also find later examples of interpolation in the Almagest (“The Mathematical Compilation”, approx. 140 AD) by Claudius Ptolemy (the Egypt-born Greek astronomer-mathematician who propounded the geocentric view of the universe which prevailed until the 16th century). Apart from theory, this work also contains numerical tables of a wide variety of trigonometric functions defined for astronomical purposes. To avoid the tedious calculations that were involved in the construction of tables of functions of more than one variable, Ptolemy used an approach that implies:

- (i) only tabulating the function for the variable for which the function varies most (given two bounding values of the other variable),
- (ii) providing a table of coefficients to be used in an “adaptive” linear interpolation scheme for computation of the function for intermediate values of this latter variable (see [141]).

Analysis of the computational techniques that early-medieval Chinese ephemerides are based on, reveals the use of higher-order interpolation formulae. The astronomer Liu Zhuo (544–610 AD) was probably the first person to use quadratic interpolation for computing the positions of the sun and the moon whilst constructing a calendar. Around 600 AD he used this technique to produce the so-called “Huangji li”, or “Imperial Standard Calendar”. According to Li Yan & Du Shiran (see [154]), the formula involved in his computations was (in modern notation):

$$f(x_0 + \xi\Delta t) = f(x_0) + \frac{\xi}{2}(\Delta f_1 + \Delta f_2) + \xi(\Delta f_1 - \Delta f_2) - \frac{\xi^2}{2}(\Delta f_1 - \Delta f_2), \quad (5.1)$$

where $0 < \xi < 1$, $\Delta t > 0$, $\Delta f_1 = f(x_0 + \Delta t) - f(x_0)$, and $\Delta f_2 = f(x_0 + 2\Delta t) - f(x_0 - \Delta t)$, and where $f(x_0)$, $f(x_0 + \Delta t)$, and $f(x_0 + 2\Delta t)$ are the observed results at times x_0 , $x_0 + \Delta t$, and $x_0 + 2\Delta t$, respectively. This formula is closely related to later so-called *classical “European” interpolation formulae* (shown later in this section). Methods for second-order interpolation of non-uniform interval observations were later used by the astronomer Monk Yixing (683–727 AD) whilst producing the so-called “Dayan Li” calendar (724 AD) and by Xu Ang whilst producing the “Xuan Ming” calendar (822 AD). The latter also used a second-order formula for interpolation of uniform interval observations equivalent to the formula used by Liu Zhuo. Accurate computation of the motion of celestial bodies, however, requires more sophisticated interpolation techniques than just second order.

More complex techniques were later developed by Guo Shoujing and others. In 1280 AD they produced the so-called “Shoushi li”, or “Works and Days Calendar”, for which they used third-order interpolation. Although they did not explicitly write down third-order interpolation formulae, it follows from their computations recorded in tables that they had grasped the principle. Important contributions in the area of finite-difference computation were made by the Chinese mathematician Zhu Shijie. In his book *Siyuan yujian* (“Jade Mirror of the Four Origins”, 1303 AD), he gave the following problem (quoted freely from Martzloff, [101]):

Soldiers are recruited in cubes. On the first day, the side of the cube is three. On the following days, it is one more per day. At present, it is fifteen. Each soldier receives two hundred and fifty guan per day. What is the number of soldiers recruited and what is the total amount paid out?

In explaining the answer to the first question, Zhu Shijie gives a sequence of verbal instructions (a “resolutory rule”) for finding the solution, which, when cast in modern algebraic notation, suggests using the following formula:

$$\begin{aligned} f(n) = n\Delta_0 + \frac{1}{2!}n(n-1)\Delta_0^2 \\ + \frac{1}{3!}n(n-1)(n-2)\Delta_0^3 \\ + \frac{1}{4!}n(n-1)(n-2)(n-3)\Delta_0^4, \end{aligned} \quad (5.2)$$

where $f(n)$ is the total number of soldiers recruited in n days and the differences are defined by $\Delta_j = f(j+1) - f(j)$ and $\Delta_j^i = \Delta_{j+1}^{i-1} - \Delta_j^{i-1}$, where $i > 1$ and $j > 0$ are integers. Although the specific problem requires only differences up to the fourth order, the proposed formula to solve it can easily be generalized to any arbitrary degree, and has close connections with later “European” interpolation formulae which are discussed later in the section.

In India, work on higher-order interpolation started around the same time as in China. In his work “Dhyanagraha” (approx. 625 AD), the astronomer-mathematician Brahmagupta (589-668 AD) included a passage in which he proposed a method for second-order interpolation of the sine “($\sin x$)” and versed sine “($1 - \cos x$)” functions. Rephrasing the original Sanskrit text in algebraic language, Gupta (see [75]) arrived at the following formula:

$$\begin{aligned} f(x_0 + \xi\Delta t) = f(x_0) + \frac{\xi}{2}(\Delta f(x_0 - \Delta t) + \Delta f(x_0)) \\ + \frac{\xi^2}{2}(\Delta f(x_0) - \Delta f(x_0 - \Delta t)), \end{aligned} \quad (5.3)$$

where $\Delta f(x_0) = f(x_0 + \Delta t) - f(x_0)$. In a later work, Khandakhadyaka (665 AD), Brahmagupta also described a more general method that allowed for interpolation of unequal-interval data. In the case of equal intervals, this method reduces to (5.3). Another rule for making second-order interpolations can be found in a commentary in the seventh-century

work *Mahabhaskariya* by Bhaskara I, ascribed to Govindasvami (approx. 800-850 AD). From [75], expressed in algebraic notation, it is:

$$f(x_0 + \xi\Delta t) = f(x_0) + \xi\Delta f(x_0) + \frac{\xi(\xi-1)}{2} (\Delta f(x_0) - \Delta f(x_0 - \Delta t)), \quad (5.4)$$

It is not difficult to see that this formula (5.4) is equivalent to (5.3). According to Gupta [75], this formula (rule) is also found in two early 15th-century commentaries by Paramesvara (1360-1425 AD).

Use of these described second-order interpolation formulae amounts to fitting a parabola through three consecutive tabular values. Kennedy mentions (in [86]) that parabolic interpolation schemes are also found in several Arabic and Persian sources. Noteworthy are the works “al-Qanun al-Masudi” (“Canon Masudicus”, 11th century) by al-Biruni (973-1048 AD), and “Zij-i- Khaqani” (early 15th century) by al-Kashi (1390-1450 AD). As regards the parabolic interpolation methods described therein, Gupta has in [75] pointed at possible Indian influences, since the important works of Brahmagupta were translated into Arabic as early as the eighth century AD. Not to mention the fact that al-Biruni himself traveled through and resided in several parts of India, studied original Indian literature, wrote a book about India, and translated several Sanskrit texts into Arabic (see [86]).

In Europe, apparently unaware of the important results obtained much earlier in other parts of the world, interpolation theory started to develop after the great “scientific revolution” started about 1500 AD. In particular the new developments in astronomy and physics, initiated by Copernicus, continued by Kepler and Galileo, and culminating in the theories of Newton, gave strong impetus to the further advancement of mathematics, including what now is called “classical” interpolation theory.

Before reviewing some of the classical interpolation formulae, we shall first study one of the better known formulae. Suppose that we are given measurements of some quantity at $\dots, x_0, x_0 + \Delta t, x_0 + 2\Delta t, \dots$, and that in order to obtain its value at any intermediate point $x_0 + \xi\Delta t$, $\xi \in \mathbb{R}$, we locally model it as a polynomial $f : \mathbb{R} \rightarrow \mathbb{R}$ of given degree $n \in \mathbb{N}$, that is,

$$f(x_0 + \xi\Delta t) = a_0 + a_1\xi + a_2\xi^2 + \dots + a_n\xi^n, \quad \text{for some } \{a_i\}_{i=0}^n.$$

It is easy to show (see ex. [150]) that any such polynomial can be written in terms of factorials

$$[\xi]^k = \xi(\xi-1)(\xi-2)\cdots(\xi-k+1), \quad \text{where } k \in \mathbb{N}, \quad k > 0,$$

as

$$f(x_0 + \xi\Delta t) = c_0 + c_1[\xi] + c_2[\xi]^2 + \dots + c_n[\xi]^n \quad \text{for some } \{c_i\}_{i=0}^n. \quad (5.5)$$

If we now define the first-order difference of any function $\phi : \mathbb{R} \rightarrow \mathbb{R}$ at any ξ as

$$\Delta\phi(\xi) = \phi(\xi+1) - \phi(\xi),$$

and similarly the higher-order differences as

$$\Delta^p\phi(\xi) = \Delta^{p-1}\phi(\xi+1) - \Delta^{p-1}\phi(\xi), \quad \text{for all } p \in \mathbb{N}, \quad p > 1, \quad (5.6)$$

it follows that $\Delta[\xi]^k = k[\xi]^{k-1}$. By repeated application of the difference operator Δ to the factorial representation of $f(x_0 + \xi\Delta t)$, and taking $\xi = 0$, we find that the coefficients c_k , $k = 0, 1, \dots, n$ (from (5.5)), can be expressed as $c_k = \Delta^k f(x_0)/k!$, so that if n could be made arbitrarily large, we would have:

$$\begin{aligned} f(x_0 + \xi\Delta t) &= f(x_0) + \xi\Delta f(x_0) \\ &\quad + \frac{1}{2!}\xi(\xi - 1)\Delta^2 f(x_0) \\ &\quad + \frac{1}{3!}\xi(\xi - 1)(\xi - 2)\Delta^3 f(x_0) \\ &\quad + \frac{1}{4!}\xi(\xi - 1)(\xi - 2)(\xi - 3)\Delta^4 f(x_0) + \dots \end{aligned} \tag{5.7}$$

This general formula was first written down in 1670 by Gregory and can be found in a letter by him to Collins [70]. Particular cases of it, however, had been published several decades earlier by Briggs, who brought to fruition the work of Napier on logarithms. In the introductory chapters to one of his major works [15] he described the precise rules by which he carried out his computations, including interpolations, for constructing the tables contained therein. For example, he described a subtabulation rule that, when written in algebraic notation, amounts to (5.7) for the case when third- and higher-order differences are negligible (see [67]). It is known [67] that still earlier, around 1611, Harriot used a formula equivalent to (5.7) up to fifth-order differences. But even he was not the first in the world to write such rules down. It is not difficult to see that the right-hand side of the second-order interpolation formula used by Liu Zhuo, (5.1), can be rewritten so that it equals the first three terms of (5.7). And if we replace the integer argument n by the real variable x in Zhu Shijie's formula, (5.2), we obtain at once (5.7) for the case when $x_0 = 0$, $f(0) = 0$, and $\Delta t = 1$.

Notwithstanding these facts, it is justified to say that “there is no single person who did so much for this field, as for so many others, as Newton” (quotation from [67]). His enthusiasm becomes clear in a letter he wrote to Oldenburg [114], where he first describes a method by which certain functions may be expressed in series of powers of x , and then goes on to say:

But I attach little importance to this method because when simple series are not obtainable with sufficient ease, I have another method not yet published by which the problem is easily dealt with. It is based upon a convenient, ready and general solution of this problem. To describe a geometrical curve which shall pass through any given points. (...) And although it may seem to be intractable at first sight, it is nevertheless quite the contrary. Perhaps indeed it is one of the prettiest problems that I can ever hope to solve.

The contributions of Newton to the subject are contained in

- i) a letter [113] to Smith in 1675,
- ii) a manuscript entitled *Methodus Differentialis*, [112], published in 1711, although earlier versions were probably written in the mid-1670s,
- iii) a manuscript entitled *Regula Differentiarum*, written in 1676 but first discovered and published in the 20th century, [59], and

- iv) Lemma V in Book III of his celebrated *Principia*, [115], which appeared in 1687. The latter was published first and contains two formulae. The first deals with equal-interval data and is precisely (5.7), which Newton seems to have discovered independently of Gregory. The second formula deals with the more general case of arbitrary-interval data and may be derived as follows. Suppose that the values of the aforementioned quantity are given at $\dots, x_0, x_1, x_2, \dots$, which may be arbitrary, and that in order to obtain its value at intermediate points we model it again as a polynomial function $f : \mathbb{R} \rightarrow \mathbb{R}$. If we then define the first-order divided difference² of any function $\phi : \mathbb{R} \rightarrow \mathbb{R}$ for any two $\xi_0 \neq \xi_1$ as

$$\phi[\xi_0, \xi_1] = (\phi(\xi_0) - \phi(\xi_1)) / (\xi_0 - \xi_1), \quad (5.8)$$

it follows that the value of f at any $x \in \mathbb{R}$ could be written as

$$f(x) = f(x_0) + (x - x_0)f[x, x_0].$$

And if we define higher-order divided differences as

$$\phi[\xi_0, \dots, \xi_p] = (\phi[\xi_0, \dots, \xi_{p-1}] - \phi[\xi_1, \dots, \xi_p]) / (\xi_0 - \xi_p), \quad \text{for all } p > 1, \quad (5.9)$$

we can substitute for $f[x, x_0]$ the expression that follows from the definition of $f[x, x_0, x_1]$, and subsequently for $f[x, x_0, x_1]$ the expression that follows from the definition of $f[x, x_0, x_1, x_2]$, etc., so that if we could go on, we would have

$$\begin{aligned} f(x) = & f(x_0) + (x - x_0)f[x_0, x_1] \\ & + (x - x_0)(x - x_1)f[x_0, x_1, x_2] \\ & + (x - x_0)(x - x_1)(x - x_2)f[x_0, x_1, x_2, x_3] + \dots \end{aligned} \quad (5.10)$$

It is this formula that can be considered the most general of all classical interpolation formulae. As we will see in the sequel, all later formulae can easily be derived from it.

A very elegant alternative representation of Newton's general formula (5.14) that does not require the computation of finite or divided differences was published in 1779 by Waring [145], it is:

$$\begin{aligned} f(x) = & f(x_0) \frac{(x - x_1)(x - x_2)(x - x_3) \dots}{(x_0 - x_1)(x_0 - x_2)(x_0 - x_3) \dots} + \\ & f(x_1) \frac{(x - x_0)(x - x_2)(x - x_3) \dots}{(x_1 - x_0)(x_1 - x_2)(x_1 - x_3) \dots} + \\ & f(x_2) \frac{(x - x_0)(x - x_1)(x - x_3) \dots}{(x_2 - x_0)(x_2 - x_1)(x_2 - x_3) \dots} + \end{aligned} \quad (5.11)$$

It is nowadays usually attributed to Lagrange, who, in apparent ignorance of Waring's paper, published it 16 years later [88]. The formula (5.11) may also be obtained from a

²Divided differences can be studied more deeply in i.g [24] or [33].

closely related representation of Newton's formula due to Euler [51]. According to Joffe [84], it was Gauss who first noticed the logical connection and proved the equivalence of the formulae by Newton, Euler, and Waring-Lagrange, as appears from his posthumous works [63], although Gauss did not refer to his predecessors.

By the beginning of the 20th century, the problem of interpolation by finite or divided differences had been studied by astronomers, mathematicians, statisticians, and actuaries, and most of the now well-known variants of Newton's original formulae had been worked out. This is not to say, however, that there are no more advanced developments to report on. Quite the contrary. Already in 1821, Cauchy [21] studied interpolation by means of a ratio of two polynomials and showed that the solution to this problem is unique, the Waring-Lagrange formula being the special case for the second polynomial equal to one. Generalizations for solving the problem of multivariate interpolation in the case of fairly arbitrary point configurations began to appear in the second half of the 19th century, in the works of Borchardt and Kronecker (see [62]). A generalization of a different nature was published in 1878 by Hermite [78], who studied and solved the problem of finding a polynomial of which also the first few derivatives assume pre-specified values at given points, where the order of the highest derivative may differ from point to point. In a paper [12] published in 1906, Birkhoff studied the even more general problem: Given any set of points, find a polynomial function that satisfies pre-specified criteria concerning its value and/or the value of any of its derivatives for each individual point. Hermite and Birkhoff type of interpolation problems (and their multivariate versions, not necessarily on Cartesian grids) have received much attention in the past decades. A more detailed process follows in the next section.

5.1.1 Divided differences and Newton polynomial

We first define the first-order divided difference³ of any function $f : \mathbb{R} \rightarrow \mathbb{R}^p$ for any two $t_0 \neq t_1$ as

$$f[t_0, t_1] = \frac{f(t_0) - f(t_1)}{t_0 - t_1}. \quad (5.12)$$

Higher-order divided differences are then defined recursively,

$$f[t_0, \dots, t_n] = \frac{f[t_0, \dots, t_{n-1}] - f[t_1, \dots, t_n]}{t_0 - t_n}, \quad \text{for all } n > 1. \quad (5.13)$$

it follows that the value of f at any $t \in I \subset \mathbb{R}$ can be written as

$$f(t) = f(t_0) + (t - t_0)f[t, t_0].$$

we can substitute for $f[t, t_0]$ the expression that follows from the definition of $f[t, t_0, t_1]$, and subsequently for $f[t, t_0, t_1]$ the expression that follows from the definition of $f[t, t_0, t_1, t_2]$,

³Divided differences can be studied more deeply in i.g [24] or [33].

etc., so that if we could go on, we would have

$$\begin{aligned} f(t) &= f(t_0) + (t - t_0)f[t_0, t_1] \\ &\quad + (t - t_0)(t - t_1)f[t_0, t_1, t_2] \\ &\quad + (t - t_0)(t - t_1)(t - t_2)f[t_0, t_1, t_2, t_3] + \dots \end{aligned} \tag{5.14}$$

It is this formula that can be considered the most general of all classical interpolation formulae. As we will see in the sequel, all later formulae can easily be derived from it.

Newton polynomials are used for polynomial interpolation. For a given set of distinct parameter values $\{t_j\}_{j=0}^d$ and corresponding points $\{p_j\}_{j=0}^d$, the set and Lagrange polynomials gives the polynomial of the least degree that at each value t_j fit the corresponding point p_j (i.e. the function interpolate all points).

The interpolating polynomial of the least degree is unique, however, and it is therefore more appropriate to speak of "the Lagrange form" of that unique polynomial rather than "the Lagrange interpolation polynomial," since the same polynomial can be arrived at through multiple methods. Although named after Joseph Louis Lagrange, it was first discovered in 1779 by Edward Waring and rediscovered in 1783 by Leonhard Euler.

The definition of divided differences can be reformulated to:

$$f[x_0, \dots, x_n] = \frac{f[x_0, \dots, x_{n-1}] - f[x_1, \dots, x_n]}{x_0 - x_n}, \tag{5.15}$$

$$f[x_0, \dots, x_n] = \sum_{k=0}^n \frac{f(x_k)}{\omega'(x_i)}, \tag{5.16}$$

where $\omega(x) = (x - x_0) \cdots (x - x_n)$.

$$\omega_n(x) = \prod_{k=0}^n (x - x_k) \tag{5.17}$$

It follows that the derivative is,

$$\omega'_n(x_i) = \prod_{\substack{k=0 \\ k \neq i}}^n (x_i - x_k), \tag{5.18}$$

5.1.2 Lagrange polynomial

Lagrange polynomials are used for polynomial interpolation. For a given set of distinct parameter values $\{t_j\}_{j=0}^d$ and corresponding points $\{p_j\}_{j=0}^d$, the set and Lagrange polynomials gives the polynomial of the least degree that at each value t_j fit the corresponding point p_j (i.e. the function interpolate all points).

The interpolating polynomial of the least degree is unique, however, and it is therefore more appropriate to speak of "the Lagrange form" of that unique polynomial rather than "the Lagrange interpolation polynomial," since the same polynomial can be arrived at through multiple methods. Although named after Joseph Louis Lagrange, it was first discovered in 1779 by Edward Waring and rediscovered in 1783 by Leonhard Euler.

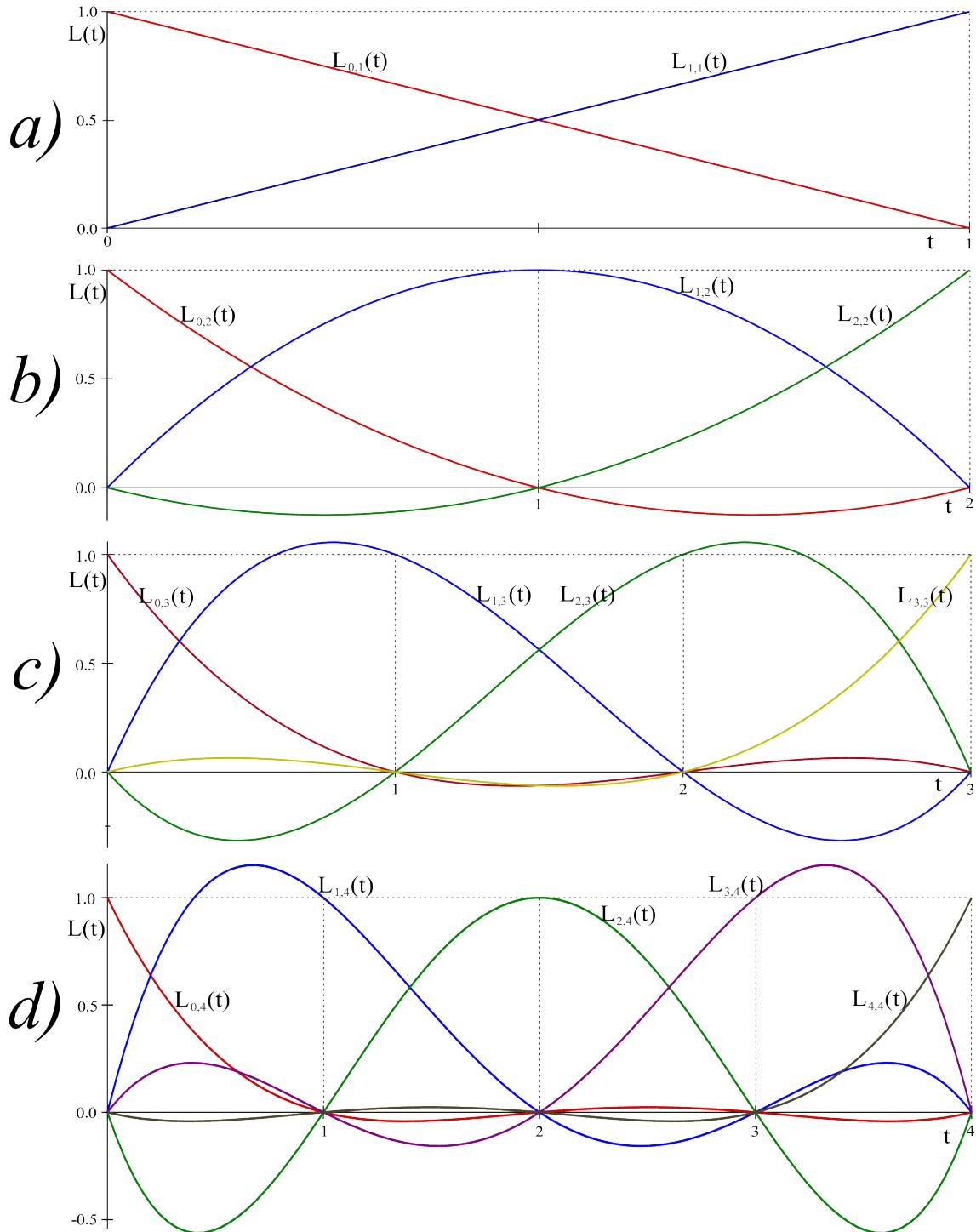


Figure 5.1: A plot of four sets of Lagrange interpolation polynomials. The set **a**) is the set of degree one, forming first degree curves. The set **b**) is the set of degree two, forming second degree curves. The set **c**) is the set of degree three, forming third degree curves. The set **d**) is the set of degree four, forming forth degree curves.

5.1.3 Hermite interpolation

Hermite interpolation is, actually, a method closely related to Newton classical interpolation formulae defined using divided differences (5.14). However, it allows us to consider given derivatives at data points, as well as the data points themselves. The interpolation will give a polynomial that has a degree equal to the total number of both data points and their derivatives, minus 1 (can also be less than this if the resulting polynomial is “degenerated”). We can treat the derivatives as extra points, and in the divided difference table, the points are repeated. To avoid division by zero, the values where the division by zero would take place are replaced with the derivatives, multiplied by a constant, depending on the order. For example, if a value t_i is repeated n times, we get

$$f[t_i, t_i, \dots, t_i] = \frac{f^{(n-1)}(t_i)}{(n-1)!}, \quad (5.19)$$

i.e. for $n = 4$ is, $f[t_i, t_i, t_i, t_i] = f^{(3)}(t_i)/3!$. (NB, divided differences can be vector valued.)

As an example, two points are given, $f(t_i)$ and $f(t_{i+1})$, $f : \mathbb{R} \rightarrow \mathbb{R}^d$, $d = 1, 2, 3, \dots$, and the respective derivatives at these two points $f'(t_i)$ and $f'(t_{i+1})$. We can now compute the following 3th degree polynomial by Hermite interpolating the two points and the respective derivatives using (5.14):⁴

$$\begin{aligned} c_i(t) = & f(t_i) + (t - t_i) f[t_i, t_i] \\ & + (t - t_i)(t - t_i) f[t_i, t_i, t_{i+1}] \\ & + (t - t_i)(t - t_i)(t - t_{i+1}) f[t_i, t_i, t_{i+1}, t_{i+1}] \end{aligned} \quad (5.20)$$

If we ”nest out” the divided differences, we get

$$\begin{aligned} c_i(t) = & f(t_i) + (t - t_i) f'(t_i) \\ & + (t - t_i)(t - t_i) \frac{f'(t_i) - f[t_i, t_{i+1}]}{t_i - t_{i+1}} \\ & + (t - t_i)(t - t_i)(t - t_{i+1}) \frac{\frac{f'(t_i) - f[t_i, t_{i+1}]}{t_i - t_{i+1}} - \frac{f[t_i, t_{i+1}] - f'(t_{i+1})}{t_i - t_{i+1}}}{t_i - t_{i+1}}, \end{aligned}$$

since $(t - t_{i+1}) = (t - t_i) + (t_i - t_{i+1})$ and if we use it to substitute the third factor in the third line, we get

$$\begin{aligned} c_i(t) = & f(t_i) + (t - t_i) f'(t_i) \\ & + (t - t_i)^2 \frac{2f'(t_i) + f'(t_{i+1}) - 3f[t_i, t_{i+1}]}{t_i - t_{i+1}} \\ & + (t - t_i)^3 \frac{f'(t_i) + f'(t_{i+1}) - 2f[t_i, t_{i+1}]}{(t_i - t_{i+1})^2}. \end{aligned} \quad (5.21)$$

We can also compare this general method with a simple method for this specific example. Given a 3th degree polynomial,

$$c_i(t) = a_{i,0} + a_{i,1}(t - t_i) + a_{i,2}(t - t_i)^2 + a_{i,3}(t - t_i)^3. \quad (5.22)$$

⁴The reason for using an index i at the curve $c_i(t)$ in (5.20) and the following expressions will be explained on page 106.

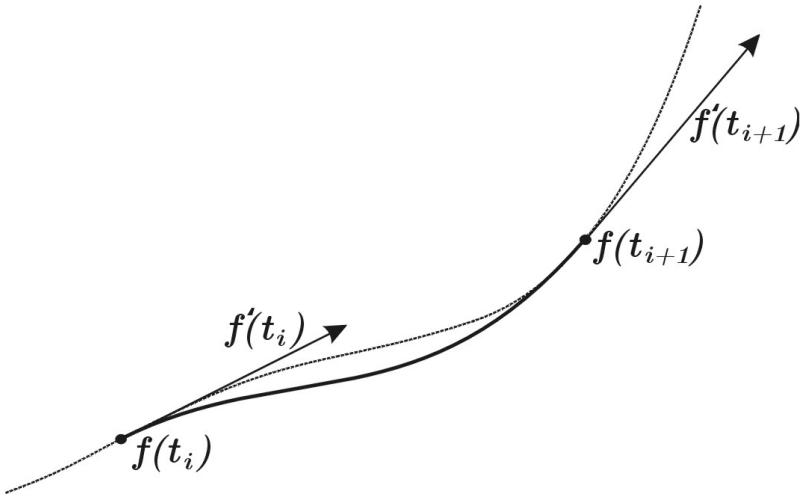


Figure 5.2: A curve $f(t)$ (dotted) and a 3th degree polynomial curve (bold) that is an Hermite interpolation of f at t_i and t_{i+1} .

Computing the derivative of this polynomial, we get

$$c'_i(t) = a_{i,1} + 2a_{i,2}(t - t_i) + 3a_{i,3}(t - t_i)^2. \quad (5.23)$$

If we use the following Hermite end conditions:

$$\begin{aligned} c_i(t_i) &= f(t_i), \\ c'_i(t_i) &= f'(t_i), \\ c_i(t_{i+1}) &= f(t_{i+1}), \\ c'_i(t_{i+1}) &= f'(t_{i+1}), \end{aligned}$$

where $f : \mathbb{R} \rightarrow \mathbb{R}^n$, $n = 1, 2, 3, \dots$ is a given curve. Then we can find the coefficients $\{a_{i,j}\}_{j=0}^3$ by substituting t with t_i and t_{i+1} in (5.22) and (5.23), and get the algebraic form:

$$\begin{aligned} a_{i,0} &= f(t_i), \\ a_{i,1} &= f'(t_i), \\ a_{i,2} &= \frac{3f[t_i, t_{i+1}] - 2f'(t_i) - f'(t_{i+1})}{\Delta t_i}, \\ a_{i,3} &= \frac{f'(t_i) + f'(t_{i+1}) - 2f[t_i, t_{i+1}]}{(\Delta t_i)^2}, \end{aligned}$$

where

$$\Delta t_i = t_{i+1} - t_i,$$

which, actually, are the same as in (5.21).

This is of course not the most convenient representation of a cubic polynomial from Hermite interpolation. The cubic polynomials can be reformulated to the geometric form:

$$c_i(t) = f(t_i)H_{i,3,0}(t) + f(t_{i+1})H_{i,3,1}(t) + f'(t_i)H_{i,3,2}(t) + f'(t_{i+1})H_{i,3,3}(t) \quad (5.24)$$

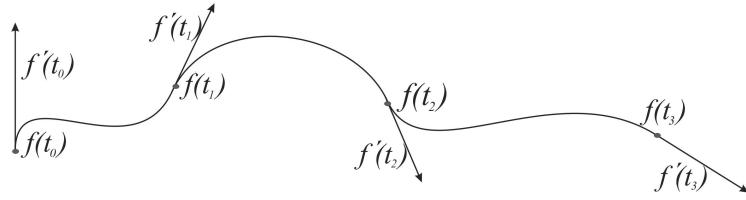


Figure 5.3: A cubic Hermite spline curve interpolating a curve $f(t)$ at 4 points (marked with dots) and 4 tangent vectors (marked as arrows).

where

$$\begin{aligned} H_{i,3,0}(t) &= 2w_i(t)^3 - 3w_i(t)^2 + 1, \\ H_{i,3,1}(t) &= -2w_i(t)^3 + 3w_i(t)^2, \\ H_{i,3,2}(t) &= \Delta t_i (w_i(t)^3 - 2w_i(t)^2 + w_i(t)), \\ H_{i,3,3}(t) &= \Delta t_i (w_i(t)^3 - w_i(t)^2), \end{aligned} \quad (5.25)$$

and where

$$w_i(t) = \frac{t - t_i}{t_{i+1} - t_i}. \quad (5.26)$$

It (5.24) is defined using the endpoints and their respective derivatives as coefficients. The basis functions (5.25) are defined using the “translation and scaling function” w , (5.26), which you also will find (with an extra index) in the definition of B-splines, see (5.43). Note that the sum of the two first basis functions in (5.25) is 1, i.e. they are invariant under affine transformations. The two last basis functions are, however, not generally invariant under affine transformations. Their coefficients are not affected by translation at all, but they are preserved under any rotations. An example of Hermite interpolation can be seen in Figure 5.2.

Remark 2. In a system of homogenous coordinates (as we find in graphical systems like OpenGL), however, both translation and rotation will work well for all four coefficients because translation is an issue for points only not for vectors. (The two first coefficients are points and the two last are vectors.)

5.2 Hermite spline

The classical Hermite interpolation method using 5.14 together with 5.19 is suitable to interpolate several points together with derivatives (of several orders) at these points. The result is of course a polynomial of very high degree. This has a lot of disadvantages so there is another way of doing this that is obviously more practical, namely using pieces of polynomial based curves glued together at their endpoints by common points and derivatives. This was earlier called “osculatory interpolation”⁵ but nowadays it is mostly called Hermite spline. A Hermite spline is a set piece of polynomial curves of degree up to d (odd number) that is $C^{\frac{d-1}{2}}$ -smooth on its domain. Cubic Hermite splines are most used,

⁵Repeated interpolation at a point was called “osculatory interpolation” since it produces higher than first order contact between the function and its interpolant (“osculari” means “to kiss” in Latin), p.7 in [36].

that is n polynomial pieces of curves, $\{c_i(t)\}_{i=0}^{n-1}$, where each curve piece is defined as in (5.24), (5.25) and (5.26). It follows that it is uniquely defined by $n+1$ knot values $\{t_i\}_{i=0}^n$, $n+1$ points $\{f(t_i)\}_{i=0}^n$ and $n+1$ vectors $\{f'(t_i)\}_{i=0}^n$, where $n > 0$. In Figure 5.3 this is illustrated by a cubic Hermite spline curve interpolating a curve $f(t)$ at 4 points, $\{f(t_i)\}_{i=0}^3$, and their appurtenant tangent vectors $\{f'(t_i)\}_{i=0}^3$.

Sometimes the tangents (derivatives) are not present, Therefor, several strategies for making smooth curves without given derivatives are developed:

◆ A cardinal spline⁶ is a cubic Hermite spline whose tangents are defined by the points and a tension parameter. This spline creates a curve from one point to another taking into account the points before and after. By taking into account the points before and after the current curve, the curves appear to join together making one seamless curve. I.e. given $n+1$ points p_0, \dots, p_n , to be interpolated with n cubic Hermite curve segments, for each curve we have a starting point p_i and an ending point p_{i+1} with starting tangent v_i and ending tangent v_{i+1} with the tangents defined by:

$$v_i = \frac{t_{i+1} - t_{i-1}}{2} (1 - c) (p_{i+1} - p_{i-1})$$

where the first and last tangent v_0 and v_n are given (or one can use the endpoints only without dividing by 2) and c is a constant that modifies the length of the tangent (the tension parameter). The tension parameter c should be between 0 and 1.

◆ A Catmull-Rom spline (see [20]) is a cardinal spline with the tension parameter $c = 0$.
 ◆ Another spline related to cubic Hermite spline is “cubic Bessel spline” (see [36]). Here one chooses the vectors (tangents) as the derivative at t_i of the polynomial of degree 2 which agrees with f at t_{i-1}, t_i, t_{i+1} . A short calculation gives the tangents

$$v_i = \frac{\Delta t_i f[t_{i-1}, t_i] + \Delta t_{i-1} f[t_i, t_{i+1}]}{\Delta t_{i-1} + \Delta t_i},$$

which shows that Bessel interpolation is also a local method.

◆ Yet another local method is Akima’s interpolation method from 1970 (see [2]). Akima chose to compute the tangents by,

$$v_i = \frac{w_{i+1} f[t_{i-1}, t_i] + w_{i-1} f[t_i, t_{i+1}]}{w_{i-1} + w_{i+1}}.$$

where

$$w_j = |f[t_j, t_{j+1}] - f[t_{j-1}, t_j]|.$$

5.3 Cubic spline interpolation

There is, however, an other “classical” interpolation method, using piecewise polynomials, which has a higher degree of continuity, namely cubic spline interpolation (see [36]). This method is not local, so changes do not only have a local affect. The computational

⁶Schoenberg (see [123]) defines Cardinal splines as the class of polynomial splines on \mathbb{R} with an infinite knot vector of simple integer knots and where all knot interval are 1.

cost is, however, only slightly higher than the local methods because the matrix to be solved is a very narrow band matrix with only 3 diagonal elements.

So, if we want a C^2 -smooth function we still can look at the cubic Hermite interpolation, but we now refrain from specifying derivatives at the internal knots. Instead we specify that:

$$c''_{i-1}(t_i) = c''_i(t_i) \quad \text{for } i = 1, 2, \dots, n-1, \quad (5.27)$$

and use these to solve a system of linear equations, i.e. involving an $(n-1 \times n-1)$ matrix, to find the derivatives in the internal knots,

$$c'_i(t_i), \quad i = 1, 2, \dots, n-1.$$

To find the equations for cubic spline interpolation we just take (5.21) and compute the second derivative. Then we use the condition (5.27) in the internal knots.

The system of equations follows,

$$\mathbf{Ax} = \mathbf{b}. \quad (5.28)$$

where

$$\mathbf{A} = \begin{pmatrix} 2 & 1-\lambda_1 & 0 & \cdots & 0 \\ \lambda_2 & 2 & 1-\lambda_2 & \ddots & \vdots \\ 0 & \lambda_3 & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & 1-\lambda_{n-2} \\ 0 & \cdots & 0 & \lambda_{n-1} & 2 \end{pmatrix},$$

$$\mathbf{x} = \begin{pmatrix} c'_1(t_1) \\ \vdots \\ \vdots \\ c'_{n-1}(t_{n-1}) \end{pmatrix} \quad \text{and} \quad \mathbf{b} = \begin{pmatrix} b_1 \\ \vdots \\ \vdots \\ b_{n-1} \end{pmatrix}.$$

Here is

$$\lambda_i = \frac{\Delta t_i}{\Delta t_{i-1} + \Delta t_i}, \quad \text{and} \quad b_i = 3\beta_i - \delta_i, \quad (5.29)$$

where

$$\beta_i = \lambda_i f[t_{i-1}, t_i] + (1 - \lambda_i) f[t_i, t_{i+1}], \quad (5.30)$$

and

$$\delta_i = \begin{cases} \lambda_1 c'_0(t_0), & i = 1, \\ 0, & 1 < i < n-1, \\ (1 - \lambda_{n-1}) c'_{n-1}(t_n), & i = n-1. \end{cases} \quad (5.31)$$

In (5.31) we can see that we need to know (or be able to estimate) the tangents at the start and end of the spline. This freedom leads to several types of boundary conditions.

In [36] 6 different boundary condition are listed. We shall now look at another way of organizing the equations that leads to one of the other boundary conditions. The equation is the same as in (5.28) (but the matrix dimension is now $n + 1 \times n + 1$). The start and end tangents (and second derivatives) are now included in a new first and last line of the matrix (computed from a second derivative of (5.21)). We now get

$$\mathbf{A} = \begin{pmatrix} 2 & 1 & 0 & \cdots & \cdots & 0 \\ \lambda_1 & 2 & 1 - \lambda_1 & \ddots & \ddots & \vdots \\ 0 & \lambda_2 & 2 & 1 - \lambda_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \lambda_{n-1} & 2 & 1 - \lambda_{n-1} \\ 0 & \cdots & \cdots & 0 & 1 & 2 \end{pmatrix},$$

$$\mathbf{x} = \begin{pmatrix} c'_0(t_0) \\ \vdots \\ c'_{n-1}(t_{n-1}) \\ c'_{n-1}(t_n) \end{pmatrix} \quad \text{and} \quad \mathbf{b} = \begin{pmatrix} b_0 \\ \vdots \\ b_n \end{pmatrix}.$$

Here λ is defined in (5.29) and β is defined in (5.30), but now

$$b_i = \begin{cases} 3f[t_0, t_1] - \frac{\Delta t_0}{2}c''_0(t_0), & i = 0, \\ 3\beta_i, & 0 < i < n, \\ 3f[t_{n-1}, t_n] - \frac{\Delta t_{n-1}}{2}c''_{n-1}(t_n), & i = n. \end{cases} \quad (5.32)$$

If, in (5.32), $c''_0(t_0) = c''_{n-1}(t_n) = 0$ (or a zero vector if vector valued) we have so called *free ends*, and the spline is a “natural spline” with a behavior quite like a real spline device.⁷ In Figure 5.4 is one example of cubic spline interpolation with free ends (the upper one). As we can see, the curvature zero is at the ends. In the other two examples end conditions are used with given tangents at both the start and end. The fact that the energy (or actually the approximation and simplification by using $c''(t)$ instead of the curvature) is minimized by a third degree spline function (in the Sobolev space $H^2(a, b)$), i.e. $\min \int_a^b |c''(t)|^2 dt$, is in a way illustrated in the figure. We can see from the plots that the free end condition has “the smallest curvature”.

According to Schumaker [124] it was noticed already in the mid-1700s by Euler and the Bernoulli brothers that the shape of a spline device is approximately given by pieces of functions that are 3th degree polynomials.

⁷To draw curves, especially for shipbuilding, draftsmen often used long, thin, flexible strips of wood, plastic, or metal called a spline. The strips were held in place with lead weights (called ducks because of their duck line shape). The elasticity of the material combined with the constraint of the control points, or knots, would cause the strip to take the shape which minimizes the energy required for bending it between the fixed points, and thus adopt “the smoothest possible” shape (according to elasticity theory).

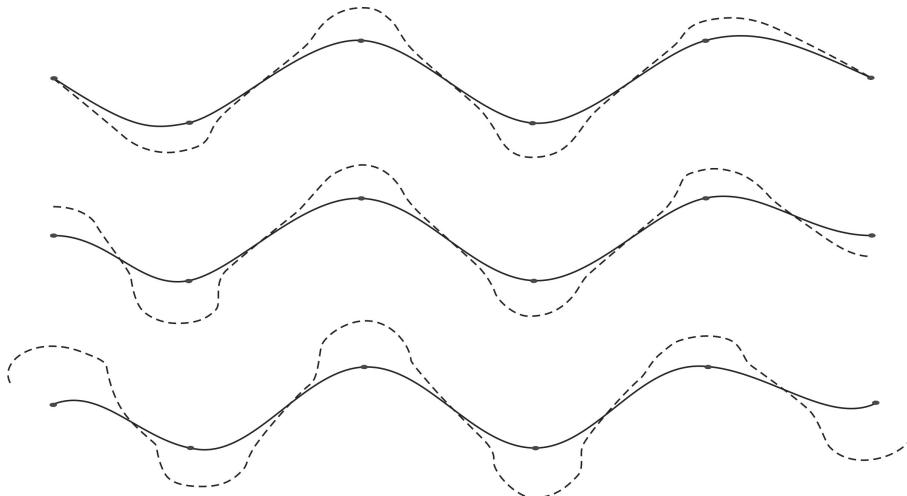


Figure 5.4: Three cubic spline curves (solid) are interpolating the same six points (marked with dots). The dotted lines mark the curvature of the curves. Free end condition is used in the upper example (second derivative is zero at start and end), this can clearly be seen in the figure. Horizontal tangent vectors used as end conditions are given in the middle example. Tangent vectors that both are 45° up to the right side used as end conditions are given in the lower example.

5.4 History of B-Splines

Having arrived at this point, we go back again more than a century to follow a parallel development from a quite different field. It is clear to us that piecewise functions (splines) is a more practical tool for interpolating data, especially if there are large amounts of data. It is, therefore, not difficult to see that the need for smoother interpolants in some applications led in the late 1800s to the development of interpolation formulae based on piecewise polynomials, so-called *osculatory interpolation techniques*, most of which appeared in the actuarial literature (see Greville [71]). A well-known example of this is the formula proposed in 1899 by Karup [85]. The formula results in a piecewise third-degree polynomial interpolant that is continuous and also continuously differentiable everywhere. By using this formula, it is possible to reproduce polynomials up to second degree. Another example is the formula proposed in 1906 by Henderson [77], which also yields a continuously differentiable, piecewise third-degree polynomial interpolant, but is capable of reproducing polynomials up to third degree. A third example is the formula published in 1927 by Jenkins [83]. The resulting composite curve is a piecewise third-degree polynomial that is twice continuously differentiable. The price to pay, however, is that this curve is not an interpolant. (All these formulae can be found in [71], and shortly in [108].)

The need for practically applicable methods for interpolation or smoothing of empirical data also formed the impetus for Schoenberg's first study of the subject. In his 1946 landmark papers [120, 121] he noted that for every *osculatory interpolation* formula applied to equidistant data, where he assumed the distance to be unity, there exists an even function $\Phi : \mathbb{R} \rightarrow \mathbb{R}$, in terms of which the formula may be written as

$$f(x) = \sum_{j=-\infty}^{\infty} a_j \Phi(x-j), \quad (5.33)$$

where Φ (which he termed as the *basic function* of the formula), completely determines the properties of the resulting interpolant and reveals itself when applying the initial formula to the impulse sequence defined by $a_0 = 1$ and $a_k = 0$, $\forall k \neq 0$. By analogy with Whittaker's cardinal series (see [149]), Schoenberg referred to the general expression (5.33) as a formula of the cardinal type, but noted that the basic function of the cardinal series, $\Phi(x) = \sin(\pi x)/(\pi x)$, is inadequate for numerical purposes due to its excessively low damping rate. The basic functions involved in Waring-Lagrange interpolation, on the other hand, possess the limiting property of being at most continuous, but not continuously differentiable. He then pointed at the smooth curves obtained by the use of a mechanical spline,⁸ argued that these are piecewise cubic arcs with a continuous first- and second-order derivative, and then continued to introduce the notion of the mathematical spline:

Definition 5.1. A real function f defined for all real x is called a *spline function (curve)* of order k and denoted by S_k if it have the following properties:

- i) it is composed of polynomial arcs of degree at most $d = k - 1$,
- ii) it is of class C^{k-2} , i.e., f has $k - 2$ continuous derivatives,
- iii) the only possible function points (knot values) of the various polynomial arcs are the values $x = k$ if k is even, or else $x = k + \frac{1}{2}$ if k is odd.

Note that these requirements are satisfied by the curves resulting from the aforementioned smoothing formula proposed by Jenkins, and also studied by Schoenberg [120], which constitutes one of the earliest examples of a spline generating formula. Note also that Hermite spline and related splines do not fulfill all the requirements.

After having defined the spline curve, Schoenberg continued to prove that any spline curve S_k may be represented uniquely in the form

$$S_k(x) = \sum_{j=-\infty}^{\infty} a_j M_k(x-j) \quad (5.34)$$

for appropriate values of the coefficients a_j . There are no convergence difficulties since, as we will see below, $M_k(x)$ vanishes for $|x| > k/2$. Thus (5.34) represents an S_k for arbitrary $\{a_j\}$ and represents the most general one.

Here, $M_k : \mathbb{R} \rightarrow \mathbb{R}$ denotes the so-called central *B-spline* of degree $d = k - 1$, which Schoenberg defines as the inverse Fourier integral

$$M_k(x) = \frac{1}{2\pi} \int_{-\infty}^{\infty} \left(\frac{\sin(\omega/2)}{\omega/2} \right)^k e^{i\omega x} d\omega, \quad (5.35)$$

⁸The use of the word “spline” for a flexible rod can be traced back to shipbuilding in east England in the 18th century. It was originally an East Anglian dialect word. Flexible rods (spline devices) have been used by shipbuilders all over the world for a very long time. In Norway there are actually 3 different names for them, depending on the location of the shipyard. In the north (where this author has his experience) it is called “rei”, pronounced and related to ray, as in “ray-tracing”.

and, when evaluating explicitly as⁹

$$M_k(x) = \frac{1}{(k-1)!} \delta^k x_+^{k-1},$$

where δ^p is the p th-order central difference operator¹⁰ and x_+^n denotes the one-sided power function defined as

$$x_+^n = \begin{cases} x^n, & \text{if } x \geq 0 \\ 0, & \text{if } x < 0. \end{cases}$$

Soon after this first paper [120, 121] was written, the possibilities of an extension to arbitrary spaced knots was implied by Curry in his review [30] of Schoenberg's paper. And in an abstract, [31], B-splines¹¹ for arbitrary spaced knots were introduced. The whole article was written in 1945-47 but was actually published 20 years later [32]. Curry and Schoenberg's definition of B-splines is as follows,

Definition 5.2. A B-spline denoted $M_k(x)$ is defined by $k+1$ increasing real values x_0, \dots, x_k , where $x_0 < x_k$, its explicit expression is

$$M_k(x; x_0, \dots, x_k) = k \sum_{v=0}^k \frac{(x_v - x)_+^{k-1}}{\omega'(x_v)} \quad (5.36)$$

where

$$\omega(x) = (x - x_0) \cdots (x - x_n). \quad (5.37)$$

Curry and Schoenberg stated the following properties for the B-splines:

- 1) They are shown to be bell-shaped.
- 2) They are also shown to be the projections onto the x-axis of the volumes of appropriate n-dimensional simplices.
- 3) This geometric interpretation allows us to conclude, by means of Brunn's theorem, that the B-spline function is logically concave.
- 4) They are also shown to form a basis for all spline functions of degree $n-1$ and given knots.
- 5) They can be defined on multiple knots.

⁹Schoenberg noted that (5.35) had been evaluated explicitly for low values of k by various authors, i.e. S. Bochner in lectures of Fourier analysis in 1936. In an article by Butzer, Schmidt and Stark [18] several predecessor in B-splines are mentioned that were probably not known by Schoenberg. Among these are Maurer in 1896 [103] (discussing a function related to the central B-spline function of Schoenberg), Sommerfeld in 1904 and 1929 [133] (making a geometric interpretation, a Box-spline like method for constructing B-splines) and Brun in 1932 [16] (developing a recursive corner cutting method like de Casteljau).

¹⁰Often called Sheppard's central-difference operator δ (see [130]), defined by

$$\delta \phi(\xi) = \phi(\xi + \frac{1}{2}) - \phi(\xi - \frac{1}{2})$$

and

$$\delta^p \phi(\xi) = \delta^{p-1} \phi(\xi + \frac{1}{2}) - \delta^{p-1} \phi(\xi - \frac{1}{2}), \quad p > 1,$$

for any function $\phi : \mathbb{R} \rightarrow \mathbb{R}$ at any ξ . Not to be mixed up with the right-sided difference operator (5.6).

¹¹Curry and Schoenberg actually first called them the fundamental spline functions. The name B-spline refers to basis splines because they form a basis for spline functions, which was proved by Curry and Schoenberg in [31]. In [122], published in 1967, Schoenberg used the name B-spline.

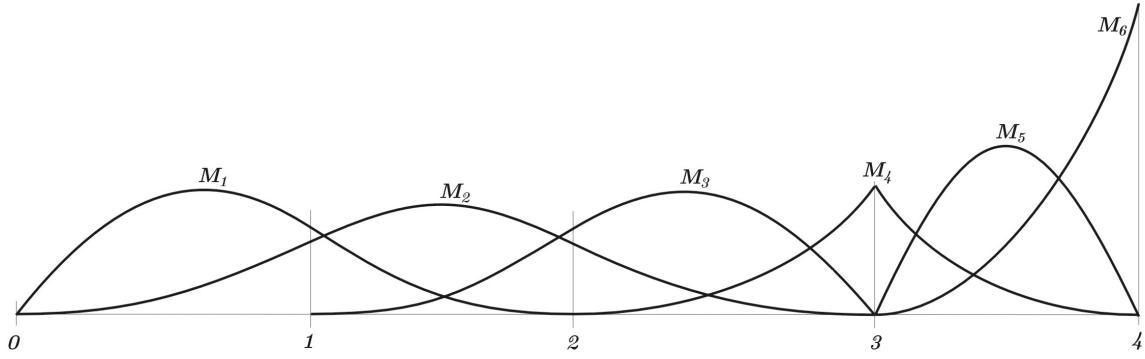


Figure 5.5: A copy of a sketch from Curry and Schoenberg [32] of six quadratic B-splines defined by 9 knots; \$x_1 = x_2 = 0\$, \$x_3 = 1\$, \$x_4 = 2\$, \$x_5 = x_6 = 3\$ and \$x_7 = x_8 = x_9 = 4\$.

- 6) The B-splines \$M_k(x)\$ (defined above) are frequency functions, which means that they are non-negative and their integral over \$\mathbb{R}\$ is 1.¹²

In Figure 5.5 there is a copy of a sketch from Curry and Schoenberg [32] of the six quadratic B-splines (\$k = 3\$) for the case when there are 9 knots, and where some of them are equal. The six B-spline functions are:

$$\begin{aligned} M_1(x) &= M_3(x; 0, 0, 1, 2), & M_2(x) &= M_3(x; 0, 1, 2, 3), \\ M_3(x) &= M_3(x; 1, 2, 3, 3), & M_4(x) &= M_3(x; 2, 3, 3, 4), \\ M_5(x) &= M_3(x; 3, 3, 4, 4), & M_6(x) &= M_3(x; 2, 4, 4, 4). \end{aligned}$$

In the classical definition of divided differences there is also a definition of divided differences for discrete data \$\{(x_i, y_i)\}_{i=0}^n\$. It is naturally not connected to any function, but is defined in the same way as divided differences in (5.12) and (5.13). There is however another expression of divided differences which e.g. can be found in expression (2.6.4) in [33],

$$[y_0, y_1, \dots, y_n] = \sum_{i=0}^n \frac{y_i}{\omega'(x_i)}, \quad (5.38)$$

where \$\omega(x)\$ is the same as in (5.37).

Expression (5.38) is in a way equal to (5.36), but because there is the function \$(x_v - x)_+^{k-1}\$ instead of scalars involved in (5.36) we have to clarify the notation. A divided difference version of a B-spline \$M_k(x)\$ is, therefore,

$$M_k(x; x_0, \dots, x_k) = k (-x)_+^{k-1} [x_0, \dots, x_k]. \quad (5.39)$$

The notation is called a “placeholder” notation and a more detailed description of this notation can be found on page 108 in [36].

Divided differences are defined recursive, so it is natural to try to develop a simple recursive algorithm for computing B-splines. This was done in 1972 by at least three different authors simultaneously. Cox [27] established it for simple knots. The result for general knots is credited to deBoor [35]. In his paper he mentioned that Louis Mansfield had also

¹²NB! Modern B-splines are scaled in a different way because in CAGD partition of unity is a more important property.

discovered the recursion. In the same paper deBoor also introduces the derivative formula for B-splines. Some other important algorithms in spline history are; knot insertion introduced simultaneously in 1980 by Boehm [13] for single knots and Cohen, Lyche and Riesenfeld [22] for general knot insertion, and degree raising in 1985 by Cohen, Lyche and Schomaker [23]. It should of course be mentioned that Paul de Casteljau in 1959 developed an algorithm for the computation of Bézier curves (but it was only published in an internal Citroë report [37]) and that Pierre Étienne Bézier in 1966 published the construction of Bézier curves/surfaces [10, 11]. (Bézier curves are the simplest B-spline curves)

As a final historical remark in this section it can be mentioned that according to Farin [53] N. Lobachevsky was as early as the nineteenth century investigating B-splines as convolutions of certain probability distributions (over a very special knot sequence). Also Peano Kernel for 3rd difference gives a second degree B-spline, for which there is a plot on page 73 in the Davis book from 1963, [33]. The name spline is referring to a mechanical spline device that is minimizing the energy while bending. This can be expressed as an equation minimizing the square of the curvature, i.e.

$$\min \int_{s=x_0}^{x_n} \kappa(s)^2 ds. \quad (5.40)$$

A cubic polynomials is minimizing the square of the second derivatives. This is of course not the same as (5.40), but it is usually an acceptable approximation. There are works on better approximations, e.g Mehlum in [105, 106]. How to use the term spline is therefore not evident. It should possibly in some sense approximate a spline device, whether it is an optimal or not an optimal solution according to some functional. today it is most common to think about splines as pieces that are glued together to one curve or surface with controlled smoothness.

This leads to the modern normalized B-splines, it's notation and algorithms which are the contents of the next section.

5.5 Modern B-splines

Modern B-spline theory has several branches. One of them is based on the blossoming method proposed by Ramshaw [117] and, in a different form by de Casteljau [39]. In this section we will, however, first look at B-splines in (what we now can call) the classical way, then we will give some examples and finally take a rather unusual but clarifying view on B-spline algorithms.

We denote B-splines in two ways, a short and a more complete notation. These notations are

$$B_{k,i}(t) = B(t; t_i, \dots, t_{i+k}).$$

When we talk about B-splines we mean normalized B-splines, which means that all B-splines defined on an infinite knot sequence sum up to 1 (also called form the partition of

unity), i.e.

$$\sum_{i=j-k}^j B(t; t_i, \dots, t_{i+k}) = 1, \quad t_j \leq t < t_{j+1}.$$

Compared to the original Curry-Schoenberg B-splines $M(t; t_i, \dots, t_{i+k})$ we thus have the following relationship

$$B(t; t_i, \dots, t_{i+k}) = \frac{t_{i+k} - t_i}{k} M(t; t_i, \dots, t_{i+k}).$$

Note that for integer non multiple knots i.e 1,2,... then $B(t; t_i, \dots, t_{i+k}) = M(t; t_i, \dots, t_{i+k})$.

This leads to the following definition of the B-spline that usually is called the Cox-de'Boor recursion formula (sometimes also called Mansfield-Cox-de'Boor).

Definition 5.3. Given k increasing real numbers $\{t_i, t_{i+1}, \dots, t_{i+k}\}$, where $t_{i+k} > t_i$ (they do not otherwise need to be strongly increasing). The B-spline of degree d and order $k = d + 1$ is defined by the recursion formula

$$B(t; t_i, \dots, t_{i+k}) = w_{d,i}(t) B(t; t_i, \dots, t_{i+k-1}) + (1 - w_{d,i+1}(t)) B(t; t_{i+1}, \dots, t_{i+k}) \quad (5.41)$$

where the termination of the recursion is

$$B(t; t_j, t_{j+1}) = \begin{cases} 1, & t_j \leq t < t_{j+1}, \\ 0, & \text{otherwise,} \end{cases} \quad (5.42)$$

and where

$$w_{d,i}(t) = \frac{t - t_i}{t_{i+d} - t_i}. \quad (5.43)$$

A list of the basic properties of $B(t; t_i, \dots, t_{i+k})$ follows.

- P1.** Every basis functions $B(t; t_i, \dots, t_{i+k})$ is positive on (t_i, t_{i+k}) and zero otherwise.
- P2.** The set of basis functions $B(t; t_i, \dots, t_{i+k})$ for $i = j-d, \dots, j$ form a partition of unity on $[t_j, t_{j+1}]$, i.e.

$$\sum_{i=j-d}^j B(t; t_i, \dots, t_{i+k}) = 1, \quad t_j \leq t < t_{j+1}.$$

- P3.** A B-spline $B(t; t_i, \dots, t_{i+k})$ is $C^r(\mathbb{R})$ where $r = d - s$ and s is ‘the number of maximum multiplicity of the knots t_i, \dots, t_{i+k} ’, i.e. maximum number of equal knots.
- P4.** The B-spline $B(t; t_i, \dots, t_{i+k})$ is, in case of simple knots “bell-shaped”, i.e. the v^{th} derivative $B^{(v)}(t; t_i, \dots, t_{i+k})$ (up to $v = d - 1$) has exactly v zeros in the interval (t_i, t_{i+k}) . If the knots are simple these zeros are distinct.

5.5.1 B-spline curves

Recall the formula for a Bézier curves from section 4.4, formula (4.4). B-spline curves share many important properties with Bézier curves, because the former is a generalization of the later. In the following we denote a B-spline curve $c(t)$, of degree d , and thus order $k = d + 1$, that is defined by n control points, and a knot vector $\bar{t} = \{t_1, t_2, \dots, t_{n+k}\}$. The formula is

$$c(t) = \sum_{i=1}^n c_i b_{k,i}(t), \quad t \in [t_d, t_n].$$

where $\{c_i\}_{i=1}^n$ is the vector of coefficients, control points. These control points defines the control polygon of the curve. $b_{k,i}(t)$ is the set of B-spline basis functions defined by the knot vector \bar{t}

If the knot vector of a B-spline curve is $\bar{t} = \{t_1, t_2, \dots, t_{n+k}\}$ with the first k and last k knots "clamped" (i.e., $t_0 = t_1 = \dots = t_d$ and $t_n = t_{n+1} = \dots = t_{n+d}$), we call it a clamped B-spline curve.

A B-spline curve has the following properties:

1. A B-spline curve $c(t)$ is a piecewise curve with each component is a curve of degree at most d . The curve $c(t)$ can be viewed as the union of curve segments defined on each knot span.
2. A Clamped B-spline curve start at the first control point and end at the last control point. Further, the curve leave the first point tangential to the line between the first and the second control point and enter the last control point tangential to the line between the second last and the last control point.
3. B-spline curves has a strong "Convex Hull Property". That is, a B-spline curve is contained in the convex hull of its control points. More specifically, if t is in the knot span $[t_i, t_{i+1}]$, then $c(t)$ is in the convex hull of the control points $c_{i-d}, c_{i-d+1}, \dots, c_i$.
4. A B-spline curve has a local Modification Scheme. That is, changing the position of control point c_i only affects the curve $c(t)$ on interval $[t_i, t_{i+k}]$.
5. A B-spline curve $c(t)$ is C^{d-j} continuous at a knot of multiplicity j . That is, if $t_i = t_{i+1} = \dots = t_{i+j-1}$, then the curve is C^{d-j} continuous at $c(t_i)$.
6. A B-spline curve has "Variation Diminishing Property". That is, in the plane, \mathbb{R}^2 , no straight line intersects a B-spline curve more times than it intersects the curve's control polygons (poly-line).
7. Bézier curves are special cases of B-spline curves. They are Clamped B-spline curves on the domain $[0, 1]$, and without internal knots (only knots at start and end).
8. The affine invariance property also holds for B-spline curves. If an affine transformation is applied to a B-spline curve, the result can be constructed from the affine images of its control points. This is a nice property. When we want to apply a geometric or even affine transformation to a B-spline curve, this property states that we can apply the transformation to control points, which is quite easy, and once the

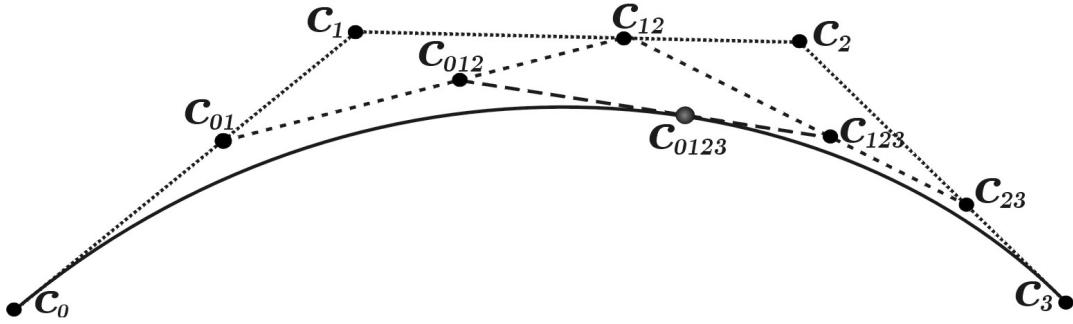


Figure 5.6: A Bézier curve (bold) and the points (bullets) and lines (dotted) illustrating the de Casteljau's algorithm for a third degree Bézier curve at $t = 0.6$.

transformed control points are obtained the transformed B-spline curve is the one defined by these new points. Therefore, we do not have to transform the curve.

5.5.2 Bézier curves and de Casteljau algorithm

To investigate the B-spline (basis) function we start with linear interpolation between two points (in the plane or in space) c_0 and c_1 .

$$c(t) = (1-t) c_0 + t c_1.$$

For $t \in [0, 1]$ we get the line segment between c_0 and c_1 .

Given a sequence of points c_0, c_1, c_2 and c_3 . If we for a given $t \in [0, 1]$ linear interpolate between pairs of two and two points, c_0 and c_1 , c_1 and c_2 , c_2 and c_3 , we will get three new points, which we call respectively c_{01} , c_{12} and c_{23} . In matrix, vector notation we have done the following,

$$\begin{pmatrix} c_{01} \\ c_{12} \\ c_{23} \end{pmatrix} = \begin{pmatrix} 1-t & t & 0 & 0 \\ 0 & 1-t & t & 0 \\ 0 & 0 & 1-t & t \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix}.$$

If we repeat the process with the same t we get

$$\begin{pmatrix} c_{012} \\ c_{123} \end{pmatrix} = \begin{pmatrix} 1-t & t & 0 \\ 0 & 1-t & t \end{pmatrix} \begin{pmatrix} c_{01} \\ c_{12} \\ c_{23} \end{pmatrix},$$

and finally

$$c_{0123} = (1-t \ t) \begin{pmatrix} c_{012} \\ c_{123} \end{pmatrix}.$$

This process is called de Casteljau's (corner cuttings) algorithm for evaluating Bézier curves (see [38] and [39]), and it is illustrated in Figure 5.6.

If we connect these three multiplications, we get the following equation for a third degree Bézier curve,

$$c(t) = \begin{pmatrix} 1-t & t \\ 0 & 1-t \end{pmatrix} \begin{pmatrix} 1-t & t & 0 & 0 \\ 0 & 1-t & t & 0 \\ 0 & 0 & 1-t & t \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix}. \quad (5.44)$$

These formulas, computed from the right side, are the de Casteljau (corner cuttings) algorithm. If we only compute the three matrices from the left we get a vector with the four Bernstein polynomials¹³ of degree three,

$$((1-t)^3, \quad 3(1-t)^2t, \quad 3(1-t)t^2, \quad t^3).$$

If we denote these Bernstein factor matrices for $T(t)$, and add an index d on it, for example, if $d = 2$, it is

$$T_2(t) = \begin{pmatrix} 1-t & t & 0 \\ 0 & 1-t & t \end{pmatrix}.$$

We now have the following general definition.

Definition 5.4. $T_d(t)$ is a matrix with dimension $d \times (d+1)$ where all lines numbered j

- i) only have element number j and $j+1$ different from zero.
- ii) where (in the Bézier case) element number j is $1-t$ and element $j+1$ is t .

Note that in the Bézier case we use t , while in general B-splines we will use the translation and scaling function $w_{d,i} : [t_i, t_{i+d}] \rightarrow [0, 1]$ (equation (5.45) below) instead of t , as described in the following section.

We also define the matrix notation of the set of Bernstein polynomials of degree n to be

$$T^n(t) = T_1(t)T_2(t) \cdots T_n(t).$$

It follows that equation (5.44) can be expressed as

$$c(t) = T_1(t)T_2(t)T_3(t) \mathbf{c} = T^3(t) \mathbf{c},$$

where $\mathbf{c} = (c_0, c_1, c_2, c_3)^T$ is a vector of points. In traditional notation this will be

$$c(t) = \sum_{i=0}^3 b_{i,d}(t) c_i.$$

Notice that the derivative of the matrix $T_d(t)$ is a matrix of constants. We, therefor, denote it T'_d . For $d = 1$ we get

$$T'_1 = (-1, \quad 1).$$

¹³The polynomial occurred as result of the work of Sergei Natanovich Bernstein, an Ukrainian mathematician (1880-1968). Bernstein introduced the curves in 1911 (published in [8]), using them for constructive proof of the Stone-Weierstrass approximation theorem. The general expression for the Bernstein polynomials of degree d are

$$b_{d,i}(t) = \binom{d}{i} t^i (1-t)^{d-i}, \quad i = 0, 1, 2, \dots, d.$$

For a study of these polynomials see page 183–186 in [81] or page 108–126 in [33].

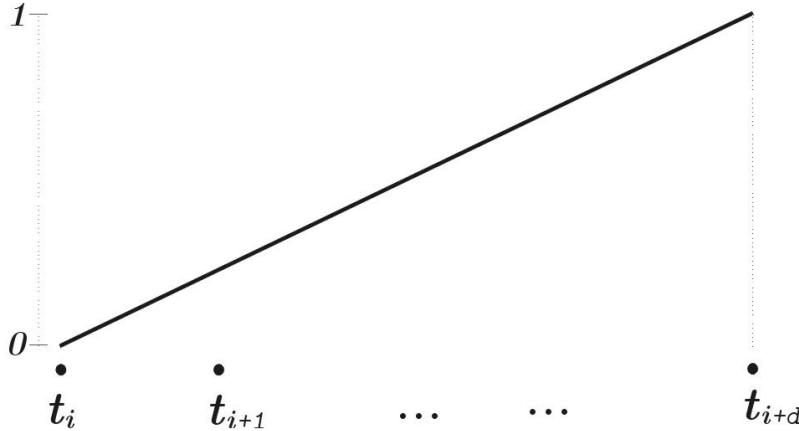


Figure 5.7: The linear translation and scaling function $w_{d,i}(t)$, a straight line that is 0 at the knot t_i and 1 at the knot t_{i+d} .

5.5.3 The B-spline factor matrix $T(t)$

In the previous section the $d \times (d + 1)$ matrix $T_d(t)$ was defined, and used in the Bézier curve definition. A more complex version, i.e. the B-spline type of this matrix is the topic of this subsection. We first recall the linear translation and scaling function from definition 5.3

$$w_{d,i}(t) = \begin{cases} \frac{t-t_i}{t_{i+d}-t_i}, & \text{if } t_i \leq t < t_{i+d} \\ 0, & \text{otherwise.} \end{cases} \quad (5.45)$$

As we can see in Figure 5.7, the function (5.45), describes a straight line going from 0 at the knot t_i to 1 at the knot t_{i+d} . The knots in the figure are marked with bullets. Note that in the case of Bernstein/Bézier type we just have $d + 1$ equal knots at 0 and $d + 1$ equal knots at 1. It then follows that $w_{d,i}(t) = t$ for all relevant d and i .

Definition 5.5. The B-spline factor matrix $T_d(t)$ is a $d \times (d + 1)$ band limited matrix with two nonzero elements on each row. The matrix is as follows

$$T_d(t) = \begin{pmatrix} 1 - w_{d,i-d+1}(t) & w_{d,i-d+1}(t) & 0 & \cdots & \cdots & 0 \\ 0 & 1 - w_{d,i-d+2}(t) & w_{d,i-d+2}(t) & 0 & \cdots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & 0 & 1 - w_{d,i-1}(t) & w_{d,i-1}(t) & 0 \\ 0 & \cdots & \cdots & 0 & 1 - w_{d,i}(t) & w_{d,i}(t) \end{pmatrix}$$

Note here that in the matrix $T_d(t)$ the first index is d for all w . The last index in the w is denoted i . This index is determined by the knot vector, and it is the index of the knot fixed

by the requirement

$$t_i \leq t < t_{i+1}. \quad (5.46)$$

We observe in the matrix $T_d(t)$ that the last index of w is decreased by 1 from one line to the next line up. For the bottom line it is denoted i and for the line above the bottom line $i - 1$ and so on.

For $d = 3$ we have the following formula for a B-spline curve, note the indices of the coefficients on the right hand side,

$$c(t) = \begin{pmatrix} 1 - w_{1,i}(t) & w_{1,i}(t) \end{pmatrix} \begin{pmatrix} 1 - w_{2,i-1}(t) & w_{2,i-1}(t) & 0 \\ 0 & 1 - w_{2,i}(t) & w_{2,i}(t) \end{pmatrix} \begin{pmatrix} 1 - w_{3,i-2}(t) & w_{3,i-2}(t) & 0 & 0 \\ 0 & 1 - w_{3,i-1}(t) & w_{3,i-1}(t) & 0 \\ 0 & 0 & 1 - w_{3,i}(t) & w_{3,i}(t) \end{pmatrix} \begin{pmatrix} c_{i-3} \\ c_{i-2} \\ c_{i-1} \\ c_i \end{pmatrix},$$

where the index i is determined by the requirement (5.46) in Definition 5.5.

A general expression for a third degree B-spline curve is thus

$$c(t) = T_1(t)T_2(t)T_3(t) \mathbf{c} = T^3(t) \mathbf{c}.$$

If we compute this set of matrices from the left hand side we get the B-splines. The first matrix contains the two first degree B-splines on the knot interval $[t_i, t_{i+1}]$,

$$T_1(t) = \begin{pmatrix} 1 - w_{1,i}(t) & w_{1,i}(t) \end{pmatrix},$$

If we compute the two first matrices from left hand side, $T^2(t)$, we get a vector with the three second degree B-splines on the knot interval $[t_i, t_{i+1}]$,

$$T^2(t) = \begin{pmatrix} (1 - w_{1,i}(t))(1 - w_{2,i-1}(t)) \\ (1 - w_{1,i}(t))w_{2,i-1}(t) + w_{1,i}(t)(1 - w_{2,i}(t)) \\ w_{1,i}(t)w_{2,i}(t) \end{pmatrix}^T.$$

Theorem 5.1. *The matrices in Definition 5.5 are doing the same as the recursion algorithm in definition 5.3 if we compute them from left hand side starting with degree 1,2,....*

Proof. Recall from the definitions 5.4 and 5.5 that

$$T^{d-1}(t) = (B(t; t_{i-d}, \dots, t_{i+1}), B(t; t_{i-d+1}, \dots, t_{i+2}), \dots, B(t; t_i, \dots, t_{i+d}))$$

that is a vector of B-splines of degree $d - 1$, and where i is determined by (5.46). Computing this vector with each of the columns of $T_d(t)$ gives each of the B-splines of degree d and is exactly the same as expression (5.41) in Definition 5.3. \square

To investigate the derivative of $T(t)$ we first look at the derivative of the linear translation and scaling function (5.45) which we denote

$$\delta_{d,i} = \begin{cases} \frac{1}{t_{i+d} - t_i}, & \text{if } t_i \leq t < t_{i+d} \\ 0, & \text{otherwise.} \end{cases}, \quad (5.47)$$

and it is a constant, independent of the translation, only reflecting the scaling. In the Bézier case is $\delta_{d,i} = 1$ for all relevant i and d .

The derivative of the matrix $T(t)$ is then defined by the following.

Definition 5.6. *The B-spline derivative matrix T' is a $d \times (d+1)$ band limited matrix with two nonzero constant elements on each row (independent of t). The matrix is as follows*

$$T'_d = \begin{pmatrix} -\delta_{d,i-d+1} & \delta_{d,i-d+1} & 0 & \dots & \dots & 0 \\ 0 & -\delta_{d,i-d+2} & \delta_{d,i-d+2} & 0 & \dots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & 0 & -\delta_{d,i-1} & \delta_{d,i-1} & 0 \\ 0 & \dots & \dots & 0 & -\delta_{d,i} & \delta_{d,i} \end{pmatrix}$$

The index i in the definition follows the rule (5.46) of the index in $T_d(t)$ from definition 5.5.

5.5.4 Commutativity relations between $T(t)$ matrices and their derivatives

Before we can look at derivatives of B-spline curves we have to look at a special property of the matrix $T(t)$ and its derivative T' .

Lemma 5.1. *Derivation of the multiplication of two $T(t)$ matrices is commutative. That is, for all $d > 0$ and for a given knot sequence $t_{j-d}, t_{j-d+1}, \dots, t_{j+d+1}$, and $t_i \leq t < t_{i+1}$ is*

$$T_d(t) T'_{d+1} = T'_{d+1} T_{d+1}(t). \quad (5.48)$$

Proof. We start by computing the left hand side of 5.48. Taking the two elements that are different from zero in one line of $T_d(t)$ and computing them with the sub-matrix of T'_{d+1} gives something different from zero,

$$\begin{pmatrix} 1 - w_{d,j}(t) & w_{d,j}(t) \end{pmatrix} \begin{pmatrix} -\delta_{d+1,j-1} & \delta_{d+1,j-1} & 0 \\ 0 & -\delta_{d+1,j} & \delta_{d+1,j} \end{pmatrix}$$

which gives

$$\begin{pmatrix} -(1 - w_{d,j}(t))\delta_{d+1,j-1} & (1 - w_{d,j}(t))\delta_{d+1,j-1} - w_{d,j}(t)\delta_{d+1,j} & w_{d,j}(t)\delta_{d+1,j} \end{pmatrix}. \quad (5.49)$$

We now use the same procedure on the right hand side of 5.48,

$$\begin{pmatrix} -\delta_{d,j} & \delta_{d,j} \end{pmatrix} \begin{pmatrix} 1 - w_{d+1,j-1}(t) & w_{d+1,j-1}(t) & 0 \\ 0 & 1 - w_{d+1,j}(t) & w_{d+1,j}(t) \end{pmatrix}$$

which gives

$$\begin{pmatrix} -(1 - w_{d+1,j-1}(t))\delta_{d,j} & (1 - w_{d+1,j}(t))\delta_{d,j} - w_{d+1,j-1}(t)\delta_{d,j} & w_{d+1,j}(t)\delta_{d,j} \end{pmatrix}. \quad (5.50)$$

We compute the first element of (5.49),

$$(1 - w_{d,j}(t))\delta_{d+1,j-1} = \frac{t_{j+d} - t}{(t_{j+d} - t_j)(t_{j+d+1} - t_{j-1})}, \quad (5.51)$$

and the first element of (5.50),

$$(1 - w_{d+1,j-1}(t))\delta_{d,j} = \frac{t_{j+d} - t}{(t_{j+d+1} - t_{j-1})(t_{j+d} - t_j)}, \quad (5.52)$$

which shows that the first element of (5.49) is equal to the first element of (5.50).

We now compute the third element of (5.49),

$$w_{d,j}(t)\delta_{d+1,j} = \frac{t - t_j}{(t_{j+d} - t_j)(t_{j+d+1} - t_j)}, \quad (5.53)$$

and the third element of (5.50),

$$w_{d+1,j}(t)\delta_{d,j} = \frac{t - t_j}{(t_{j+d+1} - t_j)(t_{j+d} - t_j)}, \quad (5.54)$$

which shows that the third element of (5.49) is equal to the third element of (5.50).

Finally we compute the second element of (5.49), which is minus the first element minus the third element of (5.49), i.e.

$$(1 - w_{d,j}(t))\delta_{d+1,j-1} - w_{d,j}(t)\delta_{d+1,j} = (5.51) - (5.53),$$

and if we reorganize the second element of (5.50), we get minus the first element minus the third element of (5.50), i.e.

$$\begin{aligned} (1 - w_{d+1,j}(t))\delta_{d,j} - w_{d+1,j-1}(t)\delta_{d,j} &= \\ (1 - w_{d+1,j-1}(t))\delta_{d,j} - w_{d+1,j}(t)\delta_{d,j} &= (5.52) - (5.54), \end{aligned}$$

which also shows that the second element of (5.49) is equal to the second element of (5.50).

This shows that they are equal for the whole line and, thus, every line and, thus, the whole expression, which completes the proof. \square

The main commutativity theorem now follows.

Theorem 5.2. *Derivation of multiplication of a set of $T(t)$ matrices is commutative. That is, for all $d > 0$ and for a given knot vector it is independent what matrix is the derivative matrix T' . i.e.*

$$T_d(t) T_{d+1}(t) \cdots T_{d+j-1}(t) T'_{d+j} = T'_d T_{d+1}(t) \cdots T_{d+j-1}(t) T_{d+j}(t). \quad (5.55)$$

Proof. This follows by an induction of the result of Lemma 5.1 \square

5.5.5 B-splines on Matrix notations

First a concrete example, given a third degree B-spline curve

$$c(t) = T_1(t) \ T_2(t) \ T_3(t) \ \mathbf{c} = T^3(t) \ \mathbf{c}.$$

It follows from Theorem 5.2 that the derivative is

$$c'(t) = (T'_1 \ T_2(t) \ T_3(t) + T_1(t) \ T'_2 \ T_3(t) + T_1(t) \ T_2(t) \ T'_3) \ \mathbf{c} = 3 \ T^2(t) \ T'_3 \ \mathbf{c}.$$

We are now ready to give a general expression of a B-spline function/curve and its derivatives on matrix notation.

Definition 5.7. A B-spline function/curve of degree d is on matrix notation as follows

$$c(t) = T^d(t) \ \mathbf{c} \quad (5.56)$$

where the set of B-splines of degree d are the

$$T^d(t) = T_1(t) \ T_2(t) \ \cdots \ T_d(t),$$

where $T_1(t) \ T_2(t) \ \cdots \ T_d(t)$ are according to Definition 5.5, and where \mathbf{c} is the coefficient vector with $d+1$ elements

$$\mathbf{c} = (c_{i-d}, c_{i-d+1}, \dots, c_i)^T,$$

and where the index i is given by the requirement

$$t_i \leq t < t_{i+1}.$$

The j derivative, $0 < j \leq d$, of a B-spline function/curve of degree d is of degree $d-j$ and is expressed as follows

$$c^{(j)}(t) = \frac{d!}{(d-j)!} T^{d-j}(t) \ T'^j \ \mathbf{c} \quad (5.57)$$

where

$$T'^j = T'_{d-j+1} \ T'_{d-j+2} \ \cdots \ T'_d.$$

5.5.6 Examples of B-splines, matrix notation, de Casteljau's algorithm and derivatives

The first example is a 2nd degree B-spline curve, that is in matrix notation,

$$c(t) = T^2(t) \ \mathbf{c}, \quad (5.58)$$

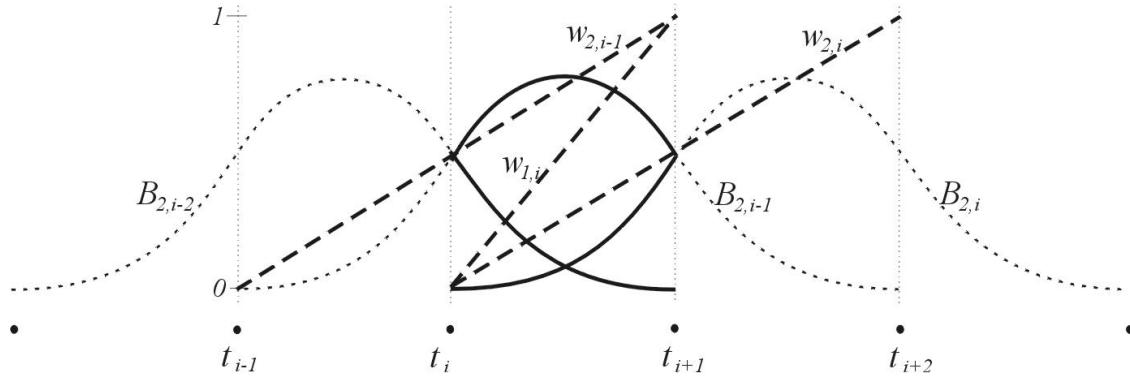


Figure 5.8: The three 2nd degree B-splines in the knot interval $[t_i, t_{i+1}]$ (solid) and the three linear translation and scaling functions (dashed) involved in the computations of these B-splines. The B-splines are dotted outside the actual interval.

where

$$\begin{aligned} T^2(t) &= T_1(t) \ T_2(t) \\ &= \begin{pmatrix} 1 - w_{1,i}(t) & w_{1,i}(t) \end{pmatrix} \begin{pmatrix} 1 - w_{2,i-1}(t) & w_{2,i-1}(t) & 0 \\ 0 & 1 - w_{2,i}(t) & w_{2,i}(t) \end{pmatrix} \quad (5.59) \\ &= \begin{pmatrix} B_{2,i-2}(t), & B_{2,i-1}(t), & B_{2,i}(t) \end{pmatrix}, \end{aligned}$$

and

$$\mathbf{c} = (c_{i-2}, \ c_{i-1}, \ c_i)^T,$$

Figure 5.8 shows the three 2nd degree B-splines, $B_{2,i-2}(t)$, $B_{2,i-1}(t)$ and $B_{2,i}(t)$ in (5.59) sketch, together with the three linear translations and the scaling function they are constructed from (remember that this is only in the interval $[t_i, t_{i+1}]$). These functions are $w_{1,i}$, in the left hand matrix $T_1(t)$, and $w_{2,i-1}$ and $w_{2,i}$ in the second matrix $T_2(t)$. In Figure 5.8 you can see them as dashed line segments.

Figure 5.9 shows de Casteljau's (corner cutting) algorithm applied on curve (5.58) (2nd degree B-spline curve) with the knots $t_{i-1} = 0$, $t_i = 1$, $t_{i+1} = 2$ and $t_{i+2} = 3$. $\hat{t} = 3/2$ is used in the example. This gives $w_{1,1}(\hat{t}) = 1/2$ from the first matrix $T_1(t)$, and we get $w_{2,i-1}(\hat{t}) = 3/4$ and $w_{2,i}(\hat{t}) = 1/4$ from the second matrix $T_2(t)$. In Figure 5.9 the first two new points p_1 and p_2 are computed by

$$\begin{pmatrix} p_1 \\ p_2 \end{pmatrix} = \begin{pmatrix} 1 - \frac{3}{4} & \frac{3}{4} & 0 \\ 0 & 1 - \frac{1}{4} & \frac{1}{4} \end{pmatrix} \begin{pmatrix} c_{i-2} \\ c_{i-1} \\ c_i \end{pmatrix} = \begin{pmatrix} \frac{1}{4}c_{i-2} + \frac{3}{4}c_{i-1} \\ \frac{3}{4}c_{i-1} + \frac{1}{4}c_i \end{pmatrix}.$$

The last point in Figure 5.9, p_3 (on the curve), is then computed

$$p_3 = \begin{pmatrix} 1 - \frac{1}{2} & \frac{1}{2} \end{pmatrix} \begin{pmatrix} p_1 \\ p_2 \end{pmatrix} = \left(\frac{1}{2} p_1 + \frac{1}{2} p_2 \right).$$

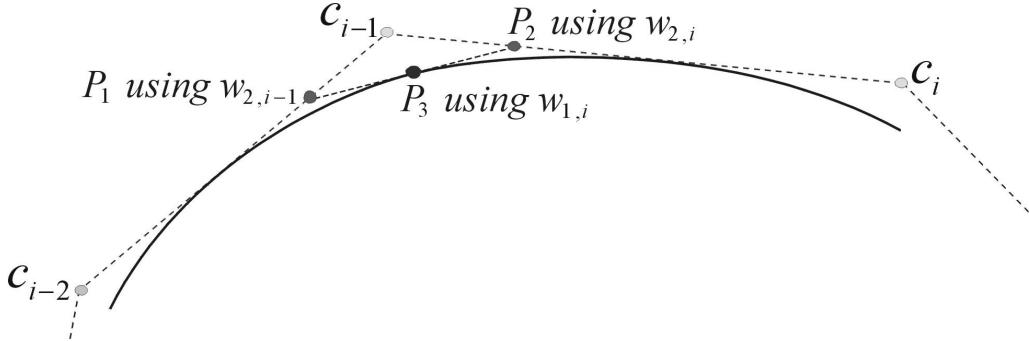


Figure 5.9: The de Casteljau's (corner cutting) algorithm applied on a 2nd degree B-spline curve with the knots $t_{i-1} = 0$, $t_i = 1$, $t_{i+1} = 2$ and $t_{i+2} = 3$. In the example the position is being computed at $t = \frac{3}{2}$.

The first derivative of the 2nd degree B-spline, (5.58), is

$$\begin{aligned} c'(t) &= 2 T(t) T' \mathbf{c} \\ &= 2T(t) \begin{pmatrix} -\delta_{2,i-1} & \delta_{2,i-1} & 0 \\ 0 & -\delta_{2,i} & \delta_{2,i} \end{pmatrix} \begin{pmatrix} c_{i-2} \\ c_{i-1} \\ c_i \end{pmatrix} \\ &= T(t) \begin{pmatrix} 2 \delta_{2,i-1}(c_{i-1} - c_{i-2}) \\ 2 \delta_{2,i}(c_i - c_{i-1}) \end{pmatrix}. \end{aligned}$$

The next example is a 3th degree B-spline curve, that is in matrix notation,

$$c(t) = T^3(t) \mathbf{c}, \quad (5.60)$$

where

$$\begin{aligned} T^3(t) &= T_1(t) T_2(t) T_3(t) \\ &= \begin{pmatrix} 1 - w_{1,i}(t) & w_{1,i}(t) \end{pmatrix} \begin{pmatrix} 1 - w_{2,i-1}(t) & w_{2,i-1}(t) & 0 \\ 0 & 1 - w_{2,i}(t) & w_{2,i}(t) \end{pmatrix} \\ &\quad \begin{pmatrix} 1 - w_{3,i-2}(t) & w_{3,i-2}(t) & 0 & 0 \\ 0 & 1 - w_{3,i-1}(t) & w_{3,i-1}(t) & 0 \\ 0 & 0 & 1 - w_{3,i}(t) & w_{3,i}(t) \end{pmatrix} \quad (5.61) \\ &= (B_{3,i-3}(t), B_{3,i-2}(t), B_{3,i-1}(t), B_{3,i}(t)), \end{aligned}$$

and

$$\mathbf{c} = (c_{i-3}, c_{i-2}, c_{i-1}, c_i)^T,$$

Figure 5.10 shows the four 3th degree B-splines, $B_{3,i-3}(t)$, $B_{3,i-2}(t)$, $B_{3,i-1}(t)$ and $B_{3,i}(t)$ in (5.61) sketch, together with the six “linear translation and scaling” functions that they are constructed from (remember that this is only in the interval $[t_i, t_{i+1}]$). These functions are $w_{1,i}$, in the left hand matrix $T_1(t)$, $w_{2,i-1}$ and $w_{2,i}$ in the second matrix $T_2(t)$, and $w_{3,i-2}$, $w_{3,i-1}$ and $w_{3,i}$ in the last matrix $T_3(t)$. In Figure 5.10 you can see them as dashed line segments.

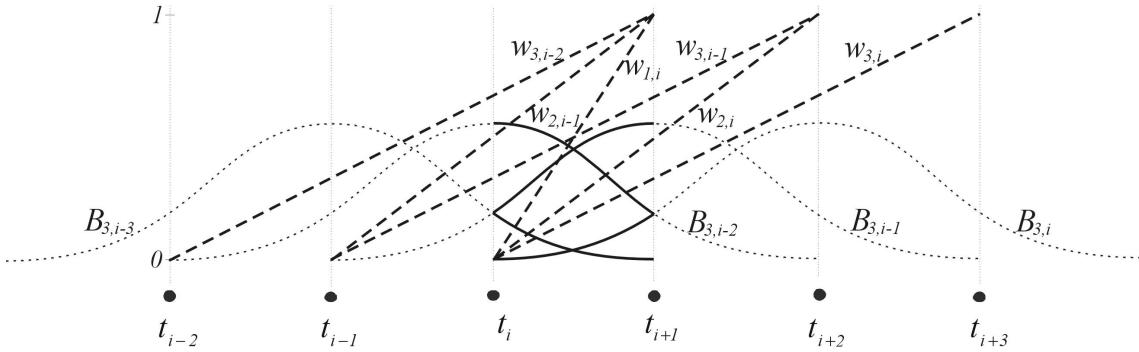


Figure 5.10: The four 3th degree B-splines in the knot interval $[t_i, t_{i+1}]$ (solid) and the six “linear translation and scaling” functions (dashed) involved in the computations of these B-splines.

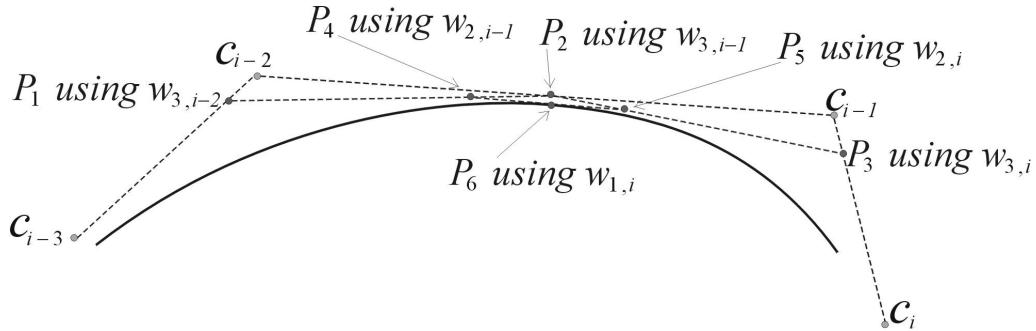


Figure 5.11: The de Casteljau’s (corner cutting) algorithm applied on a 3th degree B-spline curve with the knots $t_{i-2} = 0, t_{i-1} = 1, t_i = 2, t_{i+1} = 3, t_{i+2} = 4$ and $t_{i+3} = 5$. In the example the position is being computed at $t = \frac{5}{2}$.

Figure 5.11 shows de Casteljau’s (corner cutting) algorithm applied on curve (5.60) (a 3th degree B-spline curve) with the knots $t_{i-2} = 0, t_{i-1} = 1, t_i = 2, t_{i+1} = 3, t_{i+2} = 4$ and $t_{i+3} = 5$. In the example $\hat{t} = 5/2$ is used. This gives $w_{1,1}(\hat{t}) = 1/2$ from the first matrix $T_1(t)$, we get $w_{2,i-1}(\hat{t}) = 3/4$ and $w_{2,i}(\hat{t}) = 1/4$ from the second matrix $T_2(t)$, and we get $w_{3,i-2} = 5/6, w_{3,i-1} = 1/2$ and $w_{3,i} = 1/6$ from the last matrix $T_3(t)$. Figure 5.11 shows first three new points p_1, p_2 and p_3 computed by

$$\begin{pmatrix} p_1 \\ p_2 \\ p_3 \end{pmatrix} = \begin{pmatrix} 1 - \frac{5}{6} & \frac{5}{6} & 0 & 0 \\ 0 & 1 - \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 0 & 1 - \frac{1}{6} & \frac{1}{6} \end{pmatrix} \begin{pmatrix} c_{i-3} \\ c_{i-2} \\ c_{i-1} \\ c_i \end{pmatrix} = \begin{pmatrix} \frac{1}{6}c_{i-3} + \frac{5}{6}c_{i-2} \\ \frac{1}{2}c_{i-2} + \frac{1}{2}c_{i-1} \\ \frac{5}{6}c_{i-1} + \frac{1}{6}c_i \end{pmatrix}.$$

The next two new points in Figure 5.11, p_4 and p_5 , are computed by

$$\begin{pmatrix} p_4 \\ p_5 \end{pmatrix} = \begin{pmatrix} 1 - \frac{3}{4} & \frac{3}{4} & 0 \\ 0 & 1 - \frac{1}{4} & \frac{1}{4} \end{pmatrix} \begin{pmatrix} c_{i-2} \\ c_{i-1} \\ c_i \end{pmatrix} = \begin{pmatrix} \frac{1}{4}c_{i-2} + \frac{3}{4}c_{i-1} \\ \frac{3}{4}c_{i-1} + \frac{1}{4}c_i \end{pmatrix}.$$

The last point in Figure 5.11, p_6 (on the curve), is then computed,

$$p_6 = \left(1 - \frac{1}{2} \quad \frac{1}{2} \right) \begin{pmatrix} p_1 \\ p_2 \end{pmatrix} = \left(\frac{1}{2} p_4 + \frac{1}{2} p_5 \right).$$

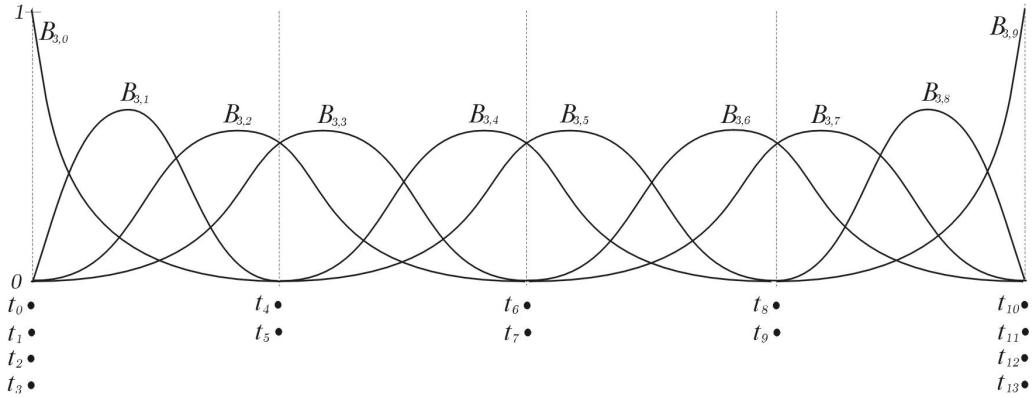


Figure 5.12: The B-spline basis functions for Hermite splines. There are 5 interpolation points and 10 B-splines. The knot values are marked by dots. One can see that the multiplicity of the internal knots is 2.

The first derivative of the 3rd degree B-spline, (5.60), is

$$\begin{aligned}
 c'(t) &= 3 T^2(t) T' \mathbf{c} \\
 &= 3T^2(t) \begin{pmatrix} -\delta_{3,i-2} & \delta_{3,i-2} & 0 & 0 \\ 0 & -\delta_{3,i-1} & \delta_{3,i-1} & 0 \\ 0 & 0 & -\delta_{3,i} & \delta_{3,i} \end{pmatrix} \begin{pmatrix} c_{i-3} \\ c_{i-2} \\ c_{i-1} \\ c_i \end{pmatrix} \\
 &= T^2(t) \begin{pmatrix} 3 \delta_{3,i-2}(c_{i-2} - c_{i-3}) \\ 3 \delta_{3,i-1}(c_{i-1} - c_{i-2}) \\ 3 \delta_{3,i}(c_i - c_{i-1}) \end{pmatrix} \quad (5.62)
 \end{aligned}$$

The second derivative of the 3rd degree B-spline, (5.60), now follows by derivation of (5.62),

$$\begin{aligned}
 c''(t) &= 2 T(t) \begin{pmatrix} -\delta_{2,i-1} & \delta_{2,i-1} & 0 \\ 0 & -\delta_{2,i} & \delta_{2,i} \end{pmatrix} \begin{pmatrix} 3 \delta_{3,i-2}(c_{i-2} - c_{i-3}) \\ 3 \delta_{3,i-1}(c_{i-1} - c_{i-2}) \\ 3 \delta_{3,i}(c_i - c_{i-1}) \end{pmatrix} \\
 &= T(t) \begin{pmatrix} 6 \delta_{2,i-1} (\delta_{3,i-1}(c_{i-1} - c_{i-2}) - \delta_{3,i-2}(c_{i-2} - c_{i-3})) \\ 6 \delta_{2,i} (\delta_{3,i}(c_i - c_{i-1}) - \delta_{3,i-1}(c_{i-1} - c_{i-2})) \end{pmatrix}.
 \end{aligned}$$

5.5.7 Hermite spline interpolation on B-spline form

In section 5.1.3, page 105–106, cubic Hermite splines were shown in both algebraic and geometric form. We shall now look at cubic Hermite spline in B-spline form.

Given m strictly increasing real numbers $\{x_i\}_{i=1}^m$, m points $\{p_i\}_{i=1}^m$ and m respective vectors $\{v_i\}_{i=1}^m$. We have to make a cubic Hermite spline on B-spline form,

$$c(t) = T^3(t) \mathbf{c},$$

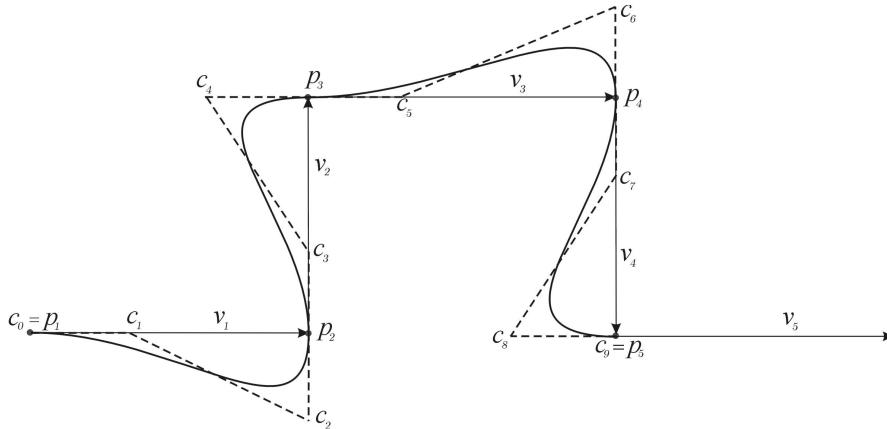


Figure 5.13: An interpolation of 5 points, $\{p_i\}_{i=1}^5$, and respective vectors $(\{v_i\}_{i=1}^5$ marked by arrows) by Hermite spline on B-spline form. The control polygon is dashed and the control points $\{c_i\}_{i=0}^9$ are marked. The resulting curve is solid.

that interpolates the points and the vectors in the given real numbers, i.e. $c(x_i) = p_i$ and $c'(x_i) = v_i$. We first make a knot vector, $\{t_i\}_{i=0}^{n+3}$, where $n = 2m$. We do this by setting

$$\begin{aligned} \text{at the start: } & t_0 = t_1 = t_2 = t_3 = x_1 \\ \text{at the end: } & t_n = t_{n+1} = t_{n+2} = t_{n+3} = x_m \\ \text{and otherwise: } & t_i = t_{i+1} = x_j, \quad \text{for } i = 4, 6, 8, \dots, 2(m-1) \quad \text{and } j = \frac{i}{2}. \end{aligned} \tag{5.63}$$

We then make the control points, that are constructed by setting

$$c_0 = p_1 \quad \text{and} \quad c_{n-1} = p_m,$$

and

$$\begin{aligned} c_i &= p_j + \frac{\Delta x_j}{3} v_j, & \text{for } i = 1, 3, \dots, n-3 & \text{where } j = \frac{i+1}{2}, \\ c_i &= p_{j+1} - \frac{\Delta x_j}{3} v_{j+1}, & \text{for } i = 2, 4, \dots, n-2 & \text{where } j = \frac{i}{2}. \end{aligned} \tag{5.64}$$

Here $\Delta x_j = x_{j+1} - x_j$. In Figure 5.12 the B-splines are plotted in an example where $m = 5$. One can observe in the figure, and recovered from procedure (5.63), that all the inner knots are of multiplicity 2. This is also according to the fact that a Hermite spline is C^1 -smooth. A 3rd degree B-spline is $C^2[t_0, t_n]$ when we only have simple internal knots, so to get it to $C_1[t_0, t_n]$ we must use multiple internal knots. In Figure 5.12 we can observe that there are 5 clusters of points, with multiplicity 2 in the internal and 4 at the ends. The position (parameter value) of these clusters is the position (parameter value) of the interpolation points. Over the internal knots there are only two B-splines different from zero. This indicates that the interpolation points lie on the control polygon, i.e. on a straight line between two control points. This is also the way we have constructed the control points in (5.64).

Figure 5.13 shows the B-spline curve in the example plotted (thick solid) together with the control polygon (dashed) and where the 9 control points $\{c_i\}_{i=0}^9$ are marked. Also the

5 interpolation points $\{p_i\}_{i=1}^5$ are marked together with their respective (tangent) vectors $\{v_i\}_{i=1}^5$ (marked as arrows).

However, a tension parameter is available in this curve construction. This is the parameter values at the interpolating points, i.e the values $\{x_i\}_{i=1}^m$. If these parameter values are scaled up, the curve will be longer, i.e. it will boucle more, and if they are scaled down the curve will be shorter. This can also be done locally.

5.5.8 Cubic spline interpolation on B-spline form

In section 5.3 a cubic spline interpolation is described. We shall now look at cubic spline interpolation on B-spline form.

In the previous case we used Hermite interpolation as the basic form, even though the curve was C^2 -smooth. This was the reason for the unknowns to be the derivatives at the interpolation points. In the B-spline case is it natural that the unknowns are the coefficients (control points).

Given m strictly increasing real numbers $\{x_i\}_{i=1}^m$ and m points $\{p_i\}_{i=1}^m$. We will now construct a 3th degree B-spline curve,

$$c(t) = T^3(t) \mathbf{c},$$

that interpolates the given points at the given real numbers, i.e. $c(x_i) = p_i$, $i = 1, 2, \dots, m$. It follows, because we use third degree B-splines, that the number of knot values must be $m + 6$. This is because we want to have single knots in the interior (because the curve is actually C^2 -smooth) and four knots at each end (i.e. $3+3=6$ more than single knots everywhere). Since the number of knots is the number of coefficients plus the order of the B-spline (that is 4) it follows that $n = m + 2$. The knot vector must be

$$\begin{aligned} \text{at the start: } & t_0 = t_1 = t_2 = t_3 = x_1 \\ \text{at the end: } & t_n = t_{n+1} = t_{n+2} = t_{n+3} = x_m \\ \text{and otherwise: } & t_i = x_{i-2}, \quad \text{for } i = 4, 5, \dots, n-1. \end{aligned}$$

Figure 5.14 shows the knot vector and the B-splines it generates. Now, to compute the control point, using $k = 4$ equal knots at the ends, it follows that coefficients are equal the interpolation points, i.e.

$$c_0 = p_1 \quad \text{and} \quad c_{n-1} = p_{n-2}.$$

To compute the remaining control points we have to solve a system of linear equations similar to equation (5.28) in section 5.3, i.e.

$$\mathbf{A} \mathbf{c} = \mathbf{b} \tag{5.65}$$

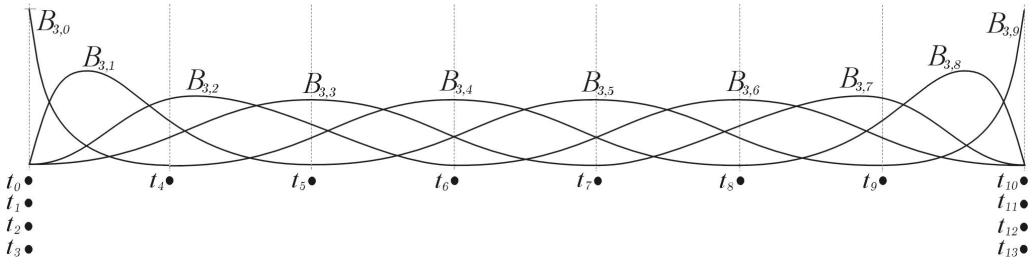


Figure 5.14: The figure shows the 10 3th degree B-splines that are generated by a knot vector of 14 elements, $\{t_i\}_{i=0}^{13}$, that is used for cubic spline interpolation of 8 points. As we can see, only 3 B-splines are different from zero at the internal knots. That is why the matrix in the equation (5.65), cubic spline interpolation, is three diagonal.

where

$$\mathbf{A} = \begin{pmatrix} B''_{3,1}(t_3) & B''_{3,2}(t_3) & 0 & \cdots & \cdots & 0 \\ B_{3,1}(t_4) & B_{3,2}(t_4) & B_{3,3}(t_4) & 0 & \ddots & \vdots \\ 0 & B_{3,2}(t_5) & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & B_{3,n-4}(t_{n-2}) & B_{3,n-3}(t_{n-2}) & 0 \\ \vdots & \ddots & 0 & B_{3,n-4}(t_{n-1}) & B_{3,n-3}(t_{n-1}) & B_{3,n-2}(t_{n-1}) \\ 0 & \cdots & \cdots & 0 & B''_{3,n-3}(t_n) & B''_{3,n-2}(t_n) \end{pmatrix},$$

and

$$\mathbf{c} = \begin{pmatrix} c_1 \\ \vdots \\ \vdots \\ c_{n-2} \end{pmatrix} \quad \text{and} \quad \mathbf{b} = \begin{pmatrix} -B''_{3,0}(t_3) c_0 \\ p_2 \\ \vdots \\ p_{n-3} \\ -B''_{3,n-1}(t_n) c_{n-1} \end{pmatrix}.$$

We have, in this equation, used the free end condition, which is that the second derivative is zero at start and end (see (5.32) in section 5.3).

In Figure 5.14 there is an example of the B-splines in cubic spline interpolation, where $m = 8$ (you can see that there are 8 clusters of knots). It follows that there are 10 B-splines, $n = 10$, and that there are 14 knots (marked with dots). If we look at the left side of the figure we can see that there are only 4 B-splines at the start that are infecting, $B_{3,0}, B_{3,1}, B_{3,2}$ and $B_{3,3}$. The second derivative of $B''_{3,3}(t_3) = 0$. It, therefore, follows that

$$c''(t_3) = B''_{3,0}(t_3) c_0 + B''_{3,1}(t_3) c_1 + B''_{3,2}(t_3) c_2$$

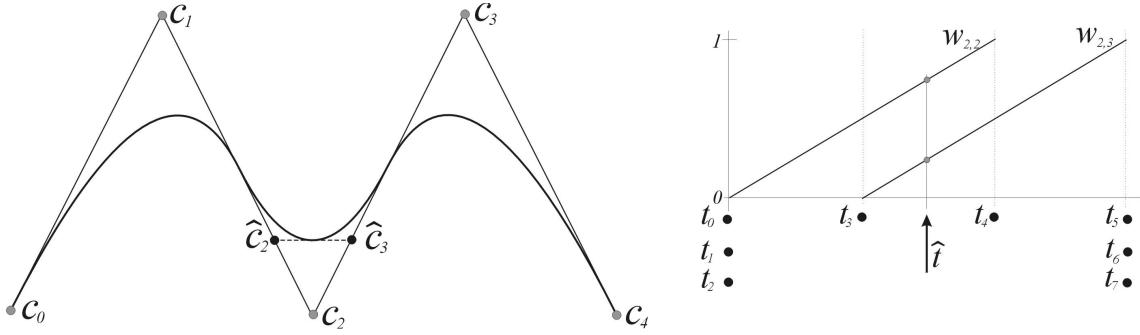


Figure 5.15: On the left hand side there is a 2nd degree B-spline curve and its control polygon (in \mathbb{R}^2). On the right hand side the knot vector is illustrated (in \mathbb{R}). One new knot, \hat{t} is inserted. The result is, as we can see from the left hand side, two new control points \hat{c}_2 and \hat{c}_3 instead of the old control point c_2 .

has to be zero to fulfill the requirement. The same must be the case on the right hand side (the end of the curve) i.e.

$$c''(t_{10}) = B''_{3,7}(t_{10}) c_7 + B''_{3,8}(t_{10}) c_8 + B''_{3,9}(t_{10}) c_9.$$

These two requirements are actually the first and the last line of the equation (5.65). In all the internal knots only three B-splines are different from zero (see Figure 5.14). To interpolate in these internal knots we can see from the figure that

$$c(t_i) = B_{3,i-3}(t_i) c_{i-3} + B_{3,i-2}(t_i) c_{i-2} + B_{3,i-1}(t_i) c_{i-1} = p_{i-2},$$

which is actually what all the other lines in equation (5.65) say.

5.5.9 B-splines and knot insertion

Knot insertion is one of the most important algorithms on B-splines. It was introduced simultaneously in 1980 by Boehm [13] for single knots and Cohen, Lyche and Riesenfeld [22] for general knot insertion.

Knot insertion can also be expressed in the context of matrix notation. In the case of insertion of single knots we have the following expression

$$\hat{\mathbf{c}} = T_d(\hat{t}) \mathbf{c} \quad (5.66)$$

where \hat{t} is the value of the new knot and $\hat{\mathbf{c}}$ is a vector of the d new coefficients replacing the $d - 1$ coefficients that are in the interior of the vector \mathbf{c} on the right hand side (except for the first and the last coefficients). Here, as usual, the index i is determined by $t_i \leq \hat{t} < t_{i+1}$. We shall look at some examples, first a second degree B-spline function where we get:

$$\begin{pmatrix} \hat{c}_{i-1} \\ \hat{c}_i \end{pmatrix} = \begin{pmatrix} 1 - w_{2,i-i}(\hat{t}) & w_{2,i-i}(\hat{t}) & 0 \\ 0 & 1 - w_{2,i}(\hat{t}) & w_{2,i}(\hat{t}) \end{pmatrix} \begin{pmatrix} c_{i-2} \\ c_{i-1} \\ c_i \end{pmatrix}. \quad (5.67)$$

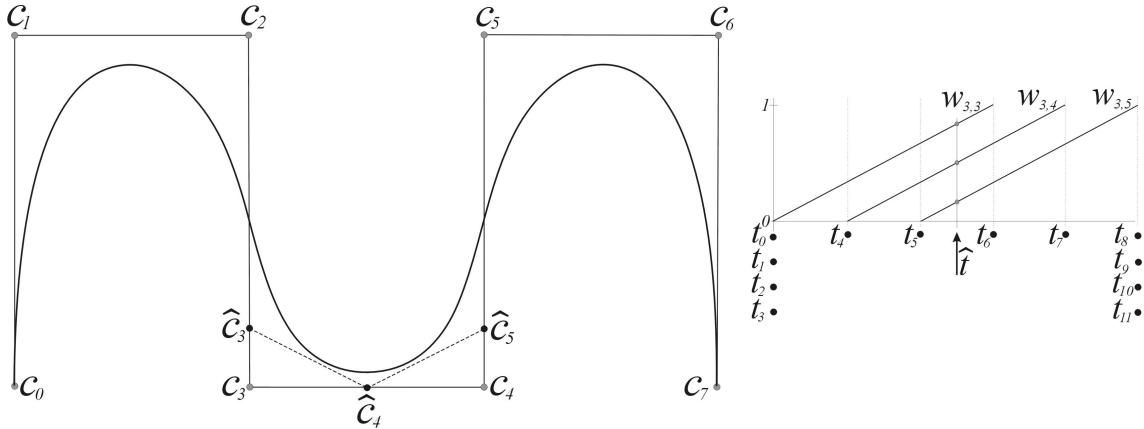


Figure 5.16: On the left hand side there is a 3th degree B-spline curve and its control polygon (in \mathbb{R}^2). On the right hand side the knot vector is illustrated (in \mathbb{R}). One new knot, \hat{t} is inserted. The result is, as we can see on the left hand side, three new control points \hat{c}_3 , \hat{c}_4 and \hat{c}_5 instead of the two old control points c_3 and c_4 .

In Figure 5.15 is there, on the left hand side, a second degree B-spline curve (in \mathbb{R}^2) and its control polygon (c_0, c_1, c_2, c_3, c_4). On the right hand side there is the knot vector illustrated (on \mathbb{R} , horizontal). There are three equal knots at the beginning and at the end of the curve. In the middle a new knot, \hat{t} , is to be inserted. It follows from the position of \hat{t} that $i = 3$. The two “linear translation and scaling” functions involved in the matrix $T_2(\hat{t})$ in expression (5.67), $w_{2,2}$ and $w_{2,3}$ are shown on the right hand side of the figure. The result of the knot insertion is that the internal coefficient on the right hand side of (5.67), c_2 , is replaced by two new coefficients on the left hand side of (5.67), \hat{c}_2 and \hat{c}_3 . This can clearly be seen on the left hand side of Figure 5.15.

We shall now look at a third degree B-spline function where we get:

$$\begin{pmatrix} \hat{c}_{i-2} \\ \hat{c}_{i-1} \\ \hat{c}_i \end{pmatrix} = \begin{pmatrix} 1 - w_{3,i-2}(\hat{t}) & w_{3,i-2}(\hat{t}) & 0 & 0 \\ 0 & 1 - w_{3,i-1}(\hat{t}) & w_{3,i-1}(\hat{t}) & 0 \\ 0 & 0 & 1 - w_{3,i}(\hat{t}) & w_{3,i}(\hat{t}) \end{pmatrix} \begin{pmatrix} c_{i-3} \\ c_{i-2} \\ c_{i-1} \\ c_i \end{pmatrix} \quad (5.68)$$

In Figure 5.16 is there, on the left hand side, a third degree B-spline curve (in \mathbb{R}^2) and its control polygon ($c_0, c_1, c_2, c_3, c_4, c_5, c_6, c_7$). On the right hand side there is the knot vector illustrated (on \mathbb{R} , horizontal). There are four equal knots at the beginning and at the end of the curve. In the middle a new knot, \hat{t} , is to be inserted. It follows from the position of \hat{t} that $i = 5$. The three “linear translation and scaling” functions involved in the matrix $T_3(\hat{t})$ in expression (5.68), $w_{3,3}$, $w_{3,4}$ and $w_{3,5}$ are shown on the right hand side of the figure. The result of the knot insertion is that the internal coefficients on the right hand side of (5.68), c_3 and c_4 are replaced by the three new coefficients on the left hand side of (5.68), \hat{c}_3 , \hat{c}_4 and \hat{c}_5 . This can clearly be seen on the left hand side of Figure 5.16.

From subsection 5.5.2 and definition 5.5 is it clear that the matrix $T_d(t)$ is a corner cutting matrix. It is, therefore, obvious that knot insertion is corner cutting. From the two examples we can also see that the new control points are lying on the old control polygon if we

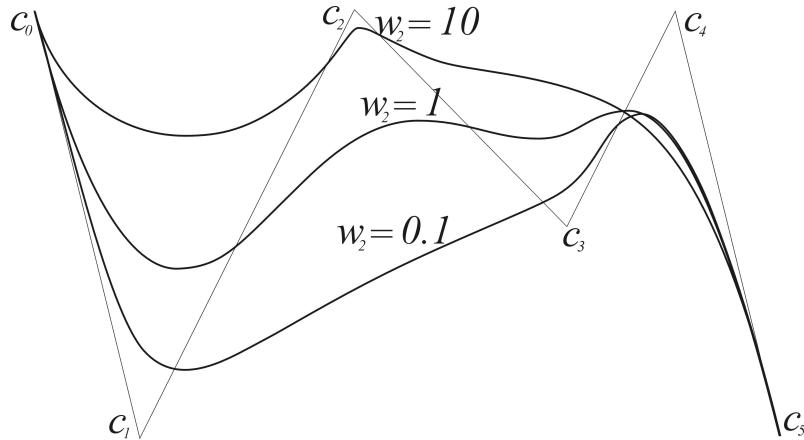


Figure 5.17: Three versions of a Rational Bezier Curve (NURBS without internal knots) It is only the weight, w_2 , of the control point c_2 that has different values in the three examples. The control polygon is the same for all three curves.

insert one new knot. This is related to discrete B-splines,¹⁴ and it leads us to subdivision techniques, that are especially developed for so called subdivision-surfaces.¹⁵

5.5.10 NURBS

NURBS is short for non uniform rational B-splines. Non uniform B-splines are, as typical modern B-splines, defined by knots that can be arbitrary spaced. This is contrary to the uniform B-spline defined (on an implicit integer knot vector) by Schoenberg in [120, 121]. To describe rational B-splines it is necessary to know about homogeneous coordinates, frequently used in graphical systems like OpenGL. A homogeneous coordinate system is connected to a projective space, \mathbb{P}^n .¹⁶ A concrete description is that \mathbb{P}^n can be defined as the space of all infinite straight lines in \mathbb{R}^{n+1} going through the origin. The main effect is that there is one extra coordinate compared to an equivalent Euclidian/affine¹⁷ space. We have $q = (x, y, z, w)$, when q is an element in a 3D space. Using this description it follows that $q \in \mathbb{P}^3$ can be expressed by

$$q = (kx, ky, kz, kw),$$

¹⁴Discrete splines (on a uniform grid) were introduced by Mangasarian and Schumaker in [100] as solutions to certain variational problems. They were discussed in detail by Tom Lyche in his Ph.D. Thesis, “Discrete polynomial spline approximation methods”, for which there is a summary in [97].

¹⁵There are several algorithms for subdivision surfaces, i.g Doo-Sabin subdivision algorithm published in [48, 49] and Catmull-Clark subdivision algorithm published in [19].

¹⁶For a closer study of projective spaces see e.g. [9] or [28]. For shorter studies see internet pages http://en.wikipedia.org/wiki/Projective_space or <http://planetmath.org/encyclopedia/ProjectiveSpace.html>.

¹⁷An affine space is an abstract structure that generalizes the affine-geometric properties of Euclidean space. In an affine space, one can subtract points to get vectors, or add a vector to a point to get another point, but one cannot add points, since there is no origin. See page 136. For more deeply studies see e.g. http://en.wikipedia.org/wiki/Affine_space.

where q is independent of k , i.e. k can be any nonzero real.

There is a canonical injection of \mathbb{R}^n into \mathbb{P}^n . This means that an affine space \mathbb{R}^n can be embedded isomorphically in \mathbb{P}^n by the standard injection

$$(x_1, \dots, x_n) \mapsto (x_1, \dots, x_n, 1).$$

Affine points can be recovered from projective ones with the mapping

$$(x_1, \dots, x_n, x_{n+1}) \sim \left(\frac{x_1}{x_{n+1}}, \dots, \frac{x_n}{x_{n+1}}, 1 \right) \mapsto \left(\frac{x_1}{x_{n+1}}, \dots, \frac{x_n}{x_{n+1}} \right).$$

Definition 5.8. Non uniform rational B-splines (NURBS) are B-splines, defined on a non uniform knot vector, in a projective space and mapped into an affine space. It follows that a B-spline in \mathbb{P}^n is

$$c(t) = T^d(t) \mathbf{c},$$

where each element c_j , $j = i-d, i-d+1, \dots, i$ of \mathbf{c} are given in homogenous coordinates

$$c_j = w_j(x_{j,1}, \dots, x_{j,n}, 1).$$

Finally, mapping this element from the projective space \mathbb{P}^n to an affine space \mathbb{R}^n gives

$$c(t) = \frac{T^d(t) \mathbf{c}}{T^d(t) \mathbf{w}}, \quad (5.69)$$

where $\mathbf{w} = (w_{i-d}, w_{i-d+1}, \dots, w_i)^T$, which is actually the NURBS definition.

It follows from expression (5.69) that if $w_i = 1$ for all i , then the NURBS is an ordinary B-spline function in an affine space. To sketch the control polygon the coefficients c_i , $i = 0, \dots, m$, where $m + 1$ is the number of coefficients, is mapped from \mathbb{P}^n to \mathbb{R}^n , i.e.

$$w_i(x_{i,1}, \dots, x_{i,n}, 1) \mapsto (x_{i,1}, \dots, x_{i,n}).$$

Figure 5.17 shows a rational Bezier curve (B-spline without interior knots) plotted with the weight of the third coefficient having the values $w_2 = 0.1$, $w_2 = 1$ and $w_2 = 10$. The effect is clearly demonstrated, as we can see, a small weight is pushing the curve from the coefficient, while a big weight is pulling it towards the coefficient.

5.5.11 Important properties of spline functions on B-spline form

There are four special properties that are of great importance for explaining why the B-spline (and also NURBS) are so popular:

1. A B-spline curve has a geometric variation diminishing property, that is, every infinite line (in \mathbb{R}^2 , or infinite plane in \mathbb{R}^3 etc.) that intersects a B-spline curve is also intersecting the control polygon the same number or more times than it intersects the curve (in Figure 5.18 is this property illustrated).

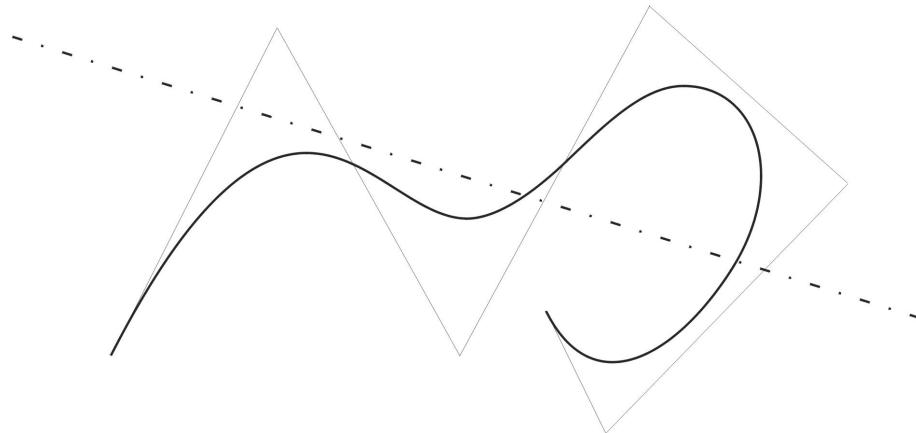


Figure 5.18: A 3th degree B-spline curve and its control polygon. A line is intersecting the curve 2 times and the control polygon 4 times.

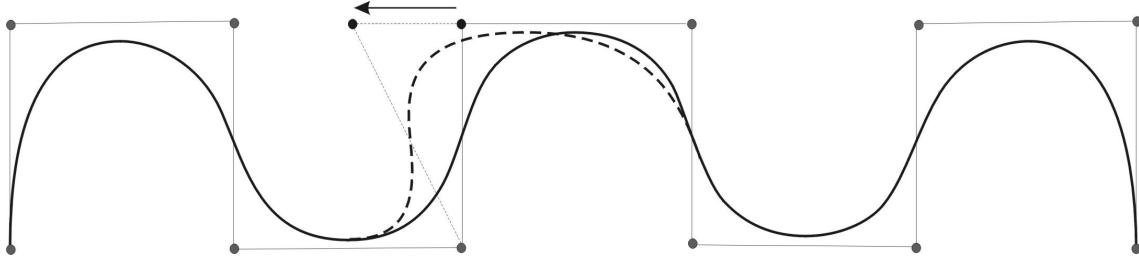


Figure 5.19: A 3th degree B-spline curve and its control polygon. One of the control points is moved to the left. To illustrate the local control, part of the curve that is changed is also plotted (dashed).

2. There is a convex hull property of the control points for B-spline curves, that is, the convex hull of the control points is entirely covering the curve. This follows from the convex affine combination property of B-splines (explained in section 5.7). Figure 5.18 also illustrate this property.
3. The control polygon of a B-spline curve is converging towards the B-spline curve when the number of knots goes towards infinity. This can be clearly seen from the knot insertion algorithm shown in section 5.5.9.
4. B-splines offer local control (low degree polynomials), this means, moving a control point only changes the curve in the $d + 1$ knot intervals that are the domain of the appurtenant B-spline. Figure 5.19 illustrates this.

5.6 Blossoming

5.7 Use of blending in construction of geometry

The affine space is a space of points, \mathbf{p} , and vectors, \mathbf{v} . The following operations are legal:

$$\begin{aligned}\mathbf{p} &= \mathbf{p}_1 + k \mathbf{v}, \\ \mathbf{v} &= k_1 \mathbf{v}_1 + k_2 \mathbf{v}_2.\end{aligned}\tag{5.70}$$

where the k 's are real values. From these two rules we can derive; from the first line in (5.70),

$$\mathbf{v} = k (\mathbf{p}_1 - \mathbf{p}_2)$$

by turning the expression. From a combination of all three expressions above we get,

$$\mathbf{p} = \sum_{i=0}^n k_i \mathbf{p}_i, \quad \text{where} \quad \sum_{i=0}^n k_i = 1.\tag{5.71}$$

This because (5.71) can be rewritten to

$$\mathbf{p} = \mathbf{p}_0 + \mathbf{v}, \quad \text{where} \quad \mathbf{v} = \sum_{i=1}^n k_i (\mathbf{p}_i - \mathbf{p}_0),$$

and it, therefore, follows that

$$k_0 = 1 - \sum_{i=1}^n k_i.$$

The expression (5.71) is called the affine combination (or barycentric combination), and it is a very important operation. If all coefficients k_i , $i = 0, \dots, n$ are nonnegative, we call (5.71) for a convex affine combination.

The basis for spline functions, the B-splines, forms a convex affine combination, i.e. for B-splines of degree d and for $t_i \leq t < t_{i+1}$ is

$$B_{d,i-d}(t) + \dots + B_{d,i-1}(t) + B_{d,i}(t) = 1.$$

This follows from the fact that each of the lines in all of the matrices $T^d(t)$ perform a linear interpolation, so that the elements in each line sums up to 1.¹⁸

Basis functions that fulfill an affine combination are invariant under affine maps. These are the most used maps in computer graphics, CAD/CAM etc. They are; translation, scaling, rotation, shear and parallel projections, and are in general taken on the familiar form

$$\Theta \mathbf{p} = A \mathbf{p} + \mathbf{v}$$

where \mathbf{p} is a point and \mathbf{v} is a vector in an affine space \mathbb{R}^3 , and A is a 3×3 matrix.

¹⁸The Bernstein polynomials also sum up to 1, and so do the two first basis functions (that are connected to points) in Hermite interpolation, see (5.25).

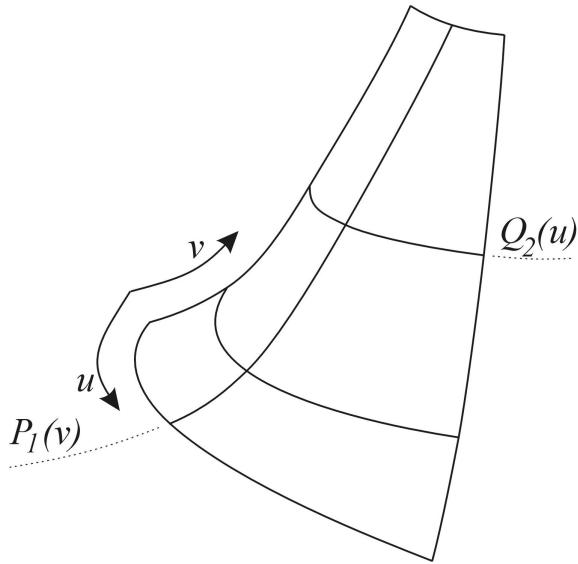


Figure 5.20: Example of a Gordon surface, a rectangular network of curves is given and the result is an interpolating surface.

B-splines are blending points, and Hermite interpolation in (5.25) is blending points and vectors. We shall now look at methods that use higher dimensional geometric objects¹⁹ in the blending. These methods are also called **transfinite interpolation** methods.

William J. Gordon introduced in the late 1960s, [68], a generalization of Coons patch. It is based on a net of curves, see Figure 5.20. The procedure is the same as in Coons patch, first interpolation in the two directions, then interpolation of the crossing points between all curves and finally the “affine combination” of these three surfaces to get the final surface.

We shall now look at an interpolation method which blends circular arcs. In 1996 Hans-Jörg Wenz published an interpolation method based on blending circular arcs, [148]. Given a sequence of points \$\{p_i\}_{i=0}^n\$. In each internal point \$p_i\$, \$i = 1, 2, \dots, n - 1\$ there is a circular arc \$\vartheta_i(u)\$, \$u \in [0, 1]\$ starting in \$p_{i-1}\$, interpolating \$p_i\$ and ending in \$p_{i+1}\$ but parameterized from 0 to 1 in both segments, \$[p_{i-1}, p_i]\$ and \$[p_i, p_{i+1}]\$. To make a smooth curve segment between two point \$p_i\$ and \$p_{i+1}\$, \$i = 0, 1, \dots, n - 1\$, Wench used a simple blending scheme (giving a \$C^1\$-smooth curve),

$$c_i(u) = \begin{pmatrix} 1-u & u \end{pmatrix} \begin{pmatrix} \vartheta_i(u) \\ \vartheta_{i+1}(u) \end{pmatrix}.$$

Wenz was followed by Márta Szilvási-Nagy and Teréz P. Vendel, [138]. They improved the blending scheme, by replacing the linear interpolation function with a trigonometric

¹⁹The dimension of a geometric object can briefly be described as follows; if there exist an homeomorphism between an Euclidean space \$R^n\$ and open sets in the object, the dimension is \$n\$. Thus, a point has dimension 0, a curve 1 and a surface 2. If the object is parameterized then the number of parameter is equal to the dimension.

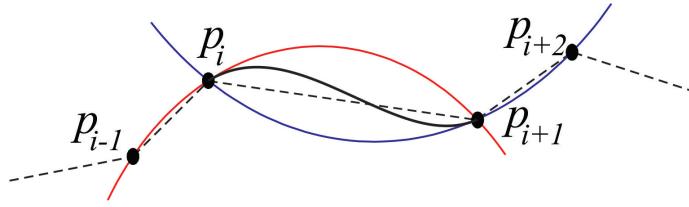


Figure 5.21: Example of Circle splines. Between the point p_i and p_{i+1} are two circular arcs blended to one curve. As we can see, the curve segment between the points p_i and p_{i+1} is only defined by the four points: p_{i-1} , p_i , p_{i+1} and p_{i+2} .

cally weighted blending function that also sums up to 1 (giving a C^2 -smooth curve),

$$c_i(u) = \begin{pmatrix} \cos^2\left(\frac{\pi u}{2}\right) & \sin^2\left(\frac{\pi u}{2}\right) \end{pmatrix} \begin{pmatrix} \vartheta_i(u) \\ \vartheta_{i+1}(u) \end{pmatrix}.$$

Figure 5.21 shows an example of constructing a circle spline curve by blending two circular arcs using a trigonometrically weighted blending function.

To avoid loops and cusps Carlo Séquin, Jane Yen Kiha Lee introduced another blending technic (see [129, 128]). They used the trigonometrically weighted blending function to compute the directional angle $\tau(u)$ from τ_i and τ_{i+1} ,

$$\tau(u) = \begin{pmatrix} \cos^2\left(\frac{\pi u}{2}\right) & \sin^2\left(\frac{\pi u}{2}\right) \end{pmatrix} \begin{pmatrix} \tau_i \\ \tau_{i+1} \end{pmatrix}.$$

The angle τ_i is the angle between the line segment $[p_i, p_{i+1}]$ and the tangent $\vartheta'_i(0)$. They then compute the position on the curve $c_i(u)$ by computing the angle between the line segment $[p_i, p_{i+1}]$ and the line segment $[p_i, c_i(u)]$,

$$\phi(u) = (1 - u)\tau(u),$$

and the distance $|p_i, c_i(u)|$ by

$$|p_i, c_i(u)| = \frac{\sin(u \tau(u))}{\sin(\tau(u))} |p_i, p_{i+1}|.$$

5.8 Subdivision

5.8.1 Catmull-Rom Spline

The points that define a spline are known as "Control Points". One of the features of the Catmull-Rom spline is that the specified curve will pass through all of the control points.

To calculate a point on the curve, two points on either side of the desired point are required

$$\alpha(t) = (1, t, t^2, t^3) \begin{bmatrix} 0 & 1 & 0 & 0 \\ -\frac{1}{2} & 0 & \frac{1}{2} & 0 \\ 1 & -\frac{5}{2} & 2 & -\frac{1}{2} \\ -\frac{1}{2} & \frac{3}{2} & -\frac{3}{2} & \frac{1}{2} \end{bmatrix} \begin{pmatrix} p_{i-1} \\ p_i \\ p_{i+1} \\ p_{i+2} \end{pmatrix}.$$

Chapter 6

Blending

Blending of points is the technique that Bézier and B-splines are based on, and Hermite curves are based on blending of points and vectors. It is therefore of interest to investigate blending in more detail, and also expanding the blending technics to blend functions in general, not only constants as points or vectors. We start with defining the B-function.

6.1 B-functions

B-function is short for blending function. It is to be used for blending functions or vector valued functions or point-valued functions such as points, curves, tensor product or triangular surfaces etc. In the following, we restrict B functions to be monotone, and we will also look specific at the properties of symmetry in the blending. The definition is:

Definition 6.1. A *B-function* is:

D1 a permutation function $B : I \rightarrow I$ ($I = [0, 1] \subset \mathbb{R}$),

D2 where $B(0) = 0$,

D3 and $B(1) = 1$,

D4 and that is monotone, i.e. $B'(t) \geq 0$, $t \in I$.

D5 A B-function is called symmetric if, $B(t) + B(1-t) = 1$, $t \in I$.

This symmetry is a point symmetry (around the point $(0.5, 0.5)$). Other types of symmetry will be introduced later in sections 6.8. To give an idea of what a B-function is, we will look at four simple examples of symmetric B-functions:

- a) linear function $B(t) = t$
- b) trigonometric function $B(t) = \sin^2 \frac{\pi}{2} t$
- c) polynomial function of first order $B(t) = 3t^2 - 2t^3$
- d) rational function of first order $B(t) = \frac{t^2}{t^2 + (1-t)^2}$

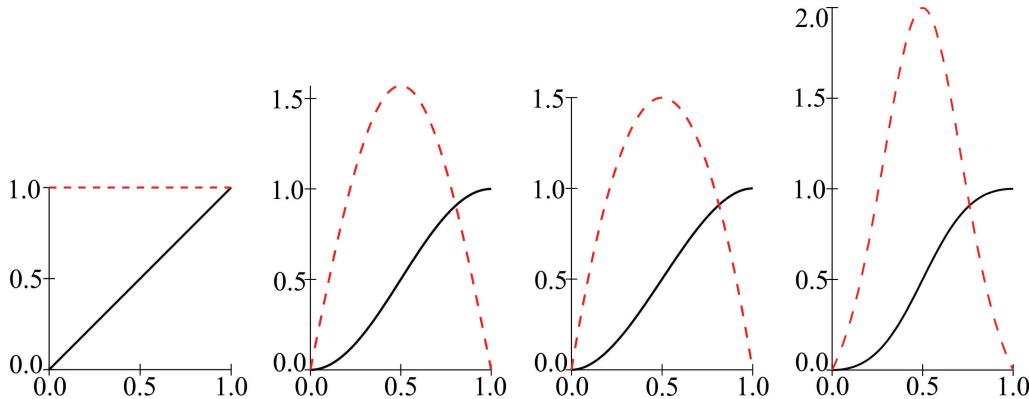


Figure 6.1: Four B-functions (solid black) and their derivatives (dashed red). To the left is $B(t) = t$, next is $B(t) = \sin^2 \frac{\pi}{2}t$, then $B(t) = 3t^2 - 2t^3$ and to the right is $B(t) = \frac{t^2}{(1-t)^2+t^2}$.

Figure 6.1 shows these four example of B-functions plotted together with their derivatives. We can clearly see from the figure that these four functions all starts at 0 and ends at 1 and that they are all monotone. The symmetry follows from;

- a) $B(t) + B(1-t) = t + 1 - t = 1$
- b) $B(t) + B(1-t) = \sin^2 \frac{\pi}{2}t + \sin^2 \frac{\pi}{2}(1-t) = \sin^2 \frac{\pi}{2}t + \cos^2 \frac{\pi}{2}t = 1$
- c) $B(t) + B(1-t) = 3t^2 - 2t^3 + 3(1-t)^2 - 2(1-t)^3 = 3t^2 - 2t^3 - 3t^2 + 2t^3 + 1 = 1$
- d) $B(t) + B(1-t) = \frac{t^2}{t^2+(1-t)^2} + \frac{(1-t)^2}{(1-t)^2+t^2} = 1$

B-functions can be organized in groups. The examples a), b), c) and d) are all members of different groups of B-functions. Later we will conduct a thorough investigation of some of the groups of B-functions and their special properties. But first let's look at the definition of an important property.

Definition 6.2. A property that plays an important role is the number of subsequent derivatives which is zero at start and end. We call this

The order of B-functions

The order of a B-function, short for the Hermite order of a B-function, denoted S , is for a symmetric B-function determined by

$$B^{(j)}(0) = B^{(j)}(1) = 0, \quad j = 1, 2, \dots, S. \quad (6.1)$$

For a non-symmetric B-function we have to differ between the start and the end,

$$\begin{aligned} B^{(j)}(0) &= 0, & j &= 1, 2, \dots, S_0 \\ B^{(j)}(1) &= 0, & j &= 1, 2, \dots, S_1. \end{aligned} \quad (6.2)$$

This is the Hermite property to a B-function, explained further in Theorem 6.1.

The three examples b), c) and d) all have $B'(0) = 0$, i.e. the first derivative is zero (only). Because of the symmetry is also $B'(1) = 0$. It follows that these are 1st order B-functions.

We will later in this chapter look at higher order B-functions, and also complete B-functions, where all derivatives are zero at start and end.

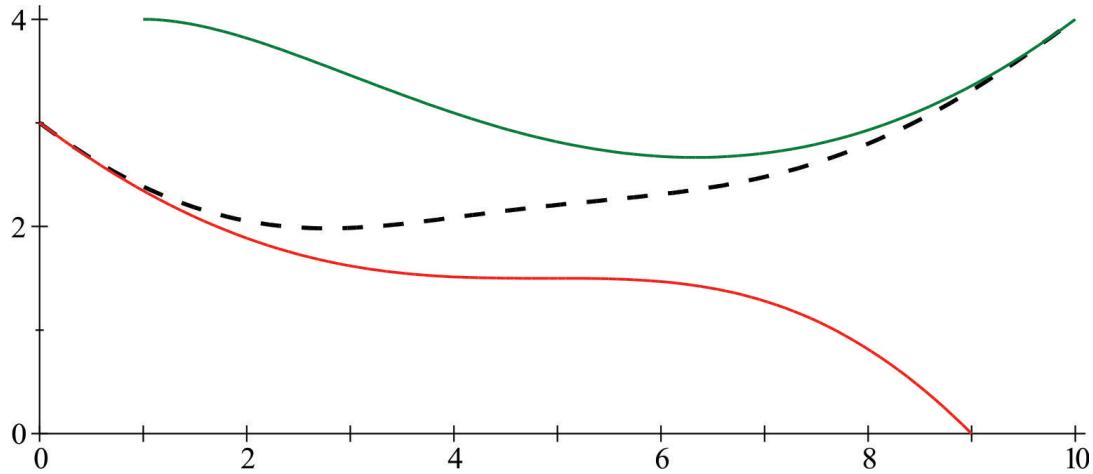


Figure 6.2: In the figure is two Bézier-curves blended to one curve using a B-function. The original Bézier-curves are shown in solid red and solid green and the resulting curve is shown in dashed black.

6.2 Blending of two functions

The most simple blending is blending of two functions (curves), $g_1(t)$ and $g_2(t)$, organized in a sequence where $g_1(t)$ is the first function and $g_2(t)$ is the last function. In Figure 6.2 is an example shown. Two Bézier-curves are plotted in solid red, $g_1(t)$, and solid green, $g_2(t)$. The result of the blending is shown as a dashed black curve. The B-function that is used in the plot is $B(t) = 3t^2 - 2t^3$. The behavior of the resulting curve comes from the sequence of the original curves, and the order of the B-function used in the blending. This will be further discussed later in this section. First, we look at the formulas.

The formulas for a two functions blending

The formulas for a two functions blending is

$$\begin{aligned} f(t) &= (1 - B(t)) g_1(t) + B(t) g_2(t) \\ &= g_1(t) + B(t) (g_2(t) - g_1(t)). \end{aligned} \tag{6.3}$$

where $B(t)$ is a B-function. If we denote the difference function $h(t) = g_2(t) - g_1(t)$ we get the formula

$$f(t) = g_1(t) + B(t) h(t). \tag{6.4}$$

The first derivative is

$$f'(t) = g'_1(t) + B(t) h'(t) + B'(t) h(t), \tag{6.5}$$

and the general formula for derivatives is

$$f^{(j)}(t) = g_1^{(j)}(t) + \sum_{i=0}^j \binom{j}{i} B^{(i)}(t) h^{(j-i)}(t). \tag{6.6}$$

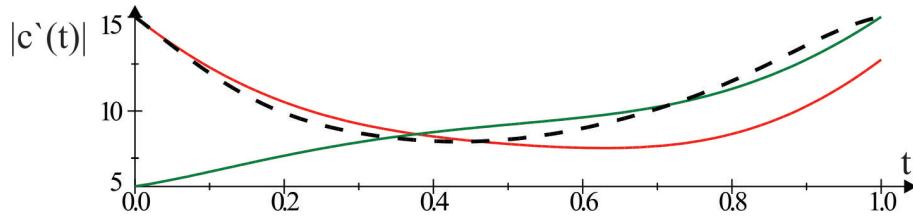


Figure 6.3: In the figure is the speed, $|c'(t)|$, of the two Bézier-curves and the resulting curve shown. The same color and style as in Figure 6.2 are used.

In Figure 6.3 is the speed of the curves from Figure 6.2 plotted. We can see that the speed of the resulting curve is the same as the speed of the first curve at start, $|f'(0)| = |g_1'(0)|$, and the speed of the resulting curve is the same as the speed of the second curve at end, $|f'(1)| = |g_2'(1)|$.

The relation between the two original functions called local functions and the resulting function called the global function is of great interest. Especially what happens at the start and at the end of the global function. This we will now investigate further.

Theorem 6.1. *A two-functions blending by B-function has the following Hermite interpolation property*

The Hermite interpolation property

In a two-functions blending using a B-function, $f(t) = g_1(t) + B(t)(g_2(t) - g_1(t))$, it follows that the global function interpolate the first local function at the start point with position and derivatives up to order S_0 , i.e.

$$f^{(j)}(0) = g_1^{(j)}(0), \quad j = 0, 1, \dots, S_0, \quad (6.7)$$

and the global function interpolate the last local function at the end point with position and derivatives up to order S_1 , i.e.

$$f^{(j)}(1) = g_2^{(j)}(1), \quad j = 0, 1, \dots, S_1, \quad (6.8)$$

where S_0 and S_1 are the (Hermite) orders at start and end of the current B-function.

Proof. We first investigate the situation at start, $t = 0$. Recall from (6.1) that $B^j(0) = 0$ for $j = 0, 1, \dots, S$. From (6.4) we see that $f(0) = g_1(0)$, and from formula (6.6) we see that $f^{(j)}(0) = g_1^{(j)}(0)$ for $j = 1, \dots, S$.

At the end point, $t = 1$ we can see that we get a similar result because of the symmetry. Remember from definition 6.1 that $B(1) = 1$. From (6.3) it follows that $f(1) = g_2(1)$. If we rewrite (6.6) by taking out the first term we get

$$f^{(j)}(t) = g_1^{(j)}(t) + B(t)(g_2^{(j)}(t) - g_1^{(j)}(t)) + \sum_{i=0}^j \binom{j}{i} B^{(i)}(t) h^{(j-i)}(t).$$

We can now see that $f^{(j)}(1) = g_2^{(j)}(1)$ for $j = 1, \dots, S_1$. □

The Hermite interpolation property is also influenced by the local function. This can be summarized in the following theorem.

Theorem 6.2. *The Hermite interpolation property of a two-functions blending is influenced by*

The expanded Hermite interpolation property

In a two-function blending, if the two local functions is equal at start, i.e. $g_1(0) = g_2(0)$, then the order of the Hermite interpolation increase by 1 at start, i.e.

$$f^{(j)}(0) = g_1^{(j)}(0), \quad j = 0, 1, \dots, S_0 + 1, \quad (6.9)$$

and if the two local functions is equal at end, i.e. $g_1(1) = g_2(1)$, then the order of the Hermite interpolation increase by 1 at end, i.e.

$$f^{(j)}(1) = g_2^{(j)}(1), \quad j = 0, 1, \dots, S_1 + 1, \quad (6.10)$$

where S_0 and S_1 are the (Hermite) orders at start and end of the current B-function.

In general, if the two local functions have subsequent derivatives equal at start, i.e. $g_1^{(j)}(0) = g_2^{(j)}(0)$, $j = 0, 1, \dots, d_0$, then the order of the Hermite interpolation increase by $d_0 + 1$ at start, i.e.

$$f^{(j)}(0) = g_1^{(j)}(0), \quad j = 0, 1, \dots, S_0 + d_0 + 1, \quad (6.11)$$

and if the two local functions have subsequent derivatives equal at end, i.e. $g_1^{(j)}(1) = g_2^{(j)}(1)$, $j = 0, 1, \dots, d_1$, then the order of the Hermite interpolation increase by $d_1 + 1$ at end, i.e.

$$f^{(j)}(1) = g_2^{(j)}(1), \quad j = 0, 1, \dots, S_1 + d_1 + 1, \quad (6.12)$$

where S_0 and S_1 are the (Hermite) orders at start and end of the current B-function.

Proof. If we rewrite (6.6) by taking out the last term of the sum in the expression we get

$$f^{(j)}(t) = g_1^{(j)}(t) + \sum_{i=0}^{j-1} \binom{j}{i} B^{(i)}(t) h^{(j-i)}(t) + B^{(j)}(t) (g_2(t) - g_1(t)).$$

We can now see that, if $g_1(0) = g_2(0)$ then the last term is zero and it follows that if the order of the B-function is S then the order of the Hermite interpolation is $S + 1$.

The same argument is valid also for the end of the curve, which ends the proof of the first part of the theorem.

If we rewrite (6.6) by taking out the d last terms of the sum in the expression we get

$$f^{(j)}(t) = g_1^{(j)}(t) + \sum_{i=0}^{j-d} \binom{j}{i} B^{(i)}(t) h^{(j-i)}(t) + \sum_{i=j-d+1}^j \binom{j}{i} B^{(j)}(t) (g_2(t) - g_1(t)).$$

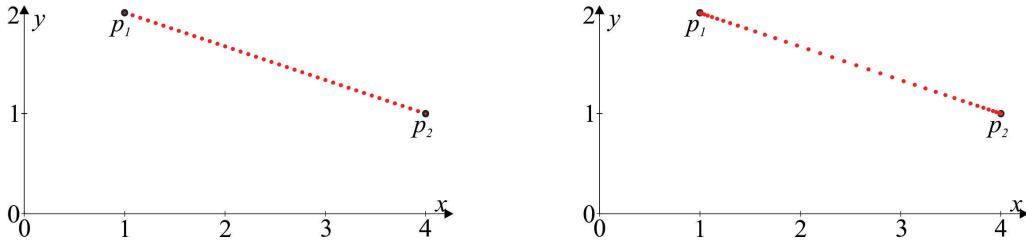


Figure 6.4: The figure on left hand side shows a curve that is a linear blending between the two points $p_1 = (1, 2)$ and $p_2 = (4, 1)$. The curve is plotted using equal spaced (in the parameter line) dots to show that the speed is constant. On right hand side is an interpolation between the same point, but now the trigonometric B-function is used. We can clearly see that the speed is zero at the start and that the speed is increasing to the middle of the curve and then decreasing to zero at the end.

We can now see that, if $g_1^{(i)}(0) = g_2^{(i)}(0)$, $i = 0, 1, \dots, d$ then the d last sum is zero and it follows that if the order of the B-function is S then the order of the Hermite interpolation is $S + d + 1$.

The same argument is valid also for the end of the curve. This ends the proof of the second part of, and thereby the whole theorem. \square

6.2.1 Examples, blending of order zero and order one

Examples with blending of two points:

The classical B-function is the linear blending function, $B(t) = t$, most known from linear interpolation. On left hand side in Figure 6.1 is this linear, order zero, B-function plotted together with its first derivative, and to the left in Figure 6.4 is a curve plotted that is a linear blending of two constant functions, i.e. two points p_1 and p_2 . From (6.4) we get

$$\begin{aligned} c(t) &= p_1 + t(p_2 - p_1), \\ c'(t) &= p_2 - p_1. \end{aligned}$$

As we can see is the derivative $c'(t)$ a constant and the speed of the curve is thus the same all over. This can be clearly seen on left hand side in Figure 6.4 because the plot is done with dots equally spaced in the parameter space, and we can see that the dots are equally spaced in the 2D-plane to.

On right hand side in Figure 6.4 is a linear curve made by using a B-function of order one (the trigonometric B-function),

$$\begin{aligned} c(t) &= p_1 + \left(\sin^2 \frac{\pi}{2} t\right)(p_2 - p_1) \\ c'(t) &= \left(\pi \cos \frac{\pi}{2} t \sin \frac{\pi}{2} t\right)(p_2 - p_1) \end{aligned}$$

From the expression, we can clearly see that the speed is not constant. We can see that the speed is zero at start, because $\sin 0 = 0$, and zero at end because $\cos \frac{\pi}{2} = 0$. This can also be clearly seen on right hand side in Figure 6.4 because the plot is done with dots equally

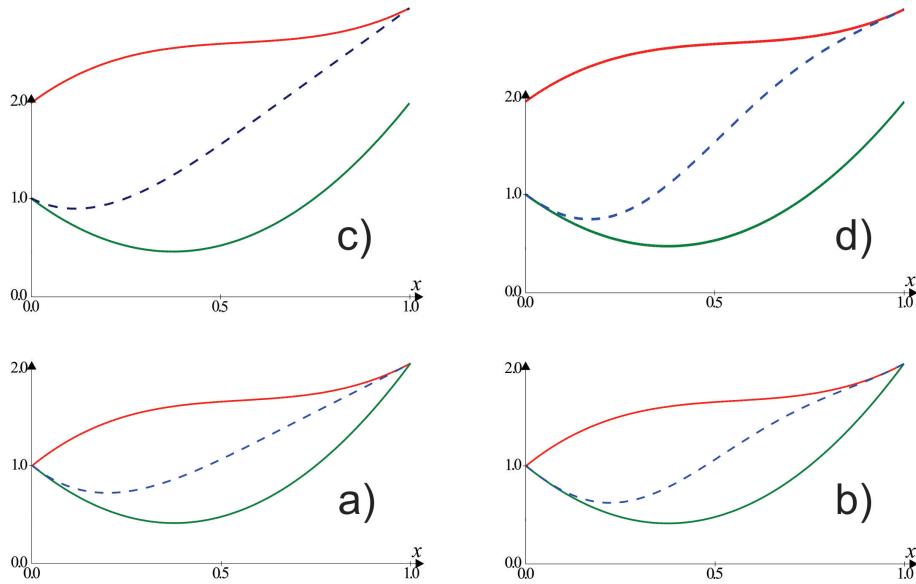


Figure 6.5: All green curves are g_1 , expression (6.13). The red curves in the two lower plots are g_2 , expression (6.14). The red curves in the two upper plots are \tilde{g}_2 , expression 6.15. The two plots on left hand side are using a linear B-function of order zero, while the two plots on right hand side are using a trigonometric B-function of order one.

spaced in the parameter space, and as we can see is the increase the density of the dots at both ends.

Examples with blending of two functions:

In the next example we blend two functions. The first function is

$$g_1(x) = 4x^2 - 3x + 1 \quad (6.13)$$

that is shown as a green curve in Figure 6.5. The second function is

$$g_2(x) = 3x^3 - 5x^2 + 3x + 1. \quad (6.14)$$

that is shown as a red curve in Figure 6.5, plot a) and plot b). In plot c) and d) is g_2 translated with 1, i.e.

$$\tilde{g}_2(x) = 3x^3 - 5x^2 + 3x + 2. \quad (6.15)$$

In the examples we use two different B-function, the zero order linear B-function and the first order trigonometric B-function. Using the zero order linear B-function we get

$$\begin{aligned} f(x) &= g_1(x) + x(g_2(x) - g_1(x)), \\ f'(x) &= g'_1(x) + x(g'_2(x) - g'_1(x)) + g_2(x) - g_1(x), \\ f''(x) &= g''_1(x) + x(g''_2(x) - g''_1(x)) + 2(g'_2(x) - g'_1(x)). \end{aligned}$$

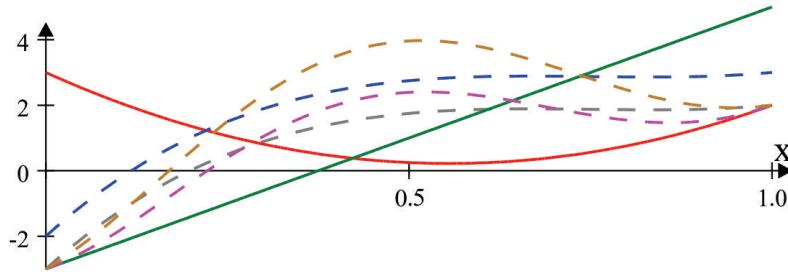


Figure 6.6: The plot shows the derivatives of the four resulting curves in Figure 6.5 together with the derivatives of the original functions.

Using the first order trigonometric B-function we get

$$\begin{aligned} h(x) &= g_1(x) + \left(\sin^2 \frac{\pi}{2}x\right)(g_2(x) - g_1(x)) \\ &= \frac{1}{2}(g_1 + g_2 + (g_1 - g_2)\cos \pi x), \\ h'(x) &= \frac{1}{2}((g'_1 + g'_2) + (g'_1 - g'_2)\cos \pi x - \pi(g_1 - g_2)\sin \pi x), \\ h''(x) &= \frac{1}{2}((g''_1 + g''_2) + ((g''_1 - g''_2) - \pi^2(g_1 - g_2))\cos \pi x - 2\pi(g'_1 - g'_2)\sin \pi x). \end{aligned}$$

Since we use two blending functions and two second functions g_2 and \tilde{g}_2 , we get 4 resulting curves $f(x)$, $\tilde{f}(x)$, $h(x)$ and $\tilde{h}(x)$. These four curves are shown in Figure 6.5 as dashed blue curves. Observe that it look like that the curve in plot c) only interpolates g_1 at start, and g_2 at end (order 0). The curve in plot a) and d) also interpolates the first derivatives (order 1), and the curve in plot b) interpolates also the second derivatives (order 2).

To verify the observation we compute the first and second derivatives. Notice that all derivatives of \tilde{g}_2 are equal to all derivatives of g_2 . Computing the first and second derivatives at start of g_1 and end of g_2 , and at start and end of the resulting curves we get:

green/red	$g'_1(0) = -3$	$g'_2(1) = 2$	$g''_1(0) = 8$	$g''_2(1) = 8$	order
a) gray	$f'(0) = -3$	$f'(1) = 2$	$f''(0) = 20$	$f''(1) = 2$	1
b) magenta	$h'(0) = -3$	$h'(1) = 2$	$h''(0) = 8$	$h''(1) = 8$	2
c) blue	$\tilde{f}'(0) = -2$	$\tilde{f}'(1) = 3$	$\tilde{f}''(0) = 20$	$\tilde{f}''(1) = 2$	0
d) brown	$\tilde{h}'(0) = -3$	$\tilde{h}'(1) = 2$	$\tilde{h}''(0) = 8 + \frac{\pi^2}{2}$	$\tilde{h}''(1) = 8 - \frac{\pi^2}{2}$	1

The first column in the table is referring to Figure 6.5. The second row to the color of the different plot of the first derivatives in Figure 6.6. The red numbers indicate match to g_1 and g_2 . Order is order of interpolation. In Figure 6.6 is g'_1 solid green, $g'_2 = \tilde{g}'_2$ is solid red. The dashed grey is the first derivative of the resulting curve in a), the dashed violet is the first derivative in b), the dashed blue is the first derivative in c), and the dashed brown is the first derivative in d).

6.2.2 Examples, connecting two curves by using a B-function

We will look at an example where two curves that are not connected is being connected by using a B-function. We will find several articles on this topic and the topic of the previous

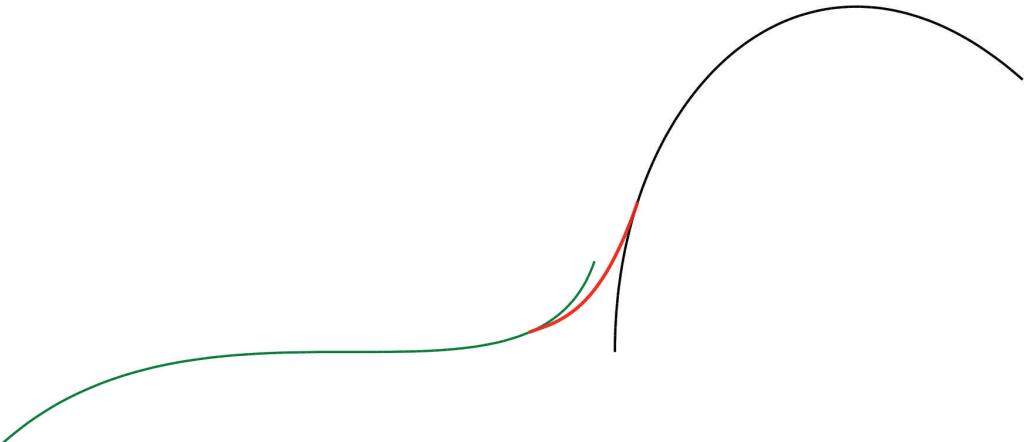


Figure 6.7: In the figure is there two curves connected by using a B-function. The green curve $c_1(t)$ is connected to the black curve $c_2(t)$ with the red curve $c_3(t)$ that connects the two original curves in a smooth way.

subsection, such as [152], [104] and [76]. We start with two 3th degree Bézier-curves $\in \mathbb{R}^2$,

$$C_1(t) = (1-t)^3 \begin{pmatrix} 1 \\ 1 \end{pmatrix} + 3t(1-t)^2 \begin{pmatrix} 2 \\ 2 \end{pmatrix} + 3t^2(1-t) \begin{pmatrix} 3.6 \\ 1 \end{pmatrix} + t^3 \begin{pmatrix} 3.9 \\ 2 \end{pmatrix},$$

and

$$C_2(t) = (1-t)^3 \begin{pmatrix} 4 \\ 1.5 \end{pmatrix} + 3t(1-t)^2 \begin{pmatrix} 4 \\ 3 \end{pmatrix} + 3t^2(1-t) \begin{pmatrix} 5 \\ 4 \end{pmatrix} + t^3 \begin{pmatrix} 6 \\ 3 \end{pmatrix}.$$

In Figure 6.7 are these two curves shown, $c_1(t)$ is shown in green on left hand side, and $c_2(t)$ is shown in black on right hand side in the figure.

We decide to use 20% of the original curves in the joining-curve and get the following formula for the new piece of curve,

$$C_3(t) = c_1(t) + B(5t - 4)(c_2(t - 0.8) - c_1(t)), \quad 0.8 \leq t < 1.$$

where $B(t) = 3t^2 - 2t^3$. This new curve segment $c_3(t)$ is shown as red in Figure 6.7.

The total curve is now,

$$f(t) = \begin{cases} c_1(t), & \text{if } 0 \leq t < 0.8, \\ c_3(t), & \text{if } 0.8 \leq t < 1, \\ c_2(t - 0.8), & \text{if } 1 \leq t \leq 1.8. \end{cases} \quad (6.16)$$

The domain of $f(t)$ defined in (6.16) is $[0, 1.8]$. The curve is C_1 -smooth because we used a B-function of order 1, see Theorem 6.1. The curve can be seen in Figure 6.7 as a subsequent green, red and black curve.

6.3 Beta-functions, the group of polynomial B-functions

The Euler beta function, or sometimes called the Euler integral of the first kind, is a special function defined by

$$\mathcal{B}(a, b) = \int_0^1 x^a (1-x)^b dx \quad (6.17)$$

for $a, b \geq 0$, see [1]. The Euler beta function is symmetric, i.e. $\mathcal{B}(a, b) = \mathcal{B}(b, a)$. If a and b are positive integers the expression is:

$$\mathcal{B}(a, b) = \frac{a! b!}{(a+b+1)!}. \quad (6.18)$$

The Euler beta function was first studied by Euler and Legendre and was given its name by Jacques Binet.

The incomplete beta function is a generalization of the Euler beta function that replaces the definite integral of the Euler beta function with an indefinite integral, i.e.

$$\mathcal{B}(t; a, b) = \int_0^t x^a (1-x)^b dx. \quad (6.19)$$

The regularized incomplete beta function is defined in terms of the incomplete beta function and the complete beta function,

$$I_t(a, b) = \frac{\mathcal{B}(t; a, b)}{\mathcal{B}(a, b)}. \quad (6.20)$$

Working out the integral for integer values of a and b , one finds (computet in [116])

$$I_t(a, b) = \sum_{j=a+1}^{a+b+1} \frac{(a+b+1)!}{j! (a+b+1-j)!} t^j (1-t)^{a+b+1-j}.$$

Lemma 6.1. *The regularized incomplete beta function, $I_t(a, b)$, has the following properties:*

- | | |
|---|--|
| I Zero at $t = 0$
II One at $t = 1$
III Hermite order
IV Antisymmetric
V Recursive | $I_0(a, b) = 0,$
$I_1(a, b) = 1,$
$\frac{d^j}{dt^j} I_0(a, b) = 0, \quad j = 1, 2, \dots, a,$
$\frac{d^j}{dt^j} I_1(a, b) = 0, \quad j = 1, 2, \dots, b,$
$I_t(a, b) = 1 - I_{1-t}(b, a),$
$I_t(a, b) = t I_t(a-1, b) + (1-t) I_t(a, b-1).$
$I_t(a, b) = I_t(a-1, b) - \frac{t^a (1-t)^{b+1}}{a \mathcal{B}(a-1, b)},$
$I_t(a, b) = I_t(a, b-1) + \frac{t^{a+1} (1-t)^b}{b \mathcal{B}(a, b-1)}.$ |
|---|--|

Proof. **I** follows directly from (6.19), **II** follows from (6.20) because $\mathcal{B}(1; a, b) = \mathcal{B}(a, b)$. To prove **III** we start differentiating (6.19), $\frac{d}{dt} \mathcal{B}(t; b, a) = t^a(1-t)^b$. It follows that the subsequent derivatives are a sum where each term contains a factor t^j , $j > 0$ for all derivatives of order up to a , and a factor $(1-t)^j$, $j > 0$ for all derivatives of order up to b . It follows that if $t = 0$ then all derivatives up to order a is zero, and that if $t = 1$ then all derivatives up to order b is zero, which completes the proof of **III**.

To prove **IV** we start computing

$$\begin{aligned}\mathcal{B}(1-t; b, a) &= \int_0^{1-t} x^b (1-x)^a dx \\ &= \int_t^1 (1-x)^b (1-(1-x))^a dx \\ &= \int_t^1 x^a (1-x)^b dx.\end{aligned}$$

It follows that $\mathcal{B}(t; a, b) + \mathcal{B}(1-t; b, a) = \mathcal{B}(a, b)$, which together with (6.20) completes the proof of **IV**.

To prove **V**, the recursion, we start differentiating the kernel of (6.19),

$$\frac{d}{dx} x^a (1-x)^b = a x^{a-1} (1-x)^b - b x^a (1-x)^{b-1}.$$

Then we go back by antiderivative (integration) and divide both sides with ab

$$\frac{1}{ab} x^a (1-x)^b = \frac{a}{ab} \int_0^x t^{a-1} (1-t)^b dt - \frac{b}{ab} \int_0^x t^a (1-t)^{b-1} dt. \quad (6.21)$$

We then add $(a+b)x^a(1-x)^b$ on both left and right side and reorganize the right side

$$\begin{aligned}\frac{a+b+1}{ab} x^a (1-x)^b &= \frac{a}{ab} \left(x^a (1-x)^b + \int_0^x t^{a-1} (1-t)^b dt \right) + \\ &\quad \frac{b}{ab} \left(x^a (1-x)^b - \int_0^x t^a (1-t)^{b-1} dt \right).\end{aligned}$$

Integrating once more,

$$\frac{a+b+1}{ab} \int_0^x t^a (1-t)^b dt = \frac{1}{b} x \int_0^x t^{a-1} (1-t)^b dt + \frac{1}{a} (1-x) \int_0^x t^a (1-t)^{b-1} dt.$$

then we multiply both sides with $\frac{(a+b)!}{(a-1)!(b-1)!}$,

$$\frac{(a+b+1)!}{a!b!} \int_0^x t^a (1-t)^b dt = \frac{(a+b)!}{(a-1)!b!} \int_0^x t^{a-1} (1-t)^b dt + \frac{(a+b)!}{a!(b-1)!} (1-x) \int_0^x t^a (1-t)^{b-1} dt.$$

which together with (6.18) and (6.20) completes the proof of the first part of the recursion. The proof of the two last parts is just a reorganizing of (6.21), which ends the proof. \square

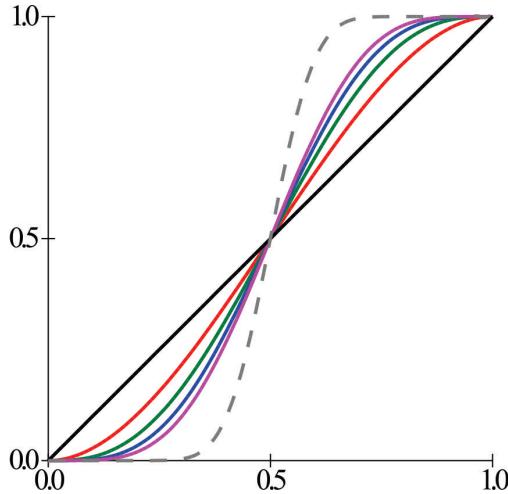


Figure 6.8: The figure shows six symmetric Beta-functions; order 0 (black), order 1 (red), order 2 (green), order 3 (blue), order 4 (violet) and order 20 (triplet gray).

Theorem 6.3. *The regularized incomplete beta function is a B-function when the shape-values a, b are integers. For short we name it Beta-function (see [40]), and it is expressed by*

$$B_{a,b}(t) = I_t(a, b), \quad \text{for } t \in [0, 1],$$

and for $a, b \geq 0$ ($a, b \in \mathbb{N} \cup 0$), and where $a = S_0$ and $b = S_1$ are the (Hermite) orders of the Beta-function.

Proof. The theorem follows from the definitions 6.1 and 6.2 and lemma 6.1. □

Note that the Beta-function of lowest order is linear and the series of the group converges towards step-functions. Except for the linear function they have a kind of S-shape.

We will first look at the symmetric series of Beta-functions.

Symmetric Beta-functions

A symmetric Beta-function $B_s(t)$ is symmetric because of property **III** in lemma 6.1. The order of the Beta-function is S . The first five Beta-functions (of order 0,1,2,3,4) are:

$$B_0(t) = t$$

$$B_1(t) = -2t^3 + 3t^2$$

$$B_2(t) = 6t^5 - 15t^4 + 10t^3$$

$$B_3(t) = -20t^7 + 70t^6 - 84t^5 + 35t^4$$

$$B_4(t) = 70t^9 - 315t^8 + 540t^7 - 420t^6 + 126t^5$$

In Figure 6.8 is the five first symmetric Beta-functions plotted together with the order 20 symmetric Beta-function. Here we see that the series appears to converge towards a step function in $t = 0.5$.

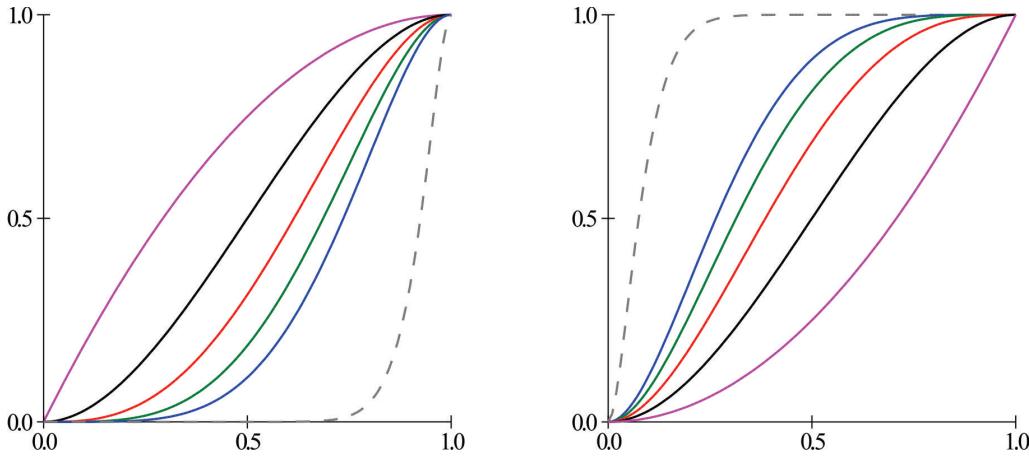


Figure 6.9: Examples of asymmetric Beta-functions. On left hand side in the figure is the right side order constant 1, on right hand side in the figure is the left side order constant 1. The plots are violet: left - $B_{0,1}(t)$ and right - $B_{1,0}(t)$, black: left and right- $B_{1,1}(t)$, red: left - $B_{2,1}(t)$ and right - $B_{1,2}(t)$, green: left - $B_{3,1}(t)$ and right - $B_{1,3}(t)$, blue: left - $B_{4,1}(t)$ and right - $B_{1,4}(t)$ and triplet gray: left - $B_{20,1}(t)$ and right - $B_{1,20}(t)$.

Another remark is that the symmetric Beta-functions is the same function we find as basis function for Hermite-curves of degree 3,5,7,..., see the second basis function in (4.16) and (4.23).

Now we look at asymmetric Beta-functions

Asymmetric Beta-functions

A series of asymmetric Beta-functions appears when we keep one parameter of order constant and let the other vary from 0, 1, 2, 3,.. towards ∞ .

An example of an series of asymmetric Beta-functions is when we keep the order on right hand side constant $S_1 = 1$, and let the order on left hand side vary from $S_0 = 0, 1, 2, 3, 4, \dots$. The formula for the first five functions are

$$\begin{aligned} B_{0,1}(t) &= t^2(2-t), \\ B_{1,1}(t) &= t^2(3-2t), \\ B_{2,1}(t) &= t^3(4-3t), \\ B_{3,1}(t) &= t^4(5-4t), \\ B_{4,1}(t) &= t^5(6-5t). \end{aligned}$$

On left hand side in Figure 6.9 is there some examples from the series of Beta-functions described above. It is Beta-functions of orders $B_{0,1}, B_{1,1}, B_{2,1}, B_{3,1}, B_{4,1}$ and $B_{20,1}$. Here we see that the series appears to converge towards a step function in $t = 1$.

On right hand side in Figure 6.9 is the mirror series plotted. The Beta-functions where the left side order is keep constant 1 and the right side order vary from $S_0 = 0, 1, 2, 3, 4$ and 20. Here we see that the series appears to converge towards a step function in $t = 0$. Later, in section 6.8 we will investigate what we call order-symmetry of B-functions.

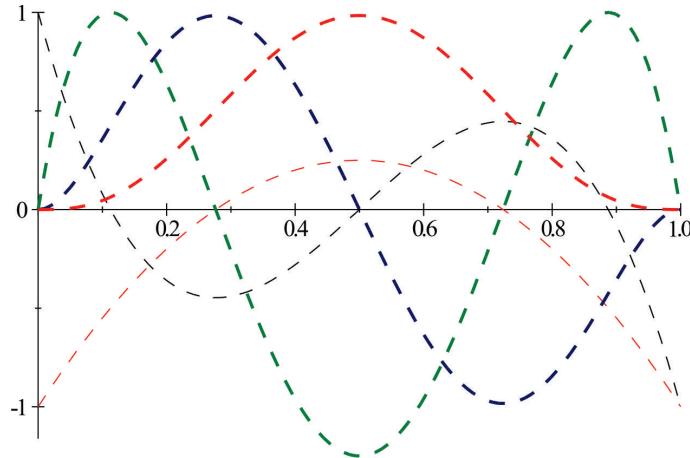


Figure 6.10: The figure shows the five first subsequent derivatives of a 3th order symmetric Beta-functions; 1st derivative (bold red), 2nd derivative (bold blue), 3th derivative (bold green), 4th derivative (thin black) and 5th derivative (thin red). The functions are scaled to approximately fit inside $[-1, 1]$.

6.3.1 Beta-functions, differentiation

The derivatives of a general Beta-function are quite complex especially on the high order derivatives. The three first derivatives for the general Beta-function are:

$$\begin{aligned} B'_{a,b}(t) &= \frac{(a+b+1)!}{a!b!} t^a (1-t)^b, \\ B''_{a,b}(t) &= \frac{(a+b+1)!}{a!b!} (a(1-t)-bt) t^{a-1} (1-t)^{b-1}, \\ B'''_{a,b}(t) &= \frac{(a+b+1)!}{a!b!} (a(a-1)-(a+b-1)(2a-(a+b)t)t) t^{a-2} (1-t)^{b-2}. \end{aligned}$$

For a concrete Beta-function is the derivatives much more easy. We use the symmetric 3th order Beta-functions as example. The formula and the five first derivatives are

$$\begin{aligned} B_3(t) &= -20t^7 + 70t^6 - 84t^5 + 35t^4, \\ B'_3(t) &= 140t^3(1-t)^3, \\ B''_3(t) &= 420t^2(1-2t)(1-t)^2, \\ B'''_3(t) &= 840t(1-t)(5t^2-5t+1), \\ B^{(4)}_3(t) &= 840(1-2t)(10t^2-10t+1), \\ B^{(5)}_3(t) &= 10080(-5t^2+5t-1). \end{aligned}$$

In Figure 6.10 is the five derivatives of the symmetric 3th order Beta-function shown. To show them in the same figure we have to scale them to approximately fit inside $[-1, 1]$. The bold-red is the first derivative (divided by 2.2), the bold-blue is the second derivative (divided by 7.5), the bold-green is the third derivative (divided by 42). Note that the value of all these three are zero at start and end. This is because the order of the Beta-function is three. The fourth derivatives are plotted in thin-black (divided by 840) and the fifth derivatives are thin-red (divided by 10080) in the figure.

6.4 RB-functions, the group of rational B-functions

In 2001, parametric G^n blending was discussed by E. Hartmann in [76]. He introduced a rational blending function

$$b_{n,\mu}(t) = \frac{(1-\mu)t^{n+1}}{(1-\mu)t^{n+1} + \mu(1-t)^{n+1}}, \quad t \in [0, 1], \quad 0 < \mu < 1, \quad n \geq 0.$$

If $\mu = 0.5$ the curves are point symmetric about the point $(0.5, 0.5)$. For other μ the curves are asymmetric. Therefor he called μ the balance of the blending function $b_{n,\mu}$. The number n determine the number of subsequent derivatives to be zero at start and end.

To use a balance variable μ can be useful and we will look into it later. However, if we want to limit the number of shape parameters to only the order parameters we can slightly modify the formula to get a more convenient B-function.

Theorem 6.4. *A Rational-function is a B-function. For short we name it an RB-function. It is expressed by*

$$B_{a,b}(t) = \frac{t^{a+1}}{t^{a+1} + (1-t)^{b+1}}, \quad \text{for } t \in [0, 1], \quad (6.22)$$

and for $a, b \geq 0$ ($a, b \in \mathbb{N} \cup 0$), where $a = S_0$ is the left side (Hermite) order and where $b = S_1$ is the right side (Hermite) order. If $a = b$ then the RB-function is symmetric according to **D5** in definition 6.1.

Proof. We follow the definitions 6.1. First it is clearly a permutation function (**D1**). If we compute (6.23) we see that $B_{a,b}(0) = 0$ (**D2**) and $B_{a,b}(1) = 1$ (**D3**). The first derivativ of (6.23) is

$$B'_{a,b}(t) = (a(1-t) + bt + 1) \frac{t^a(1-t)^b}{(t^{a+1} + (1-t)^{b+1})^2}, \quad (6.23)$$

It is clear that $B'_{a,b}(t) \geq 0$, because t and $1-t \geq 0$ for $t \in [0, 1]$, and it follows that all terms and factors are greater or equal zero. Thus the RB-function is monotone (**D4**).

Reformulating to $B_{a,b}(t) = \frac{\alpha(t)}{\beta(t)}$. The derivative $B_{a,b}^{(j)}(t)$ is a fraction where the numerator is a sum where each term has factors of $\alpha^{(r)}(t)$, $r \leq j$ and $\beta^{(s)}(t)$, $s \leq j$ where j is the order of the derivative, and the denominator is $\beta(t)^{2j}$. Note that $\beta > 0$ when $t \in [0, 1]$.

- a) If $j \leq a$ in $B_{a,b}^{(j)}(t)$ then every term in the numerator has a factor t^{a+1-r} , $r \leq j$. It follows that $B_{a,b}^{(j)}(0) = 0$ (left side Hermite order).
- b) Because $\lim_{t \rightarrow 1} (1-t) = 0$ it follows that if $j \leq b$ then $\lim_{t \rightarrow 1} \beta^{(j)}(t) = \alpha^{(j)}(t)$. Thus if $j \leq b$ then $B_{a,b}^{(j)}(1) = 0$ (right side Hermite order).

Finally, the RB-function $B_s(t)$ is symmetric (**D5**) since

$$B_a(t) + B_a(1-t) = \frac{t^{a+1}}{t^{a+1} + (1-t)^{a+1}} + \frac{(1-t)^{a+1}}{(1-t)^{a+1} + t^{a+1}} = 1,$$

which ends the proof. □

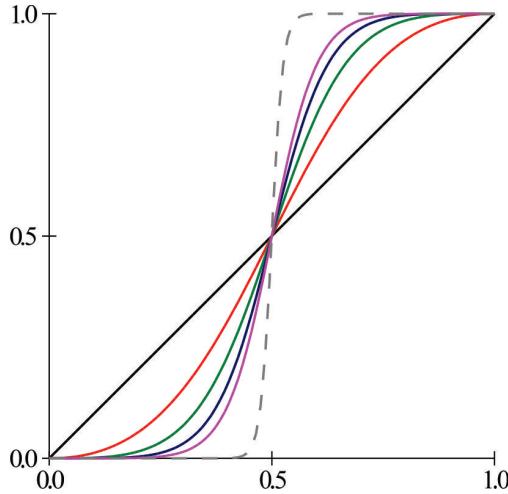


Figure 6.11: The figure shows six symmetric RB-functions; order 0 (black), order 1 (red), order 2 (green), order 3 (blue), order 4 (violet) and order 20 (triplet gray).

Note that the RB-function of lowest order is linear,

$$B_{0,0}(t) = \frac{t}{t + (1-t)} = t,$$

and that the series of the group seems to converge towards step-functions. Except for the linear function they have a kind of S-shape.

We first look at the symmetric series of RB-functions.

Symmetric RB-functions

In Theorem 6.4 is shown that a symmetric RB-function $B_s(t)$ is symmetric, and the order of $B_s(t)$ is shown to be s in Theorem 6.4. The general formula for the series of symmetric RB-functions is:

$$B_s(t) = \frac{t^{s+1}}{t^{s+1} + (1-t)^{s+1}}$$

The four first symmetric RB-function in the series (of order 0,1,2,3) are:

$$\begin{aligned} B_0(t) &= \frac{t^1}{t^1 + (1-t)^1} = t \\ B_1(t) &= \frac{t^2}{t^2 + (1-t)^2} = \frac{t^2}{2t^2 - 2t + 1} \\ B_2(t) &= \frac{t^3}{t^3 + (1-t)^3} = \frac{t^3}{3t^2 - 3t + 1} \\ B_3(t) &= \frac{t^4}{t^4 + (1-t)^4} = \frac{t^4}{2t^4 - 4t^3 + 6t^2 - 4t + 1} \end{aligned}$$

In Figure 6.11 are the five first symmetric RB-functions plotted together with the order 20 symmetric RB-function. Here we see that the series appears to converge towards a step-function in $t = 0.5$. Compared with the symmetric Beta-functions we see in Figure 6.8, we can see in Figure 6.11 that RB-functions seems to be slightly more steep.

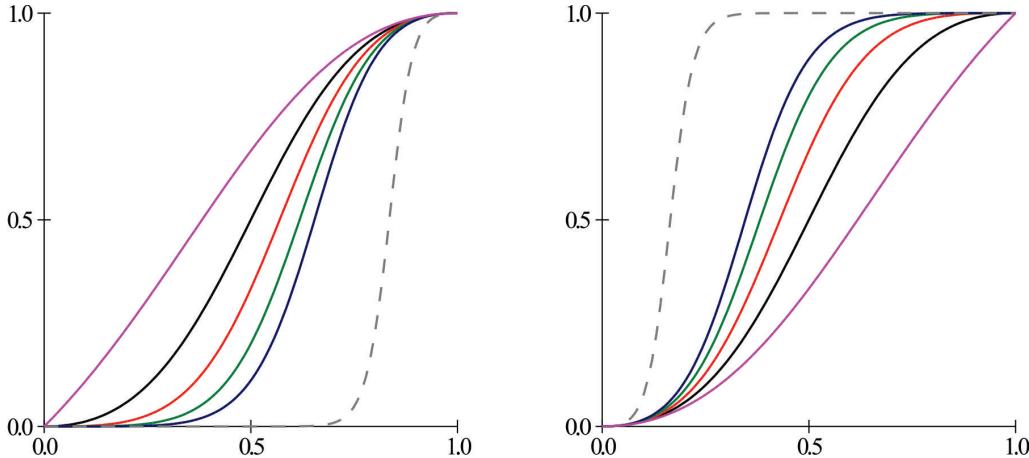


Figure 6.12: Examples of asymmetric RB-functions. On left hand side in the figure is the right side order a constant 1, on right hand side in the figure is the left side order a constant 1. The plots are violet: left - $B_{0,1}(t)$ and right - $B_{1,0}(t)$, black: left and right - $B_{1,1}(t)$, red: left - $B_{2,1}(t)$ and right - $B_{1,2}(t)$, green: left - $B_{3,1}(t)$ and right - $B_{1,3}(t)$, blue: left - $B_{4,1}(t)$ and right - $B_{1,4}(t)$ and triplet gray: left - $B_{20,1}(t)$ and right - $B_{1,20}(t)$.

Now we look at asymmetric RB-functions

Asymmetric RB-functions

A series of asymmetric RB-functions appears when we keep one parameter of order constant and let the other vary from 0, 1, 2, 3,.. towards ∞ .

An example of an series of asymmetric RB-functions is when we keep the order on right hand side constant $S_1 = 1$, and let the order on left hand side vary from $S_0 = 0, 1, 2, 3, 4, \dots$. The formula for the five first functions are

$$\begin{aligned} B_{0,1}(t) &= \frac{t^1}{t^1 + (1-t)^2} = \frac{t}{t^2 - t + 1} \\ B_{1,1}(t) &= \frac{t^2}{t^2 + (1-t)^2} = \frac{t^2}{2t^2 - 2t + 1} \\ B_{2,1}(t) &= \frac{t^3}{t^3 + (1-t)^2} = \frac{t^3}{t^3 + t^2 - 2t + 1} \\ B_{3,1}(t) &= \frac{t^4}{t^4 + (1-t)^2} = \frac{t^4}{t^4 + t^2 - 2t + 1} \\ B_{4,1}(t) &= \frac{t^5}{t^5 + (1-t)^2} = \frac{t^5}{t^5 + t^2 - 2t + 1} \end{aligned}$$

On left hand side in Figure 6.12 is there some examples from the series of RB-functions described above. It is RB-functions of orders $B_{0,1}, B_{1,1}, B_{2,1}, B_{3,1}, B_{4,1}$ and $B_{20,1}$. Here we see that the series appears to converge towards a step function in $t = 1$.

On right hand side in Figure 6.12 is the mirror series plotted. The Beta-functions where the left side order is keep constant 1 and the right side order vary from $S_0 = 0, 1, 2, 3, 4$ and 20. Here we see that the series appears to converge towards a step function in $t = 0$.

Also in the asymmetric case the RB-function seems to be steeper than the Beta-functions. Later, in section 6.8 we will investigate what we call order-symmetry of B-functions.

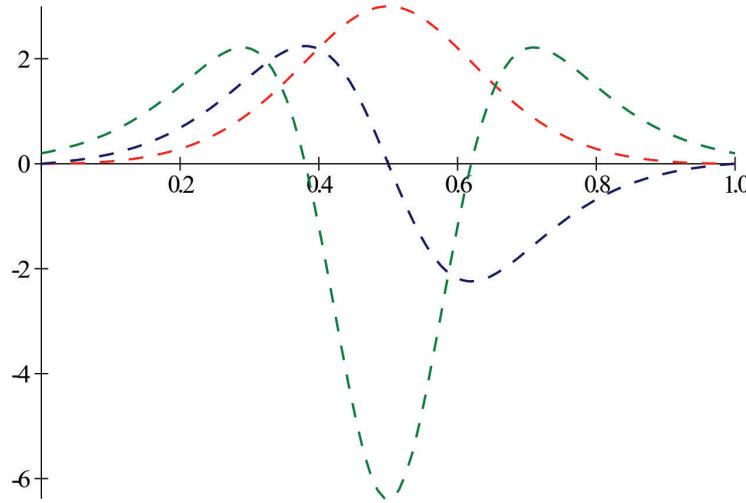


Figure 6.13: The figure shows the three first subsequent derivatives of a 2th order symmetric RB-functions; 1st derivative (bold red), 2nd derivative divided by 6 (bold blue) and 3th derivative divided by 30 (bold green).

6.4.1 RB-functions, differentiation

The derivatives of a general RB-function are quite complex especially on the high order derivatives. The two first derivatives for the general RB-function are:

$$\begin{aligned} B'_{a,b}(t) &= (a(1-t) + bt + 1) \frac{t^a(1-t)^b}{(t^{a+1} + (1-t)^{b+1})^2}, \\ B''_{a,b}(t) &= \left[[t^2(a(a+1) - b(b+1)) + a(a+1)(1-2t)] \left(t^{a+1} + (1-t)^{b+1} \right) - \right. \\ &\quad \left. 2(a(1-t) + bt + 1)t(1-t) \left((a+1)t^a - (b+1)(1-t)^b \right) \right] \frac{t^{a-1}(1-t)^{b-1}}{((1-t)^{b+1} + t^{a+1})^3}. \end{aligned}$$

For a concrete RB-function is the derivatives much easier. We use the symmetric 2nd order RB-functions as example. The formula and the three first derivatives are

$$\begin{aligned} B_2(t) &= \frac{t^3}{t^3 + (1-t)^3}, \\ B'_2(t) &= 3 \frac{t^2(1-t)^2}{(t^3 + (1-t)^3)^2}, \\ B''_2(t) &= -6 \frac{(2t-1)t(1-t)}{(t^3 + (1-t)^3)^3}, \\ B'''_2(t) &= -6 \frac{18t^2(1-t)^2 - 1}{(t^3 + (1-t)^3)^4}. \end{aligned}$$

In Figure 6.13 is the three first derivatives of the symmetric 2nd order RB-function shown. To show them in the same figure we have to scale them. The bold-red is the first derivative (not scaled), the bold-blue is the second derivative (divided by 6) and the bold-green is the third derivative (divided by 30). Note that the value of the first and second derivatives are zero at start and end. This is because the order of the Beta-function is two. The third derivatives is not zero at start and end as we can see in the figure.

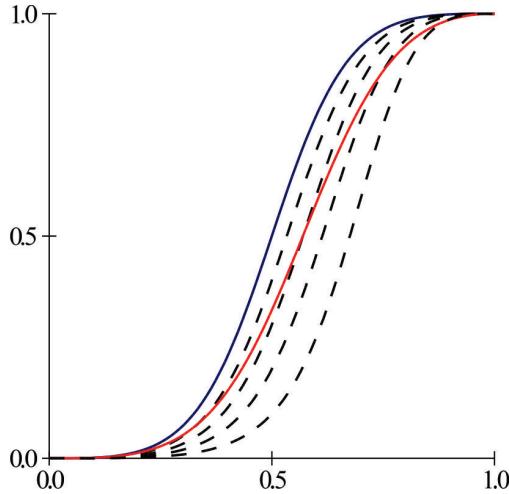


Figure 6.14: Examples of $R\mu$ -functions. The blue curve is the symmetric B-function $B_{0.5,2,2}(t) = B_2(t)$. The four black dashed curves are $R\mu$ -functions where μ is 0.6, 0.7, 0.8 and 0.9. The red solid curve is an asymmetric RB-function, $B_{2,1}(t)$.

6.4.2 RB-functions with a balance parameter

If we include the balance parameter to the rational B-function we get:

Corollary 6.1. A $R\mu$ -function is a B-function defined by

$$B_{\mu,a,b}(t) = \frac{(1-\mu)t^{a+1}}{(1-\mu)t^{a+1} + \mu(1-t)^{b+1}}, \quad \text{for } t \in [0, 1], \quad (6.24)$$

and where $a, b \in \mathbb{N}$ ($a, b > 0$), and $0 < \mu < 1$, $\mu \in \mathbb{R}$.

It follows that a $R\mu$ -function is a B-function because of Theorem 6.4.

As for RB-functions, the left side order is $a = S_0$, and the right side order is $b = S_1$. If $a = b$ and $\mu = 0.5$ then the $R\mu$ -function is symmetric according to D5 in definition 6.1.

To compare $R\mu$ -functions with RB-functions we first look at a second order symmetric RB-function $B_2(t)$. We then expand this RB-function to a $R\mu$ -functions where $a = b = 2$ and $\mu = 0.5$, i.e. $B_{0.5,2,2}(t)$. Then we change μ to see what happens.

In Figure 6.14 is the function, $B_{0.5,2,2}(t) = B_2(t)$ plotted in solid blue. We can also see four functions plotted in dashed black. These are $B_{0.6,2,2}(t)$, $B_{0.7,2,2}(t)$, $B_{0.8,2,2}(t)$ and $B_{0.9,2,2}(t)$. As we can see, they are pushed to the right according to the value of μ . Remember that all five function are of order 2 on both sides, but only the solid blue is symmetric.

To see the effect of the balance parameter μ we also shows an asymmetric RB-function $B_{2,1}(t)$ plotted in solid red in Figure 6.14. The balance parameter μ does not influence the steepness of the function, while the order affects the steepness. Later, in section 6.8, we will investigate what we call a balance-symmetry of B-functions.

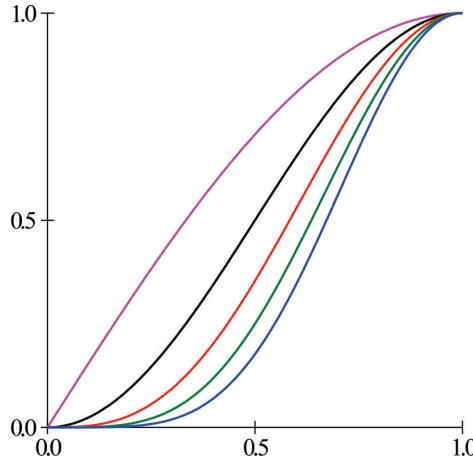


Figure 6.15: The figure shows the TB-functions: $B_0(t)$ (violet), $B_1(t)$ (black), $B_2(t)$ (red), $B_3(t)$ (green), $B_4(t)$ (blue).

6.5 TB-functions, the series of trigonometric B-functions

In 1996 H. Wenz published an interpolation method based on blending circular arcs, [148]. Given a sequence of points $\{p_k\}_{k=0}^n$. In each internal point p_k , $k = 1, 2, \dots, n-1$ there is a circular arc $\vartheta_k(u)$, $u \in [0, 1]$ starting in p_{k-1} , interpolating p_k and ending in p_{k+1} but parameterized from 0 to 1 in both segments, $[p_{k-1}, p_k]$ and $[p_k, p_{k+1}]$. To make a smooth curve segment between two point p_k and p_{k+1} , $k = 0, 1, \dots, n-1$, Wench used a simple blending scheme,

$$c_k(t) = \begin{pmatrix} 1-t & t \end{pmatrix} \begin{pmatrix} \vartheta_k(t) \\ \vartheta_{k+1}(t) \end{pmatrix}.$$

Recall that despite that the B-function $B(t) = t$ is of order zero this is a C^1 -smooth curve because every arcs, local curves, interpolates three points and then theorem 6.2 tells us that this raise the order by one.

Wenz was followed by M. Szilvási-Nagy and T. Vendel, [138]. They improved the blending scheme, by replacing the linear interpolation function with a trigonometrically weighted blending function, and thus giving a C^2 -smooth curve,

$$c_k(t) = \begin{pmatrix} \cos^2 \frac{\pi}{2}t & \sin^2 \frac{\pi}{2}t \end{pmatrix} \begin{pmatrix} \vartheta_k(t) \\ \vartheta_{k+1}(t) \end{pmatrix}.$$

Now we can glimpse what the next group/series is.

Theorem 6.5. *The Trigonometric-function is a B-function. For short we name it a TB-function, and it is expressed by*

$$B_s(t) = \sin^{s+1} \frac{\pi}{2}t, \quad t \in [0, 1], \tag{6.25}$$

and where S is the left side order and where the right side order is 1. It follows that it is only $B_1(t) = \sin^2 \frac{\pi}{2}t$ which is symmetric according to **D5** in definition 6.1.

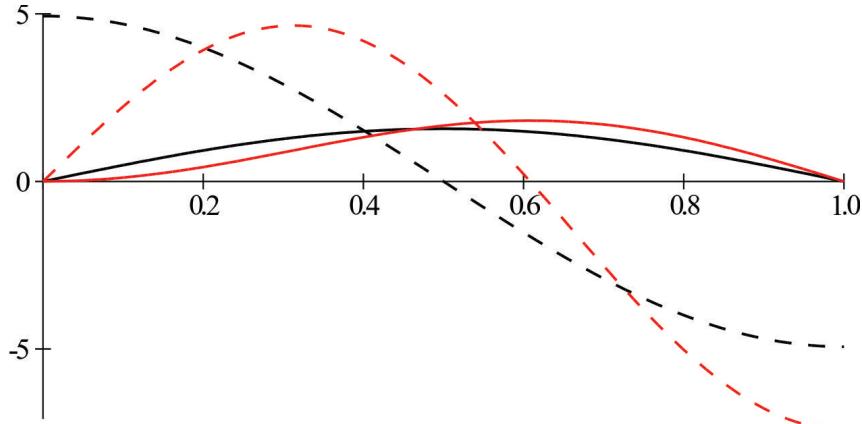


Figure 6.16: The figure shows the first derivatives (solid) and second derivatives (triplet) of $B_1(t)$ in black and $B_2(t)$ in red.

Proof. We follow the definitions 6.1. First it is clearly a permutation function (**D1**). If we compute (6.25) we see that $B_s(0) = 0$ (**D2**) and $B_s(1) = 1$ (**D3**). The first derivative of (6.25) is

$$B'_s(t) = \frac{(s+1)\pi}{2} \sin^s \frac{\pi}{2} t \cos \frac{\pi}{2} t,$$

It is clear that $B'_s(t) \geq 0$, because $\sin \frac{\pi}{2} t \geq 0$ and $\cos \frac{\pi}{2} t \geq 0$ for $t \in [0, 1]$. Thus the RB-function is monotone (**D4**).

It follows that the subsequent derivatives are sums where each term is on the form

$$k \cos^a \frac{\pi}{2} t \cos^b \frac{\pi}{2} t \quad \text{where } k > 0, a \geq 0 \text{ and } b \geq 0.$$

Left side order: It follows that for derivatives up to order S are $a > 0$ for all terms. This shows that the left side (Hermite) order is S since $\sin(0) = 0$.

Right side order: The second derivative has one term where $b = 0$. Since $\sin \frac{\pi}{2} = 1$ it follows that the right side (Hermite) order is 1.

The symmetry of $B_1(t)$ follows from

$$B_1(t) + B_1(1-t) = \sin^2 \frac{\pi}{2} t + \sin^2 \frac{\pi}{2} (1-t) = \sin^2 \frac{\pi}{2} t + \cos^2 \frac{\pi}{2} t = 1,$$

which completes the proof. \square

To illustrate the behavior of the TB-series we have in Figure 6.15 plots of the 5 first functions in the TB-function series. All functions has right side order 1, the left side order is zero for $B_0(t)$ (violet), one for $B_1(t)$ (black), two for $B_2(t)$ (red), three for $B_3(t)$ (green) and four for $B_4(t)$ (blue). In Figure 6.16 is the first derivatives, $B'_1(t)$ plotted in solid black and $B'_2(t)$ plotted in solid red. The second derivatives, $B''_1(t)$ is plotted in triplet black and $B''_2(t)$ is plotted in triplet red. The figure illustrate the symmetry of $B_1(t)$ (in black) and the asymmetry of $B_2(t)$ (in red). Note that another formulation for the first order TB-function is $B_1(t) = \frac{1}{2}(1 - \cos \pi t)$.

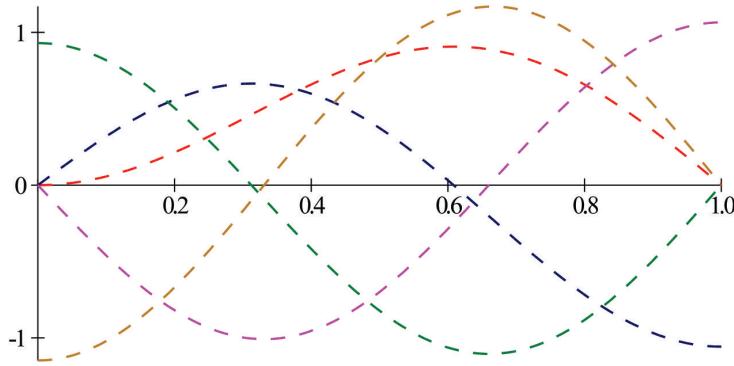


Figure 6.17: The figure shows the five first subsequent derivatives of a 2th order asymmetric TB-functions; $B'_2(t)$ divided by 2 (red), $B''_2(t)$ divided by 7 (blue), $B'''_2(t)$ divided by 25 (green), $B_2^{(4)}(t)$ divided by 120 (violet), $B_2^{(5)}(t)$ divided by 500 (brown).

6.5.1 TB-functions, differentiation

The derivatives of a general TB-function are quite complex especially on the high order derivatives. The two first derivatives for the general TB-function are:

$$\begin{aligned} B'_s(t) &= \frac{(s+1)\pi}{2} \sin^s \frac{\pi}{2} t \cos \frac{\pi}{2} t, \\ B''_s(t) &= \frac{s(s+1)\pi^2}{4} \sin^{s-1} \frac{\pi}{2} t \cos^2 \frac{\pi}{2} t - \frac{(s+1)\pi^2}{4} \sin^{s+1} \frac{\pi}{2} t, \\ B'''_s(t) &= \frac{(s-1)s(s+1)\pi^3}{8} \sin^{s-2} \frac{\pi}{2} t \cos^3 \frac{\pi}{2} t - \frac{(3s^2+4s+1)\pi^3}{8} \sin^s \frac{\pi}{2} t \cos \frac{\pi}{2} t. \end{aligned}$$

For a concrete RB-function is the derivatives much easier. We use the symmetric 2nd order RB-functions as example. The formula and the three first derivatives are

$$\begin{aligned} B_2(t) &= \sin^3 \frac{\pi}{2} t, \\ B'_2(t) &= \frac{3\pi}{2} \cos \frac{\pi}{2} t \sin^2 \frac{\pi}{2} t, \\ B''_2(t) &= \frac{3\pi^2}{16} (3 \sin \frac{3\pi}{2} t - \sin \frac{\pi}{2} t), \\ B'''_2(t) &= \frac{3\pi^3}{32} (9 \cos \frac{3\pi}{2} t - \cos \frac{\pi}{2} t), \\ B_2^{(4)}(t) &= \frac{3\pi^4}{64} (\sin \frac{\pi}{2} t - 27 \sin \frac{3\pi}{2} t), \\ B_2^{(5)}(t) &= \frac{3\pi^5}{128} (\cos \frac{\pi}{2} t - 81 \cos \frac{3\pi}{2} t). \end{aligned}$$

In Figure 6.17 is the five first derivatives of the asymmetric 2nd order TB-function shown. The plots are scaled to be shown in the same figure. Remember that the order of a TB-function is the left side order. The right side order is always 1. In Figure 6.17 we can see that only the first derivative is zero at start and end. The second derivative is zero at start only and the next derivatives are alternately zero at either the end or at the start. For TB-function where the left side order is an odd number will the derivatives $2, 3, \dots, S$ be zero only at start, but the next derivatives be alternately different from zero both at start and end and zero at both start and end. This is the property of alternation that is typical for TB-splines.

6.6 Expo-Rational B-function, a complete B-function

In 2004 Expo-Rational B-splines were present by L. Dechevsky, A. Lakså and B. Bang at the conference of Mathematical methods for Curves and Surfaces, and later published in [95],[43],[44] and [89]. The Expo-Rational basis function is related to the Gaussian bell function and the cumulative distribution function (see [64]). However, the spline will be treated in an other chapter, therefor, we will here focus on the Expo-Rational function as a B-function, and start with a simple function without extra parameters, which we call the default function.

The Expo-Rational B-function

Theorem 6.6. *The Expo-rational function is a B-function. We name it for short ERB-function. The default function is expressed by*

$$B_d(t) = S_d \int_0^t \phi(s) ds, \quad t \in [0, 1], \quad (6.26)$$

where

$$\phi(s) = e^{-\frac{(s-\frac{1}{2})^2}{s(1-s)}}, \quad (6.27)$$

and

$$S_d = \left[\int_0^1 \phi(t) dt \right]^{-1} \approx 1.6571376797382103. \quad (6.28)$$

The (Hermite) order of the ERB-function is infinite, and the default function is symmetric according to **D5** in definition 6.1.

Proof. We follow the definitions 6.1. First it is clearly a permutation function (**D1**). If we compute (6.26) we see that $B_d(0) = 0$ (**D2**) and $B_d(1) = 1$ (**D3**). The first derivative of (6.25) is

$$B'_d(t) = S_d \phi(t).$$

It is clear that $B'_d(t) \geq 0$, because $\phi(t) \geq 0$ for all t . Thus the ERB-function is monotone (**D4**).

The derivatives of all order can be described on the following form,

$$B_d^{(j)}(t) = S_d f_j(t) \phi(t),$$

where $f_j(t)$ is a rational function and j is the order of the derivative. Because both $\lim_{t \rightarrow 0+} \phi(t)$ and $\lim_{t \rightarrow 1-} \phi(t)$ converges faster towards zero than the polynomial function in the denominator of $f_j(t)$ for any j , it follows that the (Hermite) order of the ERB-function is infinite.

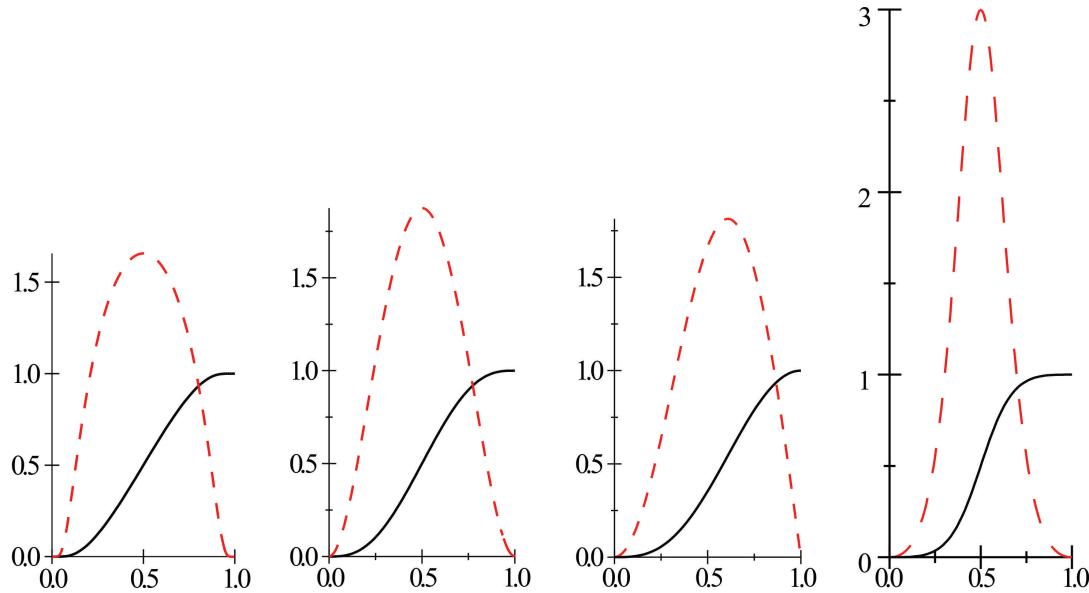


Figure 6.18: Four B-functions (solid black) and their derivatives (dashed red). On left hand side in the figure is the Expo-Rational B-function, the ERB-function. Then follows the 2nd order symmetric Beta-function, the 2nd order asymmetric TB-function, left side order 2 and right side order 1, and on right hand side in the figure is the 2nd order symmetric RB-function.

Because $(t - \frac{1}{2})^2 = (1 - t - \frac{1}{2})^2$ the symmetry of $B(t)$ follows from

$$B(t) + B(1-t) = S_d \int_0^t \phi(s) ds + S_d \int_t^1 \phi(s) ds = 1$$

which completes the proof. \square

The default ERB-function is a point symmetric B-function. In Figure 6.18 is an ERB-function (solid black) and it's 1st derivative (dashed red) plotted on left hand side in the figure. To compare with other B-functions are 2nd order functions from different groups plotted on right side of the ERB-function.

The ERB-function is a complete B-function because the order is infinite. This also shows that Taylor expansion will not work at $t = 0$ and $t = 1$ since the derivatives of all order are zero. Thus, the ERB-function is not analytic at $t = 0$ and $t = 1$.

So far in this chapter we have investigated groups of B-functions. There are, however, other groups and also single functions that are difficult to put into groups. Examples of other blending functions with the properties that fit into the B-function concept are discussed in [104] and [131].

In the following subsections we will introduce four intrinsic parameters to the ERB-function.

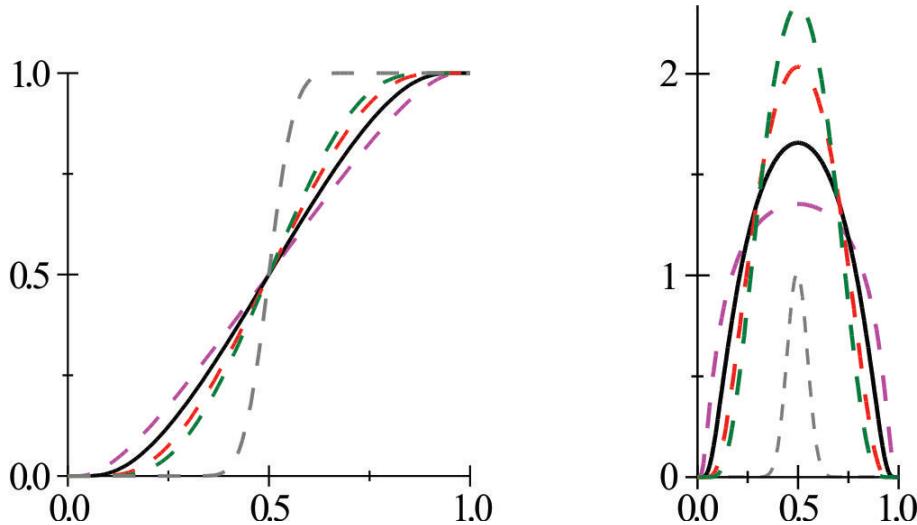


Figure 6.19: Five symmetric ERB-functions can be seen on left hand side in the figure. The dashed violet is with the slope parameter $\gamma = 0.4$. The solid black is the default ERB-function where $\gamma = 1$. The dashed red has the slope parameter $\gamma = 2$. The dashed green has the slope parameter $\gamma = 3$ and the dashed grey has the slope parameter $\gamma = 50$. On right hand side in the figure is the first derivative of the same functions plotted using the same color and style as on left hand side in the figure. The first derivative of the grey ERB-curve with the slope parameter $\gamma = 50$ is scaled by $\frac{1}{8}$.

6.6.1 The slope parameter γ

The first intrinsic parameter is the slope parameter γ . We just add the slope parameter to (6.27) in Theorem 6.6, so that the $\phi(s)$ in $B_\gamma(t)$ becomes:

$$\phi(s; \gamma) = e^{-\gamma \frac{(s-\frac{1}{2})^2}{s(1-s)}}, \quad \gamma > 0, \gamma \in \mathbb{R}. \quad (6.29)$$

The effect is that we can adjust the slope of the ERB-function. Note that the function still is (point) symmetric. In Figure 6.19 is the effect of the slope parameter illustrated. Four ERB-functions with different slope parameters are displayed on the left hand side in the figure. The least steep ERB-function is the violet function with the slope parameter, $\gamma = 0.4$. The function plotted in black is the default one with $\gamma = 1$. Then we can see the red function with $\gamma = 2$ and the green function with $\gamma = 3$. Finally in the figure is the ERB-function with the slope parameter $\gamma = 50$. We can clearly see that when the slope parameter goes toward infinity the B-function goes toward a step function at $t = 0.5$.

On the right hand side in Figure 6.19 is the first derivatives of the ERB-functions showed on the left hand side. The colors and the style is the same for the functions and the derivatives. The first derivative of the ERB-function with the slope parameter $\gamma = 50$ is scaled by 8.

The slope parameter has the same effect on the shape of the B-function, as the order parameter of the symmetric Beta-functions and the symmetric RB-functions have.

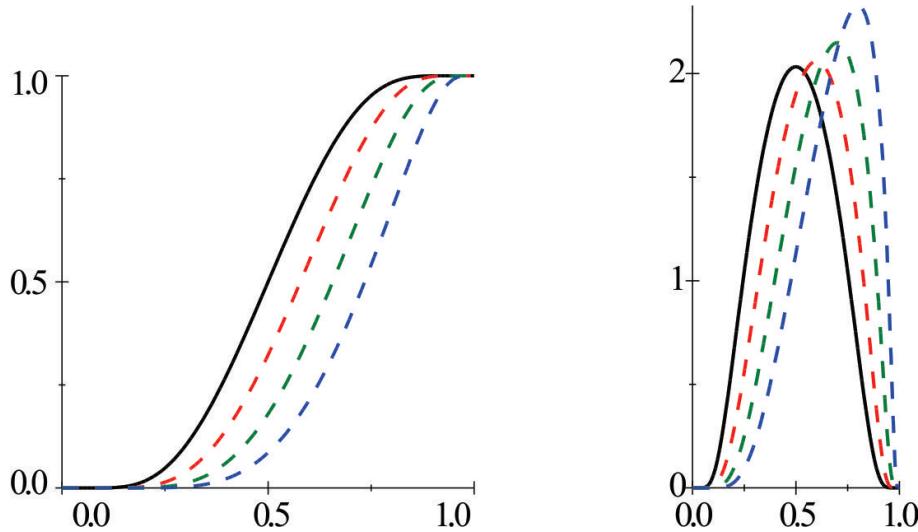


Figure 6.20: Four ERB-functions and their first derivatives are shown. On left hand side in the figure is the symmetric ERB-function, $B_{3,0.5}(t)$ in solid black, the asymmetric $B_{3,0.6}(t)$ in dashed red, the asymmetric $B_{3,0.7}(t)$ in dashed green and the asymmetric $B_{3,0.8}(t)$ in dashed blue. On the right hand side is their first derivatives plotted using the same color and style as the function itself.

6.6.2 The balance parameter μ

The next intrinsic parameter is the balance parameter μ . This parameter is analogous to the balance parameter in the $R\mu$ -function, see Corollary 6.1 in subsection 6.4.2. We now also add a balance parameter to (6.27) in Theorem 6.6, so that the $\phi(s)$ in $B_{\gamma,\mu}(t)$ now becomes:

$$\phi(s; \gamma, \mu) = e^{-\gamma \frac{(s-\mu)^2}{s(1-s)}}, \quad \gamma > 0 \quad \text{and} \quad 0 < \mu < 1 \quad \gamma, \mu \in \mathbb{R}. \quad (6.30)$$

The effect is just the same as for the $R\mu$ -function. An ERB-function with a balance parameter $\mu = 0.5$ is point symmetric, but all other values of μ gives asymmetric functions.

To provide an overview of the effect of a balance parameter, four functions together with their first derivatives are plotted in Figure 6.20. Note that the sequence of the parameters in the notation of the ERB-function is $B_{\gamma,\mu}(t)$.

To make it visually more easy, we have chosen to use a slope parameter $\gamma = 3$ in the example. The balance parameter varies from 0.5 to 0.8. On the left hand side in Figure 6.20 is $B_{3,0.5}(t)$ plotted in solid black, $B_{3,0.6}(t)$ in dashed red, $B_{3,0.7}(t)$ in dashed green and $B_{3,0.8}(t)$ is plotted in dashed blue.

As we can see, they are pushed to the right according to the value of μ . We can clearly see that it is only $B_{3,0.5}$, the solid black one, that is point symmetric. This is the same behavior as we could observe for the $R\mu$ -function from subsection 6.4.2.

In section 6.8 we will investigate what we call a balance-symmetry of B-functions.

6.6.3 The asymmetric tightening parameters α and β

The next intrinsic parameters are probably most of theoretical interest. We call them the asymmetric tightening parameters α and β . They are in a way analogous to the order parameter a and b in the Beta- and RB-function, see section 6.3, Theorem 6.3 and section 6.4, Theorem 6.4. Remember that, unlike Beta- and RB-functions, these parameters do not effect the order of the ERB-function, because the order is infinite. The shape of the function is of cause effected. We now also add the asymmetric tightening parameters to (6.27) in Theorem 6.6, such that the $\phi(s)$ in $B_{\gamma,\mu,\alpha,\beta}(t)$ is renewed.

This leads to a new definition

The complete Expo-Rational B-function

Theorem 6.7. *The complete Expo-rational function is an ERB-function and, thus a B-function. The expression is*

$$B_{\gamma,\mu,\alpha,\beta}(t) = S_{\gamma,\mu,\alpha,\beta} \int_0^t \phi(s; \gamma, \mu, \alpha, \beta) ds, \quad t \in [0, 1], \quad (6.31)$$

where

$$\phi(s; \gamma, \mu, \alpha, \beta) = e^{-\gamma \frac{|s-\mu|^{\alpha+\beta}}{s^\alpha(1-s)^\beta}}, \quad (6.32)$$

and

$$S_{\gamma,\mu,\alpha,\beta} = \left[\int_0^1 \phi(t; \gamma, \mu, \alpha, \beta) dt \right]^{-1}, \quad (6.33)$$

for

$$\gamma, \mu \in \mathbb{R}, \quad \text{restricted to} \quad \gamma > 0, \quad 0 < \mu < 1, \quad (6.34)$$

and

$$\alpha, \beta \in \mathbb{N}, \quad (\text{not including zero, i.e. } \alpha, \beta > 0). \quad (6.35)$$

The (Hermite) order of the complete ERB-function is infinite, and if $\alpha = \beta$ and $\mu = \frac{1}{2}$ then the complete ERB-function is symmetric according to **D5** in definition 6.1.

The complete ERB-function is C^∞ -smooth on $[0, 1] \subset \mathbb{R}$ if the intrinsic parameters $\gamma, \mu, \alpha, \beta$ are restricted according to (6.34) and (6.35).

A complete proof for Theorem 6.7 is given in [89], in a proof of Theorem 2.1, page 18–24. If α and/or β are not positive integers, then it is shown by Theorem 2.1 in [89] and by examples on page 27–30 in [89] that the second derivative and/or the third derivative are discontinue at $t = \mu$.¹

¹In [95], [43] and [89] is there an other naming of the intrinsic parameters. The slope parameter was called β in the previous articles. The balance parameter was called λ . α has the same name as before, but in this book has β replaced $\gamma\alpha$ from the previous articles. The previous articles also include a parameter σ as the exponent in the numerator of the expression. This was called the general definition. In Theorem 2.2 in [89] it was shown that to be able to scale and translate the function, we have to restrict the parameters so that the degree of the polynomial in the numerator and the denominator is the same. This was called the scalable subset. In this book we follow the definition of the scalable subset.

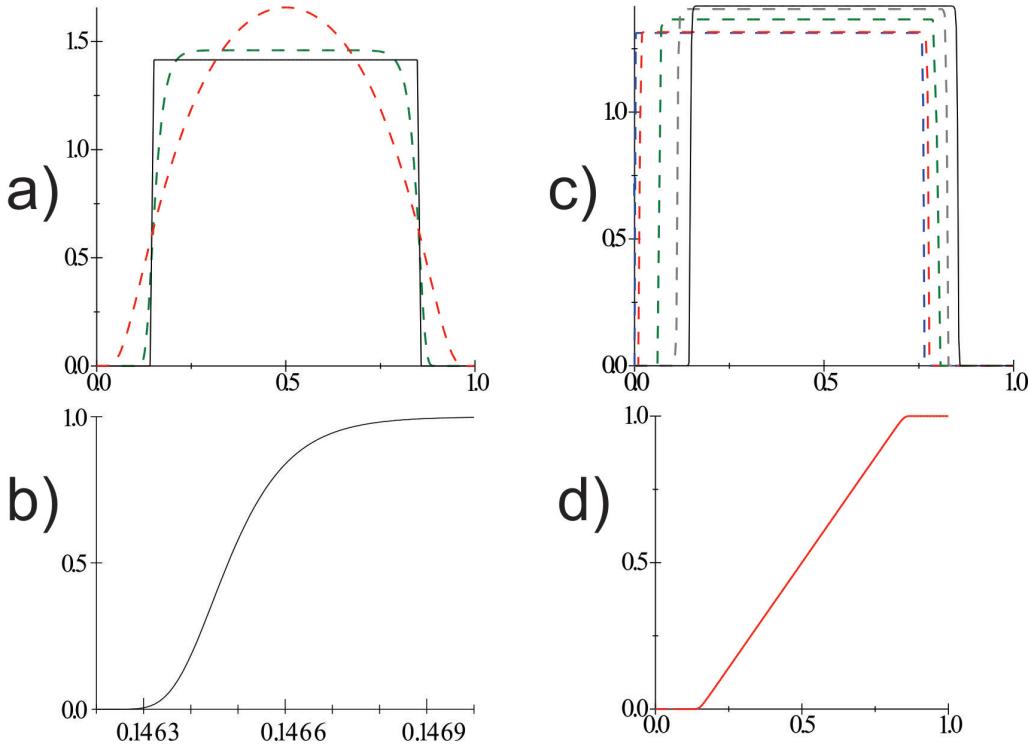


Figure 6.21: In plot a) is the first derivatives of: $B_{1,0.5,1,1}(t)$ plotted in dashed red, $B_{1,0.5,5,5}(t)$ in dashed green and $B_{1,0.5,1000,1000}(t)$ in solid black. In plot b) is the t-axis of plot a) sharply scaled, and we can see a part of the first derivative of $B_{1,0.5,1000,1000}(t)$. In plot c) is the first derivative of: $B_{1,0.5,50,50}(t)$ plotted in solid black, $B_{1,0.5,40,60}(t)$ in dashed gray, $B_{1,0.5,30,75}(t)$ in dashed green, $B_{1,0.5,16,80}(t)$ in dashed red and $B_{1,0.5,10,100}(t)$ in dashed blue. In plot d) is $B_{1,0.5,10,10}(t)$ plotted in solid red.

The asymmetric tightening parameters α and β are tightening the function so it goes toward a linear function on a partition of the domain. An example can be seen in Figure 6.21, plot d). The red curve in the figure is $B_{1,0.5,10,10}(t)$, and we can clearly see that it seems to be a linear function on approximately $[0.15, 0.85]$. The ERB-function is, however, still C^∞ -smooth. This can be seen in the next example. In Figure 6.21, plot a) is the first derivative of $B_{1,0.5,1,1}(t)$ plotted in dashed red, the first derivative of $B_{1,0.5,5,5}(t)$ is plotted in dashed green, and the first derivative of $B_{1,0.5,1000,1000}(t)$ is plotted in solid black. The shape of the function is first an ordinary hat function, the red one, to a sharper hat function, the green one, and the solid black one look very much like a step function. However, in plot b) is the t-axis of plot a) sharply scaled and we can now see that even the first derivative of $B_{1,0.5,1000,1000}(t)$, the solid black curve in a), is smooth.

In Figure 6.21, plot c) is the asymmetry illustrated. The first derivative of $B_{1,0.5,50,50}(t)$ is plotted in solid black, the first derivative of $B_{1,0.5,40,60}(t)$ is plotted in dashed gray, the first derivative of $B_{1,0.5,30,75}(t)$ is plotted in dashed green, the first derivative of $B_{1,0.5,16,80}(t)$ is plotted in dashed red, and the first derivative of $B_{1,0.5,10,100}(t)$ is plotted in dashed blue. The first one, $B_{1,0.5,50,50}(t)$ in solid black, is point symmetric, and we can see from the figure that it is centered about $t = 0.5$. The other four functions are pushed to the left, and the last one, $B_{1,0.5,10,100}(t)$ in dashed blue, is nearly pushed to the start point $t = 0$.

6.6.4 ERB-functions, differentiation

In Theorem 6.7 is the Expo-rational B-function including all intrinsic parameters defined. From (6.31) we have

$$B_{\gamma,\mu,\alpha,\beta}(t) = S_{\gamma,\mu,\alpha,\beta} \int_0^t \phi(s; \gamma, \mu, \alpha, \beta) ds, \quad t \in [0, 1].$$

We see that the first derivatives is,

$$B'_{\gamma,\mu,\alpha,\beta}(t) = S_{\gamma,\mu,\alpha,\beta} \phi(t; \gamma, \mu, \alpha, \beta),$$

and for other derivatives of order $j > 1$ we have

$$B_{\gamma,\mu,\alpha,\beta}^{(j)}(t) = f_j(t; \gamma, \mu, \alpha, \beta) B'_{\gamma,\mu,\alpha,\beta}(t), \quad (6.36)$$

where $f_j(t; \gamma, \mu, \alpha, \beta)$ will be described further below.

To make the text more readable we will in the following skip the intrinsic parameters in the notation, thus $f_j(t) = f_j(t; \gamma, \mu, \alpha, \beta)$. First, we name the exponent in (6.32) for

$$g(t) = -\gamma \frac{|t - \mu|^{\alpha+\beta}}{t^\alpha (1-t)^\beta}.$$

It follows that

$$g'(t) = \left(\frac{\alpha + \beta}{t - \mu} - \frac{\alpha - t(\alpha + \beta)}{t(1-t)} \right) g(t)$$

and

$$g''(t) = \left(\frac{\alpha + \beta}{t - \mu} - \frac{\alpha - t(\alpha + \beta)}{t(1-t)} \right) g'(t) - \left(\frac{(2t-1)\alpha - t^2(\alpha + \beta)}{t^2(1-t)^2} - \frac{\alpha + \beta}{(t-\mu)^2} \right) g(t),$$

and the higher order derivatives can be computed in the same way.

We look at the expression of the derivatives in (6.36) and get the following expression for $f_j(t)$ for j up to 5,

$$\begin{aligned} f_2(t) &= g'(t), \\ f_3(t) &= g''(t) + g'(t)^2, \\ f_4(t) &= g'''(t) + 3g'(t)g''(t) + g'(t)^3, \\ f_5(t) &= g''''(t) + 4g'''(t)g'(t) + 6g''(t)g'(t)^2 + 3g''(t)^2 + g'(t)^4. \end{aligned} \quad (6.37)$$

An easier task is to investigating differentiation of the default ERB-function defined in Theorem 6.6. This equation is defined by the default set of intrinsic parameters, $\mu = \frac{1}{2}$ and $\gamma = \alpha = \beta = 1$.

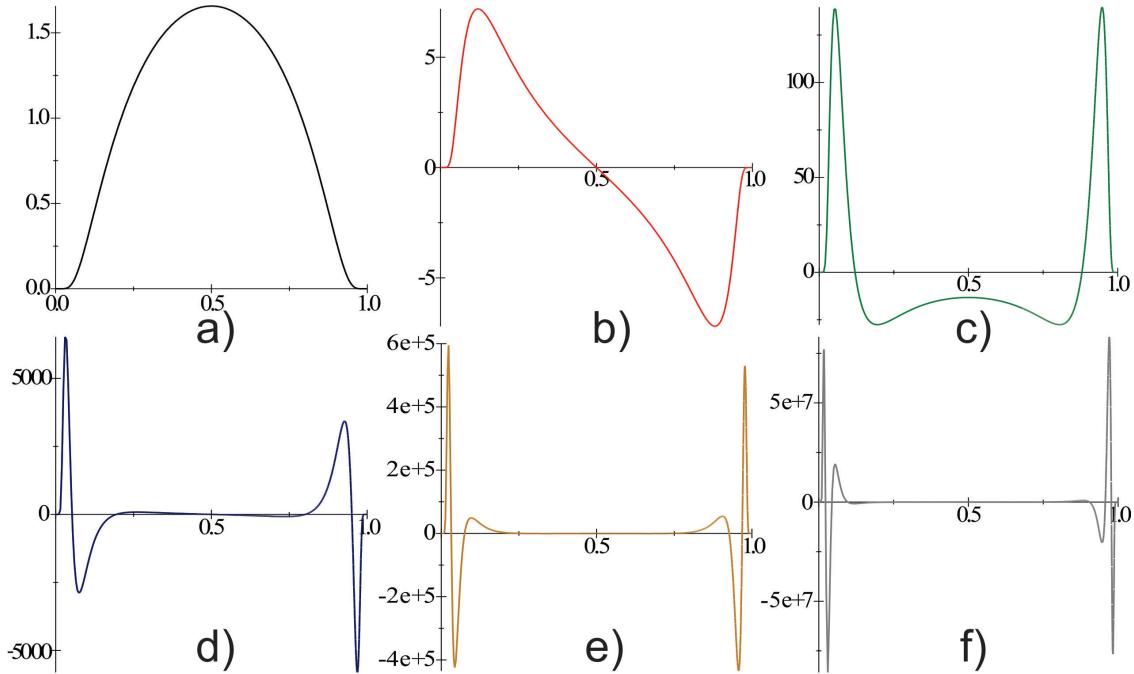


Figure 6.22: The first six derivatives of the default ERB-function defined in Theorem 6.6 are plotted. In figure a) is $D B(t)$, in b) is $D^2 B(t)$, in c) is $D^3 B(t)$, in d) is $D^4 B(t)$, in e) is $D^5 B(t)$ and in f) is $D^6 B(t)$ plotted.

The derivatives for the simple ERB-function, defined in Theorem 6.6, are

$$B'(t) = S \phi(t) \quad \text{and} \quad B^{(j)}(t) = f_j(t) B'(t), \text{ for } j = 2, 3, \dots$$

The exponent in (6.27) and its three first derivatives are

$$\begin{aligned} g(t) &= -\frac{(t-\frac{1}{2})^2}{t(1-t)}, & g'(t) &= \frac{1-2t}{4t^2(1-t)^2}, \\ g''(t) &= \frac{3t^2-3t+1}{2t^3(1-t)^3}, & g'''(t) &= \frac{3(3t^3-6t^2+4t-1)}{2t^4(1-t)^4}, \end{aligned}$$

and inserting these into (6.37), we get the expressions for $f_j(t)$ for j up to 5,

$$\begin{aligned} f_2(t) &= \frac{(1-2t)}{(2t(1-t))^2}, \\ f_3(t) &= \frac{\frac{3}{2}(1-2t)^4 - \frac{1}{2}}{(2t(1-t))^4}, \\ f_4(t) &= \frac{(3(1-2t)^6 + \frac{3}{2}(1-2t)^4 - 5(1-2t)^2 + \frac{3}{2})(1-2t)}{(2t(1-t))^6}, \\ f_5(t) &= \frac{\frac{15}{2}(1-2t)^{10} + \frac{45}{4}(1-2t)^8 - 33(1-2t)^6 + \frac{29}{2}(1-2t)^4 + \frac{3}{2}(1-2t)^2 - \frac{3}{4}}{(2t(1-t))^8}. \end{aligned} \tag{6.38}$$

In Figure 6.22 are the six first derivatives of the simple ERB-function plotted. We can observe that the extreme values grows rapidly with increasing order of the derivatives. The maximum value for the sixth derivative is thus more than 50000000.

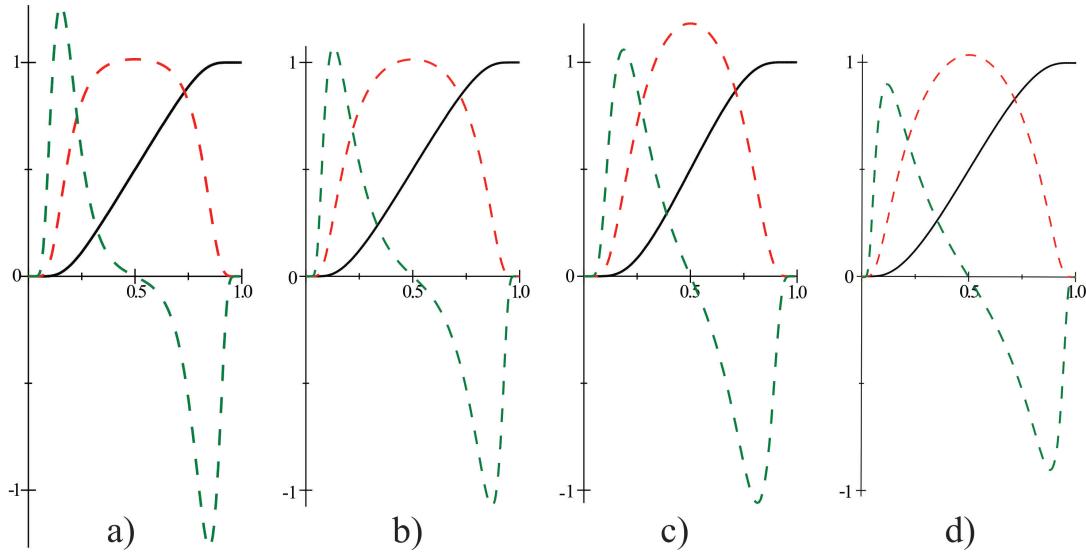


Figure 6.23: Four different ERB-functions are plotted. The function values are plotted in solid black, the first derivatives are scaled down by 1.6 and plotted in dashed red and the second derivatives are scaled down by 8 and plotted in dashed green. In plot a) is the function (6.40), in plot b) is the function (6.42) with $\gamma = 1.5$, in plot c) is the function (6.42) with $\gamma = 2$, and in d) is the simple ERB-function from Theorem 6.6 plotted.

6.7 Other Expo-Rational B-functions

In 2004 Ying and Zorin published a work on constructing surfaces of arbitrary smoothness [155]. The work was based on Grimm and Hughes work from 1995 on the same topic [72], and Navau and Garcia [110]. Navau and Garcia used a basis function related to a B-function, but that was not point-symmetric according to definition 6.1. This function was

$$\phi(t) = e^{\frac{-2}{t}} e^{\frac{-1}{1-t}}. \quad (6.39)$$

Ying and Zorin made a symmetric B-function by combining (6.39) with the rational construction of Hartmann [76],

$$B(t) = \frac{e^{\frac{-2}{t}} e^{\frac{-1}{1-t}}}{e^{\frac{-2}{t}} e^{\frac{-1}{1-t}} + e^{\frac{-2}{1-t}} e^{\frac{-1}{t}}}. \quad (6.40)$$

This ERB-function (6.40) together with it's first and second derivatives are shown in Figure 6.23, plot a). To be able to compare the shape is the simple ERB-function from Theorem 6.6 shown in plot d).

However, it is a standard way to create a symmetric B-function from a non-symmetric B-function. It is, summing the function with its antisymmetric twin and dividing the result by two. This gives the following ERB-function,

$$B(t) = \frac{1}{2} \left(1 + e^{\frac{-2}{t}} e^{\frac{-1}{1-t}} - e^{\frac{-2}{1-t}} e^{\frac{-1}{t}} \right). \quad (6.41)$$

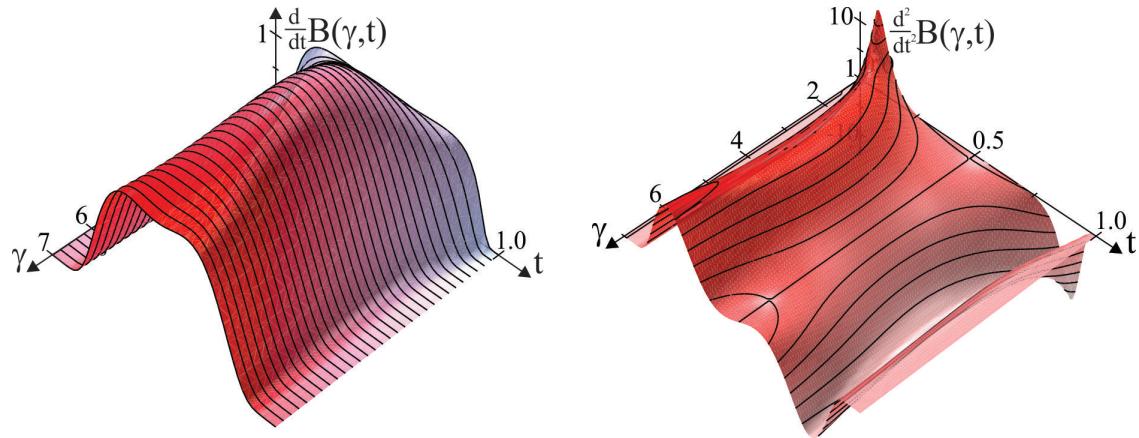


Figure 6.24: On left hand side is a plot of the first derivative of the function (6.42) where γ vary from 1 to 7. On right hand side is a plot of the first derivative of the function (6.42) where γ vary from 1 to 7.

In the following, we extend the ERB-function described in (6.41) by introducing a “slope” variable γ . The result is

$$B(t) = \frac{1}{2} \left(1 + e^{\frac{-\gamma}{t}} e^{\frac{-1}{1-t}} - e^{\frac{-\gamma}{1-t}} e^{\frac{-1}{t}} \right). \quad (6.42)$$

Notice that γ is not in general a true slope variable. This can be seen in Figure 6.24. In the figure is the first derivative shown on left hand side, sliced vertical with a constant γ , and the second derivative on right hand side, sliced vertically on hight levels. The plots are surfaces because the derivatives are plotted with γ varying from 1 to 7. From the second derivative, shown on left hand side in the figure, it follows that, to avoid internal local minima on the first derivative, we must restrict γ to be between approximately 1.13 and 6.26. Further, it follows that the maximum value of the first derivative, on left hand side in the figure, is at $t = \frac{1}{2}$ and $\gamma = \frac{e^2}{2} \approx 3.7$. Therefor, γ is a slope variable when it is restricted to approximately [1.13, 3.7].

In figure Figure 6.23, plot b) is (6.42) with $\gamma = 1.5$ shown together with its first and second derivatives. In plot c) is (6.42) with $\gamma = 2$ shown together with its first and second derivatives.

The first derivative of (6.39) when the number 2 is replaced by γ is

$$\phi'(t) = \gamma \frac{t^2 - t + 1}{t^2 (1-t)^2} e^{\frac{-1}{1-t}} e^{\frac{-\gamma}{t}}. \quad (6.43)$$

It follows that the first derivative of (6.42) is

$$B'(t) = \gamma \frac{t^2 - t + 1}{t^2 (1-t)^2} \frac{e^{\frac{-1}{1-t}} e^{\frac{-\gamma}{t}} + e^{\frac{-1}{t}} e^{\frac{-\gamma}{1-t}}}{2}. \quad (6.44)$$

We also see here that, with the same argument as for the first ERB-function the derivatives of all order are zero at $t = 0$ and $t = 1$ both for (6.40) and (6.42). It follows that the (Hermite) order is infinite for both.

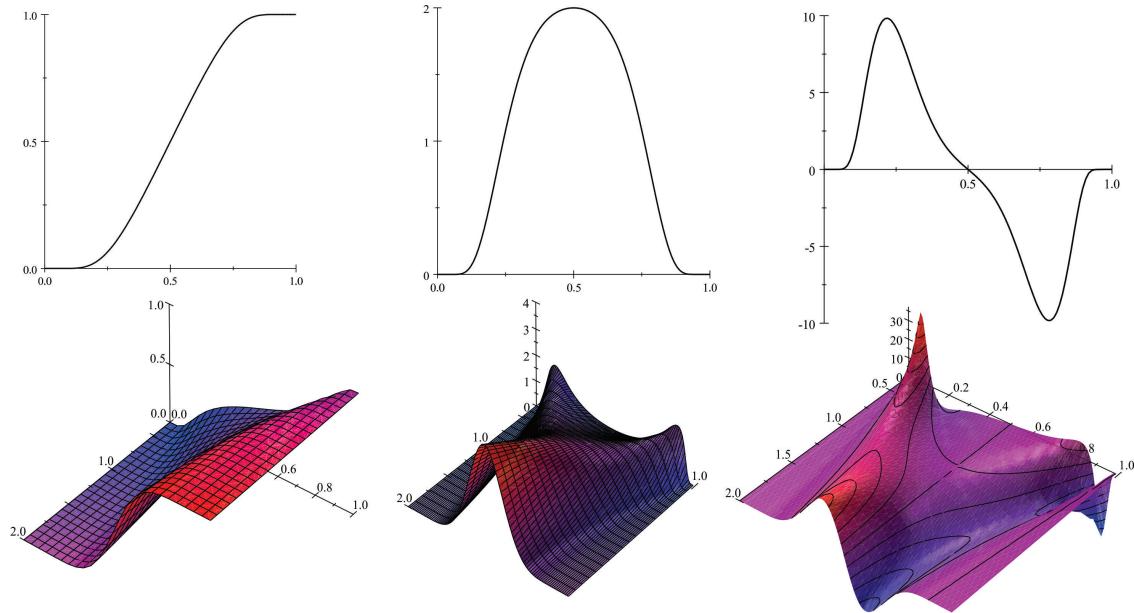


Figure 6.25: On left hand side is a plot of the first derivative of the function (6.42) where γ vary from 1 to 7. On right hand side is a plot of the first derivative of the function (6.42) where γ vary from 1 to 7.

6.7.1 Logistic Expo-Rational B-function

$$B(t) = \frac{1}{e^{\frac{1}{t-1} + \frac{1}{t}} + 1}. \quad (6.45)$$

$$B(t) = \frac{1}{e^{\frac{a}{t-1} + \frac{a}{t}} + 1}. \quad (6.46)$$

6.8 Point-, Order- and Balance-symmetry of B-functions

The symmetry properties are quite important for the B-functions. It tells us how the local functions influence the result, and it is important to control this essential property, if we want to introduce blending-splines by using B-functions.

We introduced in section 6.1 the point-symmetry of a B-function. It is that the function is symmetric about the point $(0.5, 0.5)$.

The Point-symmetry of a B-function

A B-function is symmetric or Point-symmetric if

$$B(t) + B(1-t) = 1, \quad t \in [0, 1] \subset \mathbb{R}.$$

This is the most important type of symmetry and tells us that the influence on the result of both local functions are the same. We have shown that for all groups and series of B-functions there exist Point-symmetric functions.

The next type of symmetry is Order-symmetry. This type of symmetry is opening for different orders, connected to knots in a spline-function based on blending.

The Order-symmetry of a B-function

A B-function is called Order-symmetric if

$$B_{a,b}(t) + B_{b,a}(1-t) = 1. \quad (6.47)$$

where a and b are the (Hermite) orders that are changing side in the two functions.

Theorem 6.8. *The following B-functions are Order-symmetric:*

- Beta-functions are Order-symmetric.
- RB-functions are Order-symmetric.
- ERB-functions are Order-symmetric in the sense that it is symmetric according to α and β .

Proof. The proof of Theorem 6.8 is divided in three parts, one for each type of B-functions.

Beta-functions: The Beta-function is the regularized incomplete beta function defined in (6.19). If we first look at the denominator of the fraction, we see that $\mathcal{B}(a, b) = \mathcal{B}(b, a)$ because it follows from (6.18). Therefor, we have a common denominator and we can just sum up the numerators. If we put (6.19) into (6.47) we get

$$\mathcal{B}(t; a, b) + \mathcal{B}(1-t; b, a) = \int_0^t x^a (1-x)^b dx + \int_t^1 (1-x)^b x^a dx = \int_0^1 x^a (1-x)^b dx.$$

It follows that the numerators and the denominator are equal, and it verifies that the sum is 1, and that the Beta-functions are Order-symmetric.

RB-functions: The RB-function is defined in Theorem 6.4, (6.23). If we put (6.23) into (6.47) we get

$$B_{a,b}(t) + B_{b,a}(1-t) = \frac{t^{a+1}}{t^{a+1} + (1-t)^{b+1}} + \frac{(1-t)^{b+1}}{(1-t)^{b+1} + t^{a+1}} = 1,$$

which verifies that the RB-functions are Order-symmetric.

ERB-functions: The ERB-function is defined in Theorem 6.7. If we exchange α and β and replace t with $1 - t$ in (6.32) we get

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

If $\mu = 0.5$, we see that $|(1-t)-\mu| = |-(t-(1-\mu))| = |t-\mu|$. It then follows that $\phi(1-t; \gamma, 0.5, \beta, \alpha) = \phi(t; \gamma, 0.5, \alpha, \beta)$. Thus

$$\begin{aligned} B_{\gamma, 0.5, \alpha, \beta}(t) + B_{\gamma, 0.5, \beta, \alpha}(1-t) &= \\ S_{\gamma, 0.5, \alpha, \beta} \int_0^t \phi(s; \gamma, 0.5, \alpha, \beta) ds + S_{\gamma, 0.5, \beta, \alpha} \int_t^1 \phi(s; \gamma, 0.5, \beta, \alpha) ds &= \\ S_{\gamma, 0.5, \beta, \alpha} \int_0^1 \phi(s; \gamma, 0.5, \alpha, \beta) ds &= 1. \end{aligned}$$

which verifies that the ERB-functions are Order-symmetric in the sense that it is symmetric according to α and β , if $\mu = 0.5$. \square

The last type of symmetry is Balance-symmetry. This type of symmetry is opening for different balance and thus weight, connected to knots in a spline-function based on blending.

The Balance-symmetry of a B-function

A B-function is called Balance-symmetric if

$$B_\mu(t) + B_{1-\mu}(1-t) = 1. \quad (6.48)$$

Theorem 6.9. *The following B-functions are Balance-symmetric:*

- *R μ -functions are Balance-symmetric.*
- *ERB-functions are Balance-symmetric.*

Proof. The proof of Theorem 6.9 is divided in two parts, one for each type of B-functions.

R μ -functions: The R μ -function is defined in Corollary 6.1, (6.24). If we put (6.24) into (6.48) we get

$$B_{\mu, a, b}(t) + B_{1-\mu, a, b}(1-t) = \frac{(1-\mu)t^{a+1}}{(1-\mu)t^{a+1} + \mu(1-t)^{b+1}} + \frac{\mu(1-t)^{a+1}}{\mu(1-t)^{a+1} + (1-\mu)t^{b+1}}. \quad (6.49)$$

it follows that if the order parameters $a = b$, then (6.49) sums up to 1. This verifies that the R μ -functions are Balance-symmetric if $a = b$.

ERB-functions: The ERB-function is defined in Theorem 6.7. If we replace μ and $1 - \mu$ and replace t with $1 - t$ in (6.32) we get

$$\phi(1-t; \gamma, 1-\mu, \alpha, \beta) = e^{-\gamma \frac{|(1-t)-(1-\mu)|^{\alpha+\beta}}{(1-t)^{\alpha}t^{\beta}}}. \quad (6.50)$$

If we look at the numerator of the exponent in (6.50), we see that $|(1-t) - (1-\mu)| = |t - \mu|$. Thus, it follows that if $\beta = \alpha$ then $\phi(1-t; \gamma, 1-\mu, \alpha, \alpha) = \phi(t; \gamma, \mu, \alpha, \alpha)$ which in turn means that $S_{\gamma, 1-\mu, \alpha, \alpha} = S_{\gamma, \mu, \alpha, \alpha}$, and we get

$$\begin{aligned} B_{\gamma, \mu, \alpha, \alpha}(t) + B_{\gamma, 1-\mu, \alpha, \alpha}(1-t) &= \\ S_{\gamma, \mu, \alpha, \alpha} \int_0^t \phi(s; \gamma, \mu, \alpha, \alpha) ds + S_{\gamma, 1-\mu, \alpha, \alpha} \int_t^1 \phi(s; \gamma, 1-\mu, \alpha, \beta) ds &= \\ S_{\gamma, \mu, \alpha, \alpha} \int_0^1 \phi(1-s; \gamma, \mu, \alpha, \alpha) ds &= 1. \end{aligned}$$

which verifies that the ERB-functions are Balance-symmetric if $\alpha = \beta$. \square

As a conclusion of the section we will show that some B-functions can be Order-symmetric and Balance-symmetric simultaneously.

Theorem 6.10. *The following B-functions are Order-symmetric and Balance-symmetric simultaneously:*

R μ -functions fulfills

$$B_{\mu, a, b}(t) = B_{1-\mu, b, a}(1-t) \quad (6.51)$$

ERB-functions fulfills

$$B_{\gamma, \mu, \alpha, \beta}(t) = B_{\gamma, 1-\mu, \beta, \alpha}(1-t). \quad (6.52)$$

Proof. The proof of Theorem 6.10 is divided in two parts, one for each type of B-functions.

R μ -functions: The R μ -function is defined in Corollary 6.1, (6.24). If we put (6.24) into (6.51) we get

$$B_{\mu, a, b}(t) + B_{1-\mu, b, a}(1-t) = \frac{(1-\mu)t^{a+1}}{(1-\mu)t^{a+1} + \mu(1-t)^{b+1}} + \frac{\mu(1-t)^{b+1}}{\mu(1-t)^{b+1} + (1-\mu)t^{a+1}}. \quad (6.53)$$

We clearly see that (6.53) sums up to 1. This confirms that the R μ -functions simultaneously are Balance- and Order-symmetric.

ERB-functions: The ERB-function is defined in Theorem 6.7. If we exchange α and β , replace μ and $1-\mu$ and replace t with $1-t$ in (6.32) we get

$$\phi(1-t; \gamma, 1-\mu, \beta, \alpha) = e^{-\gamma \frac{|(1-t)-(1-\mu)|^{\beta+\alpha}}{(1-t)^{\beta} t^{\alpha}}}. \quad (6.54)$$

If we look at the numerator of the exponent in (6.54), we see that $|(1-t) - (1-\mu)| = |t - \mu|$. Thus, it follows that $\phi(1-t; \gamma, 1-\mu, \beta, \alpha) = \phi(t; \gamma, \mu, \alpha, \beta)$ which in turn means

that $S_{\gamma,1-\mu,\beta,\alpha} = S_{\gamma,\mu,\alpha,\beta}$, and we get

$$\begin{aligned} B_{\gamma,\mu,\alpha,\beta}(t) + B_{\gamma,1-\mu,\beta,\alpha}(1-t) &= \\ S_{\gamma,\mu,\alpha,\beta} \int_0^t \phi(s; \gamma, \mu, \alpha, \beta) ds + S_{\gamma,1-\mu,\beta,\alpha} \int_t^1 \phi(s; \gamma, 1-\mu, \beta, \alpha) ds &= \\ S_{\gamma,\mu,\beta,\alpha} \int_0^1 \phi(1-s; \gamma, \mu, \alpha, \beta) ds &= 1, \end{aligned}$$

confirming that the ERB-functions simultaneously are Balance- and Order-symmetric. \square

6.9 Computing B-functions

For most B-functions and their derivatives may it readily be made directly algorithms from their formulas. An algorithm for calculating B-functions, including their derivatives is called an *evaluator*. Implementations of efficient evaluators for Beta-functions, RB-functions and TB-functions for given intrinsic parameters is easy. However, a general evaluator for B-functions is more complicated to make, and an evaluator for ERB-functions in general is not possible to make directly, because it require numerical integrations and/or approximations.

In Appendix A is a numerical evaluator described. The evaluator was initially made for ERB-functions, but is later modified to other B-functions.² The evaluator is based on approximation of the B-function by piecewise 3th degree Hermite functions, where the coefficients for all of the Hermite functions in all subareas are pre-computed and stored. The default partitioning is 1024 and this partitioning requires $1024 \times 6 \times 8 = 48k$ -byte memory.

The the error bounds are important and for ERB-functions is the error bound: 10^{-13} for the function value, 10^{-9} for the first derivative and 10^{-6} for the second derivative.

The efficiency of the evaluator is 6 multiplications for the function value, 4 multiplications for the first derivative, 5 multiplications for the second derivative ... C++ codes for the evaluator are present at <http://episteme.hin.no/downloads/B-evaluator>.

²The evaluator was introduced in [89], made for Expo-Rational B-splines. It was made for the scalable subset, which is practically identical with the ERB-function described here. Later Gancheva and Delistoyanova expanded the evaluator in [61], to also include Beta-functions.

Chapter 7

Blending splines

Recall the formula for a B-spline curve,

$$c(t) = \sum_{j=1}^n c_j b_{k,j}(t).$$

where $\{c_j\}_{j=1}^n$, is the vector of coefficients, control points. These points defines the control polygon of the curve. The set of B-spline basis functions $\{b_{k,j}(t)\}_{j=1}^n$ is defined by the knot vector $\bar{t} = \{t_1, t_2, \dots, t_{n+k}\}$, where the order $k = d + 1$ and where d is the polynomial degree of the B-splines.

Because $b_{k,j}(t)$ has local support, i.e. is different from zero only on the interval $t \in [t_j, t_{j+k}]$, we can formulate the B-spline curve formula as following,

$$c(t) = \sum_{j=i-k}^i c_j b_{k,j}(t).$$

where the index i is determined by $t_i \leq t < t_{i+1}$.

In subsection 5.5.3 was matrix formulation introduced. Thus, for a third degree B-spline curve, $d = 3$, we have the following formula,

$$c(t) = \begin{pmatrix} 1 - w_{1,i}(t) & w_{1,i}(t) \end{pmatrix} \begin{pmatrix} 1 - w_{2,i-1}(t) & w_{2,i-1}(t) & 0 \\ 0 & 1 - w_{2,i}(t) & w_{2,i}(t) \end{pmatrix} \begin{pmatrix} 1 - w_{3,i-2}(t) & w_{3,i-2}(t) & 0 & 0 \\ 0 & 1 - w_{3,i-1}(t) & w_{3,i-1}(t) & 0 \\ 0 & 0 & 1 - w_{3,i}(t) & w_{3,i}(t) \end{pmatrix} \begin{pmatrix} c_{i-3} \\ c_{i-2} \\ c_{i-1} \\ c_i \end{pmatrix}, \quad (7.1)$$

where

$$w_{d,i}(t) = \begin{cases} \frac{t-t_i}{t_{i+d}-t_i}, & \text{if } t_i \leq t < t_{i+d} \\ 0, & \text{otherwise,} \end{cases} \quad (7.2)$$

and where the index i is determined by $t_i \leq t < t_{i+1}$.

The matrix formulation illustrate that B-spline curves formula is based on corner cutting. Some important properties of B-spline curves comes from this corner cutting algorithm illustrated in the matrix form in (7.1). As described in section 5.5.1 the properties connected to the corner cutting are:

- The strong convex hull property,
- the variation diminishing property.

In addition can the following properties easily be seen from the matrix formula,

- the local modification scheme,
- the affine invariance property.

Local modification follows from that the formula (for degree 3) only depend on four points, and the affine invariance follows from that each row in all matrices sums up to 1.

The continuity property can be directly derived from (7.2), i.e. $w_{d,i}(t) : [t_i, t_{i+d}] \rightarrow [0, 1]$. Recall the Hermite-order of the B-function in definition 6.2. The linear B-function $B(t) = t$ has Hermite-order zero. It follows that t^2 has left side Hermite-order 1, t^3 has left side Hermite-order 2, etc. It follows that the Hermite-order of the B-spline basis is $d - 1$ at the ends because it start with t^d (where t is translated and scaled) and ends with $(1-t)^d$ (translated and scaled). and at the internal knots is there only one linear function ending and starting which reduce the continuity from d to $d - 1$. This is the explanation for the B-spline continuity at a simple knot, the continuity is the polynomial degree minus one. By the same arguments it follows that the continuity is reduced by 1 if two knots has the same value and that the continuity in general is the degree minus the multiplicity of the knots.

$$b_{i,d}(t) = w_{1,i}(t)w_{2,i}(t) \cdots w_{d,i}(t), \quad t \in [t_i, t_{i+1}].$$

$$b_{i,d}(t) = (1 - w_{1,i}(t))(1 - w_{2,i}(t)) \cdots (1 - w_{d,i}(t)), \quad t \in [t_{i-d-1}, t_d].$$

$$b_{i,d}(t) = w_{1,i}(t)w_{2,i}(t) \cdots w_{d,i}(t), \quad t \in [t_{i-2}, t_{i+1}].$$

7.1 B-splines with B-function

Recall the matrices in (7.1). Most elements in the matrices are zero. At each row of the matrices has the elements that are not zero the values $w_{d,j}(t)$ and $1 - w_{d,j}(t)$. Since $w_{d,j}(t)$ is a linear function from 0 to 1, are each row a linear interpolation between to points. It is still a linear interpolation if we instead of linear functions use higher order B-function as elements of the B-spline matrices. We then get the following expression for B-spline

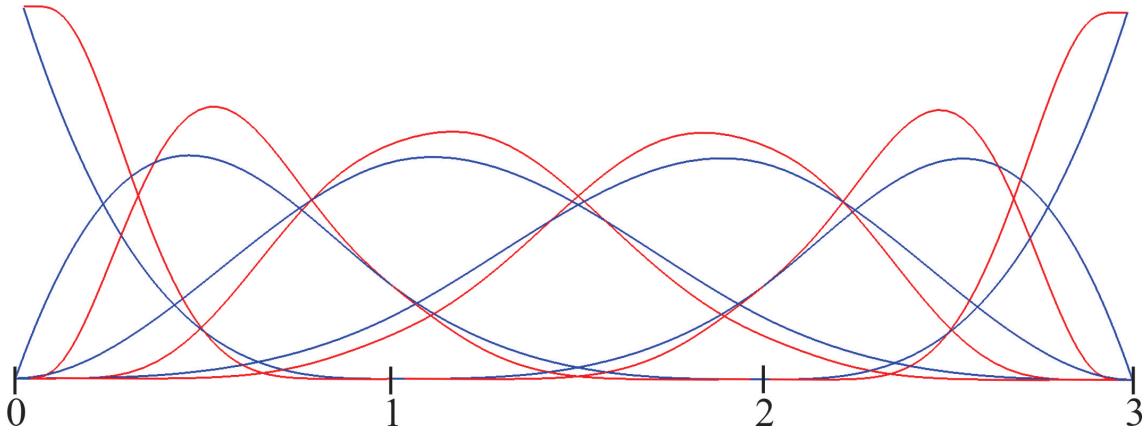


Figure 7.1: The blue curves are the polynomial B-splines on a given knot vector $\vec{t} = \{0,0,0,0,1,2,3,3,3,3\}$. The red curves are the Expo-Rational B-splines on the same knot vector.

curves:

$$\begin{aligned} c(t) = & \begin{pmatrix} 1 - B \circ w_{1,i}(t) & B \circ w_{1,i}(t) \end{pmatrix} \begin{pmatrix} 1 - B \circ w_{2,i-1}(t) & B \circ w_{2,i-1}(t) & 0 \\ 0 & 1 - B \circ w_{2,i}(t) & B \circ w_{2,i}(t) \end{pmatrix} \\ & \begin{pmatrix} 1 - B \circ w_{3,i-2}(t) & B \circ w_{3,i-2}(t) & 0 & 0 \\ 0 & 1 - B \circ w_{3,i-1}(t) & B \circ w_{3,i-1}(t) & 0 \\ 0 & 0 & 1 - B \circ w_{3,i}(t) & B \circ w_{3,i}(t) \end{pmatrix} \begin{pmatrix} c_{i-3} \\ c_{i-2} \\ c_{i-1} \\ c_i \end{pmatrix}, \end{aligned} \quad (7.3)$$

where the index i is determined by $t_i \leq t < t_{i+1}$.

The extension do not change the main properties of the B-spline curve. We still have the corner cutting algorithm inducing all the properties of convex hull, local modification scheme, variation diminishing etc, and the linear interpolation scheme induce the affine invariant property.

The main difference compared to polynomial B-splines is the continuity properties. The continuity at the knots will increase with the order of the B-function. If the B-function has order infinity, then the curve will be C^∞ -smooth.

In Figure 7.1 is the B-splines (the basis functions) plotted for both polynomial and Expo-Rational B-splines. The main difference that can be observed is that the derivatives of all basis functions are zero at the start and at the end of their support. Most clearly can this be seen for the first and second basis function at $t = 0$ and for the 5th and 6th basis function at $t = 3$.

$$\overrightarrow{\mathcal{B}}_{g,i,d} = \left(B \circ w_{g,i}(t) \quad \delta_{g,i} B' \circ w_{g,i}(t) \quad \delta_{g,i}^2 B'' \circ w_{g,i}(t) \cdots \delta_{g,i}^d B^{(d)} \circ w_{g,i}(t) \right) \quad (7.4)$$

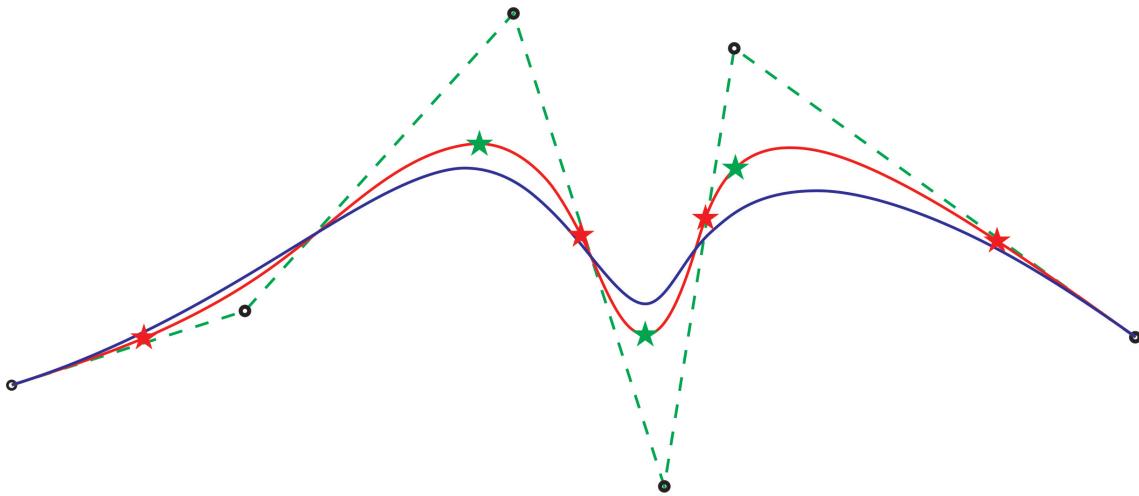


Figure 7.2: A plot of a 3th degree polynomial B-spline in blue. The dashed green lines are the control polygon. The red curve is an Expo-Rational B-spline (ERBS) with the same knot vector and control points as the polynomial B-spline. The red and green stars marks extreme values of the speed of the ERBS.

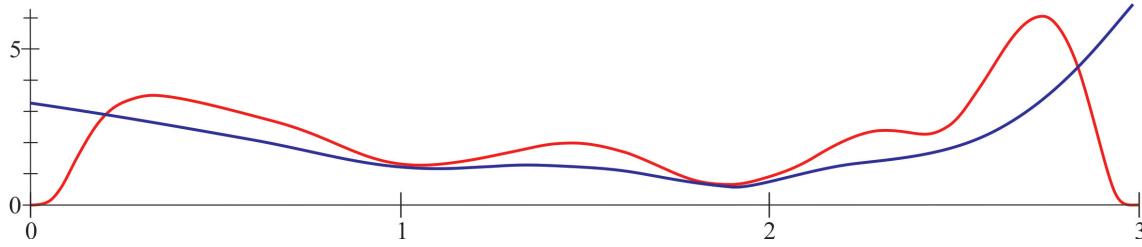


Figure 7.3: The speed of the curves in Figure 7.2. The blue function is the speed of the polynomial B-spline and the red function is the speed of the Expo-Rational B-splines.

$$M_{k,d}(t) = \begin{pmatrix} \vec{\mathcal{B}}_{1,i,d} \\ \vec{\mathcal{B}}_{2,i-1,d} \\ \vec{\mathcal{B}}_{2,i,d} \\ \hline \vec{\mathcal{B}}_{3,i-2,d} \\ \vec{\mathcal{B}}_{3,i-1,d} \\ \vec{\mathcal{B}}_{3,i,d} \\ \dots \end{pmatrix}$$

where the index i is determined by $t_i \leq t < t_{i+1}$ and where the number of rows is $\frac{k(k-1)}{2}$ and the number of columns is $d + 1$.

- A simple version of Expo-Rational B-splines will be given, followed by a short description of their basic properties (section 7.2).
- Next there is a general definition of Expo-Rational B-splines (section 7.3). Compared to the simple version in section 7.2, 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. I.e. in [79] Klaus Höllig introduced FEM with B-splines.

- In section 7.4 the basic properties of the general definition of Expo-Rational B-splines are presented. We also give the range of the parameters which these properties hold.
- The “scalable subset” is defined in section 7.5. 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 computations tremendously.
- In section 7.6 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 book.
- In section 7.7 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 7.8 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 7.9. 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, 7.10, 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 3. *The simple version described in section 7.2, is actually the same as the default set described in section 7.6, but the default set is, according to the computation, further simplified.*

7.2 A simple version

Let $t_k \in \mathbb{R}$ and $t_k < t_{k+1}$ for $k = 0, 1, 2, \dots, n$, i.e. to simplify we first 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”, definition 7.4, in section 7.6) is then defined by the following.

Definition 7.1. 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:

$$B_k(t) = \begin{cases} S_{k-1} \int_{t_{k-1}}^t \Psi_{k-1}(s) ds, & \text{if } t_{k-1} < t \leq t_k \\ S_k \int_t^{t_{k+1}} \Psi_k(s) ds, & \text{if } t_k < t < t_{k+1} \\ 0, & \text{otherwise} \end{cases} \quad (7.5)$$

with

$$\Psi_k(t) = e^{-\frac{(t-\frac{t_k+t_{k+1}}{2})^2}{(t-t_k)(t_{k+1}-t)}}, \quad (7.6)$$

and the constant scaling factor

$$S_k = \left(\int_{t_k}^{t_{k+1}} \Psi_k(t) dt \right)^{-1}.$$

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 7.4 and the text about the partition of unity in e.g [36], [124] and [53]).

An example of the plot of $B_k(t)$, $\Psi_{k-1}(t)$ and $-\Psi_k(t)$ is illustrated in Figure 7.4. 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 function $B_k(t)$ is positive on (t_{k-1}, t_{k+1}) and zero otherwise.
- P2.** The set of basis functions $B_k(t)$ for $k = 1, 2, \dots, n$ 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 $k = 1, 2, \dots, n$ $B_k(t_k) = 1$ holds, which is also a property of the Lagrange form of polynomials.
- P4.** Every basis function $B_k(t)$ is C^∞ -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 7.4 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 7.4 divided into 2 properties, 4 and 5. It, thus, follows that **P5** goes to property 6 in section 7.4.

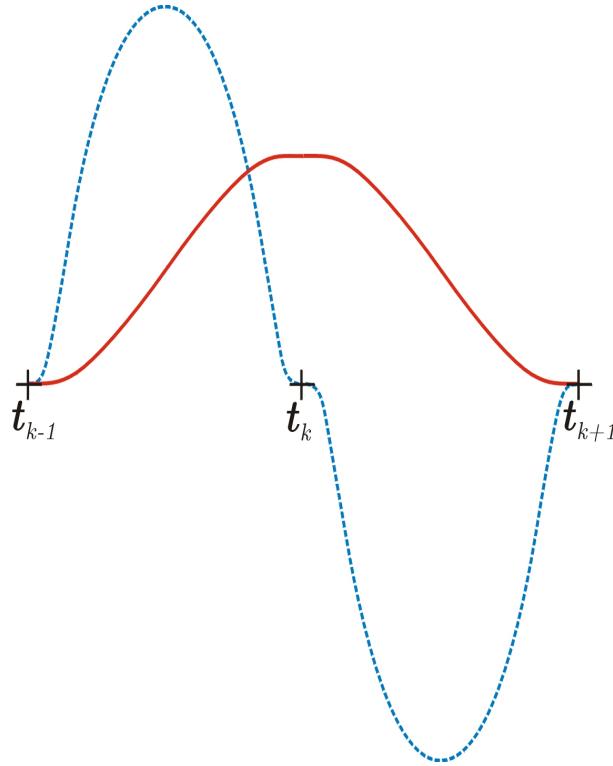


Figure 7.4: A graph of $B_k(t)$ (solid red) and its first derivative (dotted blue). The knots t_{k-1}, t_k and t_{k+1} are also marked on the plot.

7.3 Generalized definition

Let $t_k \in \mathbb{R}$ and $t_k \leq t_{k+1}$ for $k = 0, 1, 2, \dots, n$, 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 [95], [41] and [42]). The definition is according these previous articles (and was first proposed by Lubomir Dechevsky).

Definition 7.2. An Expo-Rational B-spline, associated with three increasing knots t_{k-1} , t_k and t_{k+1} , $B_k(t) = B_k(t; \alpha_{k-1}, \beta_{k-1}, \gamma_{k-1}, \lambda_{k-1}, \sigma_{k-1}, \alpha_k, \beta_k, \gamma_k, \lambda_k, \sigma_k)$ is defined by

$$B_k(t) = \begin{cases} S_{k-1} \int_{t_{k-1}}^t \Psi_{k-1}(s) ds, & \text{if } t_{k-1} < t \leq t_k, \\ S_k \int_t^{t_{k+1}} \Psi_k(s) ds, & \text{if } t_k < t < t_{k+1}, \\ 0, & \text{otherwise,} \end{cases} \quad (7.7)$$

with

$$\Psi_k(t) = e^{-\beta_k \frac{|t - ((1-\lambda_k)t_k + \lambda_k t_{k+1})|^{2\sigma_k}}{((t-t_k)(t_{k+1}-t))^{\alpha_k}}}, \quad (7.8)$$

and the constant scaling factor

$$S_k = \frac{1}{\int_{t_k}^{t_{k+1}} \Psi_k(t) dt},$$

where

$$\alpha_k > 0, \quad \beta_k > 0, \quad \gamma_k > 0, \quad 0 \leq \lambda_k \leq 1, \quad \sigma_k \geq 0.$$

Remark 4. Equation (7.7) allows the use of multiple knots. As with polynomial splines, here multiple knots are also a tool for making discontinuities.

It is possible to use multiple knots, but only with multiplicity 2. Multiplicity will only distinguish between C^∞ -smoothness and discontinuity. Multiple knots are discussed further in the sections 7.4 and 7.8.

The scaling factor S_k is introduced because of the need of normalizing in definition 7.2, that $B_k(t_k) = 1$, and to simplify the equations.

To study the derivatives of the ERBS defined in definition 7.2 we need some additional notations. If the respective limits exist, the right derivative will be,

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

and the left derivative

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

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 will now update expression 7.7, by including the derivatives,

$$B_k(t) = \begin{cases} S_{k-1} \int_{t_{k-1}}^t \Psi_{k-1}(s) ds, & \text{if } t_{k-1} < t \leq t_k, \\ S_k \int_t^{t_{k+1}} \Psi_k(s) ds, & \text{if } t_k < t < t_{k+1}, \\ 0, & \text{otherwise,} \end{cases} \quad (7.9)$$

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

$$DB_k(t) = \begin{cases} S_{k-1} \Psi_{k-1}(t), & \text{if } t_{k-1} < t \leq t_k, \\ -S_k \Psi_k(t), & \text{if } t_k < t < t_{k+1}, \\ 0, & \text{otherwise.} \end{cases} \quad (7.10)$$

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

$$D^j B_k(t) = \begin{cases} f_{j,k-1}(t) \Psi_{k-1}(t), & \text{if } t_{k-1} < t \leq t_k, \\ -f_{j,k}(t) \Psi_k(t), & \text{if } t_k < t < t_{k+1}, \\ 0, & \text{otherwise,} \end{cases} \quad (7.11)$$

where $f_{1,k}(t) = S_k$. In Figure 7.4, there is an example of an Expo-Rational B-spline basis and its first derivative. The next derivatives involve $f_{j,k}$, $j = 2, 3, \dots$ 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 (7.9-7.11) 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 (7.9) is continuous on (t_{k-1}, t_{k+1}) .

The structure of the equation 7.11 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 7.6 will be given the seven first factors for the default set of intrinsic parameters. These factors will be plotted in Figure 7.11 (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 7.1. 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 7.8, which will be denoted by

$$c_k(t) = t - ((1 - \lambda_k) t_k + \lambda_k t_{k+1}). \quad (7.12)$$

Using (7.12), we define $g_k(t)$ to be the derivative with respect to t of the exponent in (7.8). Thus the extra factor generated when computing $D\Psi_k(t)$ is

$$g_k(t) = \beta_k \left(\alpha_k \left(\frac{1}{t - t_k} - \frac{\gamma_k}{t_{k+1} - t} \right) c_k(t) - 2\sigma_k \right) \frac{c_k(t) (c_k(t)^2)^{\sigma_k-1}}{((t - t_k)(t_{k+1} - t))^{\gamma_k}}. \quad (7.13)$$

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

$$\begin{aligned} g'_k(t) = & \beta_k \left(2\sigma_k \left(\alpha_k \left(\frac{1}{t - t_k} - \frac{\gamma_k}{t_{k+1} - t} \right) c_k(t) - 1 \right) - \alpha_k \left(\frac{1}{(t - t_k)^2} - \frac{\gamma_k}{(t_{k+1} - t)^2} \right) c_k(t)^2 + \right. \\ & \left. \left(2\sigma_k - \alpha_k \left(\frac{1}{t - t_k} - \frac{\gamma_k}{t_{k+1} - t} \right) c_k(t) \right) \left(\frac{\alpha_k(t_{k+1}-t(1+\gamma_k)+t_k\gamma_k)c_k(t)}{(t-t_k)(t_{k+1}-t)} \right) \right) \frac{(c_k(t)^2)^{\sigma_k-1}}{((t - t_k)(t_{k+1} - t))^{\gamma_k}}. \end{aligned} \quad (7.14)$$

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

$$f_{j,k}(t) = f'_{j-1,k}(t) + f_{j-1,k}(t) g_k(t), \quad j > 1, \quad (7.15)$$

where $f_{j,k}(t)$, $j > 1$, are rational functions at least when α_k , γ_k and $\frac{\sigma_k}{2}$ are nonnegative integers. Starting with $f_{1,k}(t) = S_k$ we can expand definition (7.15) to get the next functions. For $j = 2, 3, 4, 5$ we get

$$\begin{aligned} f_{1,k}(t) &= S_k, \\ f_{2,k}(t) &= S_k g_k(t), \\ f_{3,k}(t) &= S_k (g'_k(t) + g_k^2(t)), \\ f_{4,k}(t) &= S_k (g''_k(t) + 3g_k(t)g'_k(t) + g_k^3(t)), \\ f_{5,k}(t) &= S_k (g'''_k(t) + 3g_k'^2(t) + 4g_k(t)g''_k(t) + 6g_k^2(t)g'_k(t) + g_k^4(t)). \end{aligned} \quad (7.16)$$

where g''_k and g'''_k are computed recursively from 7.14.

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 7.1, and in the proof on page 194 and in section 7.5.

7.4 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 7.1. *There are six basic properties of the basis function $B_k(t)$, $k = 1, \dots, n$. 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.,*

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

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

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

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.,*

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

$$\lim_{t \rightarrow t_{k+}} B_k(t) = 1 \quad \text{if } t_k < t_{k+1}.$$

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^∞ -smooth at the respective exterior knot, t_{k-1} or t_{k+1} , i.e.,

$$\begin{aligned} \lim_{t \rightarrow t_{k-1}^+} D^j B_k(t) &= 0 \quad \text{for } j = 0, 1, 2, \dots, \text{ if } t_{k-1} < t_k \text{ and} \\ \lim_{t \rightarrow t_{k+1}^-} D^j B_k(t) &= 0 \quad \text{for } j = 0, 1, 2, \dots, \text{ if } t_k < t_{k+1}. \end{aligned}$$

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

$$\begin{aligned} D^j B_k(t_k) &= 0 \quad \text{for } j = 1, 2, \dots, \\ \text{and they are actually zero at every knot.} \end{aligned}$$

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 5. The name Expo-Rational B-splines refers to that the exponent of Ψ_k in (7.8), is a rational function. Because the intrinsic parameters σ , α and γ are not only positive integers, the exponent in Ψ_k are actually extended to more general functions. The exponent of Ψ_k will not be a true rational polynomial function if $\sigma = \frac{n}{2}$, for $n = 1, 2, \dots$ and α and γ 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 σ , α and γ all are positive integers is entirely inside the restrictions on the intrinsic parameters in Theorem 7.1, and they will ensure that the ERBS will have all properties from Theorem 7.1.

Remark 6. From Theorem 7.1, it is clear that $B_k(t)$, under the three additional restrictions, is C^∞ -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 converge 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 (7.8) 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 [53]).
- 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^∞ on \mathbb{R} when the knots are simple.
- Property **P6** extends the interpolation property to Hermite interpolation.

Proof. The proof (of Theorem 7.1) is first dealing with the three first properties **P1**, **P2** and **P3** in a numbered list.

1. Property **P1** follows directly from definition 7.2. 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 (7.9)). 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 7.2 we know that S_k is the inverse of the integral and therefore

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

3. Both parts of property **P3** follow directly from the fact that the numerator and the denominator, in both parts of (7.7) 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 $j = 0, 1, 2, \dots$ 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 [119]),

$$\lim_{x \rightarrow 0^+} \frac{e^{-\frac{a}{x}}}{bx^n} = 0, \quad \text{for any } n, \text{ and any } a > 0 \text{ and } b \neq 0, \quad (7.17)$$

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) $\lim_{t \rightarrow t_{k+1}-} B_k(t) = 0$. This follows directly from definition 7.2.
- (b) $\lim_{t \rightarrow t_k+} B_{k+1}(t) = 0$. This also follows directly from definition 7.2.
- (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 7.1, 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 (7.8) (named $\bar{\zeta}(t)$) goes towards $-\infty$ when $t \rightarrow t_{k+1}-$. If $\lambda_k = 1$, the exponent is,

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

It then follows that

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

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

$$2\sigma_k < \gamma_k \alpha_k \quad \text{if } \lambda_k = 1, \quad (7.18)$$

or the constraint

$$\lambda_k < 1, \quad (7.19)$$

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

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

We shall see that under the additional restriction 2 in Theorem 7.1, 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 (7.8) (named $\hat{\zeta}(t)$) goes toward $-\infty$ when $t \rightarrow t_k+$. If $\lambda_k = 0$, the exponent is,

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

It thus follows that

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

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

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

or the constraint

$$\lambda_k > 0$$

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

- (e) $\lim_{t \rightarrow t_k+} DB_{k+1}(t) = 0$. The proof is the same as for (d) because $\lim_{t \rightarrow t_k+} DB_{k+1}(t) = \psi_k(t_k)$.
- (f) $\lim_{t \rightarrow t_{k+1}-} DB_k(t) = 0$. The proof is the same as for (c) because $\lim_{t \rightarrow t_{k+1}-} DB_k(t) = \psi_k(t_{k+1})$.
- (g) $D^j B_{k+1}(t_{k+1}) = 0$ for $j = 2, 3, \dots$. The proof follows.

We have to show that $\lim_{t \rightarrow t_{k+1}-} f_{j,k}(t) \psi_k(t) = 0$. The question is, therefore, whether $\psi_k(t)$ goes faster towards 0 than the denominator in $f_{j,k}(t)$ for $j = 2, 3, \dots$ when $t \rightarrow t_{k+1}-$. 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 7.1). Suppose first that $\lambda_k < 1$, the limit when $t \rightarrow t_{k+1}-$ of the exponent in (7.8) (named $\zeta(t)$) can in this case be reformulated to

$$\lim_{t \rightarrow t_{k+1}-} \zeta(t) = \lim_{t \rightarrow t_{k+1}-} \left[-\frac{a}{(t_{k+1} - t)^{\gamma_k \alpha_k}} \right], \quad (7.20)$$

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 (7.13), (7.14) and (7.15), we observe that the contributing factors in the denominator are $(t - t_k)$ and $(t_{k+1} - t)$. When $t \rightarrow t_{k+1}-$, 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,

$$\lim_{t \rightarrow t_{k+1}-} f_{j,k}(t) = \lim_{t \rightarrow t_{k+1}-} \left[\frac{1}{b(t_{k+1} - t)^n} \right], \quad (7.21)$$

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

If we combine (7.20) with (7.21) we get

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

where $\gamma_k \alpha_k > 0$ (initial constraints). This expression is of the same type as (7.17) and, therefore, tends to zero when $t \rightarrow t_{k+1}-$.

If alternatively $\lambda_k = 1$ and $2\sigma_k < \gamma_k \alpha_k$, the limit when $t \rightarrow t_{k+1}-$ of the exponent in (7.8) can in this case be reformulated to

$$\lim_{t \rightarrow t_{k+1}-} \zeta(t) = \lim_{t \rightarrow t_{k+1}-} \left[-\frac{a}{(t_{k+1} - t)^{\gamma_k \alpha_k - 2\sigma_k}} \right], \quad (7.22)$$

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 (7.22) with (7.21) we get

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

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 (7.17) and therefore tends to zero when $t \rightarrow t_{k+1}-$.

This shows that if we use the additional restriction 3 in Theorem 7.1, then

$$\lim_{t \rightarrow t_{k+1}-} D^j B_{k+1}(t) = \lim_{t \rightarrow t_{k+1}-} f_{j,k}(t)\psi_k(t) = 0, \quad \text{for } j = 2, 3, \dots.$$

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

We have to show that $\lim_{t \rightarrow t_k+} f_{j,k}(t)\psi_k(t) = 0$. The question is if $\psi_k(t)$ goes “faster” towards 0 than the denominator in $f_{j,k}(t)$, for $j = 2, 3, \dots$ when $t \rightarrow t_k+$. 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 7.1). Suppose first that $\lambda_k > 0$, the limit when $t \rightarrow t_k+$ of the exponent in (7.8) (named $\zeta(t)$) can be reformulated to

$$\lim_{t \rightarrow t_k+} \zeta(t) = \lim_{t \rightarrow t_k+} \left[-\frac{a}{(t - t_k)^{\alpha_k}} \right], \quad (7.23)$$

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 (7.13), (7.14) and (7.15), we can clearly see that the contributing factors in the denominator are $(t - t_k)$ and $(t_{k+1} - t)$. When $t \rightarrow t_k+$, 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 $t \rightarrow t_k+$, of $f_{j,k}(t)$ can be reformulated to,

$$\lim_{t \rightarrow t_k+} f_{j,k}(t) = \lim_{t \rightarrow t_k+} \left[\frac{1}{b(t - t_k)^n} \right], \quad (7.24)$$

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

If we combine (7.23) with (7.24) we get

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

where $\alpha_k > 0$ (initial constraint). The expression is of the same type as (7.17), and therefore tends to zero when $t \rightarrow t_k+$.

If alternatively $\lambda_k = 1$ and $2\sigma_k < \gamma_k \alpha_k$, the limit when $t \rightarrow t_k+$ of the exponent in (7.8) (named $\zeta(t)$) can be reformulated to,

$$\lim_{t \rightarrow t_k+} \zeta(t) = \lim_{t \rightarrow t_k+} \left[-\frac{a}{(t_k - t)^{\gamma_k \alpha_k - 2\sigma_k}} \right], \quad (7.25)$$

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 (7.25) with (7.24) we get

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

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 (7.17) and, therefore, tends to zero when $t \rightarrow t_k+$.

This shows that if we use the additional restriction 2 in Theorem 7.1, then

$$\lim_{t \rightarrow t_k+} D^j B_k(t) = \lim_{t \rightarrow t_k+} f_{j,k}(t)\psi_k(t) = 0, \quad \text{for } j = 2, 3, \dots$$

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

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

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

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

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

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 (7.12) it follows that

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

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 (7.16) (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 (7.15), 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 (7.13), 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

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

Recall that when computing a derivative of an absolute value of the polygon, a $\text{sign}(c_k(t))$ 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 $\text{sign}(c_k(t))$. It follows that, when the exponent is zero, then $|c_k(t)|^0 = 1$. If $\text{sign}(c_k(t))$ 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, $\sigma = \frac{1}{2}, \frac{3}{2}, \frac{5}{2}, \dots$ can not be used.

The conclusion is as follows:

to guarantee $D^j B_k(t)$, for $j = 2, 3, \dots$, is in $C^\infty(t_k, t_{k+1})$ the additional restriction 1 in Theorem 7.1 must be fulfilled (in addition to restriction 2 and 3 shown by the other items).

- (l) $D^j B_{k+1}(t)$ for $j = 2, 3, \dots$ 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 7.1 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 7.1 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 7.1 is fulfilled, then **P6** is a property of ERBS because of:
item (c), item (d), item (g) and item (h).

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

This completes the proof of Theorem 7.1. □

7.5 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 7.2, expression (7.8).

Theorem 7.2. *Let the intrinsic values be restricted according to definition 7.2, and let an additional restriction be $2\sigma_k = (1 + \gamma_k)\alpha_k$, then, generally,*

$$\int_{t_k}^t \psi_k(s) ds = (t_{k+1} - t_k) \int_0^{\frac{t-t_k}{t_{k+1}-t_k}} e^{-\beta_k \frac{|s-\lambda_k|^{2\sigma_k}}{(s(1-s)^{\gamma_k})^{\alpha_k}}} ds, \quad \text{if } t_k < t \leq t_{k+1}. \quad (7.26)$$

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

$$\int_{t_k}^{t_{k+1}} \psi_k(t) dt = (t_{k+1} - t_k) \int_0^1 e^{-\beta_k \frac{|t-\lambda_k|^{2\sigma_k}}{(t(1-t)^{\gamma_k})^{\alpha_k}}} dt. \quad (7.27)$$

Proof. Relation (7.27) follows directly from (7.26). To prove (7.26) it is only necessary to prove that the exponent from (7.8),

$$-\beta_k \frac{|t - ((1 - \lambda_k)t_k + \lambda_k t_{k+1})|^{2\sigma_k}}{((t - t_k)(t_{k+1} - t)^{\gamma_k})^{\alpha_k}}, \quad (7.28)$$

is preserved during translation and scaling of the domain and thus t, t_k, t_{k+1} . Translation with $-t_k$ changes (7.28) to

$$-\beta_k \frac{|\bar{t} - \lambda_k(t_{k+1} - t_k)|^{2\sigma_k}}{(\bar{t}(t_{k+1} - t_k - \bar{t})^{\gamma_k})^{\alpha_k}}, \quad (7.29)$$

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, $(\frac{1}{t_{k+1}-t_k})^{(1+\gamma_k)\alpha_k}$, (7.29) remains unchanged, and we get,

$$-\beta_k \frac{(\frac{1}{t_{k+1}-t_k})^{(1+\gamma_k)\alpha_k} |\bar{t} - \lambda_k(t_{k+1} - t_k)|^{2\sigma_k}}{(\frac{1}{t_{k+1}-t_k})^{(1+\gamma_k)\alpha_k} (\bar{t}(t_{k+1} - t_k - \bar{t})^{\gamma_k})^{\alpha_k}}. \quad (7.30)$$

It follows that if $2\sigma_k = (1 + \gamma_k) \alpha_k$ then (7.30) is the same as

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

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

The property in Theorem 7.2 leads us to a new definition. Assume the additional restriction in Theorem 7.2. 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 7.2 that both the “general” integral (7.26) and the “particular” integral (7.27) is scaled by the constant $(t_{k+1} - t_k)$. In the definition below this scaling is initially in both the numerator ϕ and the denominator of S and is therefore canceled.

Definition 7.3. *The scalable subset \mathfrak{B} of Expo-Rational B-splines contains elements defined by $B_k(t) = B_k(t; \alpha_{k-1}, \beta_{k-1}, \gamma_{k-1}, \lambda_{k-1}, \alpha_k, \beta_k, \gamma_k, \lambda_k)$, as follows,*

$$B_k(t) = \begin{cases} S(\alpha_{k-1}, \beta_{k-1}, \gamma_{k-1}, \lambda_{k-1}) \int_0^{w_{k-1}(t)} \phi(s; \alpha_{k-1}, \beta_{k-1}, \gamma_{k-1}, \lambda_{k-1}) ds, & \text{if } t_{k-1} < t \leq t_k, \\ S(\alpha_k, \beta_k, \gamma_k, \lambda_k) \int_{w_k(t)}^1 \phi(s; \alpha_k, \beta_k, \gamma_k, \lambda_k) ds, & \text{if } t_k < t < t_{k+1}, \\ 0, & \text{otherwise} \end{cases} \quad (7.31)$$

where

$$w_k(t) = \frac{t - t_k}{t_{k+1} - t_k}, \quad (7.32)$$

and

$$\phi(t; \alpha, \beta, \gamma, \lambda) = e^{-\beta \frac{|t - \lambda|^{(1+\gamma)\alpha}}{(t(1-t))^\alpha}}, \quad (7.33)$$

and the scaling factor is

$$S(\alpha, \beta, \gamma, \lambda) = \frac{1}{\int_0^1 \phi(t; \alpha, \beta, \gamma, \lambda) dt}, \quad (7.34)$$

where

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

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

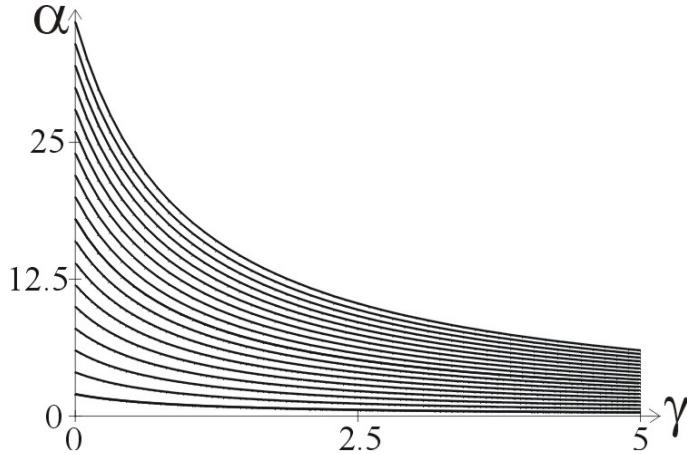


Figure 7.5: The graph shows the equation $(1 + \gamma)\alpha = 2n$, for $n = \{1, 2, \dots, 20\}$. $B_k(t)$ is $C^\infty(t_{k-1}, t_{k+1})$ when using the intrinsic parameters (γ, α) describing the curves in the plot, if $0 < \lambda < 1$.

derivative for the scalable subset is

$$DB_k(t) = \begin{cases} \frac{S_{k-1}}{t_k - t_{k-1}} \phi_{k-1} \circ w_{k-1}(t), & \text{if } t_{k-1} < t \leq t_k, \\ -\frac{S_k}{t_{k+1} - t_k} \phi_k \circ w_k(t), & \text{if } t_k < t < t_{k+1}, \\ 0, & \text{otherwise.} \end{cases} \quad (7.35)$$

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

$$D^j B_k(t) = \begin{cases} \left(\frac{1}{t_k - t_{k-1}}\right)^j f_{j,k-1}(t) \phi_{k-1} \circ w_{k-1}(t), & \text{if } t_{k-1} < t \leq t_k, \\ -\left(\frac{1}{t_{k+1} - t_k}\right)^j f_{j,k}(t) \phi_k \circ w_k(t), & \text{if } t_k < t < t_{k+1}, \\ 0, & \text{otherwise,} \end{cases} \quad (7.36)$$

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 7.1 for the scalable subset (and thus also that (7.35) and (7.36) is in $C^\infty(t_{k-1}, t_{k+1})$), we have to look at the additional restrictions on the constraints on the intrinsic parameters. If, according to the restriction on the intrinsic parameters in Theorem 7.2, we replace σ with $\frac{(1+\gamma)\alpha}{2}$, the three additional constraints in Theorem 7.1 will also have to be changed. It follows that if we do this replacement, the first additional restriction in Theorem 7.1 will be,

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

In Figure 7.5 there are 20 graphs of the equation $(1 + \gamma)\alpha = 2n$, for $n = 1, 2, \dots, 20$. What we can see is the α and γ values for which $B_k(t)$ is C^∞ -smooth on \mathbb{R} . For the second additional restriction, when we plug $\sigma = \frac{(1+\gamma)\alpha}{2}$, we get that

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

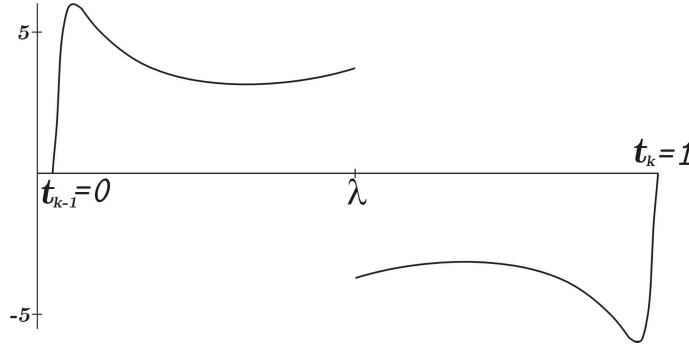


Figure 7.6: The second derivative $D^2B_k(t)$ where the intrinsic parameters are $\alpha = 0.5$, $\beta = 1$, $\gamma = 1$, $\lambda = 0.5$. Notice the discontinuity at $t = \lambda$.

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

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

This is of course a contradiction. To fulfill all the properties in Theorem 7.1 we, therefore, get the following set of restrictions on the intrinsic parameters replacing the three items in Theorem 7.1 for the scalable subset.

1. $\frac{(1+\gamma)\alpha}{2} \in \mathbb{N}$, for property 4 in Theorem 7.1,
2. $\lambda > 0$, for property 4 and 5 in Theorem 7.1,
3. $\lambda < 1$, for property 4, 5 and 6 in Theorem 7.1,

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.

We first consider the exponent in (7.33)

$$\zeta(\alpha, \beta, \gamma, \lambda; t) = -\beta \frac{|t - \lambda|^{(1+\gamma)\alpha}}{(t(1-t)^\gamma)^\alpha}, \quad (7.37)$$

so that (7.33) can be rewritten as

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

The derivative of ϕ is thus

$$\phi'_k(t) = \zeta'_k(t) \phi_k(t). \quad (7.38)$$

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

$$\zeta'_k(t) = x_{2,k}(t) \zeta_k(t), \quad (7.39)$$

where

$$x_{2,k}(t) = \alpha_k \left(\frac{t(1+\gamma_k) - 1}{t(1-t)} + \frac{\gamma_k + 1}{t - \lambda_k} \right). \quad (7.40)$$

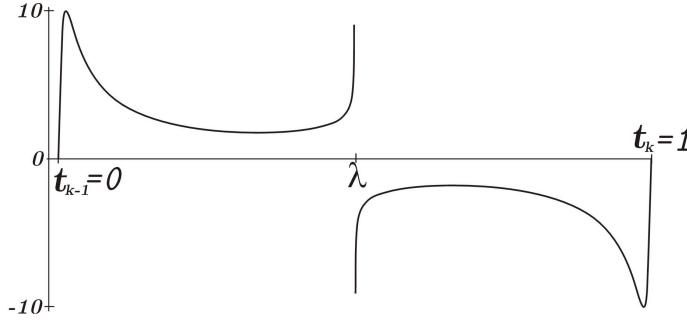


Figure 7.7: The second derivative $D^2B_k(t)$ where the intrinsic parameters are $\alpha = 0.4$, $\beta = 1$, $\gamma = 1$, $\lambda = 0.5$. Notice the asymptotic behavior at $t = \lambda$.

We can see that $x_{2,k}(t)$ in (7.40), and thus $\zeta'_k(t)$ in (7.39), 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 $t \rightarrow t_k$ from both left and right, we will see that

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

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 7.6). If $(1 + \gamma_k)\alpha_k < 1$ there is an asymptotic behavior at $t = \lambda_k$ (illustrated in Figure 7.7). But if we restrict the intrinsic parameters α and γ to

$$(1 + \gamma)\alpha > 1, \quad (7.41)$$

we get a continuous function

$$f_{2,k}(t) = \begin{cases} 0, & \text{if } t = \lambda_k, \\ S_k x_{2,k}(t) \zeta_k(t), & \text{otherwise.} \end{cases} \quad (7.42)$$

Computing the derivative of $x_{2,k}(t)$ (7.40) 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 ϕ , and thus the derivative of (7.38), we get

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

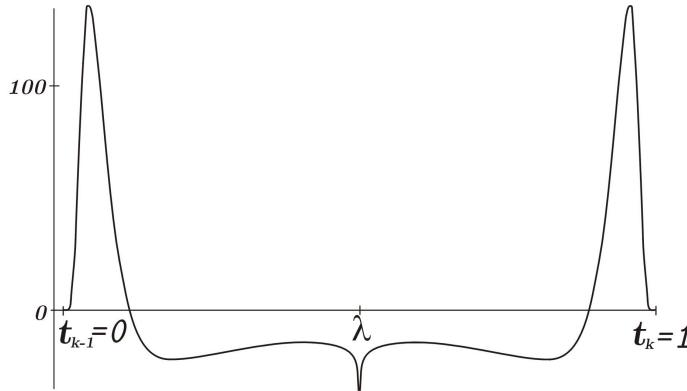


Figure 7.8: The third derivative $D^3 B_k(t)$ where the intrinsic parameters are $\alpha = 0.9$, $\beta = 1$, $\gamma = 1$, $\lambda = 0.5$. At $t = \lambda$ there is a very steep asymptote (here truncated).

This can be reorganized into the following formula,

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

where

$$\begin{aligned} x_{3,k}(t) &= \alpha_k \left(\frac{t^2(\gamma_k + 1) + 1 - 2t}{(t(t-1))^2} - \frac{\gamma_k + 1}{(t - \lambda_k)^2} + \right. \\ &\quad \left. \alpha_k \left(\frac{(t(\gamma_k - \lambda_k - \lambda_k \gamma_k) + \lambda_k)^2}{(t(1-t)(t - \lambda_k))^2} \right) \left(1 - \beta_k \frac{|t - \lambda_k|^{(1+\gamma_k)\alpha_k}}{(t(1-t)^{\gamma_k})^{\alpha_k}} \right) \right). \end{aligned} \quad (7.43)$$

Looking at $x_{3,k}(t)$ (7.43), 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 (7.41) is fulfilled. Multiplying the rest by $\zeta_k(t)$ gives

$$\left(\gamma_k + 1 - \alpha_k \left(\frac{t(\gamma_k - \lambda_k - \lambda_k \gamma_k) + \lambda_k}{t(1-t)} \right)^2 \right) \frac{\alpha_k \beta_k |t - \lambda_k|^{(1+\gamma_k)\alpha_k - 2}}{(t(1-t)^{\gamma_k})^{\alpha_k}}, \quad (7.44)$$

which, if $(1 + \gamma)\alpha < 2$, is obviously not defined at $t = \lambda_k$. Two figures shows examples of asymptotes. In Figure 7.8 $\alpha = 0.9$ and thus $(1 + \gamma)\alpha = 1.8$, and in Figure 7.9 $\gamma = 0.8$ 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 (7.44) (except for $|t - \lambda_k|$), we can see that

$$\lim_{t \rightarrow \lambda_k \pm} x_{3,k}(t) \zeta_k(t) = \lim_{t \rightarrow \lambda_k \pm} -\frac{\beta_k \alpha_k (\gamma_k + 1) (\alpha_k (1 + \gamma_k) - 1)}{(\lambda_k (1 - \lambda_k)^{\gamma_k})^{\alpha_k}} |t - \lambda_k|^{(1+\gamma_k)\alpha_k - 2}. \quad (7.45)$$

If we then restrict the intrinsic parameters α and γ to

$$(1 + \gamma)\alpha \geq 2, \quad (7.46)$$

we get a continuous function

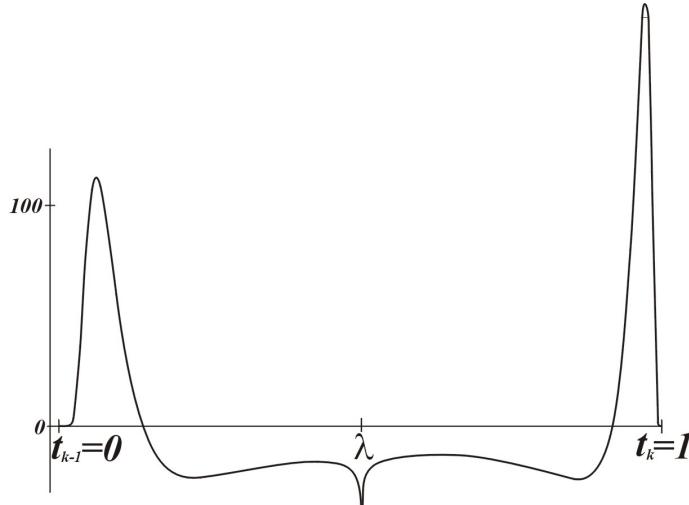


Figure 7.9: The third derivative $D^3 B_k(t)$ where the intrinsic parameters are $\alpha = 1$, $\beta = 1$, $\gamma = 0.8$, $\lambda = 0.5$. At $t = \lambda$ there is a very steep asymptote (here truncated).

$$f_{3,k}(t) = \begin{cases} 0, & \text{if } t = \lambda_k \text{ end } (1 + \gamma_k) \alpha_k > 2, \\ \frac{-2\beta_k}{\lambda_k^{\alpha_k} (1 - \lambda_k)^{2 - \alpha_k}}, & \text{if } t = \lambda_k \text{ end } (1 + \gamma_k) \alpha_k = 2, \\ S_k x_{3,k}(t) \zeta_k(t), & \text{otherwise.} \end{cases} \quad (7.47)$$

In section 7.10 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.

7.6 The default set of intrinsic parameters

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

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

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^∞ -smooth. Using these intrinsic values, we get a simpler definition.

Definition 7.4. *The default set of Expo-Rational B-splines is as follows,*

$$B_k(t) = \begin{cases} S_d \int_0^{w_{k-1}(t)} \phi(s) ds, & \text{if } t_{k-1} < t \leq t_k, \\ S_d \int_0^1 \phi(s) ds, & \text{if } t_k < t < t_{k+1}, \\ 0, & \text{otherwise} \end{cases}$$

where

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

and

$$\phi(t) = e^{-\frac{(t-\frac{1}{2})^2}{t(1-t)}}, \quad (7.48)$$

and where the scaling factor is

$$S_d = \left[\int_0^1 \phi(t) dt \right]^{-1} \approx 1.6571376797382103. \quad (7.49)$$

In Figure 7.4, there is a graph of $B_k(t)$ and its first derivative, 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 (7.35), 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 7.4 the two knot intervals have both length 1.

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

$$\begin{aligned} f_{1d}(t) &= S_d, \\ f_{2d}(t) &= S_d \frac{(1-2t)}{(2t(1-t))^2}, \\ f_{3d}(t) &= S_d \frac{\frac{3}{2}(1-2t)^4 - \frac{1}{2}}{(2t(1-t))^4}, \\ f_{4d}(t) &= S_d \frac{(3(1-2t)^6 + \frac{3}{2}(1-2t)^4 - 5(1-2t)^2 + \frac{3}{2})(1-2t)}{(2t(1-t))^6}, \\ f_{5d}(t) &= S_d \frac{\frac{15}{2}(1-2t)^{10} + \frac{45}{4}(1-2t)^8 - 33(1-2t)^6 + \frac{29}{2}(1-2t)^4 + \frac{3}{2}(1-2t)^2 - \frac{3}{4}}{(2t(1-t))^8}, \\ f_{6d}(t) &= S_d \frac{\left(\frac{45}{2}(1-2t)^{12} + \frac{135}{2}(1-2t)^{10} - \frac{765}{4}(1-2t)^8 + 75(1-2t)^6 + 66(1-2t)^4 - \frac{85}{2}(1-2t)^2 + \frac{15}{4}\right)(1-2t)}{(2t(1-t))^{10}}, \\ f_{7d}(t) &= S_d \frac{\frac{315}{4}(1-2t)^{16} + \frac{1575}{4}(-)^{14} - \frac{8505}{8}(-)^{12} + \frac{465}{4}(-)^{10} + \frac{9645}{8}(-)^8 - \frac{3551}{4}(-)^6 + \frac{1005}{8}(-)^4 + \frac{135}{4}(-)^2 - \frac{15}{8}}{(2t(1-t))^{12}}. \end{aligned} \quad (7.50)$$

The expressions in (7.50) 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 7.1, and fits with the proof of the same theorem, and thus guarantees that the continuity is not destroyed by a change of sign.

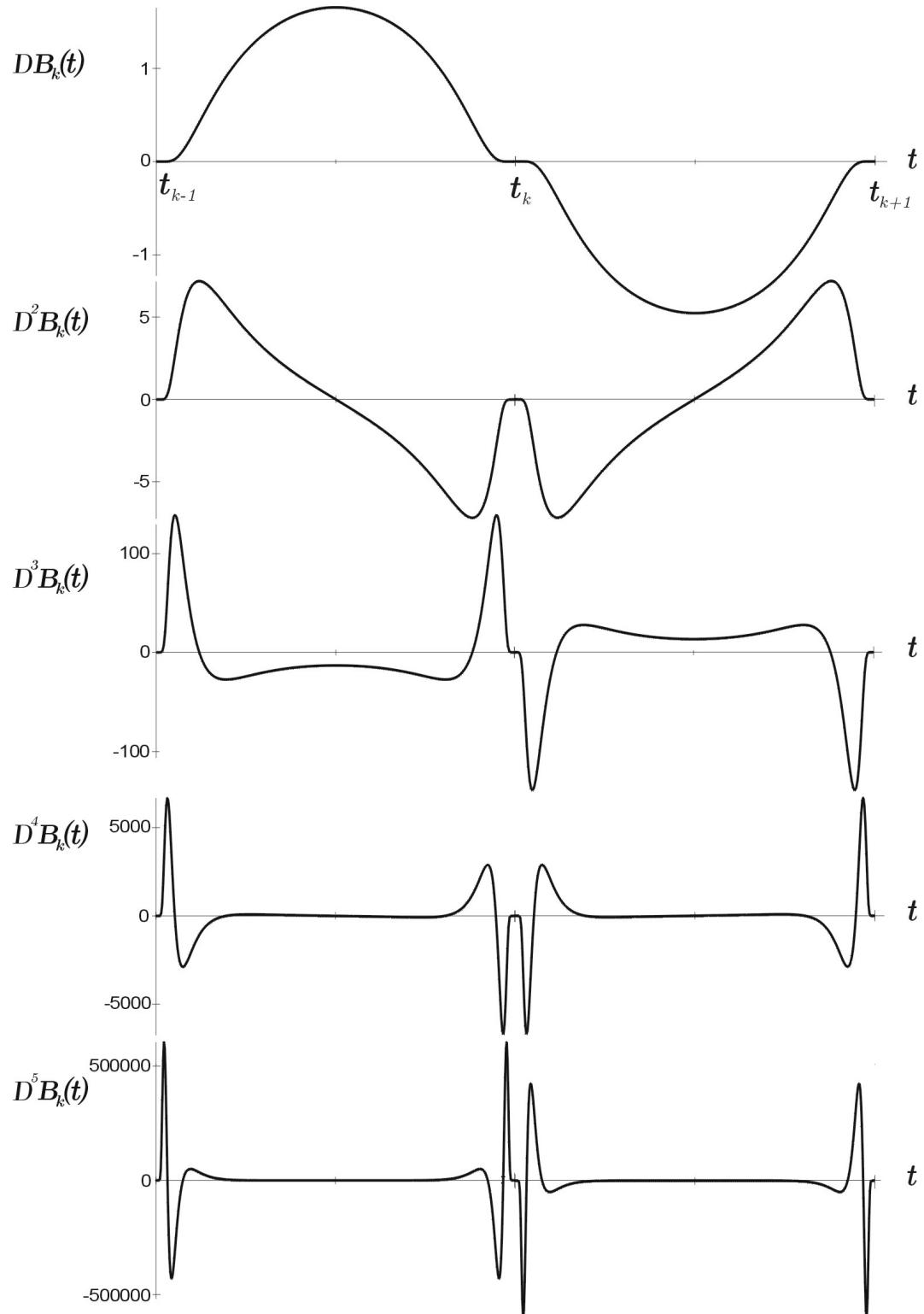


Figure 7.10: A graph of the first, second, third, fourth and fifth derivatives of $B_k(t)$ when the default values of the intrinsic parameters are used. The knots t_{k-1} , t_k and t_{k+1} are also given on the graph, and the function values in this example correspond to $t_{k+1} - t_k = t_k - t_{k-1} = 1$. 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.

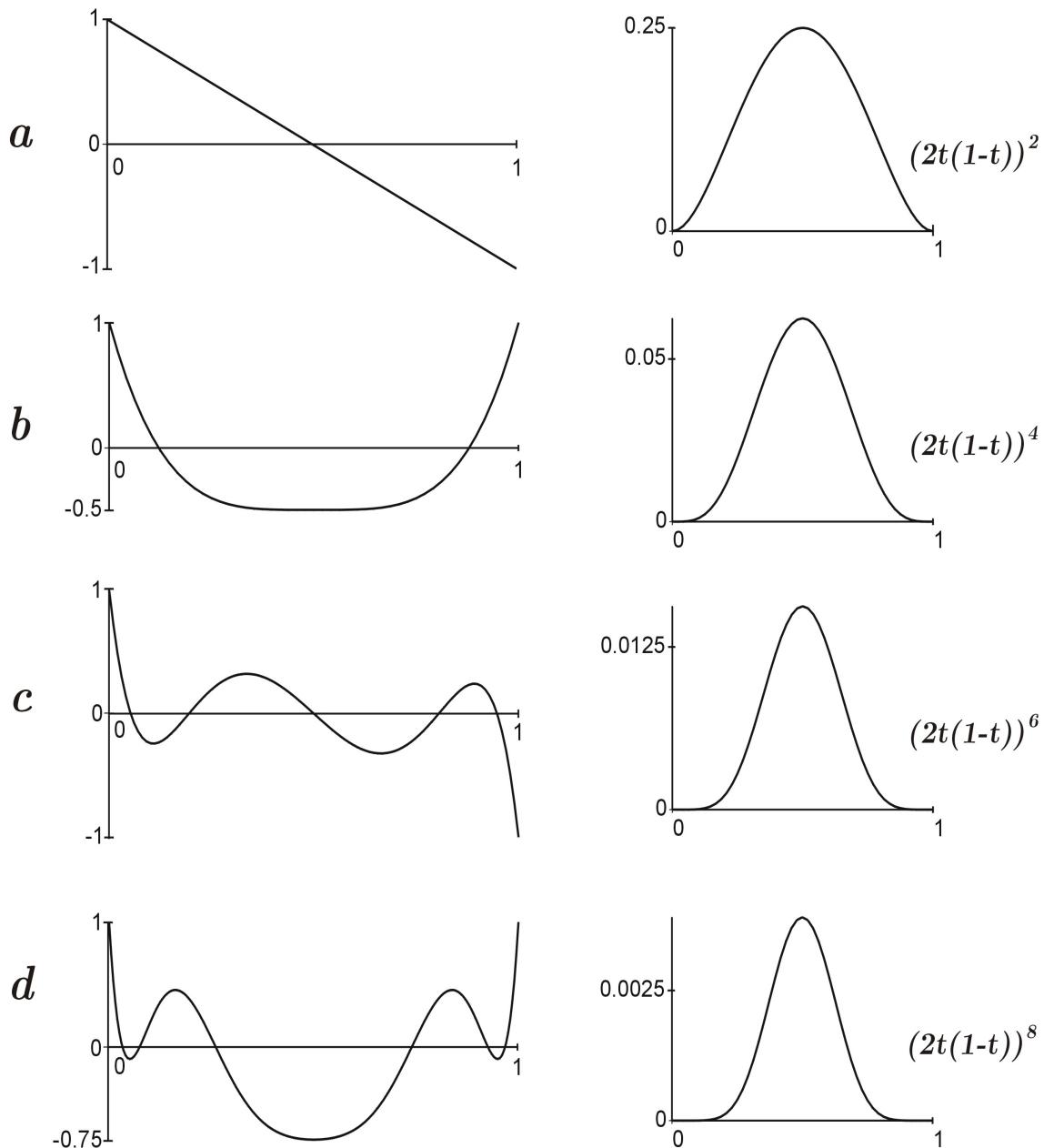


Figure 7.11: A graph of the numerator (on the left hand side) and the denominator (on the right hand side) of the fraction in $f_{j,k}(t)$, $j = 2, 3, 4, 5$ (in the second, third, fourth and fifth derivatives of $B_k(t)$). The default set of the intrinsic parameters is used. The expressions for the four graphs on the left hand side are

- a** $(1 - 2t),$
- b** $\frac{3}{2}(1 - 2t)^4 - \frac{1}{2},$
- c** $(3(1 - 2t)^6 + \frac{3}{2}(1 - 2t)^4 - 5(1 - 2t)^2 + \frac{3}{2})(1 - 2t),$
- 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 progressively 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.

In Figure 7.10, 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 7.1, 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 7.11, 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 $(\frac{1}{2})^8 \approx 0.0039$. 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 starts in 1 and ends in 1 for odd j values (provided that the denominator is expanded in the same way as in Figure 7.11 and in expression (7.50)).

7.7 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 7.1 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 7.12, and in function (7.52) below. We, therefore, will have the following function inside a knot interval,

$$f(t) = c_i B_i(t) + c_{i+1} B_{i+1}(t), \quad \text{if } t_i < t < t_{i+1}. \quad (7.51)$$

Recall the basic property 2 in Theorem 7.1. It follows that (7.51) can be reformulated to

$$f(t) = c_{i+1} + (c_i - c_{i+1}) B_i(t), \quad \text{if } t_i < t < t_{i+1}. \quad (7.52)$$

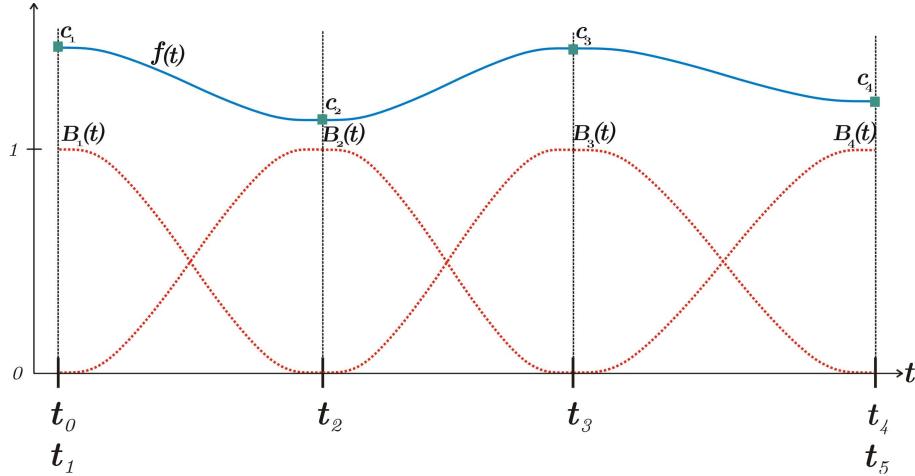


Figure 7.12: A “global” Expo-Rational B-spline function $f(t)$ (blue) with four scalar coefficients $\{c_i\}_{i=1}^4$ (green). The global function is a blending of the coefficients, with the Expo-Rational B-splines $\{B_i(t)\}_{i=1}^4$ (red) being the blending functions. The global function interpolates 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 7.8).

Secondly, we shall consider the general case of non-constant (scalars, vectors or points) local functions as coefficients (illustrated in Figure 7.13). The domain of these functions has to be the same as the domain of the respective ERBS basis function. It means that the 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 7.7 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 7.1.

Definition 7.5. To classify local functions $l_k(t)$, we define a set of function spaces,

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

where the intrinsic parameters on the segments (t_{k-1}, t_k) and (t_k, t_{k+1}) support property 5 described in Theorem 7.1.

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 7.6. 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

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

where $l_k(t)$ are local (scalar or vector-valued) functions defined on (t_{k-1}, t_{k+1}) , $k =$

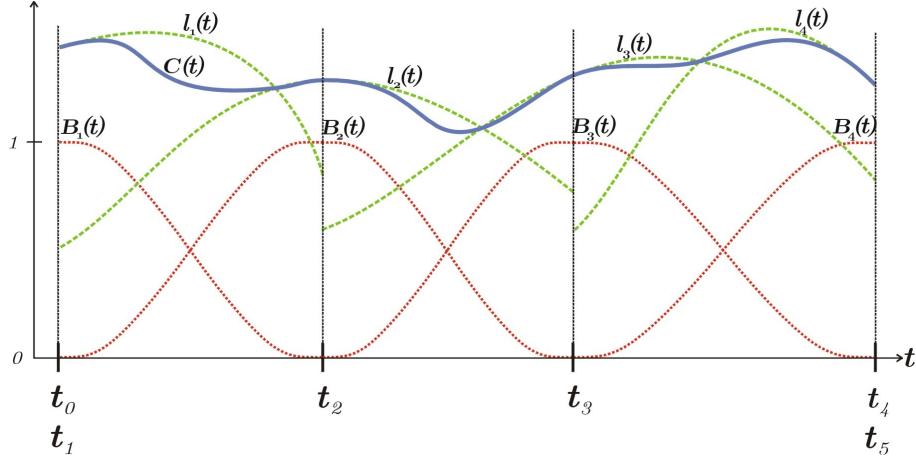


Figure 7.13: A “global” Expo-Rational B-spline function $f(t)$ (blue) with four local functions $\{l_i(t)\}_{i=1}^4$ (green). The global function is a blending of its local functions, with the Expo-Rational B-splines $\{B_i(t)\}_{i=1}^4$ (red) being the blending functions. The global function also completely 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 7.8).

1, ..., n , and $B_k(t)$, $k = 1, \dots, n$ is defined in definition 7.2. 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 7.9). Figure 7.13 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 7.7. We denote the local domain for the local function l_k to be,

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

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 ω_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 do-

main I_k ,

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

ω_k is, therefore, the affine mapping

$$\omega_k(t) = s_{k0} + \frac{t - t_{k-1}}{t_{k+1} - t_{k-1}}(s_{k1} - s_{k0}). \quad (7.53)$$

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

$$\omega_k^{-1}(s) = t_{k-1} + \frac{s - s_{k0}}{s_{k1} - s_{k0}}(t_{k+1} - t_{k-1}). \quad (7.54)$$

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 discovered.

7.8 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 B-spline functions, but here the knot vector of ERBS is only determining if the function is C^∞ or if it is discontinuous at a knot. When the knot is multiple ($t_k = t_{k+1}$), it can be seen that Definition 7.2 and Figure 7.4 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 7.6 depends on the local curve, as shown in the following Theorem.

Theorem 7.3. *Provided that the intrinsic parameters are restricted so that all properties in Theorem 7.1 are fulfilled, an Expo-Rational B-spline function f belongs to $C^\infty(t_1, t_n]$ if, for $k = 1, \dots, n-1$, the knot 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 $t_k < t_{k+1}$, $k = 1, \dots, n-1$, then f is C^∞ -smooth because of properties 3,4,5 and definition 7.6. If $t_k = t_{k+1}$ then because of property 3

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

and

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

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. \square

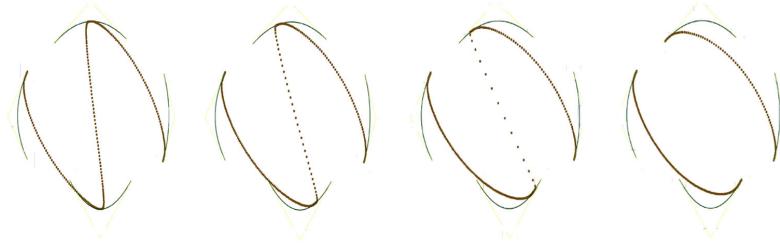


Figure 7.14: Four versions of a curve. In the left figure the knots are uniformly distributed except for the multiple knots at each end. In curve two from the left, the distance between the two middle simple knots are half of the original size. In the next curve the size is halved once more, and in the 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).

Figure 7.14, 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.

7.9 Hermite Interpolation properties

Here we shall see that, under the general assumptions that the intrinsic parameters fulfill all properties in Theorem 7.1 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

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

which is property 6 in Theorem 7.1. Let us now increase the generalization to a higher dimensional value of the function,

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

and let us also consider the definition of the affine global/local mapping ω_i from definition 7.7, 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 7.4. *Let the sequence of functions $c_i : [s_{i0}, s_{i1}] \subset \mathbb{R} \rightarrow \mathbb{R}^d$ where $1 \leq d < \infty$, be $\in \mathfrak{F}(B_k)$. Let*

$$f(t) = \sum_{i=1}^n c_i \circ \omega_i(t) B_i(t) \tag{7.55}$$

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

by

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

If the properties in Theorem 7.1 are fulfilled, then

$$D^j f(t_i) = \delta_i^j D^j c_i \circ \omega_i(t_i), \quad \text{for } j = 0, 1, 2, \dots \text{ and } i = 1, \dots, n. \quad (7.56)$$

Proof. Deriving (7.55) according to t we get

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

Recalling that $B_i(t_i) = 1$ (property 3 in Theorem 7.1), $B_i(t_j) = 0$ 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 (7.56) follows. Which completes the proof. \square

Remark 7. 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 $i = 1, \dots, n$, the scaling factor δ_i^j for $i = 1, \dots, n$ and for $j = 1, 2, \dots$ 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 $t_i = x_i$, $i = 1, \dots, n$ 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,

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

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

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

and we use monomial basis functions in the Taylor series,

$$c_{j,i}(t) = f(t_i) + f'(t_i)(t - t_i) + \frac{f''(t_i)}{2!}(t - t_i)^2 + \frac{f'''(t_i)}{3!}(t - t_i)^3 + \dots, \quad (7.57)$$

for $i = 1, 2, \dots, n$, as local functions. The index j is denoting the number of derivatives used, i.e. the number of terms in the equation (7.57) is $j+1$. It follows that the global/local scaling factors, connected to the respective local functions, are $\delta_i = 1$.

Table 7.1 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

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

where

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

are the ERBS functions. In Table 7.1 we can see $\varepsilon_{n,j}$, for $n = 5, 10, 20, 40, 80, 160$ and $j = 1, 2, 3, 4$.

In Figure 7.15 the “errors” shown in Table 7.1 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 book.

7.10 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 7.3). The basis function $B_k(t)$ (see (7.31),

number of knots	number of derivatives used			
	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 7.1: 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.

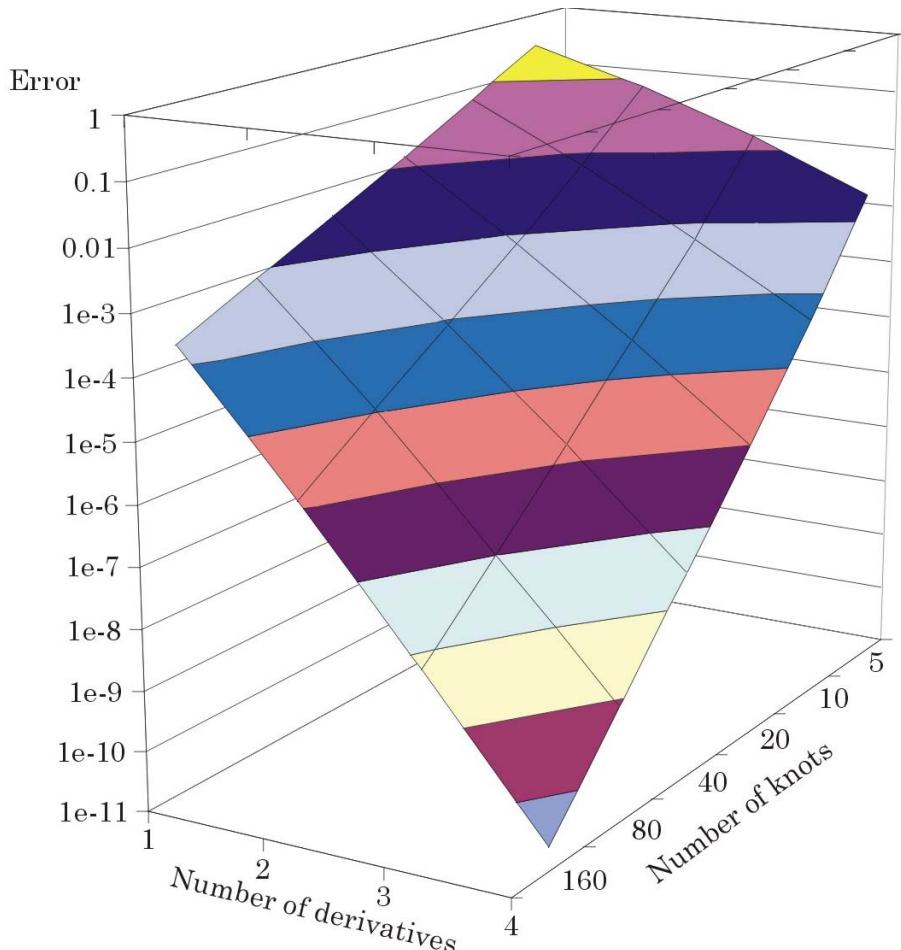


Figure 7.15: The surface shows the error (maximum deviation between the ERBS function and the sine function). Notice that the vertical axis uses a logarithmic scale. The horizontal axes show the number of derivatives and the number of knots, where the scale of the number of knots is logarithmic with 2 as the base. The numbers used to construct the surface can be found in table 7.1.

(7.32), (7.33) and (7.34)) 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 of this section the basis function will be denoted

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

We will see four figures (7.16, 7.17, 7.18, 7.19), 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
7.16	α	0.01	0.2	1	2	6
7.17	β	0.01	0.2	1	6	100
7.18	γ	0.001	0.1	1	2	10
7.19	λ	0.01	0.3	0.5	0.7	0.99

In Figure 7.16, all intrinsic parameters have the default values except for α . 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

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

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

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

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 7.16 are not in $C^\infty(0, 1)$.

In Figure 7.17, all intrinsic parameters have the default values except for β . 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

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

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

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

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 β obeys the restriction in definition

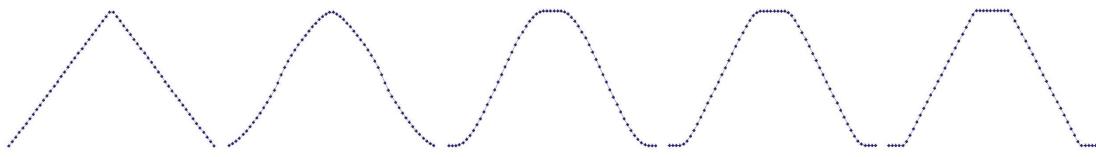


Figure 7.16: Five versions of $B(\alpha, \beta, \gamma, \lambda; t)$ where all intrinsic parameters have the default values, except for α . The domains for each of the five functions is $[0, 1]$. From left to right $\alpha = \{0.01, 0.2, 1 \text{ (default value)}, 2, 6\}$.

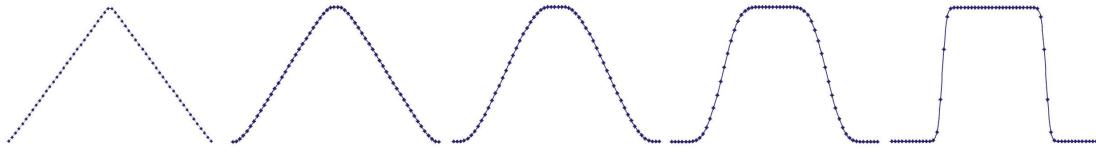


Figure 7.17: Five versions of $B(\alpha, \beta, \gamma, \lambda; t)$ where all intrinsic parameters have the default values, except for β . The domains for each of the five functions is $[0, 1]$. From left to right $\beta = \{0.01, 0.2, 1 \text{ (default value)}, 6, 100\}$.

7.2 which, of course, is the case for all functions in Figure 7.17. Notice also that variation of these two parameters, α and β preserves the symmetry, while varying of the other two parameters γ and λ does not preserve the symmetry.

In Figure 7.18, all intrinsic parameters have the default values, except for γ . 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

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

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 7.18, are not in $C^\infty(0, 1)$.

In Figure 7.19, all intrinsic parameters have the default values, except for λ . This param-

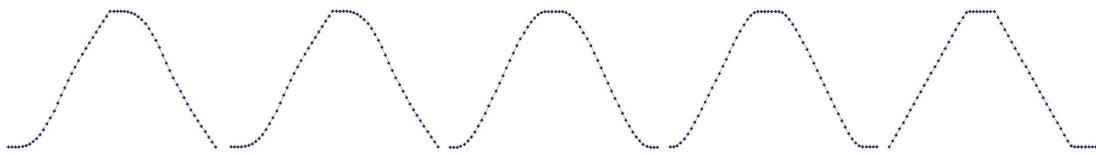


Figure 7.18: Five versions of $B(\alpha, \beta, \gamma, \lambda; t)$ where all intrinsic parameters have the default values, except for γ . The domains for each of the five functions is $[0, 1]$. From left to right $\gamma = \{0.001, 0.1, 1 \text{ (default value)}, 2, 10\}$. Notice that the difference between $\gamma = 0.1$ and $\gamma = 0.001$ is small

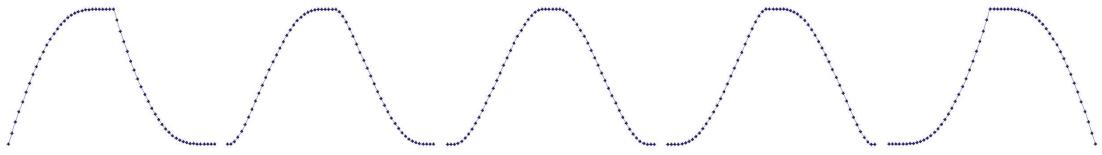


Figure 7.19: Five versions of $B(\alpha, \beta, \gamma, \lambda; t)$ where all intrinsic parameters have the default values, except for λ . The domains for each of the five functions is $[0, 1]$. From left to right $\lambda = \{0.01, 0.2, 0.5 \text{ (default value)}, 0.7, 0.99\}$. One can clearly see that the two functions on the left hand side is mirroring the two functions on the right hand side, i.e., mirror around $\lambda = 0.5$.

eter, 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 7.19.

Remark 8. Note that the four figures (7.16, 7.17, 7.18, 7.19) 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 8

GERBS – Curves

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

$$f(t) = \sum_{i=1}^n c_i(t) B_i(t), \quad (8.1)$$

where $c_i(t)$, $i = 1, \dots, n$, are local curves and $B_i(t)$, $i = 1, \dots, n$, 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 8.1 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 8.1 can be divided into three parts, and the “dividing points” are the 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 7.1) implies that*

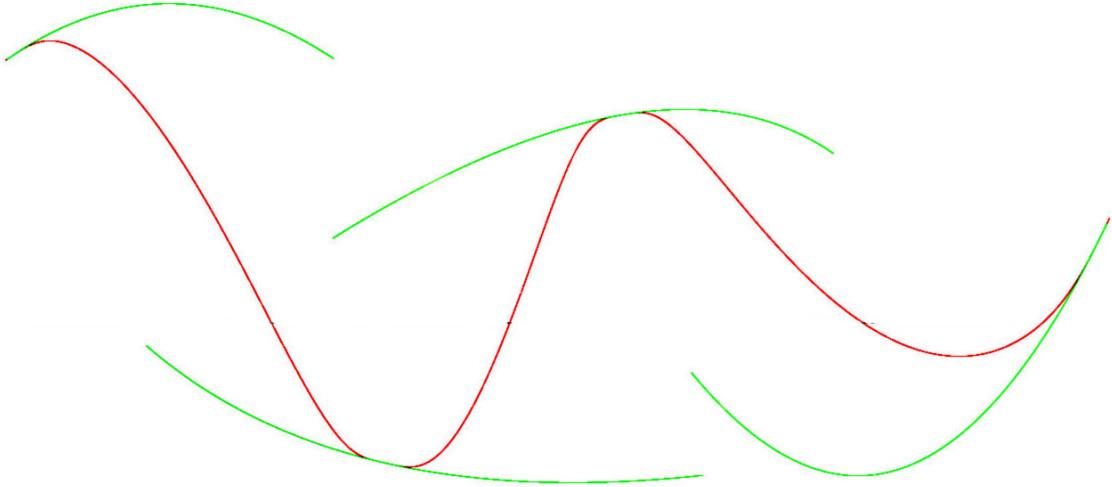


Figure 8.1: A (global) Expo-Rational B-spline curve (red) with four local curves (green). The global curve is a blending of its local curves, with the Expo-Rational B-splines being the blending functions. The global curve also completely interpolates all existing derivatives of each of the adjacent local curves at the “middle” knot.



Figure 8.2: 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.

all derivatives of f must be zero at every knot: $D^j f(t_i) = 0$, $j = 1, 2, \dots$, $i = 1, \dots, n$ (this is illustrated on the right hand side in Figure 8.2). 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 8.2).

In definition 7.5 the classification of local function to support the basic properties, i.e. the C^∞ 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 7.5).

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 8.3, 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

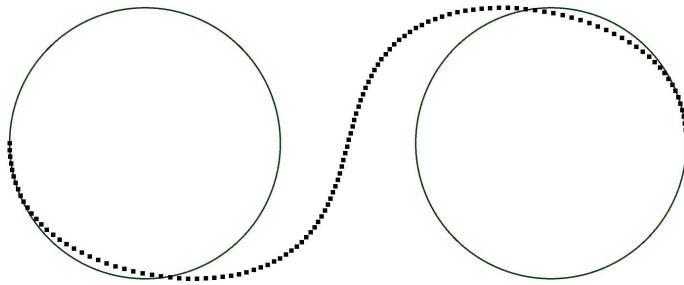


Figure 8.3: Two circles as local curves. The complete curve is the dotted curve. At the ends, the global curve interpolates the local curves with the derivatives of every order.

obvious, and will, therefore, be discussed in the first of the following sections.

In the remainder of this chapter the following subjects will be discussed. In section 8.2, the ERBS curve evaluation, including derivatives, will be discussed. Then, in section 8.3 Bézier curves as local curves will be investigated, including Hermite interpolation, and numerous examples. In section 8.4 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.

8.1 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 8.1. A parametrized differentiable curve is a differentiable map $\alpha : I \rightarrow \mathbb{R}^n$ of an open interval $I = (s, e) \subset \mathbb{R}$ into \mathbb{R}^n for $n = 2, 3, \dots$.

With this as a background the next definitions are introduced:

Definition 8.2. 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 7.2, an “open” ERBS curve α 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 α is apparently also defined on the half open interval $(a, b]$. But because a and b are defined to be identical, and $\lim_{t \rightarrow a+} D^j \alpha(t) = D^j \alpha(b)$, $j = 0, 1, 2, \dots$ 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 7.6, 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.

8.2 Evaluation, value and derivatives

The general equation for an ERBS curve (8.1) and its derivatives are straightforward to compute,

$$\begin{aligned} f(t) &= \sum_{i=1}^n c_i(t) B_i(t), \\ Df(t) &= \sum_{i=1}^n (Dc_i(t) B_i(t) + c_i(t) DB_i(t)), \\ D^2f(t) &= \sum_{i=1}^n (D^2c_i(t) B_i(t) + 2Dc_i(t) DB_i(t) + c_i(t) D^2B_i(t)), \\ D^3f(t) &= \sum_{i=1}^n (D^3c_i(t) B_i(t) + 3D^2c_i(t) DB_i(t) + 3Dc_i(t) D^2B_i(t) + c_i(t) D^3B_i(t)). \end{aligned} \quad (8.2)$$

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. Analogously to equation (7.52) we can simplify the first part of (8.2) to

$$f(t) = \begin{cases} c_k(t), & \text{if } t = t_k \\ c_{k+1}(t) + (c_k(t) - c_{k+1}(t)) B_k(t), & \text{if } t_k < t < t_{k+1} \end{cases} \quad (8.3)$$

Computing the derivatives of (8.3) we can see that at $t = t_k$, $k = 1, \dots, n$, all derivatives of the ERBS curve are equal to the respective derivatives of the local curve, i.e.

$$D^j f(t_k) = D^j c_k(t_k), \quad \text{for } k = 1, \dots, n \quad \text{and } j = 0, 1, 2, \dots \quad (8.4)$$

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

$$\hat{c}_k(t) = c_k(t) - c_{k+1}(t), \quad \text{if } t_k < t < t_{k+1}, \quad (8.5)$$

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

$$\begin{aligned} f(t) &= c_{k+1}(t) + \hat{c}_k(t)B_k(t), \\ Df(t) &= Dc_{k+1}(t) + \hat{c}_k(t)DB_k(t) + D\hat{c}_k(t)B_k(t), \\ D^2f(t) &= D^2c_{k+1}(t) + \hat{c}_k(t)D^2B_k(t) + 2D\hat{c}_k(t)DB_k(t) + D^2\hat{c}_k(t)B_k(t), \\ D^3f(t) &= D^3c_{k+1}(t) + \hat{c}_k(t)D^3B_k(t) + 3D\hat{c}_k(t)D^2B_k(t) + 3D^2\hat{c}_k(t)DB_k(t) + D^3\hat{c}_k(t)B_k(t) \end{aligned} \quad (8.6)$$

These expressions are similar to (8.2), but there is an extra term $D^j c_{k+1}(t)$ for $j = 0, 1, 2, 3, \dots$, and there is $\hat{c}_k(t)$ 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 $D^j B_k(t)$, $j = 1, 2, \dots$, i.e. algorithm 9, algorithm 12 and algorithm 16, 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 9, algorithm 12 and algorithm 16 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) $D^j B_k(t) = -D^j B_{k+1}(t)$ for $t_k < t < t_{k+1}$ and $j = 1, 2, \dots$

Both i) and ii) follow directly from property 2 in Theorem 7.1.

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

Algorithm 3. (For notation, see section “Algorithmic Language”, page 18.)

The algorithm computes $D^j B_k(t)$, for $j = 0, 1, \dots, d$. It is assumed that either algorithm 9, algorithm 12 or algorithm 16 is used as the basic algorithm. The input variables are: $t \in [t_1, t_n]$, $k | t_k < t \leq t_{k+1}$, and $d \in \{0, 1, 2, \dots\}$ (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 $D^1 B_k(t)$ and where the last element is $D^d B_k(t)$.

```
vector⟨double⟩  $\bar{B}$  ( double t, int k, int d )
    vector⟨double⟩  $\tilde{B} = B(t, k, d);$  // Result evaluating ERBS-basis, (Alg. 16).
     $\tilde{B}_0 = 1 - \tilde{B}_0;$  // Remark 11, point i.
```

```

for ( int j=1; j ≤ d; j++ )
     $\tilde{B}_j = -\tilde{B}_{j-1};$            // Remark 11, point ii.
return  $\tilde{B};$ 

```

To make the finite algorithm for an ERBS curve evaluator we have to recall the equations in (8.3–8.6). 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 $\hat{c}_k(t)$ is “turned” compared with the order of the derivative of $B_k(t)$. In the computation of (8.6), we, therefore, must turn the vector from the evaluator of the basis function, algorithm 3. 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 3), 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 $\text{vector}\langle\text{double}\rangle(3)$, a vector with dimension 3. Note that the global/local affine mapping (see definition 7.7) is used in the following algorithm, it is used in line 3 and 5, i.e. in the call to the local curve evaluators.

Algorithm 4. (*For notation, see section “Algorithmic Language”, page 18.*)

The algorithm computes $\{D^j f(t)\}_{j=0}^d$ for an ERBS curve. The algorithm assumes that evaluators for the local curves (for $\omega_k(t)$, see def. 7.7) 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 $d \in \{0, 1, 2, \dots, p\}$ (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 $Df(t), \dots, D^d f(t)$.

```

vector⟨T⟩ eval ( double t, int d )
int k = k : \; t_k ≤ t < t_{k+1};           // Index for the current knot-interval.
vector⟨T⟩ c_0 = {D^j c_k(ω_k(t))}_{j=0}^d; // Result evaluating local curve - index k.
if (t == t_k)   return c_0;                  // Return only local curve - k, see (8.3).
vector⟨T⟩ c_1 = {D^j c_{k+1}(ω_{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 =  $\bar{B}(t, k, d);$           // Result evaluating ERBS-basis, (Alg. 3).
c_0 -= c_1;                                  // c_0 is now  $\hat{c}_0$ , see (8.4).
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};      // Computing (8.6), “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 than the ERBS curve.*

8.3 Bézier curves as local curves

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

$$c(t) = \sum_{i=0}^d c_i b_{d,i}(t), \quad \text{if } 0 \leq t \leq 1, \quad (8.7)$$

where the basis functions are the Bernstein polynomials

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

and where $c_i \in \mathbb{R}^n$, $i = 0, 1, \dots, d$, 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 (8.7) used for Bézier curves,

- i) de Casteljau algorithm,
- ii) a Cox-de’Boor recursion 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 [124]) for use in both Hermite interpolation and in preevaluation (both will be discussed later)

$$\mathbf{B}_d(t, \delta) = \begin{pmatrix} b_{d,0}(t) & b_{d,1}(t) & \dots & b_{d,d}(t) \\ \delta D b_{d,0}(t) & \delta D b_{d,1}(t) & \dots & \delta D b_{d,d}(t) \\ \vdots & \vdots & \ddots & \vdots \\ \delta^d D^d b_{d,0}(t) & \delta^d D^d b_{d,1}(t) & \dots & \delta^d D^d b_{d,d}(t) \end{pmatrix}. \quad (8.8)$$

The special issue about this matrix (8.8) is the scaling δ^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; $\mathbf{T}(t) : \mathbb{R}^k \rightarrow \mathbb{R}^{k-1}$, 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 as described in section 5.5.2. In the following equations we will see a 3rd degree Bézier curve and its three derivatives in matrix form,

$$\begin{aligned} c(t) &= T^3 C = T_1(t)T_2(t)T_3(t) C, \\ c'(t) &= 3 T^2 T' C = 3 T_1(t)T_2(t) T'_3 C, \\ c''(t) &= 6 T^1 T'^2 C = 6 T_1(t) T'_2 T'_3 C, \\ c'''(t) &= 6 T'^1 C = 6 T'_1 T'_2 T'_3 C, \end{aligned} \quad (8.9)$$

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 (8.9), we get the following equations,

$$\begin{aligned} c(t) &= \begin{pmatrix} 1-t & t \end{pmatrix} \begin{pmatrix} 1-t & t & 0 \\ 0 & 1-t & t \\ 0 & 0 & 1-t \end{pmatrix} \begin{pmatrix} 1-t & t & 0 & 0 \\ 0 & 1-t & t & 0 \\ 0 & 0 & 1-t & t \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{pmatrix}, \\ c'(t) &= 3 \begin{pmatrix} 1-t & t \end{pmatrix} \begin{pmatrix} 1-t & t & 0 \\ 0 & 1-t & t \end{pmatrix} \begin{pmatrix} -1 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & -1 & 1 \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{pmatrix}, \\ c''(t) &= 6 \begin{pmatrix} 1-t & t \end{pmatrix} \begin{pmatrix} -1 & 1 & 0 \\ 0 & -1 & 1 \\ 0 & 0 & -1 \end{pmatrix} \begin{pmatrix} -1 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & -1 & 1 \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{pmatrix}, \\ c'''(t) &= 6 \begin{pmatrix} -1 & 1 \end{pmatrix} \begin{pmatrix} -1 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & -1 & 1 \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{pmatrix}. \end{aligned} \quad (8.10)$$

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

¹In section 5.5, is the same matrix notation used 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 also discussed in [99].

In section 7.9, the Hermite interpolation properties are discussed. (On page 385 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 (7.53) in definition 7.7), then in order to compute, for instance, the local Bézier curve $c_i(t)$, the j th derivatives actually have to be scaled by the global/local “scaling factor” δ_i^j where

$$\delta_i = \frac{1}{t_{i+1} - t_{i-1}}, \quad (8.11)$$

as described in Theorem 7.4. The numerator in the fraction is 1 because the domain of Bézier curves is $[0, 1]$. Because the matrix (8.8) 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 (8.8).

Algorithm 5. (*For notation, see section “Algorithmic Language”, page 18.*)

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, $\{D^j b_{d,i}(t)\}_{i=0}^d$, $j = 1, 2, \dots, d$, 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 δ^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 δ_i , see (8.11), 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 δ .

```
Matrix<double> mat ( int d, double t, double δ )
    Matrix<double> B(d+1,d+1);      // The return matrix, dimension  $(d + 1) \times (d + 1)$ .
    Bd-1,0 = 1 - t;
    Bd-1,1 = t;                  // The general Cox/deBoor like algorithm for
    for ( int i=d-2; i ≥ 0; i -- ) // - Bézier/Bernstein computing the triangle
        Bi,0 = (1 - t) Bi+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.
            Bi,j = t Bi+1,j-1 + (1 - t) Bi+1,j;
        Bi,d-i = t Bi+1,d-i-1;

    Bd,0 = -δ;
    Bd,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 (8.10), and the scalings,
        for ( int i = d; i > d - k; i -- ) // - so every row extends the number
            Bi,k = s Bi,k-1;    // - of elements to d.
            for ( int j = k - 1; j > 0; j -- )
                Bi,j = s (Bi,j-1 - Bi,j);
            Bi,0 = -s Bi,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 essential 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.

8.3.1 Local Bézier curves and Hermite interpolation

We start by recalling the settings from section 7.9, and adapting them to ERBS curves.

- Given is a curve $g(t)$, $g : [t_s, t_e] \subset \mathbb{R} \rightarrow \mathbb{R}^n$, where we might have $n = 1, 2, 3, \dots$,
- 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 $i = 2, 3, \dots, m-1$ 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 $i = 1, \dots, m$.

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 7.4 that (adjusted with the local domain for Bézier curves)

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

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 7.7, and

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

is the global/local scaling factor of the domain defined in Theorem 7.4. 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 $i = 1, \dots, m$. We now get the equation for the Hermite interpolations for a local Bézier curve with index i ,

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

This can be formulated in a vector/matrix form,

$$\mathbf{B}_{d_i}(\omega_i(t_i), \delta_i) \mathbf{c}_i = \hat{\mathbf{g}}_i \quad (8.12)$$

where $\mathbf{B}_{d_i}(\omega_i(t_i), \delta_i)$ is the Bernstein/Hermite matrix described in equation (8.8) and computed in algorithm 5, 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

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

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 (8.12) according to the Bézier coefficients \mathbf{c}_i ,

$$\mathbf{c}_i = \mathbf{B}_{d_i}(\omega_i(t_i), \delta_i)^{-1} \hat{\mathbf{g}}_i. \quad (8.13)$$

The conclusion is that, in order to compute the coefficient to the local Bézier curves (8.13), one has to compute the expanded Bernstein/Hermite matrix using algorithm 5, 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., [151].

For several reasons, it is advantageous to translate all coefficients after computing (8.13), 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 system². The premise is, of course, that this homogeneous matrix system is involved in the total evaluator. This is discussed further in section 8.5.

8.3.2 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

²By “homogeneous matrix” we understand that the coordinates given in the matrix are homogeneous

preevaluation of the basis functions is a very useful method for speeding up the computations.

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

- 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 $c_i(t)$, $i = 1, 2, \dots, n$ 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,

$$x_j = t_1 + j\Delta x, \quad \text{for } j = 0, 1, 2, \dots, m - 1. \quad (8.14)$$

To distribute this sampling vector (8.14) to the local curves, we have to define the index of the first sample position touching the local domain of a local curve

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

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

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

Now we have the following three parameters to distribute to each of the local curves for local sampling and preevaluations (ω_i defined in definition 7.7):

- 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 (8.8) and in algorithm 5. This means that for a local curve, indexed by i , the total local preevaluation is computing a vector of the matrices,

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

8.3.3 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 [147], and defined by the formula,

$$g(t) = \begin{pmatrix} \cos t \cos\left(\frac{7}{4}t\right) \\ \sin t \cos\left(\frac{7}{4}t\right) \\ 0 \end{pmatrix} \quad \text{for } t \in [0, 8\pi). \quad (8.15)$$

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 (8.15). 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 8.4, 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 8.4 where, compared to the original curve, the approximating curve is buckling between two

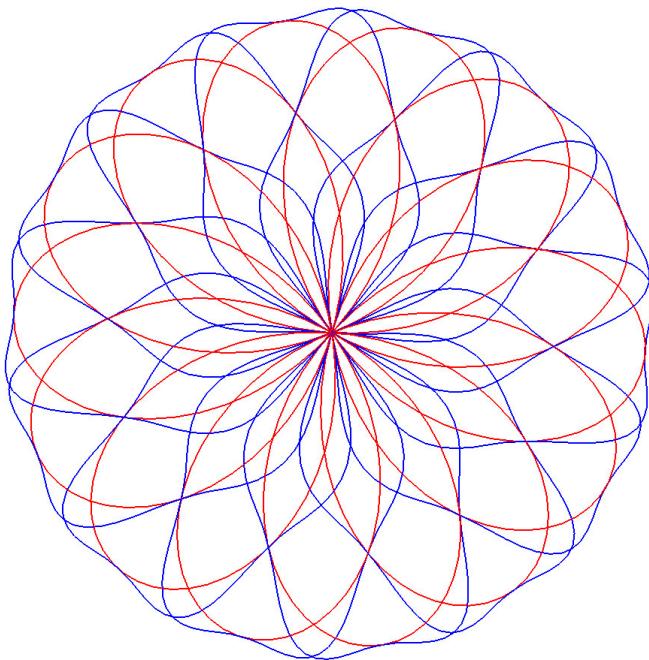


Figure 8.4: An ERBS curve (blue) approximating a “Rose-curve” (red) by using 56 interpolation points; only the position and the first derivative in each point are used. This approximation seems to make the curve too long, so that the curve is buckling between the interpolation points.

intersection points. In Figure 8.5, 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 8.6, the same interpolation as in the previous example (Figure 8.4) is done, but the resulting local curves are now scaled by 0.5, as one can see in Figure 8.7. When scaling, it is important that the interpolation point is the origin of the local coordinate system 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 8.8, 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

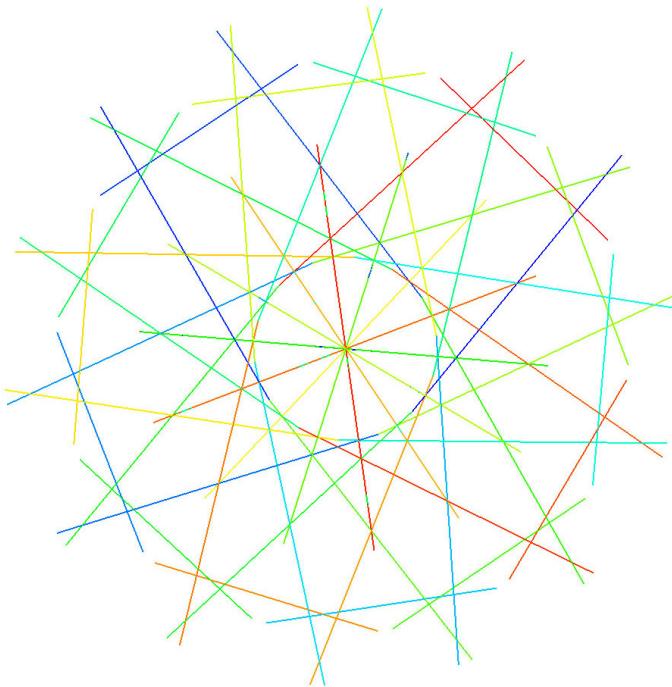


Figure 8.5: The 56 local Bézier curves (lines) of the ERBS curve from Figure 8.4, plotted in different colors (all 1st degree). Because the “Rose-curve” has 14 petals, and there are $14 \cdot 4 = 56$ interpolation points, the local curves (lines) are “symmetrical around the rose center”. In addition, every pair of subsequent lines intersect.

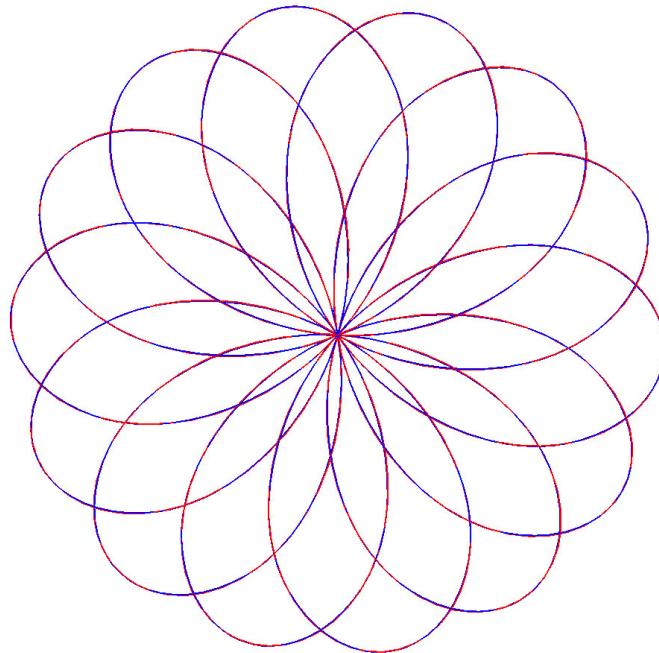


Figure 8.6: An ERBS curve (blue) approximating a “Rose-curve” (red) by using 56 interpolation points; only the position and the first derivative in each point are used, but the local curves are scaled by 0.5 after the interpolation. This approximation seems to be, geometrically, very close to the original curve.

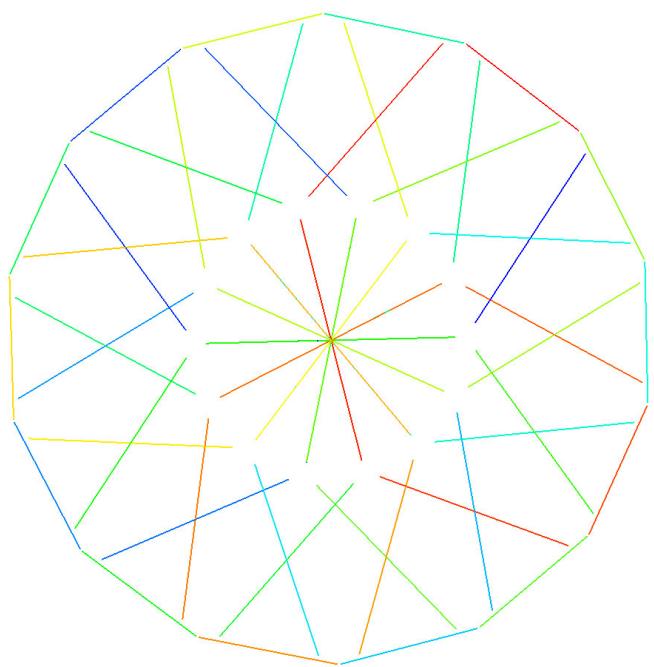


Figure 8.7: The 56 local Bézier curves (lines) of the ERBS curve (all 1st degree) from Figure 8.6 plotted in different colors. Because the “Rose-curve” has 14 petals, and there are $14 \times 4 = 56$ interpolation points, the local lines are “symmetrical around the rose center”. In addition, each line does not intersect with the previous and next line in the sequence.

Figure 8.5, 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 8.1 and Table 8.2 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 [69], and defined by the formula,

$$g(t) = \begin{pmatrix} 2\cos t(1 + \cos t) \\ 2\sin t(1 + \cos t) \\ 0 \end{pmatrix}, \quad \text{for } t \in [0, 2\pi). \quad (8.16)$$

A plot of (8.16) will look like an apple, with a cusp at the top. In Figure 8.10, 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 8.11, 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.

In Figure 8.12, 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 8.13 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 221.

In the middle of Figure 8.13, 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.

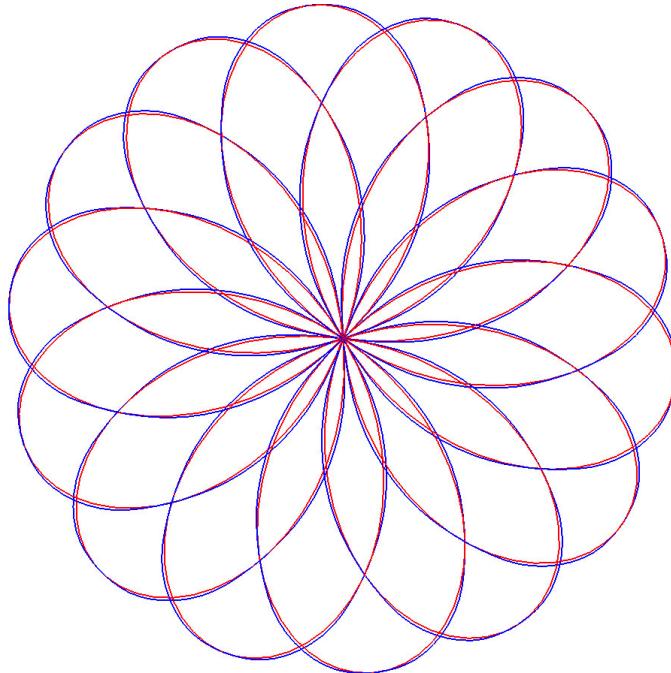


Figure 8.8: Two curves partly covering each other. The red curve is the original “Rose-curve”, the blue curve is the approximating ERBS curve. The approximation is made by 56 interpolation points, the position and the first and second derivatives in each point are used.

On the right hand side of Figure 8.13, 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.

8.4 Circular arcs as local curves

Beside B-splines and NURBS, Arc-splines have been one of the successful spline types in practical use (see [17], or [107]). 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^∞ 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 the nonlinear splines described

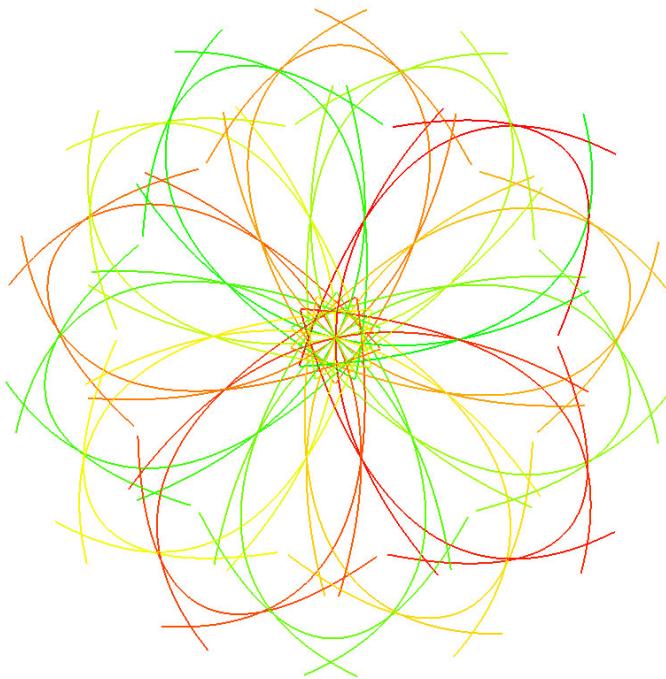


Figure 8.9: The 56 local Bézier curves of the ERBS curve from Figure 8.8 plotted gradually from green to red. They are all 2nd degree Bézier curves, forming the original curve around each interpolating point. Because the “Rose-curve” has 14 petals, and we are using $14 \times 4 = 56$ interpolation points, the local curves are “symmetrical around the rose center”.

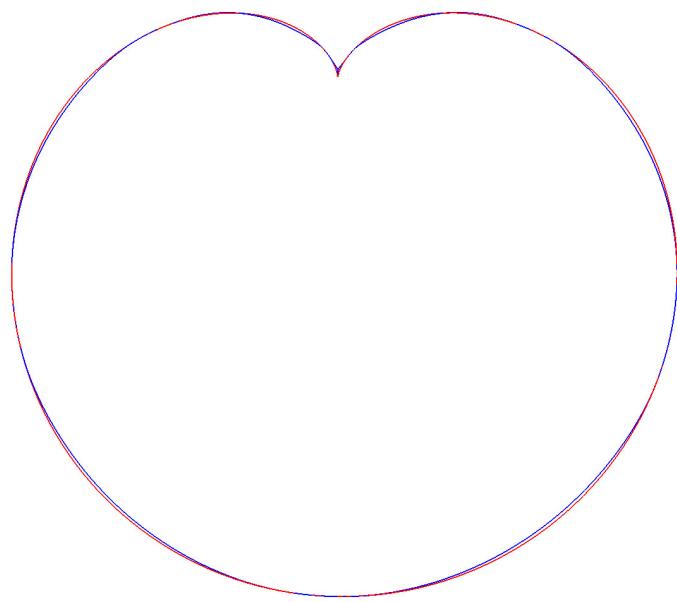


Figure 8.10: Two curves partly covering each other. The red curve is the original “Cardioid curve”. The blue curve is the approximating ERBS curve. The approximation is made by 7 interpolation points; the position and the three first derivatives in each point are used.

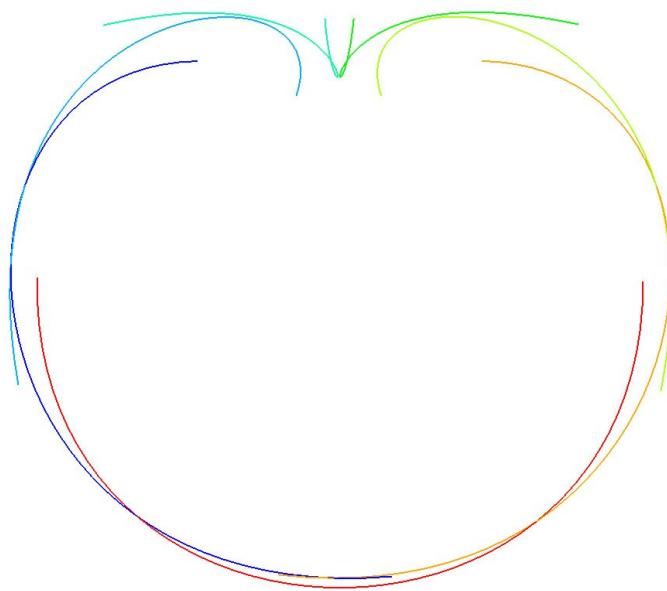


Figure 8.11: The seven local Bézier curves of the ERBS curve, from Figure 8.10, plotted in different colors. All seven curves are 3rd degree Bézier curves with 4 control points.

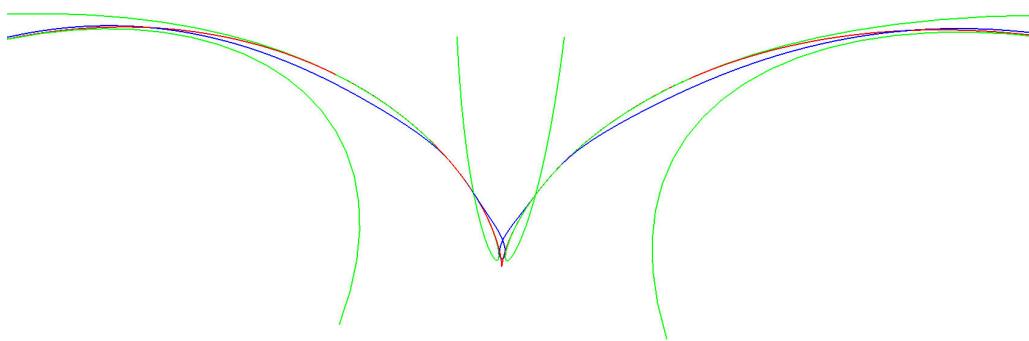


Figure 8.12: A zoomed picture of both the original “Cardioid curve” (red), the approximating ERBS curve (blue), and four of the local Bézier curves (green). The picture shows how the cusp is approximated (even when using uniform sampling).

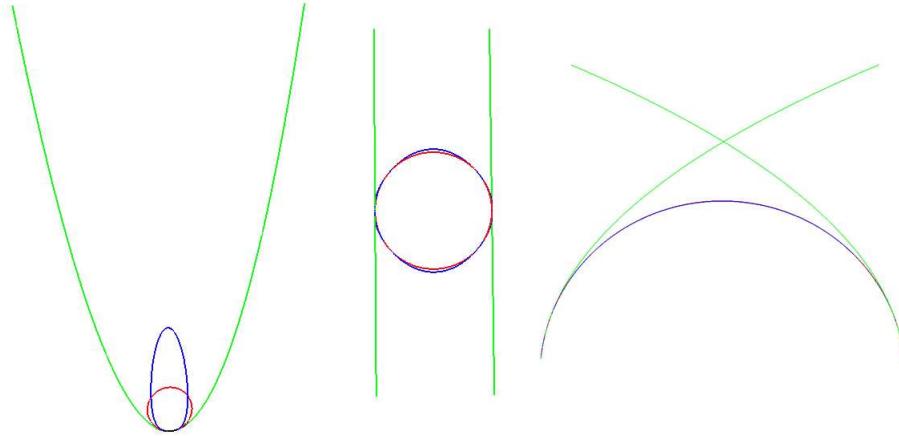


Figure 8.13: Three examples of approximations of circles/circular arcs by ERBS curves using local Bézier curves. The original curves are red, the ERBS curves are blue, and the local Bézier curves are green. On the left hand side only one local curve is used. In the middle, two local curves of 1st degree (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.

by Even Mehlum in [105] and [106].³

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

$$\tilde{c}(\alpha) = \frac{1}{\kappa} \begin{pmatrix} \sin(\kappa\delta\alpha) \\ 1 - \cos(\kappa\delta\alpha) \end{pmatrix}, \quad \text{if } \kappa > 0. \quad (8.17)$$

Equation (8.17) has the following fine properties for the use as local curves:

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

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

$$c(\alpha) = \begin{cases} \begin{pmatrix} \delta\alpha \\ 0 \end{pmatrix}, & \text{if } \kappa = 0, \\ \frac{1}{\kappa} \begin{pmatrix} \sin(\kappa\delta\alpha) \\ 1 - \cos(\kappa\delta\alpha) \end{pmatrix}, & \text{if } \kappa > 0. \end{cases} \quad (8.18)$$

This function (8.18) 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,

³A relationship to the nonlinear splines described by Even Mehlum 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.

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 $\delta = 1$.

8.4.1 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:

- Given is a curve $g(t)$, $g : [t_s, t_e] \subset \mathbb{R} \rightarrow \mathbb{R}^n$, where we might have $n = 1, 2, 3, \dots$,
- 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 $i = 2, 3, \dots, m - 1$, 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 $\kappa(t_i)$, for $i = 1, \dots, m$.

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 8.4.3, 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 7.4 implies that we have the following requirements for the interpolations. For all interior knots t_i , $i = 1, \dots, m$,

$$\begin{aligned} g(t_i) &= f(t_i) = c_i \circ \omega_i(t_i), \\ Dg(t_i) &= Df(t_i) = \delta_i Dc_i \circ \omega_i(t_i), \\ \kappa_g(t_i) &= \kappa_f(t_i) = \kappa_{c_i}(t_i), \end{aligned}$$

where $g(t)$ is the original curve, $f(t)$ is the ERBS curve, and $c_i(t)$, $i = 1, \dots, m$ 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 7.7 is the mapping

$$\omega_i(t) : [t_{i-1}, t_{i+1}] \rightarrow [(t_{i-1} - t_i) |Dg(t_i)|, (t_{i+1} - t_i) |Dg(t_i)|] \quad (8.19)$$

defined by the affine mapping

$$\omega_i(t) = (t - t_i) |Dg(t_i)|. \quad (8.20)$$

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 7.4, which is then

$$\delta_i = |Dg(t_i)|, \quad (8.21)$$

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 , $i \in \{1, 2, \dots, m\}$ it is given

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

By (8.21), the scaling factor δ_i follows directly from the input. The knot value t_i , together with δ_i , is needed to determine $\omega_i(t)$. What remains is to compute the curvature κ_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,

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

and it follows now that

$$\kappa_i = |\hat{\mathbf{y}}|.$$

We now have two possibilities concerning equation 8.18, namely that $\kappa_i > 0$ or $\kappa_i = 0$.

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

$$\mathbf{y} = \frac{\hat{\mathbf{y}}}{\kappa_i}.$$

In the general case of \mathbb{R}^n , $n = 2, 3, \dots$, 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

$$M = \begin{bmatrix} \mathbf{x}_x & \mathbf{y}_x & \mathbf{z}_x & g(t_i)_x \\ \mathbf{x}_y & \mathbf{y}_y & \mathbf{z}_y & g(t_i)_y \\ \mathbf{x}_z & \mathbf{y}_z & \mathbf{z}_z & g(t_i)_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{if } \kappa > 0, \quad (8.22)$$

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.

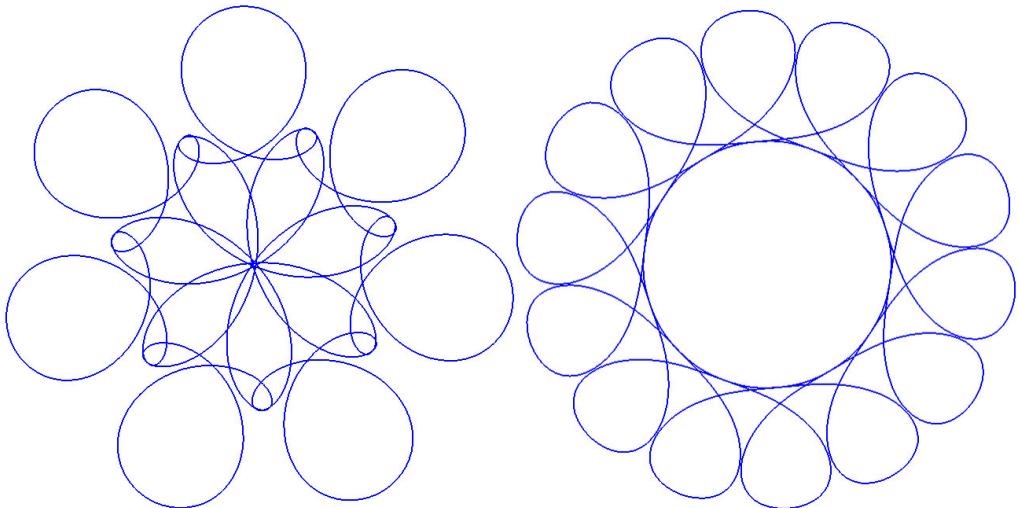


Figure 8.14: Two examples of approximations of the “Rose-curve” by ERBS curves using Circular arcs as local curves. On the left hand side only seven local curves are used. The “Rose-curve” is not completely reconstructed, only half of the “petals” are present. On the right hand side is an ERBS curve approximating the “Rose-curve” using 14 circular arcs as local curves.

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, therefore, 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.

8.4.2 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

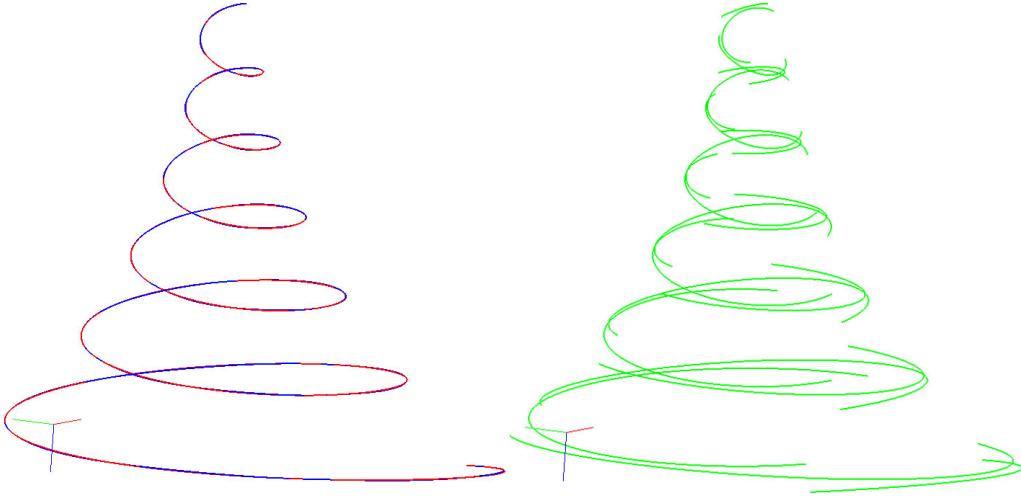


Figure 8.15: An example of approximation of a Helical curve over a logarithmic spiral by an “ERBS” curve using local Arc curves. The original curve is red, the “ERBS” curve is blue and they are both plotted on the left hand side. The local Arc curves are plotted on the right hand side, and are colored green. The number of interpolation points and, thus, local Arc curves, is 20.

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 8.4, Figure 8.6, and Figure 8.8. But now the approximation is different from the previous ones. In Figure 8.14, 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.

The second example is a Helical curve over a logarithmic spiral. In Figure 8.15, 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 [69], 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

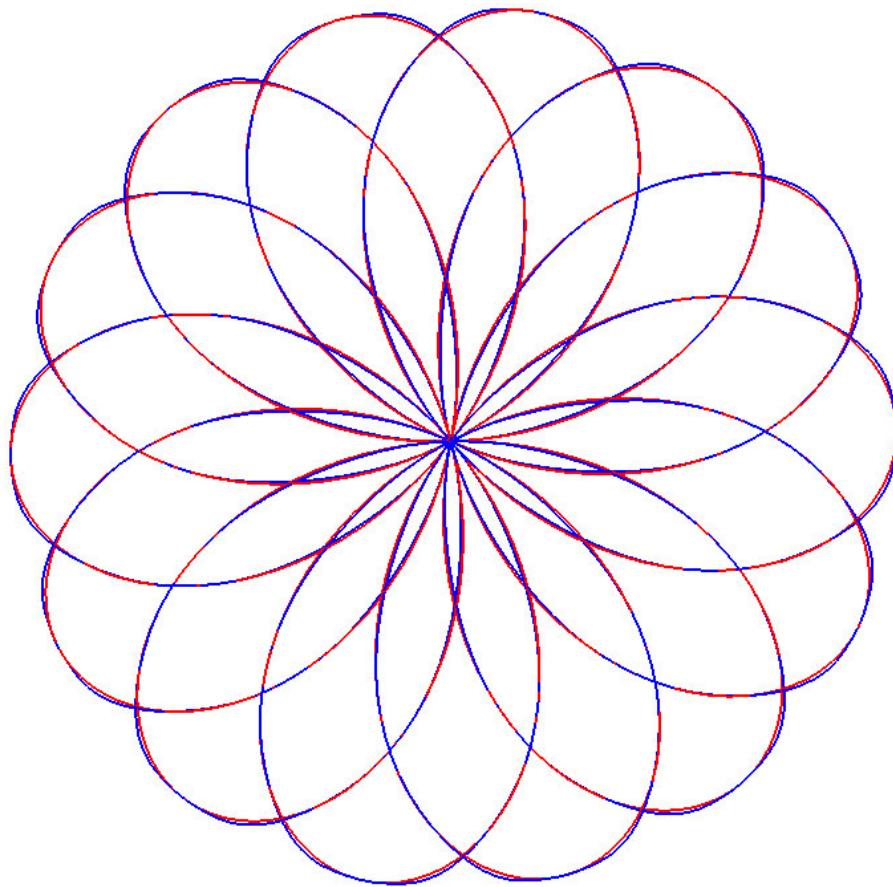


Figure 8.16: Two curves partly covering each other. The red curve is the original “Rose-curve”, the blue curve is the approximating ERBS curve using local Arc curves. The approximation is made by 56 interpolation points, using the position, the first derivative and the curvature at each of the points.

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 8.15, 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.

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 8.8 and Figure 8.9. The two figures, 8.16 and 8.17, show us the same type of approximation as in figures 8.8 and 8.9. 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

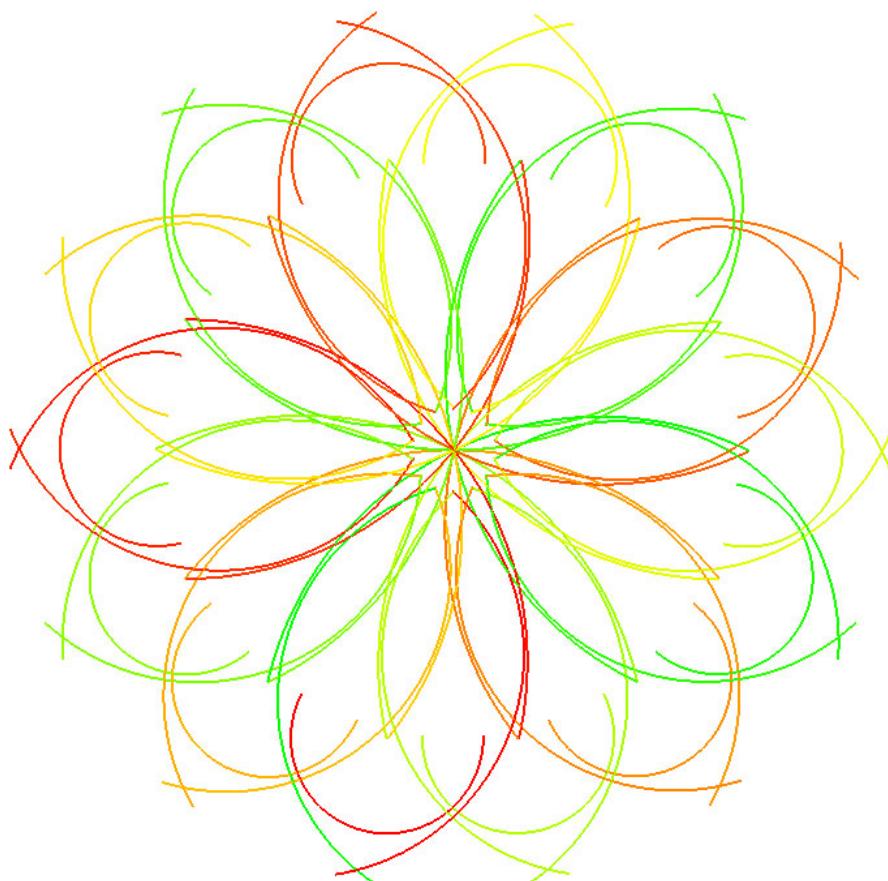


Figure 8.17: The 56 local Arc curves of the ERBS curve from Figure 8.16 plotted in colors gradually changing from green to red. They are forming the original curve locally around the interpolating points. One can see that they are quite similar to the Bézier curves at figure 8.9.

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 × parameter-interval	0.049	0.028
	deg. 2	parameter-interval	0.0092	0.0065
Arc curves	Circular arc	speed × parameter-interval	0.021	0.012
		curve length	0.0110	0.0060

Table 8.1: 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 (8.23) and (8.24)). The computations are done by using numerical integration (algorithm 13) 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.

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 8.17) with the previous example (Figure 8.9). 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

$$L^\infty(f - g) = \max_{t \in [t_1, t_n+1]} |f(t) - g(t)|, \quad (8.23)$$

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

$$L^2(f - g) = \sqrt{\frac{1}{t_{n+1} - t_1} \int_{t=t_1}^{t_{n+1}} |f(t) - g(t)|^2 dt}, \quad (8.24)$$

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 8.1 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 8.1 are computed from this diameter. One can clearly see that using 2nd degree local Bézier curves gives the best result, according to the L^∞ 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 8.1 is a modification of the affine mapping ω_i , defined in (8.20). 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 δ_i , defined in (8.21) 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.

8.4.3 Reparametrization and using an approximative curve length parametrization

Circular Arc curves as described in equation (8.18) 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 $n = 1, 2, 3, \dots$,
- 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

$$t_i = t_{i-1} + \int_{t=x_{i-1}}^{x_i} |Dg(t)| dt, \quad \text{for } i = 2, 3, \dots, m,$$

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 $\kappa_g(t_i)$, all for $i = 1, \dots, m$.

The consequences are that we have to change (8.19), (8.20), and (8.21). Since the knot vector already represents the curve length, ω_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,

$$\delta_i = 1.$$

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 8.18, there is a plot of the speed of the original

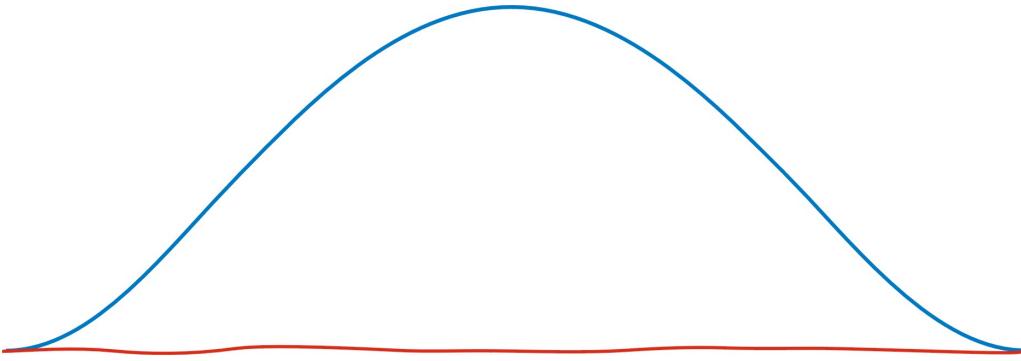


Figure 8.18: The blue curve is the plot of the function describing the speed of the original “Rose-curve”, but for only one of the fourteen petals. The value at the start and end is 1.0, and the top value in the middle is at 1.75. The red curve is the plot of the function describing the speed of the approximating ERBS curve. The value is close to 1.0 throughout.

“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 $L^\infty(Df - 1)$	Norm type $L^2(Df - 1)$	Norm type $L^1(Df - 1)$
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 [90].)

Definition 8.3. Given is a point $\mathbf{p} \in \mathbb{R}^n$, and a curve $f \subset \mathbb{R}^n$, with a map $\mathbf{f}: I \subset \mathbb{R} \rightarrow f$. We define

$$C_f(\mathbf{p}) : \mathbb{R}^n \rightarrow f,$$

to be the closest point on a curve f from a point \mathbf{p} . By closest point we mean that the distance (metric)

$$d(\mathbf{p}, C_f(\mathbf{p})) < d(\mathbf{p}, \mathbf{q}), \quad \text{for all } \mathbf{q} \in O(C_f(\mathbf{p}))$$

where $O(\mathbf{p}_f)$ is a neighborhood, on the curve f , around the point $\mathbf{p}_f \in f$.

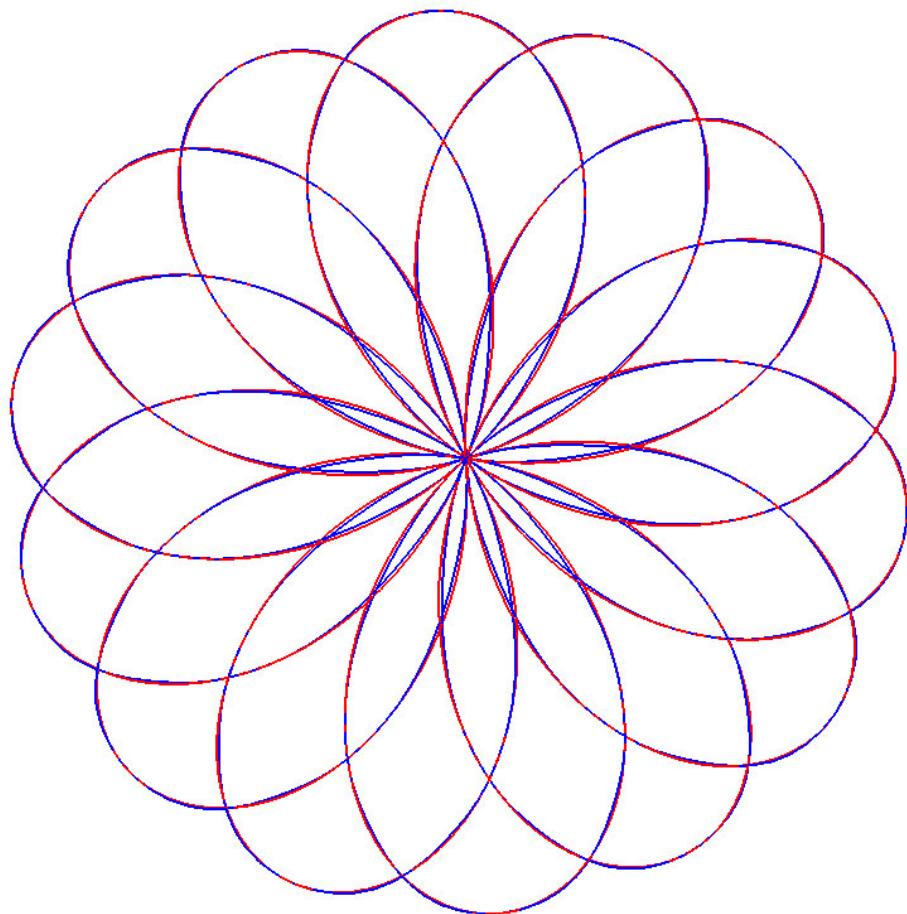


Figure 8.19: Two curves partly covering each other. The red curve is the original “Rose-curve”, the blue curve is the approximating ERBS curve using local Arc curves. The approximation is a reparametrization to curve length parametrization.

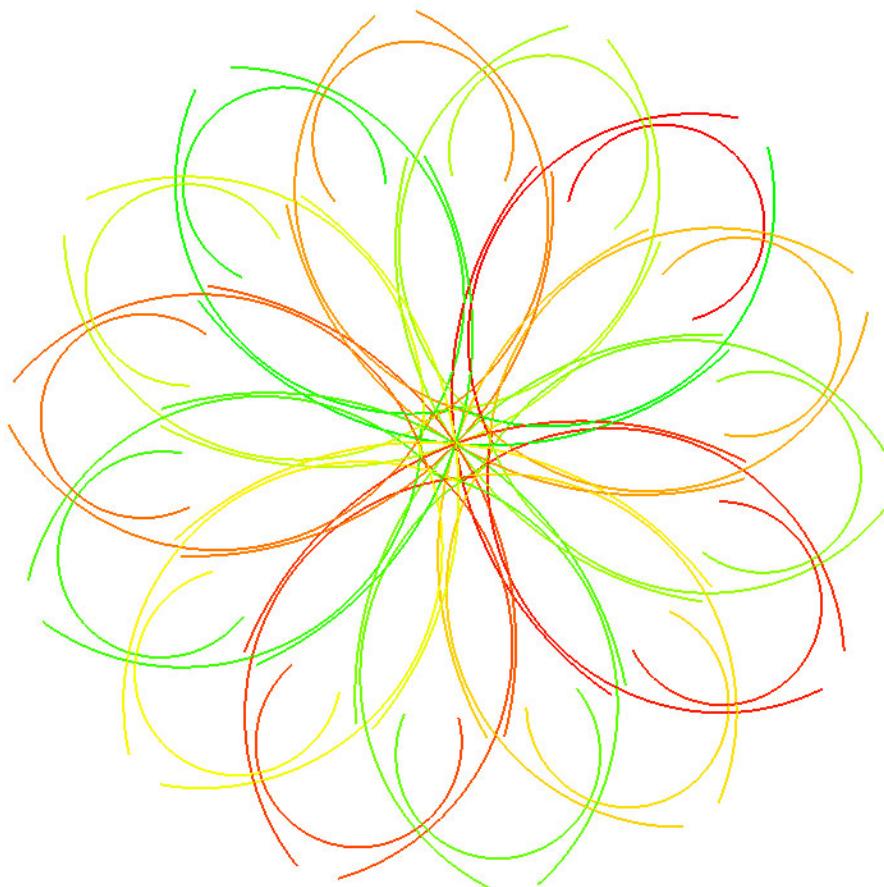


Figure 8.20: The 56 local Arc curves of the ERBS curve from Figure 8.19 plotted in colors gradually changing from green to red. They are forming the original curve locally around the interpolating points. One can see that the curves on the sides of each petal are shorter than the ones in Figure 8.17, and the curves at the tip of the petals are longer than the ones in Figure 8.17.

Local Curves	degree/parametrization	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 8.2: 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 (8.25) and the $L_G^2(f, g)$ in (8.26) metrics. The computations are done by using numerical integration (algorithm 13). 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. Recall 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 ,

$$L_G^\infty(f, g) = \max_{t \in [t_1, t_{n+1}]} |f(t) - C_g(f(t))|. \quad (8.25)$$

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,

$$L_G^2(f, g) = \sqrt{\frac{\int_{t_1}^{t_{n+1}} |f(t) - C_g(f(t))|^2 |Df(t)| dt}{\int_{t_1}^{t_{n+1}} |Df(t)| dt}}, \quad (8.26)$$

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 8.2 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 8.6. 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 re-parametrization 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 8.1 we get $L^\infty(f - g) \approx 0.0092$ and from Table 8.2, $L_G^\infty(f, g) \approx 0.0089$). This is a very likely result because the local Bézier curves are also approximating the parametrization.

In Figure 8.19 we can see the original “Rose-curve” (red) and the new “curve length parametrised” ERBS curve (in blue), and in Figure 8.20 the belonging arc-curves are plotted in colors gradually changing from green to red.

8.5 Affine transformation on local curves

At the end of subsection 8.3.1 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 8.4 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 8.2.
 - 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 384.
- 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 (8.22). 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

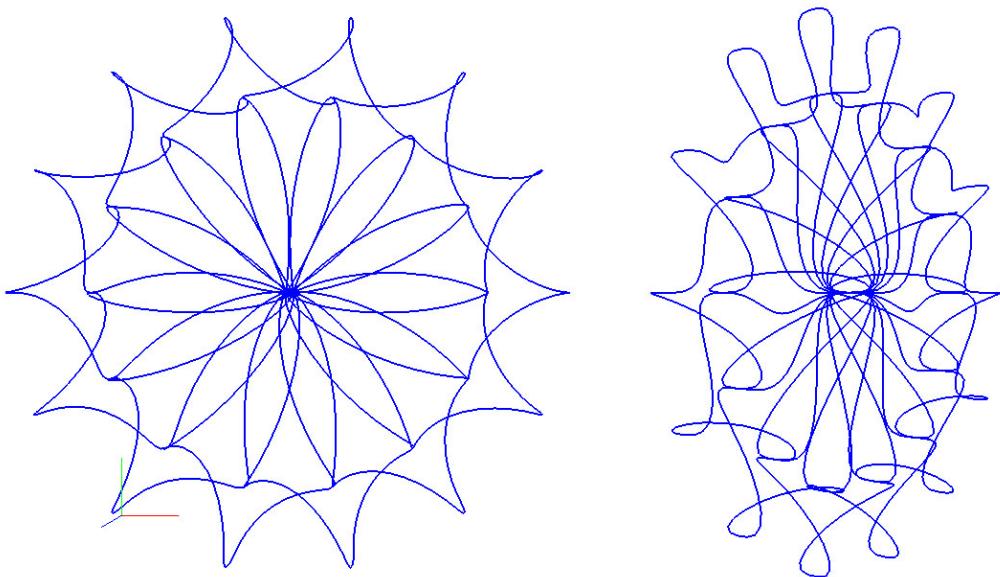


Figure 8.21: The “Rose-curve” is approximated by an ERBS curve using 56 interpolation points and local arc-curves, as in the previous examples. But in this case all local Arc curves are rotated 90° around their local x-axis, which are the first derivatives at the interpolation points. In the figure, 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.

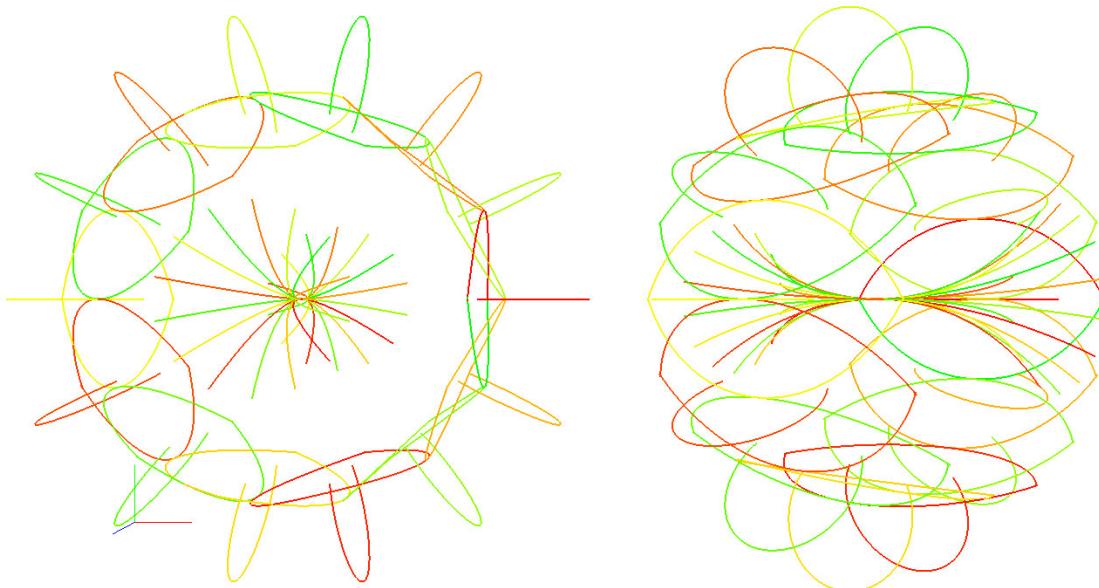


Figure 8.22: The local curves of the ERBS curve from Figure 8.21 can be seen from the same angles as in Figure 8.21. They are plotted in colors gradually changing from green to red so they can be separated.

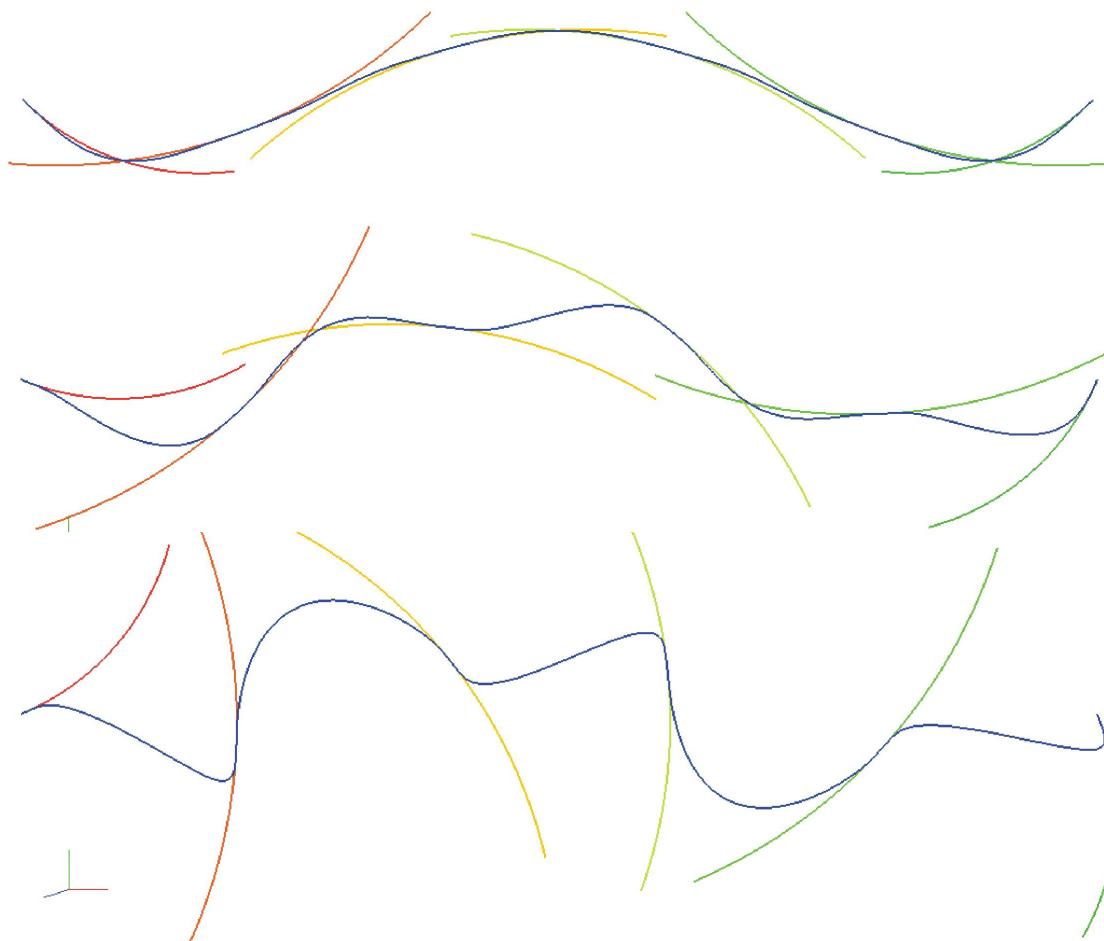


Figure 8.23: An example of a dynamical moving rope. The initial curve is a 4th degree Bézier curve, and the approximating ERBS curve uses 6 local Arc curves. First the curve is at the initial state, then the local curves start to rotating around their local z-axis. the second and the last plot shows snapshots after the rotation has started.

result can be seen in Figure 8.21. 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 8.22, 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 8.23, 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.

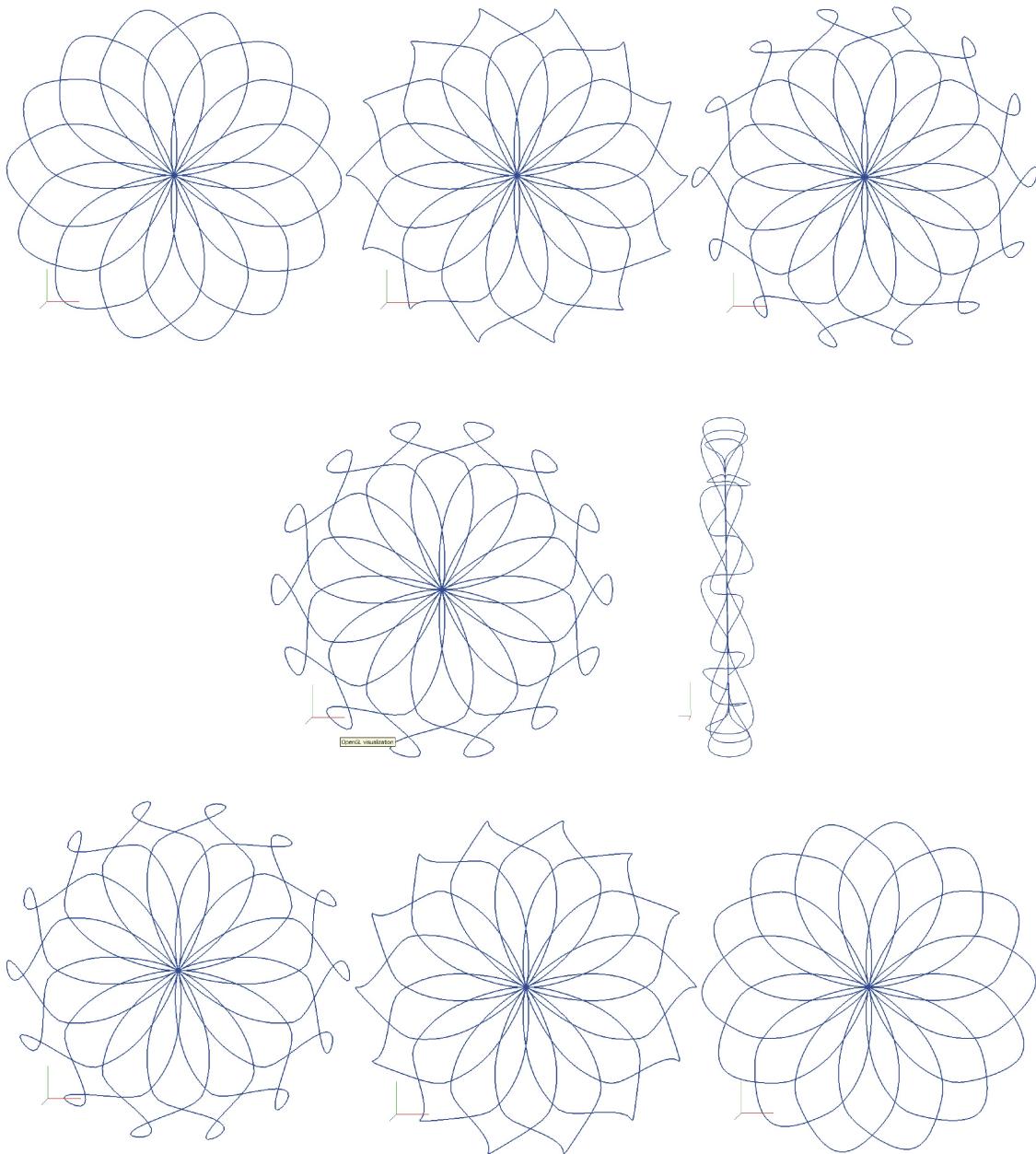


Figure 8.24: The “Rose-curve” approximated by an ERBS curve using 56 interpolation points and local arc-curves, as in the previous example. But in this case only the local Arc curves at the tip of the petals are rotated (actually, 7 times around their local y-axis which coincide with the direction of 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 8.24, 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° 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.

Part II

Surfaces

Chapter 9

Parametric Surfaces

We think about a surface as the outer boundary of an object in the 3 dimensional space. It is generally not always possible to find a relation, a homeomorphism, between a surface and the plane. It is however always possible to pick a part of a surface that is homeomorphic to a plane (or a part of a plane).

To give an ide of what a parametric surface is, imagine \mathbb{R}^2 as an infinite plane. To make a surface we just pick a part of this plane, for which we call a parameter plan. Then we put it into the required space that may be the Euclidian \mathbb{R}^3 , deform it by either stretching or pressing it and bending it in several ways. Then finally we put it in the desired position and orientation. Note that to cut and glue is *not* an option here. Figure 9.2 give an example of this concept. A surface is a 2-dimensional object. If a surface is neither degenerated nor self intersecting we also call it a 2-dimensional manifold.

This is an attempt to illustrate the concept of parametric Surfaces. Usually is the output to be embedded in \mathbb{R}^3 but it can be in \mathbb{R}^n , $n > 0$. A more formal definition is:

Definition 9.1. A parameterized differentiable surface is a differentiable map $S : U \subset \mathbb{R}^2 \rightarrow \mathbb{R}^n$ of an open set $U \subset \mathbb{R}^2$ into \mathbb{R}^n . Note that a half open or closed set is just a restriction of an open set.

For the rest of the chapter we will assume that surfaces are embedded in \mathbb{R}^3 . A map is usually notated by a letter, eg. $S : U \subset \mathbb{R}^2 \rightarrow \mathbb{R}^3$ where U is the domain, the parameter plane, where the coordinates usually are notated by the letters (u, v) . A surface can have a boundary as in the monkey saddle example or partly boundary as in a cylinder or no boundary as in a torus. The first example is a monkey saddle defined as following

$$S(u, v) = \begin{pmatrix} u \\ v \\ uv^2 \end{pmatrix}, \quad u \in [-1, 1], \quad v \in [-1, 1]. \quad (9.1)$$

Here the domain $U = [-1, 1] \times [-1, 1]$ and the parameter plane is reflecting the $x - y$ plane in \mathbb{R}^3 ($x = u$ and $y = v$). This means that the surface is stretched in the mapping process and that the area must be bigger than the parameter plane, $2^2 = 4$. A plot of the surface is shown in figure 9.1.

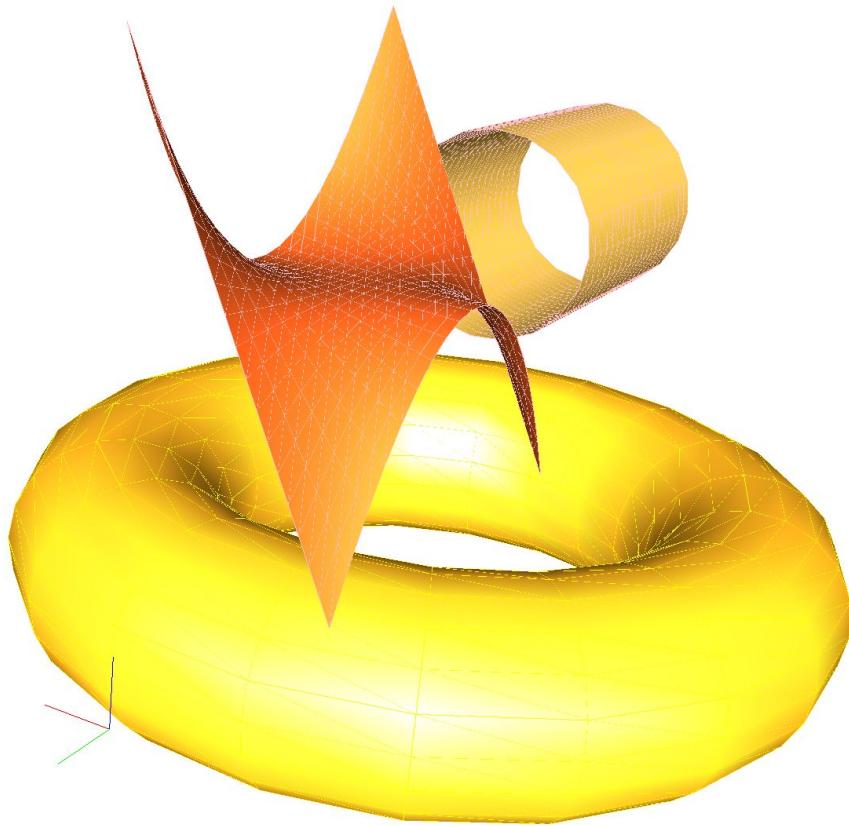


Figure 9.1: Three example surfaces, a so called monkey saddle from expression (9.1), a cylinder from expression (9.2) and a torus from expression (9.3).

The next example is a cylinder,

$$S(u, v) = \begin{pmatrix} \cos v \\ \sin v \\ u \end{pmatrix}, \quad u \in [0, 3], \quad v \in [0, 2\pi]. \quad (9.2)$$

Here the domain $U = [0, 2\pi] \times [0, 3]$. This mapping is more like taking a piece of paper and roll it. The surface is not stretched in the mapping process and the area is the same as in the parameter plane, 6π . A plot of the surface is shown in figure 9.1.

Example three is a torus,

$$S(u, v) = \begin{pmatrix} (\cos u + 3) \cos v \\ (\cos u + 3) \sin v \\ \sin u \end{pmatrix}, \quad u \in [0, 2\pi], \quad v \in [0, 2\pi]. \quad (9.3)$$

Here the domain $U = [0, 2\pi] \times [0, 2\pi]$. A plot of the surface is shown in figure 9.1.

Figure 9.2 is an attempt to provide a picture of what a parametric surface is. In the figure the parameter plane is shown at lower left side. It is named U and the coordinates are named u and v . The map is called S and marked with a dotted curve (arrow). The surface, upper right hand is marked as $S(U)$ in the figure. In the figure is S mapping a point p in the parameter plane to a point q on the surface. Summing up:

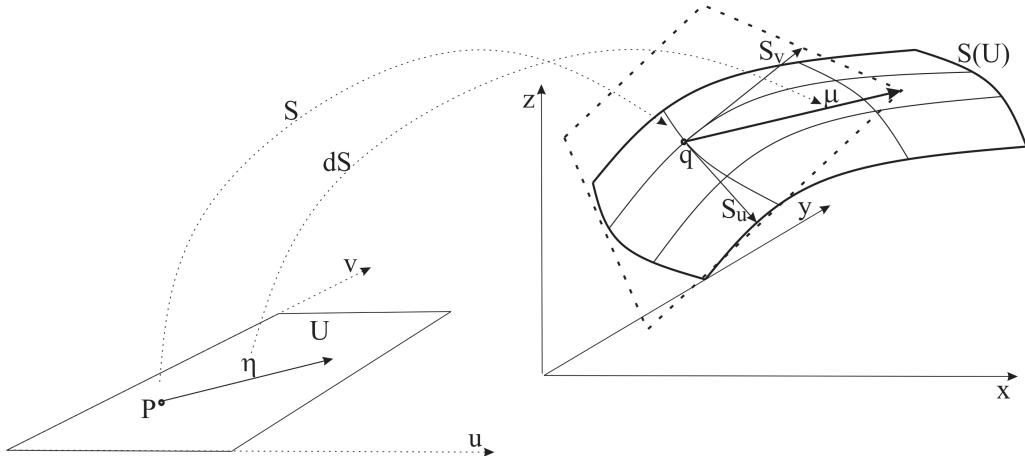


Figure 9.2: On lower left hand side is the parameter plane $U \subset \mathbb{R}^2$ shown. On upper right hand side is there the surface, $S(U)$, embedded in \mathbb{R}^3 . A point p in the parameter plane is mapped by S to a point $q = S(p)$ on the surface. A vector η from the point p in the parameter plane is mapped by dS to a vector $\mu = dS_p(\eta)$ laying in the tangent plane $T_q(S)$ at the point q on the surface.

- $S(U)$ is the entire surface embedded in \mathbb{R}^3 .
- $p = (u, v)$ is a point in the parameter plane \mathbb{R}^2 , (u, v) are the coordinates of the point.
- $q = S(p) = S(u, v)$ is a point on the surface (usually in \mathbb{R}^3).
- S maps points in the parameter plane to points on the surface.

9.1 Differentiation

The next step which also is illustrated in figure 9.2 is differentiation. Recall that S maps a point p from the parameter plane to a point q on the surface. We want to find the direction we have to move if we are located in q and want to move in the direction where only one of the two parameter values are changing. We call these direction vectors the partial derivatives. A surface has two parameter values and thus two partial derivatives. There are several notations used for derivatives. In the following we will see 3 different notations that are common in use.

- The partial derivative in u direction can be denoted in three different ways, $D_u S = S_u = \frac{\partial S}{\partial u}$, and is a 3-dimensional vector (if the surface is embedded in \mathbb{R}^3) and a vector tangential to the surface at the given position.
- The partial derivative in v direction can be denoted in three different ways, $D_v S = S_v = \frac{\partial S}{\partial v}$, and is a 3-dimensional vector (if the surface is embedded in \mathbb{R}^3) and a vector tangential to the surface at the given position.

The two tangent vectors S_u and S_v spans a tangent plane at the point $q = S(p)$. The tangent plane, denoted $T_q(S)$, is also shown in figure 9.2 with dotted lines. Remember that the dimension of the tangent vectors, the partial derivatives, is equal the dimension of the space for which the surface is embedded. As an example of partial derivatives we can see

at the monkey saddle in expression (9.1). We then get the following partial derivatives (using all three notations),

$$D_u S = S_u = \frac{\partial S}{\partial u} = \begin{pmatrix} 1 \\ 0 \\ v^2 \end{pmatrix}, \quad D_v S = S_v = \frac{\partial S}{\partial v} = \begin{pmatrix} 0 \\ 1 \\ 2uv \end{pmatrix}, \quad (9.4)$$

9.1.1 The differential dS_p

The next map is the differential dS , a linear map, a matrix where the two partial derivatives are the columns,

$$dS = [S_u \ S_v], \quad \in \mathbb{M}(s, 2),$$

where s is the dimension of the space the surface is embedded in. This means that dS has two columns and the number of rows is equal the dimension of the space the surface is embedded into (i.e. 3 in \mathbb{R}^3). It maps a vector in point p in the parameter plane, to a vector in the tangent plane $T_q(S)$ of the point $q = S(p)$.

In figure 9.2 is there an example where a vector η is mapped to a vector μ by dS_p . i.e. $dS_p(\eta) = \mu$. We call μ the directional derivative in direction η . If the surface is embedded in \mathbb{R}^3 then dS_p is a 2×3 matrix, i.e.

$$dS_p : \mathbb{R}^2 \rightarrow \mathbb{R}^3, \quad p \in U \subset \mathbb{R}^2. \quad (9.5)$$

It also follows that d is a differential operator making a map to a differential map, we therefor get

$$\begin{aligned} d(S_u) &= [S_{uu} \ S_{uv}], \\ d(S_v) &= [S_{vu} \ S_{vv}], \end{aligned}$$

and so on.

9.1.2 Curves on surfaces

We will describe the construction of how curves defined in the parameter plane of a surface is mapped into \mathbb{R}^3 .

Given a curve $h : I \subset \mathbb{R} \rightarrow \mathbb{R}^2$,

$$h(t) = \sum_{i=1}^n c_i b_i(t).$$

where $c_i, \ i = 1, \dots, n$ are points or vectors in \mathbb{R}^2 and $b_i(t)$ are basis functions spanning a finite dimensional function space.

The question is, what curve will we get when the curve in the parameter space is mapped into 3D-space. Given a surface $S : U \subset \mathbb{R}^2 \rightarrow \mathbb{R}^3$. The curve definition then is,

$$c(t) = S(h(t)) = S \circ h(t), \quad t \in I \subset \mathbb{R}. \quad (9.6)$$

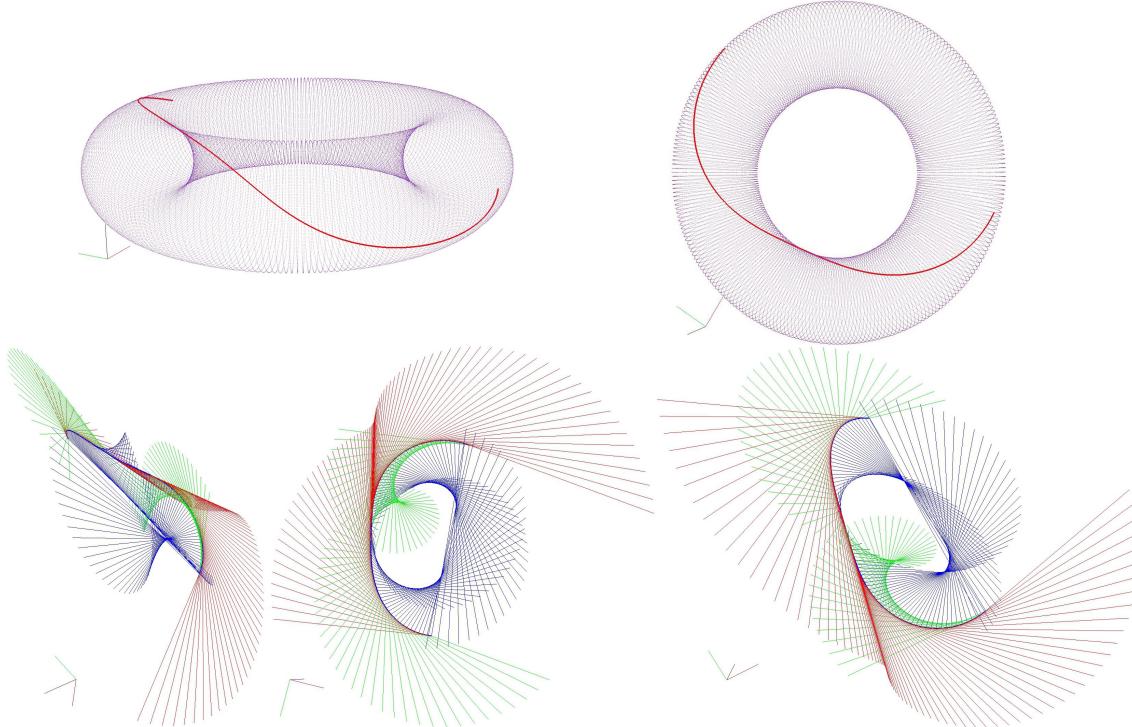


Figure 9.3: On top is two views of a curve on a torus displayed together with the torus. On the lower part is there three views of the same curve. But now also the first derivatives are shown in red, the second derivatives is shown in blue and the third derivatives is shown in green.

Using the chain rule on (9.68) we get the first, second and third derivative,

$$\begin{aligned} c'(t) &= dS(h'), \\ c''(t) &= d(dS(h'))(h') + dS(h''), \\ c'''(t) &= d(d(dS(h'))(h'))(h') + d(dS(h'))(h'') + d(dS(h''))(h') + dS(h'''), \end{aligned} \quad (9.7)$$

where the differentials matrices, there are used in the derivatives above are,

$$\begin{aligned} d(dS(h')) &= d([S_u, S_v] h') \\ &= [([S_u, S_v] h')]_u, ([S_u, S_v] h')]_v \\ &= [[S_{uu}, S_{vu}] h', [S_{uv}, S_{vv}] h'], \end{aligned}$$

and

$$\begin{aligned} d(d(dS(h'))(h')) &= d([[S_{uu}, S_{vu}] h', [S_{uv}, S_{vv}] h'] h') \\ &= [([S_{uu}, S_{vu}] h', [S_{uv}, S_{vv}] h')]_u, ([S_{uu}, S_{vu}] h', [S_{uv}, S_{vv}] h')]_v \\ &= [[[S_{uuu}, S_{vuu}] h', [S_{uvu}, S_{vvu}] h'] h', [[S_{uuv}, S_{vuv}] h', [S_{uvv}, S_{vvv}] h']] h']. \end{aligned}$$

In Figure 9.15 is a first degree Bezier curve, a straight line, expression (4.28), in the parameter plane of a torus, expression (9.3), mapped into \mathbb{R}^3 by expression (9.68). On

top in Figure 9.15 is two different views of the curve together with the torus shown, on the lower part is there three different views of the curve and its first derivative (red), second derivatives (blue) and third derivatives (green) from expression (9.69).

Another example of “curves” on a surface is a vector valued function describing the directional derivatives along a curve, not in the curve direction, but perpendicular to the curve direction. In this case we need two vector valued functions in the parameter plane, one vector valued function describing the position in the parameter plane depending on the parameter, another vector valued function describing the vector in \mathbb{R}^2 to find the directional derivatives on the surface. Vi therefore must have the two vector valued functions,

$$h(t) = \sum_{i=1}^{n_1} c_i b_i(t), \quad (9.8)$$

$$r(t) = \sum_{i=1}^{n_2} d_i b_i(t), \quad (9.9)$$

both for $t \in I \subset \mathbb{R}$. The map from parameter space of the surface to 3D-space is then

$$\tilde{c}(t) = dS_{h(t)}(r(t)). \quad (9.10)$$

The derivatives are,

$$\tilde{c}'(t) = d(dS(r))(h') + dS(r'), \quad (9.11)$$

$$\tilde{c}''(t) = d(d(dS(r))(h'))(h') + d(dS(h'))(r') + d(dS(r'))(h') + dS(r''),$$

where the differentials matrices are,

$$\begin{aligned} d(dS(r)) &= d([S_u, S_v] r) \\ &= [[S_u, S_v] r]_u, [[S_u, S_v] r]_v \\ &= [[S_{uu}, S_{vu}] r, [S_{uv}, S_{vv}] r], \end{aligned}$$

and

$$\begin{aligned} d(d(dS(r))(h')) &= d([[S_{uu}, S_{vu}] r, [S_{uv}, S_{vv}] r] h') \\ &= [[[S_{uu}, S_{vu}] r, [S_{uv}, S_{vv}] r] h']_u, [[[S_{uu}, S_{vu}] r, [S_{uv}, S_{vv}] r] h']_v \\ &= [[[S_{uuu}, S_{vuu}] r', [S_{uvu}, S_{vvu}] r] h', [[S_{uuv}, S_{vuv}] r, [S_{uvv}, S_{vvv}] r] h']. \end{aligned}$$

An example of this type of vector valued function, where the expressions (9.70), (9.72) and (9.73) are used is given in section 9.3.

9.1.3 The tangent plane $T_q(S)$

Consider any curve in a surface passing a point $q = S(p)$ on the surface. The derivatives of these curves at the common point q is, from (9.69),

$$c' = dS_p(h') = [S_u, S_v] \begin{pmatrix} h'_u \\ h'_v \end{pmatrix} = h'_u S_u + h'_v S_v.$$

We can see that the first derivative, the tangent vector, of every curve on the surface passing q can be described as a linear combination of the two vectors S_u and S_v . This shows that they all lay in a plane spanned by S_u and S_v . This plane is therefore a tangent plane of surface $S(U)$ at the point q , and we denote it

$$T_q(S) = dS_q(\mathbb{R}^2).$$

It is obvious that the tangent plane still is the same even if we change parametrization. It is an intrinsic property independent of the parametrization.

In \mathbb{R}^3 we denote the unit vector orthogonal to the tangent plane to be the normal of the surface. The notation and orientation is in the following way. We first define the normal with a small letter,

$$n(q) = S_u \wedge S_v(q), \quad (9.12)$$

where \wedge denote the 3D vector product. We then define the unit normal to be denoted

$$N(q) = \frac{n(q)}{|n(q)|}. \quad (9.13)$$

The unit normal is also obvious independent of the parametrization and thus an intrinsic property.

9.1.4 First fundamental form

Here we shall study geometric structures carried by the surface and that is connected to the tangent plane. The first fundamental form is the inner product on the tangent plane of a surface embedded in a 3D Euclidean space which is induced canonically from the dot product of \mathbb{R}^3 . It permits the calculation of curvature and metric properties of a surface such as length and area in a manner consistent with the ambient space. The first fundamental form is denoted by the Roman numeral I,

$$I_q(r, s) = \langle r, s \rangle.$$

Let $S(p)$, $p = (u, v) \in \mathbb{R}^2$ be a parametrization of a surface. Then the inner product of two tangent vectors $r = dS_p(\hat{r})$ and $s = dS_p(\hat{s})$, $\hat{r} = (a, b)$ and $\hat{s} = (c, d)$, is

$$\begin{aligned} I_q(r, s) &= \langle dS_p(\hat{r}), dS_p(\hat{s}) \rangle_{S(p)} \\ &= \langle aS_u + bS_v, cS_u + dS_v \rangle \\ &= ac\langle S_u, S_u \rangle + (ad + bc)\langle S_u, S_v \rangle + bd\langle S_v, S_v \rangle \\ &= ac E + (ad + bc) F + bd G, \end{aligned}$$

where E , F and G are the coefficients of the first fundamental form.

First fundamental form

The coefficients of the first fundamental form are

$$\begin{aligned} E &= \langle S_u, S_u \rangle \\ F &= \langle S_u, S_v \rangle \\ G &= \langle S_v, S_v \rangle \end{aligned} \quad (9.14)$$

The first fundamental form, represented as a symmetric matrix.

$$I_q(r, s) = \hat{r}^T \begin{pmatrix} E & F \\ F & G \end{pmatrix} \hat{s}.$$

where $r, s \in T_q(S)$, the tangent plane of S in the point $q = S(p)$, and $\hat{r}, \hat{s} \in \mathbb{R}^2$, connected to the point p in the parameter plane.

If the vectors r and s are the same vector we denote

$$I_q(\mu) = \hat{\mu}^T \begin{pmatrix} E & F \\ F & G \end{pmatrix} \hat{\mu}.$$

where $\mu = dS_p(\hat{\mu})$, $\hat{\mu} \in \mathbb{R}^2$.

The first fundamental form completely describes the metric properties of a surface. Thus, it enables one to calculate the lengths of curves on the surface and the areas of regions on the surface. The line element ds may be expressed in terms of the coefficients of the first fundamental form as

$$ds^2 = I_q(d\hat{s})$$

where $d\hat{s} = \begin{pmatrix} du \\ dv \end{pmatrix}$, the elements in \mathbb{R}^2 . Computing this we get

$$ds^2 = (du, dv) \begin{pmatrix} E & F \\ F & G \end{pmatrix} \begin{pmatrix} du \\ dv \end{pmatrix} = E du^2 + 2F du dv + G dv^2.$$

The classical area element given by $dA = |S_u \wedge S_v|$ can be expressed in terms of the first fundamental form with the assistance of Lagrange's identity,

$$dA = |S_u \wedge S_v| du dv = \sqrt{EG - F^2} du dv.$$

First example is computation of length of curve on surfaces.

$$\begin{aligned} l(c) &= \int_a^b \frac{ds}{dt} dt \\ &= \int_a^b \sqrt{E \frac{du^2}{dt} + 2F \frac{du}{dt} \frac{dv}{dt} + G \frac{dv^2}{dt}} dt \\ &= \int_a^b \sqrt{E u'^2 + 2F u' v' + G v'^2} dt \end{aligned}$$

Using the curve on a torus, equation (9.3), shown in Figure 9.15, we get,

$$S_u = \begin{pmatrix} -\sin u \cos v \\ -\sin u \sin v \\ \cos u \end{pmatrix},$$

$$S_v = \begin{pmatrix} -(\cos u + 3) \sin v \\ (\cos u + 3) \cos v \\ 0 \end{pmatrix},$$

and than,

$$E = 1, \quad F = 0, \quad G = (\cos u + 3)^2$$

The curve in Figure 9.15 defined in the parameter plane of the torus is

$$h(t) = \begin{pmatrix} \frac{2}{5}\pi \\ \frac{1}{5}\pi \\ t \end{pmatrix} + \begin{pmatrix} \frac{6}{5}\pi \\ \frac{8}{5}\pi \\ 0 \end{pmatrix} t,$$

and the derivative is

$$h'(t) = \begin{pmatrix} u' \\ v' \\ 1 \end{pmatrix} = \begin{pmatrix} \frac{6}{5}\pi \\ \frac{8}{5}\pi \\ 1 \end{pmatrix}$$

We now get

$$l(c) = \int_0^1 \frac{\pi}{5} \sqrt{36 + 64(\cos u + 3)^2} dt \approx 10.547,$$

where the approximated solution is computed using composite Simpson's rule.

The second example is computing the area of the torus defined in (9.3),

$$\begin{aligned} A(S) &= \int_U dA \\ &= \int_0^{2\pi} \int_0^{2\pi} (\cos u + 3) du dv \\ &= 2\pi \int_0^{2\pi} (\cos u + 3) du \\ &= 12\pi^2 \end{aligned}$$

9.1.5 Second fundamental form

The second fundamental form (or shape tensor) is a quadratic form on the tangent plane of a smooth surface in the three dimensional Euclidean space, usually denoted by II (read

“two”). Together with the first fundamental form, it serves to define extrinsic invariants of the surface, its principal curvatures.

We start with the normal $N(q)$. There is a close relation between a unit sphere and a unit normal. We define the map

$$N : S \rightarrow S^2,$$

where S^2 is the unit 2D sphere. We call this the Gauss map of S . It follows that the differential of the Gauss map N is

$$dN_q : T_q(S) \rightarrow T_{N(q)}(S^2).$$

But since $T_q(S)$ and $T_{N(q)}(S^2)$ are parallel planes we can regard that

$$dN_q : T_q(S) \rightarrow T_q(S).$$

The linear operator dN_q is clearly related to changes of directional derivatives because the normal and all tangents are orthogonal. It follows that $|dN_q(r)|$ is the curvature in direction of the unit vector r .

Second fundamental form is a quadratic form on the tangent plane. It is similar to the first fundamental form an inner product. Given a vector $\mu = dS_p(\hat{\mu}) \in T_p(S)$, where $\hat{\mu} = (a, b)$ The second fundamental form is

$$\begin{aligned} II_p(\mu) &= -\langle dN_q(\mu), \mu \rangle \\ &= -\langle N_u a + N_v b, S_u a + S_v b \rangle \\ &= a^2 \langle N_u, S_u \rangle + ab (\langle N_u, S_v \rangle + \langle N_v, S_u \rangle) + b^2 \langle N_v, S_v \rangle \\ &= a^2 e + 2ab f + b^2 g, \end{aligned}$$

where e, f and g are the coefficients of the second fundamental form. We also know that $\langle N, S_u \rangle = \langle N, S_v \rangle = 0$. The derivatives in both u and v is therefore also zero. It follows that we get

$$\langle N, S_u \rangle_u = \langle N, S_{uu} \rangle + \langle N_u, S_u \rangle = 0.$$

and so on.

Second fundamental form

The coefficients of the first fundamental form are

$$\begin{aligned} e &= \langle N, S_{uu} \rangle = -\langle N_u, S_u \rangle, \\ f &= \langle N, S_{uv} \rangle = -\langle N_u, S_v \rangle = -\langle N_v, S_u \rangle, \\ g &= \langle N, S_{vv} \rangle = -\langle N_v, S_v \rangle. \end{aligned} \tag{9.15}$$

The second fundamental form, represented as a symmetric matrix.

$$II_p(\mu) = \mu^T \begin{pmatrix} e & f \\ f & g \end{pmatrix} \mu.$$

where $\mu \in T_q(S)$, i.e in the the tangent plane of S in the point $q = S(p)$.

We have the following definition related to curvature:

- The Gaussian curvature $K = \det(dN_q)$ of S .
- The maximum normal curvature k_1 and the minimum normal curvature k_2 are called the principal curvatures of S at q . It follows that $dN_q(e_1) = k_1 e_1$ and $dN_q(e_2) = k_2 e_2$. e_1 and e_2 are orthogonal to each other, they are called principal directions at q and are eigenvectors to dN_q . k_1 and k_2 are thus eigenvalues of dN_q .
- The mean curvature of S at q is $H = \frac{k_1+k_2}{2}$.
- It follows that the Gaussian curvature $K = k_1 k_2$.

By computations (see [46], page 154-156) we get the following formulas:

$$\begin{aligned} K &= \frac{eg - f^2}{EG - F^2} \\ H &= \frac{1}{2} \frac{eG - 2fF + gE}{EG - F^2} \\ k &= H \pm \sqrt{H^2 - K} \\ dN &= \frac{-1}{EG - F^2} \begin{pmatrix} e & f \\ f & g \end{pmatrix} \begin{pmatrix} G & -F \\ -F & E \end{pmatrix} = \frac{-1}{EG - F^2} \begin{pmatrix} fF - eG & eF - fE \\ gF - fG & fF - gE \end{pmatrix}. \end{aligned}$$

9.2 Surface of revolution

This includes most of the classical surfaces as cone, cylinder, sphere and torus. The basic construction is:

- We start with a curve $c(u)$ in \mathbb{R}^2 ,

$$c(u) = \begin{pmatrix} c_x(u) \\ c_y(u) \end{pmatrix} \quad (9.16)$$

- Then we decide the axis to rotate around. If we choose the z-axis we get

$$S(u, v) = \begin{pmatrix} c_x(u) \cos v \\ c_x(u) \sin v \\ c_y(u) \end{pmatrix} \quad (9.17)$$

- Finally we choose the domain. In u-direction is it uncomplicated and we can choose any positive real number, but in v-direction it has to be $[a, b]$ if $0 < b - a < 2\pi$ (not complete rotation) or $[a, b]$ if $b - a = 2\pi$ (complete rotation).

We have already seen two surfaces of revolution, a cylinder, equation (9.2), and a torus, equation (9.3). Both equations fits the recipe of making a surface of revolution, equations (9.16) and (9.17).

Another example is shown in figure 9.4. In the figure is the curve $c(u)$ shown on right hand side. The curve is a second degree Bezier curve and we can see the control polygon

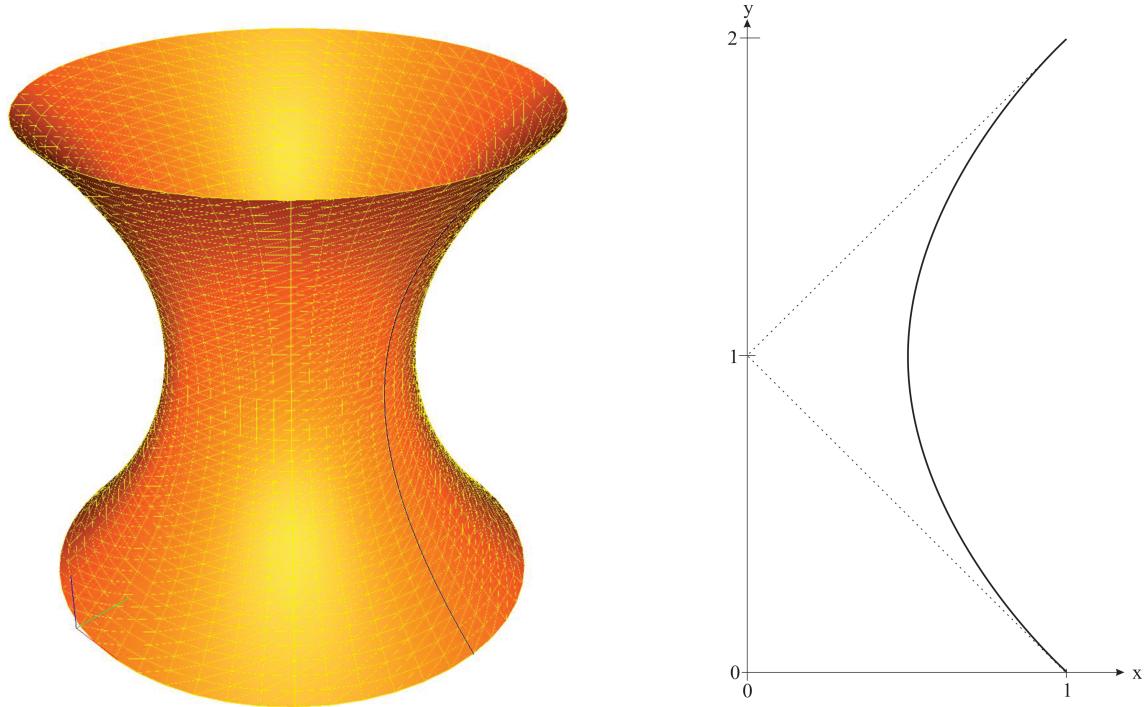


Figure 9.4: An example of a surface of revolution. The curve on right hand side is rotated 360° around it's y-axis to get the surface shown on left hand side.

and thus the coefficients (control points) in the figure. It follows that the curve is

$$c(u) = \begin{pmatrix} 1 \\ 0 \end{pmatrix}(1-u)^2 + \begin{pmatrix} 0 \\ 1 \end{pmatrix}2u(1-u) + \begin{pmatrix} 1 \\ 2 \end{pmatrix}u^2, \quad u \in [0, 1],$$

and the surface is

$$S(u, v) = \begin{pmatrix} (1-2u+2u^2)\cos v \\ (1-2u+2u^2)\sin v \\ 2u \end{pmatrix} \quad u \in [0, 1], \quad v \in [0, 2\pi].$$

9.3 Surface by sweeping

Imagine that we have two curves, and that we want to sweep one of the curves, g - called a cross section or profile curve, along the other curve, h - called a spine curve.

$$s(u, v) = h(v) + A(v)(g(u) - h(v_0)), \quad (9.18)$$

where $A(v)$ is an orthonormal 3×3 matrix,

$$A(v) = [t(v) \quad f_2(v) \quad f_3(v)], \quad (9.19)$$

where $t(v)$ is the unit tangent vector to the spine curve h ,

$$t(v) = \frac{h'(v)}{|h'(v)|}, \quad (9.20)$$

and $f_2(v)$ and $f_3(v)$ are unit vectors normal to each other and $t(v)$.

The matrix $A(v)$ is a rotation matrix describing how to rotate the profile curve when we move along the spine curve. We want the profile curve to keep the orientation according to the spine curve. We therefor need the first column vector to be the tangent vector of the spine curve. There are however several choices of rotation around the tangent vector, and we shall look at two different choices.

Frenet frame (also called TNB frame) is one choice. Here we denote the column of the matrix by

$$A(v) = [T \quad N \quad B]. \quad (9.21)$$

where

$$T = t(v) = \frac{h'}{|h'|}$$

and

$$B = T \wedge N = \frac{h' \wedge h''}{|h' \wedge h''|} \quad \text{and} \quad N = \frac{T'}{|T'|} = B \wedge T.$$

The partial derivatives of a surface by sweeping are

$$\begin{aligned} S_u &= A(v)g'(u) \\ S_v &= h'(v) + A'(v)(g(u) - h(v_0)) \end{aligned} \quad (9.22)$$

To compute $A'(v)$ using Frenet frame, we can use the Frenet-Serret formulas,

$$\frac{d}{dv} [T \quad N \quad B] = |h'| [T \quad N \quad B] \begin{bmatrix} 0 & -\kappa & 0 \\ \kappa & 0 & -\tau \\ 0 & \tau & 0 \end{bmatrix} = |h'| [\kappa N \quad \tau B - \kappa T \quad -\tau N].$$

where κ is the curvature of the spine curve h , and τ is the torsion.

The formulas for computing the curvature and the torsion are

$$\kappa = \frac{|h' \wedge h''|}{|h'|^3}, \quad \tau = \frac{\langle (h' \wedge h''), h''' \rangle}{|h' \wedge h''|^2}.$$

Proofs of all formulas connected to Frenet frames can be found from page 130 in [69].

There is a problem connected to the use of Frenet frame. If h'' is zero or parallel to h' at a point on the spine curve h , then we do not have an orthonormal matrix at that point and we also might get a jump in the rotation over the point. The surface might even look strange if h'' is close to zero or to be parallel to h' .

It is not easy to avoid singularities, but it is possible to make the rotation smoother. Another choice is therefor using rotation minimizing frames, RMF, developed and discussed in [87], [73], [102] and [143].

In relation to the Frenet frame (9.21), the RMF (9.19) can be represented as

$$\tilde{A}(v) = A(v) R(v). \quad (9.23)$$

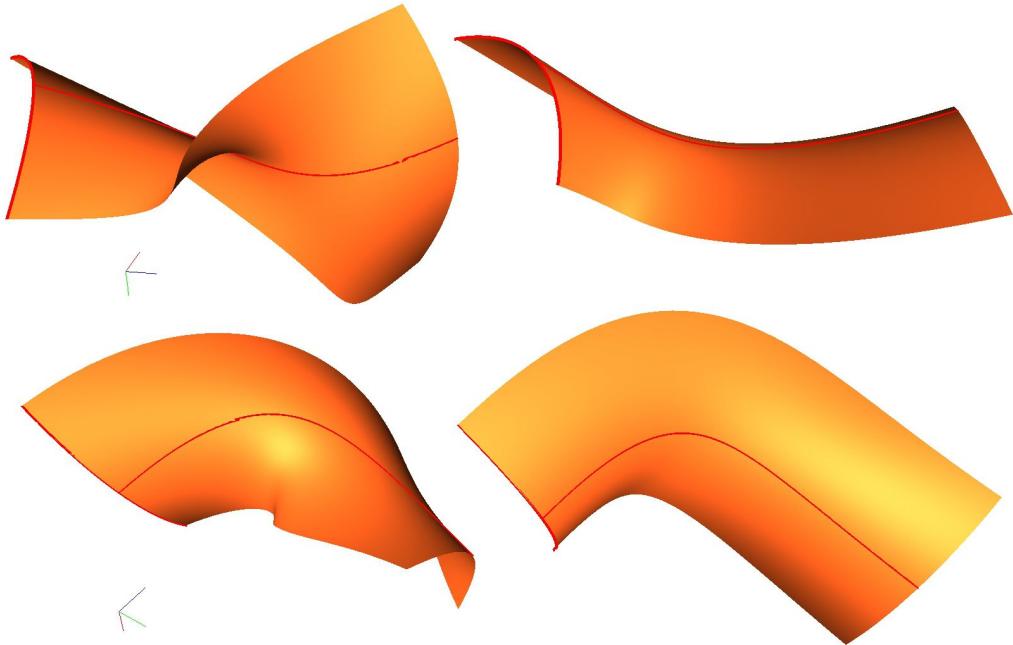


Figure 9.5: On left hand side is there two different views of a surface by sweeping using TNB-frame. On right hand side is there two views of a surface by sweeping using RMF-frame. The same profile and spine curves are used for both surfaces.

where $A(v)$ is defined in (9.21) and the orthonormal matrix

$$R(v) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \omega & -\sin \omega \\ 0 & \sin \omega & \cos \omega \end{bmatrix}. \quad (9.24)$$

with a rotation angle $\omega = \omega(v)$ specifying the rotation around the T -axis, i.e. the difference between TNB and RMF. The angle ω can be computed from the integral formula

$$\omega(v) = \omega_0 - \int_{v_0}^v \tau(t)|h'(t)|dt, \quad (9.25)$$

with an integration constant ω_0 , giving an initial rotation.

To compute the derivative, $\tilde{A}'(v)$, we get

$$\tilde{A}'(v) = A'(v)R(v) + A(v)R'(v). \quad (9.26)$$

where

$$R'(v) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & -\omega' \sin \omega & -\omega' \cos \omega \\ 0 & \omega' \cos \omega & -\omega' \sin \omega \end{bmatrix}. \quad (9.27)$$

and

$$\omega'(v) = \tau(v)|h'(v)|. \quad (9.28)$$

In figure 9.5 is one profile curve, a curve along one edge of the surfaces, and one spine curve, a curve from one edge to the opposite edge in the middle of the surfaces, marked

with red. Two different surfaces are made by the same profile and spine curve, one on left hand side and one on right hand side. The surface on left hand side is made using a TNB-frame (Frenet frame) along the spine curve, and the surface on right hand side is made by using a RMF frame (rotation minimizing frame) along the spine curve. In this case is $\omega_0 = 0$ chosen, see (9.25). Both surfaces are shown in two different views to give a better impression of them. The surface on right hand side is obvious much smoother than the other one.

9.4 Surfaces based on Curve constructions

Recall the Curve constructions, Hermite, Bezier, B-splines etc. All these constructions are based on blending points or points and vectors by using different types of blending functions. We can make a surface in the same way by just replacing the points (and vectors) by curves.

We then have the following general construction, similar to the general curve construction in (4.10),

$$S(u, v) = \sum_{i=1}^n b_i(u) c_i(v). \quad (9.29)$$

Here all curves must have the same domain,

$$c_i(v) : I \subset \mathbb{R}^2 \rightarrow \mathbb{R}^3, \quad i = 1, 2, \dots, n,$$

and all basis functions $b_i(u)$, $i = 1, 2, \dots, n$, must be linearly independent to each other and span a finite dimensional function space.

The differentiation is straight forward, and the partial derivatives are thus

$$\begin{aligned} S_u &= \sum_{i=1}^n b'_i(u) c_i(v), \\ S_v &= \sum_{i=1}^n b_i(u) c'_i(v), \\ S_{uv} &= \sum_{i=1}^n b'_i(u) c'_i(v). \end{aligned}$$

Alternatively we can swap the parameters,

$$\bar{S}(u, v) = \sum_{i=1}^n b_i(v) c_i(u). \quad (9.30)$$

In the rest of the chapter will a bar over the map of a surface based on curve construction denote a swap of the parameters between the curves and the basis functions.

9.5 Boolean sum surface

Boolean sum surface is based on equating a surfaces to a Boolean sum set. If $A(F)$ and $B(F)$ are two set operators acting on F , then the Boolean sum $(A \oplus B)(F)$ is defined as the union of the two sets, which contains the intersection set, $A(B(F)) = B(A(F)) = AB(F)$, only once. Hence

$$(A \oplus B)(F) = A(F) + B(F) - AB(F).$$

Imagine that we know the boundary of a surface, represented by 4 connected curves. To make a surface with this known boundary, we can use a method called Coons Patch, bilinear blends.

When we are modeling shapes, we often want smooth surfaces. If we put together several surfaces, we therefor often want, not only that the surfaces are connected, but also that the derivative over an edge is the same on two adjacent surfaces. This we can implement using Coon Patch, bicubically blends (see [25]).

Another possibilities is that we know a net of curves describing a surface. The curves in one direction intersect all curves in the other direction. A surface made by these curves is called Gordon surface (see [68]).

The construction is the same for these three types of blending and can be adapted to any other type of boolean sum surfaces. The fundamental construction is:

- We first make a surface on "surface based on curve construction", where the curves is located in the first parameter direction. The type of basis function to be used decides the type and order of the curves we have to use. We call this surface $S_1(u, v)$, the set of basis functions for $\{b_i\}$.
- We then make a new surface on "surface based on curve construction", where the curves is located in the second parameter direction. The type of basis function to be used decides the type and order of the curves we have to use. We call this surface $\bar{S}_2(u, v)$, the set of basis functions for $\{\bar{b}_i\}$.
- Finally make a surface where $\{\bar{b}_i\}$ are the basis functions in the first direction and $\{b_i\}$ are the basis functions in the second direction. The coefficients depend of the type of basis functions. It might be the corner points, or both the point, the first derivatives and the cross derivative in all corners. We call this surface $S_3(u, v)$.
- A Boolean sum surface is defined as,

$$S(u, v) = S_1(u, v) + \bar{S}_2(u, v) - S_3(u, v). \quad (9.31)$$

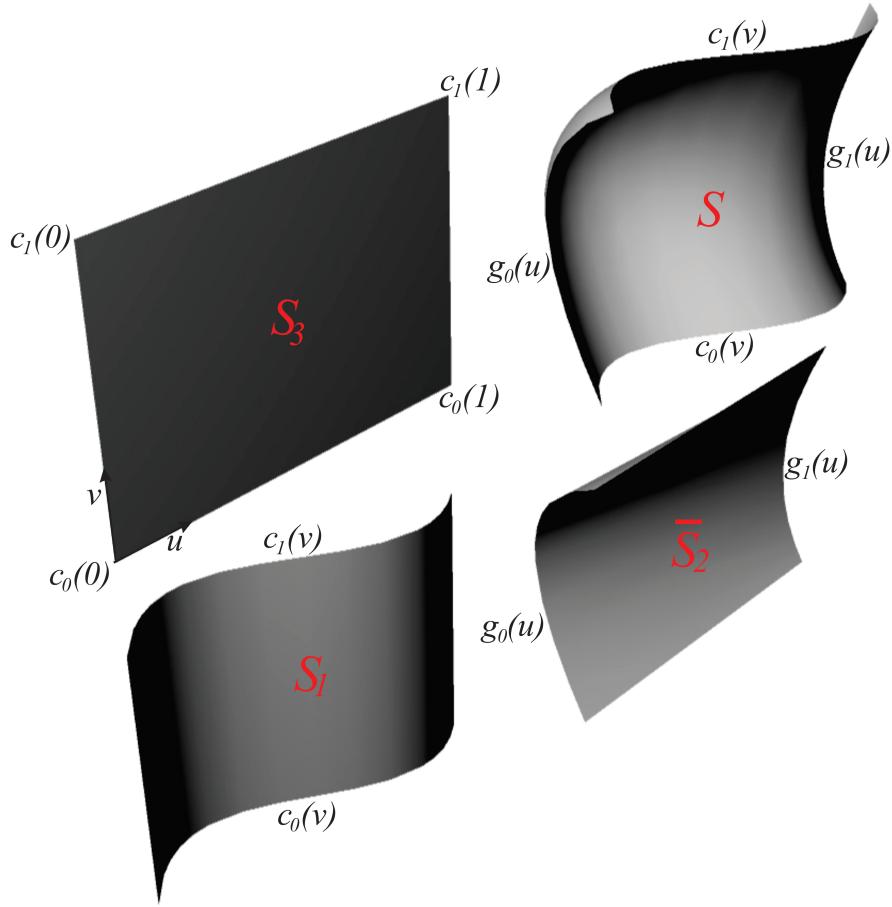


Figure 9.6: Example of Coons patch. The surface S is a Boolean sum of the three surfaces S_1 , \bar{S}_2 and S_3 , i.e. $S = S_1 + \bar{S}_2 - S_3$.

9.5.1 Coons Patch, bilinear blending

Steven Anson Coons¹ introduced the Coons patch in 1964, [25]. The Coons patch approach is based on the premise that a patch can be described in terms of four distinct boundary curves. A simple approach is the bilinearly blended Coons patch. It is based on four boundary curves, i.e. two sets, where the first set is

$$c_0(v) \quad \text{and} \quad c_1(v), \quad v \in [0, 1], \quad \text{see Figure 9.6,}$$

and the second set is

$$g_0(u) \quad \text{and} \quad g_1(u), \quad u \in [0, 1], \quad \text{see Figure 9.6,}$$

The curves must be connected, and it is therefore an assumption that

¹ Steven Anson Coons (1900 – 1979) was a professor at the Massachusetts Institute of Technology in the mechanical engineering department. During world war II, he worked on the design of aircraft surfaces, developing the mathematics to describe generalized "surface patches". Later in the 1960is he published several works on what today is known as Coons Patch.

$$\begin{aligned} c_0(0) &= g_0(0), & c_0(1) &= g_1(0), \\ c_1(0) &= g_0(1), & c_1(1) &= g_1(1). \end{aligned} \quad (9.32)$$

We first makes three surfaces by linear interpolation in u direction (surface S_1), in v direction (surface \bar{S}_2) and finally bilinear interpolation (both directions) at the corner points (surface S_3),

$$\begin{aligned} S_1(u, v) &= \begin{pmatrix} c_0(v) & c_1(v) \end{pmatrix} \begin{pmatrix} 1-u \\ u \end{pmatrix}, \\ \bar{S}_2(u, v) &= \begin{pmatrix} 1-v & v \end{pmatrix} \begin{pmatrix} g_0(u) \\ g_1(u) \end{pmatrix}, \\ S_3(u, v) &= \begin{pmatrix} 1-v & v \end{pmatrix} \begin{pmatrix} c_0(0) & c_1(0) \\ c_0(1) & c_1(1) \end{pmatrix} \begin{pmatrix} 1-u \\ u \end{pmatrix}. \end{aligned} \quad (9.33)$$

Note that in S_1 is defined as an inner product between a vector of vectors and a vector of scalars, \bar{S}_2 is defined as an inner product between a vector of scalars and a vector of vectors, and S_3 is defined as a product of a vector of scalars times a matrix of vectors times a vector of scalars. To make the final surface (S) we just use a boolean sum of the three surfaces, i.e.

$$S = S_1 + \bar{S}_2 - S_3,$$

where S_1 , \bar{S}_2 and S_3 are defined in (9.33). The process is clearly illustrated in Figure 9.6.

As a control we compute the edges,

$$\begin{aligned} S(u, 0) &= S_1(u, 0) + \bar{S}_2(u, 0) + S_3(u, 0) \\ &= (1-u) c_0(0) + u c_1(0) + g_0(u) - (1-u) c_0(0) - u c_1(0) \\ &= g_0(u), \\ S(u, 1) &= S_1(u, 1) + \bar{S}_2(u, 1) + S_3(u, 1) \\ &= (1-u) c_0(1) + u c_1(1) + g_1(u) - (1-u) c_0(1) - u c_1(1) \\ &= g_1(u), \\ S(0, v) &= S_1(0, v) + \bar{S}_2(0, v) + S_3(0, v) \\ &= c_0(v) + (1-v) g_0(0) + v g_1(0) - (1-v) c_0(0) - v c_0(1) \\ &= c_0(v), \\ S(1, v) &= S_1(1, v) + \bar{S}_2(1, v) + S_3(1, v) \\ &= c_1(v) + (1-v) g_0(1) + v g_1(1) - (1-v) c_1(0) - v c_1(1) \\ &= c_1(v). \end{aligned}$$

This shows that the surface satisfies the requirement that the boundary curve that was given really is the boundary curves of the surface, if the requirements in (9.32) is fulfilled.

9.5.2 Coons Patch, bicubically blending

It exist also a method of constructing Coons Patch that give higher degrees of control of the boundary, i.e. which also give the derivative in v direction on the boundary where

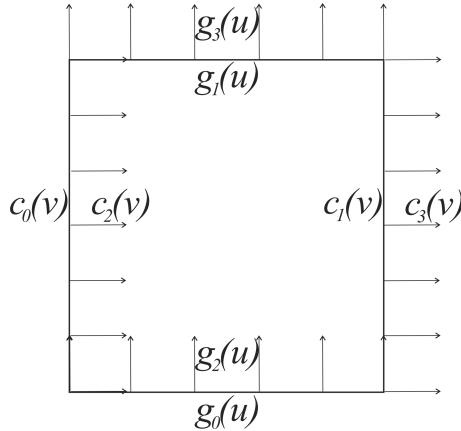


Figure 9.7: Four boundary curves $c_0(v)$, $c_1(v)$, $g_0(u)$ and $g_1(u)$, and four vector valued functions $c_2(v)$, $c_3(v)$, $g_2(u)$ and $g_3(u)$, describing the derivatives orthogonal to the boundary curves.

u is varying, and the derivative in u direction on the boundary where v is varying. This is called a bicubically blended Coons patch, and is constructed in the same way as the bilinearly blended Coons patch. The difference is that we now have to use an Hermite interpolation rule instead of linear interpolation.

We need four boundary curves, i.e. two sets, where the first set is

$$c_0(v) \quad \text{and} \quad c_1(v), \quad v \in [0, 1],$$

and the second set is

$$g_0(u) \quad \text{and} \quad g_1(u), \quad u \in [0, 1].$$

Then we have the two vector valued functions that gives the derivatives in u direction along the boundary curves in v direction,

$$c_2(v) \quad \text{and} \quad c_3(v), \quad v \in [0, 1].$$

And the two vector valued functions that gives the derivatives in v direction along the boundary curves in u direction,

$$g_2(u) \quad \text{and} \quad g_3(u), \quad u \in [0, 1].$$

The curves must be connected and consistent with each other, and therefore it is an assumption that

$$\begin{aligned} c_0(0) &= g_0(0), & c_0(1) &= g_1(0), \\ c_1(0) &= g_0(1), & c_1(1) &= g_1(1). \\ c'_0(0) &= g_2(0), & c'_0(1) &= g_3(0), \\ c'_1(0) &= g_2(1), & c'_1(1) &= g_3(1), \\ g'_0(0) &= c_2(0), & g'_0(1) &= c_3(0), \\ g'_1(0) &= c_2(1), & g'_1(1) &= c_3(1). \end{aligned} \tag{9.34}$$

The connection between the curves and the vector valued functions is shown in Figure

9.7. Before we start constructing Coons patch bicubic blending, we define the vectors

$$\begin{aligned}\mathbf{H}(t) &= [H_1(t), \quad H_2(t), \quad H_3(t), \quad H_4(t)], \\ \mathbf{c}(t) &= [c_1(t), \quad c_2(t), \quad c_3(t), \quad c_4(t)], \\ \mathbf{g}(t) &= [g_1(t), \quad g_2(t), \quad g_3(t), \quad g_4(t)].\end{aligned}$$

We now makes three surfaces by Hermite interpolation in u direction (surface S_1), in v direction (surface \bar{S}_2) and finally makes an Hermite surface using the position, the partial derivatives for u and v and the cross derivatives in all four corners. (surface S_3),

$$\begin{aligned}S_1(u, v) &= \langle \mathbf{c}(v), \mathbf{H}(u) \rangle, \\ \bar{S}_2(u, v) &= \langle \mathbf{H}(v), \mathbf{g}(u) \rangle, \\ S_3(u, v) &= \mathbf{H}(v) \mathbf{M} \mathbf{H}(u)^T,\end{aligned}\tag{9.35}$$

where the matrix \mathbf{M} is

$$\mathbf{M} = \begin{pmatrix} c_0(0) & c_1(0) & c_2(0) & c_3(0) \\ c_0(1) & c_1(1) & c_2(1) & c_3(1) \\ c'_0(0) & c'_1(0) & a_{11} & a_{12} \\ c'_0(1) & c'_1(1) & a_{21} & a_{22} \end{pmatrix} = \begin{pmatrix} g_0(0) & g_0(1) & g'_0(0) & g'_0(1) \\ g_1(0) & g_1(1) & g'_1(0) & g'_1(1) \\ g_2(0) & g_2(1) & a_{11} & a_{12} \\ g_3(0) & g_3(1) & a_{21} & a_{22} \end{pmatrix}.$$

The four numbers on the lower right square are the cross derivatives to S_3 in the four corners. We will later show what these four values must be if the requirements at the edges shall be fulfilled. But the pattern also indicates what these values should be.

Recall the Hermite basis property described in (4.17),

$$\begin{aligned}\mathbf{H}(0) &= [1, \quad 0, \quad 0, \quad 0], \\ \mathbf{H}(1) &= [0, \quad 1, \quad 0, \quad 0], \\ \mathbf{H}'(0) &= [0, \quad 0, \quad 1, \quad 0], \\ \mathbf{H}'(1) &= [0, \quad 0, \quad 0, \quad 1].\end{aligned}$$

We first compute the edges

$$\begin{aligned}S(u, 0) &= S_1(u, 0) + \bar{S}_2(u, 0) - S_3(u, 0) \\ &= \langle \mathbf{c}(0), \mathbf{H}(u) \rangle + g_0(u) - \langle \mathbf{c}(0), \mathbf{H}(u) \rangle \\ &= g_0(u), \\ S(u, 1) &= S_1(u, 1) + \bar{S}_2(u, 1) - S_3(u, 1) \\ &= \langle \mathbf{c}(1), \mathbf{H}(u) \rangle + g_1(u) - \langle \mathbf{c}(1), \mathbf{H}(u) \rangle \\ &= g_1(u), \\ S(0, v) &= S_1(0, v) + \bar{S}_2(0, v) - S_3(0, v) \\ &= c_0(v) + \langle \mathbf{H}(v), \mathbf{g}(0) \rangle - \langle \mathbf{H}(v), \mathbf{g}(0) \rangle \\ &= c_0(v), \\ S(1, v) &= S_1(1, v) + \bar{S}_2(1, v) - S_3(1, v) \\ &= c_1(v) + \langle \mathbf{H}(v), \mathbf{g}(1) \rangle - \langle \mathbf{H}(v), \mathbf{g}(1) \rangle \\ &= c_1(v).\end{aligned}$$

This shows that the edges are as expected. The derivative in the opposite direction on the four edges must be equal to the four given vector valued functions, thus

$$\begin{aligned} S_v(u, 0) &= D_v S_1(u, 0) + D_v \bar{S}_2(u, 0) - D_v S_3(u, 0) \\ &= \langle \mathbf{c}'(0), \mathbf{H}(u) \rangle + g_2(u) - \langle (g_2(0), g_2(1), a_{11}, a_{12}), \mathbf{H}(u) \rangle \\ &= g_2(u), \end{aligned}$$

which requires that $a_{11} = c'_2(0)$ and $a_{12} = c'_3(0)$. Further is

$$\begin{aligned} S_v(u, 1) &= D_v S_1(u, 1) + D_v \bar{S}_2(u, 1) - D_v S_3(u, 1) \\ &= \langle \mathbf{c}'(1), \mathbf{H}(u) \rangle + g_3(u) - \langle (g_3(0), g_3(1), a_{21}, a_{22}), \mathbf{H}(u) \rangle \\ &= g_3(u), \end{aligned}$$

which requires that $a_{21} = c'_2(1)$ and $a_{22} = c'_3(1)$. Further is

$$\begin{aligned} S_u(0, v) &= D_u S_1(0, v) + D_u \bar{S}_2(0, v) - D_u S_3(0, v) \\ &= c_2(v) + \langle \mathbf{H}(v), \mathbf{g}'(0) \rangle - \langle \mathbf{H}(v), (c_2(0), c_2(1), a_{11}, a_{21}) \rangle \\ &= c_2(v), \end{aligned}$$

which requires that $a_{11} = g'_2(0)$ and $a_{21} = g'_3(0)$. Further is

$$\begin{aligned} S_u(1, v) &= D_u S_1(1, v) + D_u \bar{S}_2(1, v) - D_u S_3(1, v) \\ &= c_3(v) + \langle \mathbf{H}(v), \mathbf{g}'(1) \rangle - \langle \mathbf{H}(v), (c_3(0), c_3(1), a_{12}, a_{22}) \rangle \\ &= c_3(v). \end{aligned}$$

which requires that $a_{12} = g'_2(1)$ and $a_{22} = g'_3(1)$.

If the resulting surface shall fulfill the requirements, that the four edges and the orthogonal derivatives to the four edges shall be equal the given curves and vector valued functions, the following must be fulfilled:

$$\begin{aligned} a_{11} &= c'_2(0) = g'_2(0), \\ a_{12} &= c'_3(0) = g'_3(1), \\ a_{21} &= c'_2(1) = g'_3(0), \\ a_{22} &= c'_3(1) = g'_3(1). \end{aligned}$$

It also follows that the surface then will have the following cross derivative in the corners,

$$\begin{aligned} S_{uv}(0, 0) &= c'_2(0) + g'_2(0) - a_{11} = c'_2(0), \\ S_{uv}(1, 0) &= c'_3(0) + g'_3(1) - a_{12} = c'_3(0), \\ S_{uv}(0, 1) &= c'_2(1) + g'_3(0) - a_{21} = c'_2(1), \\ S_{uv}(1, 1) &= c'_3(1) + g'_3(1) - a_{22} = c'_3(1). \end{aligned}$$

9.5.3 Hermite blending surface

Coon Patch bicubically blending suffer from the fact that the cross derivative at the four corners are zero. This limited the application of the method strongly. Therefore, we

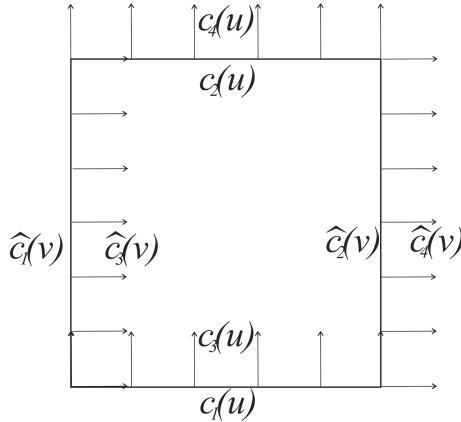


Figure 9.8: Four boundary curves $c_1(u)$, $c_2(u)$, $\hat{c}_1(v)$ and $\hat{c}_2(v)$, and four vector valued functions $c_3(u)$, $c_4(u)$, $\hat{c}_3(v)$ and $\hat{c}_4(v)$, describing the derivatives orthogonal to the boundary curves.

will investigate a surface construction where we blends two surfaces to one, by using a blending function with some special properties.

The problem to solve for the simplest version of this new method is the same as we can use Coon Patch bicubically blending to solve. We start with four curves, that are connected in such a way that they define a boundary of a surface. Two curves defines two opposite edges. It follows that the start point of one curve will be the start point of a curve in the other parameter direction, and that the end point will be the start point of the other curve in the other parameter direction. Beside this we have four vector function defining the derivative in opposite direction to the boundary curves. In Figure 9.8 is this illustrated. Two curves $c_1(u)$ and $c_2(u)$ defines two edges opposite to each other. The vector valued functions $c_3(u)$ and $c_4(u)$ are defining the derivatives in the other direction of these two edges. The two other curves $\hat{c}_1(v)$ and $\hat{c}_2(v)$ defines the two other edges. The vector valued functions $\hat{c}_3(v)$ and $\hat{c}_4(v)$ are defining the derivatives in the opposite direction of these last defined edges. It follows that these 8 curves (functions) have to fulfill the following requirements,

- i) $c_1(0) = \hat{c}_1(0)$, $c_1(1) = \hat{c}_2(0)$, $c_2(0) = \hat{c}_1(1)$ and $c_2(1) = \hat{c}_2(1)$,
- ii) $c'_1(0) = \hat{c}_3(0)$, $c'_1(1) = \hat{c}_4(0)$, $c'_2(0) = \hat{c}_3(1)$ and $c'_2(1) = \hat{c}_4(1)$,
- iii) $\hat{c}'_1(0) = c'_3(0)$, $\hat{c}'_1(1) = c_4(0)$, $\hat{c}'_2(0) = c_3(1)$ and $\hat{c}'_2(1) = c_4(1)$.

It now follows that we use hermite interpolation methods known from curve construction in section 4.3. First we make a surface using the curves and vector valued function in the u parameter direction,

$$\hat{S}(u, v) = \sum_{i=1}^4 H_i(v) c_i(u) \quad (9.36)$$

where H_i is defined in expression (4.16). Then we make a new surface using the curves

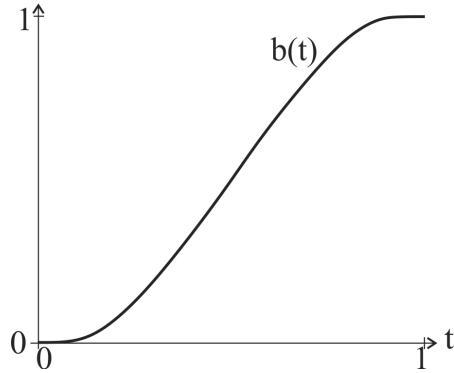


Figure 9.9: Example of a function $b(t)$ with properties described in the points **a**, **b**. The function is called an expo-rational B-splines, ERBS, and d is actually ∞ .

and vector valued function in the v parameter direction,

$$\check{S}(u, v) = \sum_{i=1}^4 H_i(u) c_i(v). \quad (9.37)$$

Given these two surfaces $\hat{S}(u, v)$ and $\check{S}(u, v)$ defined on a common domain $U = [0, 1] \times [0, 1]$. Define the distance surface

$$\tilde{S}(u, v) = \check{S}(u, v) - \hat{S}(u, v) \quad (9.38)$$

Define then the new blended surface

$$S(u, b) = \hat{S}(u, v) + B(u, v) \tilde{S}(u, v). \quad (9.39)$$

We will look at how we can construct $B(u, v)$, which conditions we can set to B to guarantee S to be $\in C^1(U)$. We will immediately make it more general, we will see which conditions we can set to B and \tilde{S} to guarantee S to be $\in C^d(U)$, $d \geq 1$.

First, we need an at least d -times differentiable monotonous function $b(t)$, $t \in [0, 1] \subset \mathbb{R}$, with the following properties:

- a)** $b(t)|_{t=0} = 0$,
- b)** $b(t)|_{t=1} = 1$,
- c)** $b^{(j)}(t)|_{t=0,1} = 0$, $j = 1, 2, \dots, d$,

We assume that $d \geq 1$. Several functions fulfilling these requirements will be discussed later, in Figure 9.9 is there a plot of a function where d actually is ∞ .

We start with a help function

$$g(u) = \begin{cases} \left(1 - \frac{1}{2}b(2u)\right), & \text{if } u \leq \frac{1}{2}, \\ \left(1 - \frac{1}{2}b(2-2u)\right), & \text{otherwise,} \end{cases} \quad (9.40)$$

which is a continuous and at least $C^d([0, 1])$ smooth function that is symmetric about $u = \frac{1}{2}$. It follows that

$$g'(u) = \begin{cases} -b'(2u), & \text{if } u \leq \frac{1}{2} \\ b'(2-2u), & \text{otherwise} \end{cases} \quad g''(u) = \begin{cases} -2b''(2u), & \text{if } u \leq \frac{1}{2}, \\ -2b''(2-2u), & \text{otherwise.} \end{cases}$$

The next two functions are, first

$$a(u) = 2u(1-u), \quad \text{where } a' = 2(1-2u), \quad \text{and } a'' = -4, \quad (9.41)$$

and then

$$t(u, v) = \begin{cases} \frac{v}{a(u)}, & \text{if } v < a(u) \\ \frac{1-v}{a(u)}, & \text{if } v > 1-a(u) \\ 1 & \text{otherwise} \end{cases} \quad (9.42)$$

We can see that $t(u, v)$ is continuous. It follows that

$$\begin{aligned} t_u &= \begin{cases} \frac{-v a'}{a^2}, & \text{if } v < a \\ \frac{-(1-v) a'}{a^2}, & \text{if } v > 1-a \\ 0, & \text{otherwise} \end{cases}, \quad t_v = \begin{cases} \frac{1}{a}, & \text{if } v < a \\ \frac{-1}{a}, & \text{if } v > 1-a \\ 0, & \text{otherwise} \end{cases} \\ t_{uu} &= \begin{cases} \frac{v(2(a')^2 - a''a)}{a^3}, & \text{if } v < a \\ \frac{(1-v)(2(a')^2 - a''a)}{a^3}, & \text{if } v > 1-a \\ 0, & \text{otherwise} \end{cases}, \quad t_{uv} = \begin{cases} \frac{-a'}{a^2}, & \text{if } v < a \\ \frac{a'}{a^2}, & \text{if } v > 1-a \\ 0, & \text{otherwise} \end{cases} \end{aligned} \quad (9.43)$$

and $t_{vv} = 0$.

Finally a blending function can be constructed,

$$B(u, v) = g(u) b \circ t(u, v). \quad (9.44)$$

It follows that

$$\begin{aligned} B_u &= g' b + g b' t_u, \\ B_v &= g b' t_v, \\ B_{uu} &= g'' b + 2g' b' t_u + g(b'' t_u^2 + b' t_{uu}), \\ B_{uv} &= g' b' t_v + g(b'' t_u t_v + b' t_{uv}), \\ B_{vv} &= g b'' t_v^2. \end{aligned} \quad (9.45)$$

One problem with the blending function B is that it does not behave symmetric in the sense that

$$B(u, v) + B(v, u) = 1 \quad (9.46)$$

One would expect that a blending function will behave the same way in the two parameter directions. To modify a blending function to behave like this we can define,

$$\bar{B}(u, v) = \frac{1}{2}(1 + B(u, v) - B(v, u)). \quad (9.47)$$

A plot of the blending functions $B(u, v)$ and $\bar{B}(u, v)$ can be seen in Figure 9.10. The difference is small, but $\bar{B}(u, v)$ is slightly smoother.

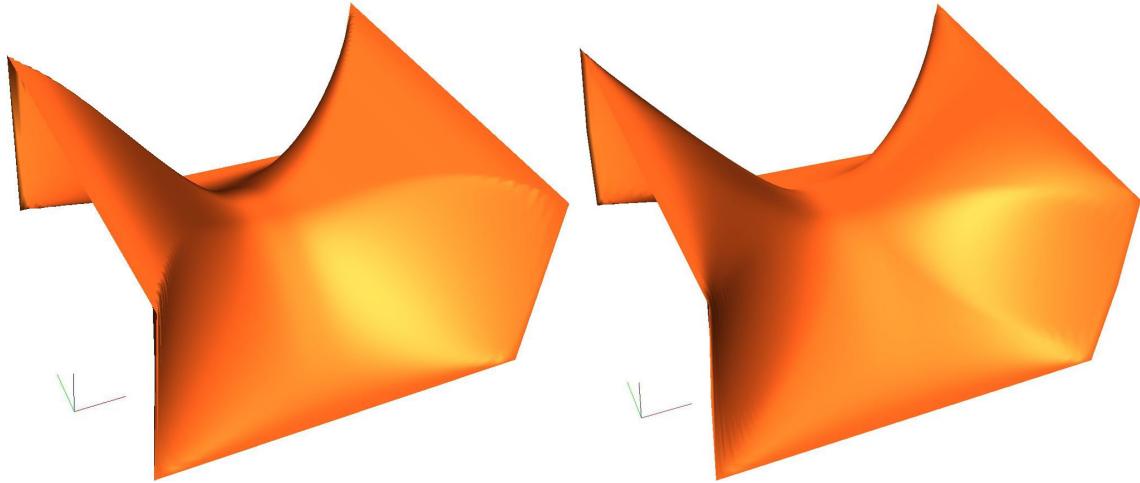


Figure 9.10: A plot of the blending function $B(u, v)$ defined in expression (9.44) on left hand side and $\bar{B}(u, v)$ defined in expression (9.46) on right hand side.

Both $B(u, v)$ and $\bar{B}(u, v)$ is designed for some special properties that is described in the following two lemmas. In the following we will only look at $B(u, v)$. But B and \bar{B} will have the same properties so all lemmas end theorem that applies to B will also apply to \bar{B} .

Lemma 9.1. At two opposite edges is the function value 0 (note that it does not include the corner points), at the other two edges is the function value 1 (including the corner points). i.e.

$$B(0, v) = B(1, v) = 1, \quad v \in [0, 1], \quad (9.48)$$

$$B(u, 0) = B(u, 1) = 0, \quad u \in (0, 1). \quad (9.49)$$

Further, at all corners and all edges has all derivatives up to order d the value zero,

$$\begin{aligned} D_u^{(i)} D_v^{(j)} B(0, v) &= D_u^{(i)} D_v^{(j)} B(1, v) = 0, \quad v \in [0, 1], \quad 0 \leq i, j \leq d, \quad i + j > 0, \\ D_u^{(i)} D_v^{(j)} B(u, 0) &= D_u^{(i)} D_v^{(j)} B(u, 1) = 0, \quad u \in (0, 1), \quad 0 \leq i, j \leq d, \quad i + j > 0. \end{aligned} \quad (9.50)$$

Proof. Both (9.48) and (9.49) follows from computing (9.40), (9.42) and (9.44). From (9.42) we see that $t(0, v) = t(1, v) = 1$, $v \in [0, 1]$, this because the edges $(0, v)$ and $(1, v)$ is between or on the two curves defined by $v = a(u)$ and $v = 1 - a(u)$.

It also follows, that by definition is all partial derivatives of all order zero on these edges, i.e.

$$D_u^{(i)} D_v^{(j)} t(0, v) = D_u^{(i)} D_v^{(j)} t(1, v) = 0, \quad v \in [0, 1], \quad i, j \geq 0, \quad i + j > 0. \quad (9.51)$$

Since B is a product of g and $b \circ t$ will any differentiation result in a set of terms containing either g or $g^{(j)}$. Since $g^{(j)}(0) = g^{(j)}(1) = 0$, $j = 1, \dots, d$ it follows that these terms will be zero. In the terms where one of the factor are g must the other factor be a differentiation. Because of the chain rule will there in these terms be a a partial derivative of t of some kind and order. Because of (9.51) will also these terms be zero. This confirms the upper part of (9.50) in Lemma 9.1.

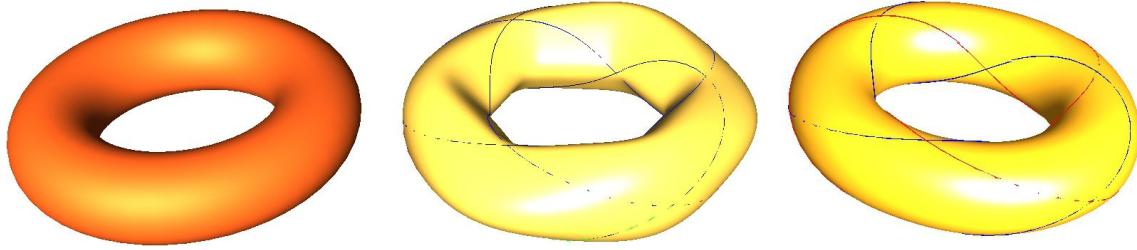


Figure 9.11: On left hand side is there a plot of a torus defined in expression 9.3. In the middle is there 12 surfaces that together makes a C^1 -smooth surface, approximating a torus. We can see the boundary curves that actually lies on a torus. The derivatives orthogonal to this boundary curves are also according to a torus. The 12 surfaces are made by using Coons Patch bicubically blending described in section 9.5.2. On right hand side is there also 12 surfaces that together makes a C^1 -smooth surface, approximating a torus. The boundary curves are also here on a torus, and the derivatives orthogonal to the curves are also here according to the torus. These surfaces are made by the new method described in this section.

From (9.42) we see that $t(u, 0) = t(u, 1) = 0$, $u \in (0, 1)$. It follows that $b \circ t(u, 0) = b \circ t(u, 1) = b^{(j)} \circ t(u, 0) = b^{(j)} \circ t(u, 1) = 0$, $j = 1, \dots, d$, $u \in (0, 1)$. In differentiation of B will every term contain either b or $b^{(j)}$, $j = 1, \dots, d$. Thus it follows that the lower part of (9.50) in Lemma 9.1 is true, which ends the proof \square

We finally look at the blending of two surfaces \hat{S} and \check{S} . From expression (9.39), we have:

$$S(u, v) = \hat{S}(u, v) + B(u, v) \check{S}(u, v),$$

where $\check{S}(u, v)$ is defined in (9.38). It follows that

$$\begin{aligned} S_u &= \hat{S}_u + B_u \check{S} + B \check{S}_u, \\ S_v &= \hat{S}_v + B_v \check{S} + B \check{S}_v, \\ S_{uu} &= \hat{S}_{uu} + B_{uu} \check{S} + 2 B_u \check{S}_u + B \check{S}_{uu}, \\ S_{uv} &= \hat{S}_{uv} + B_{uv} \check{S} + B_u \check{S}_v + B_v \check{S}_u + B \check{S}_{uv}, \\ S_{vv} &= \hat{S}_{vv} + B_{vv} \check{S} + 2 B_v \check{S}_v + B \check{S}_{vv}. \end{aligned} \tag{9.52}$$

In figure 9.11 is there on right hand side a plot of 12 surfaces, approximating a torus, made by this method.

To prove that $S(u, v)$ is in $C^d(U)$, $U = [0, 1] \times [0, 1] \subset \mathbb{R}^2$, we start with the following lemma.

Lemma 9.2. $B(u, v)$ is $\in C^d(V)$, $V = U \setminus (0, 0) \cup (1, 0) \cup (0, 1) \cup (1, 1)$.

- From the point $p_1 = (0, 0)$ in the parameter plane is B discontinuous in the direction $v_1 = (1, z)$, $0 \leq z < 2$
- From the point $p_2 = (1, 0)$ in the parameter plane is B discontinuous in the direction $v_2 = (-1, z)$, $0 \leq z < 2$

- From the point $p_3 = (0, 1)$ in the parameter plane is B discontinuous in the direction $v_3 = (1, -z), 0 \leq z < 2$
- From the point $p_4 = (1, 1)$ in the parameter plane is B discontinuous in the direction $v_4 = (-1, -z), 0 \leq z < 2$

Proof. The blending function $B(u, v) = g(u) b \circ t(u, v)$ consist of the function b described on page 281, the function g defined in (9.40) and the function t described in (9.42). The proof will analyze each of these factors and finally analyze the result. The properties of b is important and is also transferred to B . The proof is divided in 4 parts, 1) is about g , 2) is about t , 3) is about $b \circ t$ and 4) gives a conclusion.

1) From (9.40) we see that $g(u)$ is symmetric about $u = \frac{1}{2}$. It follows that the number of derivatives that are zero at $u = \frac{1}{2}$ decide the continuity level of g . It follows that $g^{(j)}(u)|_{u=\frac{1}{2}} = b^{(j)}(u)|_{u=\frac{1}{2}}$. Hence (from property **c**), page 281)

$$g(u) \in C^d([0, 1]).$$

2) Analyzing $t(u, v)$ in expression (9.42) we see that:

- For a fixed u -value, $\bar{u} \in [0, 1]$ (closed interval), is $\tilde{t}(v) = t(\bar{u}, v) \in C^0([0, 1])$. $\tilde{t}(v)$ is piecewise linear.
- For a fixed v -value, $\bar{v} \in (0, 1)$, (open interval), is $\hat{t}(u) = t(u, \bar{v}) \in C^0([0, 1])$. $\hat{t}(u)$ is piecewise smooth.
- For $v = 0$ we see that $\hat{t}(u) = t(u, 0) = \frac{0}{a(u)} = 0$ when $0 < u < 1$ (open interval).
- For $v = 1$ we see that $\hat{t}(u) = t(u, 1) = \frac{1-1}{a(u)} = 0$ when $0 < u < 1$ (open interval).

It follows that t is continuous on V (V defined in Lemma 9.2). We see that at the four corner points is t continuous in the v -direction, but discontinuous in u -direction.

- A closer investigation of the corner points shows the following:

a - Computing the formula (9.42) we get $t(p_1) = 1$.

b - From (9.42) it follows that $t(u, v) = 1$ for (u, v) in the area between the two curves in the parameter plane, defined by (9.41) and (9.42), $v = a(u)$ and $v = 1 - a(u)$, and that $t(u, v) < 1$ else. It follows that in the start point of the curve $v = a(u)$ is $(v', a')|_{(u,v)=p_1} = (1, 2)$. Investigating the directions $(1, z), 0 \leq z < 2$ we see that

$$\lim_{u \rightarrow 0+} t(u, z u) = \frac{z}{2}. \quad (9.53)$$

Contrary to paragraph a- above is $\lim_{u \rightarrow 0+} t(u, z u) < 1$. Hence $t(u, v)$ is discontinuous from the corner point $p_1 = (0, 0)$ in the direction $v_1 = (1, z), 0 \leq z < 2$.

Using symmetry it follows that $t(u, v)$ is discontinuous from the corner points

- $p_2 = (1, 0)$ in the direction $v_2 = (-1, z), 0 \leq z < 2$.
- $p_3 = (0, 1)$ in the direction $v_3 = (1, -z), 0 \leq z < 2$.
- $p_4 = (1, 1)$ in the direction $v_4 = (-1, -z), 0 \leq z < 2$.

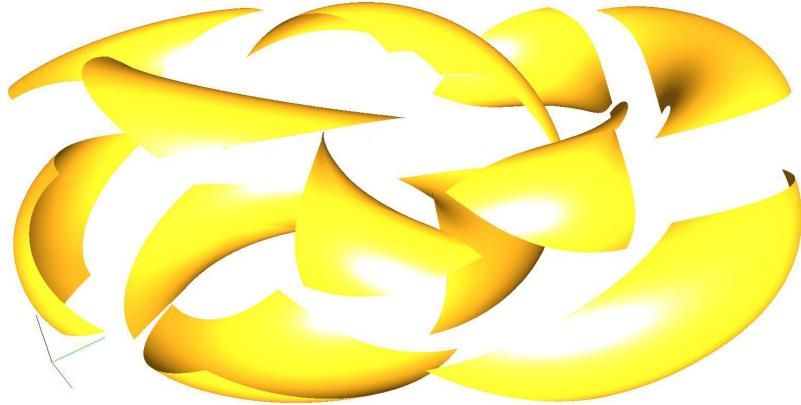


Figure 9.12: The 12 surfaces that together are approximating a torus. Here they are moved away from each other so we can see them separately.

3) Since $t(u, v)$ is continuous on V (V defined in Lemma 9.2), and discontinuous at the four corner points as described above, it follows that $b \circ t(u, v)$ is the same.

We see that $t(u, v)$ is divided in three parts, by two curves in the parameter plane, $v = a(u)$ and $v = 1 - a(u)$. Each part is internally smooth, continuous first, second, etc. derivatives. On the two curves we see the following,

$$\begin{aligned} t(u, a(u)) &= t(u, 1 - a(u)) = 1, \\ b^{(j)}(t(u, a(u))) &= b^{(j)}(t(u, 1 - a(u))) = 0, \quad j = 1, 2, \dots, d. \end{aligned}$$

It follows that $b \circ t(u, v)$ is not only continuous on V but is $C^d(V)$. This because every place a derivative of some order of t is discontinuous is the value of t equal 1 and all derivatives up to the current order is zero (this is illustrated for B in (9.45)).

4) It follows that $B(u, v)$ will inherit all discontinuities from both g and $b \circ t(u, v)$, hence $B(u, v)$ is in $C^d(V)$, but it is discontinuous from the points p_1 in direction v_1 , p_2 in direction v_2 , p_3 in direction v_3 and p_4 in direction v_4 , which ends the proof. \square

The following theorems will show that it is possible to fill a square hole with a surface so that the result has a desired degree of continuity.

Theorem 9.1. Given two surfaces \hat{S} and \check{S} defined on a common domain $U = [0, 1] \times [0, 1] \subset \mathbb{R}^2$ and a number d describing the degree of continuity. A surface $S(u, v) = B(u, v) \hat{S}(u, v) + (1 - B(u, v)) \check{S}(u, v)$ is in $C^d(U)$ if,

- a) the blending function B definition in 9.44 with properties described in (9.48) – (??) is in $C^d(V)$, V defined in Lemma 9.2,
- b) and the two surfaces $\hat{S}(u, v)$ and \check{S} both are $\in C^d(U)$,
- c) and that in the four corner points, the position and all derivatives up to order d , have the same values in both surfaces \hat{S} and \check{S} .

Proof. From Lemma 9.2 and the restriction b) on the surfaces \hat{S} and \check{S} , in Theorem 9.1, it follows that $S \in C^d(V)$, for V defined in Lemma 9.2.

Lemma 9.2 tells us that B is discontinuous from the four corner points, p_i , $i = 1, 2, 3, 4$, in given directions v_i , $i = 1, 2, 3, 4$. However, the behavior in the four corners are symmetrical or antisymmetric about the center point of the surface. We therefore only need to look at one of the corner points. We use the corner point $p_1 = (0, 0)$ for the investigation.

- We first examine the position at the corner point p_1 ,

$$S(0,0) = \hat{S}(0,0) + \tilde{S}(0,0) = \hat{S}(0,0) + 1(\check{S}(0,0) - \hat{S}(0,0)) = \check{S}(0,0). \quad (9.54)$$

To investigate the limit value when we move in direction v_1 towards the corner point p_1 on the surface S , we use expression (9.39), i.e.

$$S(u, zu) = \hat{S}(u, zu) + B(u, zu) \tilde{S}(u, zu), \quad 0 \leq z < 2.$$

It follows from (9.44) that

$$B(u, zu) \leq 1, \quad 0 \leq z < 2,$$

and together with the restriction **c**) in Theorem 9.1 it follows that

$$\lim_{u \rightarrow 0} |\tilde{S}(u, zu)| = 0.$$

Therefore we get

$$\lim_{u \rightarrow 0} S(u, zu) = \hat{S}(u, zu). \quad (9.55)$$

Because of restriction **c**) in Theorem 9.1 is $\hat{S}(0,0) = \check{S}(0,0)$, it follows from (9.54) and (9.55) that S is continuous on $U = [0, 1] \times [0, 1] \subset \mathbb{R}^2$.

- We then examine the first order derivatives at the corner point p_1 ,

$$S_u(0,0) = \hat{S}_u(0,0) + 1(\check{S}_u(0,0) - \hat{S}_u(0,0)) = \check{S}_u(0,0), \quad (9.56)$$

$$S_v(0,0) = \hat{S}_v(0,0) + 1(\check{S}_v(0,0) - \hat{S}_v(0,0)) = \check{S}_v(0,0). \quad (9.57)$$

To investigate the limit value when we move in direction v_1 towards the corner p_1 on the surface S , we use expression (9.5.3), i.e.

$$\begin{aligned} S_u(u, zu) &= \hat{S}_u(u, zu) + B_u(u, zu) \tilde{S}(u, zu) + B(u, zu) \tilde{S}_u(u, zu), \\ S_v(u, zu) &= \hat{S}_v(u, zu) + B_v(u, zu) \tilde{S}(u, zu) + B(u, zu) \tilde{S}_v(u, zu), \end{aligned}$$

where $0 \leq z < 2$. It follows from (9.44) that

$$B(u, zu) \leq 1, \quad 0 \leq z < 2,$$

and together with the restriction **c**) in Theorem 9.1 it follows that

$$\lim_{u \rightarrow 0+} |B(u, zu) \tilde{S}_u(u, zu)| = 0,$$

$$\lim_{u \rightarrow 0+} |B(u, zu) \tilde{S}_v(u, zu)| = 0,$$

From (9.40) we see that

$$\begin{aligned}\lim_{u \rightarrow 0^+} g(u) &= 1, \\ \lim_{u \rightarrow 0^+} g^{(j)}(u) &= 0, \quad j = 1, 2, \dots, d,\end{aligned}$$

and from expression (9.53) it follows that

$$\begin{aligned}\lim_{u \rightarrow 0^+} b \circ t(u, zu) &= b\left(\frac{z}{2}\right), \\ \lim_{u \rightarrow 0^+} b^{(j)} \circ t(u, zu) &= b^{(j)}\left(\frac{z}{2}\right),\end{aligned}$$

and from (9.43),

$$\begin{aligned}t_u(u, zu) &= \frac{zu \cdot 2(1-2u)}{(2u(1-u))^2} = \frac{1-2u}{(1-u)^2} \left(\frac{z}{2}\right) \frac{1}{u}, \\ t_v(u, zu) &= \frac{1}{2(1-u)} = \frac{1}{1-u} \left(\frac{1}{2}\right) \frac{1}{u}.\end{aligned}$$

Therefor it follows that

$$\begin{aligned}\lim_{u \rightarrow 0^+} B_u(u, zu) \tilde{S}(u, zu) &= \lim_{u \rightarrow 0^+} \left((g'b + gb't_u) \tilde{S}(u, zu) \right) \\ &= b'\left(\frac{z}{2}\right) \frac{z}{2} \lim_{u \rightarrow 0^+} \left(\left(\frac{1-2u}{(1-u)^2} \right) \frac{\tilde{S}(u, zu)}{u} \right), \\ &= b'\left(\frac{z}{2}\right) \frac{z\sqrt{1+z^2}}{2} \lim_{u \rightarrow 0^+} \frac{\tilde{S}(u, zu)}{u\sqrt{1+z^2}}, \\ &= k d\tilde{S}_{p_1}(\hat{v}_1),\end{aligned}$$

where $k = b'\left(\frac{z}{2}\right) \frac{z\sqrt{1+z^2}}{2}$ and $\hat{v}_1 = \frac{v_1}{|v_1|}$.

$d\tilde{S}_{p_1}(\hat{v}_1)$ is the directional derivatives at the point p_1 in direction \hat{v}_1 . From restriction **c**) in Theorem 9.1 it follows that, if $d > 0$ then all derivatives of order 1 has the same values in the two surfaces \hat{S} and \check{S} . Thus all directional derivatives, of order 1, to the surface \tilde{S} at p_1 must be zero-vectors. Hence

$$\lim_{u \rightarrow 0^+} |B_u(u, zu) S(u, zu)| = 0, \quad d > 0. \quad (9.58)$$

We now use the same method to treat the other partial derivatives.

$$\begin{aligned}\lim_{u \rightarrow 0^+} B_v(u, zu) \tilde{S}(u, zu) &= \lim_{u \rightarrow 0^+} \left(gb't_v \tilde{S}(u, zu) \right) \\ &= k d\tilde{S}_{p_1}(\hat{v}_1),\end{aligned}$$

where $k = b'\left(\frac{z}{2}\right) \frac{\sqrt{1+z^2}}{2}$ and $\hat{v}_1 = \frac{v_1}{|v_1|}$.

$d\tilde{S}_{p_1}(\hat{v}_1)$ is the directional derivatives at the point p_1 in direction \hat{v}_1 . It also now follows that

$$\lim_{u \rightarrow 0^+} |B_v(u, zu) S(u, zu)| = 0, \quad d > 0. \quad (9.59)$$

It follows from (9.65) and (9.59), that if $d > 0$, the partial derivatives are

$$\lim_{u \rightarrow 0^+} S_u(u, zu) = \hat{S}_u(u, zu), \quad 0 \leq z < 2, \quad (9.60)$$

$$\lim_{u \rightarrow 0^+} S_v(u, zu) = \hat{S}_v(u, zu), \quad 0 \leq z < 2. \quad (9.61)$$

Because of restriction **c**) in Theorem 9.1 is $\hat{S}_u(0, 0) = \check{S}_u(0, 0)$ and $\hat{S}_v(0, 0) = \check{S}_v(0, 0)$, it follows from (9.56), (9.57), (9.60), (9.61) that $S \in C^1(U)$, $U = [0, 1] \times [0, 1] \subset \mathbb{R}^2$.

- We finally examine the second and higher order derivatives at the corner point p_1 .

We know from Lemma 9.1 that

$$B(0, 0) = 1, \quad D_u^{(i)} D_v^{(j)} B(0, 0) = 0, \quad 0 \leq i, j \leq d, \quad i + j > 0. \quad (9.62)$$

It follows that

$$\begin{aligned} D_u^{(i)} D_v^{(j)} S(0, 0) &= D_u^{(i)} D_v^{(j)} \hat{S}(0, 0) + (D_u^{(i)} D_v^{(j)} \check{S}(0, 0) - D_u^{(i)} D_v^{(j)} \hat{S}(0, 0)) \\ &= D_u^{(i)} D_v^{(j)} \check{S}(0, 0), \quad 0 \leq i, j \leq d, \quad i + j > 0. \end{aligned} \quad (9.63)$$

Before we continue, we will look at a generalization of the derivative of the function $t(u, v)$. We know from (9.43) that

$$D_u t = -v \frac{a'}{a^2}, \quad (9.64)$$

where $a' \in \mathcal{P}^1$ (polynomials of degree most one) and $a \in \mathcal{P}^2$. Assume that

$$D_u^{(j)} t = -v \frac{r_j}{a^{j+1}}, \quad \text{where } r_j \in \mathcal{P}^j, \quad j \geq 0. \quad (9.65)$$

Than

$$\begin{aligned} D_u^{(j+1)} t &= -v \frac{r'_j a^{j+1} - (j+1)r_j a^j a'}{a^{2j+2}}, \\ &= -v \frac{r'_j a - (j+1)r_j a'}{a^{j+2}}, \\ &= -v \frac{r_{j+1}}{a^{j+2}}, \quad \text{where } r_{j+1} \in \mathcal{P}^{j+1}, \end{aligned}$$

which together with (9.64) shows that (9.65) is a general formula for partial derivatives with respect to u . From (9.43) we see that $r_0 = -1$ and $r_1 = a'$.

From (9.65) it follows that

$$D_v D_u^{(j)} t = -\frac{r_j}{a^{j+1}}, \quad j \geq 0. \quad (9.66)$$

Further it follows that

$$D_v^{(j)} D_u^{(i)} t = 0, \quad j > 1, i \geq 0. \quad (9.67)$$

To investigate the limit value when we move in direction v_1 towards the corner p_1 on the surface S , we use expression (4.47), i.e.

To investigate the direction v_1 towards the corner p_1 on the surface S we use expression (9.5.3),

$$\begin{aligned} S_u(u, zu) &= \hat{S}_u(u, zu) + B_u(u, zu) \tilde{S}(u, zu) + B(u, zu) \tilde{S}_u(u, zu), \\ S_v(u, zu) &= \hat{S}_v(u, zu) + B_v(u, zu) \tilde{S}(u, zu) + B(u, zu) \tilde{S}_v(u, zu), \end{aligned}$$

where $0 \leq z < 2$.

From (9.5.3) we have

$$\begin{aligned} t_{uu}(u, zu) &= \frac{zu (2(2(1-2u))^2 + 8u(1-u))}{(2u(1-u))^3} = \frac{1-3u+3u^2}{(1-u)^2} (z) \frac{1}{u^2}, \\ t_{uv}(u, zu) &= \frac{-2(1-2u)}{(2u(1-u))^2} = \frac{(1-2u)}{(1-u)^3} \left(-\frac{1}{2}\right) \frac{1}{u^2}, \\ t_{vv}(u, zu) &= 0. \end{aligned}$$

□

9.5.4 Curves on triangular surfaces

Given a 2D compact manifold, a surface S , that locally is homoeomorphic to \mathbb{R}^2 (in a neighborhood about every point). The surface is thus parameterized with local maps that overlap and together cover the entire surface.

The surface is topologically described by vertices which are ordered and arranged with edges connecting pairs of vertices. At each vertex p_i is there a "local surface" S_i . In the parameter plane of each of these local surfaces are the related vertex and (only) all neighboring vertices present. The neighboring vertices are the vertices that are connected to the central vertex of the local surface by an edge.

We thus have a point set

$$\{p_i\}_{i=1}^n, \quad p_i \in \mathbb{R}^3, \quad i < 01, \dots, n.$$

These points are vertices in a triangulation, where edges connects the vertices and two triangles share an edge.

The vertices and the edges makes a triangulation of the manifold.

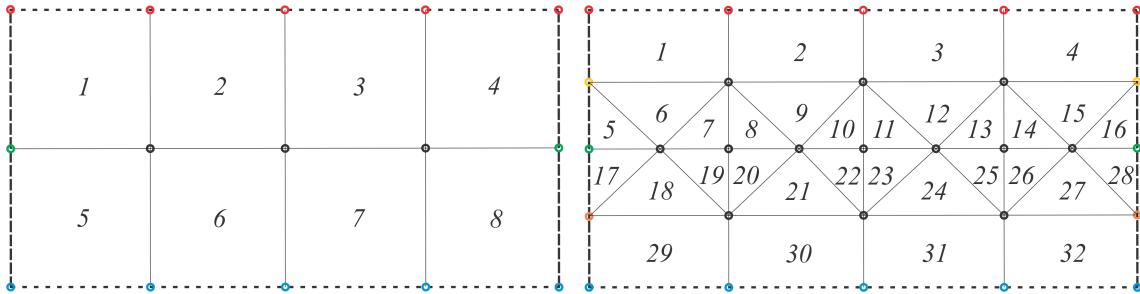


Figure 9.13: On top is two views of a curve on a torus displayed together with the torus. On the lower part is there three views of the same curve. But now also the first derivatives are shown in red, the second derivatives is shown in blue and the third derivatives is shown in green.

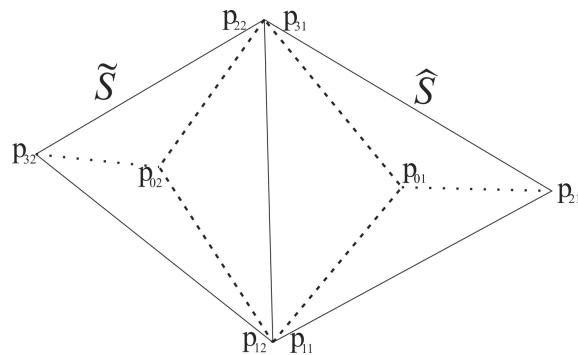


Figure 9.14: On top is two views of a curve on a torus displayed together with the torus. On the lower part is there three views of the same curve. But now also the first derivatives are shown in red, the second derivatives is shown in blue and the third derivatives is shown in green.

The problem to solve here comes from the modeling of 2D manifolds, ie surfaces that are compact but with a different genus. The surfaces are parameterized with local maps that together cover the entire surface, and there is a homeomorphism between local maps that overlap.

Given two neighboring triangular surface, where the position and all derivatives up to order d are equal in the two common corners. There is a common edge between the two triangles, and the surface is not necessarily smooth over the joint edge even if it is smooth in the corners. We will describe the construction of a dual set of surfaces that how curves defined in the parameter plane of these two triangles is mapped into \mathbb{R}^3 . In Figure ?? is one triangle defined by p_1 , p_2 , p_{32} and the other triangle by p_1 , p_{31} , p_2 . The two common corners are in p_1 and p_2 , and the common edge is $p_1 + t(p_2 - p_1)$, $t \in [0, 1]$. We denote the triangular surfaces by $S_1(u, v, w)$ and $S_2(u, v, w)$

Given a curve $h : I \subset \mathbb{R} \rightarrow \mathbb{R}^2$,

$$h(t) = \sum_{i=1}^n c_i b_i(t).$$

where c_i , $i = 1, \dots, n$ are points or vectors in \mathbb{R}^2 , but in barycentric coordinates. and $b_i(t)$

are basis functions spanning a finite dimensional function space.

The question is, what curve will we get when the curve in the parameter space is mapped into 3D-space. Given a surface $S : U \subset \mathbb{R}^2 \rightarrow \mathbb{R}^3$. The curve definition then is,

$$c(t) = S(h(t)) = S \circ h(t), \quad t \in I \subset \mathbb{R}. \quad (9.68)$$

Using the chain rule on (9.68) we get the first, second and third derivative,

$$\begin{aligned} c'(t) &= dS(h'), \\ c''(t) &= d(dS(h'))(h') + dS(h''), \\ c'''(t) &= d(d(dS(h'))(h'))(h') + d(dS(h'))(h'') + d(dS(h''))(h') + dS(h'''), \end{aligned} \quad (9.69)$$

where the differentials matrices, there are used in the derivatives above are,

$$\begin{aligned} d(dS(h')) &= d([S_u, S_v] h') \\ &= [[S_u, S_v] h']_u, [[S_u, S_v] h']_v \\ &= [[S_{uu}, S_{vu}] h', [S_{uv}, S_{vv}] h'], \end{aligned}$$

and

$$\begin{aligned} d(d(dS(h'))(h')) &= d([[S_{uu}, S_{vu}] h', [S_{uv}, S_{vv}] h'] h') \\ &= [[[S_{uu}, S_{vu}] h', [S_{uv}, S_{vv}] h'] h']_u, [[[S_{uu}, S_{vu}] h', [S_{uv}, S_{vv}] h'] h']_v \\ &= [[[S_{uuu}, S_{vuu}] h', [S_{uvu}, S_{vvu}] h'] h', [[[S_{uuv}, S_{vuv}] h', [S_{uvv}, S_{vvv}] h'] h']. \end{aligned}$$

In Figure 9.15 is a first degree Bezier curve, a straight line, expression (4.28), in the parameter plane of a torus, expression (9.3), mapped into \mathbb{R}^3 by expression (9.68). On top in Figure 9.15 is two different views of the curve together with the torus shown, on the lower part is there three different views of the curve and its first derivative (red), second derivatives (blue) and third derivatives (green) from expression (9.69).

Another example of “curves” on a surface is a vector valued function describing the directional derivatives along a curve, not in the curve direction, but perpendicular to the curve direction. In this case we need two vector valued functions in the parameter plane, one vector valued function describing the position in the parameter plane depending on the parameter, another vector valued function describing the vector in \mathbb{R}^2 to find the directional derivatives on the surface. Vi therefore must have the two vector valued functions,

$$h(t) = \sum_{i=1}^{n_1} c_i b_i(t), \quad (9.70)$$

$$r(t) = \sum_{i=1}^{n_2} d_i b_i(t), \quad (9.71)$$

both for $t \in I \subset \mathbb{R}$. The map from parameter space of the surface to 3D-space is then

$$\tilde{c}(t) = dS_{h(t)}(r(t)). \quad (9.72)$$

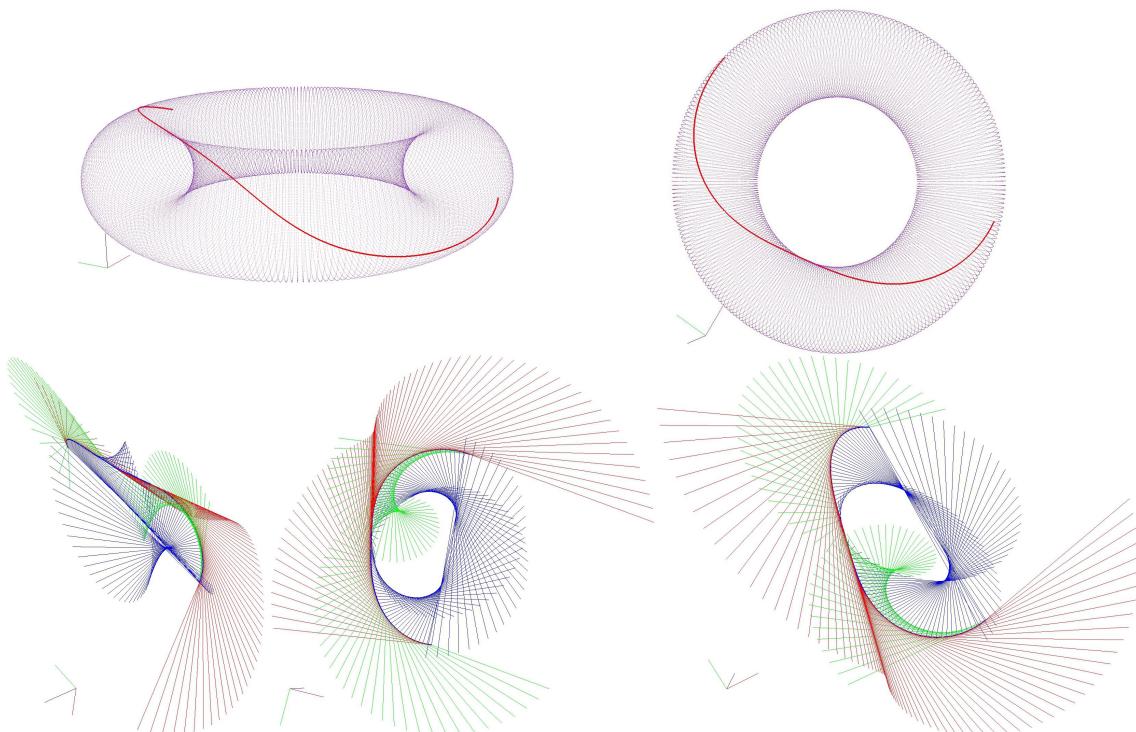


Figure 9.15: On top is two views of a curve on a torus displayed together with the torus. On the lower part is there three views of the same curve. But now also the first derivatives are shown in red, the second derivatives is shown in blue and the third derivatives is shown in green.

The derivatives are,

$$\begin{aligned}\tilde{c}'(t) &= d(dS(r))(h') + dS(r'), \\ \tilde{c}''(t) &= d(d(dS(r))(h'))(h') + d(dS(h'))(r') + d(dS(r'))(h') + dS(r''),\end{aligned}\tag{9.73}$$

where the differentials matrices are,

$$\begin{aligned}d(dS(r)) &= d([S_u, S_v] r) \\ &= [[S_u, S_v] r]_u, [[S_u, S_v] r]_v \\ &= [[S_{uu}, S_{vu}] r, [S_{uv}, S_{vv}] r],\end{aligned}$$

and

$$\begin{aligned}d(d(dS(r))(h')) &= d([[S_{uu}, S_{vu}] r, [S_{uv}, S_{vv}] r] h') \\ &= [[[S_{uu}, S_{vu}] r, [S_{uv}, S_{vv}] r] h']_u, [[[S_{uu}, S_{vu}] r, [S_{uv}, S_{vv}] r] h']_v \\ &= [[[S_{uuu}, S_{vuu}] r', [S_{uvu}, S_{vvu}] r] h', [[[S_{uuv}, S_{vuv}] r, [S_{uvv}, S_{vvv}] r] h']].\end{aligned}$$

An example of this type of vector valued function, where the expressions (9.70), (9.72) and (9.73) are used is given in section 9.3.

Chapter 10

Tensor Product Surfaces

For tensor product surfaces we have the following general formula using Expo-Rational B-splines (ERBS),

$$S(u, v) = \sum_{i=1}^{n_u} \sum_{j=1}^{n_v} s_{ij}(u, v) B_i(u) B_j(v), \quad (10.1)$$

where $s_{ij}(u, v)$, $i = 1, \dots, n_u$, $j = 1, \dots, n_v$, 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 therefore 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 10.1, 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 10.1, three of the local patches are plotted. The red cubes represent the control points of the Bézier patches. The knot vector in u direction is

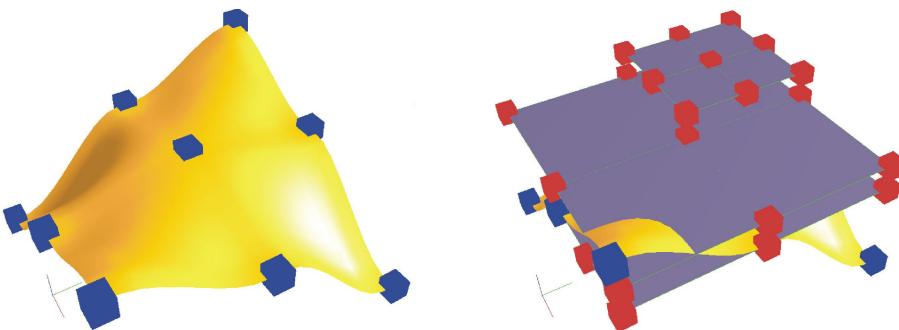


Figure 10.1: A global tensor product ERBS-surface with 9 local surfaces (interpolation points marked with blue cubes). All local surfaces are 2nd degree Bézier patches that are planar; some of them are viewed on the left hand side (the red cubes are control points).

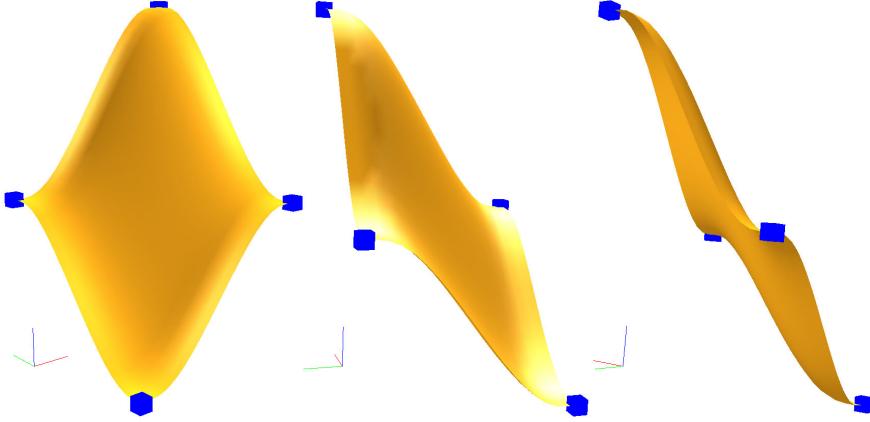


Figure 10.2: A global tensor product ERBS-surface with four local surfaces (interpolation points marked with blue cubes). All local surfaces are planar Bézier patches which are equal and also parallel. The surface is viewed from three different angles, and on the left hand side of each view, there is an RGB-frame showing the orientations.

$\{u_i\}_{i=0}^4$, 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 10.1, 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 10.2 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 7.5 the classification of local function supporting the basic properties, i.e. the C^∞ 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 [41]).

In the remainder of this chapter the following subjects will be discussed. In section 10.1, there are definitions of “open/closed” surfaces. Then, in section 10.2, surface evaluation including derivatives is discussed, and the section shows some algorithms for use in implementations. In section 10.3 Bézier patches are discussed as local surfaces. This includes 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, 10.4,

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, 10.5, computational aspects concerning ERBS tensor product surfaces are discussed.

10.1 Definition/implementation of “open/closed” Surfaces

We regard a tensor product ERBS surface to be a “2-dimensional object”, i.e. a parameterized 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 10.1. *A parameterized differentiable surface is a differentiable map $S : \Omega \rightarrow \mathbb{R}^n$ of an open set $\Omega \subset \mathbb{R}^2$ into \mathbb{R}^n for $n = 2, 3, \dots$.*

With this as a background the next definitions are introduced:

Definition 10.2. *“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 7.2, 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

$$\lim_{t \rightarrow a^+} D_u^j S(t, v) = D_u^j S(b, v), \quad \text{for } j = 0, 1, 2, \dots \quad \text{and } v \in [v_1, v_{n_v}]$$

or

$$\lim_{t \rightarrow a^+} D_v^j S(u, t) = D_v^j S(u, b), \quad \text{for } j = 0, 1, 2, \dots \quad \text{and } u \in [u_1, u_{n_u}]$$

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.

10.2 Evaluation, value and derivatives

The general expression for a tensor product “ERBS surface” (10.1) and some of its partial derivatives can be computed as follows,

$$\begin{aligned} S(u, v) &= \sum_{j=1}^{n_v} \sum_{i=1}^{n_u} s_{i,j}(u, v) B_i(u) B_j(v), \\ D_u S(u, v) &= \sum_{j=1}^{n_v} \sum_{i=1}^{n_u} (D_u s_{i,j}(u, v) B_i(u) B_j(v) + s_{i,j}(u, v) DB_i(u) B_j(v)), \\ D_v S(u, v) &= \sum_{j=1}^{n_v} \sum_{i=1}^{n_u} (D_v s_{i,j}(u, v) B_i(u) B_j(v) + s_{i,j}(u, v) Bi(u) DB_j(v)), \\ D_u D_v S(u, v) &= \sum_{j=1}^{n_v} \sum_{i=1}^{n_u} (D_u D_v s_{i,j}(u, v) B_i(u) B_j(v) + D_u s_{i,j}(u, v) Bi(u) DB_j(v) + \\ &\quad D_v s_{i,j}(u, v) DB_i(u) B_j(v) + s_{i,j}(u, v) DB_i(u) DB_j(v)), \end{aligned} \quad (10.2)$$

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 $D_u D_v = \frac{\partial^2}{\partial u \partial v}$. Note that $s_{i,j}(u, v)$, and all respective derivative vectors have in expression (10.2) 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

$$c_j(u, v) = \sum_{i=1}^{n_u} s_{i,j}(u, v) B_i(u), \quad \text{for } j = 1, 2, \dots, n_v. \quad (10.3)$$

It now follows that the first line of equation (10.2) can be reformulated by using (10.3).

We therefor get a new version of the first line of (10.2),

$$S(u, v) = \sum_{j=1}^{n_v} c_j(u, v) B_j(v). \quad (10.4)$$

Both equations, (10.3) and (10.4) are comparable with the curve formula (8.2). 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 7.52 we can simplify the first part of formula (10.3). For all knot intervals $j = 1, 2, \dots, n_v - 1$,

$$c_j(u, v) = \begin{cases} s_{i,j}(u, v), & \text{if } u = u_i \\ s_{i+1,j}(u, v) + (s_{i,j}(u, v) - s_{i+1,j}(u, v)) B_i(u), & \text{if } u_i < u < u_{i+1} \end{cases} \quad (10.5)$$

Computing the partial derivatives of (10.5) only according to u , we can see that for $u = u_i$, $i = 1, \dots, n_u$, all partial derivatives with respect to u are equal to the respective derivatives of the local surface, i.e.

$$D_u^{d_u} c_j(u, v) = D_u^{d_u} s_{i,j}(u, v), \quad \text{for } j = 1, \dots, n_u, \text{ and } d_u = 0, 1, 2, \dots \quad (10.6)$$

To simplify for all other $u \in [u_1, u_{n_u}]$, we first define

$$\hat{s}_{i,j}(u, v) = s_{i,j}(u, v) - s_{i+1,j}(u, v), \quad \text{if } u_i < u < u_{i+1}, \quad (10.7)$$

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,

$$\begin{aligned} c_j(u, v) &= s_{i+1,j}(u, v) + \hat{s}_{i,j}(u, v) B_i(u), \\ D_u c_j(u, v) &= D_u s_{i+1,j}(u, v) + D_u \hat{s}_{i,j}(u, v) B_i(u) + \hat{s}_{i,j}(u, v) D B_i(u), \\ D_u^2 c_j(u, v) &= D_u^2 s_{i+1,j}(u, v) + D_u^2 \hat{s}_{i,j}(u, v) B_i(u) + 2 D_u \hat{s}_{i,j}(u, v) D B_i(u) + \hat{s}_{i,j}(u, v) D^2 B_i(u). \end{aligned} \quad (10.8)$$

The expressions are close to being the same as what we used in the curve case (8.6). 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 (10.8) 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 (10.8) to a vector of vectors instead of only vectors. To expand formula (10.8) to also include derivatives in the v direction we have to first look at the first line of (10.8). It follows that

$$D_v^d c_j(u, v) = D_v^d s_{i+1,j}(u, v) + D_v^d \hat{s}_{i,j}(u, v) B_i(u), \quad \text{for } d = 1, 2, \dots,$$

and that in general for computing

$$D_u^{d_u} D_v^{d_v} c_j(u, v), \quad \text{for } d_u = 1, 2, \dots \text{ and } d_v = 1, 2, \dots,$$

on the right hand side in expression 10.8, we just have to replace

$$\begin{aligned} D_u^{d_u} s_{i+1,j}(u, v) &\quad \text{with} \quad D_u^{d_u} D_v^{d_v} s_{i+1,j}(u, v) \quad \text{and} \\ D_u^{d_u} \hat{s}_{i,j}(u, v) &\quad \text{with} \quad D_u^{d_u} D_v^{d_v} \hat{s}_{i,j}(u, v). \end{aligned}$$

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,

$$\mathbf{C}_{j,d_u,d_v}(u, v), \quad (10.9)$$

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

$$\mathbf{C}_{j,2,2}(u, v) = \begin{bmatrix} c_j(u, v) & D_v c_j(u, v) & D_v^2 c_j(u, v) \\ D_u c_j(u, v) & D_u D_v c_j(u, v) & D_u D_v^2 c_j(u, v) \\ D_u^2 c_j(u, v) & D_u^2 D_v c_j(u, v) & D_u^2 D_v^2 c_j(u, v) \end{bmatrix}. \quad (10.10)$$

The notation of the equivalent matrix from the local patches is

$$\tilde{\mathbf{S}}_{i,j,d_u,d_v}(u, v), \quad (10.11)$$

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 (10.10). An example of this matrix, where $d_u = 2$ and $d_v = 2$, is

$$\tilde{\mathbf{S}}_{i,j,d_u,d_v}(u, v) = \begin{bmatrix} s_{i,j}(u, v) & D_v s_{i,j}(u, v) & D_v^2 s_{i,j}(u, v) \\ D_u s_{i,j}(u, v) & D_u D_v s_{i,j}(u, v) & D_u D_v^2 s_{i,j}(u, v) \\ D_u^2 s_{i,j}(u, v) & D_u^2 D_v s_{i,j}(u, v) & D_u^2 D_v^2 s_{i,j}(u, v) \end{bmatrix}. \quad (10.12)$$

To sum up before constructing an algorithm: The first column in (10.9–10.10) is equal to the left hand side of (10.8), and only elements from the first column in (10.11–10.12) are used on the right hand side of (10.8). To compute the other columns we have to replace elements from the first column in (10.11) with respective elements from the other columns. The conclusion is, therefore, that we only have to replace individual elements from the first column of (10.11), that is, on right hand side of (10.8), with the respective rows of (10.11), to expand (10.8) to return (10.9).

Remark 14. It is, of course, possible and usual, to construct an evaluator which only returns the upper left half of the matrix (10.9). 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 4 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++). Note that the global/local affine mapping (see definition 7.7) is used in the following algorithm, it is used in line 1 and 3, i.e. in the call to the local patch evaluators (see algorithm 16). However, 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 3 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 6. (For notation, see section “Algorithmic Language”, page 18.)

The algorithm computes the matrix $\mathbf{C}_{j,d_u,d_v}(u,v)$ defined in (10.10). It is assumed that evaluators for the ERBS basis function, and the local patches are present (for $\omega_k(t)$, see definition 7.7), and that these evaluators return a matrix $\tilde{\mathbf{S}}_{i,j,d_u,d_v}(u,v)$ defined in (10.11) and analogous to (10.10). 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 $k_u : \backslash; u_{k_u} \leq u < u_{k_u+1}$, $k_v : \backslash; v_{k_v} \leq v < v_{k_v+1}$, and $d_u \in \{0, 1, 2, \dots, p\}$ (the number of derivatives in u direction) and $d_v \in \{0, 1, 2, \dots, p\}$ (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 10.8).

```

Matrix⟨T⟩ C ( double u, double v, int ku, int kv, int du, int dv )
  Matrix⟨T⟩ C0 =  $\tilde{\mathbf{S}}_{k_u,k_v,d_u,d_v}(\omega_{k_u}(u), \omega_{k_v}(v))$ ; // Result evaluating local patch
  if (u == uk_u)   return C0; // Return only local patch, see (10.5)
  Matrix⟨T⟩ C1 =  $\tilde{\mathbf{S}}_{k_u+1,k_v,d_u,d_v}(\omega_{k_u+1}(u), \omega_{k_v}(v))$ ; // Result evaluating local patch
  vector⟨double⟩ a(du+1); // For numbers - “Pascals triangle”
  vector⟨double⟩ B =  $\bar{B}(u, k_u, d_u)$ ; // Result evaluating ERBS-basis, (Alg. 3)
  C0 -= C1; // The matrix c0 is now  $\hat{c}_0$ , expanded (10.7)
  for ( int i=0; i ≤ du; i++ )
    ai = 1;
    for ( int j=i-1; j > 0; j-- )
      aj += aj-1; // Computing “Pascals triangle”-numbers
    for ( int j=0; j ≤ i; j++ )
      C1,i += (aj Bj) C0,i-j;
  return C1;

```

Note that the algorithm is updating whole rows of the matrix C_1 by summing up a row with scaled rows of matrix \hat{C}_0 . This is directly followed by the fact that equation (10.8) is expanded to compute not only the first column of (10.10), 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 (10.4). This equation is equivalent to equation (10.3), therefore, analogical to formula 7.52. We can simplify the first part of formula (10.4). It follows that for all knot intervals $j = 1, 2, \dots, n_v - 1$,

$$S(u, v) = \begin{cases} c_j(u, v), & \text{if } v = v_j, \\ c_{j+1}(u, v) + (c_j(u, v) - c_{j+1}(u, v)) B_j(v), & \text{if } v_j < v < v_{j+1}. \end{cases} \quad (10.13)$$

Computing the partial derivatives of (10.13) only according to v , we can see that for $v = v_j$, $j = 1, \dots, n_v$, all partial derivatives of the ERBS-surface with respect to v are equal to the respective derivatives from the inner loop, i.e.

$$D_v^{d_v} S(u, v) = D_v^{d_v} c_j(u, v), \quad \text{for } j = 1, \dots, n_u, \text{ and } d_v = 0, 1, 2, \dots$$

To simplify for all other v values in $[v_1, v_{n_v}]$, we first define

$$\hat{c}_j(u, v) = c_j(u, v) - c_{j+1}(u, v), \quad \text{if } v_j < v < v_{j+1},$$

then for $v_j < v < v_{j+1}$ we get the following equation for the function value and the derivatives in the v direction,

$$\begin{aligned} S(u, v) &= c_{j+1}(u, v) + \hat{c}_j(u, v) B_j(v) \\ D_v S(u, v) &= D_v c_{j+1}(u, v) + \hat{c}_j(u, v) D_v B_j(v) + D_v \hat{c}_j(u, v) B_j(v) \\ D_v^2 S(u, v) &= D_v^2 c_{j+1}(u, v) + \hat{c}_j(u, v) D_v^2 B_j(v) + 2D_v \hat{c}_j(u, v) D_v B_j(v) + D_v^2 \hat{c}_j(u, v) B_j(v). \end{aligned} \quad (10.14)$$

Studying (10.14) we can see that it has the same structure as (10.8). 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,

$$\mathbf{S}_{d_u, d_v}(u, v), \quad (10.15)$$

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:

$$\mathbf{S}_{2,2}(u, v) = \begin{bmatrix} S(u, v) & D_v S(u, v) & D_v^2 S(u, v) \\ D_u S(u, v) & D_u D_v S(u, v) & D_u D_v^2 S(u, v) \\ D_u^2 S(u, v) & D_u^2 D_v S(u, v) & D_u^2 D_v^2 S(u, v) \end{bmatrix}. \quad (10.16)$$

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 (10.14) we do not have the first column in the resulting matrix (10.15–10.16), but we have the first *row*. In addition, on the right hand side of (10.14) we only find elements from the first *row* of the matrix $C_{j,d_u,d_v}(u,v)$ (10.9–10.10). The algorithm becomes quite similar to algorithm 4 used for curves, and to algorithm 6 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 \mathbf{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 7. (*For notation, see section “Algorithmic Language”, page 18.*)

The algorithm computes the matrix $\mathbf{S}(u,v)$ equivalent to (10.10) and described in (10.2). 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 $\tilde{\mathbf{S}}_{i,j}(u,v)$ described in (10.11). 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 $d_u \in \{0, 1, 2, \dots, p\}$ (the number of derivatives in u direction), and $d_v \in \{0, 1, 2, \dots, p\}$ (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 $\mathbf{S}(u,v)$, and where the elements in the matrix are matching the elements in the matrix $C_{j,d_u,d_v}(u,v)$ described in (10.10).

```
Matrix⟨T⟩ eval ( double u, double v, int du, int dv )
    int ku = ku : \; uku ≤ u < uku+1;           // Index for the current knot-interval for u.
    int kv = kv : \; vkv ≤ v < vkv+1;           // Index for the current knot-interval for v.
    Matrix⟨T⟩ S0 = Cku,kv,du,dv}(u,v);          // Result from inner loop - kv.
    if (v == vkv)   return S0;                      // Return only inner loop, see (10.13).
    Matrix⟨T⟩ S1 = Cku,kv+1,du,dv}(u,v);        // Result from inner loop - kv + 1.
    vector⟨double⟩ a(d+1);                                // For numbers - “Pascals triangle”.
    vector⟨double⟩ B = B̄(v,kv,dv);                  // Result evaluating ERBS-basis, (Alg. 3).
    S0 -= S1;                                         // C0 is now Ĉ0, the whole matrix, see (10.6).
    for ( int i=0; i ≤ dv; i++ )
        ai = 1;
        for ( int j=i-1; j > 0; j-- )
            aj += aj-1;                               // Computing “Pascals triangle”-numbers.
            for ( int j=0; j ≤ i; j++ )
                (S1T)i += (aj Bj) (S0T)i-j;      // “column += scalar×column”, (10.14).
    return S1;
```

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.

10.3 Bézier patches as local patches

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

$$s(u, v) = \sum_{i=0}^{d_u} \sum_{j=0}^{d_v} c_{i,j} b_{d_u,i}(u) b_{d_v,j}(v) \quad \text{for } 0 \leq u \leq 1 \text{ and } 0 \leq v \leq 1, \quad (10.17)$$

where the basis functions are the Bernstein polynomials

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

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 (8.8) the generalized Bernstein/Hermite matrix was introduced, and in algorithm 5 an algorithm to make the same matrix is developed and described. In the generalized Bernstein/Hermite matrix, each row is scaled by δ^j , where j is the row number (starting with 0). The matrix looks like this:

$$\mathbf{B}_d(t, \delta) = \begin{pmatrix} \delta^0 D^0 b_{d,0}(t) & \dots & \delta^0 D^0 b_{d,d}(t) \\ \vdots & \ddots & \vdots \\ \delta^d D^d b_{d,0}(t) & \dots & \delta^d D^d b_{d,d}(t) \end{pmatrix}. \quad (10.18)$$

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/quattro/... 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, (10.18), we can first make an expanded matrix version of equation (10.17), not only returning the value, but also all the derivatives. We now get:

$$\tilde{\mathbf{S}}_{d_u, d_v}(u, v) = \mathbf{B}_{d_u}(u, \delta_u) C \mathbf{B}_{d_v}(v, \delta_v)^T \quad \text{for } 0 \leq u \leq 1 \text{ and } 0 \leq v \leq 1, \quad (10.19)$$

where C is the control polygon (matrix), and δ_u and δ_v are the scaling as a result of the affine global/local mapping (definition 7.7). If $d_u = d_v = 2$ the matrix $\tilde{\mathbf{S}}_{d_u, d_v}(u, v)$ will be

$$\tilde{\mathbf{S}}_{2,2}(u, v) = \begin{bmatrix} s(u, v) & D_v s(u, v) & D_v^2 s(u, v) \\ D_u s(u, v) & D_u D_v s(u, v) & D_u D_v^2 s(u, v) \\ D_u^2 s(u, v) & D_u^2 D_v s(u, v) & D_u^2 D_v^2 s(u, v) \end{bmatrix}. \quad (10.20)$$

As one can see, this matrix (10.20) 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 δ_u and δ_v , completely describes the patch.

10.3.1 Local Bézier patches and Hermite interpolation

We start by recalling the settings from section 7.9, and adapting them to tensor product ERBS surfaces.

- Given is a surface $g(u, v)$, $g : \Omega \subset \mathbb{R}^2 \rightarrow \mathbb{R}^n$, where $\Omega = [u_s, u_e] \times [v_s, v_e]$, and where we might have $n = 1, 2, 3, \dots$,
- 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 $i = 2, 3, \dots, m_u - 1$, generate u_i so that $u_{i-1} < u_i$ and where $u_{m_u-1} < u_{m_u}$, and for $j = 1, \dots, m_v$, 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 $i = 1, \dots, m_u$ and $j = 1, \dots, m_v$,

$$D_u^{s_u} D_v^{s_v} S(u_i, v_j) = D_u^{s_u} D_v^{s_v} g(u_i, v_j), \text{ for } s_u = 0, \dots, d_{u,i} \text{ and } s_v = 0, \dots, d_{v,j}.$$

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 7.4, but adjust it with the local domain for Bézier patches. We now get, for $i = 1, \dots, m_u$ and $j = 1, \dots, m_v$,

$$D_u^{s_u} D_v^{s_v} S(u_i, v_j) = \delta_{u,i}^{s_u} \delta_{v,j}^{s_v} D_u^{s_u} D_v^{s_v} s_{i,j} \circ (\omega_i(u_i), \omega_j(v_j)), \text{ for } s_u = 0, \dots, d_{u,i} \text{ and } s_v = 0, \dots, d_{v,j},$$

where $S(u, v)$ is the tensor product ERBS-surface, $s_{i,j}(u, v)$ are for all i, j 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 7.7, and

$$\delta_{u,i} = \frac{1}{u_{i+1} - u_{i-1}} \quad \text{for } i = 1, 2, \dots, m_u,$$

and

$$\delta_{v,j} = \frac{1}{v_{j+1} - v_{j-1}} \quad \text{for } j = 1, 2, \dots, m_v,$$

are the domain scaling factors defined in Theorem 7.4. 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 $i = 1, \dots, m_u$ and $j = 1, \dots, m_v$. We look at the equation for the Hermite interpolations for a local Bézier patch with the pair of indices (i, j) ,

$$\begin{aligned} D_u^{s_u} D_v^{s_v} g(u_i, v_j) &= D_u^{s_u} D_v^{s_v} S(u_i, v_j) \\ &= \delta_{u,i}^{s_u} \delta_{v,j}^{s_v} D_u^{s_u} D_v^{s_v} s_{i,j} \circ (\omega_i(u_i), \omega_j(v_i)) \\ &= \sum_{r=0}^{d_1} \sum_{s=0}^{d_2} c_{i,j,r,s} \delta_{u,i}^{s_u} D_u^{s_u} b_{d_u,r} \circ \omega_i(u_i) \delta_{v,j}^{s_v} D_v^{s_v} b_{d_v,s} \circ \omega_j(v_j) \\ &\quad \text{for } s_u = 0, \dots, d_u, \text{ and } s_v = 0, \dots, d_v. \end{aligned}$$

This is nearly the same as the formulation in (10.19), but we now have included the global/local mapping. The matrix form now is, for $i = 1, \dots, m_u$, and $j = 1, \dots, m_v$,

$$\mathbf{g}_{d_u, d_v}(u_i, v_j) = \mathbf{B}_{d_u}(\omega_i(u_i), \delta_{u,i}) C_{i,j} \mathbf{B}_{d_v}(\omega_j(v_j), \delta_{v,j})^T, \quad (10.21)$$

where

$$\mathbf{g}_{d_u, d_v}(u_i, v_j) = \begin{pmatrix} g(u_i, v_j) & \cdots & D_v^{d_v} g(u_i, v_j) \\ \vdots & \ddots & \vdots \\ D_u^{d_u} g(u_i, v_j) & \cdots & D_u^{d_u} D_v^{d_v} g(u_i, v_j) \end{pmatrix}.$$

Then the final step to generate the local Bézier patches is to solve equation 10.21 according to the Bézier coefficients (control polygon) $\mathbf{C}_{i,j}$,

$$\mathbf{C}_{i,j} = \mathbf{B}_{d_u}(\omega_i(u_i), \delta_{u,i})^{-1} \mathbf{g}_{d_u, d_v}(u_i, v_j) \mathbf{B}_{d_v}(\omega_j(v_j), \delta_{v,j})^{-T}.$$

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 5, 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 further here, but there are a lot of available programming libraries including optimized algorithms for matrix inversions, see, e.g., [151].

Remark 16. Note that the three matrices on the right hand side in 10.21 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.

10.3.2 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 [3]. The formula is

$$s(u, v) = \begin{pmatrix} 2 \frac{\sin(3u)}{2 + \cos v} \\ 2 \frac{\sin u + 2 \sin(2u)}{2 + \cos(v + \frac{2}{3}\pi)} \\ \frac{(\cos u - 2 \cos(2u))(2 + \cos v)(2 + \cos(v + \frac{2}{3}\pi))}{4} \end{pmatrix} \quad \begin{array}{l} \text{for } u \in (-\pi, \pi], \\ \text{and } v \in (-\pi, \pi]. \end{array} \quad (10.22)$$

In Figure 10.3 there is a plot of a tensor product ERBS surface interpolating a “Trianguloid Trefoil” surface (10.22) at 5×5 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 (10.18) has the dimension 3×3 . In the Figure 10.3 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 fairly 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 [4]. The formula for this surface is

$$s(u, v) = \begin{pmatrix} (2 + \cos u) (\frac{v}{3} - \sin v) \\ (2 + \cos(u - \frac{2}{3}\pi)) (\cos v - 1) \\ (2 + \cos(u + \frac{2}{3}\pi)) (\cos v - 1) \end{pmatrix} \quad \begin{array}{l} \text{for } u \in (-\pi, \pi], \\ \text{and } v \in (-2\pi, 2\pi]. \end{array} \quad (10.23)$$

In Figure 10.4 there is a plot of a tensor product ERBS surface interpolating a ‘Bent Horns’ surface (10.23) at 5×5 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 (10.18) has the dimension 3×3 . In the Figure 10.4 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

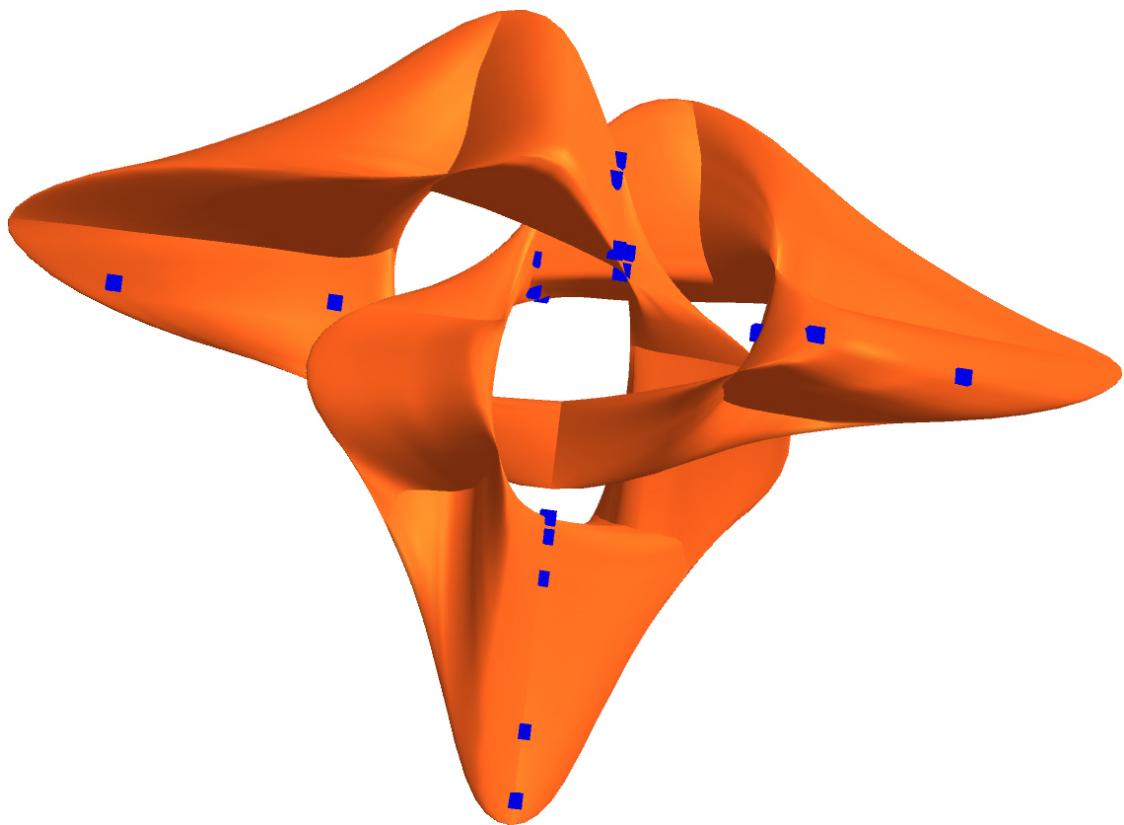


Figure 10.3: This Expo-Rational B-spline tensor product surface is made by Hermite interpolation of a “Trianguloid Trefoil” surface [3] at 5×5 points. The positions of the interpolating points are seen as cubes.

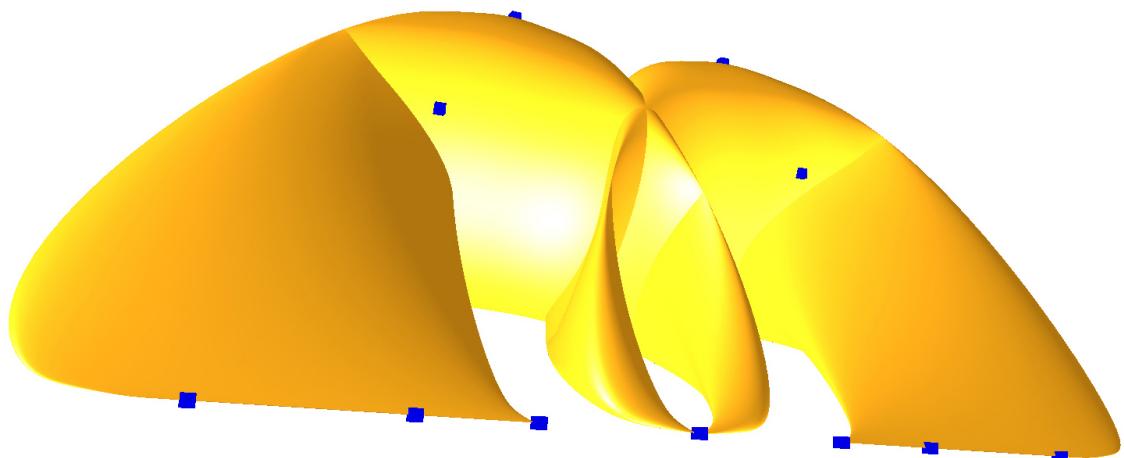


Figure 10.4: This Expo-Rational B-spline tensor product surface is made by interpolating (position, first and second derivative) a “Bent Horns” surface [4] at 5×5 points. The positions of the interpolating points are seen as cubes.

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,

$$s(u, v) = \begin{pmatrix} r \cos u \cos v \\ r \sin u \cos v \\ r \sin v \end{pmatrix} \quad \begin{array}{l} \text{for } u \in (0, 2\pi], \\ \text{and } v \in (-\frac{\pi}{2}, \frac{\pi}{2}]. \end{array} \quad (10.24)$$

In the equation r is the radius of the sphere. Figure 10.5 is a plot of a tensor product ERBS surface interpolating a sphere (10.24) at 4×4 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 (10.18) has the dimension 3×3 . In Figure 10.5 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 10.6 there is a plot of the same tensor product ERBS surface as in Figure 10.5, 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 10.7 there is another plot of the same tensor product ERBS surface as in the figures 10.5 and 10.6, 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

$$s(u, v) = \begin{pmatrix} \cos u(R + r \cos v) \\ \sin u(R + r \cos v) \\ r \sin v \end{pmatrix} \quad \begin{array}{l} \text{for } u \in (0, 2\pi], \\ \text{and } v \in (0, 2\pi]. \end{array} \quad (10.25)$$

Here r is the small radius, the radius in the tube, and R is the big radius, the radius of the tube. In Figure 10.8 there is a plot of a tensor product ERBS surface interpolating a torus (10.25) at 5×5 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 (10.18) has the dimension 3×3 . In Figure 10.8 the interpolation points are marked as blue cubes, and some of them can clearly be seen. In Figure 10.9, 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 10.9 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

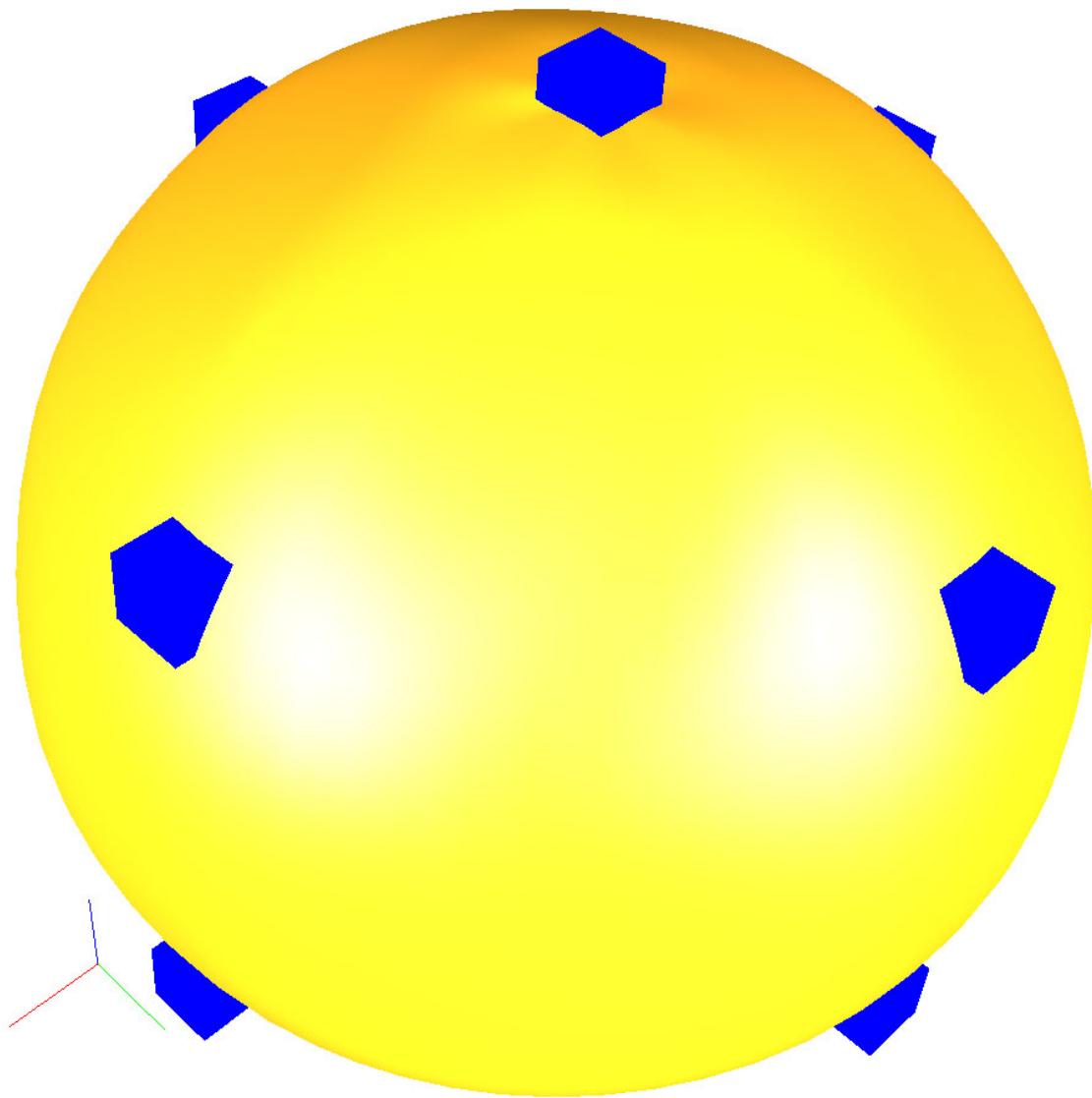


Figure 10.5: This Expo-Rational B-spline tensor product surface is made by interpolating a sphere at 4×4 points (position, first and second derivative). The positions of the interpolating points are seen as blue cubes.

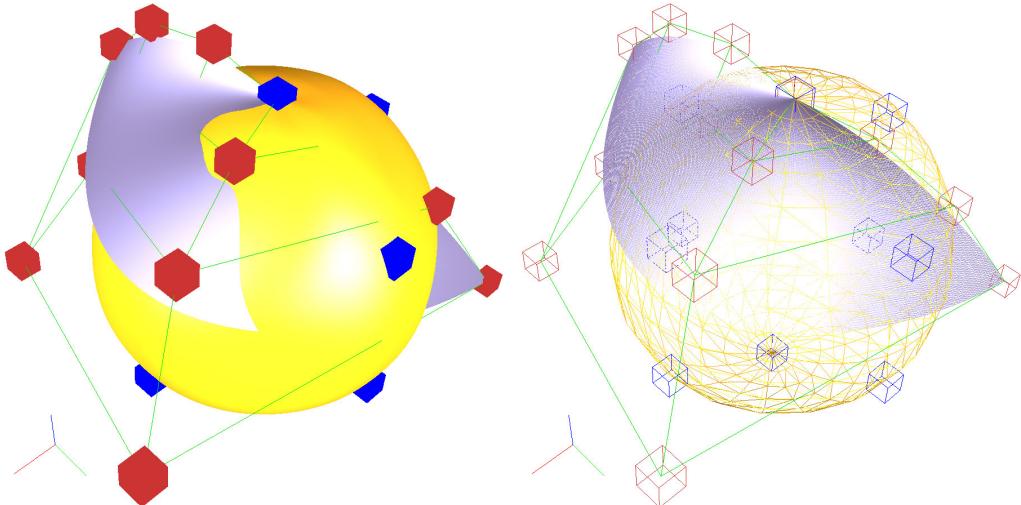


Figure 10.6: A plot of the approximated sphere from Figure 10.5, with one of the local Bézier patches located at the “north pole”. The control polygon of the Bézier patch is marked in green, and the nodes in the polygon are marked as red cubes. On the right hand side there is a wireframe version of the figures that we can see on the left hand side.

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 [14]. The formula for this surface is

$$s(u, v) = \begin{pmatrix} \cos v + \frac{v}{10} (\cos u \cos v + a \cos u \sin v) \\ \sin v + \frac{v}{10} (\cos u \sin v - a \cos u \cos v) \\ (b \sin u + \frac{6}{10}) v \end{pmatrix} \quad \begin{array}{l} \text{for } u \in (0, 2\pi], \\ \text{and } v \in (\frac{\pi}{4}, 5\pi]. \end{array} \quad (10.26)$$

In Figure 10.11 there is a plot of an tensor product ERBS surface interpolating the “Sea Shell” surface (10.26) at 4×8 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 (10.18) has the dimension 3×3 . In Figure 10.11 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 10.12, there are two plots of the tensor product ERBS surface from Figure 10.11, 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 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

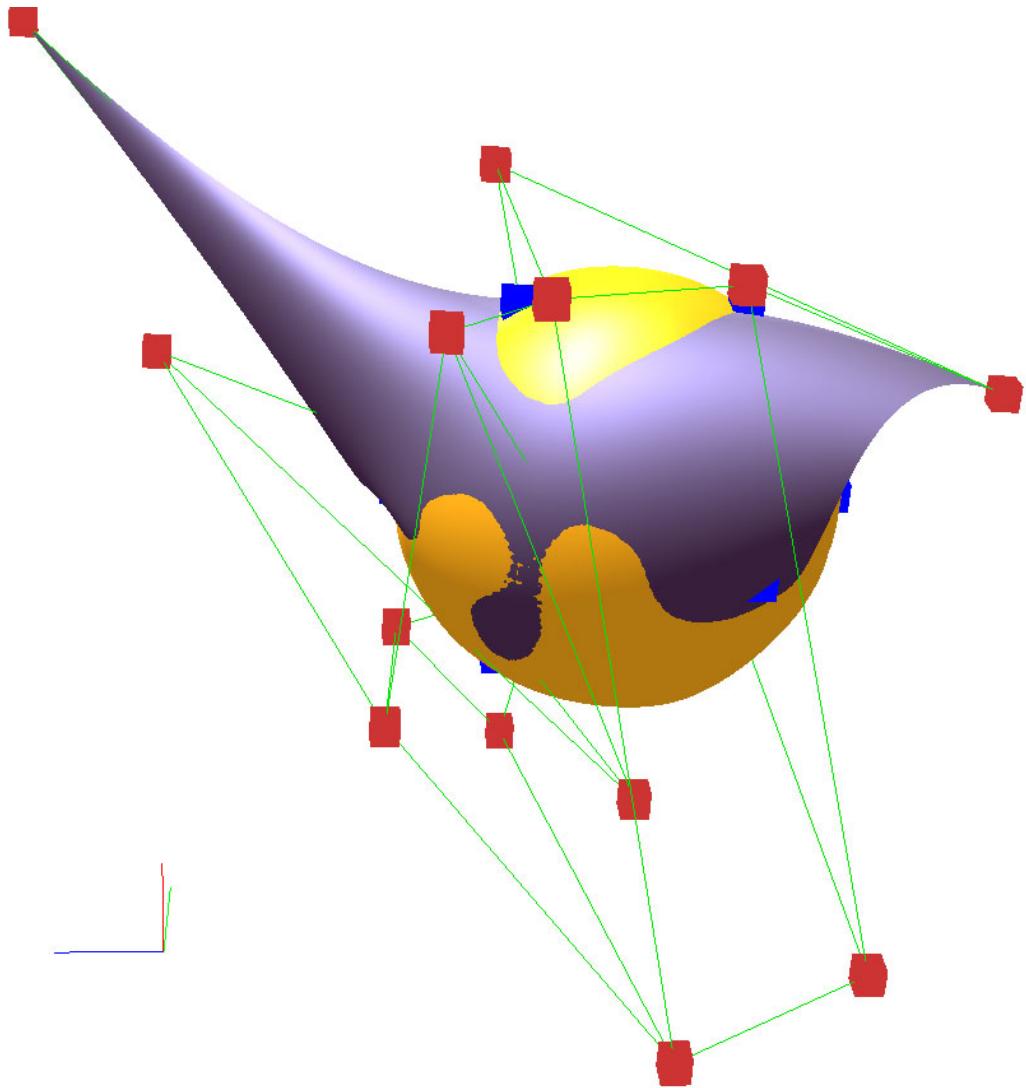


Figure 10.7: A plot of the approximated sphere from Figure 10.5, where one of the local Bezier patches is shown. The control polygon of the Bezier patch is marked in green, and the nodes in the polygon are marked as red cubes. Note that the figures are rotated, as can be seen on the RGB-frame in the bottom left hand corner. The “north pole”, in both this figure and the previous figures, is in the direction of the color blue (B).

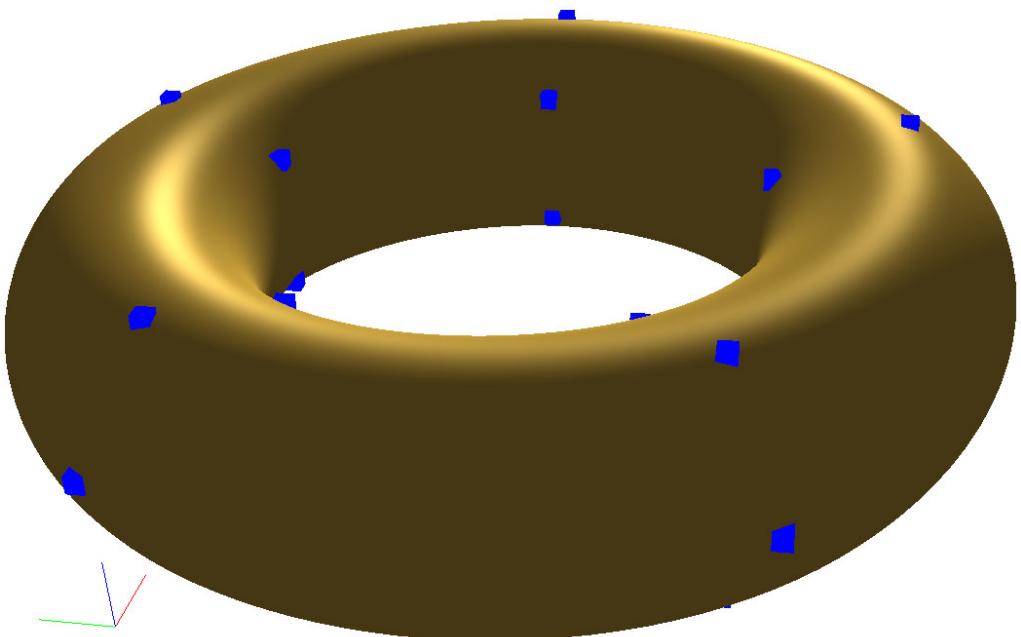


Figure 10.8: This Expo-Rational B-spline tensor product surface is made by interpolating a torus at 5×5 points (position, first and second derivative). The positions of the interpolating points are seen as blue cubes.

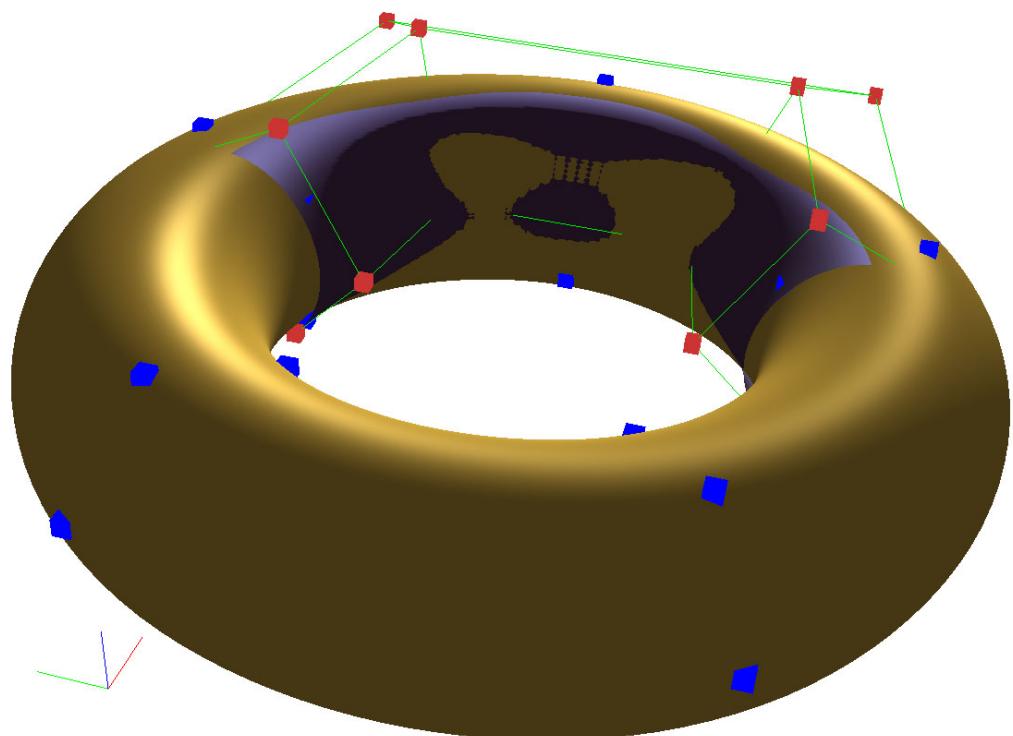


Figure 10.9: A plot of the approximated torus from Figure 10.8, where one of the local Bézier patches is also plotted. Also the control polygon of the local Bézier patch is plotted in green, while the nodes are plotted as red cubes.

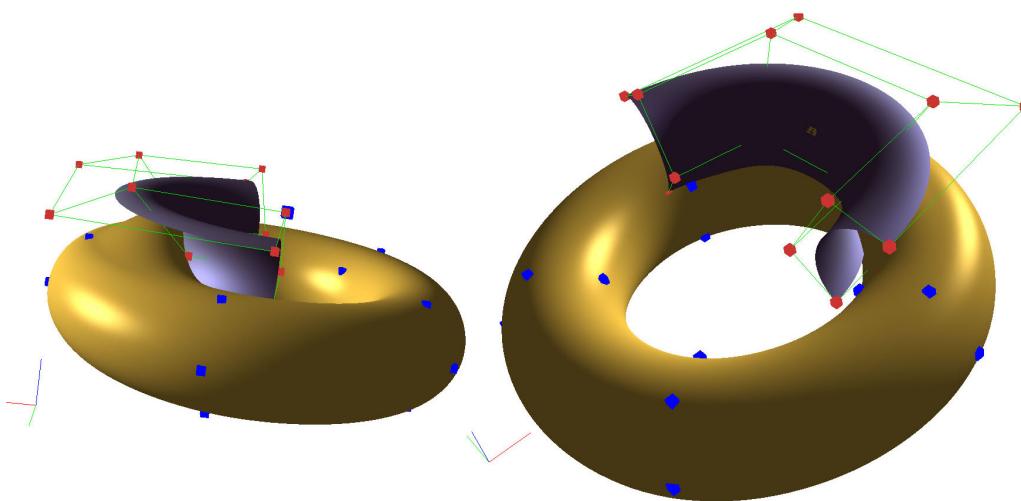


Figure 10.10: Two plots of the approximated torus from Figure 10.8, where one of the local Bézier patches is also plotted. In these plots the local patch is moved slightly upwards. In the plot on the left hand side one can clearly see that the ERBS surface has followed the movement. One can clearly see what the local Bézier patch looks like.

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, 10.13, 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, 10.14, 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. Gaussian and mean curvature can be studied in more detail in [137] and [46].

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.

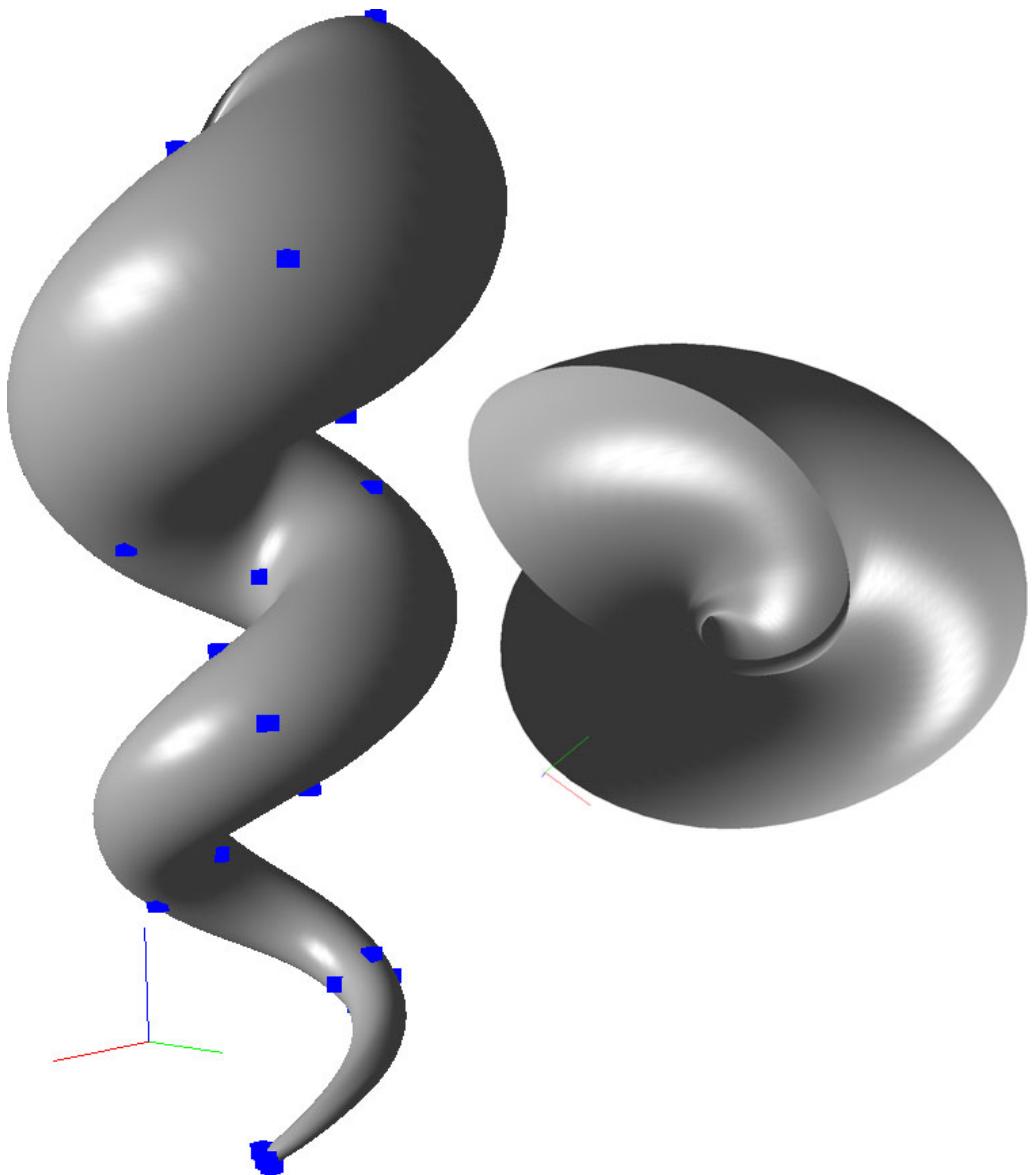


Figure 10.11: This Expo-Rational B-spline tensor product surface is made by interpolating (position, first and second derivative) a “Sea Shell” surface [14] at 4×8 points. The positions of the interpolating points are seen as cubes. The surface is plotted from two different angles. On the right hand side the interpolation points are hidden.

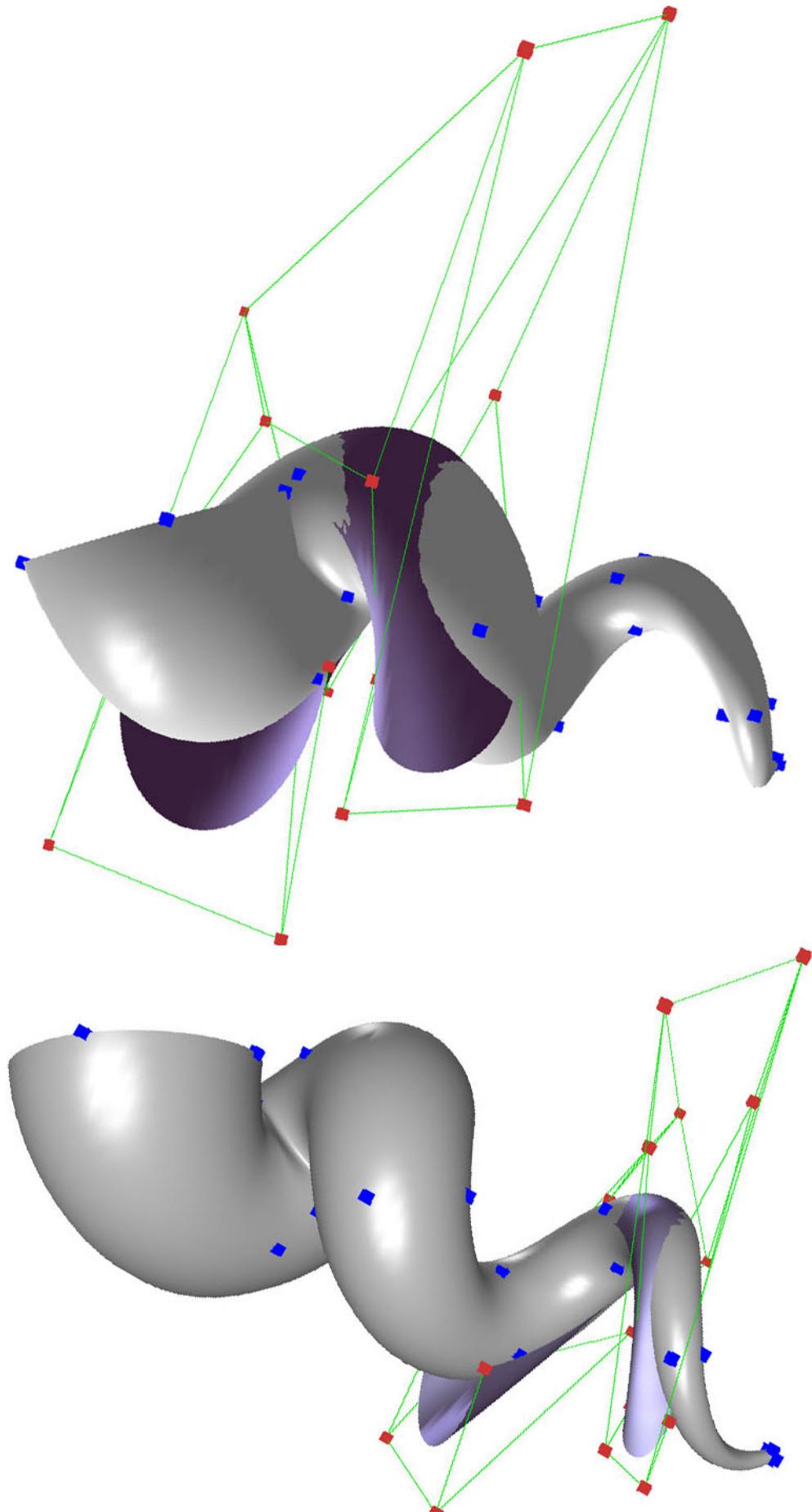


Figure 10.12: Two plots of the Expo-Rational B-Spline tensor product surface made by interpolating a “Sea Shell” surface plotted in Figure 10.11. The two plots are seen from different angles. In each of the two plots one of the local Bézier patches is also plotted. In the upper plot one of the bigger local patches is plotted. In the lower one a smaller patch

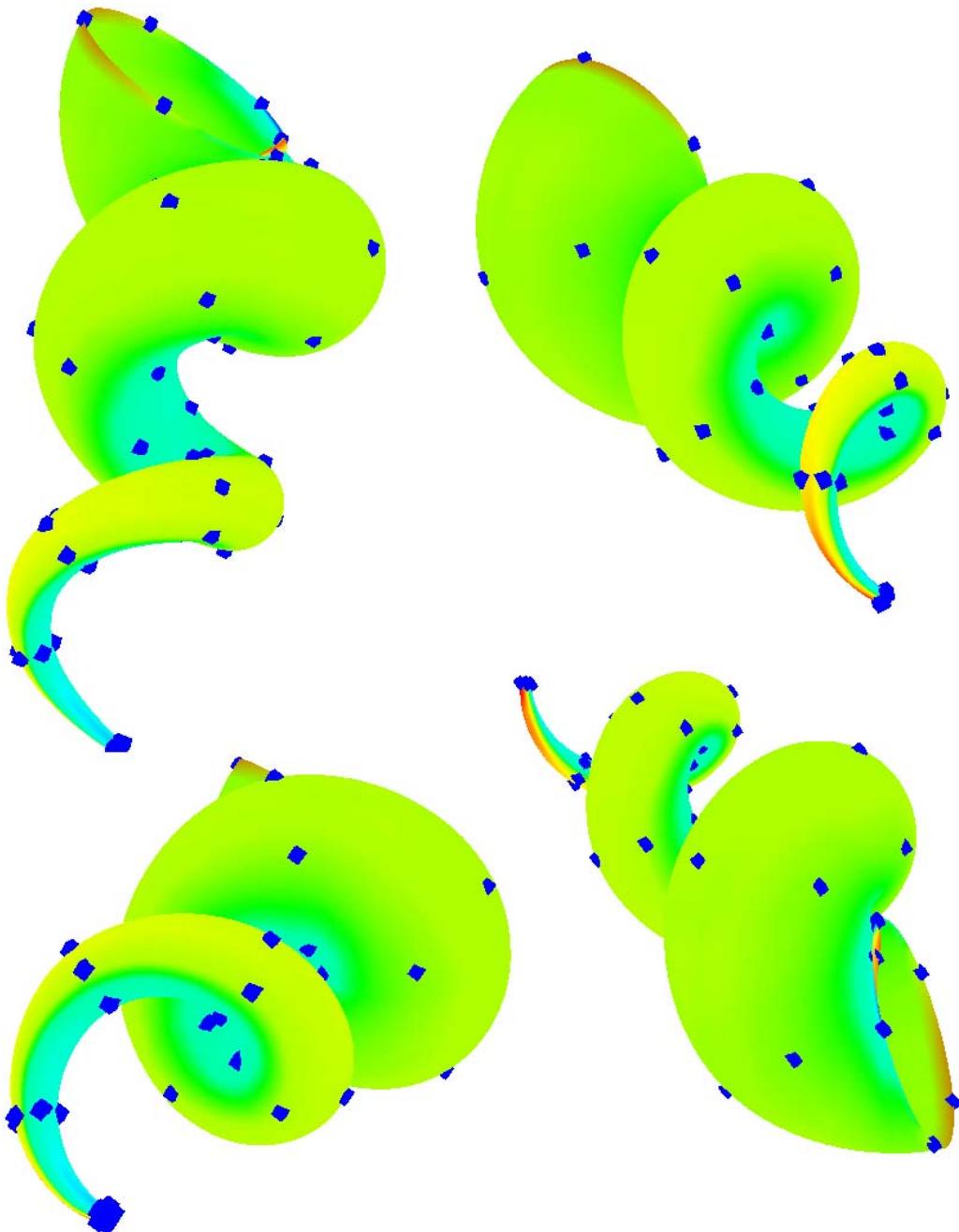


Figure 10.13: This Expo-Rational B-spline tensor product surface is made by interpolating (position, first and second derivative) a “Sea Shell” surface [14] at 5×12 points. The positions of the interpolating points are seen as blue cubes. The surface is plotted from four different angles. The color is based on the Gaussian curvature of the plotted ERBS tensor product surface.

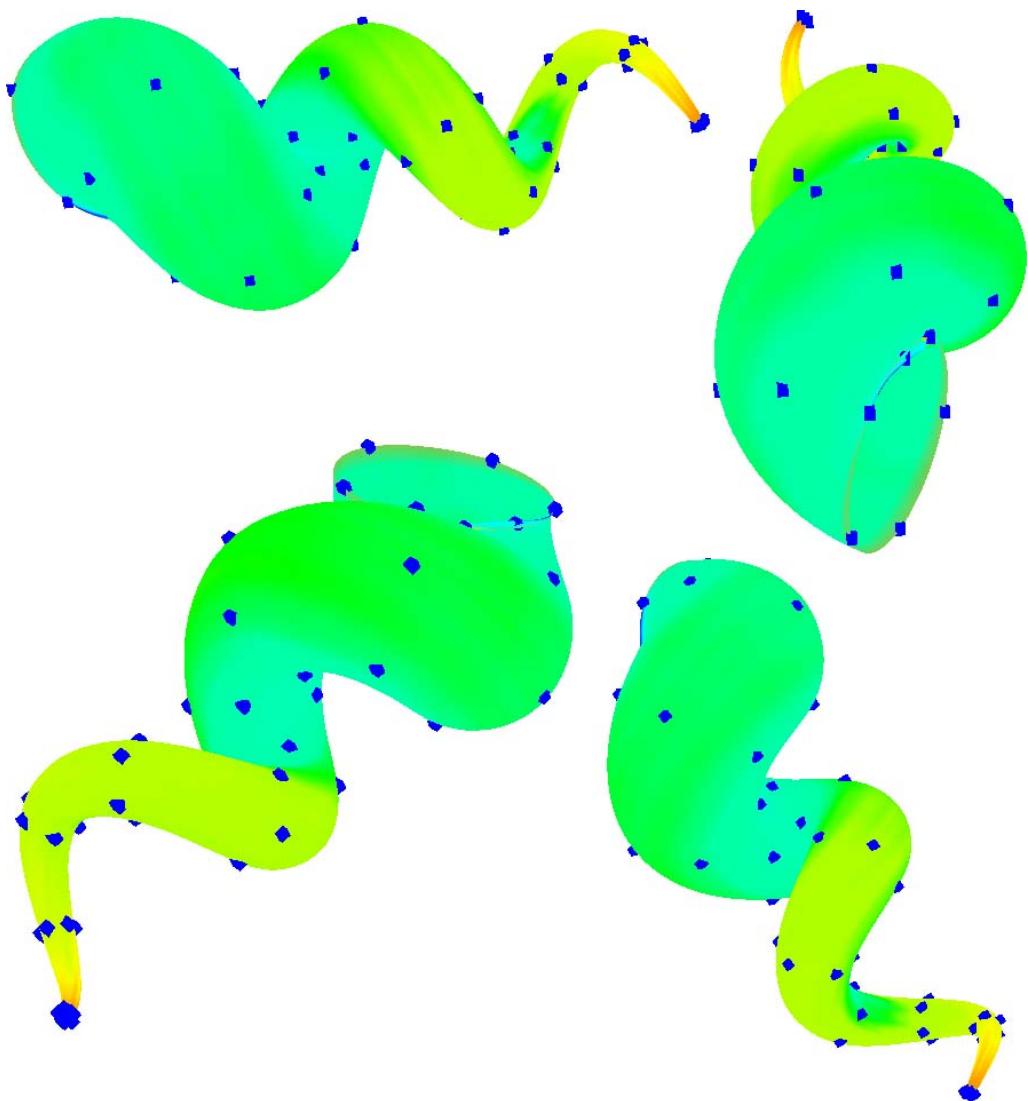


Figure 10.14: This Expo-Rational B-spline tensor product surface is made by interpolating (position, first and second derivative) a “Sea Shell” surface [14] at 5×12 points. The positions of the interpolating points are seen as blue cubes. The surface is plotted from four different angles. The color is based on the Mean curvature of the plotted ERBS tensor product surface.

10.4 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 [6] 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 [127] 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 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 [26] [7] [74] [153]). 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 5×5 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 (10.18) has the dimension 3×3 . After the interpolation, two of the points marked as yellow cubes in Figure 10.15 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 10.16 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 8×4 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 (10.18) has the dimension 3×3 . 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 10.17 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 4×4 points. The number of partial derivatives used results in the matrix $\mathbf{g}_{d_u,d_v}(u_i, v_j)$ defined in (10.18) which has the dimension 3×3 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 10.18.

The second sphere example can be seen in Figure 10.19. This example is based on constructing an ERBS surface by Hermite interpolation of a sphere at 6×6 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 (10.18) which has the dimension 3×3 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^∞ (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 10.20 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 3×3 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 (10.18) has the dimension 2×2 . 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

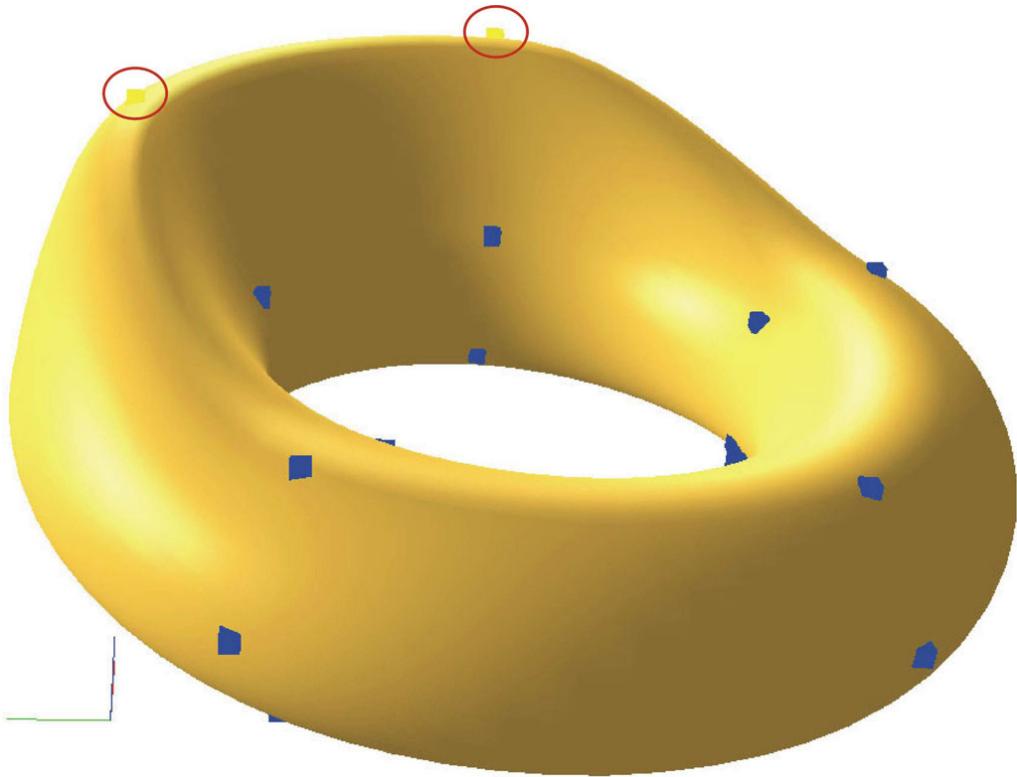


Figure 10.15: The approximated ERBS torus from Figure 10.8 is edited by moving two of the local Bézier patches upwards. The two Bézier patches are marked by yellow cubes (in red circles) at the interpolation points, while the other interpolation points are marked by blue cubes.

central interpolation point is first moved upwards and is then rotated approximately 140° around the blue axis of the RGB frame. The result can be seen in Figure 10.20, a kind of a twisted surface.

The last example is the “airplane”. This example can be seen in Figure 10.21. The “airplane” construction also starts as an Hermite interpolation of a planar rectangle with 3×3 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 (10.18) has the dimension 3×3 . 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 [98]. One of the motives for the knot removal was to reduce the amount of data. 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 includes one of the parameter values that defined the current area. Later, some attentions were made to solve

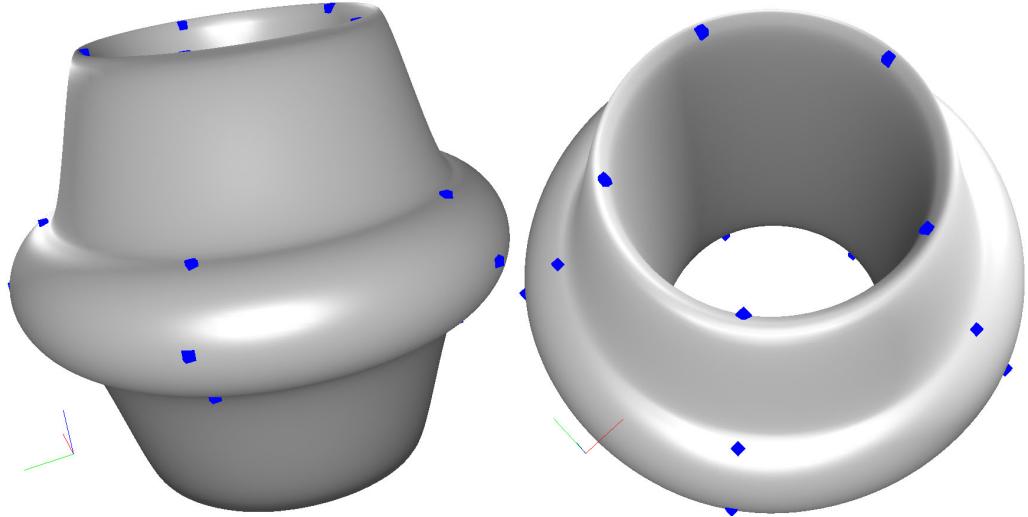


Figure 10.16: The approximated ERBS torus from Figure 10.8 is edited by moving five Bézier patches (interpolation points) on either side of the ERBS torus in opposite directions. In the figure there are two plots of the same object shown from different angles.

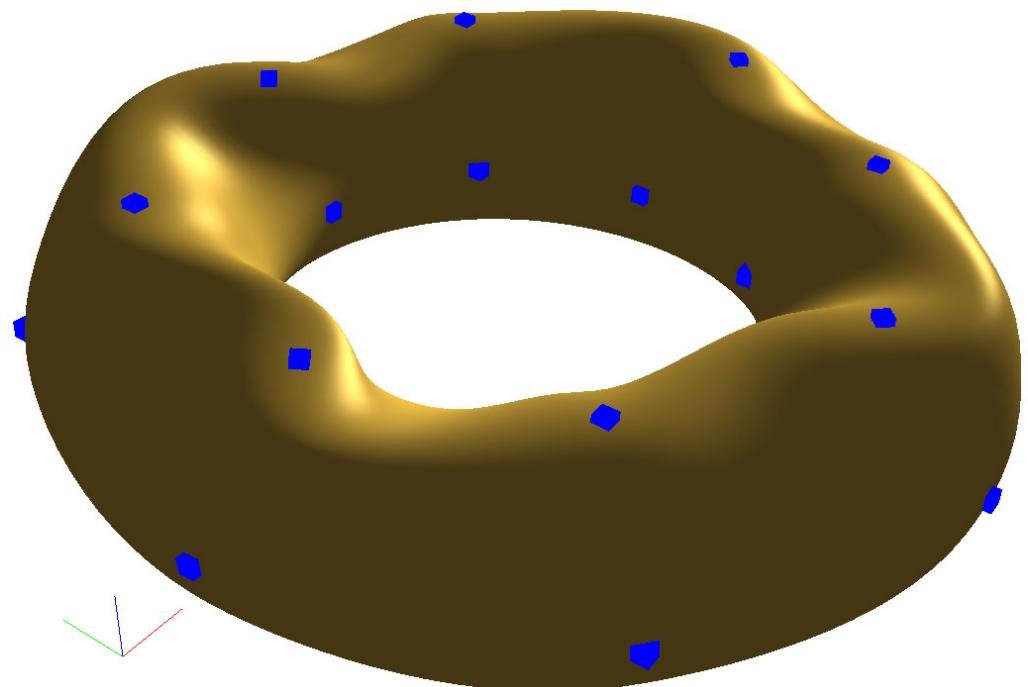


Figure 10.17: The approximated ERBS torus from Figure 10.8 is edited by rotating half of the interpolation points at the top around the z-axis. Note that none of the interpolation points are actually translated (only rotated).

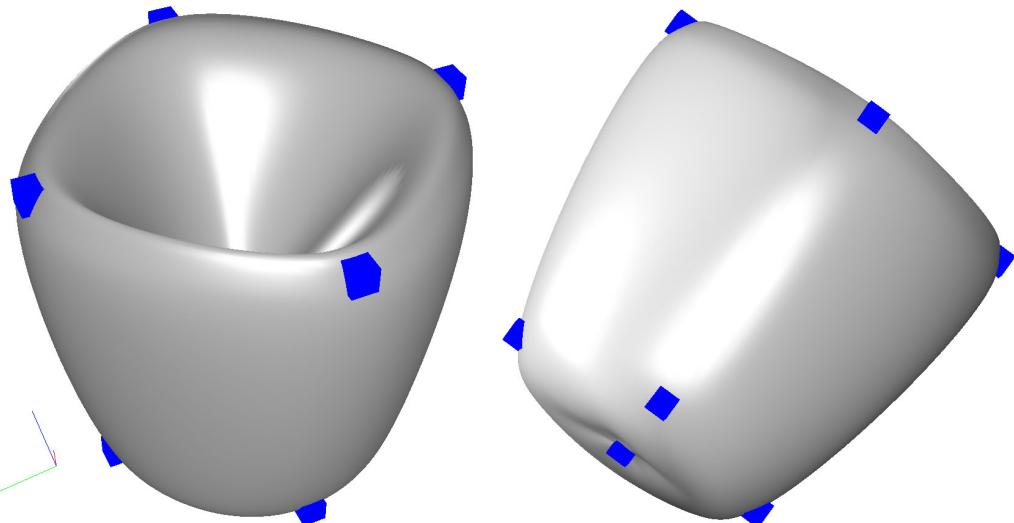


Figure 10.18: The approximated ERBS sphere from Figure 10.5 is edited by moving four interpolation points upwards and apart, and four points downwards. The four top points that coincide are also moved downwards. The result is a “mortar”, seen from two angles.

this problem by using (partial) hierarchical B-splines [96]. Even later, T-splines were introduced by Thomas Sederberg et all. [126], and a conversion from NURBS to T-spline was presented in [125]. Recently, a conversion between hierarchical B-splines and T-splines has also been introduced [144]. 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 (multilevel 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.

10.5 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-

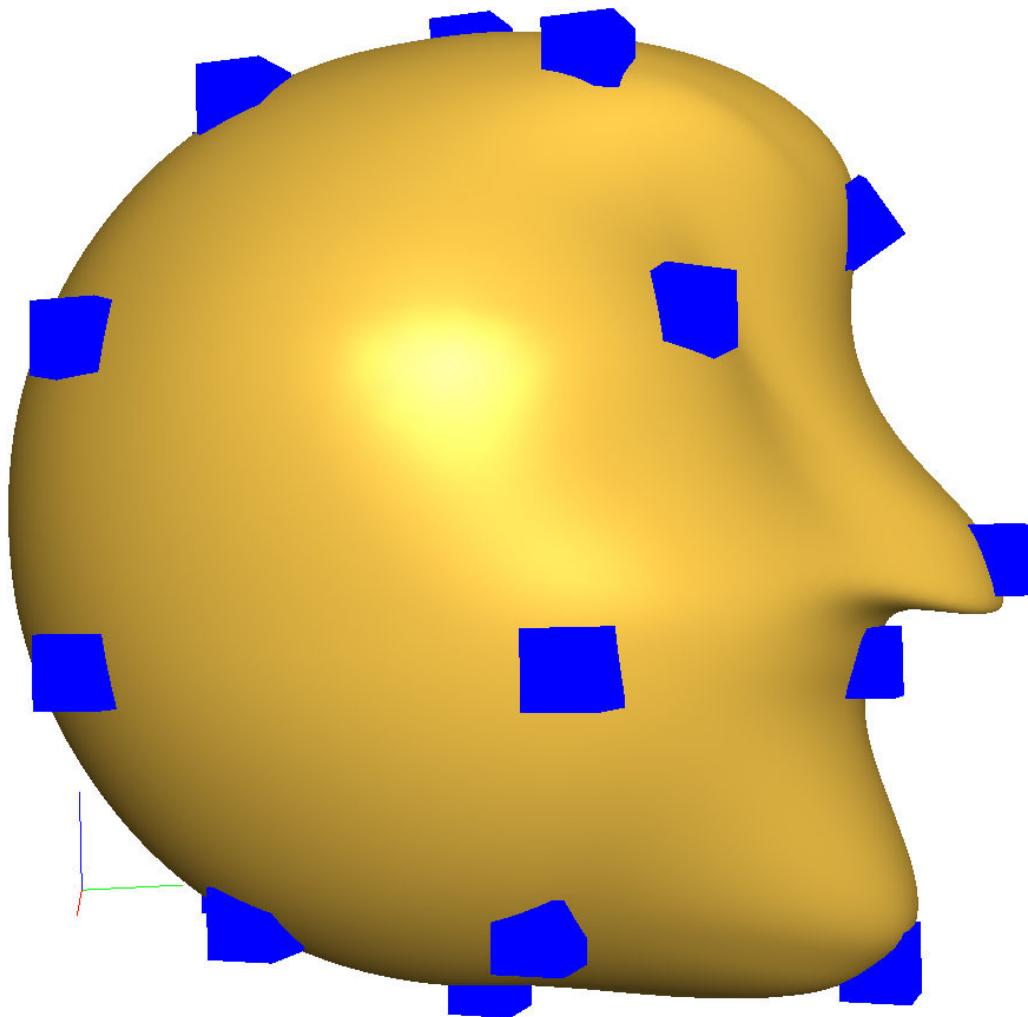


Figure 10.19: The figure, “head”, shows one Expo-Rational B-spline tensor product surface made by first interpolating (position, first and second derivative) a sphere at 6×6 points. Then some of the interpolation points (five of the blue cubes) are moved, and some of them, three points at the eyes and nose, are also rotated.

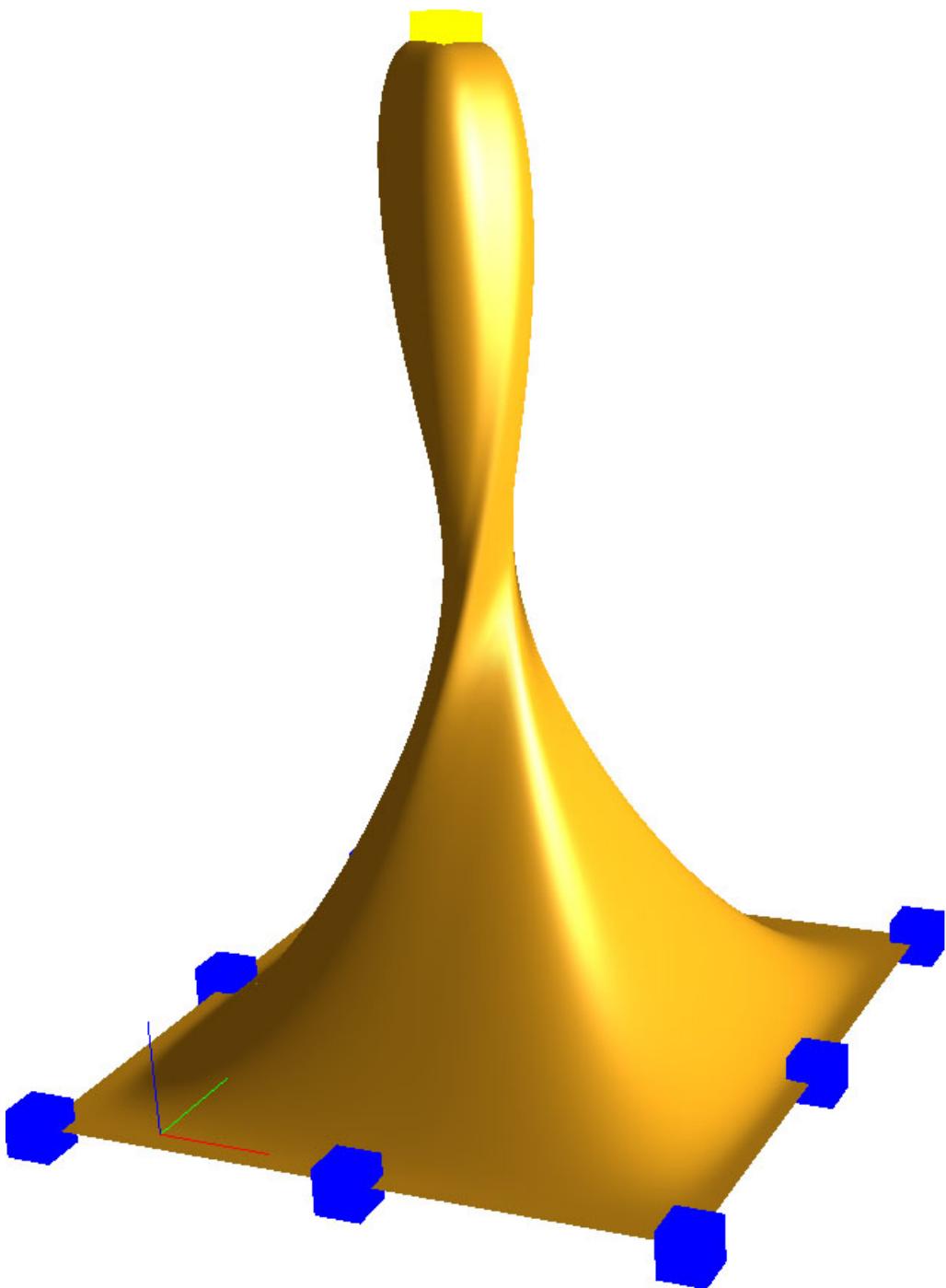


Figure 10.20: This Expo-Rational B-spline tensor product surface is first made by interpolating (position, first and second derivative) a plane at 3×3 points. The positions of the interpolating points are seen as cubes. The interpolation point in the center is moved upwards. Finally the center point is twisted, i.e. rotated approximately 140° around the z -axis.

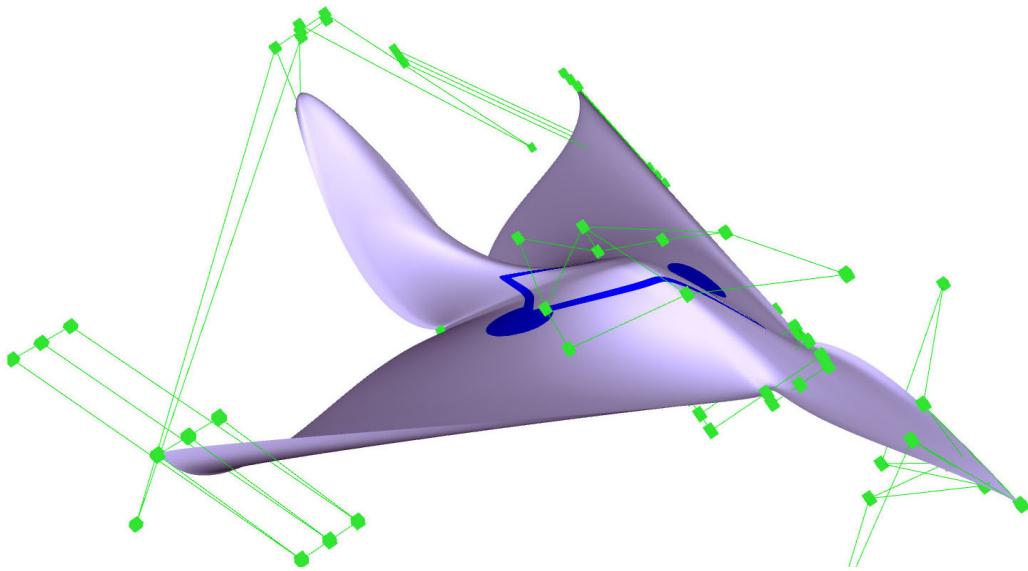


Figure 10.21: This Expo-Rational B-spline tensor product surface is made by first interpolating (position, first and second derivative) a plane at 3×3 points. Then the interpolation points are moved, and finally the control polygon on all of the local Bézier patches is edited. In the figure, some of the control polygons for the local Bézier patches are shown. (The surface is designed by Børre Bang.)

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 compute only the sample points which are to 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 (10.19), 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 (10.18), 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 11

Triangular Surfaces

Triangular surfaces (an example can be seen in Figure 11.1) 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 holes/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

$$\chi(S) = F - E + V,$$

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 . $\chi(S)$ 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 related to the genus, g , of the surface in the following way,

$$\chi(S) = 2(1 - g).$$

A closer study of this can be found in connection with a study of the Gauss-Bonnet Theorem in [46].

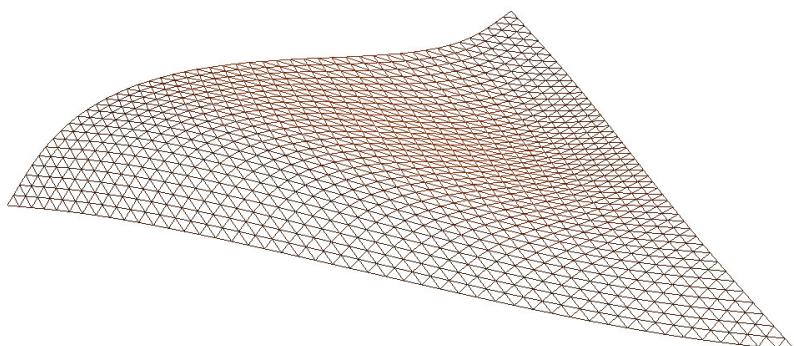


Figure 11.1: A smooth and non-planar triangular surface in \mathbb{R}^3 .

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 [53]) first introduced early in the 1960s by de Casteljau in two internal Citroën technical reports [37] and [38]. The basic constructions of the Bézier triangle will shortly be repeated in section 11.2.

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.

11.1 Homogenous barycentric coordinates for simplices

Barycentric coordinates were introduced by Möbius as early as in 1827, see [55]. 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 Warren, in [146] and in later publications, and Michael Floater and his colleagues investigating mean value/barycentric coordinates in [58] and [57].

In this section we will only repeat some main facts about homogeneous barycentric coordinates. Notice that contrary to barycentric coordinates¹, 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 11.2 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 11.2 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 11.2, the restriction on the parameters is,

$$u, v, w \geq 0.$$

¹There is a connection between barycentric coordinates and the Projective space. i.e. the set of all straight line going through the origin.

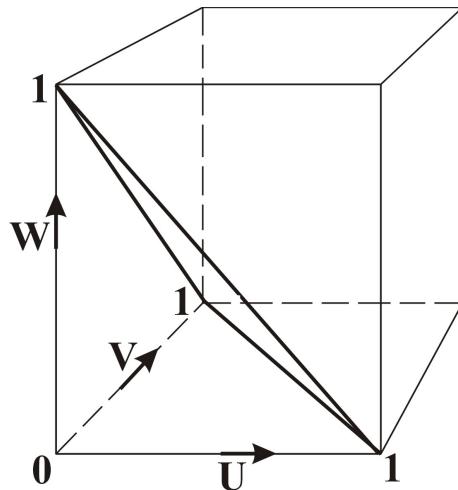


Figure 11.2: A parametric unit cube described by the coordinates u , v and w . The “main diagonal” is marked with thicker lines, and can be seen as a triangle in the figure.

The name “barycenter” is the Latin word for the center of gravity. In fact it is referring 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.

Now follows definition of homogeneous barycentric coordinates of a point.

Definition 11.1. We first denote the convex set of points on the “main diagonal” of a unit $(n+1)$ -dimensional hypercube as Ξ_n . The definition of points in homogeneous barycentric coordinates $\mathbf{p} \in \Xi_n$ is

$$\mathbf{p} = \{u_i\}_{i=1}^{n+1}, \quad \text{where} \quad \sum_{i=1}^{n+1} u_i = 1,$$

fulfilling a convexity property when

$$u_i \geq 0 \quad \text{for } i = 1, 2, \dots, n+1.$$

A point $\mathbf{p} \in \Xi_2$ (a point on the triangle in Figure 11.2) 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 11.2, 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 11.2. We first denote the n -dimensional vector space parallel to the “main diagonal” of an unit $(n+1)$ -dimensional hypercube as Υ_n . The definition of a vector in homogeneous barycentric coordinates $\mathbf{d} \in \Upsilon_n$ is

$$\mathbf{d} = \mathbf{p}_2 - \mathbf{p}_1 = \{r_i\}_{i=1}^{n+1}, \quad \text{where} \quad \sum_{i=1}^{n+1} r_i = 0 \quad \text{and} \quad \mathbf{p}_1, \mathbf{p}_2 \in \Xi_n.$$

For a triangle (2-dimensional simplex), we then see that a vector $d \in \Upsilon_2$ is defined in the following way,

$$\mathbf{d} = (r, s, t), \quad \text{where} \quad r + s + t = 0.$$

11.2 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,

$$D_{\mathbf{d}} S(u, v, w) = \sum_{\substack{i+j+k=d, \\ i, j, k \geq 0}} c_{i,j,k} D_{\mathbf{d}} b_{d,i,j,k}(u, v, w) \quad \text{for} \quad u + v + w = 1, \quad (11.1)$$

where

$$D_{\mathbf{d}} b_{d,i,j,k}(u, v, w) = r D_u b_{d,i,j,k}(u, v, w) + s D_v b_{d,i,j,k}(u, v, w) + t D_w b_{d,i,j,k}(u, v, w). \quad (11.2)$$

Cross derivatives are defined in the same way,

$$D_{\mathbf{d}_2} D_{\mathbf{d}_1} S(u, v, w) = \sum_{\substack{i+j+k=d, \\ i,j,k \geq 0}} c_{i,j,k} D_{\mathbf{d}_2} D_{\mathbf{d}_1} b_{d,i,j,k}(u, v, w) \quad \text{for } u+v+w=1,$$

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 [52] or [53].

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.

11.3 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 dimension. 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 7.2, 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 (7.2), and the functionality of the scalable set, definition (7.3).

We, therefore, first introduce the underlying basic ERBS basis function $B(u)$ for homogeneous barycentric coordinates.

Definition 11.3. *The underlying basic Expo-Rational B-splines for homogeneous barycentric coordinates is defined by $B(t) = B(t; \alpha, \beta, \gamma, \lambda, \sigma)$ as follows,*

$$B(t) = \begin{cases} S(\alpha, \beta, \gamma, \lambda, \sigma) \int_0^t \phi(s; \alpha, \beta, \gamma, \lambda, \sigma) ds, & \text{if } 0 < t \leq 1, \\ 0, & \text{otherwise} \end{cases}$$

where

$$\phi(t; \alpha, \beta, \gamma, \lambda, \sigma) = e^{-\beta \frac{|t-\lambda|^{2\sigma}}{(t(1-t))^\alpha}},$$

and the scaling factor

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

where

$$\alpha > 0, \quad \beta > 0, \quad \gamma > 0, \quad 0 \leq \lambda \leq 1, \quad \sigma \geq 0.$$

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 11.4. Given is a point $\mathbf{u} = (u_1, u_2, \dots, u_k) \in \Xi_{k-1}$ (in homogeneous barycentric coordinates, fulfilling the convexity property, see Definition 11.1). 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,

$$\mathcal{B}_{k,i}(\mathbf{u}) = \frac{B(u_i)}{\sum_{j=1}^k B(u_j)} \quad \text{for } k > 1 \quad \text{and} \quad i = 1, 2, \dots, k, \quad (11.3)$$

and where $B(u_i)$, $i = 1, 2, \dots, k$, is defined in definition 11.3.

For $k = 3$, i.e. Expo-Rational B-spline triangles, the basis functions will be,

$$\mathcal{B}_{3,i}(\mathbf{u}) = \frac{B(u_i)}{B(u_1) + B(u_2) + B(u_3)} \quad \text{for } i = 1, 2, 3.$$

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 11.3, 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 $i = 1, 2, \dots, k$ we have for the following six equations, first for the first-order partial derivatives, where

$$D_{u_i} \mathcal{B}_{k,i}(\mathbf{u}) = DB(u_i) \frac{\sum_{j=1}^k B(u_j) - B(u_i)}{\left(\sum_{j=1}^k B(u_j) \right)^2}, \quad (11.4)$$

and, for $j = 1, 2, \dots, k$, but where $j \neq i$,

$$D_{u_j} \mathcal{B}_{k,i}(\mathbf{u}) = DB(u_j) \frac{-B(u_i)}{\left(\sum_{j=1}^k B(u_j) \right)^2}. \quad (11.5)$$

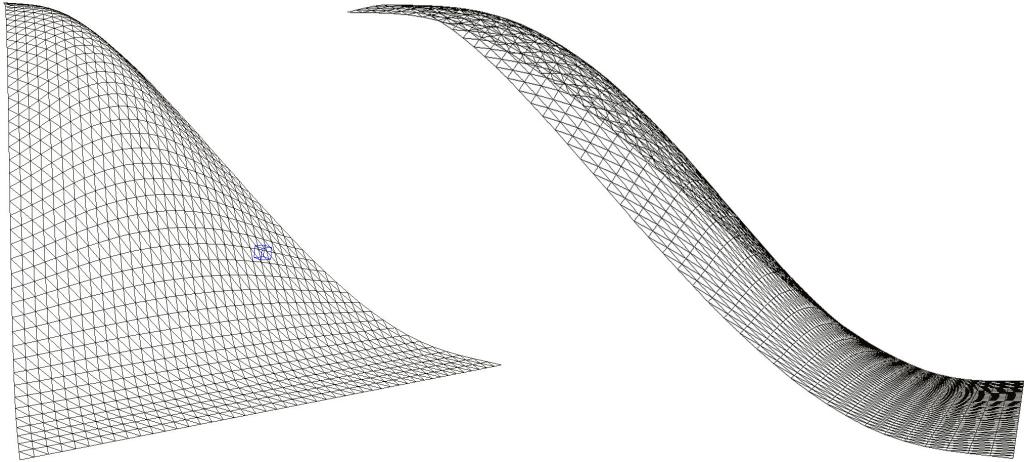


Figure 11.3: The Expo-Rational B-spline basis functions $\mathcal{B}_{k,i}(\mathbf{u})$ in homogeneous barycentric coordinates for triangles shown from two different angles.

For second order partial derivatives we have,

$$D_{u_i}^2 \mathcal{B}_{k,i}(\mathbf{u}) = \left(D^2 B(u_i) - \frac{2(DB(u_i))^2}{\sum_{j=1}^k B(u_j)} \right) \frac{\sum_{j=1}^k B(u_j) - B(u_i)}{\left(\sum_{j=1}^k B(u_j) \right)^2}, \quad (11.6)$$

and for $j = 1, 2, \dots, k$ but where $j \neq i$ for both

$$D_{u_j}^2 \mathcal{B}_{k,i}(\mathbf{u}) = \left(D^2 B(u_j) - \frac{2(DB(u_j))^2}{\sum_{j=1}^k B(u_j)} \right) \frac{-B(u_i)}{\left(\sum_{j=1}^k B(u_j) \right)^2}, \quad (11.7)$$

and

$$D_{u_i} D_{u_j} \mathcal{B}_{k,i}(\mathbf{u}) = DB(u_i) DB(u_j) \frac{2B(u_i) - \sum_{j=1}^k B(u_j)}{\left(\sum_{j=1}^k B(u_j) \right)^3}. \quad (11.8)$$

And finally, for $j = 1, 2, \dots, k$, but where $j \neq i$, and for $h = 1, 2, \dots, k$, but where $h \neq i, j$, we have

$$D_{u_h} D_{u_j} \mathcal{B}_{k,i}(\mathbf{u}) = DB(u_h) DB(u_j) \frac{2B(u_i)}{\left(\sum_{j=1}^k B(u_j) \right)^3}. \quad (11.9)$$

Given is a vector $\mathbf{d} \in \Upsilon_{k-1}$, i.e.,

$$\mathbf{d} = \mathbf{u}_1 - \mathbf{u}_2 = (d_1, d_2, \dots, d_k), \quad \text{where } \mathbf{u}_1 \text{ and } \mathbf{u}_2 \in \Xi_{k-1}.$$

We can now define the directional derivatives for the ERBS basis function in homogeneous barycentric coordinates. We now get

$$D_{\mathbf{d}} \mathcal{B}_{k,i}(\mathbf{u}) = \sum_{j=1}^k d_j D_{u_j} \mathcal{B}_{k,i}(\mathbf{u}). \quad (11.10)$$

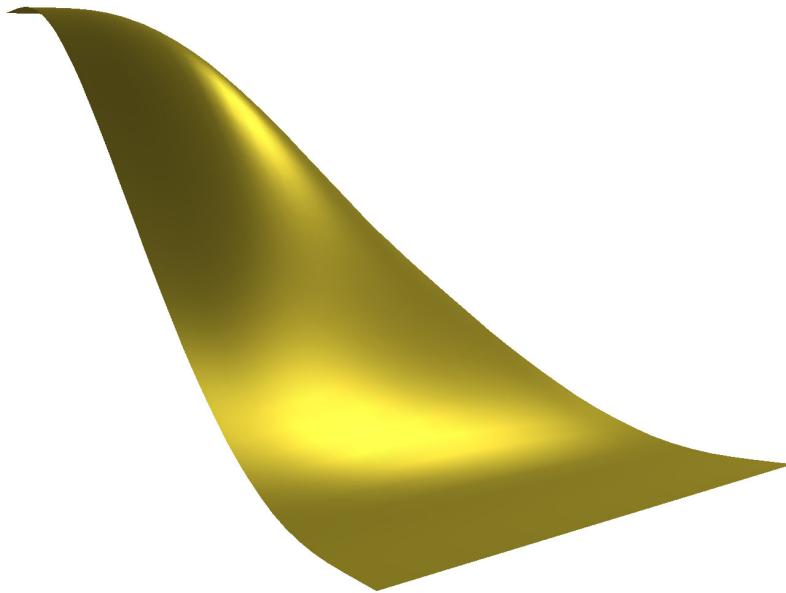


Figure 11.4: A shaded version of the Expo-Rational B-spline basis functions $\mathcal{B}_{k,i}(\mathbf{u})$ in homogeneous barycentric coordinates for triangles.

In Figure 11.4 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,

$$\begin{aligned}\mathbf{d}_1 &= (-1, 1, 0), \\ \mathbf{d}_2 &= (-1, 0, 1).\end{aligned}\tag{11.11}$$

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,

$$\begin{aligned}D_{\mathbf{d}_1} \mathcal{B}_{2,i}(u, v, w) &= D_v \mathcal{B}_{2,i}(u, v, w) - D_u \mathcal{B}_{2,i}(u, v, w), \\ D_{\mathbf{d}_2} \mathcal{B}_{2,i}(u, v, w) &= D_w \mathcal{B}_{2,i}(u, v, w) - D_u \mathcal{B}_{2,i}(u, v, w), \\ D_{\mathbf{d}_1}^2 \mathcal{B}_{2,i}(u, v, w) &= D_v^2 \mathcal{B}_{2,i}(u, v, w) - D_u D_v \mathcal{B}_{2,i}(u, v, w) + D_u^2 \mathcal{B}_{2,i}(u, v, w), \\ D_{\mathbf{d}_2}^2 \mathcal{B}_{2,i}(u, v, w) &= D_w^2 \mathcal{B}_{2,i}(u, v, w) - D_u D_w \mathcal{B}_{2,i}(u, v, w) + D_u^2 \mathcal{B}_{2,i}(u, v, w), \\ D_{\mathbf{d}_1} D_{\mathbf{d}_2} \mathcal{B}_{2,i}(u, v, w) &= D_v D_w \mathcal{B}_{2,i}(u, v, w) - D_u D_v \mathcal{B}_{2,i}(u, v, w) \\ &\quad - D_u D_w \mathcal{B}_{2,i}(u, v, w) + D_u^2 \mathcal{B}_{2,i}(u, v, w).\end{aligned}$$

It is clear that the properties for the generalized ERBS defined in Theorem 7.1 is also valid for the underlying basic ERBS for homogeneous barycentric coordinates in (def. 11.3).

Before looking at the properties of the sets of ERBS basis functions in homogeneous barycentric coordinates (def. 11.4), we will look at the following definition of corner points and points on an “edge”, in homogeneous barycentric coordinates.

Definition 11.5. *The corner points, denoted $\hat{\mathbf{u}}_i$, for $i = 1, 2, \dots, k$, are defined as follows,*

$$\hat{\mathbf{u}}_i \in \Xi_{k-1}, \quad \text{where the coordinate } u_i = 1,$$

and it follows that the rest of the coordinates $u_j = 0$, $j = 1, 2, \dots, k$ but where $j \neq i$.

The edge points, denoted $\bar{\mathbf{u}}_i$, for $i = 1, 2, \dots, k$, are defined as follows,

$$\bar{\mathbf{u}}_i \in \Xi_{k-1}, \quad \text{where the coordinate } u_i = 0.$$

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 11.4. The properties will be described in the following Theorem.

Theorem 11.1. *There are five important properties for the sets of the k ERBS basis-functions, $\mathcal{B}_{k,i}(\mathbf{u})$, $i = 1, 2, \dots, k$, in homogeneous barycentric coordinates, where $k = 2, 3, \dots$. For these properties it is assumed that $\mathbf{u} \in \Xi_{k-1}$ (def. 11.1), and $\hat{\mathbf{u}}_i$, for $i = 1, 2, \dots, k$ are corner points, and $\bar{\mathbf{u}}_i$, for $i = 1, 2, \dots, k$ are edge points (def. 11.5). 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. $\mathcal{B}_{k,i}(\hat{\mathbf{u}}_i) = 1,$

P4. $D_{\mathbf{d}_1}^r D_{\mathbf{d}_2}^s \mathcal{B}_{k,i}(\hat{\mathbf{u}}_j) = 0 \text{ for } j = 1, 2, \dots, k, \text{ where } \mathbf{d}_1, \mathbf{d}_2 \in \Upsilon_{k-1}, \text{ and } r+s > 0,$

P5. $D_{\mathbf{d}_1}^r D_{\mathbf{d}_2}^s \mathcal{B}_{k,i}(\bar{\mathbf{u}}_i) = 0 \text{ where } \mathbf{d}_1, \mathbf{d}_2 \in \Upsilon_{k-1}, \text{ and } r+s \geq 0.$

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 11.4. Because of property 1 in Theorem 7.1, 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

$$\sum_{i=1}^k \mathcal{B}_{k,i}(\mathbf{u}) = \frac{\sum_{i=1}^k B(u_i)}{\sum_{j=1}^k B(u_j)} = 1, \quad \text{if } \mathbf{u} \in \Xi_{k-1}.$$

3. Property 3 defines the “top” corner, and it follows from Theorem 7.1, property 2, because the denominator in equation (11.3) 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 11.3) are zero at $u = 0$ (which follows from property 1 in Theorem 7.1), and all their derivatives are zero at $u = 0$ and $u = 1$ (which follows from property 6 in Theorem 7.1). This can clearly be seen in the equations from (11.4) to (11.9). 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 (11.4) to (11.9) all have either a factor $B(u_i)$ or $D^j B(u_i)$ that is zero because $u_i = 0$ when $\mathbf{u} = \bar{\mathbf{u}}_i$. It is also clear that this will be the case also for higher derivatives.

□

11.4 ERBS triangles, definition and evaluators

The general formula for the ERBS triangle is,

$$S(u, v, w) = \sum_{i=1}^3 s_i(u, v, w) \mathcal{B}_{3,i}(u, v, w), \text{ where } u + v + w = 1 \text{ and } u, v, w \geq 0, \quad (11.12)$$

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 book, 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 7.7). In figures 11.5, 11.6 and 11.7 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 11.1) 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 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

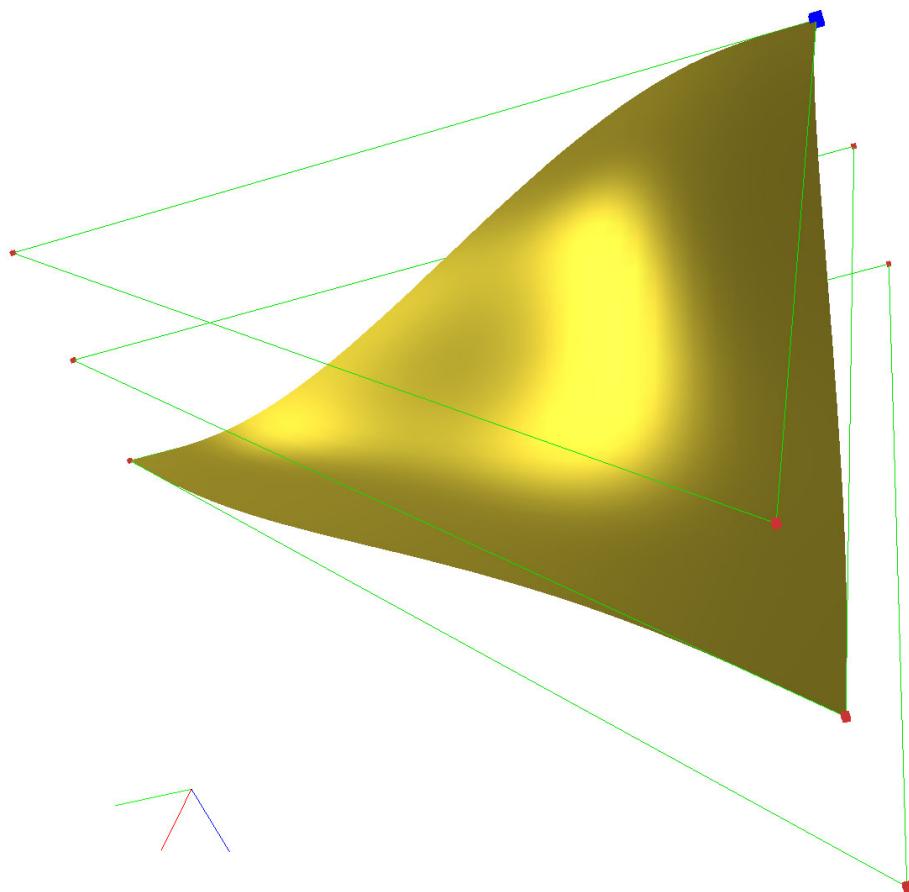


Figure 11.5: An Expo-Rational B-spline triangle surface with its three local triangles. The boundary of the local triangles are marked with green lines. As can be seen, the local triangles are all 1st degree Bézier triangular patches, and they are parallel to each other.

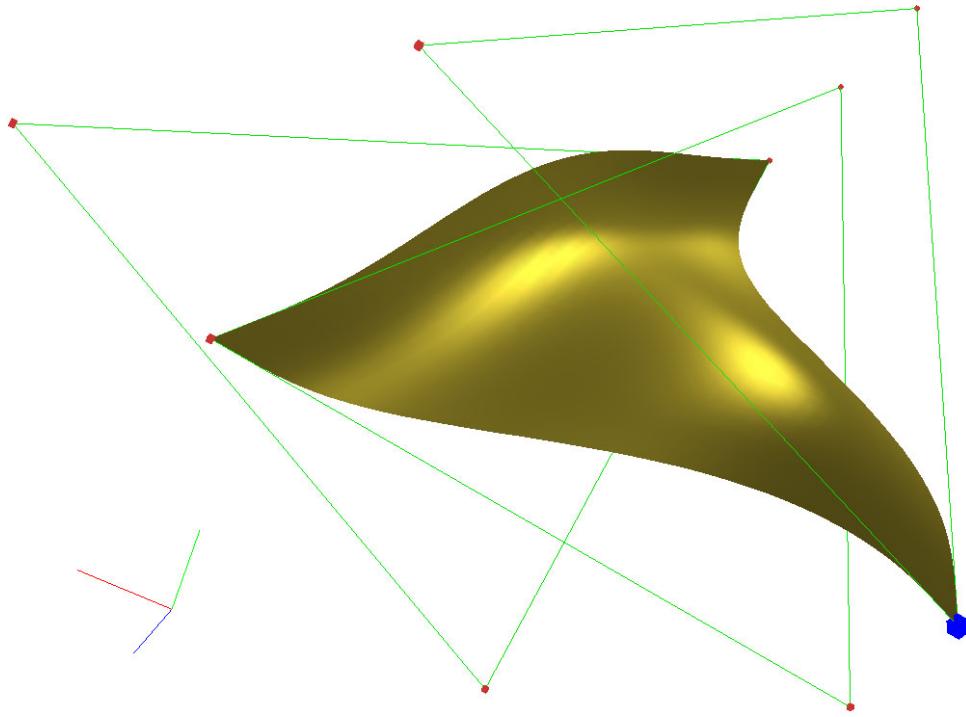


Figure 11.6: An Expo-Rational B-spline triangular surface where the local triangles initially were the same as in Figure 11.5, but are now translated and rotated. The blue cube shows us the interpolation point of the first local triangle.

mapping is implemented in the following example,

$$\begin{aligned} S(u, v, w) = & \bar{s}_1(u, v, w) \mathcal{B}_{3,1}(u, v, w) + \\ & \bar{s}_2(v, w, u) \mathcal{B}_{3,2}(u, v, w) + \\ & \bar{s}_3(w, u, v) \mathcal{B}_{3,3}(u, v, w), \end{aligned} \quad (11.13)$$

where $u + v + w = 1$ and $u, v, w \geq 0$, and where \bar{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,

$$\begin{aligned} D_{\mathbf{d}} S(u, v, w) = & D_{\mathbf{d}} \bar{s}_1(u, v, w) \mathcal{B}_{3,1}(u, v, w) + \bar{s}_1(u, v, w) D_{\mathbf{d}} \mathcal{B}_{3,1}(u, v, w) + \\ & D_{\mathbf{d}} \bar{s}_2(v, w, u) \mathcal{B}_{3,2}(u, v, w) + \bar{s}_2(v, w, u) D_{\mathbf{d}} \mathcal{B}_{3,2}(u, v, w) + \\ & D_{\mathbf{d}} \bar{s}_3(w, u, v) \mathcal{B}_{3,3}(u, v, w) + \bar{s}_3(w, u, v) D_{\mathbf{d}} \mathcal{B}_{3,3}(u, v, w), \end{aligned} \quad (11.14)$$

where $D_{\mathbf{d}} \mathcal{B}_{3,i}(u, v, w)$, $i = 1, 2, 3$ is described in (11.10). If the local triangles $\bar{s}_i(u, v, w)$, $i = 1, 2, 3$ are Bézier triangles, then the directional derivatives can be found in (11.1).

Recall the definition of the two main directions for derivatives in (11.11), $\mathbf{d}_1 = (-1, 1, 0)$ 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

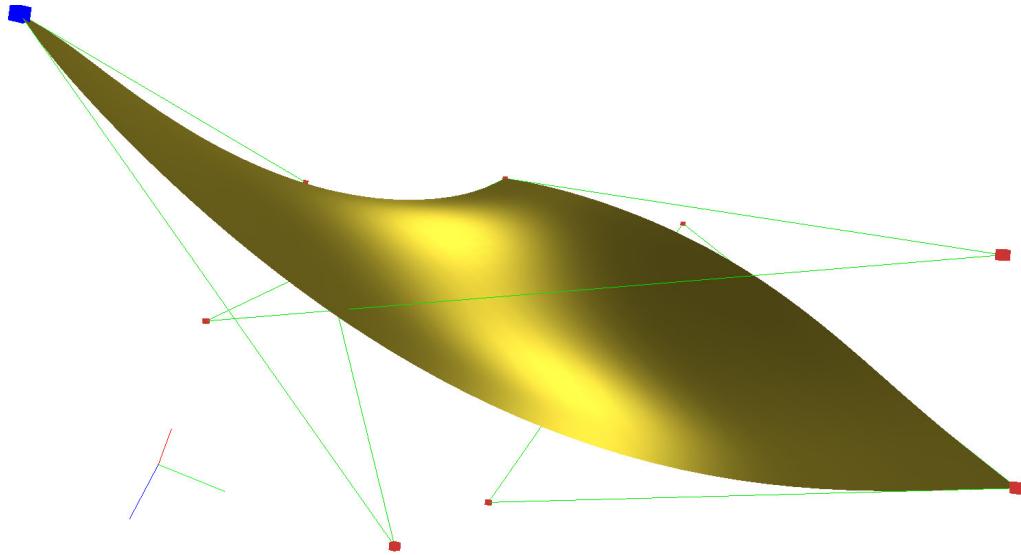


Figure 11.7: An Expo-Rational B-spline triangular surface where also the local triangles are all 1st degree Bézier triangular patches as in Figure 11.5 and in Figure 11.6. The shape can vary tremendous only by affine transformations of the local patches.

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,

$$\begin{aligned}
 D_{\mathbf{d}_1} S(u, v, w) = & (D_2 \bar{s}_1(u, v, w) - D_1 \bar{s}_1(u, v, w)) \mathcal{B}_{3,1}(u, v, w) \\
 & + \bar{s}_1(u, v, w) (D_v \mathcal{B}_{3,1}(u, v, w) - D_u \mathcal{B}_{3,1}(u, v, w)) \\
 & + (D_1 \bar{s}_2(v, w, u) - D_3 \bar{s}_2(v, w, u)) \mathcal{B}_{3,2}(u, v, w) \\
 & + \bar{s}_2(v, w, u) (D_v \mathcal{B}_{3,2}(u, v, w) - D_u \mathcal{B}_{3,2}(u, v, w)) \\
 & + (D_3 \bar{s}_3(w, u, v) - D_2 \bar{s}_3(w, u, v)) \mathcal{B}_{3,3}(u, v, w) \\
 & + \bar{s}_3(w, u, v) (D_v \mathcal{B}_{3,3}(u, v, w) - D_u \mathcal{B}_{3,3}(u, v, w)).
 \end{aligned} \tag{11.15}$$

For the derivatives in the second main direction the formula is,

$$\begin{aligned}
 D_{\mathbf{d}_2} S(u, v, w) = & (D_3 \bar{s}_1(u, v, w) - D_1 \bar{s}_1(u, v, w)) \mathcal{B}_{3,1}(u, v, w) \\
 & + \bar{s}_1(u, v, w) (D_w \mathcal{B}_{3,1}(u, v, w) - D_u \mathcal{B}_{3,1}(u, v, w)) \\
 & + (D_2 \bar{s}_2(v, w, u) - D_3 \bar{s}_2(v, w, u)) \mathcal{B}_{3,2}(u, v, w) \\
 & + \bar{s}_2(v, w, u) (D_w \mathcal{B}_{3,2}(u, v, w) - D_u \mathcal{B}_{3,2}(u, v, w)) \\
 & + (D_1 \bar{s}_3(w, u, v) - D_2 \bar{s}_3(w, u, v)) \mathcal{B}_{3,3}(u, v, w) \\
 & + \bar{s}_3(w, u, v) (D_w \mathcal{B}_{3,3}(u, v, w) - D_u \mathcal{B}_{3,3}(u, v, w)).
 \end{aligned} \tag{11.16}$$

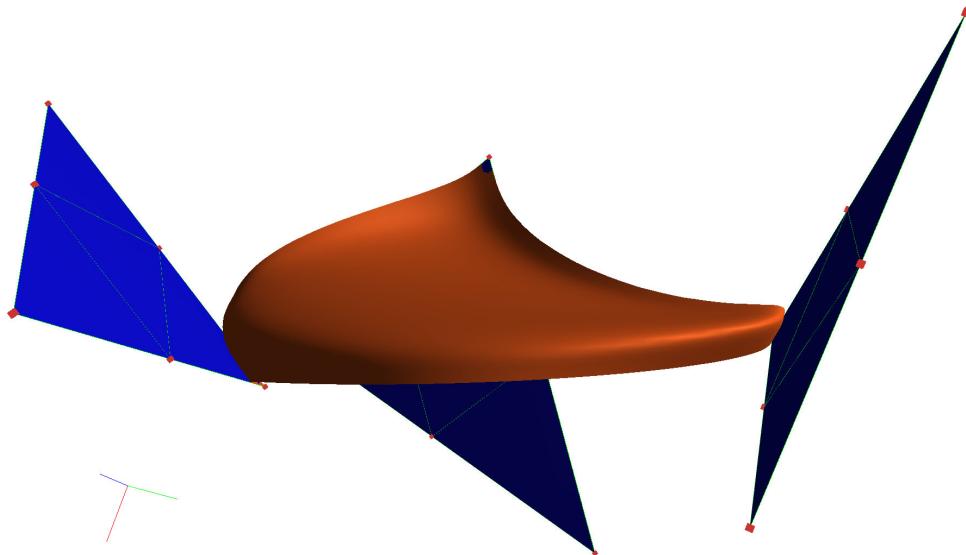


Figure 11.8: An Expo-Rational B-spline triangular surface is plotted. The local triangles are 2nd degree Bézier triangles (blue), but still planar. The control polygons of the Bézier triangles are shown as green lines, and the control points as red cubes.

For the second order derivatives, the process has to be repeated ones more. We then get,

$$\begin{aligned}
 D_{\mathbf{d}_1}^2 S(u, v, w) = & \left(D_2^2 \bar{s}_1(u, v, w) - 2D_2 D_1 \bar{s}_1(u, v, w) + D_1^2 \bar{s}_1(u, v, w) \right) \mathcal{B}_{3,1}(u, v, w) \\
 & + 2(D_2 \bar{s}_1(u, v, w) - D_1 \bar{s}_1(u, v, w)) (D_v \mathcal{B}_{3,1}(u, v, w) - D_u \mathcal{B}_{3,1}(u, v, w)) \\
 & + \bar{s}_1(u, v, w) (D_v^2 \mathcal{B}_{3,1}(u, v, w) - 2D_u D_v \mathcal{B}_{3,1}(u, v, w) + D_u^2 \mathcal{B}_{3,1}(u, v, w)) \\
 & + \left(D_2^2 \bar{s}_2(v, w, u) - 2D_2 D_3 \bar{s}_2(v, w, u) + D_3^2 \bar{s}_2(v, w, u) \right) \mathcal{B}_{3,2}(u, v, w) \\
 & + 2(D_2 \bar{s}_2(v, w, u) - D_1 \bar{s}_2(v, w, u)) (D_v \mathcal{B}_{3,2}(u, v, w) - D_u \mathcal{B}_{3,2}(u, v, w)) \\
 & + \bar{s}_2(v, w, u) (D_v^2 \mathcal{B}_{3,2}(u, v, w) - 2D_u D_v \mathcal{B}_{3,2}(u, v, w) + D_u^2 \mathcal{B}_{3,2}(u, v, w)) \\
 & + \left(D_3^2 \bar{s}_3(w, u, v) - 2D_1 D_3 \bar{s}_3(w, u, v) + D_1^2 \bar{s}_3(w, u, v) \right) \mathcal{B}_{3,3}(u, v, w) \\
 & + 2(D_3 \bar{s}_3(w, u, v) - D_1 \bar{s}_3(w, u, v)) (D_v \mathcal{B}_{3,3}(u, v, w) - D_u \mathcal{B}_{3,3}(u, v, w)) \\
 & + \bar{s}_3(w, u, v) (D_v^2 \mathcal{B}_{3,3}(u, v, w) - 2D_u D_v \mathcal{B}_{3,3}(u, v, w) + D_u^2 \mathcal{B}_{3,3}(u, v, w)). \tag{11.17}
 \end{aligned}$$

The other second order derivatives, $D_{\mathbf{d}_2}^2 S(u, v, w)$ 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 11.8 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 11.5, 11.6 and 11.7, 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 11.8, it is possible to edit the local

Bezier triangle into a “true” quadratic patch, which was impossible to do in the case of figures 11.5, 11.6 and 11.7. This is, actually what we have done in the next figure.

In Figure 11.9 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.

11.5 Local Bézier triangles and Hermite interpolation

We start by recalling the settings from section 7.9, 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 $g : \Omega_g \subset \mathbb{R}^2 \rightarrow \mathbb{R}^n$, where we might have $n = 1, 2, 3, \dots$,
- Given are three points in the domain of g i.e. Ω_g . These points are denoted by p_k , $k = 1, 2, 3$ (they can be defined by either ordinary coordinates (μ, v) 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 Ω_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 11.13 and the description below 11.13) in such a way that the ERBS triangle is Hermite interpolating the local triangles in each corner, so that: at the first corner

$$D_{(-1,1,0)}^i D_{(-1,0,1)}^j \bar{s}_1(1,0,0) = D_{(-1,1,0)}^i D_{(-1,0,1)}^j S(1,0,0) = D_{p_2-p_1}^i D_{p_3-p_1}^j g(p_1), \quad (11.18)$$

at the second corner:

$$D_{(-1,1,0)}^i D_{(-1,0,1)}^j \bar{s}_2(1,0,0) = D_{(0,-1,1)}^i D_{(1,-1,0)}^j S(0,1,0) = D_{p_3-p_2}^i D_{p_1-p_2}^j g(p_2), \quad (11.19)$$

and at the third corner:

$$D_{(-1,1,0)}^i D_{(-1,0,1)}^j \bar{s}_3(1,0,0) = D_{(1,0,-1)}^i D_{(0,1,-1)}^j S(0,0,1) = D_{p_1-p_3}^i D_{p_2-p_3}^j g(p_3), \quad (11.20)$$

is fulfilled, where for the respective three equations, it holds

$$0 \leq (i + j) \leq d_k, \quad \text{for } k = 1, 2, 3. \quad (11.21)$$

The right hand side of (11.18–11.20) looks like the ordinary Hermite interpolation method used for generating an approximation of a triangular part of a parametric surface. The specific part is, however, the generation of the local triangles on the left hand side of (11.18–11.20), and thus the rotational global/local mapping. Remember from Theorem

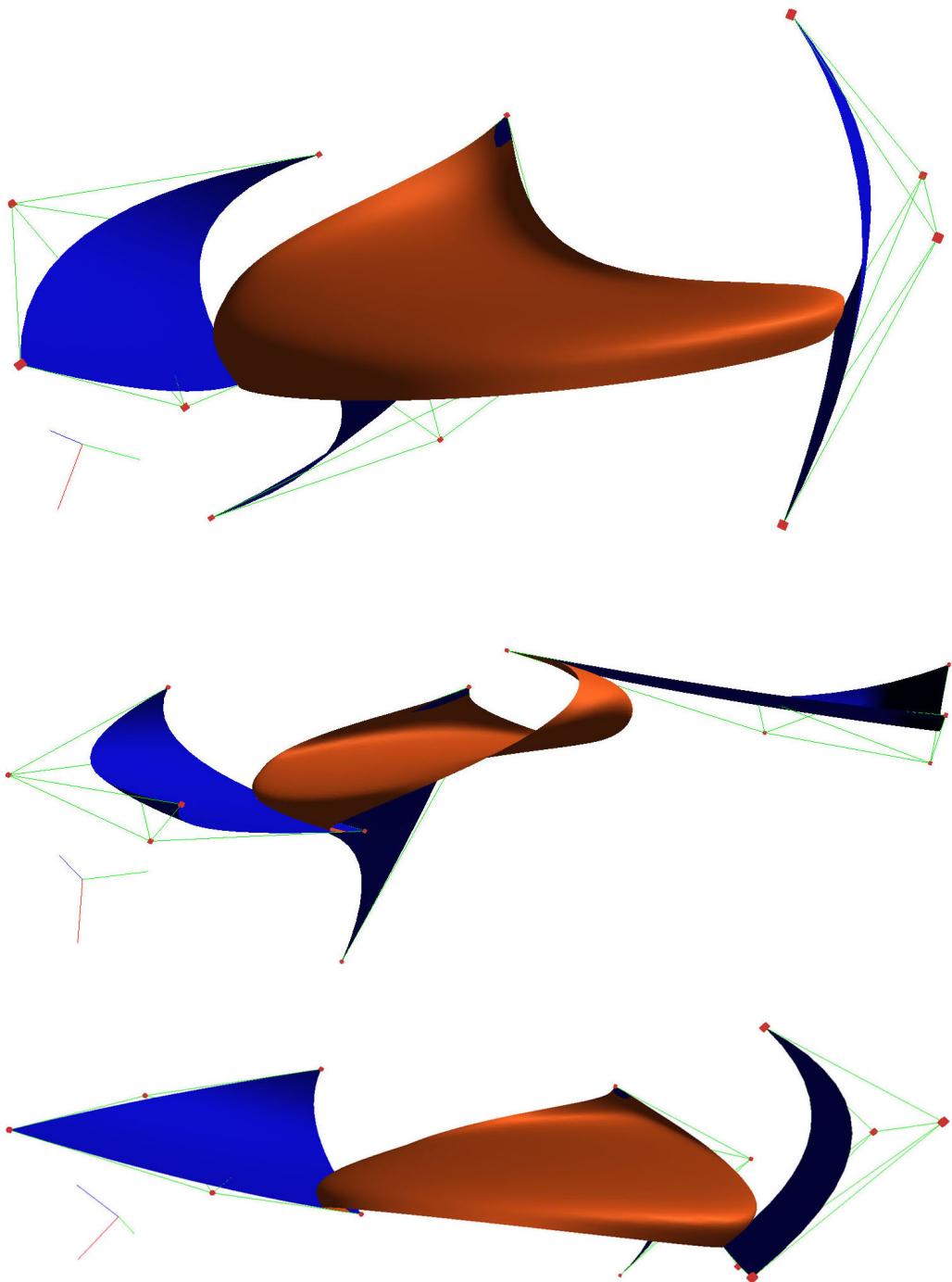


Figure 11.9: Three different Expo-Rational B-spline triangular surfaces are plotted. The local triangles are 2nd degree Bézier triangles (not planar any more), and we can see them as blue triangles, the control polygons of the local Bézier triangles as green lines, and their control points as red cubes.

11.1 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 11.1. This is, of course, the basis for the equations (11.18–11.21), and it can also clearly be seen in figures 11.5–11.9. 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 (11.18–11.20), together with the rightmost side of the equations (11.18–11.20). We then get,

$$D_{(-1,1,0)}^i D_{(-1,0,1)}^j s(1,0,0) = D_{\mathbf{d}_1}^i D_{\mathbf{d}_2}^j g(\mathbf{p}), \quad \text{where } 0 \leq (i+j) \leq d, \quad (11.22)$$

where d is the degree of the Bézier triangle, \mathbf{p} is a point in Ω_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 (11.22) 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 (11.22) 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 (11.1) and (11.2). 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,

$$D_{\mathbf{p}} s(\mathbf{u}) = p_1 D_u s(\mathbf{u}) + p_2 D_v s(\mathbf{u}) + p_3 D_w s(\mathbf{u}), \quad (11.23)$$

$$\begin{aligned} D_{\mathbf{q}} D_{\mathbf{p}} s(\mathbf{u}) = & q_1 p_1 D_u^2 s(\mathbf{u}) + q_1 p_2 D_u D_v s(\mathbf{u}) + q_1 p_3 D_u D_w s(\mathbf{u}) \\ & + q_2 p_1 D_u D_v s(\mathbf{u}) + q_2 p_2 D_v^2 s(\mathbf{u}) + q_2 p_3 D_v D_w s(\mathbf{u}) \\ & + q_3 p_1 D_u D_w s(\mathbf{u}) + q_3 p_2 D_v D_w s(\mathbf{u}) + q_3 p_3 D_w^2 s(\mathbf{u}), \end{aligned} \quad (11.24)$$

$$\begin{aligned}
D_{\mathbf{h}} D_{\mathbf{q}} D_{\mathbf{p}} s(\mathbf{u}) = & h_1 q_1 p_1 D_u^3 s(\mathbf{u}) \\
& + (h_1 q_1 p_2 + h_1 q_2 p_1 + h_2 q_1 p_1) D_u^2 D_v s(\mathbf{u}) \\
& + (h_1 q_2 p_2 + h_2 q_1 p_2 + h_2 q_2 p_1) D_u D_v^2 s(\mathbf{u}) \\
& + (h_1 q_1 p_3 + h_1 q_3 p_1 + h_3 q_1 p_1) D_u^2 D_w s(\mathbf{u}) \\
& + (h_1 q_3 p_3 + h_3 q_1 p_3 + h_3 q_3 p_1) D_u D_w^2 s(\mathbf{u}) \quad (11.25) \\
& + h_2 q_2 p_2 D_v^3 s(\mathbf{u}) \\
& + (h_2 q_2 p_3 + h_2 q_3 p_2 + h_3 q_2 p_2) D_v^2 D_w s(\mathbf{u}) \\
& + (h_2 q_3 p_3 + h_3 q_2 p_3 + h_3 q_3 p_2) D_v D_w^2 s(\mathbf{u}) \\
& + h_3 q_3 p_3 D_w^3 s(\mathbf{u}).
\end{aligned}$$

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

$$\begin{aligned}
D_u s(u, v, w) &= \mathbf{c}_1, \\
D_v s(u, v, w) &= \mathbf{c}_2, \\
D_w s(u, v, w) &= \mathbf{c}_3,
\end{aligned}$$

and using the equation and the partial derivatives together with equation (11.23), 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 (11.24), we get

$$\begin{aligned}
 s(1, 0, 0) &= \mathbf{c}_1, \\
 D_{(-1,1,0)} s(1, 0, 0) &= 2(\mathbf{c}_2 - \mathbf{c}_1), \\
 D_{(-1,0,1)} s(1, 0, 0) &= 2(\mathbf{c}_3 - \mathbf{c}_1), \\
 D_{(-1,1,0)}^2 s(1, 0, 0) &= 2(\mathbf{c}_1 - 2\mathbf{c}_2 + \mathbf{c}_4), \\
 D_{(-1,0,1)}^2 s(1, 0, 0) &= 2(\mathbf{c}_1 - 2\mathbf{c}_3 + \mathbf{c}_6), \\
 D_{(-1,0,1)} D_{(-1,1,0)} s(1, 0, 0) &= 2(\mathbf{c}_1 - \mathbf{c}_2 - \mathbf{c}_3 + \mathbf{c}_5).
 \end{aligned}$$

Reorganizing this, the formula for the Hermite interpolation of a parametric surface by the 2nd degree Bézier triangle becomes

$$\begin{aligned}
 \mathbf{c}_1 &= g(\mathbf{p}), \\
 \mathbf{c}_2 &= \mathbf{c}_1 + \frac{1}{2} D_{\mathbf{d}_1} g(\mathbf{p}), \\
 \mathbf{c}_3 &= \mathbf{c}_1 + \frac{1}{2} D_{\mathbf{d}_2} g(\mathbf{p}), \\
 \mathbf{c}_4 &= -\mathbf{c}_1 + 2\mathbf{c}_2 + \frac{1}{2} D_{\mathbf{d}_1}^2 g(\mathbf{p}), \\
 \mathbf{c}_5 &= -\mathbf{c}_1 + \mathbf{c}_2 + \mathbf{c}_3 + \frac{1}{2} D_{\mathbf{d}_1} D_{\mathbf{d}_2} g(\mathbf{p}), \\
 \mathbf{c}_6 &= -\mathbf{c}_1 + 2\mathbf{c}_3 + \frac{1}{2} D_{\mathbf{d}_2}^2 g(\mathbf{p}).
 \end{aligned}$$

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

$$\begin{aligned}
 s(u, v, w) &= u^3 \mathbf{c}_1 + 3u^2 v \mathbf{c}_2 + 3u^2 w \mathbf{c}_3 + 3uv^2 \mathbf{c}_4 + 6uvw \mathbf{c}_5 \\
 &\quad + 3uw^2 \mathbf{c}_6 + v^3 \mathbf{c}_7 + 3v^2 w \mathbf{c}_8 + 3vw^2 \mathbf{c}_9 + w^3 \mathbf{c}_{10}.
 \end{aligned}$$

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 (11.25), we get

$$\begin{aligned}
s(1, 0, 0) &= \mathbf{c}_1, \\
D_{(-1,1,0)} s(1, 0, 0) &= 3(\mathbf{c}_2 - \mathbf{c}_1), \\
D_{(-1,0,1)} s(1, 0, 0) &= 3(\mathbf{c}_3 - \mathbf{c}_1), \\
D_{(-1,1,0)}^2 s(1, 0, 0) &= 6(\mathbf{c}_1 - 2\mathbf{c}_2 + \mathbf{c}_4), \\
D_{(-1,0,1)}^2 s(1, 0, 0) &= 6(\mathbf{c}_1 - 2\mathbf{c}_3 + \mathbf{c}_6), \\
D_{(-1,0,1)} D_{(-1,1,0)} s(1, 0, 0) &= 6(\mathbf{c}_1 - \mathbf{c}_2 - \mathbf{c}_3 + \mathbf{c}_5), \\
D_{(-1,1,0)}^3 s(1, 0, 0) &= 6(-\mathbf{c}_1 + 3\mathbf{c}_2 - 3\mathbf{c}_4 + \mathbf{c}_7), \\
D_{(-1,0,1)}^3 s(1, 0, 0) &= 6(-\mathbf{c}_1 + 3\mathbf{c}_3 - 3\mathbf{c}_6 + \mathbf{c}_{10}), \\
D_{(-1,1,0)} D_{(-1,0,1)}^2 s(1, 0, 0) &= 6(-\mathbf{c}_1 + \mathbf{c}_2 + 2\mathbf{c}_3 - 2\mathbf{c}_5 - \mathbf{c}_6 + \mathbf{c}_9), \\
D_{(-1,0,1)} D_{(-1,1,0)}^2 s(1, 0, 0) &= 6(-\mathbf{c}_1 + 2\mathbf{c}_2 + \mathbf{c}_3 - \mathbf{c}_4 - 2\mathbf{c}_5 + \mathbf{c}_8).
\end{aligned}$$

Reorganizing this, the formula for the Hermite interpolation of a parametric surface by

the 3rd degree Bézier triangle becomes

$$\begin{aligned}
 \mathbf{c}_1 &= g(\mathbf{p}), \\
 \mathbf{c}_2 &= \mathbf{c}_1 + \frac{1}{3} D_{\mathbf{d}_1} g(\mathbf{p}), \\
 \mathbf{c}_3 &= \mathbf{c}_1 + \frac{1}{3} D_{\mathbf{d}_2} g(\mathbf{p}), \\
 \mathbf{c}_4 &= -\mathbf{c}_1 + 2\mathbf{c}_2 + \frac{1}{6} D_{\mathbf{d}_1}^2 g(\mathbf{p}), \\
 \mathbf{c}_5 &= -\mathbf{c}_1 + \mathbf{c}_2 + \mathbf{c}_3 + \frac{1}{6} D_{\mathbf{d}_1} D_{\mathbf{d}_2} g(\mathbf{p}), \\
 \mathbf{c}_6 &= -\mathbf{c}_1 + 2\mathbf{c}_3 + \frac{1}{6} D_{\mathbf{d}_2}^2 g(\mathbf{p}), \\
 \mathbf{c}_7 &= \mathbf{c}_1 - 3\mathbf{c}_2 + 3\mathbf{c}_4 + \frac{1}{6} D_{\mathbf{d}_1}^3 g(\mathbf{p}), \\
 \mathbf{c}_8 &= \mathbf{c}_1 - 2\mathbf{c}_2 - \mathbf{c}_3 + \mathbf{c}_4 + 2\mathbf{c}_5 + \frac{1}{6} D_{\mathbf{d}_1}^2 D_{\mathbf{d}_2} g(\mathbf{p}), \\
 \mathbf{c}_9 &= \mathbf{c}_1 - \mathbf{c}_2 - 2\mathbf{c}_3 + 2\mathbf{c}_5 + \mathbf{c}_6 + \frac{1}{6} D_{\mathbf{d}_1} D_{\mathbf{d}_2}^2 g(\mathbf{p}), \\
 \mathbf{c}_{10} &= \mathbf{c}_1 - 3\mathbf{c}_3 + 3\mathbf{c}_6 + \frac{1}{6} D_{\mathbf{d}_2}^3 g(\mathbf{p}).
 \end{aligned}$$

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:

$$\begin{aligned}
 D_{\mathbf{d}_1}^2 g(\mathbf{p}) &= u_1 D_u D_{\mathbf{d}_1} g(\mathbf{p}) + v_1 D_v D_{\mathbf{d}_1} g(\mathbf{p}), \\
 D_{\mathbf{d}_2}^2 g(\mathbf{p}) &= u_2 D_u D_{\mathbf{d}_2} g(\mathbf{p}) + v_2 D_v D_{\mathbf{d}_2} g(\mathbf{p}), \\
 D_{\mathbf{d}_1} D_{\mathbf{d}_2} g(\mathbf{p}) &= u_1 D_u D_{\mathbf{d}_2} g(\mathbf{p}) + v_1 D_v D_{\mathbf{d}_2} g(\mathbf{p}).
 \end{aligned}$$

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 11.10, 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 elsewhere 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 (10.25). In Figure 11.11, we can see three different views of four ERBS triangles computed by Hermite interpolation of a torus,

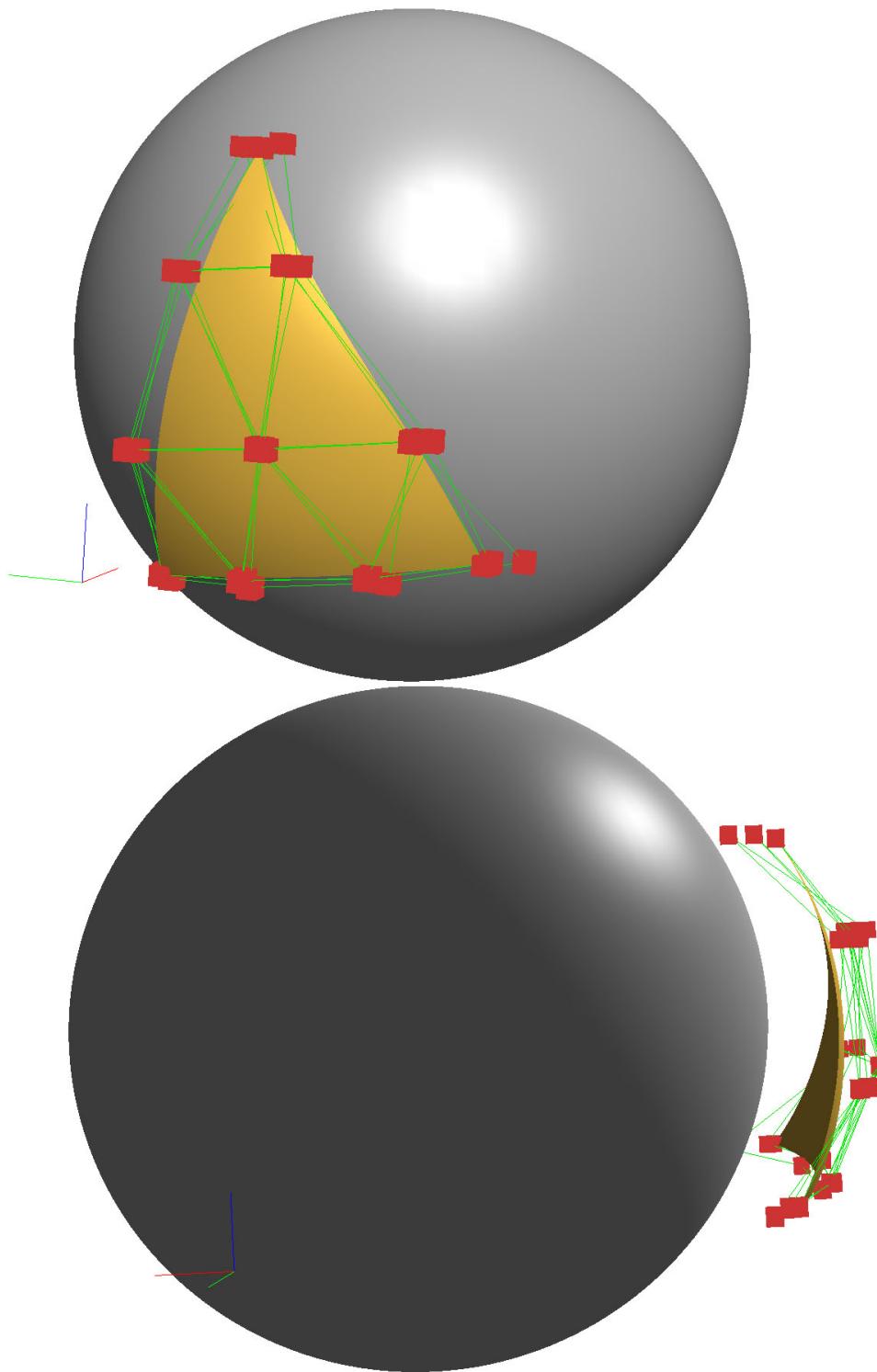


Figure 11.10: Two views of a sphere and one Expo-Rational B-spline triangular surface interpolating a part of the sphere. The ERBS triangle is slightly translated away from the sphere. The local triangles are 3rd degree Bézier triangles, and we can see all the control polygons of the local Bézier triangles as green lines, and the control points as red cubes.

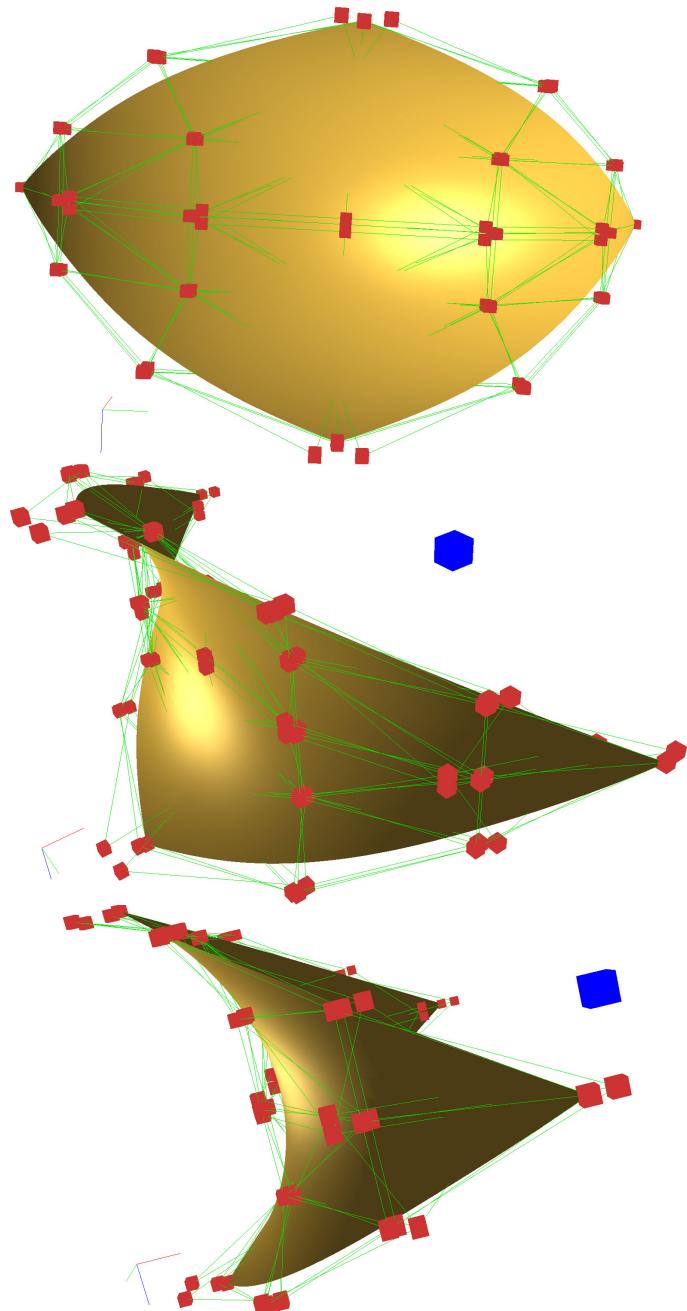


Figure 11.11: Three different views of a surface that is composed by four ERBS triangles Hermite interpolating a part of a torus, equation (10.25), at 5 points. The 4×3 local triangles are 3rd degree Bézier triangles; we can see their control polygons as green lines, and the control points as red cubes. The blue cube is the center of the original torus.

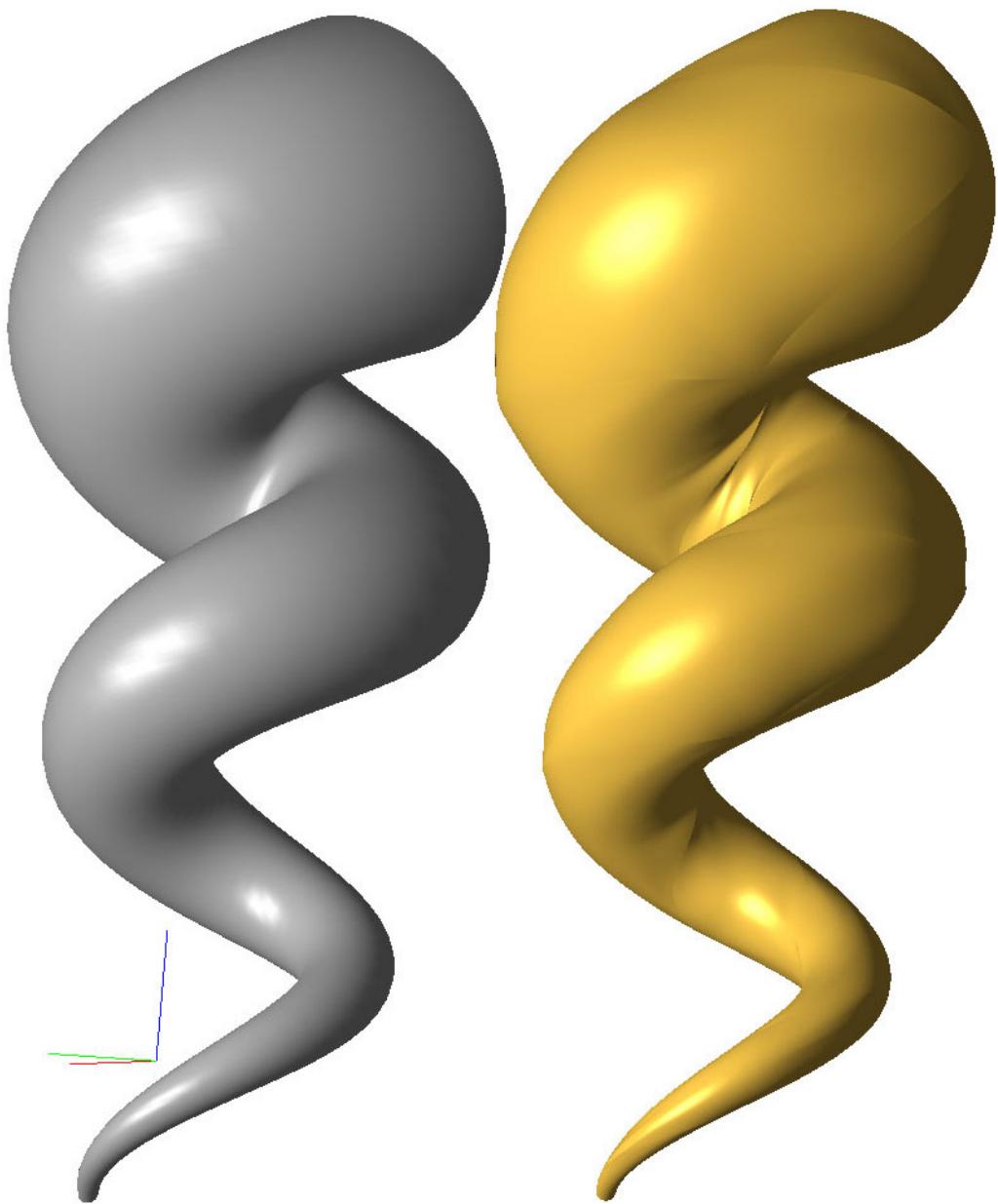


Figure 11.12: On the left hand side there is a plot of a Seashell, equation (10.26), and on the right hand side there is a plot of a set of 80 ERBS triangles, Hermite interpolating a Seashell at 44 different points. The 80 ERBS triangles are all independent of each other, but are “continuously connected”; this means that there are no holes in the composite surface after the interpolation.

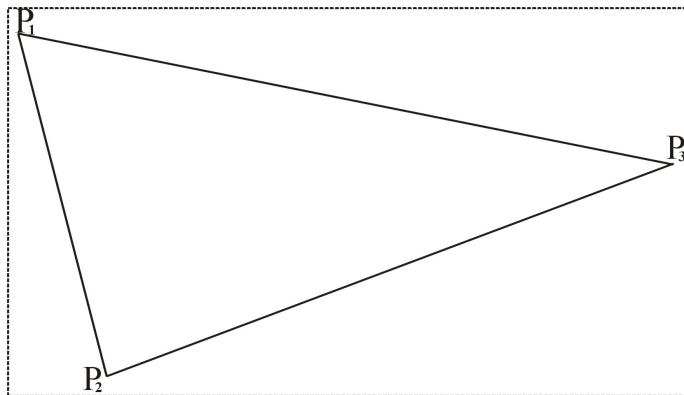


Figure 11.13: The parameter plane Ω of a surface $\bar{S}(\mathbf{u}) : \Omega \subset \mathbb{R}^2 \rightarrow \mathbb{R}^n$, $n > 0$. The three points $\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3 \in \Omega$ describe a triangle in the parameter plane.

(10.25), at totally 5 different points. The composition of the four ERBS triangles is clearly continuous. Although this cannot be easily seen, the composition is actually based on four different triangles. But even if they together look as being G^∞ (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 (10.26). On the left hand side of Figure 11.12 there is a plot of a “Sea Shell” surface, (10.26), 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^∞ . It is actually G^∞ 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 11.8).

11.6 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 $\bar{S} : \Omega \subset \mathbb{R}^2 \rightarrow \mathbb{R}^n$, 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 \bar{S} , see Figure 11.13).

To clarify the notation, we have the surface,

$$\bar{S}(\mathbf{u}) = \bar{S}(u_1, u_2) \in \mathbb{R}^n,$$

and the differential,

$$d\bar{S}_{\mathbf{u}} = [D_{u_1} \bar{S}(\mathbf{u}) \quad D_{u_2} \bar{S}(\mathbf{u})] \in \mathbb{R}^{n \times 2}.$$

A triangular surface is then defined by,

$$S(u, v, w) = \bar{S}(u\mathbf{p}_1 + v\mathbf{p}_2 + w\mathbf{p}_3).$$

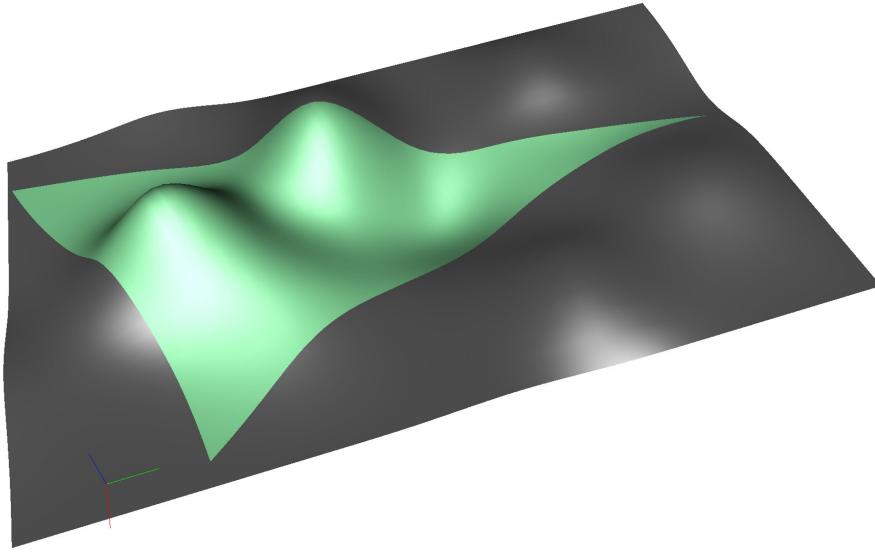


Figure 11.14: The grey surface is a B-spline tensor product surface. The triangular green surface is defined by three points in the parameter plane of the B-spline tensor product surface, and its domain is the minimum convex set including these three points (that is, a triangle in the parametric domain).

If we denote

$$\tilde{\mathbf{u}} = u\mathbf{p}_1 + v\mathbf{p}_2 + w\mathbf{p}_3,$$

where the three points are

$$\mathbf{p}_i = \begin{pmatrix} p_{i1} \\ p_{i2} \end{pmatrix} \quad \text{for } i = 1, 2, 3.$$

We then get the first order partial derivatives,

$$\begin{aligned} D_u S(u, v, w) &= d\bar{S}_{\tilde{\mathbf{u}}}(\mathbf{p}_1) = D_{u_1} \bar{S}(\tilde{\mathbf{u}}) p_{11} + D_{u_2} \bar{S}(\tilde{\mathbf{u}}) p_{12}, \\ D_v S(u, v, w) &= d\bar{S}_{\tilde{\mathbf{u}}}(\mathbf{p}_2) = D_{u_1} \bar{S}(\tilde{\mathbf{u}}) p_{21} + D_{u_2} \bar{S}(\tilde{\mathbf{u}}) p_{22}, \\ D_w S(u, v, w) &= d\bar{S}_{\tilde{\mathbf{u}}}(\mathbf{p}_3) = D_{u_1} \bar{S}(\tilde{\mathbf{u}}) p_{31} + D_{u_2} \bar{S}(\tilde{\mathbf{u}}) p_{32}, \end{aligned}$$

and the second order partial derivatives,

$$\begin{aligned} D_u^2 S(u, v, w) &= D_{u_1}^2 \bar{S}(\tilde{\mathbf{u}}) p_{11}^2 + 2D_{u_1} D_{u_2} \bar{S}(\tilde{\mathbf{u}}) p_{11} p_{12} + D_{u_2}^2 \bar{S}(\tilde{\mathbf{u}}) p_{12}^2, \\ D_v^2 S(u, v, w) &= D_{u_1}^2 \bar{S}(\tilde{\mathbf{u}}) p_{21}^2 + 2D_{u_1} D_{u_2} \bar{S}(\tilde{\mathbf{u}}) p_{21} p_{22} + D_{u_2}^2 \bar{S}(\tilde{\mathbf{u}}) p_{22}^2, \\ D_w^2 S(u, v, w) &= D_{u_1}^2 \bar{S}(\tilde{\mathbf{u}}) p_{31}^2 + 2D_{u_1} D_{u_2} \bar{S}(\tilde{\mathbf{u}}) p_{31} p_{32} + D_{u_2}^2 \bar{S}(\tilde{\mathbf{u}}) p_{32}^2, \\ D_u D_v S(u, v, w) &= D_{u_1}^2 \bar{S}(\tilde{\mathbf{u}}) p_{11} p_{21} + D_{u_1} D_{u_2} \bar{S}(\tilde{\mathbf{u}}) (p_{11} p_{22} + p_{12} p_{21}) + D_{u_2}^2 \bar{S}(\tilde{\mathbf{u}}) p_{12} p_{22}, \\ D_u D_w S(u, v, w) &= D_{u_1}^2 \bar{S}(\tilde{\mathbf{u}}) p_{11} p_{31} + D_{u_1} D_{u_2} \bar{S}(\tilde{\mathbf{u}}) (p_{11} p_{32} + p_{12} p_{31}) + D_{u_2}^2 \bar{S}(\tilde{\mathbf{u}}) p_{12} p_{32}, \\ D_v D_w S(u, v, w) &= D_{u_1}^2 \bar{S}(\tilde{\mathbf{u}}) p_{21} p_{31} + D_{u_1} D_{u_2} \bar{S}(\tilde{\mathbf{u}}) (p_{21} p_{32} + p_{22} p_{31}) + D_{u_2}^2 \bar{S}(\tilde{\mathbf{u}}) p_{22} p_{32}. \end{aligned}$$

Notice that when using the two last sets of formulas, all formulas in section 11.4, the formulas (11.12) to (11.17), are valid.

In Figure 11.14 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 11.13.

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.*

11.7 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 $\mathbf{x} : U \rightarrow V \cap G$ 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 329, besides discussing triangulations is not among the purposes of this book). The triangulation introduces vertices, edges and triangles, and it is restricted by the requirement that in each vertex, there is a local map $\mathbf{x}_i : U_i \subset \mathbb{R}^2 \rightarrow \mathbb{R}^3$, 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 11.6, and Figure 11.15). 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 11.6).

The following example illustrates this procedure.

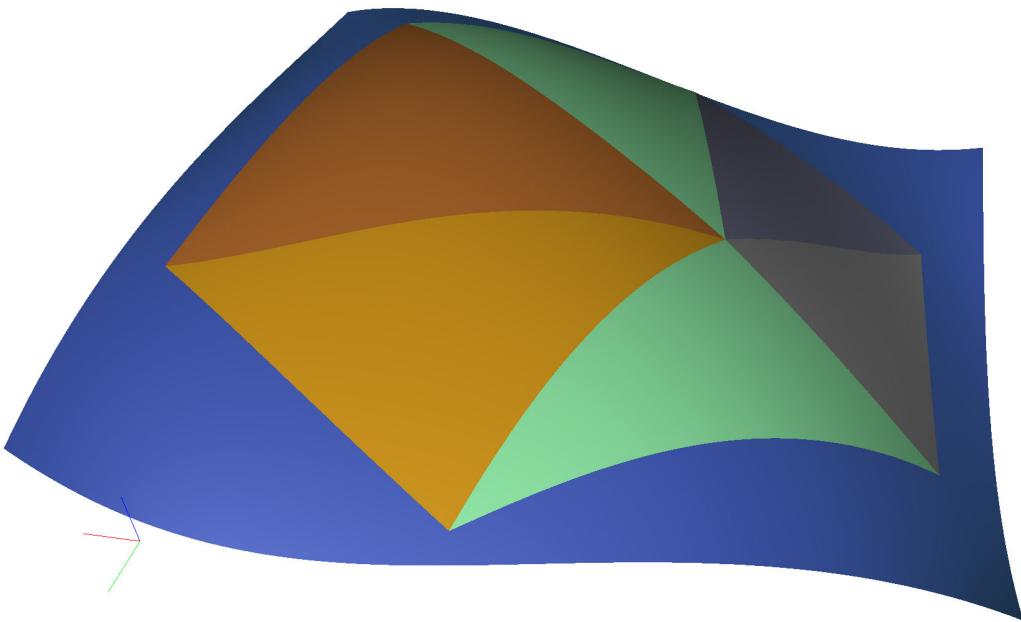


Figure 11.15: The blue surface is a Bézier tensor product surface. The different colored triangular surfaces are defined by a point (vertex) and 6 neighboring points (vertices) in the parameter plane of the Bézier tensor product surface. All triangular surfaces are, therefore, just sub-surfaces of the Bézier tensor product patch.

- 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

$$\mathbf{x}_i(u, v) = \begin{pmatrix} \cos u \cos v \\ \sin u \cos v \\ \sin v \end{pmatrix}, \quad (11.26)$$

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 10.3.1. 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 $\omega_u = \omega_v = \frac{1}{2}$ (ω_u and ω_v are described in section 10.3.1). 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 π (in the local map the distance from a pole to equator is $\frac{\pi}{2}$). This implies that $\delta_u = \delta_v = \pi$ (δ_u and δ_v is described in section 10.3.1).
- The construction of the local triangles (sub-triangles of the Bézier patch) in the

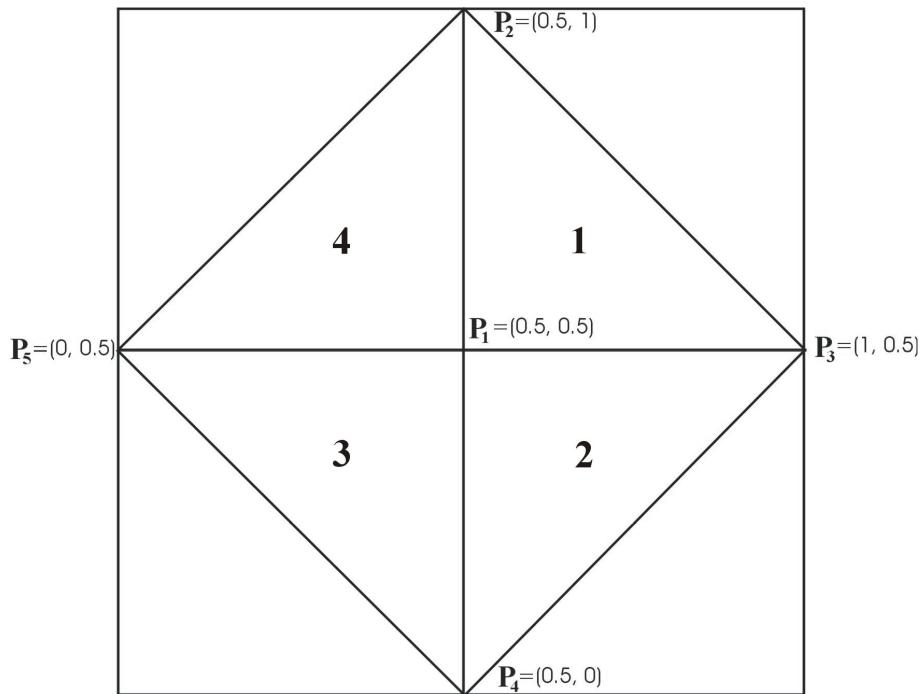


Figure 11.16: The parameter plane of a Bézier tensor product patch connected to one of the vertices in the triangulation of a sphere. The coordinates of the current vertex \mathbf{p}_1 , and the four neighboring vertices $\mathbf{p}_2, \mathbf{p}_3, \mathbf{p}_4, \mathbf{p}_5$ are all marked in the figure. This parameter plane looks in the same way for each of the vertices.

ERBS triangle is straightforward. The coordinates in the parameter plane of the Bézier patch can be seen in Figure 11.16. 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 11.16, 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 11.16 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 11.16), 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 11.16 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 11.17 we can see parts of some of the Bézier patches and the local sub-triangles from the sphere example. In Figure 11.18 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 11.8).

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 [34]). 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.

One important feasible feature in surface approximation by triangulation using ERBS is the object editation feature. This feature is used in figures 11.19, 11.20, 11.21 and 11.22. In Figure 11.19 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 11.20 the Bézier patch

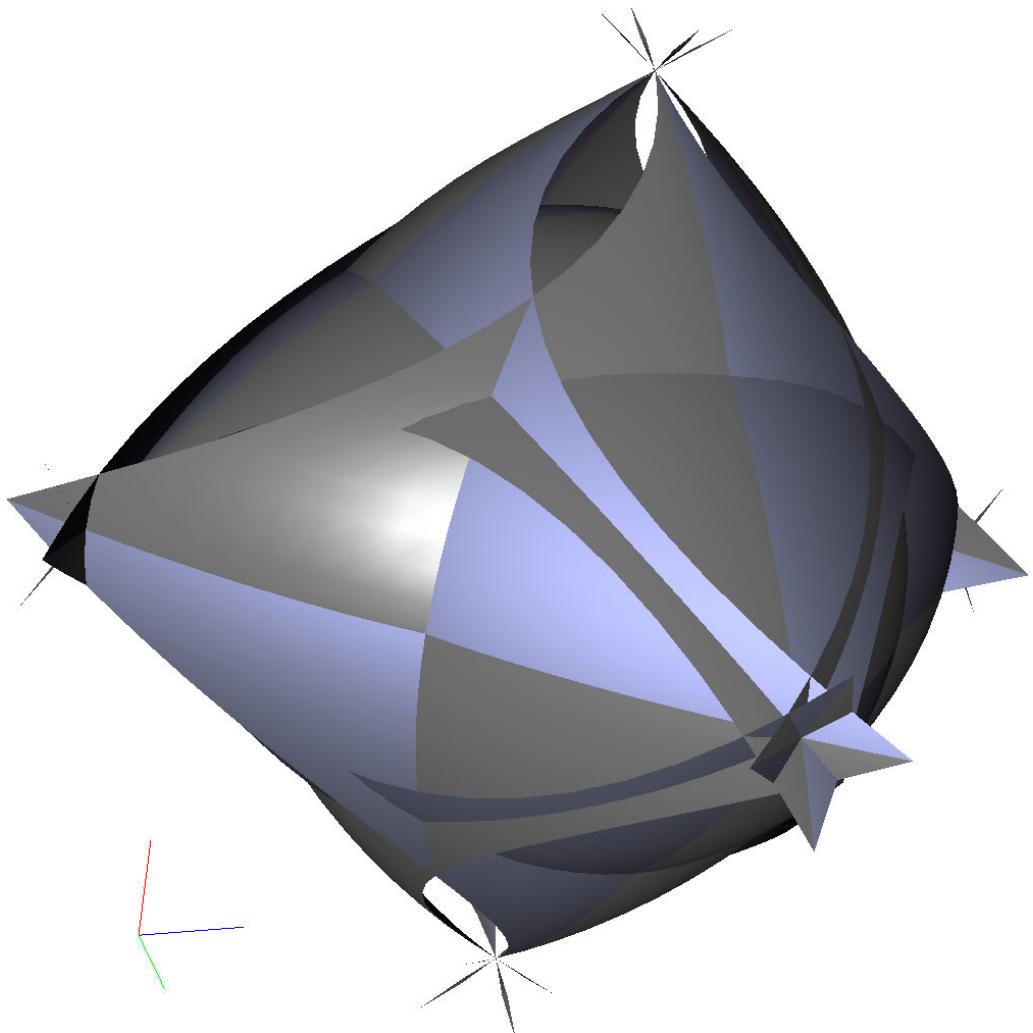


Figure 11.17: All 24 local triangles in the sphere example described in section 11.6 are displayed in the figure. We can clearly see some of them, and we can also see trimmed parts of the 6 Bézier patches connected to each of the 6 vertices. The center of each Bézier patch actually interpolates the original sphere in each of the vertices.

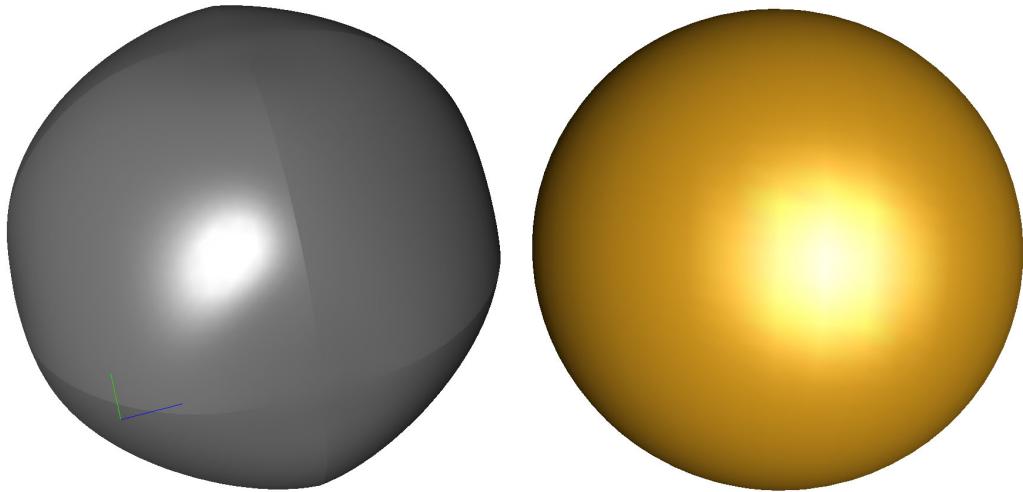


Figure 11.18: The brass sphere on the right hand side is the original sphere. The silver sphere on the left hand side is an ERBS approximation by triangulation. The approximation is done with only 8 triangles, and all the triangles are “equilateral”, symmetric according to all three vertices in each triangle.

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 11.21. 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 11.22 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 11.20, 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 11.20, 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 11.20, 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 11.20, 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, 11.23, there is one object plotted from four different angles. All local 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 11.22. 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^∞ , which follows from using local triangles (which are sub-triangles on a

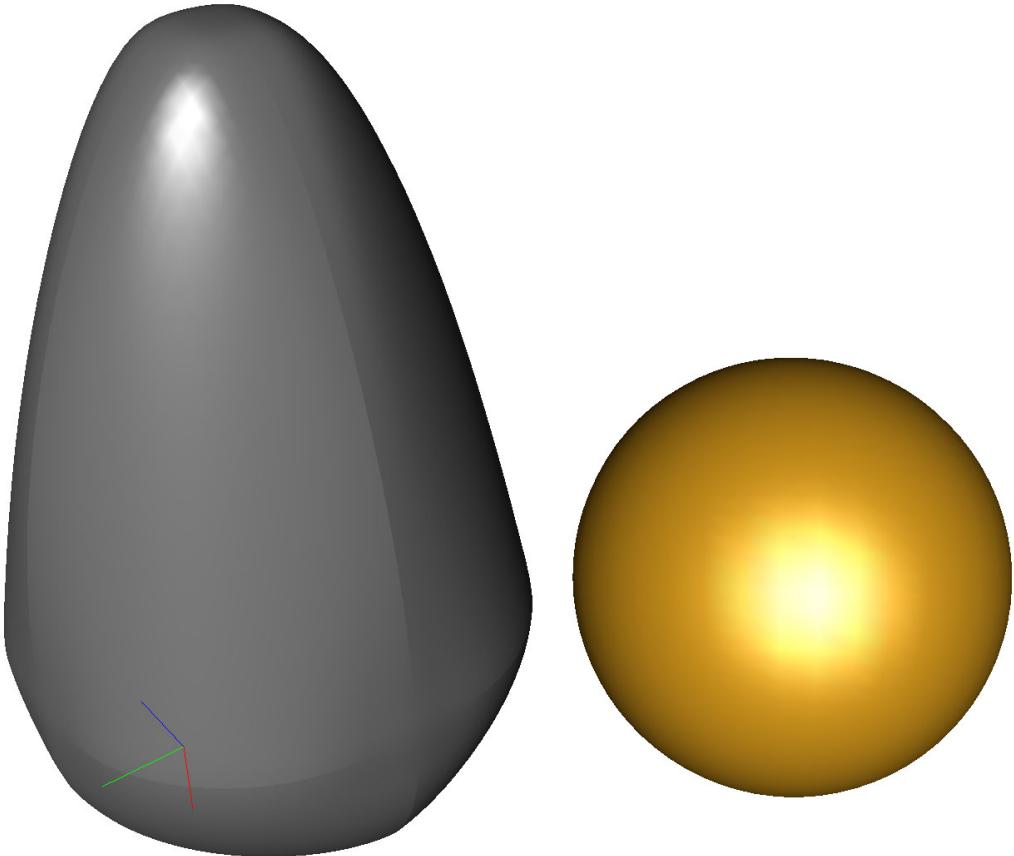


Figure 11.19: The brass sphere on the right hand side is the original sphere. The silver object on the left hand side is the ERBS approximation by triangulation, after the Bézier patch associated to the vertex at the north pole has been translated $1\frac{1}{2}$ to the north. The total length of the object (from south to north) is $3\frac{1}{2}$ (the diameter of the sphere is 2).

common local smooth surface at each of the vertices) and from property 4 in Theorem 11.1. 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 11.23, 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.

11.8 Epilogue

ERBS have the potential to make the most impact as a new approach in the case of surfaces on triangulations. It is clear from the properties defined in Theorem 11.1 that the continuity at the vertices of an ERBS triangulated surface is G^∞ (actually also C^∞). The continuity at the edges are clearly only continuous, although the parametrization seems

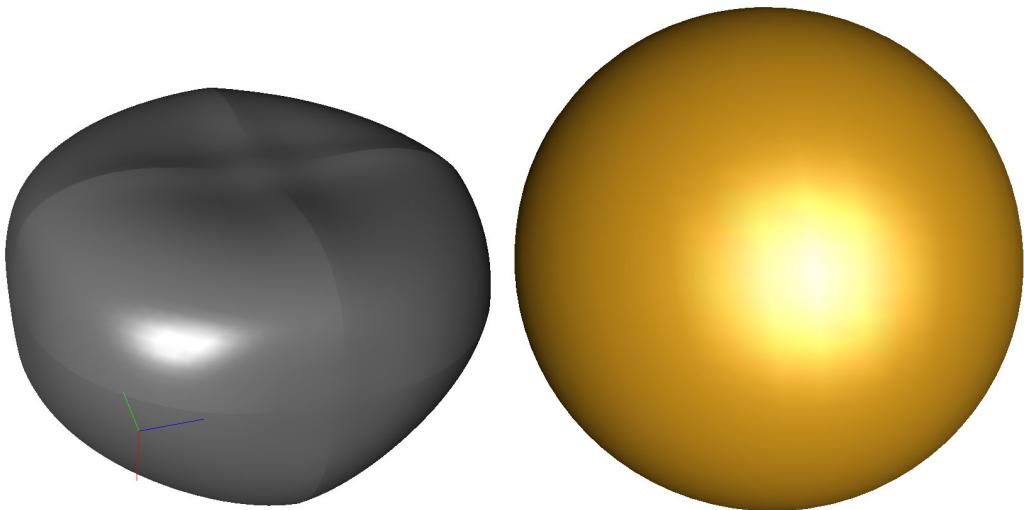


Figure 11.20: The brass sphere on the right hand side is the original unit sphere. The silver object on the left hand side is the ERBS approximation by triangulation, after the Bézier patch associated to the vertex at the north pole has been translated $\frac{1}{2}$ to the south. The total length of the object (from south to north) is $1\frac{1}{2}$ (the diameter of the sphere is 2).

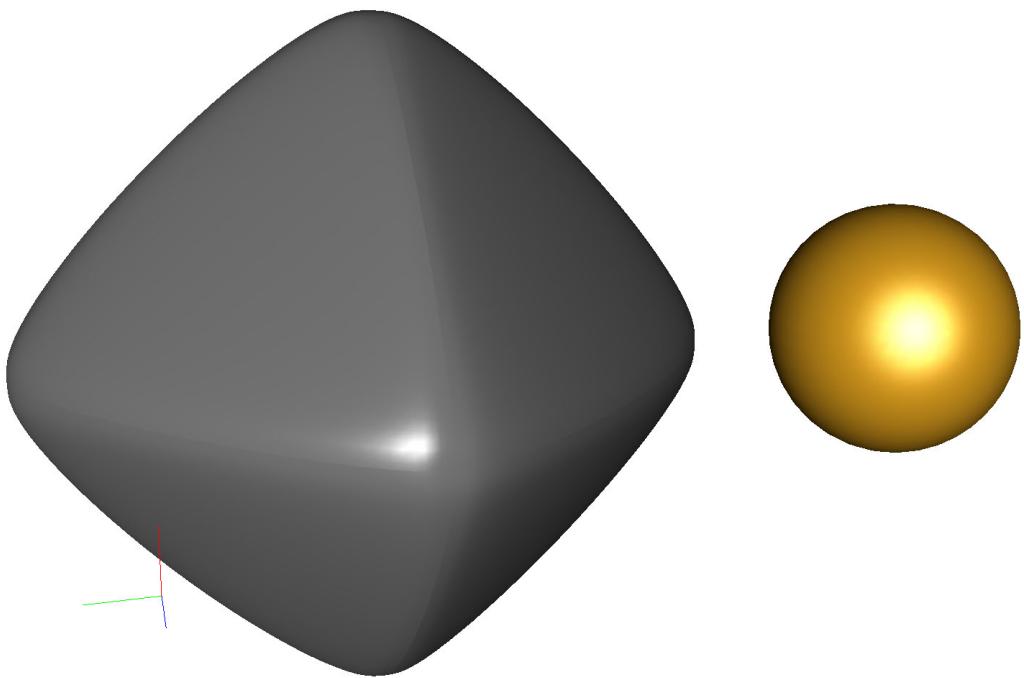


Figure 11.21: The brass sphere on the right hand side is the original unit sphere. The silver object on the left hand side is the ERBS approximation by triangulation, after the Bézier patch associated to each vertex has been translated $1\frac{1}{2}$ out from the center of the sphere. The total length of the object, from one vertex to the opposite vertex is 5 (the diameter of the sphere is 2).

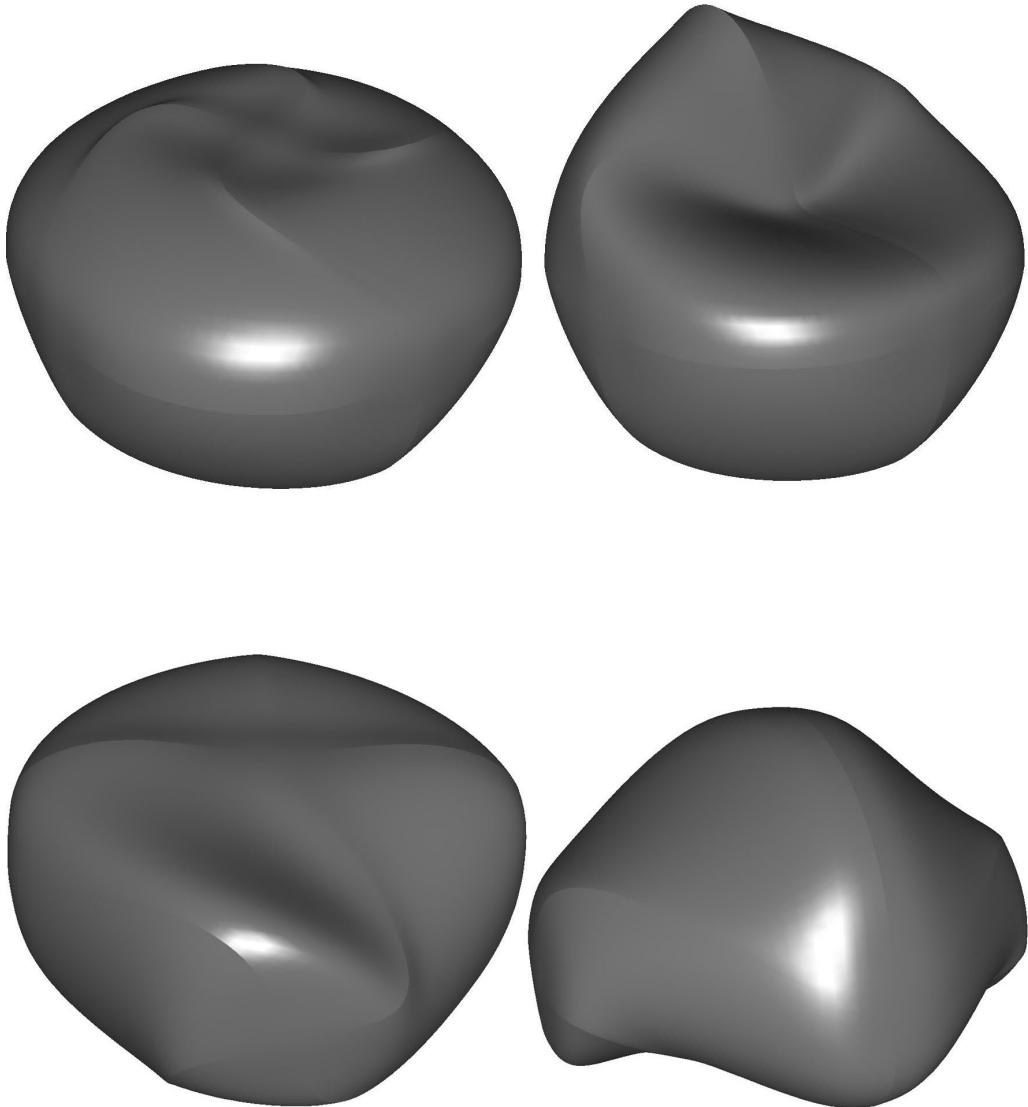


Figure 11.22: In this figure there are four objects obtained as a result of rotating local Bézier patches. All objects were initially the same object as the “silver object” in Figure 11.20. The object in the upper left hand corner has been further edited in a way that the Bézier patch at the top has been rotated an additional 70° 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° 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° 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° clockwise around a horizontal axis.

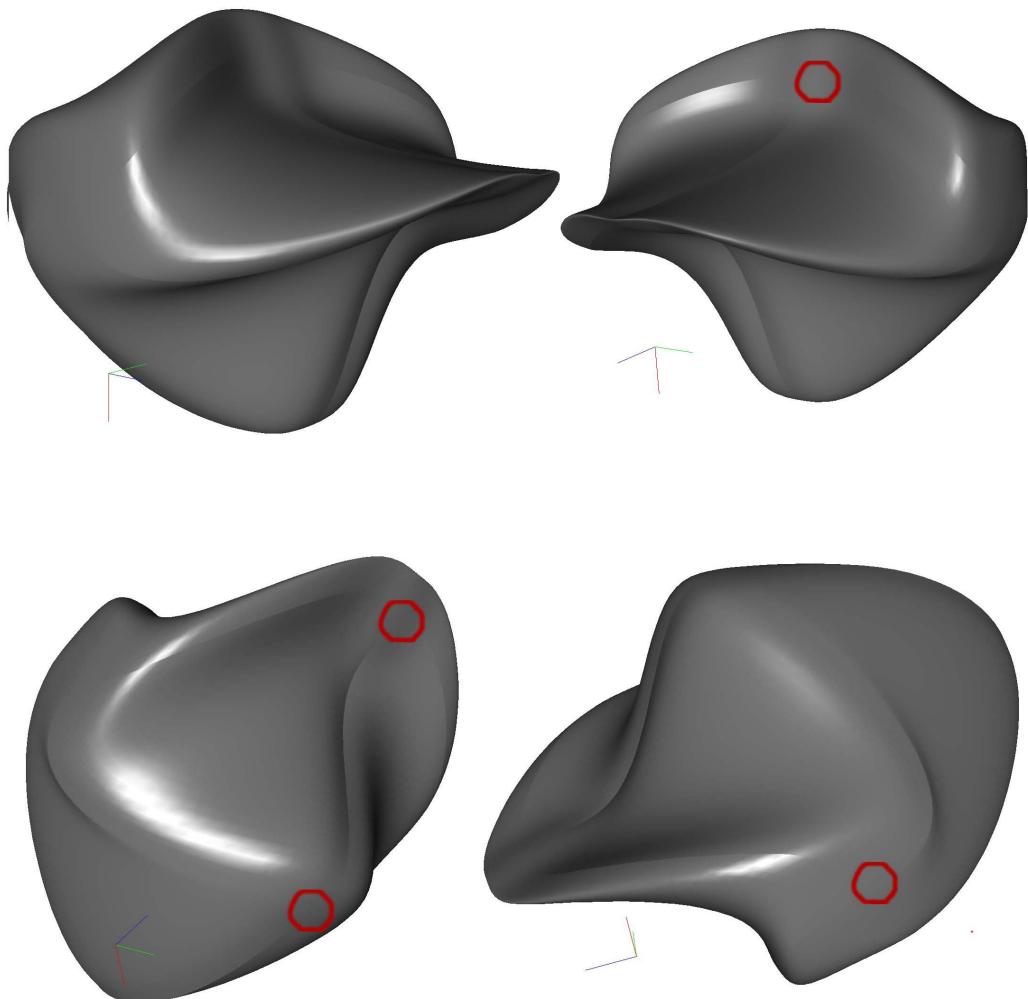


Figure 11.23: The four objects in this figure are actually four plots of the same object, only seen from a different angle. In this object all local Bézier patches, except for the one at the south pole, are rotated 70° clockwise around the same horizontal axis. The horizontal axis has the same direction as the one in the upper right hand corner in Figure 11.22. 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.

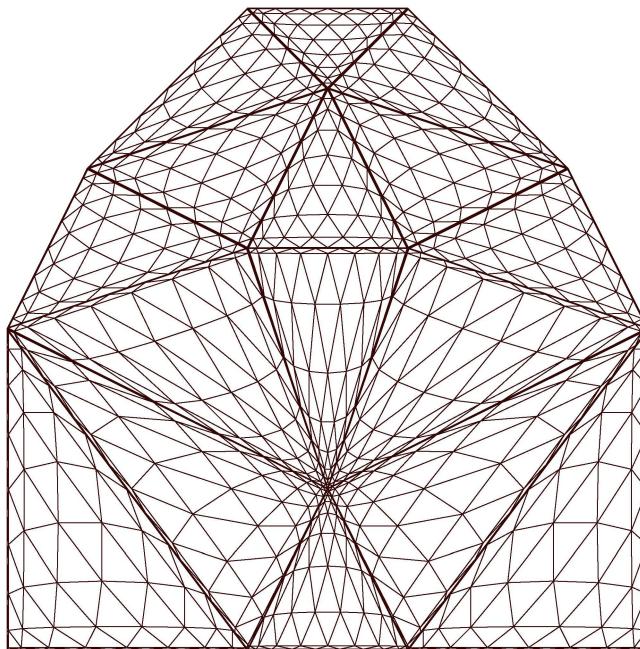


Figure 11.24: The figure shows a set of ERBS triangles where each is plotted with constant parameter lines: $\text{Line}(v, w)$, where u is a constant, $\text{line}(w, u)$, where v is a constant and $\text{line}(u, v)$, where w is a constant (recall that the parameters are homogeneous barycentric coordinates also for the curves).

to be smooth (see article [43]). 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^∞ 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. The case of surfaces on triangulations is similar. In Figure 11.24 an example is given 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^∞ 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 at the edges, although in the vertices the surface is also G -smooth. This can be judged from the fact that the constant parameter lines; $\text{line}(v, w)$, where u is a constant, $\text{line}(w, u)$, where v is a constant and $\text{line}(u, v)$, 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 line smoothly continues into the neighboring triangle. This behavior is due to Property **P5** in Theorem 11.1.

For practical use of surfaces on triangulations the continuity is, of course, of great importance. We shall, therefore, discuss some observations concerning continuity. First, recall



Figure 11.25: The figure shows a 3D plot of a function over the union of the domain of all local triangles connected to one vertex. The function is 1 at all the neighboring vertices and zero at this vertex (in the center).

some basic structures and properties:

- Recall that a surface on triangulation is C^∞ -smooth at all vertices, but initially only G^0 over the edges,
- and that an edge is determined by two neighboring vertices.
- Also remember that one local patch is connected to each vertex in a surface on triangulation. We also know that all local triangles connected to a respective vertex are a part of the local patch of the vertex and that these local triangles are disjunct except for the borders that include the actual vertex. This can clearly be observed in Figure 11.15.

The first observation when investigating smoothness over the edges is:

Observation 1. When investigating restrictions on the local surfaces we see the following:

The smoothness over an edge only depends on the local surfaces at the two neighboring vertices defining the edge, and moreover, the smoothness only depends on the two surfaces at the respective edge. This is a result of:

- i) If all local surfaces are parts of one common surface S and if the local triangles are organized in such a way that at each ERBS triangle the three local triangular surfaces are equal and that the local triangles cover the whole surface S , then the total ERBS surface is equal to the common surface S .
- ii) If we, when we have this coincidence between a surface S and an ERBS surface, change/move the surface connected to the third vertex in a triangle, we can see that it will not affect the edge defined by the two other vertices. This follows by Property P5 in Theorem 11.1.

This first observation tells us that we can achieve smoothness by introducing restrictions on the local surfaces. In Figure 11.25 there is a 3D plot of a function over the union of the domain of all local triangles connected to a vertex. This function is 1 at all the neighboring vertices and zero at this vertex, and it is constructed using ERBS both along the edges in

the direction from the center vertex and orthogonal to this direction. The function can be used as a blending function between the local patch of one vertex and the local patches of the neighboring vertices on their respective parts of the domain. Using this we will fulfill the condition in observation 1 to get a smooth ERBS surface on triangulation. It will, however, not always give the nicest shape because the position of the edges are preserved and the curvatures might therefore increase.

A second observation for investigating smoothness over the edges is:

Observation 2. *When investigating the parametrization of a single triangular patch we see the following:*

Theorem 11.1 shows that all derivatives are zero in all directions at the "top point" and at the "bottom edge" of a basis function (see figures 11.3 and 11.4). For the two other edges it is obvious that there is only one directional derivative at each point that is zero. This direction is, starting at one vertex, parallel to the opposite edge and then smoothly changed to the edge opposite the other vertex. At the middle point on the edge, the direction is the vector going through the opposite vertex. This can be seen from expressions 11.4, 11.5 and 11.10. This because the expressions can be simplified at the edges because at the edges are $\sum_{j=1}^k B(u_j) = 1$. The observation is also clearly visually "verified" in Figure 11.12. From the object at right hand side of the figure one can clearly see that the object is smooth at all vertices and also at a point between two vertices. The point between two vertices looks like it is on the straight line between two opposite vertices in two neighboring triangles.

For practical use in CAGD it is an advantage to develop a complete/controllable geometrically smooth version of the surface approximation by triangulation using ERBS. Several possible solutions can be seen, restrictions on local surfaces, reparameterization of the triangles or using fillets over the edges. All though, solving the smoothness, 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.

Appendix A

Evaluators, ERB-function, reliability, precision and efficiency

An Expo-Rational B-function “evaluator” is a computation of $B(t)$ (and thus the integrals in theorem 6.6 or 6.7 and the computation of the exponential function $\phi_k(t)$ in expression (6.27) or (6.32)). In addition, the computation of $B^{(j)}(t)$ in (6.36) for $j = 1, 2, \dots, d$ for some d is required, which includes computation of $f_j(t)$. (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 7.3), where $B_k(t)$ is defined in (7.31), and $D^j B_k(t)$, $j = 1, 2, \dots$ is defined in (7.36). 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 A.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 A.3), we will investigate an algorithm for numerical integration of the expression in (7.31) and (7.32), namely the integral

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

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 [43], section 6, a sequence of numerical methods for computing ERBS were considered. In this book 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.

A.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 [135] describes the formats of floating-point numbers (there is an ongoing revision called IEEE 754R [80]). According to the present standard, binary floating point numbers shall be of the form

$$(-1)^s 2^E (b_0.b_1b_2\dots b_{p-1}), \quad (\text{A.1})$$

where p is the precision and,

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

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 (A.1) is defining the so-called normal values. In addition, the IEEE standard specifies the following special values for numbers: ± 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 [65]) shows us how the 5 different types of values can be separated.

Type	values	implemented exponent	implemented precision
Special values	± 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

$$1.11\dots11 \times 2^{2^{8-1}-1} = (2 - 2^{-23}) 2^{127} \approx 3.4028237 e + 38.$$

For double precision it is

$$1.11\dots11 \times 2^{2^{11-1}-1} = (2 - 2^{-52}) 2^{1023} \approx 1.7976931348623159 e + 308.$$

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 is¹

$$1.00\dots00 \times 2^{2-2^{8-1}} = 2^{-126} \approx 1.1754944 e - 38.$$

For double precision it is

$$1.00\dots00 \times 2^{2-2^{11-1}} = 2^{-1022} \approx 2.22507385850720138 e - 308.$$

Values smaller than this are subnormal values with lower precision. Notice that the significant bit will be 0.11...11 (in binary representation) for the first subnormal value, and 0.00...01 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 ± 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 ≤ 1 then the fraction will not produce an overflow.

For a closer study of how the number system affects algorithms, we recommend [65]. An edited reprint can be found at "http://docs.sun.com/source/806-3568/ncg_goldberg.html".

Summing up:

¹The 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 ± 0 .
- Fractions might produce a signaled overflow.
 - Division by zero clearly produces a signaled overflow.
 - If the numerator in a fraction is ≤ 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 (7.33), namely,

$$-\beta \frac{|t - \lambda|^{(1+\gamma)\alpha}}{(t(1-t)^\gamma)^\alpha}, \quad \text{where } \alpha > 0, \beta > 0, \gamma > 0, 0 \leq \lambda \leq 1 \text{ and } t \in [0, 1], \quad (\text{A.2})$$

and later also the fraction $f_{j,k}(t)$, $j = 2, 3, \dots$, described in (7.36). When we look at the numerator of the fraction in (A.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

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

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

Remark 19. *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 (A.2), (7.33) and (7.48)) 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 8. (For notation, see section “Algorithmic Language”, page 18.)

The algorithm computes $\phi(\alpha, \beta, \gamma, \lambda; t)$ from expression (7.33), where α, β, γ and λ 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 (7.48)).

```

double φ ( double t )
    double d = (t(1-t)^\gamma)^\alpha;
    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;

```

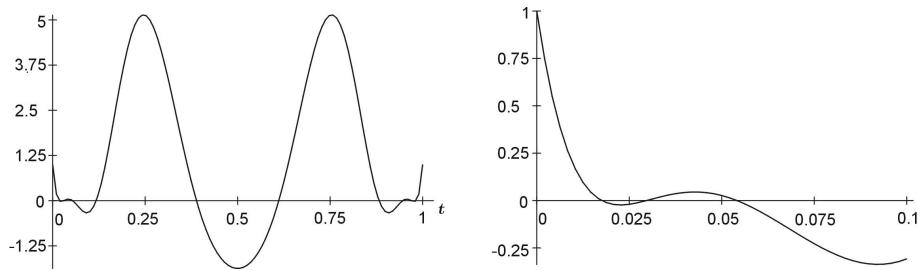


Figure A.1: The numerator for $f_{7,d}(t)$. On the left hand side is a graph from 0 to 1, on the right hand side a “zoomed version” from 0 to 0.1. One can clearly see that $|f_{7,d}(t)| \leq 1$, $t \in [0, 0.1]$.

```
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 (7.48), and plotted in Figure 7.4, 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

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

we get

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

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 $1 - \hat{t}$ is the limit on the right hand side.

In expression (7.50) and in Figure 7.11, 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 ± 1 . In Figure A.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 $t = \hat{t}$ we get

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

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

Remark 20. 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.

A.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 9. (For notation, see section “Algorithmic Language”, page 18.)

The algorithm computes $D^j B_k(t)$, $j = 0, 1, \dots, d$, for the default set of intrinsic parameters. It is, thus, an implementation of (7.48–7.50). 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 $t_k < t \leq t_{k+1}$, and $d \in \{0, 1, \dots, 6, 7\}$ (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 $DB(t), \dots, D^d B(t)$.

```

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 = φ(t); // See algorithm 8, part 2.
  if (p ≠ normal value) return R; // Test if underflow.
  R = S_d; // Each of the d + 1 elements equal to S_d.
  double  $\tilde{t} = (1 - 2t)^2$ ;
  switch (d)
    case 7: R7 * = ((((((( $\frac{315}{4}\tilde{t} + \frac{1575}{4}$ ) $\tilde{t} - \frac{8505}{8}$ ) $\tilde{t} + \frac{465}{4}$ ) $\tilde{t} + \frac{9645}{8}$ ) $\tilde{t} + \dots$  see (7.50));
    case 6: R6 * = ((((( $\frac{45}{2}\tilde{t} + \frac{135}{2}$ ) $\tilde{t} - \frac{765}{4}$ ) $\tilde{t} + 75$ ) $\tilde{t} + 66$ ) $\tilde{t} - \frac{85}{2}$ ) $\tilde{t} + \frac{15}{4}$ ;
    case 5: R5 * = ((( $\frac{15}{2}\tilde{t} + \frac{45}{4}$ ) $\tilde{t} - 33$ ) $\tilde{t} + \frac{29}{2}$ ) $\tilde{t} + \frac{3}{2}$ ) $\tilde{t} - \frac{3}{4}$ ;
    case 4: R4 * = (( $3\tilde{t} + \frac{3}{2}$ ) $\tilde{t} - 5$ ) $\tilde{t} + \frac{3}{2}$ ;
    case 3: R3 * =  $\frac{3}{2}\tilde{t}^2 - \frac{1}{2}$ ;
  R0 * =  $\int_{s=0}^t \phi(s) ds$ ; // See algorithm 13 in the next section.
  for ( int i=1; i ≤ d; i++ ) Ri * =  $\frac{p}{(t_{k+1}-t_k)^i (2t(1-t))^{2(i-1)}}$ ;
  for ( int i=2; i ≤ d; i+=2 ) Ri * = (1 - 2t);
  return R;

```

²It is preferable that the value of k for which $t_k < t \leq t_{k+1}$, see (7.31), is computed separately from the evaluation of the basis function, because this knowledge is also needed for the local functions, see (7.53). The question as to whether the local mapping, see (7.32), and scaling of the derivatives, see (7.36), should be done inside the evaluator (as in the first and third last line in Algorithm 9) 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 $f_{j,k}(t)$ (7.15). 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 (7.40), (when $(1+\gamma)\alpha < 1$, see (7.41)), and for $f_{3,k}(t)$ in (7.43) (when $(1+\gamma)\alpha < 2$, see (7.46)). This is done under the additional restriction $0 < \lambda < 1$. The implementation of $f_{2,k}(t)$ in (7.42) leads to implementation of $\zeta'_k(t)$ in (7.39). Because $x_{2,k}(t)$ in (7.40) consists of two parts where one might have an asymptote, we start by multiplying and factorizing, and we get

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

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

Algorithm 10. (*For notation, see section “Algorithmic Language”, page 18.*)

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 λ_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 (7.42) and, thus, uses (A.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 (A.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 (7.42), upper part.
    return 0.0;
  double h =  $\frac{t - \frac{1}{1+\gamma}}{t(1-t)}$  |t - λ|;           // First part of the second factor from (A.3).
  if ( t < λ )   h -= 1;                         // Last part of second factor (A.3).
  else            h += 1;
  h *=  $\frac{-S_k\alpha\beta(\gamma+1)}{(t(1-t)^\gamma)^\alpha}$ ;      // Inserting  $S_k$  and the first factor of (A.3).
  if ( (1+γ)α < 1 )                            // Asymptote at  $t = \lambda$  is present.
    double g =  $\frac{|t-\lambda|^{1-(1+\gamma)\alpha}}{h}$ ;        // The inverse of (A.3).
    if ( g ≠ normal value )
      return 0.0;
    else
      return  $\frac{1}{g}$ ;
  else if ( (1+γ)α > 1 )                        // Ordinary solution.
    return h |t - λ| $^{(1+\gamma)\alpha-1}$ ;
  else                                              // Discontinuity at  $t = \lambda$ .
    return h;
```

A comment to the foregoing algorithm; it is, as said above, treating the case of possible asymptote $(1+\gamma)\alpha < 1$, see (7.41), 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.

³The guarantee is that we have to introduce practical restrictions on the intrinsic parameters because of the binary number system. See Table A.1 and the following comments to that table.

The implementation of $f_{3,k}(t)$ (7.47) leads to implementation of $x_{3,k}(t)\zeta_k(t)$ from (7.47). Because $x_{3,k}(t)$ (7.43) 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

$$x_{3,k}(t)\zeta_k(t) = \frac{-\alpha\beta}{(t(1-t)^\gamma)^\alpha} \left(|t-\lambda|^2 \frac{t^2(\gamma+1)-2t+1}{(t(t-1))^2} - (\gamma+1) + \alpha \left(1 - \frac{\beta|t-\lambda|^{(1+\gamma)\alpha}}{(t(1-t)^\gamma)^\alpha} \right) \left(\frac{t(\gamma-\lambda-\lambda\gamma)+\lambda}{t(1-t)} \right)^2 \right) |t-\lambda|^{(1+\gamma)\alpha-2}. \quad (\text{A.4})$$

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

Algorithm 11. (*For notation, see section “Algorithmic Language”, page 18.*)

The algorithm computes $f_{3,k}(t)$ for the scalable subset \mathfrak{B} of the intrinsic parameters, where α_k , β_k , γ_k and λ_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 (7.47) and, thus, uses (A.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 (A.4) has a normal value on the open segment $(0, 1)$.

```
double f3 ( double t )
if ( t == λ )                                // See (7.47), 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(t-1))^2 - (γ+1); // First part of the second factor of (A.4).
h += α * (1 - β * |t-λ|^(1+γ)α / (t(t-1)γ)α) * (t(γ-λ-λγ)+λ / t(t-1))^2; // Second part of the second factor.
h *= -S_k * α * β / (t(t-1)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 (A.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 (7.41), 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 $\lim_{t \rightarrow \lambda} f_3(t) = \mp\infty$ 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 (7.45), 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 10, and also connected to Algorithm 11: 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:

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

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 A.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 $t \rightarrow 0+$ and $t \rightarrow 1-$, and for second and third derivatives, can be seen in Table A.1.

From Table A.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
	3.146×10^{-13}	$< \beta <$	$+\infty$
	0.201	$< \gamma <$	534.37
	6.72×10^{-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
	3.146×10^{-13}	$< \beta <$	1.436×10^{34}
	0.201	$< \gamma <$	496
	1.337×10^{-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

⁴When $(1+\gamma)\alpha = 1$, all even derivatives will only be discontinuous, and all odd derivatives will be continuous (see proof of Theorem 7.1, item k-II on page 195).

par.	der.	bound	$t \rightarrow 0$	$t \rightarrow 1$
α	2	lower	$9.294556548602447 \times 10^{-3}$	0.1856551589777102
		upper	1075	1075
	3	lower	0.01864443248188489	0.1856551589777102
		upper	466.889890736804	465.9770256841861
β	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	$5.747194528456591 \times 10^{34}$	$1.436798632114148 \times 10^{34}$
γ	2	lower	0	0.2013156366896773
		upper	1002.994782602334	534.3735944906107
	3	lower	0	0.2013156366896773
		upper	496.0047816968047	534.3735944906107
λ	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 A.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 $t \rightarrow 0+$ and when $t \rightarrow 1-$. 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 α and γ are the most “influential” parameters, which is natural, because they are exponents. It is also remarkable that the range of β is so large that we can, by only varying β , 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 12. (*For notation, see section “Algorithmic Language”, page 18.*)

The algorithm computes $D^j B_k(t)$, $j = 0, 1, \dots, d$ 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 $0 \leq t \leq 1$. 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 $t_k < t \leq t_{k+1}$, 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 $DB(t), \dots, D^d B(t)$.

```

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 = φ(t);           // See algorithm 8, part 2.
  if (p ≠ normal value) return R; // Test if underflow.
  switch (d)
    case 3: R3 = f3(t);
    case 2: R2 = f2(t);
    case 1: R1 = Sk;
    R0 = Sk  $\int_{s=0}^t \phi(s)ds$ ;           // See algorithm 13 in the next section.
    for ( int i=1; i ≤ d; i++ ) Ri * =  $\frac{p}{(t_{k+1}-t_k)^i}$ ;
  return R;

```

A.3 ERBS-evaluator based on Romberg-integration

The principal part of the ERBS-evaluator $B_k(t)$ defined in Definition 7.2 is the integration of $\psi_k(t)$ defined in (7.8) or $\phi(t)$ defined in (7.33). In this section, only one of the possible reliable and controllable numerical integration will be investigated. The choice fell on Romberg integrations (see [118] and [24]), which is based on repeated Richardson extrapolation to eliminate error terms (see [82]). The background for the algorithm is the Euler-MacLaurin integration formula (see [142]). 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

$$I(f) - T_n(f) = \frac{1}{n^2} \sum_{j=0}^{\infty} A_j^{(0)} \frac{1}{n^{2j}} = \frac{1}{n^2} A_0^{(0)} + \frac{1}{n^4} A_1^{(0)} + \frac{1}{n^6} A_2^{(0)} + \dots, \quad (\text{A.5})$$

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

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

Richardson extrapolation can now be used in conjunction with (A.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 (A.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

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

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

$$B(h) = \frac{2^{2j} T_{h/2}(f) - T_h(f)}{2^{2j} - 1}, \quad (\text{A.6})$$

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 (A.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 (7.33),

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 8. (Remark 19 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 A.2 and in Figure A.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

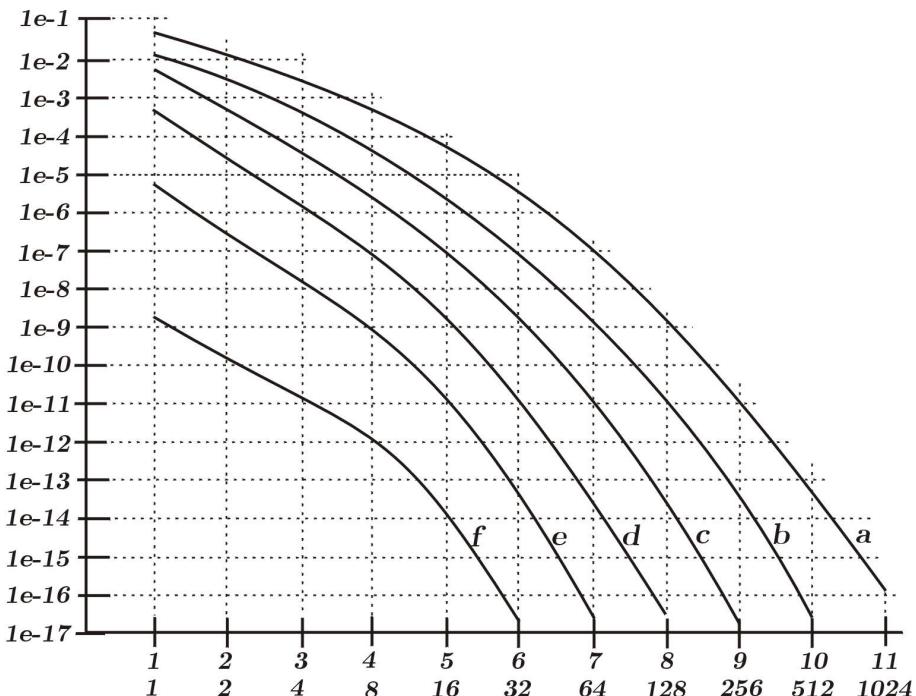


Figure A.2: The figure shows the relationship between the precision and the number of iterations in the Romberg-integration algorithm for ERBS-evaluation when using the default intrinsic set of parameters. The upper scale on the horizontal axis is the number of iterations, and the lower scale is the new number of evaluations (i.e. at step 11 there are 1024 new computations of $\phi(t)$ and a total of $1+1+2+\dots+1024=2048$ 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 A.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	$7.2e-2$	$1.4e-2$	$8.3e-3$	$7.7e-4$	$7.3e-6$
2	2	$2.0e-2$	$2.6e-3$	$7.9e-4$	$1.1e-5$	$1.4e-6$
3	4	$1.3e-3$	$9.3e-4$	$7.0e-5$	$1.6e-6$	$8.4e-11$
4	8	$9.5e-4$	$8.5e-5$	$2.9e-6$	$6.9e-8$	$1.6e-9$
5	16	$8.9e-5$	$3.3e-6$	$8.0e-8$	$1.7e-9$	$1.9e-11$
6	32	$3.4e-6$	$8.3e-8$	$1.8e-9$	$2.0e-11$	$4.8e-14$
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 A.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

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

Each computation is named by a letter, “a)” means that the integration interval is $[0, \frac{1}{2}]$. 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 $[0, \frac{1}{64}]$. In Table A.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 A.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 A.2 is the values from Table A.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, $[\frac{1}{2}, 1]$.

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 13. (*For notation, see section "Algorithmic Language", page 18.)*
The algorithm computes the integral $\tilde{B} = \int_{s=0}^t \phi(s)ds$, $t \in (0, 1]$, where \tilde{B} is inside the given tolerance ϵ . The value of the input "tolerance" variable ϵ is recommended to be in $[1e-2, 1e-15]$ (according to Table A.2).

```
double integrate ( double t, double ε )
    double M[16][16];
    double sum =  $\frac{\phi(t)}{2}$ ;
    M0,0 = t * sum;
    for ( int i=1; i < 16; i++ )
        double s = 0;
        int k =  $2^i$ ;                                // C++ implementation: k = 1 << i
        t /= 2;
        for ( int j=1; j < k; j+=2 ) s +=  $\phi(j * t)$ ;
        M0,i = t * (sum += s);
        for ( int j=1; j ≤ i; j++ )
            double c =  $4^j$ ;                      // C++ implementation: c = 1 << (j << 1)
            Mj,i-j =  $\frac{c * M_{j-1,i-j+1} - M_{j-1,i-j}}{c - 1}$ ;      // Richardson extrapolation scheme
            if ( |Mi,0 - Mi-1,0| < ε ) return Mi,0;
    return M15,0;
```

A.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 (7.34)), 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 (7.34)).
$\mathbf{b} = \{\int_0^{t_i} \phi(s) ds\}_{i=0}^m$	// 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 A.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:

<i>initiate</i> ($\alpha, \beta, \gamma, \lambda, m$)	// Changing states and thus computing $S, \mathbf{b}, \mathbf{a}$ and Δt .
<i>B</i> (t, k, d)	// For a given $t \in [0, 1]$, k - knot interval, d - nr. of derivatives, // analogous to Algorithm 12, computing $D^j B(t)$, $j = 0, \dots, d$.

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

<i>interpolate</i> ($i, \phi_0, \phi_1, \phi'_0, \phi'_1$)	// Computing $\mathbf{a}_{i,0}, \mathbf{a}_{i,1}, \mathbf{a}_{i,2}, \mathbf{a}_{i,3}$ and $\mathbf{a}_{i,4}$ (using (A.12)).
$\phi(t)$	// Defined in Algorithm 8.
$f_2(t)$	// Defined in Algorithm 10.
<i>integrate</i> ($t_0, t_1, \phi_0, \phi_1, \epsilon$)	// Modified version of Algorithm 13. 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 \mathbf{b} storing the incrementing integral samples, and the matrix \mathbf{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 ≤ 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 3rd degree polynomial equation,

$$f(x) = a_0 + a_1x + a_2x^2 + a_3x^3, \quad (\text{A.7})$$

and its first derivative,

$$f'(x) = a_1 + 2a_2x + 3a_3x^2. \quad (\text{A.8})$$

Integrating (A.7) from 0 to a given value $\hat{t} \in (0, 1)$ gives the following result

$$\int_{x=0}^{\hat{t}} f(x) = \hat{t} \left(a_0 + \hat{t} \left(\frac{a_1}{2} + \hat{t} \left(\frac{a_2}{3} + \hat{t} \left(\frac{a_3}{4} \right) \right) \right) \right), \quad (\text{A.9})$$

and evaluating (A.7) at \hat{t} yields

$$f(\hat{t}) = a_0 + \hat{t} \left(a_1 + \hat{t} \left(a_2 + \hat{t} \left(a_3 \right) \right) \right). \quad (\text{A.10})$$

Then evaluating (A.8) gives

$$f'(\hat{t}) = a_1 + \hat{t} \left(2a_2 + \hat{t} \left(3a_3 \right) \right). \quad (\text{A.11})$$

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

$$\begin{aligned} a_0 &= f(0), \\ a_1 &= f'(0), \\ a_2 &= 3(f(1) - f(0)) - f'(1) - 2f'(0), \\ a_3 &= -2(f(1) - f(0)) + f'(1) + f'(0), \\ a_4 &= \frac{f(0)+f(1)}{2} + \frac{f'(0)-f'(1)}{12}. \end{aligned} \quad (\text{A.12})$$

The question now is: how do we use these four expressions (A.9), (A.10), (A.11) and (A.12) in the evaluator? Recall that scaling of the domain influences the derivatives and the antiderivatives (integrals). Suppose the real interval is Δt (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 7.7 (also discussed in [46]), we have to do the following three adjustments:

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

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

Algorithm 14. (*For notation, see section “Algorithmic Language”, page 18.*)

The algorithm computes the internal state variables Δt , $S = S(\alpha, \beta, \gamma, \lambda)$, $\mathbf{b} = \{B(t_i)\}_{i=0}^m$ 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  $\alpha$ , double  $\beta$ , double  $\gamma$ , double  $\lambda$ , int  $m$  )
    double  $\phi_0, \phi_1$ ;                                // Value at the start and end of each interval
    double  $f_0, f_1$ ;                                // Derivative at start and end of each interval
    set( $\alpha, \beta, \gamma, \lambda, m$ );                  // Store intrinsic parameters and  $m$  in object
    Allocate memory for  $\mathbf{b}, \mathbf{a}$ ;                // Set size =  $m+1$  for  $\mathbf{b}$ , and  $m \times 4$  for  $\mathbf{a}$ 
     $\Delta t = \frac{1}{m}$ ;                            // Set interval size
     $S = 1$ ;                                    // Temporary, to use  $\phi'(t) = f_2(t)$ 
     $\mathbf{b}_0 = \phi_0 = f_0 = 0.0$ ;                // Defined to be zero (basic properties)
    for ( int  $i=1; i < m; i++$  )                  // For each sampling interval
        double  $t = i * \Delta t$ ;                  // The sampling vector marked  $t_i$  in definition
         $\phi_1 = \phi(t)$ ;                        // Algorithm 8
         $f_1 = \phi_1 f_2(t)$ ;                  // Algorithm 10
         $\mathbf{b}_i = \text{integrate}(t - \Delta t, t, \phi_0, \phi_1, 1 \times 10^{-16})$ ; // Algorithm 13 (using max tolerance)
        interpolate( $i - 1, \phi_0, \phi_1, \Delta t f_0, \Delta t f_1$ ); // Updating Hermite coefficients  $\{\mathbf{a}_{i-1,k}\}_{k=0}^3$ 
         $\phi_0 = \phi_1$ ;                          // Preparing for the next step
         $f_0 = f_1$ ;
         $\mathbf{b}_m = \text{integrate}(\Delta t(m - 1), 1, \phi_0, 0, 1 \times 10^{-16})$ ; // Algorithm 13 (using max tolerance)
        interpolate( $m - 1, \phi_0, 0, \Delta t f_0, 0$ ); // Updating Hermite coefficients  $\{\mathbf{a}_{m-1,k}\}_{k=0}^3$ 
    for ( int  $i=1; i \leq m; i \lll= 1$  )            // These three following lines are introduced
        for ( int  $j=m; j \geq i; j - = 1$  )          // because there in line 13/17 is not used += =
             $b_j += b_{j-i}$ ;                      // See below for explanation
         $S = \frac{1}{b_m}$ ;
    
```

In line 13 and 17 $\mathbf{b} += \text{integrate}(...)$, 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 Δt 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 (A.12)).

Algorithm 15. (*For notation, see section “Algorithmic Language”, page 18.*)

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 (A.12).

```
void interpolate ( int i, double f0, double f1, double f0', double f1' )
    ai,0 = f0;
    ai,1 = f0';
    ai,2 = 3(f1 - f0) - f1' - 2f0';
    ai,3 = -2(f1 - f0) + f1' + f0';
    ai,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 12. 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 16. (*For notation, see section “Algorithmic Language”, page 18.*)

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: $t \in [t_1, t_n]$, $k|t_k < t \leq t_{k+1}$, 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(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: R2 * =  $\frac{\mathbf{a}_{j,1}+dt(2\mathbf{a}_{j,2}+dt 3\mathbf{a}_{j,3})}{\Delta t}$ ; // Using (A.11), scaled by  $\frac{1}{\Delta t}$ 
        case 1: R1 * =  $\mathbf{a}_{j,0}+dt(\mathbf{a}_{j,1}+dt(\mathbf{a}_{j,2}+dt \mathbf{a}_{j,3}))$ ; // Using (A.10), no scaling
        if (dt > 0.5) // Integrating: dt - 1 (A.12)
            R0 * =  $b_{j+1} - \Delta t (\mathbf{a}_{j,4} - dt (\mathbf{a}_{j,0} + dt (\frac{\mathbf{a}_{j,1}}{2} + dt (\frac{\mathbf{a}_{j,2}}{3} + dt \frac{\mathbf{a}_{j,3}}{4}))))$ ; // Scaled by  $\Delta t$ 
        else // Integrating: 1 - dt (A.12)
            R0 * =  $b_j + \Delta t dt (\mathbf{a}_{j,0} + dt (\frac{\mathbf{a}_{j,1}}{2} + dt (\frac{\mathbf{a}_{j,2}}{3} + dt \frac{\mathbf{a}_{j,3}}{4})))$ ; // Scaled by  $\Delta t$ 
        for ( int i=1; i ≤ d; i++ ) Ri / =  $(t_{k+1}-t_k)^i$ ; // Due to (7.36)
        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.

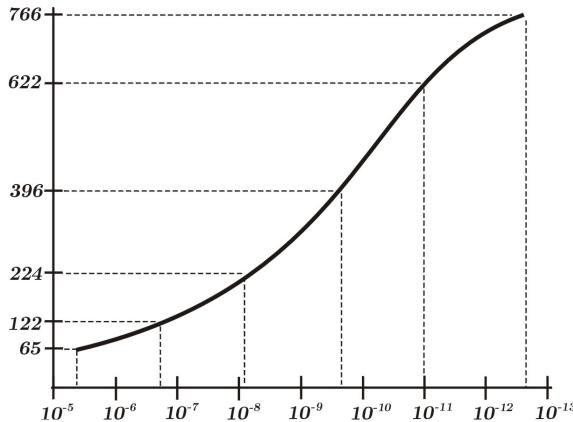


Figure A.3: The relationship between the precision (tolerance) and the speed of the evaluation compared between the use of the $B()$ function in the “ERBS-evaluator” and the use of the $B()$ function in Algorithm 9 from section A.2. For a sample rate of 1024, which gives a precision of $3.3e-13$, the “ERBS-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 A.3 there is a graph of the relationship between the use of Algorithm 9 (with two derivatives) and Algorithm 16, the “ERBS-evaluator”. Algorithm 9 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 9 is very dependent on the precision, especially the integration part using Romberg integration. This can clearly be seen in Figure A.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 “ERBS-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 9 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 16.

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: $\frac{2}{3} \times \max \text{ error of the function} \times \frac{1}{m}$ (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).

Looking at the precision of the “ERBS-evaluator”, Table A.3 shows the connection be-

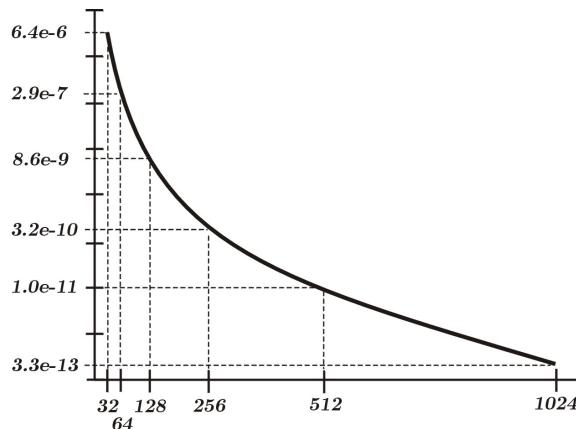


Figure A.4: The relationship between the number of samples and the precision (max norm) of $B(t)$ (function value) using the “ERBS-evaluator”. For a sample rate of 32, the precision is $6.4 \text{ e-}6$. For a sample rate of 1024, the precision is $3.3 \text{ e-}13$.

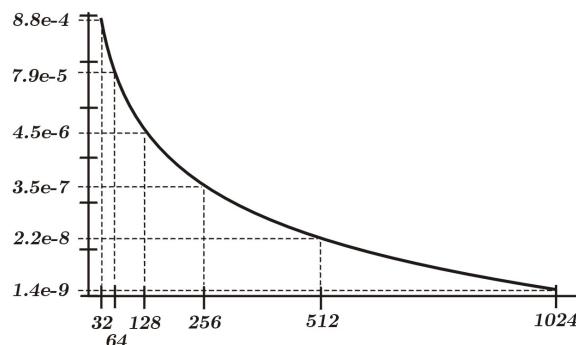


Figure A.5: The relationship between the number of samples and the precision (max norm) of $DB(t)$ (first derivatives) using the “ERBS-evaluator”. For a sample rate of 32, the precision is $8.8 \text{ e-}4$. For a sample rate of 1024, the precision is $1.4 \text{ e-}9$.

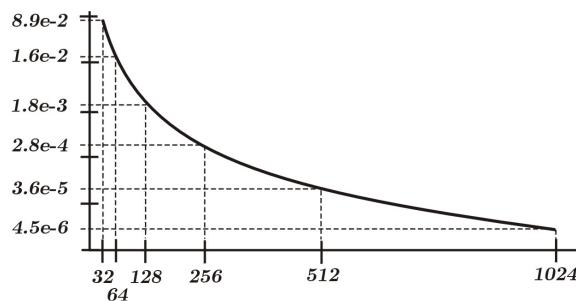


Figure A.6: The relationship between the number of samples and the precision (max norm) of $D^2B(t)$ (second derivatives) using the “ERBS-evaluator”. For a sample rate of 32, the precision is $8.9 \text{ e-}2$. For a sample rate of 1024, the precision is $4.5 \text{ e-}6$.

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 A.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 $L^\infty[0, 1]$.

tween 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 A.4, A.5 and A.6, show graphs of the maximum error of the “ERBS-evaluator” produced according to the number of samples. In Figure A.4 the error for the function value $B(t)$, are plotted. It ranges from 6.4×10^{-6} for 32 samples, to 3.3×10^{-13} for 1024 samples. This can be regarded as quite a good result. In Figure A.5 the error for the first derivative $DB(t)$ is plotted. It ranges from 8.8×10^{-4} for 32 samples, to 1.4×10^{-9} for 1024 samples. This can in many cases still be regarded as an acceptable result. In Figure A.6 the errors for the second derivative $D^2B(t)$ are plotted. It ranges from 8.9×10^{-2} for 32 samples, to 4.5×10^{-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 9 or Algorithm 12. 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 $D^j B(t)$, $j = 2, 3, \dots, 7$ from algorithm 9.

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 $D^j B(t)$ for $j = 2, 3$ from algorithm 12.

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 16.
Inherited class 1 “ERBS-default” combine Algorithm 16 and 9.
Inherited class 2 “ERBS-scalable” combine Algorithm 16 and 12.

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

Bibliography

References

- [1] M. Abramowitz and I. A. Stegun. *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*. Dover, New York, ninth Dover printing, tenth GPO printing edition, 1964.
- [2] H. Akima. A new method of interpolation and smooth curve fitting based on local procedures. *Journal of ACM*, 17:589–602, 1970.
- [3] R. L. Bagula and P. Bourke. Trianguloid trefoil
<http://local.wasp.uwa.edu.au/~pbourke/geometry/tranguloid/>, 2002. [Online; accessed 10-Desember-2008].
- [4] R. L. Bagula and P. Bourke. Bent horns surface
<http://local.wasp.uwa.edu.au/~pbourke/geometry/benthorns/>, 2003. [Online; accessed 10-Desember-2008].
- [5] R.E. Barnhill, R.F. Riesenfeld, United States. Office of Naval Research, and University of Utah. *Computer aided geometric design: proceedings of a conference held at the University of Utah, Salt Lake City, Utah, March 18-21, 1974*. Academic Press Rapid manuscript Reproduction. Academic Press, 1974.
- [6] 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.
- [7] D. Bechmann. Multidimensional Free-form Deformation Tools. In *Eurographics'98, State of the Art Report*, 1999.
- [8] S. Bernstein. Démonstration du théorème de Weierstrass fondée sur le calcul des probabilités. *Comm. Soc. Math.*, 13(1–2), 1912.
- [9] A. Beutelspacher and U. Rosenbaum. *Projective geometry: from foundations to applications*. Cambridge University Press, Cambridge, 1998.
- [10] P. Bezier. Définition numérique des courbes et surfaces I. *Automatisme*, XI:625–632, 1966.

- [11] P. Bezier. Définition numérique des courbes et surfaces II. *Automatisme*, XII:17–21, 1967.
- [12] G. D. Birkhoff. General Mean Value and Remainder Theorems with Applications to Mechanical Differentiation and Quadrature. *Transactions of the American Mathematical Society*, 7(1):107–136, 1906.
- [13] Wolfgang Boehm. Inserting New Knots into B-spline Curves. *Journal of Computer Aided Design*, 12(4):199–201, 1980.
- [14] P. Bourke. Mathematical shell .
<http://local.wasp.uwa.edu.au/~pbourke/geometry/shell/>, 1998. [Online; accessed 10-Desember-2008].
- [15] H. Briggs. *Arithmetica Logarithmica*. Guglielmus Iones, London, 1624.
- [16] V. Brun. Gauss' fordelingslov. *Norsk Matematisk Tidsskrift*, 14:81–92, 1932.
- [17] S. Bu-qing and L. Ding-yuan. *Computational Geometry - Curve and Surface Modeling*. Academic Press Inc. [Harcourt Brace Jovanovich Publishers], London, first edition, 1989.
- [18] P. L. Butzer, M. Schmidt, and E. L. Stark. Observations on the History of Central B-Splines. *Archive for History of Exact Sciences*, 39:137–156, 1988/89.
- [19] E. Catmull and J. Clark. Recursively generated B-spline surfaces on arbitrary topological surfaces. *Computer-Aided Design*, 10(6):350–355, 1978.
- [20] E. Catmull and R. Rom. A class of local interpolating splines. *Computer Aided Geometric Design*, pages 317–326, 1974.
- [21] A.-L. Cauchy. *Cours d'Analyse de l'École Royale Polytechnique*. Imprimerie Royale, Paris, 1821.
- [22] E. Cohen, T. Lyche, and R. Riesenfeld. Discrete B-splines and subdivision techniques in computer aided geometric design and computer graphics. *Comp. Graphics and Image Process*, 14(2):87–111, 1980.
- [23] E. Cohen, T. Lyche, and L. L. Schumaker. Algorithms for degree-raising of splines. *ACM Transactions on Graphics (TOG)*, 4(3):171–181, 1985.
- [24] S. D. Conte and C. de Boor. *Elementary Numerical Analyses*. McGraw-Hill, Singapore, 1983.
- [25] S. A. Coons. Surfaces for computer aided design. Technical report, MIT, Cambridge, MA, USA, 1964. Available as AD 663 504 from the National Technical Information service, Springfield, VA 22161.
- [26] 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.

- [27] M. G. Cox. Curve fitting with piecewise polynomials. *J. Inst. Math. Appl.*, 8:36–52, 1972.
- [28] H.S.M. Coxeter. *Projective geometry*. University of Toronto Press, Toronto, Ont., second edition, 1974.
- [29] H.S.M. Coxeter. *Introduction to Geometry*. Wiley Classics Library. Wiley, 1989.
- [30] B. H. Curry. Review of the paper [120, 121]. *Math, Tables and other Aids to Comp.*, 2:167–169 and 211–213, 1947.
- [31] B. H. Curry and I. J. Schoenberg. On Pòlya frequency functions IV: The spline functions and their limits. *Bull. Amer. Math. Soc.*, 53:1114, 1947. Abstract 380t.
- [32] B. H. Curry and I. J. Schoenberg. On Pòlya frequency functions IV: The fundamental spline functions and their limits. *J. d'Analyse Math.*, 17:71–107, 1966.
- [33] P. J. Davis. *Interpolation and Approximation*. Dover Publication Inc. (unabridged republication from a first edition from 1963), New York, N.Y., 1975.
- [34] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry*. Algorithms and Applications. Springer-Verlag, New York, 1997.
- [35] C. de Boor. On calculation with B-splines. *Journal of Approximation Theory*, 6:50–62, 1972.
- [36] C. de Boor. *A Practical Guide to Splines*, volume 27 of *Applied Mathematical Sciens*. Springer-Verlag, New York, 1978.
- [37] P. de Casteljau. Outils méthodes calcul. Technical report, A. Citroën, Paris, 1959.
- [38] P. de Casteljau. Courbes et surfaces à pôles. Technical report, A. Citroën, Paris, 1963.
- [39] P. de Casteljau. Shape Mathematics and CAD. *Kogan Page*, 1986.
- [40] L. T. Dechevsky, B. Bang, and A. Lakså. Generalized Expo-Rational B-splines. *International Journal of Pure and Applied Mathematics*, 57(1):833–872, 2009.
- [41] L. T. Dechevsky, A. Lakså, and B. Bang. Expo-Rational B-splines. Preprint 2/2004, Narvik University College, Narvik, Norway, 2004.
- [42] L. T. Dechevsky, A. Lakså, and B. Bang. NUERBS form of Expo-Rational B-splines. Preprint 1/2004, Narvik University College, Narvik, Norway, 2004.
- [43] L. T. Dechevsky, A. Lakså, and B. Bang. Expo-Rational B-splines. *International Journal of Pure and Applied Mathematics*, 27(3):319–369, 2006.
- [44] L. T. Dechevsky, A. Lakså, and B. Bang. NUERBS form of Expo-Rational B-splines. *International Journal of Pure and Applied Mathematics*, 32(1):11–32, 2006.

- [45] R. Descartes. *The Geometry of Rene Descartes*. Dover classics of science and mathematics. Dover Publications, 1954.
- [46] M. P. do Carmo. *Differential geometry of curves and surfaces*. Prentic Hall, Inc., New Jersey, USA, 1976.
- [47] M. P. do Carmo. *Riemannian Geometry*. Birkhäuser, Inc., Boston, MA, USA, 1992.
- [48] D. Doo. A subdivision algorithm for smoothing down irregularly shaped polyhedrons. In *Proceedings on Interactive Techniques in Computer Aided Design*, pages 157–165, 1978.
- [49] D. Doo and M. Sabin. Behavior of recursive division surfaces near extraordinary points. *Computer-Aided Design*, 10(6):356–360, 1978.
- [50] Euclide and T. L. Heath. *The Thirteen Books of the Elements*. Number v. 3 in Dover classics of science and mathematics. Dover Publications, 1956.
- [51] L. Euler. De Eximio usu Methodi Interpolationum in Serierum Doctrina. *Opuscula Analytica*, 1:157–210, 1783.
- [52] G. Farin. Triangular Bernstein-Bezier patches. *Computer Aided Geometric Design*, 3(2):83–128, 1986.
- [53] G. Farin. *Curves and Surfaces for CAGD*. Morgan Kaufmann, San Francisco, California, fifth edition, 2002.
- [54] G. Farin. Class A Bezier curves. *Computer Aided Geometric Design*, 23(7):573–581, 2006.
- [55] 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.
- [56] T.H. Fay. The Butterfly Curve. *The American Mathematical Monthly*, 96(5):442–443, 1989.
- [57] 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.
- [58] M. S. Floater, G. Koz, and M. Reimers. Mean value coordinates in 3D. *Computer Aided Geometric Design*, 22(7):623–631, 2005.
- [59] D. C. Fraser. Newton and interpolation. In W. J. Greenstreet, editor, *Isaac Newton 1642–1727: A Memorial Volume*, pages 45–69, London, 1927. G. Bell and Sons. First published in Journal of the Institute of Actuaries, vol. 51, 1918.
- [60] Lars A. Frøyland, Arne Lakså, and Jan Pajchel. Modelling geological structures using splines. In *Mathematical methods in computer aided geometric design, II (Biri, 1991)*, pages 287–296. Academic Press, Boston, MA, 1992.

- [61] I. P. Gancheva and N. D. Delistoyanova. Euler Beta-function B-spline: definition, basic properties, and practical use in Computer Aided Geometric Design. Master theses, Narvik University College, Narvik, Norway, 2007.
- [62] M. Gasca and T. Sauer. On the History of Multivariate Polynomial Interpolation. *Journal of Computational and Applied Mathematics*, 122(1 & 2):23–35, 2000.
- [63] C. F. Gauss. *Theoria Interpolationis Methodo Nova Tractata*, pages 265–327. Göttingen, 1866.
- [64] B. V. Gnedenko. *The theory of probability*. Translated from the fourth Russian edition by B. D. Seckler. Chelsea Publishing Co., New York, 1967.
- [65] D. Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys*, 23(1):5–48, 1991.
- [66] R. Goldmann. *Pyramid Algorithms: A Dynamic Programming Approach to Curves and Surfaces for Geometric Modeling*. Morgan Kaufmann Publishers, San Francisco, California, 2003.
- [67] H. H. Goldstine. *A History of Numerical Analysis from the 16th through the 19th Century*. Springer-Verlag, Berlin, 1977.
- [68] W. J. Gordon. Blending-function method of bivariate and multivariate interpolation and approximation. *SIAM Journal on Numerical Analysis*, 8(1):158–177, 1969.
- [69] A. Gray. *Modern Differential Geometry of Curves and Surfaces*. CRC Press, Inc., Boca Raton, Florida, first edition, 1993.
- [70] J. Gregory. Letter to J. Collins. (23 November 1670). In H. W. Turnbull, editor, *James Gregory Tercentenary Memorial Volume*, London, 1939. G. Bells & Sons.
- [71] T. N. E. Greville. The General Theory of Osculatory Interpolation. *Transactions of the Actuarial Society of America*, 45:202–265, 1944.
- [72] Cindy M. Grimm and John F. Hughes. Modeling surfaces of arbitrary topology using manifolds. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, SIGGRAPH '95, pages 359–368, New York, NY, USA, 1995. ACM.
- [73] H. Guggenheimer. Computing frames along a trajectory. *Comput. Aided Geom. Des.*, 6:77–78, February 1989.
- [74] S. Guillet and J. C. Leon. Parametrically deformed free form surfaces as a part of variational model. *Computer Aided Design*, 30(1), 1998.
- [75] R. C. Gupta. Second Order Interpolation in Indian Mathematics up to the Fifteenth Century. *Indian Journal of History of Science*, 4(1 & 2):86–98, 1969.
- [76] E. Hartmann. Parametric G^n blending of curves and surfaces. *The Visual Computer*, 17:1–13, 2001.

- [77] R. Henderson. A Practical Interpolation Formula. With a Theoretical Introduction. *Transactions of the Actuarial Society of America*, 9(35):211–224, 1906.
- [78] Ch. Hermite. Sur la Formule d’Interpolation de Lagrange. *Journal für die Reine und Angewandte Mathematik*, 84(1):70–79, 1878.
- [79] K. Höllig. *Finite Element Methods with B-Splines*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2003.
- [80] IEEE. IEEE 754 Revision Work. <http://grouper.ieee.org/groups/754/revision.html>, 2006.
- [81] E. Isaacson and H. B. Keller. *Analysis of Numerical Methods*. Dover Publication Inc., New York, N.Y., 1994.
- [82] H. Jeffreys and B. S. Jeffreys. L. F. Richardson’s methode. *Methods of Mathematical Physics*, 3rd ed.:288, 1988.
- [83] W. A. Jenkins. Graduation Based on a Modification of Osculatory Interpolation. *Transactions of the Actuarial Society of America*, 28:198–215, 1927.
- [84] S. A. Joffe. Interpolation-Formulae and Central-Difference Notation. *Transactions of the Actuarial Society of America*, 18:72–98, 1917.
- [85] J. Karup. Über eine Neue Mechanische Ausgleichungsmethode. In G. King, editor, *Transactions of the Second International Actuarial Congress*, pages 31–77, London, 1899. Charles and Edwin Layton.
- [86] E. S. Kennedy. The Chinese-Uighur Calendar as Described in the Islamic Sources. *Science and Technology in East Asia*, pages 191–199, 1977.
- [87] F Klok. Two moving coordinate frames for sweeping along a 3D trajectory. *Comput. Aided Geom. Des.*, 3:217–229, November 1986.
- [88] J. L. Lagrange. Leçons élémentaires sur les Mathématiques Données a l’école Normale. In J. A. Serret, editor, *Euvres de Lagrange*, volume 7, pages 183–287, Paris, 1877. Gauthier-Villars. Lecture notes first published in 1795.
- [89] A. Lakså. *Basic properties of Expo-Rational B-splines and practical use in Computer Aided Geometric Design*. Number 606 in unipubavhandlinger. Unipub, Oslo, 2007.
- [90] 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.
- [91] A. Lakså, B. Bang, and L. T. Dechevsky. Geometric modelling with Beta-function B-splines I. *International Journal of Pure and Applied Mathematics*, 65(3):339–360, 2010.

- [92] A. Lakså, B. Bang, and L. T. Dechevsky. Geometric modelling with Beta-function B-splines II. *International Journal of Pure and Applied Mathematics*, 65(3):362–380, 2010.
- [93] A. Lakså, B. Bang, and A. R. Kristoffersen. GMlib, a C++ library for geometric modeling. Technical report, Narvik University College, Narvik, Norway, 2006.
- [94] A. Lakså and M. Floater. Reference manual SI Spline Library. Technical report, Senter for industriforskning, Oslo, Norway, 1990.
- [95] Arne Lakså, Børre Bang, and Lubomir T. Dechevsky. Exploring expo-rational B-splines for curves and surfaces. In *Mathematical methods for curves and surfaces: Tromsø 2004*, Mod. Methods Math., pages 253–262. Nashboro Press, Brentwood, TN, 2005.
- [96] 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.
- [97] T. Lyche. *Discrete polynomial spline approximation methods*, volume 501/1976 of *Lecture Notes in Mathematics*, pages 144–176. Springer, Berlin/Heidelberg., 1976.
- [98] T. Lyche and K. Mørken. Knot removal for parametric B-spline curve and surfaces. *Computer Aided Geometric Design*, 4(3):217–230, 1987.
- [99] T. Lyche and K. Strøm. Knot insertion for natural splines. *Annals of Numerical Mathematics*, 3:221–246, 1996.
- [100] O. L. Mangasarian and L. L. Schumaker. Discrete Splines via Mathematical Programming. *SIAM Journal on Control*, 9:174–183, 1971.
- [101] J. C. Martzloff. *A history of Chinese mathematics*. Springer Verlag, Berlin, 1997.
- [102] C. Mäurer and B. Jüttler. Rational approximation of rotation minimizing frames using Pythagorean-hodograph cubics. *Journal for Geometry and Graphics*, 3(2):141–159, 1999.
- [103] L. Maurer. Über die Mittelwerte der Funktionen einer reellen Variablen. *Math. Ann.*, 47:263–280, 1896.
- [104] D. S. Meek and D. J. Walton. Blending two parametric curves. *Computer-Aided Design*, 41:423–431, 2009.
- [105] E. Mehlum. Nonlinear splines. *Computer Aided Geometric Design*, pages 173–207, 1974.
- [106] 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.
- [107] 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.

- [108] E. Meijering. A chronology of interpolation: From ancient astronomy to modern signal and image processing. In *Proceedings of the IEEE*, pages 319–342, 2002.
- [109] A. F. Möbius. Über die Zusammensetzung gerader Linien und eine daraus entstehende neue Begründungsweise des barycentrischen Calculus. *Journal für reine und ang. Math*, 28:1–9, 1844.
- [110] J. Cotrina Navau and N. Pla Garcia. Modelling surfaces from planar irregular meshes. *Computer Aided Geometric Design*, 17(1):1 – 15, 2000.
- [111] O. Neugebauer. *A History of Ancient Mathematical Astronomy*. Springer-Verlag, Berlin, 1975.
- [112] I. Newton. Methodus Differentialis. In W. Jones, editor, *Analysis per Quantitates, Series, Fluxiones, ac Differentias: cum Enumeratione Linearum Tertii Ordinis*, pages 93–101, London, 1711.
- [113] I. Newton. Letter to J. Smith (8 may 1675). In H. W. Turnbull, editor, *The Correspondence of Isaac Newton, vol. I(1661-1675)*, pages 342–345, Cambridge, 1959. Cambridge University Press.
- [114] I. Newton. Letter to Oldenburg (24 October 1676). In H. W. Turnbull, editor, *The Correspondence of Isaac Newton, vol. II(1676-1687)*, pages 110–161, Cambridge, 1960. Cambridge University Press.
- [115] I. Newton. *Philosophiae Naturalis Principia Mathematica*. London, 1987. English translation by F. Cajori, *Sir Isaac Newton's Mathematical Principles of Natural Philosophy and his System of the World*, University of California Press, Berkeley, CA, 1960.
- [116] W. Press, S. Teukolsky, W. Vetterling, and B. Flannery. *Numerical Recipes in C*. Cambridge University Press, Cambridge, UK, 2nd edition, 1992.
- [117] L. Ramshaw. Blossoms are polar forms. *Computer Aided Geometric Design*, 6(4):323–359, 1989.
- [118] W. Romberg. Vereinfachte numerische Integration. *Det Kongelige Norske Vid. Selsk. Forl.*, 28:30–36, 1955.
- [119] W. Rudin. *Principles of Mathematical Analysis. I*. McGraw-Hill Inc., Singapore, third edition, 1976. Functional analysis.
- [120] I. J. Schoenberg. Contributions to the Problem of Approximation of Equidistant Data by Analytic Functions. Part A – On the Problem of Smoothing or Graduation. A First Class of Analytic Approximation Formulae. *Quarterly of Applied Mathematics*, IV(1):45–99, 1946.
- [121] I. J. Schoenberg. Contributions to the Problem of Approximation of Equidistant Data by Analytic Functions. Part B – On the Problem of Osculatory Interpolation. A Second Class of Analytic Approximation Formulae. *Quarterly of Applied Mathematics*, IV(2):112–141, 1946.

- [122] I. J. Schoenberg. On spline functions. *Inequalities*, pages 255–291, 1967.
- [123] I. J. Schoenberg. Cardinal Spline Interpolation. *CBMS-NSF Series in Applied Mathematics*, 12, SIAM, 1973.
- [124] L. L. Schumaker. *Spline Functions: Basic Theory*. A Wiley-interscience publication. John Wiley & Sons Inc., New York, 1981.
- [125] Thomas W. Sederberg, David L. Cardon, G. Thomas Finnigan, Nicholas S. North, Jianmin Zheng, and Tom Lyche. T-spline simplification and local refinement. *ACM Trans. Graph.*, 23(3):276–283, August 2004.
- [126] Thomas W. Sederberg, Jianmin Zheng, Almaz Bakenov, and Ahmad Nasri. T-splines and t-nurccs. *ACM Trans. Graph.*, 22(3):477–484, July 2003.
- [127] 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.
- [128] C. H. Séquin, K. Lee, and J. A. Yen. Fair, G^2 - and C^2 -continuous circle splines for the interpolation of sparse data points. *Computer-Aided Design*, 37(2):201–211, 2005.
- [129] C. H. Séquin and J. A. Yen. Fair and robust curve interpolation on the sphere. Sketches and Application. In *SIGGRAPH '01: Proceedings of the 17th annual Conference on Computer Graphics*, page 182, 2001.
- [130] W. F. Sheppard. Central-difference formula. *Proceedings of the London Mathematical Society*, 31:449–488, 1899.
- [131] K.L. Shi, J.H. Yong, J.G. Sun, and J.C. Paul. g^n blending multiple surfaces in polar coordinates. *Computer-Aided Design*, 42:479–494, 2010.
- [132] David Eugene Smith. *History of Modern Mathematics*. Mathematical Monographs. John Wiley & sons, New York, fourth edition, 1906.
- [133] A. Sommerfeld. Eine besonders anschauliche ableitung des gaussischen fehlergesetzes. *Festschrift Ludwig Boltzman gewidmet zum 60. Geburtstage, 20. Februar 1904*, pages 848–859, 1904.
- [134] M. Spivak. *A Comprehensive Introduction to Differential Geometry I*. Publish or Perish, Inc., Houston, Texas, USA, second edition, 1979.
- [135] 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.
- [136] D. J. Struik. *A Concise History of Mathematics*. Dover Publication Inc., New York, N.Y., 1987.
- [137] D. J. Struik. *Lectures on Classical Differential Geometry*. Dover Publication Inc., New York, N.Y., 1988.

- [138] M. Szivasi-Nagy and T.P. Vendel. Generating curves and swept surfaces by blended circles. *Computer Aided Geometric Design*, 17(2):197–206, 2000.
- [139] T. N. Thiele. *Interpolationsrechnung*. B. G. Teubner, Leipzig, Germany, 1909. In German.
- [140] G. J. Toomer. Hipparchus. In C. C. Gillispie and F. L. Holmes, editors, *Dictionary of Scientific Biography*, volume XV, pages 207–224, New York, 1978. Charles Scribner’s Sons.
- [141] G. van Brummelen. Lunar and Planetary Interpolation Tables in Ptolemy’s Almagest. *Journal for the History of Astronomy*, 25(4):297–311, 1994.
- [142] I. Vardi. The Euler-Maclaurin Formula. *Computational Recreation in Mathematics*, pages 159–163, 1991.
- [143] W. Wang, B. Jüttler, D. Zheng, and Y. Liu. Computation of rotation minimizing frames. *ACM Trans. Graph.*, 27:2:1–2:18, March 2008.
- [144] 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.
- [145] E. Waring. Problems Concerning Interpolations. *Philosophical Transactions of the Royal Society of London*, 69:59–67, 1779.
- [146] J. Warren. Barycentric coordinates for convex polytopes. *Advances in Computational Mathematics*, 6(1):97–108, 1996.
- [147] E. W. Weisstein. Rose. <http://mathworld.wolfram.com/Rose.html>, 2006.
- [148] H. Wenz. Interpolation of curve data by blended generalized circles. *Computer Aided Geometric Design*, 13(8):673–680, 1996.
- [149] E. T. Whittaker. On the Functions which are Represented by the Expansions of Interpolation-Theory. *Proceedings of the Royal Society of Edinburgh*, 35:181–194, 1915.
- [150] E. T. Whittaker and G. Robinson. *A Short Course in Interpolation*. Blackie and Son Limited, London, 1923.
- [151] Wikipedia. Basic linear algebra subprograms — wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Basic_Linear_Algebra_Subprograms&oldid=235926608, 2008. [Online; accessed 17-September-2008].
- [152] A. Wiltsche. Blending curves. *Journal for Geometry and Graphics*, 9(1):67–75, 2005.
- [153] K. C. Wu, T. Fernando, and H. Tawfik. Freesculptor: A computer-aided freeform design environment. *gmag*, 00:188, 2003.
- [154] L. Yan, D. Shiran, J. N. Crossley, and A. W. C. Lun. *Chinese Mathematics: A Concise History*. Clarendon Press, Oxford, 1988.

- [155] Lexing Ying and Denis Zorin. A simple manifold-based construction of surfaces of arbitrary smoothness. *ACM Trans. Graph.*, 23(3):271–275, August 2004.

List of 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

- L^2 norm, 245, 250
- 3D-curve, 253
 - acceptable result, 390
 - acceptable tolerance, 384
 - additional restrictions, 198
 - affine combination, 136
 - affine global/local mapping, 208
 - Affine space, 52
 - affine transformation, 17, 251, 253
 - airplane, 322
 - Akima's interpolation, 107
 - Algebraic geometry, 12
 - algorithm, 87, 92, 222, 223, 226, 302, 304, 372, 374–376, 379, 383, 386, 387
 - antisymmetric, 203
 - apple, 234
 - approximating curve, 230
 - approximative curve length parametrization, 246
 - arc-spline, 235
 - arithmetic mean, 390
 - Astronomy, 95
 - asymptote, 201, 375
 - asymptotic behavior, 200, 201
 - asymptotic case, 375
 - B-function, 141
 - B-spline, 17, 110, 114
 - B-spline curves, 116
 - B-spline factor matrix, 119
 - Bézier curve, 91, 92, 224, 226, 228
 - Bézier patche, 305
 - Bézier triangle, 330, 332
 - band matrix, 225
 - bandwidth, 225
 - barycentric coordinate, 330
 - basic properties, 188, 202
 - basis function, 206, 207, 217, 229
 - beam, 253
 - bell-shaped, 203
 - bending, 320
 - Bent Horns, 308
 - Bernstein polynomial, 84, 89, 91, 118, 221, 224, 305
 - Bernstein polynomials, 91
 - Bernstein/Hermite matrix, 91, 224, 228, 230, 305
 - bicubically blending, 276
 - bilinear blending, 275
 - binary floating point, 369
 - binary search, 377
 - blending, 136, 217
 - blending function, 141
 - Blossoming, 136
 - blue cubes, 60
 - Boolean sum surface, 274
 - boundary conditions, 108
 - buckle, 231
 - C++ class, 383
 - cardinal spline, 107
 - Cardioid curve, 234
 - Catmull-Rom spline, 107
 - celestial bodies, 96
 - central B-spline, 111
 - charts and atlas, 50
 - Chinese ephemerides, 96
 - circle, 234
 - circle spline, 138
 - circular arc, 218, 234, 235
 - close to equal, 250
 - closed curve, 218, 220, 230
 - closed parameter, 297
 - closest point, 247
 - CMA, 16

- coefficient vector, 17
color, 234
column vector, 241
commutative, 121
compact, 329
Compact space, 51
compare the result, 243
complete/controllable geometrically smooth, 368
computer game, 320
connected, 329
constraint, 196, 203
continuity, 188, 209
continuity at the edges, 362
continuity at the vertex, 362
continuous composition, 354
contraction, 298
contradiction, 199
control polygon, 132, 228, 305, 322
converging, 380
convex set, 331
Coons Patch, 270
Coons patch, 275
coordinate system, 240
corner, 296, 361
corner cutting, 132
corner point, 337
Cox-de'Boor recursion, 91, 115, 224
critical part, 372
cubic Bessel spline, 107
cubic spline interpolation, 107, 129
curvature, 229, 230, 238
curve, 217
curve length, 246
curve length parametrization, 239, 250
curves on surfaces, 262
curves on triangular surfaces, 290
cusp, 234
- de Casteljau, 224
de Casteljau algorithm, 91, 117
de Casteljau's algorithm, 88
default set, 369, 372, 380
default subset, 202
default values, 202
- definition, 184, 185, 197, 202, 207, 208, 219, 247, 297, 331–334, 337
denormalized value, 370
derivations of product, 188
derivative, 91, 186, 198, 200, 226, 231, 374
deviation, 245
deviations of speed, 247
diameter, 250
diffeomorphism, 356
difference operator, 99
differentiable manifold, 219, 297
differential, 262
Differentiation, 261
directional derivative, 332
discontinuous, 209
divided difference, 100, 101, 113
division by zero, 369
domain, 92, 226
domain mapping, 208
domain scaling factor, 307
donut, 321
double precision, 371
dynamic deforming, 228
dynamic shape varying, 229
dynamically deforming, 93, 227
- edge, 329
edge of the bill, 362
edge points, 337
efficient algorithm, 247
Elementary geometry, 9
ellipse, 220
ERBS, 21
ERBS curve, 217
ERBS tensor product surface, 295
ERBS triangle, 338
ERBS-evaluator, 384
error, 390
error term, 379, 380
Euclidean space, 48
Euler-MacLaurin integration formula, 379
Euler-Poincaré characteristic, 329
evaluation-matrix, 307
Expo-Rational B-spline triangle, 330

INDEX

- Factorization, 88
factorizing, 199, 375, 376
fast evaluator, 383
feature, 251
FFD, 320
First fundamental form, 265
flange, 321
Frenet frame, 240
function space, 207
function value, 390

game programming, 253
Gauss-Bonnet, 329
genus, 51, 95, 329, 356, 359
geocentric view, 96
geometric design, 369
geometric deviation, 250
geometric editing, 17
geometric mean, 390
geometric modeling, 383
geometrically smooth, 362
global curve, 217
global domain, 208
global surface, 296
global/local affine mapping, 92, 210, 226
global/local scaling factor, 92, 211, 226,
 227
Gordon surface, 137
graphic computation, 383
graphic display, 228
graphical system, 230, 241

head, 325
Helical curve, 242
helical curve, 241
Hermite blending surface, 279
Hermite interpolation, 17, 91, 92, 104,
 127, 210, 224, 226, 307, 308, 343,
 383
Hermite spline, 106
hierarchical B-splines, 324
history, 3, 14, 95, 110
homogeneous barycentric coordinates, 330
homogeneous coordinate, 240
homogeneous coordinates, 4
homogeneous matrix, 228, 240, 241, 251,
 307
Homogenous coordinates, 56
IEEE standard, 370
implementation, 369, 376
improved precision, 371
indexing, 220
information, 242
inheritance, 391
initializing, 386
inner loop, 299
integral, 369
integration interval, 380
interpolation, 95, 217
interpolation matrix, 91, 224
interpolation point, 231, 251
interpolation theory, 98
intersect, 231
intrinsic parameter, 185, 189, 197, 201,
 202, 214, 377
intrinsic property, 240
inverse Fourier integral, 111
invertible, 241
iterative process, 380

knot insertion, 131
knot interval, 197, 203
knot removal, 322
knot vector, 17, 185, 209, 217, 229, 387

Lagrange interpolation, 366
Lagrange polynomial, 102
Lagrange's identity, 266
linear interpolation, 88, 117
linearly independent, 241
local arc curve, 239, 242, 245
local Bézier curve, 246
local Bézier patches, 296, 308
local Bézier triangle, 343
local coordinate system, 231
local curve, 217, 224, 227, 231
local function, 17, 207
local origin, 228, 238
local patches, 295, 306
local sampling, 230

- local support, 189, 217
local surface, 304
local triangle, 338
locally equal, 350
logarithmic spiral, 241, 242
lower bounds, 377
- main diagonal, 331
main directions for derivatives, 336, 340
mathematical rigor, 12
matrix inversion, 228
matrix notation, 123, 131
matrix template, 307
max error, 388
max norm, 245, 250, 390
maximum deviation, 247
maximum negative binary exponent, 377
maximum normal value, 371
maximum positive binary exponent, 377
measuring method, 251
measuring the result, 250
mechanical spline, 111
minimal calculation, 328
minimum normal value, 371
minimum number of knots, 220
Modern geometry, 7
modified Hermite interpolation, 239
mortar, 321
movie, 320
moving, 231
multilevel Expo-Rational B-splines, 218, 296
multilevel representation, 320
multiple knots, 209, 217, 296, 297
multiple representation, 221
multiplicity, 209
- neighboring vertices, 356, 367
Newton polynomial, 101
Newton's formula, 100, 102
non uniform rational B-splines, 133
Non-Euclidean geometry, 10
nonlinear splines, 235
norm, 390
normal value, 370
normalize, 240
- Notations, 20
number of samples, 229, 384
number of steps, 380
number system, 369
numerical integration, 369
NURBS, 17, 133, 235, 315
- open curve, 218, 219, 230
open parameter, 297
OpenGL, 133
optimal approximation, 230
optimal solution, 388
orientation, 240
origin, 231
original curve, 230, 242
original parametrization, 250
orthogonal vector, 240
orthonormal vector, 241
oscillating speed, 230
osculatory interpolation, 106
outer loop, 303
overflow, 369, 371, 377
overlap, 234
overloaded matrix multiplication, 307
- parabolic interpolation, 98
parallel patches, 296
parallelizing, 305
Parametric Surfaces, 259
part, 217
partial derivative, 299
partition of unity, 184
Pascals triangle, 221
Pentium 4, 373
Pentium Mobile, 373
petal, 230
piecewise linear approximation, 228
planar patches, 296
polymorphism, 391
polynomial B-splines, 17
position, 240
practical experience, 199
precision, 370, 388
preevaluation, 224, 229, 370, 383
programming, 369
Projective space, 55

INDEX

- public function, 384
rational function, 188, 198, 203
reconstruction, 242
rectangular part, 295
recursive definition, 188
red cubes, 60
reflection of the knot interval, 220
reliable algorithm, 369, 372, 374, 377
remaining error, 380
remark, 51, 186, 211, 216–218, 222, 224,
 296, 302, 305, 307, 315, 356, 372,
 374
reparametrization, 247
representative, 60, 328
requirement, 369
restriction, 188
resulting curve, 231
RGB frame, 321
Richardson extrapolation, 379
Romberg integration, 379, 388
rope, 253
Rose-curve, 230, 245
rotating local curves, 255
rotational mapping, 340
ruled surface, 4
sample interval, 388, 390
sample rate, 388
sample step, 229
sampled values, 384
sampling, 229
sampling vector, 229
scalable set, 372
scalable subset, 196, 197, 369
scale, 231
scaling, 91, 196, 224, 231
scaling factor, 184, 186, 197, 203, 231,
 245
scaling of local curves, 255
scaling of the domain, 385
scaling rule, 387
sculpturing, 320
Sea Shell, 312, 354
Second fundamental form, 267
selector, 60, 328
selectorgrid, 60
sensitive parameter, 379
shape modeling, 18
shot in the dark, 250
sign bit, 371
signal, 372
significant bit, 370, 382, 386
simplex, 330
Simpson method, 380
simulation, 383
single precision, 371
SINTEF, 16
smooth parametrization, 366
smoothness over an edge, 367
snapshot, 253
special effect, 253
special value, 370
speed, 93, 227, 239
staircase form, 235
standard curve, 219
standard parameter, 297
step function, 379
stl, 18
straight line, 231, 238
streaming, 305
stretching, 320
strictly increasing, 211
sub-domain, 380
sub-triangle, 354
Subdivision, 138
subdivision, 133
subdivision-surface, 133
subnormal value, 370, 371
subset, 197
Surface of revolution, 269
surface on triangulation, 362
surfaces, 297
Surfaces based on Curve constructions,
 273
symmetric, 203
symmetric local curves, 234
T-junction, 324
T-splines, 324
tangent plane, 264
tapering, 320

template, 18
temporary reference, 387
tensor product, 297
tensor product surface, 295
tessellation, 329
Theorem, 120, 122, 188, 196, 209, 210,
 337
time consuming process, 382
time consumption, 388
tolerance, 380, 390
top corner, 338
torus, 310, 320, 350
total evaluator, 228
transfinite interpolation, 137
translate the domain, 246
translation, 196
trapezoidal approximation, 379
trapezoidal method, 380
triangle, 329
triangular surface, 329
triangulation, 329
Trianguloid Trefoil, 13, 308
trigonometric polynomial, 218, 296
twisting, 320
two level binary search, 377

underflow, 369, 371, 377
uniform sampling, 229
upper bounds, 377
use of memory, 385

versed sine, 97
vertex, 329
virtual world, 320

Waring-Lagrange formula, 100
web, 253

yellow cube, 321