Methods for Constraint-based Conceptual Free-form Surface Design

Dissertation
zur Erlangung des akademischen Grades
Doktoringenieur (Dr.–Ing.)

vorgelegt der Fakultät für Informatik und Automatisierung
der Technischen Universität Ilmenau

vorgelegt von	Dipl.–Inf. Pavol Michalik
geboren am	18. 07. 1971 in Trnava
vorgelegt am	16. 08. 2003
Verteidigung am	6. 11. 2003
Gutachter	1. Prof. Dr. Beat Brüderlin
	2. Prof. Dr. Guido Brunnett
	3. Prof. Dr. Hans Hagen

Abstract

The constrained-based design of free-form surfaces is currently limited to tensor-product interpolation of orthogonal curve networks or equally spaced grids of points. The, so-called, multi-patch methods applied mainly in the context of scattered data interpolation construct surfaces from given boundary curves and derivatives along them. The limitation to boundary curves or iso-parametric curves considerably lowers the flexibility of this approach. In this thesis, we propose to compute surfaces from arbitrary (that is, not only iso-parametric) curves. This allows us to deform a B-spline surface along an arbitrary user-defined curve, or, to interpolate a B-spline surface through a set of curves which cannot be mapped to iso-parametric lines of the surface. We consider three kinds of constraints: the incidence of a curve on a B-spline surface, prescribed surface normals along an arbitrary curve incident on a surface and the, so-called, variational constraints which enforce a physically and optically advantageous shape of the computed surfaces.

The thesis is divided into two parts: in the first part, we describe efficient methods to set up the equations for above mentioned linear constraints between curves and surfaces. In the second part, we discuss methods for solving such constraints. The core of the first part is the extension and generalization of the blossom-based polynomial composition algorithm for B-splines: let be given a B-spline surface and a B-spline curve in the domain of that surface. We compute a matrix which represents a linear transformation of the surface control points such that after the transformation we obtain the control points of the curve representing the polynomial composition of the domain curve and the surface. The result is a 3D B-spline curve always exactly incident on the surface. This, so-called, composition matrix represents a set of linear curve-surface incidence constraints. Compared to methods used previously our approach is more efficient, numerically more stable and does not unnecessarily increase the condition number of the matrix. The thesis includes a careful analysis of the complexity and combinatorial properties of the algorithm. We also discuss topics regarding algebraic operations on B-spline polynomials (multiplication, differentiation, integration). The matrix representation of B-spline polynomials is used throughout the thesis. We show that the equations for tangency and variational constraints are easily obtained re-using the

methods elaborated for incidence constraints.

The solving of generalized curve-surface constraints means to find the control points of the unknown surface given one or several curves incident on that surface. This is accomplished by solving of large and, generally, under-determined and badly conditioned linear systems of equations. In such cases, no unique and numerically stable solution exists. Hence, the usual methods such as Gaussian elimination or QR-decomposition cannot be applied in straightforward manner. We propose to use regularization methods based on *Singular Value Decomposition* (SVD). We apply the so-called *L-curve*, which can be seen as an numerical high-frequency filter. The filter automatically singles out a stable solution such that best possible satisfaction of defined constraints is achieved. However, even the SVD along with the L-curve filter cannot be applied blindly: it turns out that it is not sufficient to require only algebraic stability of the solution. Tensor-product surfaces deformed along arbitrary incident curves exhibit unwanted deformations due to the rectangular structure of the model space. We discuss a geometric and an algebraic method to remove this, so-called, *Surface aliasing effect*. The first method reparametrizes the surface such that a general curve constraint is converted to iso-parametric curve constraint which can be easily solved by standard linear algebra methods without aliasing. The reparametrized surface is computed by means of the approximated surface-surface composition algorithm, which is also introduced in this thesis. While this is not possible symbolically, an arbitrary accurate approximation of the resulting surface is obtained using constrained curve network interpolation. The second method states additional constraints which suppress or completely remove the aliasing. Formally we solve a constrained least square approximation problem: we minimize an surface objective function subject to defined curve constraints. The objective function is chosen such that it takes in the minimal value if the surface has optimal shape; we use a linear combination of second order surface smoothing functionals.

When solving such problems we have to deal with nearly linearly dependent equations. Problems of this type are called *ill-posed*. Therefore sophisticated numerical methods have to be applied in order to obtain a set of degrees of freedom (control points of the surface) which are sufficient to satisfy given constraints. The remaining unused degrees of freedom are used to enforce an optically pleasing shape of the surface. We apply the *Modified Truncated SVD* (MTSVD) algorithm in connection with the L-curve filter which determines a compromise between an optically pleasant shape of the surface and constraint satisfaction in a particularly efficient manner.

Zusammenfassung

Der constraint-basierte Entwurf von Freiformflächen ist eine mächtige Methode im Computer gestützten Entwurf. Bekannte Realisierungen beschränken sich jedoch meist auf Interpolation von Rand- und isoparametrischen Kurven. In diesem Zusammenhang sind die sog. “Multi-patch” Methoden die am weitesten verbreitete Vorgehensweise. Hier versucht man Flächenverbände aus einem Netz von dreidimensionalen Kurven (oft gemischt mit unstrukturierten Punktewolken) derart zu generieren, dass die Kurven und Punkte von den Flächen interpoliert werden. Die Kurven werden als Ränder von rechteckigen oder dreieckigen bi-polynomialen oder polynomialen Flächen betrachtet. Unter dieser Einschränkung leidet die Flexibilität des Verfahrens. In dieser Dissertation schlagen wir vor, beliebige, d.h. auch nicht iso-parametrische, Kurven zu verwenden. Dadurch ergeben sich folgende Vorteile: Erstens kann so beispielsweise eine B-spline Fläche entlang einer benutzerdefinierten Kurve verformt werden während andere Kurven oder Punkte fixiert sind. Zweitens, kann eine B-spline Fläche Kurven interpolieren, die sich nicht auf iso-parametrische Linien der Fläche abbilden lassen. Wir behandeln drei Arten von Constraints: Inzidenz einer beliebigen Kurve auf einer B-spline Fläche, Fixieren von Flächennormalen entlang einer beliebigen Kurve (dieser Constraint dient zur Herstellung von tangentialen Übergängen zwischen zwei Flächen) und die sog. Variational Constrains. Letztere dienen unter anderem zur Optimierung der physikalischen und optischen Eigenschaften der Flächen. Es handelt sich hierbei um die Gausschen Normalgleichungen der Flächenfunktionale zweiter Ordnung, wie sie in der Literatur bekannt sind.

Die Dissertation gliedert sich in zwei Teile. Der erste Teil befasst sich mit der Aufstellung der linearen Gleichungssysteme, welche die oben erwähnten Constraints repräsentieren. Der zweite Teil behandelt Methoden zum Lösen dieser Gleichungssysteme. Der Kern des ersten Teiles ist die Erweiterung und Generalisierung des auf Polarformen (Blossoms) basierenden Algorithmus für Verkettung von Polynomen auf Bezier und B-spline Basis: Gegeben sei eine B-spline Fläche und eine B-spline Kurve im Parameterraum der Fläche. Wir zeigen, dass die Kontrollpunkte der dreidimensionalen Flächenkurve, welche als polynomiale Verkettung der beiden definiert ist, durch eine im Voraus berechenbare lineare

Tranformation (eine Matrix) der Flächenkontrollpunkte ausgedrückt werden können. Dadurch können Inzidenzbeziehungen zwischen Kurven und Flächen exakt und auf eine sehr elegante und kompakte Art definiert werden. Im Vergleich zu den bekannten Methoden ist diese Vorgehensweise effizienter, numerisch stabiler und erhöht nicht die Konditionszahl der zu lösenden linearen Gleichungen. Die Effizienz wird erreicht durch Verwendung von eigens dafür entwickelten Datenstrukturen und sorgfältige Analyse von kombinatorischen Eigenschaften von Polarformen. Die Gleichungen zur Definition von Tangentialitäts- und Variational Constraints werden als Anwendung und Erweiterung dieses Algorithmus implementiert. Beschrieben werden auch symbolische und numerische Operationen auf B-spline Polynomen (Multiplikation, Differenzierung, Integration). Dabei wird konsistent die Matrixdarstellung von B-spline Polynomen verwendet.

Das Lösen dieser Art von Constraintproblemen bedeutet das Finden der Kontrollpunkte einer B-spline Fläche derart, dass die definierten Bedingungen erfüllt werden. Dies wird durch Lösen von, im Allgemeinen, unterbestimmten und schlecht konditionierten linearen Gleichungssystemen bewerkstelligt. Da in solchen Fällen keine eindeutige, numerisch stabile Lösung existiert, führen die üblichen Methoden zum Lösen von linearen Gleichungssystemen nicht zum Erfolg. Wir greifen auf die Anwendung von sog. Regularisierungsmethoden zurück, die auf der Singulärwertzerlegung (SVD) der Systemmatrix beruhen. Insbesondere wird die *L-curve* eingesetzt, ein "numerischer Hochfrequenzfilter", der uns in die Lage versetzt eine stabile Lösung zu berechnen. Allerdings reichen auch diese Methoden im Allgemeinen nicht aus, eine Fläche zu generieren, welche die erwünschten ästhetischen und physikalischen Eigenschaften besitzt. Verformt man eine Tensorproduktfläche entlang einer nicht isoparametrischen Kurve, entstehen unerwünschte Oszillationen und Verformungen. Dieser Effekt wird "Surface-Aliasing" genannt. Wir stellen zwei Methoden vor um diese Aliasing-Effekte zu beseitigen: Die erste Methode wird vorzugsweise für Deformationen einer existierenden B-spline Fläche entlang einer nicht isoparametrischen Kurve angewendet. Es erfogt eine Umparametrisierung der zu verformenden Fläche derart, dass die Kurve in der neuen Fläche auf eine isoparametrische Linie abgebildet wird. Die Umparametrisierung einer B-spline Fläche ist keine abgeschlossene Operation; die resultierende Fläche besitzt i.A. keine B-spline Darstellung. Wir berechnen eine beliebig genaue Approximation der resultierenden Fläche mittels Interpolation von Kurvennetzen, die von der umzuparametrisierenden Fläche gewonnen werden. Die zweite Methode ist rein algebraisch: Es werden zusätzliche Bedingungen an die Lösung des Gleichungssystems gestellt, die die Aliasing-Effekte unterdrücken oder ganz beseitigen. Es wird ein restriktionsgebundenes Minimum einer Zielfunktion gesucht, deren globales Minimum bei "optimaler" Form der Fläche eingenommen wird. Als Zielfunktionen werden Glättungsfunktionale zweiter Ordnung eingesetzt. Die stabile Lösung eines solchen Optimierungsprob-

lems kann aufgrund der nahezu linearen Abhängigkeit des Gleichungen nur mit Hilfe von Regularisierungsmethoden gewonnen werden, welche die vorgegebene Zielfunktion berücksichtigen. Wir wenden die sog. Modifizierte Singulärwertzerlegung in Verbindung mit dem L-curve Filter an. Dieser Algorithmus minimiert den Fehler für die geometrischen Constraints so, dass die Lösung gleichzeitig möglichst nah dem Optimum der Zielfunktion ist.

Contents

Chapter 1

Introduction

In the last two decades, computer-aided design systems have developed to powerful 2D and 3D modeling tools. Nevertheless, the use of a state-of-the-art modeler is still far from simple and intuitive. It requires a lot of learning and preparation before the intention of the design is met. The weak point of current systems is the way the models are constructed: the designer has to define a sequence of modeling operations which generate the model. This, so-called, design history has to be carefully planned before the actual design work starts. As a consequence, the designer must posses a relatively detailed a-priori knowledge about the model. Modern systems aid the user with sophisticated interfaces which significantly simplify the design work. For example, the concept of parameterizable features has become a *de facto* standard in computer-aided-engineering. However, the dependency of the model on a predetermined sequence of operations remains.

Recently, a new paradigm to geometric design the, so-called constraint-based design, has been proposed. Important design decisions and consistency conditions are expressed by relations (constraints) among the geometric elements of the model. The design work proceeds in a interactive manner: the designers explore new variations of shapes by interactive manipulations. There is no fixed order of design operation; instead, based on specified constraints and the intention of the user, the built-in logic of the modeling system derives the shape of the model. The major difficulty of constraint-based approaches is the *constraint solving*. This means to compute the values of degrees of freedom such that defined constraints are satisfied. For this reason, the set of supported geometric elements is usually restricted to points, line segments and circular arcs in 2D and to points and regular surfaces such as planes, spheres or cylinders in 3D. Although this is sufficient for many applications, concentrating the effort on these cases hinders the extension of this paradigm to other areas, specifically, to free-form surface modeling. We claim, that in the context of surface modeling the power of constraint-based methods have not yet been fully exploited.

1.1 Goals of the thesis

The experience we have gained from investigating methods for interactive manipulations of solids with constraints for planar and analytic surfaces [8, 7, 21] has led to the idea to extend the constraint-based approach to free-form surface models. The fundamental idea remains: points, curves and surfaces are related to each other by geometric relations, particularly, incidence and continuity constraints. The designer selects curves, points or regions on the surface and changes their shape or position. In order to satisfy the specified constraints the surface has to follow these modifications. Some results of these efforts were published in [54, 55, 56]. The, so-called, constructive constraint solving methods, [43, 42, 44], satisfy the constraints using a sequence of construction steps. At each step a geometric or algebraic operation is applied which determines the unknown objects. Roughly, modeling systems of this type consist of two substantial components:

1. A set of "constructors" (shape operators) which determine unknown degrees of freedom of an object from constraints and objects which are already known

2. A constraint solving engine which determines the order in which these constructors should be invoked and resolves potential conflicts.

It is not the intention of this thesis to present a complete modeling system. Rather, we are concerned with fundamental shape operators which can be used for constraint based design of free-form surfaces, especially, in a conceptual design phase. We will break down a construction step to interpolation of a B-spline surface from one or several arbitrary 3D curves or points and their representation in the parametric space of the surface. We are concerned with following topics:

- Definition of constraint-based shape operators for conceptual design of B-spline surfaces
- Efficient and numerically stable methods for setting up of equations for following linear constraint problems:
  - Incidence of a non iso-parametric curve on a B-spline surface
  - Fixed normals of a B-spline surface along a non iso-parametric curve (or tangent plane continuity of two B-spline surfaces along such curve)
  - Enforcing of a smooth or "visually fair" shape of a curve or surface
- Numerical solving of the resulting interpolation problems. Generally, such problems are under-determined and ill-posed, thus special numerical methods have to applied in order to obtain a numerically stable and esthetically pleasing solution

1.2 Structure and contents of the thesis

The thesis is divided into seven subsequent chapters as follows:

Chapter 2 reviews state-of-the-art free-form surface design methods. We describe the concept of an interactive sketch- and constraint-driven modeling system. The central operation is the interpolation of B-spline surfaces from arbitrary curves; we discuss known approaches and introduce our method to solve this problem.

Chapter 3 The constraints, expressed by systems of linear equations, are categorized with regard to the method how the equations are obtained. We consider incidence, tangency and variational constraints.

Chapter 4 describes an efficient algorithm to formulate equations which constrain the incidence of an arbitrary curve on a B-spline surface. We introduce the concept of "unevaluated" polynomial composition: We generalize the blossom based algorithm for composing Bezier polynomials [18] to B-splines and show that a so-called composition matrix can be obtained that transforms the control points of the surface to the control points of the incident curve. We use this matrix to formulate linear equations for curve-surface incidence. We analyze the combinatorial properties of the algorithm, discuss efficient methods for computing products of B-splines and present an efficient data structure for storing the intermediate results of the algorithm so that optimal run-time behavior is achieved.

Chapter 5 We derive linear equations which enforce tangent plane continuity of two surfaces along an arbitrary curve incident on both surfaces. We assume that the surface normals along an arbitrary curve on that surface are known and extract the necessary and sufficient conditions for the degrees of freedom of the surface by symbolically computing the scalar product of the normal and the tangent fields of the incident curve.

Chapter 6 demonstrates that the framework of methods elaborated so far, is perfectly suited to efficiently compute the normal equations of the so-called quadratic surface fairing functionals.

Chapter 7 The problem which is addressed in this chapter is that the inverse linear problem associated with generalized curve-surface incidence constraints belongs to the category of the so-called ill-posed problems which cannot be solved in the usual straight-forward manner. We consider *regularization* methods based on Singular Value Decomposition (SVD). The "L-curve"

method discussed here provides a generic and reliable tool to select a numerically stable solution best satisfying the defined constraints. It turns out that an algebraically stable solution is not satisfactory with regard to the quality of the so obtained surfaces. Tensor-product surfaces deformed along arbitrary incident curves exhibit unwanted deformations due to the rectangular structure of the model space. We name this the "Surface-aliasing effect" and identify reasons for that.

Chapter 8 discuss a geometric and an algebraic method to remove the aliasing-effect: The first method reparametrizes the surface such that a general curve constraint is converted to iso-parametric curve constraint which can be easily solved by standard linear algebra methods without aliasing. For that purpose, a method is introduced, to approximate a surface resulting from polynomial composition of two B-spline tensor-product surfaces. The so obtained surface locally reparametrizes the original surface according to stated curve incidence constraints. The second approach circumvents the reparametrization by stating additional constraints which suppress or completely remove the aliasing. Formally we solve a constrained least square problem which minimizes a surface fairing functional (a technique elaborated in chapter 6) subject to defined curve constraints. We apply an L-curve based method to solve the ill-posed variational problem: a regularization method called "Modified Truncated Singular Value Decomposition" is used to obtain a surface which compromises the satisfaction of defined constraints and the visual quality of the surface.

Chapter 9 summarizes the results of the thesis.

Appendix A contains the definitions and notations used throughout the thesis

Chapter 2

Methods for free-form surface design

The objective of our approach is to develop a surface modeling method which satisfies following criteria:

- Easy intuitive handling: Designers are usually not experts in surface mathematics. Thus, shape design operations should be decoupled from the internal mathematical description of the model.
- Flexibility: Design is an iterative process. The design intention will be rarely met by the first sketch. It should be possible to change previously made design decisions easily, without too much additional effort.
- Efficiency: Shape design operations must be performed rapidly, if possible in real-time.

In the following section we briefly describe two fundamental approaches to computer aided design. Section 2.2 reviews the currently applied free-from surface design methodologies with regard to the above criteria. We introduce an alternative to standard surface modeling methods in section 2.3: interactive sculpting of B-spline surfaces by free-hand sketches under the maintenance of curve-surface incidence and tangency constraints.

2.1 Parametric methods for CAD

Basically, two approaches to computer aided design (CAD) can be identified: the history based and the non-history based approach. The history based approach describes model by a sequence of geometric shape operations ordered in a tree or a

directed acyclic graph. This, so-called, *design history* has to be prescribed by the user. Usually, the shape operators are *parameterized* – the model is determined after specific values are assigned to external parameters of the operator. This dependency of the model on a sequence of operations and external parameters is called *parameterized design history*. The constraint-based approach describes the model by a set of geometric elements and a set of relations (or geometric constraints) among the elements – the *constraint graph*. The nodes of the graph correspond to variables (degrees of freedom) which determine each geometric element, the edges of the graph correspond to geometric constraints. Any element (its degrees of freedom) in the graph can be modified: the satisfaction of constraints is established in the *constraint solving* process where the degrees of freedom of influenced elements are recomputed such that all constraints are satisfied.

The fundamental difference between the two approaches is that editing of models is restricted to changes of the parameters of the fixed design history. In other word, the shapes which can be generated are restricted by given design history and parameterization. With constraint-based approach, a change can be applied to any element of the model within the defined constraints. The order of required operations is determined automatically during constraint solving. Thus, the possible shapes are determined entirely by defined constraints and degrees of freedom in the constrained objects. Constraints can be added or deleted as required. Therefore, this approach is occasionally called "constraint-driven". Constraint solving can be approach in different ways, see e.g. [71] for an overview of currently applied methods. For example the, so-called, constructive approaches, [43, 42, 44, 7, 21], satisfy the constraints using a sequence of construction steps. This "construction plan" is determined automatically, based on information available from the constraint graph. At each step geometric or algebraic operations are applied which determine the unknown degrees of freedom.

The power of a constructive constraint-driven modeling system depends on the set of available shape operators and the ability to determine the construction plan from given constraint graph. We will concentrate on the definition of constraint-based shape operators for free-form surfaces suited for application in a constructive constraint-based modeling system.

2.2 Known methods for free-form surface design

The goal of free-form surface design is to obtain a surface the shape of which satisfies given design criteria. In this section we look at a particular design problem which demonstrates the difficulties of this task. Without loss of generality, we will consider B-Spline surfaces as defined in §A.1. Note that all the alternative free-form surface representations, e.g. Bezier or Coons-patches, β-splines or

ν-splines, see [41, 25, 61], can be expressed in terms of B-Splines.

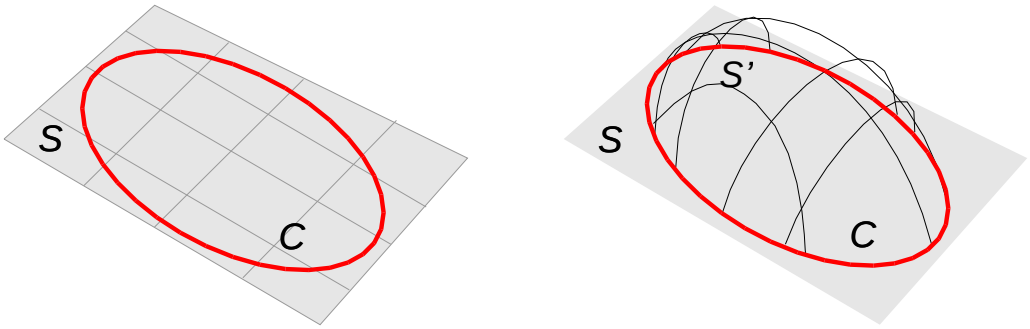
B-spline surfaces and the various related surface representation schemes have gained a large popularity in computer aided design of sculpted shapes. This results mainly from the property that the degrees of freedom (DOFs) inherent to the chosen surface representation scheme, i.e. the control points, vectors, or various auxiliary parameters have (at least to some extent) a predictable influence on the shape of the surface. Therefore, the dominant approach to interactive free-form surface sculpting is a direct manual modification of DOFs. Although, theoretically, an arbitrary shape can be modeled by re-positioning the control points of a B-spline surface, in practice, the types of changes one can achieve are quite limited. An attempt to deform a B-Spline surface this way, beyond making a few bumps or dents on it, is usually quite de-motivating:

Consider figure 2.1. The goal is to create a "dome" shaped surface S' shown in fig. 2.1(b) by editing the control mesh of an planar surface S fig. 2.1(a) such that the surface passes through a given curve C (the shape of C must remain fixed). It would be difficult, if not impossible, to design such surface by manual repositioning of the control points. The number of degrees of freedom is usually not a problem: one can insert arbitrary many control points into a B-spline surface. For example, a B-spline surface with 10×10 control points already has 100 independent degrees of freedom in each spatial dimension. However, for this purpose, the large amount of DOFs is not really helpful: it is not obvious which control points and how should be changed in order to achieve the desired shape. Thus, this approach does not meet the criteria of intuitive and easy handling. Also, its flexibility is limited: the effort of introducing a change to the dome surface (e.g. changing the shape of C) equals the effort of sculpting the surface from scratch.

Hence, we need a mathematical definition of the relationship between the DOFs (control points) of the curve and the DOFs of the surface: the change of DOFs on either side, causes the change of dependent DOFs of the other object. Basically, there are two approaches to establish this relationship described in the literature:

- the "Warping" or Free-Form deformation (FFD) methods and
- Constraint-based approaches, particularly, the so-called surface skinning and multi-patch methods

Both methods pursue the same goal: determine the degrees of freedom of the modified object such that the curve or surface takes in the desired shape. They differ in the way how the determination of DOFs is realized. Also, the philosophy behind both approaches is different: FFD methods pursue the modification as a primary goal – already deformed or otherwise "fixed" parts of the surface cannot be considered. The constraint-based methods proceed exactly the opposite way: they



(a) (b)

Figure 2.1: The "dome" surface example: it is not obvious how to change the control mesh, shown by straight lines in fig. (a), of the planar surface in fig. (a) such that the dome-shaped surface passing through curve C, see fig. (b), is obtained.

implement shape operators which create a new surface from constraints defined between the surface and previously determined ("fixed") elements. In general, it is not assumed that once determined surfaces will be further modified. In the following we will describe both approaches in more detail.

2.2.1 Warping methods

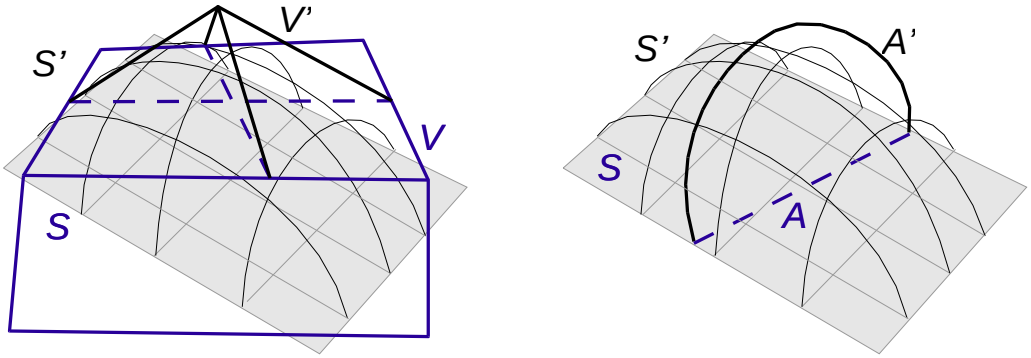
The idea of Warping methods is to embed the modified free-form shape in the, so-called, deformation space. If the deformation space is modified, the embedded shape "warps" according to the changes applied to the deformation space. The most commonly used analogy for FFD is to consider an object embedded in a parallelepiped of clear, flexible plastic. If the lattice structure is deformed, the object inside the lattice will also be deformed, [59].

The technique proposed by Sederberg and Parry in [73] embeds the modified free-form surface in E^3 in a tri-variate Bezier volume

$$V : \Omega_{u,v,w} \rightarrow E^3,\ V = V(u,v,w),\ \Omega_{u,v,w} \subseteq R^3$$

defined by tensor product of three Bezier spaces of dimensions $(l+1) \times (m+1) \times (n+1)$. The shape of V is determined by $(l+1)(m+1)(n+1)$ lattice of control points in E^3. The domain space $\Omega_{u,v,w}$ of V is a paralellpiped region of R^3 with dimensions $\langle a,b\rangle \times \langle c,d\rangle \times \langle e,f\rangle$. The deformation proceeds as follows:

1. Let $\{S_{i,j} : 0 \leq i < p, 0 \leq j < q\}$ be the set of control points of the B-Spline surface $S(s,t)$. Apart from the change in the names of the variables



(a) (b)

Figure 2.2: The Warping-methods embed the deformed surface S into a deformation space: (a) Sederbergs approach uses a tri-variate Bezier volume V. Changing V to V' "warps" S to S'. (b) The Axial-Free-form deformations use an "axis" curve to deform a surface. The dome-shape is created by deforming axis A to A' yielding a deformation of S to S'

the notation remains the same as in §A.1

2. Compute the embedding of S in the domain space of V: impose a local coordinate system on $\Omega_{u,v,w}$ by specifying any control point of S as $S_{i,j} = u_{i,j}\vec{U} + v_{i,j}\vec{V} + w_{i,j}\vec{W}$ with $\vec{U} = (b-a, 0, 0)$, $\vec{V} = (c-d, 0, 0)$ and $\vec{W} = (f-e, 0, 0)$. The $u_{i,j}, v_{i,j}, w_{i,j}$ coordinates are found by solving the linear system of equations for known $S_{i,j}$ and $\vec{U}$, $\vec{V}$, $\vec{W}$.

3. Deform the control points of V

4. There are two possibilities how to obtain the control points of the deformed surface:

   (a) Compute the deformed surface S' by setting $S'_{i,j} = V(u_{i,j}, v_{i,j}, w_{i,j})$. This is only an approximative method and may exhibit undesirable distortions of S, see [59].

   (b) Recognize that the deformed surface is a composition of the two mappings V and S and exactly compute the higher degree B-Spline surface $S' = V \circ S = V\,(u(s,t), v(s,t), w(s,t))$

This technique and its clones or extensions ([33], for example) is widely used because of its power to create interesting shapes easily and with low computational effort, if the method 4(a) is used. The method 4(b) is computationally more

involved. However, we will show in chapter 4 that with certain computational preprocessing effort the exact control points of S' can be computed very efficiently.

It has been recognized by Coquillart et. al, that one limitation of Sederberg's and Parry's method is the restriction on the parallepiped topology of the deformation space. In [13] Coquillart et al. have proposed using free-form volumes of arbitrary topology; this technique is known as Extended-Free-form deformations (EFFD). This extension allows a more flexible definition of the deformation space and enlarges the inventory of types of deformations introduced to a surface. However, the simplicity of Sederberg's approach is lost: the embedding of the modified surface in the deformation space, i.e. determination of the u, v, w coordinates for each control point of S from step 2, requires solving of non-linear systems of equations. Also, computing the exact control points as pointed out in 4(b) becomes very difficult, if not impossible.

Another interesting extension to FFD-techniques are the so-called Axial-Free-form deformations (AFFD), introduced by Lazarus et al. in [49]. The deformation space is not defined as an tri-variate free-form volume. Instead, a user defined "axis", a parametric curve in 3D, serves as a deformation tool:

1. Let the axis be defined as an 3D B-Spline curve $A(t)$ with control points $\{A_i : 0 \leq i < n\}$ and let S be the deformed surface.

2. Define a local coordinate system at each point of $A(t)$ by introducing three vector field curves $\vec{U}(t)$, $\vec{V}(t)$ and $\vec{W}(t)$ which must be linearly independent for all t. A good choice for the local coordinate system is the Frenet-frame of $A(t)$.

3. Associate each control point of $S_{i,j}$ with a point on $A(t_{i,j})$ and determine values $u_{i,j}, v_{i,j}, w_{i,j}$ so that $S_{i,j} = A(t_{i,j}) + u_{i,j}\vec{U}(t_{i,j}) + v_{i,j}\vec{V}(t_{i,j}) + w_{i,j}\vec{W}(t_{i,j})$

4. Deform the control points of A, yielding a deformed curve A' and new vector field curves $\vec{U}'$, $\vec{V}'$ and $\vec{W}'$

5. Compute the deformed surface S' by setting $S'_{i,j} = A'(t_{i,j}) + u_{i,j}\vec{U}'(t_{i,j}) + v_{i,j}\vec{V}'(t_{i,j}) + w_{i,j}\vec{W}'(t_{i,j})$.

Since AFFD are conceptually similar to EFFD, the main difficulty of AFFD-methods is the step 3: to find the relationship between the "deformation space" (in this case the axis A) and control points of the deformed surface S. The parameter values $t_{i,j}$ are usually determined by computing a point on A such that the Euclidian distance between the points $A(t_{i,j})$ and $S_{i,j}$ is minimal, which also requires numerical methods.

The axial FFD allow to deform a surface locally, according to the position of the axis curve relative to the deformed surface: In [79], Singh and Fiume determine the amount of change of each surface control point by an user-defined potential field function (the, so-called, density function)

$$f : R^1 \rightarrow \langle 0, 1\rangle \, , \; f = f\left(|S_{i,j} - A(t_{i,j})|\right)$$

emanating at the axis curve (the authors use the term "wire curve"). The intensity of the field is maximal at the wire curve and decays gradually with growing distance from it. The amount of change of each $S_{i,j}$ (determined as outlined in step 5) is weighted by the value of f which restricts the influence of a wire to a region of E^3 around the axis. Sing and Fiume have also discussed how to control the deformation induced by several wire-curves with overlapping regions of influence: the resulting deformation for a control point $S_{i,j}$ is proportional to the algebraic combination of all deformations applied to $S_{i,j}$: let $\vec{\Delta}^k_{i,j}$ be the amount of displacement of control point $S_{i,j}$ induced by the k-th of n wires and let f_k be the density function associated with k-th wire. The resulting deformation for $S_{i,j}$ is then proportional to

$$f_1(S_{i,j})\vec{\Delta}^1_{i,j} + \cdots + f_n(S_{i,j})\vec{\Delta}^n_{i,j}$$

Summarizing all Warping methods and with regard to the criteria listed in the beginning of chapter 2 we note:

Easy handling: the deformation process proceeds interactively via suited graphical user interface. The users do not have to deal with internal surface representation: the shape of the surface follows the changes of the deformation space. A great advantage is that instead of dealing with hundreds or thousands of control points which determine the surface, one only has to deal with relatively few control points which determine the shape of the deformation space. In case of the traditional FFD methods, the users have to be aware of the restriction on the paralellpiped topology of the deformation space.

Flexibility: the warping methods are well suited for design of shapes which do not have to satisfy some additional exact criteria. Consequently, while introducing new modifications is easy and intuitive, it is not possible to keep previously designed parts of the model unchanged. Consider the dome-shaped surface example from figure 2.1: we have required that the modified surface passes through given curve C. This is hard to achieve by warping the surface S by means of FFD; The deformation of S is determined solely by the deformation space – there are no additional restrictions which assure that C stays incident on S (therefore the curve C is not shown in figures 2.2(a) and 2.2(b). Also, the Axial FFD or the wires-method do not guarantee incidence of the edited curve on the surface. The surface only roughly approximates the changes of the edited curve: Although

several deformations induced by several "wire curves" can be applied simultaneously, keeping some wires fixed and changing the others does not guarantee that the surface stays locally unchanged.
Efficiency: One needs to distinguish between Sederberg's original version of FFD and the extended FFD methods (Axial-FFD, Wires). The first allow very efficient, real time modifications. The second may require considerable time to compute the embedding of the modified shape in the deformation space. Then, however, the modifications proceed at interactive speed.

2.2.2 Constraint-based methods

The goal of constraint-based surface design is to obtain a surface which satisfies some specified geometric properties. For example, it should pass through a set of 3D curves or points, or it should be smoothly connected to another surface. These properties represent geometric constraints on the shape of a curve or surface, we speak of incidence constraints or geometric (or polynomial) continuity constraints. In this context, the class of linear interpolation methods is most frequently applied. There are two main families of interpolation methods which are considered standard in CAGD: (1) tensor-product interpolation (so-called "surface skinning") and (2) multi-patch methods.

2.2.2.1 Tensor product interpolation

The simplest interpolation scheme seeks a B-Spline surface which interpolates a lattice of 3D points $\{H_{i,j} : 0 \leq i < m, 0 \leq j < n\}$. Assume that the set of two-variate B-Spline basis functions $\{b_i^{k,\tau}(u)b_j^{l,\upsilon}(v) : 0 \leq i < m,\ 0 \leq j < n\}$ is given. The interpolation problem consists of finding the control points $F_{i,j}$ of a B-Spline surface

$$F(u,v) = \sum_{j=0}^{n-1}\sum_{i=0}^{m-1} F_{i,j} b_i^{k,\tau}(u) b_j^{l,\upsilon}(v)$$

and parameter values (α_i, β_j) such that $F(\alpha_i, \beta_j) = H_{i,j}$. For that one proceeds as follows:

1. Set $(\alpha_i, \beta_j) = \left(\frac{(\tau_{i+1}+\tau_{i+k})}{k}, \frac{(\upsilon_{j+1}+\upsilon_{j+l})}{l}\right)$
2. We use the fact that a two-variate B-Spline consists of tensor-product of two univariate B-Splines and subsequently solve two one-dimensional interpolation problems. First, interpolate n B-Spline curves $E_j(u) = \sum_{i=0}^{m-1} E_{i,j} b_i^{k,\tau}(u)$ through points $H_{0,j}, \cdots, H_{m-1,j}$: Set $dofs(E_j(u)) = \mathbf{E}_j$. Denote by $\mathbf{b}_u|_{u=\alpha_i}$ the

vector of B-Spline functions evaluated at α_i, i.e. $\{b_i^{k,\tau}(\alpha_i) : 0 \leq i < m\}$. Set up matrices

$$\mathbf{H}_j = \begin{bmatrix} H_{0,j} \\ \\ H_{m-1,j} \end{bmatrix}, \mathbf{A} = \begin{bmatrix} \mathbf{b}_u|_{u=\alpha_0} \\ \\ \mathbf{b}_u|_{u=\alpha_{m-1}} \end{bmatrix}, \mathbf{E}_j = \begin{bmatrix} E_{0,j} \\ \\ E_{m-1,j} \end{bmatrix}$$

and solve n linear systems of equations

$$\mathbf{H}_j = \mathbf{A}\mathbf{E}_j,\ 0 \leq j < n$$

obtaining the control points of each $E_j(u),\ 0 \leq j < n$.

3. Interpolate the control points of $E_j(u)$ in the other parametric direction: One assumes that the curves $E_j(u)$ are mapped to n iso-parametric curves $F(u, v = const. = \beta_j)$. Then, the control points of the surface $F(u, v)$ which satisfies $F(\alpha_i, \beta_j) = H_{i,j}$ are obtained by solving m linear systems

$$\mathbf{E}'_i = \mathbf{B}\mathbf{F}_i,\ 0 \leq i < m$$

where

$$\mathbf{E}'_i = \begin{bmatrix} E_{i,0} \\ \\ E_{i,n-1} \end{bmatrix}, \mathbf{B} = \begin{bmatrix} \mathbf{b}_v|_{v=\beta_0} \\ \\ \mathbf{b}_v|_{v=\beta_{n-1}} \end{bmatrix}, \mathbf{F}_i = \begin{bmatrix} F_{i,0} \\ \\ F_{i,n-1} \end{bmatrix}$$

It can be shown, see e.g. [68], that if one chooses (α_i, β_j) as in step 1 the matrices $\mathbf{A}$ and $\mathbf{B}$ are always invertible and both equation system always have one unique solution.

The popular "surface skinning" is a special case of the above procedure: Given a set of m 3D curves $\{E_j(u) = \sum_{i=0}^{m-1} E_{i,j} b_i^{k,\tau}(u) : 0 \leq j < n\}$ and set of parameter values $\{\beta_j : 0 \leq j < n\}$ one seeks a B-Spline surface $F(u, v)$ such that $F(u, v = \beta_j) = E_j(u)$. Basically, one performs only the 3rd step of the procedure described above. The literature describes various extensions or modifications of this method. For example, one can prescribe more (or less) than mn points and solve overdetermined (or under-determined) linear systems. It is also possible to prescribe derivatives of arbitrary order at each point or curve and consider these constraints when solving for control points of F. The skinning method can be generalized to an interpolation of compatible curve network. Given two sets of curves

$$\{E_j(u) : 0 \leq j < n\},\ \{E'_i(v) : 0 \leq i < n\}$$

intersecting at 3D points $H_{i,j}$ find a B-Spline surface $F(u, v)$ such that $F(u = \alpha_i, v) = E'_i(v)$, $F(u, v = \beta_j) = E_j(u)$ and $F(\alpha_i, \beta_j) = H_{i,j}$, see [30] and chapter 7 of this thesis which contains a detailed description and a generalization of this interpolation scheme. For further references see, for example, [61, 25, 41].

2.2.2.2 Multi-patch methods

The skinning method and its clones deliver one B-spline surface which interpolates a set of iso-parametric curves or a rectangular lattice of 3D points. These methods are simple and efficient but they fail in two cases:

1. The set of interpolated points is unstructured, i.e. no lattice structure of fixed size exists

2. The curves cannot be mapped to a set of parallel iso-parametric lines in the domain of the surface

In this case one has to turn to the, so-called, *multi-patch* methods. This class of surface interpolation methods is concerned with methods for filling a 3D curve network by 4-sided tensor-product surface patches. Alternatively, 3-sided polynomial patches are used to fill triangular holes in the curve network[1]. Non-rectangular ("n-sided") holes are segmented into 3-sided or 4-sided regions and filled with surfaces based on the assumption that given 3D curves or their parts are the boundaries of the patch. Usually, the surfaces are required to meet with certain order of geometric continuity (usually G^1 or G^2) along their boundary curves. We point to Peters' paper [60] which contains a detailed overview about multi-patch methods; many details, algorithms and problems are also discussed in [41, Sec. 7.5].

Multi-patch methods are applied mainly in the context of surface reconstruction from unstructured point and/or curve data ("Reverse Engineering"). Multi-patch models resulting from Reverse engineering (RE) may consist of several thousands of patches. Often, the result of the surface interpolation is of bad quality: the choice of the wrong segmentation or a un-advantageous distribution of the point data causes unwanted "wiggles" or jumps in the resulting surfaces. Then, the process needs to be restarted with a new segmentation or point distribution. Experience shows, that from a designer's point of view, it is simpler to change the surface model interactively, inside restricted areas or near characteristic points or curves such that the overall shape of the model is kept. However, it usually not possible to manually improve the shape of RE-surfaces because there is no predefined dependency between suitable design parameters and the surface patches. Furthermore, multi-patch models are often generated "off-line", even on a different CAE platform. This introduces another problem: after re-importing the data into modeling system no editing is possible because there is no "design history" that describes the dependency of the surface patchwork on the input data.

[1] We will not further consider this kind of surfaces, see any standard book concerned with CAGD methods for details (e.g. [61, 25, 41]).

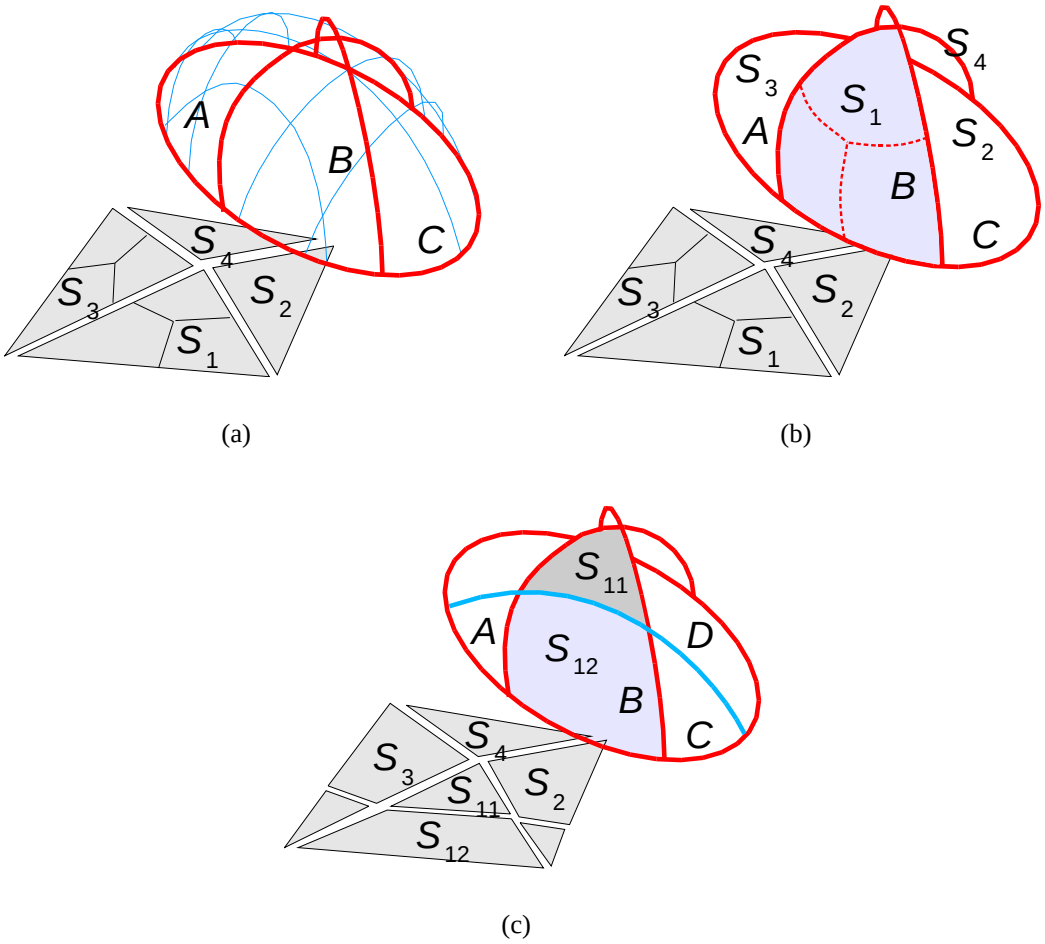


Figure 2.3: Design of “dome” surface using the multi-patch approach.

On the other hand, when a multi-patch surface model is created “from scratch”, the designer has the full freedom to choose a design history. Here, the problem is to create a model flexible enough to meet all specifications of the design. Modern surface modeling systems usually provide some kind of visual programming interface to build such multi-patch models. However, in most cases one has to create the surfaces “patch by patch” in certain order. Usually, shape operations which fill an n-sided hole given by its boundary curves, or solve the, so-called, “vertex enclosure problem”, [41, §7.5.1 and §7.5.2] are available. The user has to apply them in correct order.

For example, in order to create an editable model of the “dome”-shaped surface one could proceed as demonstrated in figure 2.3: The curve A, B and C were chosen to control the the shape of the surface, Fig. 2.3(a). The segmentation into

3 and 4-sided patches is shown in the left lower corner of each figure. The curves need to be split at points where they intersect; if no 3-sided patches are supported the triangular regions need to be further subdivided into 4-sided regions shown by the thin lines in triangles S_1 and S_3. The "holes" of the curve network are filled by surface patches, Figure 2.3(b).[2] The order in which the design operations were applied is stored in a design history with the curve A, B and C as "parameters" of the model: a modification of the curves causes re-evaluation of the design history yielding a new shape of the surface. Now consider the case that the three curves are not sufficient to control the shape – one would like to insert a new "parameter" curve, D, as demonstrated in Fig. 2.3(c). Since a different curve network arises (left lower corner of the figure) the old segmentation is no more valid and the design history of the entire model needs to be created from scratch.

For completeness we note that there are other surfaces representation schemes which allow to define surfaces on arbitrary topological domains, e.g. Seidel's B-patches [77] or Loop's generalization of B-splines to arbitrary domains [51]. However, they are not that well understood as the tensor-product representations which can be considered a de-facto standard in all common modeling systems. Thus, usage of one of these alternative surface representations causes considerable compatibility problems.

With regard to the criteria for a user-friendly surface modeling tool defined at the beginning of the chapter we conclude:

Easy handling: Surface design be means of skinning methods is relatively easy to understand even for less experienced users. However, one has to keep in mind the restrictions on the topology of the interpolated data. On the other hand, as even the simple example from figure 2.3 demonstrates, design of surface patch works requires a lot of experience, planning and relatively deep knowledge of surface mathematics; hence, we conclude that the intuitiveness and easy handling criteria are not satisfied.

Flexibility: The example from figure 2.3(c) shows that a slight correction of the design may have severe consequences, hence, the flexibility of the multi-patch method is low. However, in contrast to FFD methods, the constraints provide an exact way to control the shape of the surface; For instance, if the curve C in figure 2.3 is fixed, its incidence on S is guaranteed for any modification of A and B (under the assumption that the necessary compatibility conditions are satisfied).

Efficiency: The skinning methods are generally very efficient. One only has to evaluate the matrices and solve well-determined (and sparse) linear systems of equations as sketched in previous paragraph. The evaluation of a B-Spline func-

[2] In fact, the curves have to satisfy strict compatibility conditions; in this example, A, B and C have to intersect at fixed points. The sufficient compatibility conditions (esp. the "vertex enclosure" conditions) are more complicated and require reparametrization of the curves, see e.g. [60] or [41, §7.5.1 and §7.5.2].

tion has quadratic run-time complexity in the degree of the used B-Spline space. Therefore, usually, a low degree (such as quadratic or cubic) B-Spline basis is chosen. In order to satisfy the compatibility conditions the multi patch methods require reparametrization of the curves which may result in high-degree B-Spline spaces (see e.g. [60]). One has to consider that the segmentation into many surfaces requires solving of many high-degree interpolation problems. In addition, determination of a surface from its boundary curves usually results in an underdetermined problem, hence more sophisticated methods are required to solve the linear equation systems. For this reason, interpolation of large curve networks of complicated topology may take considerable time.

2.3 Generalized constraint-based surface modeling

One possibility to remove the disadvantages of previously described standard constraint-based methods is to remove the restriction that an interpolated curve must be mapped to an iso-parametric line. Assuming that the interpolation of B-spline surfaces from non iso-parametric curves is available, arbitrary 3D curves incident on a surface can be used as design parameters for surface sculpting: The designer inputs the curves directly in 3D. For that purpose, we have investigated free-hand sketching methods, see [47] and [57] for details: Each input curve is interpreted as a curve-surface incidence constraint. The user modifies the curves by free-hand pen strokes which yields a new shape of the surface every time a sketch transaction is finished. The model may consist of several surfaces which are connected to each other by curve incidence and tangency constraints. Internally, a sequence of shape operators is applied which determine each surface from given curves.

Proceeding this way, we combine the advantage of the FFD-approach (easy intuitive handling) with the advantages of constraint-based methods (precise control over the modified shapes). There is a direct algebraic relationship between the surface and the constraint-curves, hence, the previously designed surface can be preserved.

Non-rectangular surface patches can be simulated by, so-called, trimmed B-spline tensor-product surfaces: here, the extent of the surface is specified by a set of loops in the parametric domain of a B-spline surface. The regions defined by the loops are mapped to regions of the same topological shape on the surface. This allows to “simulate” non-rectangular surfaces of the same overall continuity as the underlying B-spline surface: an n-sided patch can be obtained by interpolating n 3D curves by one B-spline surface. On the resulting surface only the region restricted by the known curves in the domain of the surface is considered. Visualization and handling of trimmed tensor product surfaces is supported by

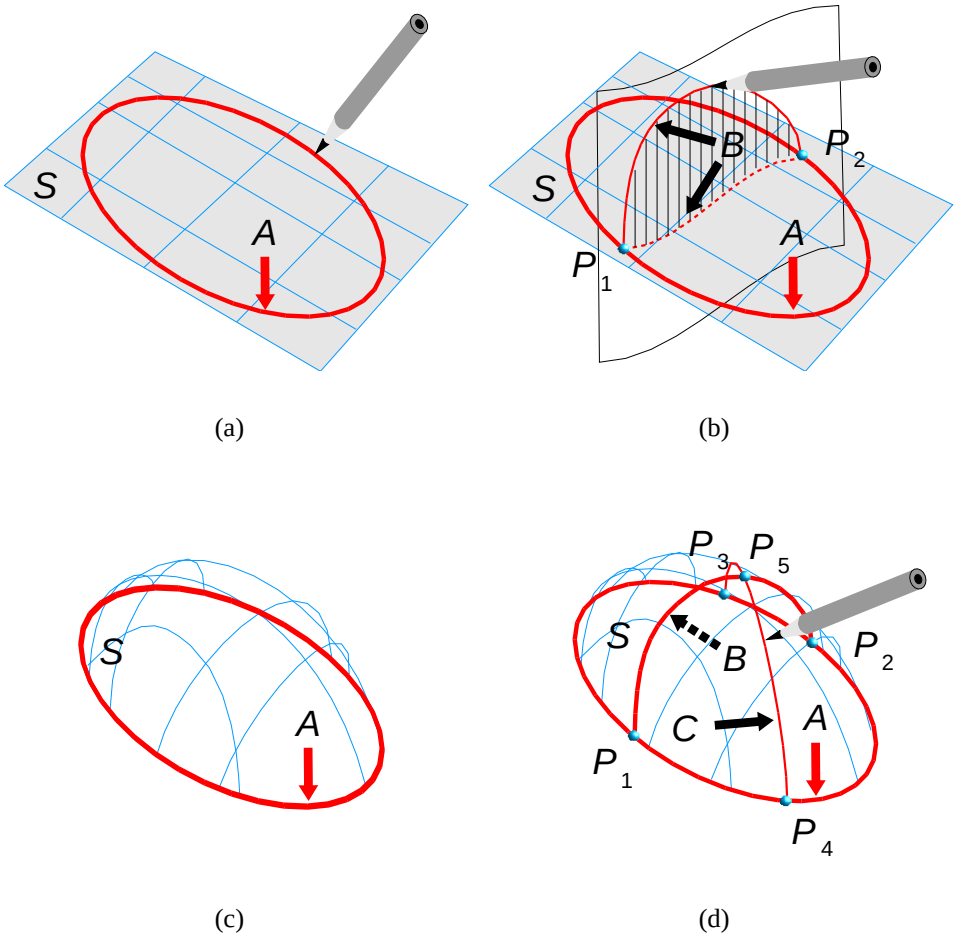


Figure 2.4: Sketch- and constraint-based design of the “dome” surface.

all common modeling systems, thus, no compatibility problems arise if trimmed surfaces are used.

In the following three paragraphs we will demonstrate this concept on design examples. We will demonstrate how the design of the dome-shaped surface used for demonstration purposes in previous paragraphs proceeds with a sketch- and constraint-based system. We will also address the creation of so-called free-form features.

2.3.1 The design example

Assume, one wishes to create the dome shaped surface, figures 2.4(a)-(d). The user sketches the closed curve A first. A curve-incidence constraint is automati-

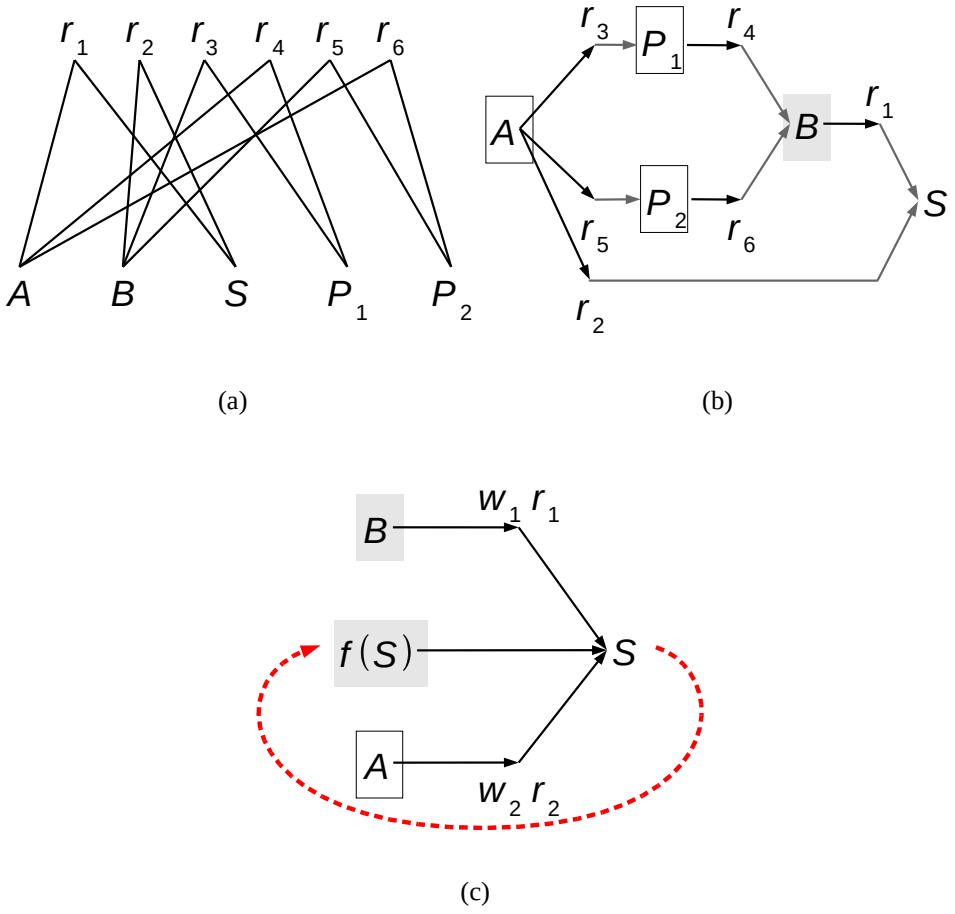


Figure 2.5: Constraint-graph and construction plan for the cap surface example. Elements A, P_1 and P_2 are fixed, B is selected for change. At the last stage the surface S is determined from the curves A and B, and from integral constraint $f(S)$. If errors at constraints $r_1(B,S)$ and $r_2(A,S)$ are larger than a prescribed tolerance the surface is refined and the last construction step is repeated.

cally generated. If desired, the region outside the closed curve can be trimmed from the surface. Since we want to preserve the shape of A it is marked as "fixed". In order to generate the dome shape of the surface, the curve B is drawn, fig. 2.4(b). The methodology for sketching 3D curves on surfaces was defined in [47] and [57].

The user modifies B by further pen strokes inside the so-called *sketch* space which is defined as an auxiliary free-form surface locally orthogonal to the modified surface, see [57]. In our example the sketch space is denoted by the hatched

region in figure 2.4(b). The curve of B is checked for interference with other elements. In this case, B intersects with A at points P_1 and P_2. It is necessary to capture these conditions by additional constraints. Here, 4 point-curve incidence constraints among both curves and points are recognized and generated automatically which yields a constraint-graph shown in fig. 2.5(a): the lower set of nodes denoted by upper case letters represent the objects (curves, points, surface), the upper row represent the incidence constraints, denoted by r_i. The edges of the graph connect a constraint node with related object nodes.

The order of evaluation is determined from current distribution of degrees of freedom in the constraint graph: in our example, the construction plan is as shown in fig. 2.4(b): the curve A is fixed which enforces fixed points P_1 and P_2. These consistency constraints consume 2 degrees of freedom from B – thus, the dependent control points of B must remain fixed. Every sketched input curve needs to be checked against these conditions.

Finally, the surface is determined from curves A and B. The resulting shape of S will depend on its initial parameterization, polynomial degree and number of DOFs; we will return to this topic in chapter 7. A good idea is to start with relatively simple surface and insert new DOFs at appropriate positions if errors at the constraints exceed defined limits. The error for each constraint is measured by back-substitution of the pre-image curve into S and evaluation of the difference among this exactly incident curve and the curve present in the model. In the following $|r_i|$ will denote this error for a specific constraint. At the last stage of the construction plan, shown in figure 2.5(c), the surface is determined (possibly in several iterations) as a solution of a constrained variational problem

$$\min_{\mathbf{S}} f(S) \;\; subject\ to \;\; |r_2| = 0 \wedge |r_1| = 0$$

where $f(S)$ is a "smoothness" constraint which regulates the overall shape of the surface. Here, the objective function $f(S)$ is a convex combination of quadratic surface functionals which minimize the area, thin-plate energy and variation of curvature of a tensor-product surface (section 3.3 and chapter 6). Following the experience of many researchers who have investigated variational problems of this kind (e.g. Brunnett, Greiner, Hagen, Hoschek, Seidel and others, [9, 5, 31, 48]), minimization of these expressions has been shown an effective approach to determine an optically pleasing surface with slowly varying curvature; more details are found in chapter 7.

The so determined construction plan stays valid as long as the user does not change the distribution of DOFs (i.e. does not fix or free an element). A new plan needs to be evaluated if elements are created and inserted into the model, or if existing elements are removed. This is illustrated in Fig. 2.4(d). Another curve, C, is introduced. New consistency conditions arise, if the user decides to keep

the shape of B: since B intersects with C in P_5 and A remains fixed, the number degrees of freedom in curve C is a-priori reduced by 3.

In summary we note: the model is represented by a constraint-graph which is updated depending on user's design actions. The construction plan is determined after an element of the graph (curve, surface, or point) has undergone a change of the state. This means that the user has marked an element as one of "fixed", "free" or "changed". Algorithms to obtain a construction plan from steady-state constraint graph have been proposed, see e.g. [7, 21]. However, these algorithms rely on the a-priori knowledge of how many DOFs of an object are consumed by a specific constraint. Since this is impossible to predict in case of ill-posed linear problems, see chapter 7, these methods have to be modified or completely re-evaluated which is one of the primary topics of our future work.

2.3.2 Design with Free-form Features

Often, only a part of a surface should be modified such that other regions are not influenced. The methods discussed so far, don't provide that possibility. Suppose, that a region on a B-spline surface is identified by sketching a loop (a loop is one closed or several connected curves), as demonstrated in Fig. 2.6(a). If we modify the surface only outside the loop B, there is no guarantee, that the shape of the shaded part of the surface will stay constant. This can be achieved only if the surface is split into two separate surfaces which share only a common boundary. This introduces additional objects into the model, therefore, we use the term "feature". We propose two methods:

The first is illustrated in figures 2.6(a)-(b): the user draws a loop A on surface S_1. The initial surface S_1 is cloned into S_2 and incidence constraints are generated between both surfaces and the curve A. Note that the loop may consist of several curves; then, an incidence constraint is generated for each curve. If the shape and position of the boundary curve is fixed, the user can edit S_2 by sketches as discussed above, figure 2.6(b). Since the surfaces are only related via their fixed boundaries no changes are carried over to S_1.

The second method differs in the way how the surface is cloned. Instead of creating a copy of current surface, a new surface is computed such that it reparametrizes the original surface inside a specified four-sided region. This is demonstrated in figures 2.6(c)-(d). The advantage of this method is that the reparametrized surface can be adjusted such, that it reacts very well to changes of a specific curve, or family of curves; the limitation is that the region must be rectangular. The surfaces S_1 and S_2 are connected by incidence and continuity constraints. Note that the boundary curves B_i are all mapped to iso-parametric lines of S_2 which makes the formulation of the respective constraints easier. This method is described in chapter 8.

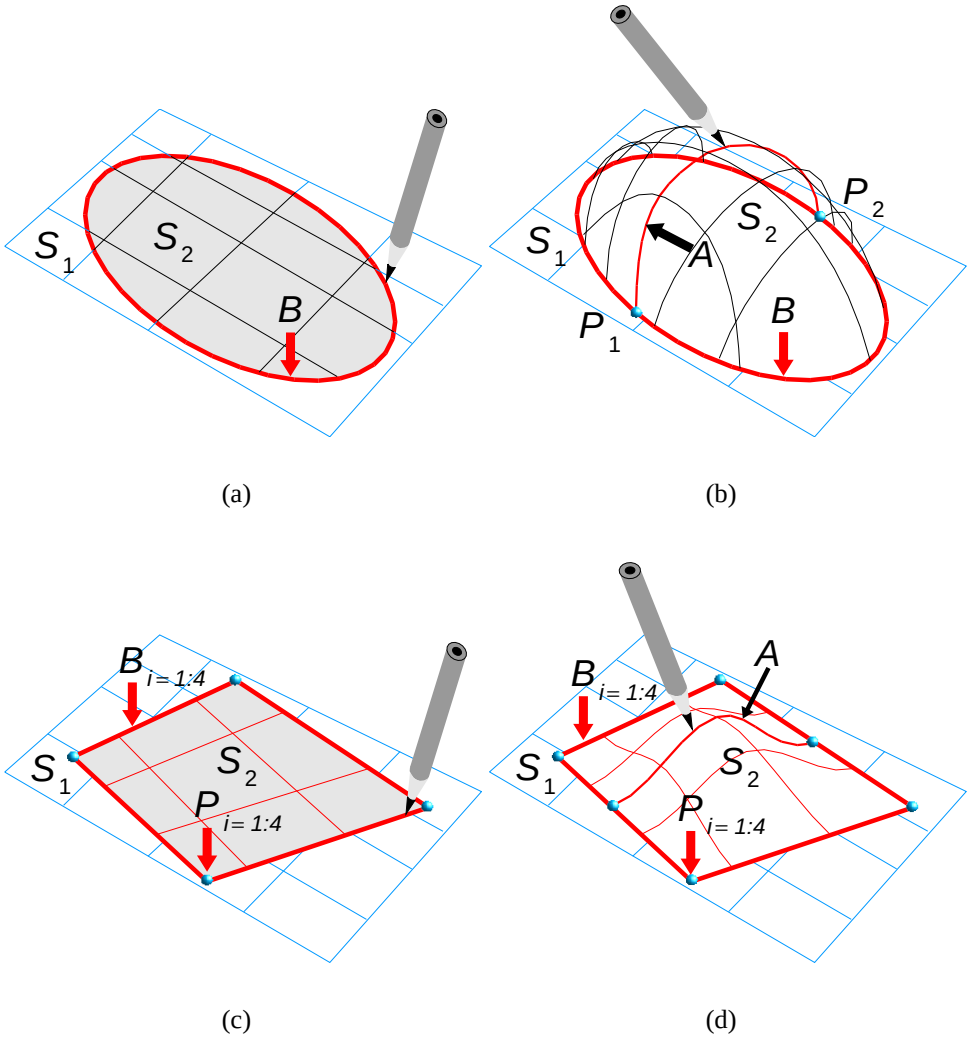


Figure 2.6: Creation of a trimmed "surface feature". Top: S_1 is cloned into S_2 and both surfaces are constrained to meet along the curve B. Then, if B is fixed, S_2 (or S_1) can be modified independently from the other surface. Bottom: Region of interest created by application of surface-surface composition. S_1 is locally reparametrized by S_2. The surfaces meet with G^1 continuity along B_1 to B_4.

The outline of a relational surface modeling platform as discussed above shows the advantages for conceptual design. Much less mathematical knowledge is required to implement a design intention compared, e.g., to the multi-patch methods. Also, the design and re-design cycles shorten considerably: the intention of the designer is immediately confirmed, or, rejected if the requirement is not compatible with specified constraints.

Chapter 3

Linear curve constraints

The central operation required in previous section is the interpolation of a surface from one or several arbitrary curves. Although, in principle, the approach is the same as described in §2.2.2, interpolation of surfaces from non iso-parametric curves is considerably more complicated. In addition, as it is commonly done for iso-parametric curves, we wish to enforce tangent plane continuity of two surfaces along an arbitrary curve incident on both surfaces. In the following section we review the methods known from the literature to approach both types of constraints. In section 3.2 we outline our approach to formulate the linear equations for these constraints. It turns out that the effort does not end with a numerically stable and efficient algorithm for formulating the equations. Solving of generalized curve incidence and tangency constraints requires so-called variational methods: We need to state additional conditions on the shape of the interpolated surface. These conditions are called variational constraints for B-Spline surfaces and are introduced in section 3.3. Although the former two types and the latter type are conceptually different, technically, their implementation requires the same set of basic operations: above all sparse matrix operations and symbolic computations with B-spline polynomials. The implementation of either constraint category is easy, once this "kernel" is available. In the remaining chapters of the thesis we will refer to this classification.

3.1 Related research

The problem is defined as follows: We seek a B-Spline surface which interpolates a given 3D B-Spline curve $Y(t)$. Assume that the set of two-variate B-Spline basis functions

$$\{b_i^{k,\tau}(u)b_j^{l,\upsilon}(v) : 0 \leq i < m,\ 0 \leq j < n\},\ (u,v) \in \Omega_{u,v} = \langle a,b\rangle \times \langle c,d\rangle$$

and a B-Spline curve in $\Omega_{u,v}$

$$Z : R^1 \rightarrow E^2,\ Z(t) = (u(t), v(t))\ ,\ u(t) \in \langle a, b\rangle \wedge v(t) \in \langle c, d\rangle$$

are given. The interpolation problem consists of finding the control points $X_{i,j}$ of a B-Spline surface

$$X\left(Z(t)\right) = X\left(u(t), v(t)\right) = \sum_{j=0}^{n-1} \sum_{i=0}^{m-1} X_{i,j} b_i^{k,\tau}\left(u(t)\right) b_j^{l,\upsilon}\left(v(t)\right) \tag{3.1}$$

such that $X\left(u(t), v(t)\right) = Y(t)$. As long as the curve $Z(t) = (u(t), v(t))$ does not change, the terms $b_i^{k,\tau}\left(u(t)\right) b_j^{l,\upsilon}\left(v(t)\right)$ are constants. Hence the non-linear part of the surface representation is eliminated and the problem becomes linear in the control points $X_{i,j}$ of the surface. Thus it is possible to obtain a linear system of equations

$$\mathbf{AX} = \mathbf{Y}$$

which restrict the incidence of $Y(t)$ on $X(u, v)$.

The problems are two-fold: first, it is not straight-forward how to formulate the equations which define the relationship between the surface control points and the control points of the interpolated curve. Second, obtaining a numerically stable solution of the above linear system of equations is a subtle procedure: The inverse problem

$$\mathbf{A}^{-1} : \mathbf{Y} \rightarrow \mathbf{X}$$

generally belongs to the class of so-called *ill-posed problems*. Simply stated, it is not possible to determine the status of the problem by counting the number of equations and unknowns; in some sense, the problem is simultaneously under- and over-determined. The following two paragraphs review the approaches to both problems known from literature.

3.1.1 Formulating the equations

The first problem can be approached in three different ways:

1. The discretization method
2. Continuous approximation
3. The Composition method

3.1.1.1 The discretization method

The first method proceeds as follows: the continuous curve-surface incidence problem is discretized by considering many point-surface constraints ordered along given 3D curve, see [20]. This approach is often satisfactory, however, it does not provide a closed formulation of the curve-surface incidence, i.e. there is no direct relationship between the control points of the incident curve and the control points of the surface. Also, the condition (or, the so-called, ill-posedness, discussed in chapter 7) of the associated inverse linear problem worsens – the equations contain too much noise (we have to postpone the definition of noise in this context to chapter 7) which makes obtaining a suitable solution harder.

3.1.1.2 Continuous approximation

The second approach was pursued by Welch et al. in [83] and [11]. Given X, Z and Y as above a continuous approximation problem is formulated: the linear system of equations in the unknown control points of the surface can be set up by minimizing the quadratic distance functional $\int_t |Y(t) - X(Z(t))|^2$ which yields a square matrix of equations linear in the unknown control points of X. The disadvantage of this approach is that the computation of the associated matrices is inefficient and numerically not stable (esp. because of the integration of high degree splines). Furthermore, the system matrix is obtained by formulating the normal equations. It is known, that the condition number of normal equations is the square of the actual condition of the problem [29].

3.1.1.3 The composition method

The third approach uses the fact that a 2D curve G is mapped to a 3D surface curve Y incident on X by taking linear combinations of Xs control points. In other words, each control point of Y can be written as a linear combination of control points of X yielding a linear system of equations in unknown DOFs of X. In [24] Elber and Cohen have shown how to obtain the equations for the special case when X is a Bezier patch and Z is one or several (arbitrarily oriented) line segments. This method is more efficient and numerically more stable. In our previous work [54, 55, 56] we have generalized this approach to B-spline surfaces and arbitrary domain curves. These papers do not contain all details; the complete description, examples and further improvements are described in chapter 4 of this thesis.

The composition method (if implemented carefully) is more efficient and numerically much more stable than the approach in [83] and [11]. It delivers a matrix with smaller overall errors and, in general, of smaller size. Furthermore it can be easily extended to obtain equations for other linear curve-surface constraints as

we will demonstrate below. Also note that constraints such as curve-surface incidence or continuity of two surfaces along an arbitrary curve need to be evaluated in both directions. For example, given several curves incident on a B-spline surface, maybe not all of them are initially selected as design parameters. I.e., after the surface is determined from given curves, the remaining curves must be recomputed. The composition method expresses a curve-surface incidence as a matrix equation in control points of that curve and surface. Then, in the "surface-curve" direction the constraint reduces to computing a matrix-vector product. This is one of the advantages of the composition approach. The alternative methods, [20, 83, 11] do not foresee that the constraints will be ever evaluated this way.

3.1.2 Obtaining a stable solution

Nevertheless, the inverse problem, obtaining the surface control points given the curve on surface, is still ill-posed – one has to turn to sophisticated methods to obtain a reasonable solution. Although the authors of [83, 11] have recognized that the linear systems of equations may be ill-conditioned, it is not clear how they remedy this. They have proposed to extract the null-space of the interpolation matrix by means of pivoted Gaussian elimination. It is known, see e.g. [36], that for ill-posed problems this method is not sufficient – we discuss this topic in more detail in chapter 7. When only Bezier patches with carefully chosen number of degrees of freedom are considered, as in [24], the ill-posedness is not that serious problem – and in most cases no particularly sophisticated equation solver is necessary.

The approaches [20, 83, 11] utilize the variational approach to surface design, see [9, 5, 48, 31]: generally, the incidence and continuity constraints are not sufficient to determine all DOFs of the surface. The undetermined DOFs of the surface are a solution of constrained variational problem which minimizes a given objective function with respect to specified constraints: A closely related topic is the, so-called, surface "fairing", very often applied in the context of scattered data interpolation. In fact, we will deal with this kind of problems, but in a more general setting: we will not restrict ourselves to iso-parametric or boundary lines of B-spline surfaces.

Our approach combines (improves and generalizes) several of the ideas: we formulate curve-surface incidence constraints in the style of [24] and solve variational problems as in [20], [83] or [11]. However, none of the mentioned publications have accessed the problem of obtaining a numerically stable solution from the ill-posed equations. For iso-parametric curve no ill-posedness occurs. Welch et al. have proposed to use pivoted Gaussian elimination. However, in most cases Gaussian elimination is not sufficient to reveal the rank of the matrix. One has to apply more sophisticated numerical methods to determine the optimal numerical

rank of the problem. Methods for solving ill-posed problems are known in linear algebra, see e.g. [36] for further backgrounds and a profound overview on *regularization* methods. An application of so-called "L-curve"-method, see [37, 10], based on Truncated Singular Value decomposition (TSVD) [35] and modified TSVD [38] is described in chapter 7.

3.2 Generalizing the composition method

3.2.1 Curve-surface incidence constraints

We use a generalized version of the composition method to approach the problem of obtaining the equations for curve-surface incidence. We generalize the results of DeRose and DeRose et al. [17, 18] which can be summarized as follows:

The so-called simploids (n-variate tensor-product of polynomial vector spaces, including the case $n = 1$) are always closed under polynomial composition. Denote by $\mathcal{R}^i$ the affine space of dimension i associated with the linear space R^i. A simploid is a mapping $M : \mathcal{R}^i \rightarrow \mathcal{R}^j$. DeRose has shown that given three Bezier simploids

$$Z : \mathcal{R}^i \rightarrow \mathcal{R}^j,\ X : \mathcal{R}^j \rightarrow \mathcal{R}^k,\ Y : \mathcal{R}^i \rightarrow \mathcal{R}^k,\ i \leq j \tag{3.2}$$

it is possible to obtain the Bezier control points of Y by computing the polynomial composition of X and Z which we denote by

$$Y = X(Z) = X \circ Z$$

A similar statement applies to piecewise polynomial B-spline simploids with the restriction that if the outer simploid X is a tensor-product of B-Splines no closed solution exists for Y in cases of $i \geq 2$ and $j \geq 2$, [18]. In this paper, the composition of B-splines was not performed directly: first, the B-spline was converted to Bezier representation and the composition was computed segment-wise for each Bezier simploid.

Each control point of Y is obtained by computing linear combinations of control points of X. We introduce the, so-called, *unevaluated composition* of two B-splines (with the restriction that $i < 2$ and $j \leq 2$) *without* the intermediate conversion to Bezier basis: Given X and Z as above we compute the matrix which transforms the control points of X to control points of Y yielding a matrix equation

$$\mathbf{Y} = \mathbf{A}\mathbf{X}$$

The matrix $\mathbf{A}$ is called *composition matrix.* The composition matrix is used in two ways:

- if $\mathbf{X}$ is known one obtains the control points of $Y = X \circ Z$ simply by performing a matrix multiplication
- for given $\mathbf{Y}$ solving this linear system of equations one obtains the control points $\mathbf{X}$ such that Y is incident on X

An efficient algorithm to compute the composition matrix is described in chapter 4.

3.2.2 Tangency constraints

We require that two surfaces join with continuous tangent planes along an arbitrary curve incident on both surfaces. Figure 3.1 demonstrates the problem: As for the incidence case, we assume that curves $Z(t) = (u(t), v(t))$ and $Z'(t) = (u'(t), v'(t))$ in the domain spaces of both surfaces are given. Both domain space curve are rendered into a 3D curve incident on both surfaces. Assume, that the control points of one of the surfaces, e.g. $X'(u', v')$, are known. We seek control points of the second surface $X(u, v)$ such that

1. The curve $Y(t)$ is incident on $X(u, v)$:

$$Y(t) = X\left(Z(t)\right) = X\left(u(t), v(t)\right)$$

2. The tangent planes of both surfaces along $Y(t)$ are coplanar

The sufficient and necessary condition for coplanar tangent planes is that the normals of both surfaces along $Y(t)$ are collinear. This is equivalent to the condition that the normals along $Y(t)$ are orthogonal to the tangent planes of the unknown surface $X(u, v)$ along $Y(t)$, see [24].

Denote by $N(t)$ the vector field curve which renders the normals of X' along the curve $Y(t)$. If the control points of X' are known the curve $N(t)$ can be computed symbolically or numerically, see e.g. [23]. The relation between $N(t)$ and the unknown surface $X(u, v)$ is established by requiring that the tangent field of the incident curve, denoted by $\frac{d}{dt}Y(t)$, is orthogonal to $N(t)$. Two vectors are orthogonal if their scalar product is zero, that is:

$$\left\langle N(t), \frac{d}{dt}Y(t) \right\rangle = \left\langle N(t), \frac{d}{dt}X\left(u(t), v(t)\right) \right\rangle = 0$$

The above equation needs to be resolved for the unknown control points of $X(u, v)$. The equations can be formulated very elegantly if the composition matrix for the incident curve is known, as demonstrated in chapter 5.

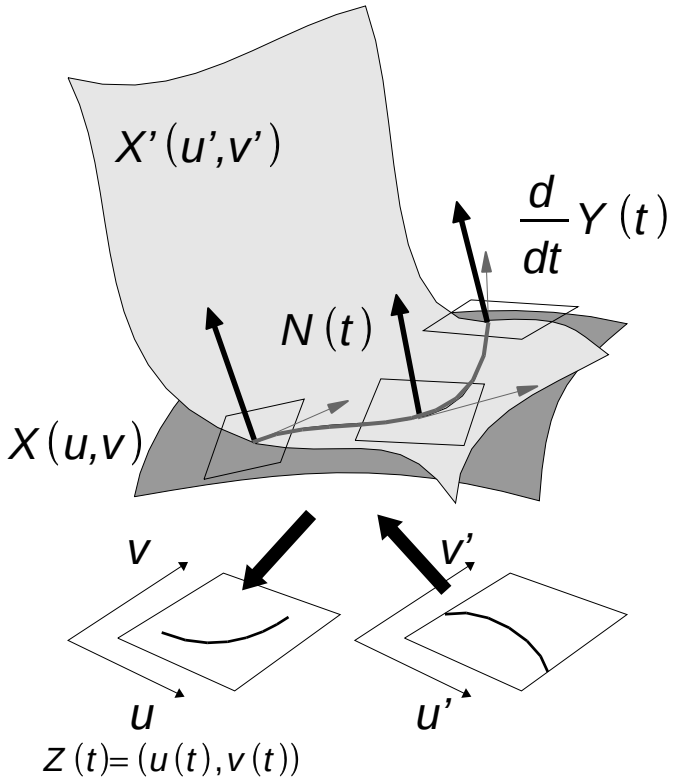


Figure 3.1: Enforcing the G^1 continuous join of two surfaces along an arbitrary curve incident on both surfaces.

3.3 Variational constraints

The goal is to construct curves or surfaces having optically pleasant shape that satisfy given curve or tangency constraints. The area of CAGD which deals with this problem is called variational design [9, 5, 31, 48, 32]. The problem is to define a mathematical criterion for "optically pleasing shape" of a surface. A commonly used approach is to define the so-called fairness functionals which take in minimal values for surfaces of optimal shape. Formally, for a B-spline surface $S(u,v) : \Omega_{u,v} \rightarrow R^3$, $\Omega_{u,v} \subset R^2$ one defines a scalar valued functional

$$\Phi : R^3 \rightarrow R^1,\ \Phi = \Phi\left(S(u,v)\right)$$

with the property that small value of Φ indicates that S has a more pleasant shape. The solution of a variational problem then consist of finding the control points $\mathbf{S}$ such that the surface satisfies given constraints and the functional takes in minimal value over the entire domain of S:

$$\min_{\mathbf{S}} \Phi\left(S(u,v)\right),\ \forall (u,v) \in \Omega_{u,v} \tag{3.3}$$

Hence, unlike the former constraint categories, variational constraints restrict the degrees of freedom of *one* object to take in such values that a property defined in terms of the functional is satisfied.

We will discuss surface functionals which minimize curvature, energy, length, area or similar properties. This selection arises from the observation that a curve or surface has an optically pleasing shape, if the functional of this type is near the global minimum. Exact formulation of these conditions results in large non-linear problems, therefore a variety of simplified functionals were proposed which provide good approximations, see e.g. [32]. What is common to all of them is that they depend on product of partial derivatives of the optimized surface and are quadratic in the degrees of freedom of the surface. Hence, they posses one well defined global minimum which can be computed efficiently by means of Gaussian normal equations. The normal equations are linear in control points of the surface, the system matrix is symmetric and, in general, sparse.

Although the usage of these constraints is frequently mentioned in the literature, not much information is available about an efficient and simple computation of matrices representing the normal equations for this type of surface functionals. We will describe a unified and efficient approach for symbolic computing of these matrices in chapter 6.

Chapter 4

Computing the incidence constraints

In previous chapter we have sketched a new application of polynomial composition: given a B-spline surface and a curve in the domain space of that surface, it is possible to obtain the control points of B-spline curve incident on the surface by a linear transformation of the control points of the surface. We use the resulting matrix equation as a constructor which determines the incident curve from known surface and vice-versa. This chapter presents an efficient method to generate the linear transformation in matrix form. Internally, a generalized version of the blossom based polynomial composition algorithm developed by DeRose et al. [18] is applied. In 4.1 we briefly review the concept of blossoming and introduce blossoms in so-called "unevaluated" form. In section 4.2 we derive the straight forward version of unevaluated composition for Bezier and B-splines. The complexity issues and efficient versions of the algorithm are investigated in section 4.3. Finally, in 4.4 some results are presented regarding run-time performance of the algorithm and the numerical stability of the composition matrix.

4.1 The blossoming kernel

4.1.1 The Blossoming principle

In a 1987's technical report [65], Ramshaw has introduced a new approach to polynomial functions in Bezier or B-spline form: the so-called *blossoming* principle. It uses the duality of a univariate polynomial of degree d and its polar form, or, in Ramshaws terminology a blossom. A blossom is a multi-affine, symmetric form in d variables. A map $f(t) : R^1 \rightarrow R^n$ is affine if it preserves affine combinations of its arguments: i.e. it satisfies $f(\sum_i \alpha_i t_i) = \sum_i \alpha_i f(t_i)$ for a set of real

scalar values $\{\alpha_i : 0 \leq i < m\}$ such that $\sum_i \alpha_i = 1$. A map $f : R^n \rightarrow R^1$, $f = f(t_0, \ldots, t_{n-1})$ is said to be multi-affine if it is affine in each of its variables when the others are kept fixed: Let all $\{t_0, \ldots, t_{n-1}\} \setminus t_j$, $0 \leq j < n$ have fixed values. Then the multi-affinity means that

$$f\left(t_0, \ldots, \sum_i \alpha_i t_{j_i}, \ldots t_{n-1}\right) = \sum_i \alpha_i f(t_0, \ldots, t_{j_i}, \ldots t_{n-1})$$

with $\sum_i \alpha_i = 1$. A map $f : R^n \rightarrow R^1$ is said to be symmetric if it does not depend on the ordering of variables $t_0, \ldots t_{n-1}$. Based on these properties of symmetric multi-affine maps Ramshaw has stated the so-called *Blossoming principle* for polynomials see [65, 64, 66]:

Associated with each polynomial $p : R^1 \rightarrow R^1$, $p = p(t)$, $p(t) \in \mathcal{P}_d$ there is a symmetric multi-affine map $f : R^d \rightarrow R^1$, $f = f(t_0, \ldots, t_{d-1})$ called a polar form, or blossom of p with following properties:

1. Identity between the polynomial and the *diagonal* of the blossom:

$$\text{if} \quad t_i = t,\ 0 \leq i < d \quad \text{then} \quad f(t_0, \ldots, t_{d-1}) = p(t)$$

2. **Symmetry**: the value of the blossom is invariant under permutations of its arguments.

3. **Multi-affinity**: f is affine in each of its arguments, or simply multi-affine

The blossoming principle applies for general vector-valued polynomial maps $P : R^1 \rightarrow R^D$, $P = P(t)$ where D denotes the dimension of the image space. One simply considers each of the D polynomials separately. Thus it is possible to blossom polynomial curves in D-dimensional space in the same manner as polynomial functions. In order to simplify notation the following paragraphs demonstrate everything on univariate one-dimensional Bezier and B-Spline functions without loss of generality.

4.1.2 Blossoming principle for Bezier polynomials

Blossoming is very useful when dealing with polynomials expressed in terms of Bezier or B-Spline basis. Consider a degree d Bezier function $c(t) \in \mathcal{P}_d$, $c(t) = \sum_{i=0}^{d-1} c_i b_i^d(t)$ defined on interval $t \in \langle a, b \rangle$. By property 1 of blossoms $c(t)$ corresponds to the value of the blossom at its diagonal. Writing t as an affine combination of a an b, $t = \frac{b-t}{b-a}a + \frac{t-a}{b-a}b$ and using the property 3 of blossoms we

express, for example, the last argument of the blossom as an affine combination of a and b:

$$\begin{aligned} c(t) &= f(\underbrace{t,\ldots,t}_{d\times}) \\ &= f(\underbrace{t,\cdots,t}_{(d-1)\times},\tfrac{b-t}{b-a}a+\tfrac{t-a}{b-a}b) \\ &= \tfrac{b-t}{b-a}f(\underbrace{t,\ldots,t}_{(d-1)\times},a)+\tfrac{t-a}{b-a}f(\underbrace{t,\ldots,t}_{(d-1)\times},b) \end{aligned} \tag{4.1}$$

This can be done recursively for all remaining blossom arguments yielding

$$\begin{aligned} c(t) &= \textstyle\sum_{i=0}^{d-1}\binom{d+1}{i}\left(\frac{b-t}{b-a}\right)^{d+1-i}\left(\frac{t-a}{b-a}\right)^{i} f(\underbrace{a,\ldots,a}_{(d-i)\times},\underbrace{b,\ldots,b}_{i\times}) \\ &= \textstyle\sum_{i=0}^{d-1} b_i^d(t)\, f(\underbrace{a,\ldots,a}_{(d-i)\times},\underbrace{b,\ldots,b}_{i\times}) \end{aligned}$$

It follows immediately that $c_i = f(\underbrace{a,\ldots,a}_{(d-i)\times},\underbrace{b,\ldots,b}_{i\times})$. Thus, each control point has a blossom formulation.

4.1.2.1 Affine and multi-affine de Casteljau algorithm

It is possible to approach formula 4.1 in the opposite direction and obtain a blossom evaluation algorithm from given control points. Suppose that the control points $\{c_i : 0 \le i \le d\}$ are given as blossoms. An affine combination of two consecutive control points c_i and c_{i+1} yields

$$\begin{aligned} &\tfrac{b-t}{b-a}\,c_i+\tfrac{t-a}{b-a}\,c_{i+1} = \\ &\quad \tfrac{b-t}{b-a}f(\underbrace{a,\ldots,a}_{(d-i)\times},\underbrace{b,\ldots,b}_{i\times})+\tfrac{t-a}{b-a}f(\underbrace{a,\ldots,a}_{(d-i-1)\times},\underbrace{b,\ldots,b}_{(i+1)\times}) = \\ &\qquad f(\underbrace{a,\ldots,a}_{(d-i-1)\times},\underbrace{t}_{1\times},\underbrace{b,\ldots,b}_{i\times}) \end{aligned}$$

Building affine combinations of all d consecutive control points yields a set of $d-1$ points $\{c_i^1 = f(\underbrace{a,\ldots,a}_{(d-i-1)\times},\underbrace{t}_{1\times},\underbrace{b,\ldots,b}_{i\times}) : 0 \le i \le d-1\}$. Repeating this procedure with the intermediate points $\{c_i^1 : 0 \le i \le d-1\}$ yields $d-2$ points $\{c_i^2 = f(\underbrace{a,\ldots,a}_{(d-i-2)\times},\underbrace{t}_{2\times},\underbrace{b,\ldots,b}_{i\times}) : 0 \le i \le d-2\}$ and finally, after the recurrence was executed d-times, one obtains a point $\{c_i^d = f(\underbrace{t,\ldots,t}_{d\times}) : i = 0\}$ such that $c(t) = c_0^d$. This leads to a recurrence formula which provides an efficient way to

evaluate a blossom (and thus the value of a Bezier function) for given t by repeated affine interpolation between two consecutive control points:

$$c_i^j = \frac{b-t}{b-a}\, c_i^{j-1} + \frac{t-a}{b-a}\, c_{i+1}^{j-1} \tag{4.2}$$

or, equivalently, in blossom format

$$\begin{array}{l} f(\underbrace{a,\ldots,a}_{(d-i-j)\times},\underbrace{t,\cdots,t}_{j\times},\underbrace{b,\ldots,b}_{i\times}) = \\ \frac{b-t}{b-a}\, f(\underbrace{a,\ldots,a}_{(d-(i+1)-j)\times},\underbrace{t,\cdots,t}_{(j-1)\times},\underbrace{b,\ldots,b}_{i\times}) + \\ \frac{t-a}{b-a}\, f(\underbrace{a,\ldots,a}_{(d-i-j)\times},\underbrace{t,\cdots,t}_{(j-1)\times},\underbrace{b,\ldots,b}_{(i+1)\times}) \end{array} \tag{4.3}$$

This recurrence is known as *de Casteljau algorithm*, investigated by de Casteljau in [15] and [16]. The Blossoming principle provides a mathematical method to access the intermediate points of the de Casteljau scheme. Moreover, a slight modification of eq. 4.3 can be used to compute arbitrary blossoms $f(t_0,\cdots,t_{d-1})$:

$$\begin{array}{l} f(\underbrace{a,\ldots,a}_{(d-i-j)\times},\underbrace{t_0,\cdots,t_j}_{j\times},\underbrace{b,\ldots,b}_{i\times}) = \\ \frac{b-t_j}{b-a}\, f(\underbrace{a,\ldots,a}_{(d-(i+1)-j)\times},\underbrace{t_0,\cdots,t_{j-1}}_{(j-1)\times},\underbrace{b,\ldots,b}_{i\times}) + \\ \frac{t_j-a}{b-a}\, f(\underbrace{a,\ldots,a}_{(d-i-j)\times},\underbrace{t_0,\cdots,t_{j-1}}_{(j-1)\times},\underbrace{b,\ldots,b}_{(i+1)\times}) \end{array} \tag{4.4}$$

This is the so-called multi-affine version of the de Casteljau algorithm. One computes an affine combination of consecutive blossoms with respect to the current argument of the blossom. It can be easily verified by inspection that the blossom $f(t_0,\cdots,t_{d-1})$ is symmetric: the value $f(t_0,\cdots,t_{d-1})$ is the same for each permutation of arguments $t_0,\cdots,t_{d-1}$. In contrast to the affine version, after d steps of the multi-affine de Casteljau algorithm one obtains a point $c_0^d = f(t_0,\cdots,t_{d-1})$ which is not incident on $c(t)$. These points are of interest in theoretical considerations, as will be demonstrated in following paragraphs.

4.1.3 Blossoming B-Splines

The blossoming principle for B-Splines is slightly more complicated. Consider a B-Spline function $c(t) \in \mathcal{P}_{d,\tau}$, $c(t) = \sum_{i=0}^{n-1} c_i b_i^{d,\tau}(t)$ defined on knot vector $\tau_i : 0 \le i \le n+d$. By properties 1-2 of B-Splines, see §A.1.1.3, $c(t)$ consists of polynomial segments defined at intervals $\tau_i \le t < \tau_{i+1}$, $d \le i < n$. Therefore,

in order to compute a blossom of a B-Spline we need to select an interval of the knot vector τ. In the following we will denote a B-Spline blossom for $\langle \tau_i, \tau_{i+1} \rangle$ by $f_i(t_0, \ldots, t_{d-1})$. At that interval $c(t)$ is a polynomial, hence, blossom properties 1-3 apply to $f_i(t_0, \ldots, t_{d-1})$.

It can be shown by a similar reasoning as for Bezier functions in previous paragraph that each B-Spline control point c_i has a blossom formulation. Consider blossom $f_i(\underbrace{t, \ldots, t}_{d\times})$. First, we write t as an affine combination of τ_i and τ_{i+1}

$$t = \frac{\tau_{i+1} - t}{\tau_{i+1} - \tau_i} \tau_i + \frac{t - \tau_i}{\tau_{i+1} - \tau_i} \tau_{i+1}$$

Second, we expand the last argument of the blossom as follows:

$$\begin{aligned} c(t) &= f_i(\underbrace{t, \ldots, t}_{d\times}) \\ &= f_i(\underbrace{t, \cdots, t}_{(d-1)\times}, \tfrac{\tau_{i+1}-t}{\tau_{i+1}-\tau_i} \tau_i + \tfrac{t-\tau_i}{\tau_{i+1}-\tau_i} \tau_{i+1}) \\ &= \tfrac{\tau_{i+1}-t}{\tau_{i+1}-\tau_i} f_i(\underbrace{t, \ldots, t}_{(d-1)\times}, \tau_i) + \tfrac{t-\tau_i}{\tau_{i+1}-\tau_i} f_i(\underbrace{t, \ldots, t}_{(d-1)\times}, \tau_{i+1}) \end{aligned}$$

In the second step of the expansion one considers the left and right neighboring spans of the knot vector and expands the left and right terms of the above affine combination with respect to intervals $\langle \tau_{i-1}, \tau_{i+1} \rangle$ and $\langle \tau_i, \tau_{i+2} \rangle$ yielding

$$\begin{aligned} f_i(\underbrace{t, \ldots, t}_{(d-1)\times}, \tau_i) &= f_i(\underbrace{t, \cdots, t}_{(d-2)\times}, \tfrac{\tau_i-t}{\tau_i-\tau_{i-1}} \tau_{i-1} + \tfrac{t-\tau_{i-1}}{\tau_i-\tau_{i-1}} \tau_i) \\ &= \tfrac{\tau_i-t}{\tau_i-\tau_{i-1}} f_i(\underbrace{t, \ldots, t}_{(d-2)\times}, \tau_{i-1}, \tau_i) + \\ &\quad + \tfrac{t-\tau_{i-1}}{\tau_i-\tau_{i-1}} f_i(\underbrace{t, \ldots, t}_{(d-2)\times}, \tau_i, \tau_{i+1}) \end{aligned}$$

and

$$\begin{aligned} f_i(\underbrace{t, \ldots, t}_{(d-1)\times}, \tau_{i+1}) &= f_i(\underbrace{t, \cdots, t}_{(d-2)\times}, \tfrac{\tau_{i+2}-t}{\tau_{i+2}-\tau_{i+1}} \tau_{i+1} + \tfrac{t-\tau_{i+1}}{\tau_{i+2}-\tau_{i+1}} \tau_{i+2}) \\ &= \tfrac{\tau_{i+2}-t}{\tau_{i+2}-\tau_{i+1}} f_i(\underbrace{t, \ldots, t}_{(d-2)\times}, \tau_{i+1}, \tau_i) + \\ &\quad + \tfrac{t-\tau_{i+1}}{\tau_{i+2}-\tau_{i+1}} f_i(\underbrace{t, \ldots, t}_{(d-2)\times}, \tau_{i+2}, \tau_{i+1}) \end{aligned}$$

Continuing this expansion for all remaining blossom arguments one obtains a triangular scheme first three stages of which look as follows:

$$
\begin{array}{ccccc}
 & & f_i(\underbrace{t,\ldots,t}_{d\times}) & & \\
 & f_i(\underbrace{t,\ldots,t}_{(d-1)\times},\tau_i) & & f_i(\underbrace{t,\ldots,t}_{(d-1)\times},\tau_{i+1}) & \\
f_i(\underbrace{t,\ldots,t}_{(d-2)\times},\tau_{i-1},\tau_i) & & f_i(\underbrace{t,\ldots,t}_{(d-2)\times},\tau_i,\tau_{i+1}) & & f_i(\underbrace{t,\ldots,t}_{(d-2)\times},\tau_{i+1},\tau_{i+2}) \\
 & & \vdots & &
\end{array}
\tag{4.5}
$$

On the way from top to the bottom of the triangle one multiplies the left lower term by $\frac{\tau_{i+j}-t}{\tau_{i+j}-\tau_i}$ and the right lower term by $\frac{t-\tau_i}{\tau_{i+j}-\tau_i}$ where j denotes the current "stage" of the triangle counted from 1. The bottom line reached after d expansions contains exactly $d+1$ blossoms with arguments consisting entirely of entries of the knot vector. These blossoms have the format

$$\{f_i(\tau_k,\cdots,\tau_{k+d-1}) : i-d \le k \le i\}$$

It can be shown by collecting and multiplying the terms $\frac{\tau_{i+j}-t}{\tau_{i+j}-\tau_i}$ and $\frac{t-\tau_i}{\tau_{i+j}-\tau_i}$ for $1 \le j \le d$ that

$$f_i(\underbrace{t,\ldots,t}_{d\times}) = \textstyle\sum_{k=i-d}^{i} b_k^{d,\tau} f_i(\tau_k,\cdots,\tau_{k+d-1}) \tag{4.6}$$

see e.g. [65] or [76]. I.e., in front of the blossom the recurrence for B-Spline basis functions is recovered. It follows that the blossom representation for control points of the B-Spline segment $c(t)$, $\tau_i \le t < \tau_{i+1}$ is $c_{i-d} = f_i(\tau_{i-d+1},\cdots,\tau_i)$. These points are called *de Boor* points due to Carl de Boor who discovered this expansion. Though there are different approaches to derive the B-Spline basis functions, see e.g. [68], we see that using blossoms and their properties this can be accomplished easily. In [65] Ramshaw uses the Blossoming principle to derive properties of the B-spline basis listed in §A.1.1.3.

4.1.3.1 Affine and multi-affine version of the de Boor algorithm

The procedure sketched above can be approached from the opposite direction which leads to an evaluation algorithm for B-Splines: Suppose the de Boor points are given in blossom-format. Given a parameter $t = const. = \mu$ such that $\tau_i \le \mu < \tau_{i+1}$ one needs to go up the triangle building the affine combination of intermediate points w.r.t μ. This yields a set of $d+1-j$ intermediate points $\{c_k^j : i-d+j \le k \le i\}$ at j-th stage. At the top of the triangle a point c_i^d is computed which satisfies $c_i^d = c(\mu)$. This is the well known affine version of *de Boor* algorithm for evaluating B-Splines [65, 41, 76]. In the following we will use a

generalized multi-affine version of the de Boor algorithm. For that we change the notation as follows:

Let be given a set of B-spline basis functions $\{b_i^{k,\tau} : 0 \leq i < n\}$ and de Boor points $\{c_i^0 : 0 \leq i < n\}$. We wish to compute the value of a blossom $f_I(t_0, \ldots, t_{d-1})$ for the interval $\langle \tau_I, \tau_{I+1})$ such that $\tau_I < \tau_{I+1}$. The intermediate de Boor points at j-th stage are computed by the recursion

$$\begin{gathered} c_i^j = \Big(1 - \alpha_i^j(t_j)\Big)\, c_{i-1}^{j-1} + \alpha_i^j(t_j)\, c_i^{j-1},\ I - d + j < i \leq I \\ \text{with}\ \ \alpha_i^j(t_j) = \tfrac{t_j - \tau_i}{\tau_{i+j} - \tau_i} \end{gathered} \tag{4.7}$$

The recurrence is executed for $1 \leq j \leq d$ obtaining a point c_I^d after d steps that corresponds to the blossom $f_I(t_0, \ldots, t_{d-1})$. The multi-affine de Boor algorithm is also "triangular", each intermediate point is the the linear combination w.r.t. current blossom argument of its two predecessors on the left:

$$\begin{array}{llll} c_{I-d+1}^0 & & & \\ \vdots & c_{I-d}^1 & & \\ & \vdots & \ddots & \\ c_I^0 & c_I^1 & \cdots & c_I^d = f_I(t_0, \ldots, t_{d-1}) \end{array} \tag{4.8}$$

The multi-affine de Boor algorithm generalizes the multi-affine version of de Casteljau algorithm in previous paragraph. It is possible to write the multi-affine de Boor recurrence in blossom format see e.g. [76] – however, the notation which uses intermediate de Boor points $c_i^j : 0 \leq j \leq d$ is easier to handle in practical applications.

4.1.4 Blossoming tensor-product B-Splines

Let be defined a two-variate B-spline function

$$s : \Omega_{u,v} \to R^1,\ s = s(u, v),\ s(u, v) \in \mathcal{P}_{k,\tau} \times \mathcal{P}_{l,v}$$

with $(u, v) \in \Omega_{u,v} \subseteq R^2$, $\Omega_{u,v} = \Omega_u \times \Omega_v$. In order to establish the blossoming principle for tensor-products of B-Splines one has to apply the univariate blossoming principle to each parametric direction separately [65, 64, 66]. The resulting blossom of a tensor-product is a map

$$f_{I,J} : R^k \times R^l \to R^1,\ f_{I,J} = f_{I,J}(u_0, \ldots, u_{k-1}; v_0, \ldots, v_{l-1})$$

where I and J denote the intervals $\langle \tau_I, \tau_{I+1})$ and $\langle v_J, v_{I+1})$ selected for blossoming $s(u, v)$ in each parametric direction. The properties of blossoms of univariate polynomials apply to each parametric direction separately:

1. Identity between s and the diagonal of the blossom:

$$\begin{array}{ll} \text{if} & u_i = u,\ 0 \leq i < k \text{ and } v_j = v,\ 0 \leq i < l \\ \text{then} & f_{I,J}(u_0, \ldots, u_{k-1}; v_0, \ldots, v_{l-1}) = s(u, v) \end{array}$$

2. **Symmetry**: the blossom $f_{I,J}$ is symmetric in variables $u_i,\ 0 \leq i < k$ and $v_j,\ 0 \leq j < l$ separately

3. **Multi-affinity**: $f_{I,J}$ is affine in each of its arguments $u_i,\ 0 \leq i < k$ and $v_j,\ 0 \leq j < l$ separately

Based on these properties one can derive the blossom representation for de Boor points of a tensor-product B-Spline and de Boor evaluation algorithm for tensor-products of B-Splines, see [76], for example.

4.1.5 Unevaluated formulation of a blossom

It is easy to compute the value a blossom, once the connection between blossoming and de Boor algorithm is established. Two famous algorithms for B-splines can be formulated in terms of blossoms: the knot insertion algorithm (the "Oslo" or "Böhm"-algorithm, see [74]) and degree raising, as Seidel has shown in [75]. Another operations frequently needed in free-form modeling can be expressed in terms of the Oslo-algorithm and thus, in terms of blossoming, for example: evaluation of the function value or its derivatives (this follows from Property 1), splitting of a curve or surface at a particular parameter value, extraction of a specified interval from curve or surface, and conversion between B-spline and Bezier bases. In [50] Liu has pursued an idea of writing a software library for CAGD based solely on blossoming. In the following, we generalize this approach by introducing unevaluated blossoms.

4.1.5.1 The polarized basis functions

The multi-affine version of the de Boor algorithm delivers a value of a blossom by recursively computing affine combinations of de Boor control points. For that the values of the control points must be known. In the following we assume that the structure of the B-spline vector space (i.e. knot vector and degree) are known but the values of the control points are not. Analogously to the definition of B-spline basis functions (eq. A.1) we seek a formula which delivers the multi-affine, or, "polarized" basis functions such that the value of the blossom is computed the same way as the value of a B-spline function for any control point values.

The value of a blossom of a B-spline $c(t) \in \mathcal{P}_{d,\tau},\ c(t) = \sum_{i=0}^{n-1} c_i b_i^{d,\tau}(t)$ is:

$$f_I(t_0,\ldots,t_{d-1}) = \sum_{i=I-d}^{I} a_i^{d,\tau}(t_0,\ldots,t_{d-1})c_i$$

where

$$\begin{aligned} a_i^{d,\tau}(t_0,\ldots,t_{d-1}) = \\ \left(1-\alpha_{i+1}^{d-1}(t_{d-1})\right) a_{i+1}^{d-1,\tau}(t_0,\ldots,t_{d-2}) + \\ \alpha_i^{d-1}(t_{d-1})\, a_i^{d-1,\tau}(t_0,\ldots,t_{d-2}) \end{aligned} \tag{4.9}$$

with

$$a_i^{0,\tau}(t_0) = \begin{cases} 1 & \text{if } i = I \\ 0 & \text{otherwise} \end{cases}$$

The functions $\alpha_i^j(t_j)$ are defined as in eq. 4.7. We call the set

$$\{a_i^{d,\tau}(t_0,\ldots,t_{d-1}) : I-d \le i \le I\}$$

the *polarized basis functions* of $\mathcal{P}_{d,\tau}$. Equation 4.9 results from collecting and multiplying the $\alpha_i^j(t_j)$ and $1-\alpha_i^j(t_j)$ terms at each stage of the de Boor algorithm (eq. 4.7).

In order to compute all non-zero values $a_i^{d,\tau}(t_0,\ldots,t_{d-1})$ efficiently for given I and $t_0,\ldots,t_{d-1}$ consider the following triangular scheme:

$$\begin{array}{lll} a_{I-d}^d(t_0,\ldots,t_{d-1}) & & \\ & \ddots & \\ \vdots & & a_{I-1}^{d-1}(t_0,t_1) \\ a_I^d(t_0,\ldots,t_{d-1}) & \cdots & a_I^{d-1}(t_0,t_1) \quad a_I^0(t_0)=1 \end{array} \tag{4.10}$$

This time the recursion starts on the right lower vertex ($a_I^0(t_0)=1$) of the triangle. We compute affine combination with respect to the current argument of the blossom moving from right to left until after d steps all $d+1$ non-zero coefficients are computed. In each column we move from bottom to top reusing the values computed in previous steps. The computational costs of the algorithm are not higher than the costs of the de Boor algorithm: There are totally d stages. At j-th stage j linear combinations need to be computed. Hence the cost of obtaining all non-zero $a_i^{d,\tau}$ is $O(d^2)$ linear combinations.

4.1.5.2 Properties

The so obtained polarized basis functions posses the following properties:

1. if $t_i = t$, $0 \leq i < d$ we have $a_i^{d,\tau}(t,\ldots,t) = b_i^{d,\tau}(t)$ and $c(t) = f_I(t,\ldots,t)$. This follows from the diagonal property of the blossom.
2. $\sum_{i=0}^{n-1} a_i^{d,\tau}(t_0,\ldots,t_{d-1}) = 1$
3. $a_i^{d,\tau}(t_0,\ldots,t_{d-1}) \begin{cases} \neq 0; \text{ if } I-d \leq i \leq I \\ = 0 \qquad \text{otherwise} \end{cases}$
4. $a_i^{d,\tau}(t_0,\ldots,t_{d-1}) \geq 0$ iff $\tau_I \leq t_j < \tau_{I+1}$ for $0 \leq j < d$

The property 4 contains an important difference to the properties of B-spline basis functions, (see §A.1.1.3): We are free to select blossom arguments which are spread over several intervals of the knot vector. Then, although property 2 still applies, not all $a_i^{d,\tau}(t_0,\ldots,t_{d-1}) : I-d \leq i \leq I$ are positive: Hence, the blossom value is not necessarily a convex[1] combination of the de Boor points. Thus the value of a blossom depends on the interval selected for blossoming. It is easy to see that $f_I(t_0,\ldots,t_{d-1}) \neq f_J(t_0,\ldots,t_{d-1})$ if $I \neq J$. Consequently if not all arguments stem from the same interval of the knot vector and the interval is not given explicitely the value of a blossom is not uniquely defined; In order to circumvent that Ramshaw has defined conventions how to determine a valid interval for blossoming given the blossom arguments the, so-called, argument overloading [65, 64, 66]. However, as we will see in section 4.2, in our application (composition of two B-splines) the intervals follow from the structure of the problem and cannot be selected by the overloading conventions.

4.1.5.3 Scalar product notation: $\mathbf{a}$-vectors

In order to simplify notation, we introduce a vector notation for polarized basis functions in the same manner as for regular basis functions in section A.1. We denote the set of polarized basis functions by a vector

$$\mathbf{a}_t^T = \left[\begin{array}{ccc} a_0^{d,\tau}(t_0,\ldots,t_{d-1}) & \cdots & a_{n-1}^{d,\tau}(t_0,\ldots,t_{d-1}) \end{array}\right]$$

and write a blossom of a B-spline $c(t)$ as a scalar product:

$$\begin{aligned} f_I(t_0,\ldots,t_{d-1}) &= \left[\begin{array}{ccc} a_0^{d,\tau}(t_0,\ldots,t_{d-1}) & \cdots & a_{n-1}^{d,\tau}(t_0,\ldots,t_{d-1}) \end{array}\right] \left[\begin{array}{c} c_0 \\ \vdots \\ c_{n-1} \end{array}\right] \\ &= \mathbf{a}_t^T \mathbf{c} \end{aligned}$$

[1] A "convex" combination is a special case of affine combination such that for some real values γ_i and $\sum_{i=0}^{n-1} \gamma_i c_i$ we have $\sum_{i=0}^{n-1} \gamma_i = 1 \wedge \{\gamma_i \geq 0 : 0 \leq i < n\}$. Then the result lies inside the convex hull spanned by the points $c_i : 0 \leq i < n$.

For simplicity, such vectors containing the values of polarized basis functions are called $\mathbf{a}$-vectors.

The notation for two-variate polarized basis functions is the same as for two-variate B-spline basis functions: Given a two-variate tensor product of B-spline spaces $s(u,v) \in \mathcal{P}_{k,\tau} \times \mathcal{P}_{l,\upsilon}$, $\sum_{j=0}^{n-1}\sum_{i=0}^{m-1} s_{i,j} b_i^{k,\tau}(u) b_j^{l,\upsilon}(v)$ we introduce two-variate polarized basis functions

$$a_K^{k,\tau,l,\nu}(u_0,\ldots,u_{k-1};v_0,\ldots,v_{l-1}) = a_i^{k,\tau}(u_0,\ldots,u_{k-1})\, a_j^{l,\upsilon}(v_0,\ldots,v_{l-1})$$

with integer indices K such that

$$K = i + mj,\ 0 \le i < m \wedge 0 \le j < n \Rightarrow 0 \le K \le (m-1)(n-1)$$

In full analogy to §A.1.1.4 we define a $(m-1)(n-1)$-size vector of polarized basis functions denoted by $\mathbf{a}_{u,v}^T$ such that K-th entry of the vector is

$$\mathbf{a}_K = a_K^{k,\tau,l,\nu}(u_0,\ldots,u_{k-1};v_0,\ldots,v_{l-1})$$

We expand the matrix of tensor-product control points in row major order as in §A.1.1.4 yielding

$$\begin{aligned} f_{I,J}(u_0,\ldots,u_{k-1};v_0,\ldots,v_{l-1}) &= \sum_{i=I}^{I-k}\sum_{j=J}^{J-l} s_K a_K^{k,\tau,l,\upsilon}(u_0,\ldots,u_{k-1};v_0,\ldots,v_{l-1}) \\ &= \mathbf{a}_{u,v}^T\mathbf{s} \end{aligned}$$

with $K = i + mj$.

4.1.5.4 Summary

In summary, we note: there is the same scalar product notation for a blossom of a B-spline function (curve, surface) as we have already defined for values of functions (curves, surfaces) in section A. A scalar product of the vector of polarized basis functions multiplied with the control points of the B-spline yields the value of the blossom. This is called the *unevaluated* formulation of a blossom expressed in terms of the $\mathbf{a}$-vector.

4.2 Unevaluated polynomial composition

In the 1988 paper [17], DeRose has introduced a new approach to polynomial composition of Bezier simplices of arbitrary dimension based on blossoming. The continuation work published in 1993 [18] extends the algorithm to rational functions and tensor products. The complexity of the algorithm was analyzed in [53],

where a modified version with optimal run-time complexity was described. We want to generalize this algorithm in two ways: First, we introduce unevaluated version of polynomial composition algorithm for both Bezier and B-splines. The term "unevaluated" was chosen because the algorithm delivers the so-called composition matrix which transforms the control points of the outer function to control points of the result. Second, we show how to obtain the result in irreducible B-spline form when both, inner and outer functions, are B-Splines (curve and surface, with restrictions from section 3.2). We will first derive the algorithm for computing the composition matrix for Bezier surface-curve case since it is more straight-forward and better suited to demonstrate the concept. We first consider the straight-forward algorithm and show how to reduce the complexity later on.

For sake of generality, in [18] and [53] a new versatile notation for indices, basis functions and control points was used. Since we only intend to cover the curve-surface case we use our own notation which is simpler. However, we will not get around "multi-indices" used in [18]: they improve the readability of formulas and are well suited to derive the implementation of the algorithms. The original hyper-indices and multi-indices are not very advantageous when the inner and outer functions are B-splines. Therefore we use a modified, more comprehensive, version of a multi-indices, the so-called, v-index and c-index which will be introduced below.

4.2.1 Revisiting the DeRose et al. algorithm

We start with a brief review of De Rose's algorithm for Bezier surface and Bezier curve. Let $F(u, v)$ be a tensor product Bezier patch of polynomial degree k and l in each parametric direction. Furthermore let $G(t)$ be a degree d Bezier curve in the domain of $F(u, v)$. The starting point of the algorithm is to represent the outer function F in blossom notation and to substitute the inner function G into the blossom expression. Split $G(t)$ into its coordinate polynomials:

$$G(t) = (u(t), v(t))^T = \left(\sum_i^d u_i b_i^d(t), \sum_i^d v_i b_i^d(t) \right)^T$$

Here (u_i, v_i) denotes the coordinates of the i-th control point of the curve $G(t)$. Using diagonality of blossoms (property 1, §4.1.1) the components $u(t)$ and $v(t)$ are inserted into the surface blossom yielding

$$H(t) = F\left(u(t), v(t)\right) = f(\underbrace{u(t), \ldots, u(t)}_{k}; \underbrace{v(t), \ldots, v(t)}_{l}) \tag{4.11}$$

Since $\sum_i^d b_i^d(t) = 1$, by blossom property 3 we may expand the above blossom into an affine combination of blossoms

$$H(t) = b_0^d(t) f(u_0, \underbrace{u(t), \ldots, u(t)}_{k-1}; \underbrace{v(t), \ldots, v(t)}_{l}) + \ldots$$
$$\ldots + b_d^d(t) f(u_d, \underbrace{u(t), \ldots, u(t)}_{k-1}; \underbrace{v(t), \ldots, v(t)}_{l})$$

This expansion is done for all remaining blossom arguments in both parametric directions of F until the argument bags consist only of coordinates of control points of G yielding

$$H(t) = \sum_{i_0}^{d} \ldots \sum_{j_{k+l}}^{d} b_{i_0}^d(t) \cdots b_{j_{k+l}}^d(t)\, f(u_{i_0}, \ldots, u_{i_k}; v_{j_{k+1}}, \ldots, v_{j_{k+l}})$$

The 1st term in the above formula is a $k+l$ fold product of Bernstein polynomials, result of which is a Bernstein polynomial of degree $d(k+l)$ times an rational factor (see [17], for example). The coefficients u_i and v_i are constant, hence, the second term reduces to a 3D point obtained by applying the generalized de Boor algorithm directly to control points of F. The control points $\{H_i : 0 \leq i < d(k+l)\}$ of the curve $H(t)$ are obtained by collecting the compatible terms on both sides of the equality

$$\sum_{i=0}^{d(k+l)} H_i b_i^{d(k+l)} = \sum_{i_0}^{d} \ldots \sum_{j_{k+l}}^{d} b_{i_0}^d \cdots b_{j_{k+l}}^d\, f(u_{i_0}, \ldots, u_{i_k}; v_{j_{k+1}}, \ldots, v_{j_{k+l}}) \tag{4.12}$$

4.2.2 Computing the Bezier composition matrix

The generalization to unevaluated form is achieved by using the polarized basis functions introduced in §4.1.5 and rewriting the equation 4.12 with regard to this new notation. How this is accomplished is shown in the next paragraphs.

4.2.2.1 The v-index

First, we introduce a new notation for tuples of integer indices $i_0 \ldots j_{k+l}$ from eq. 4.12. Consider an ordered set $P = \{0, \ldots, d\}$. As the summation in formula 4.12 goes on the subscripts $i_0, \ldots, i_k$ and $j_{k+1}, \ldots, j_{k+l}$ take in values which are permutations of entries from P. More specifically, the tuples $(i_0, \ldots, i_k)$ and $(j_{k+1}, \ldots, j_{k+l})$ are variations with repetition of size k and l from the set P. The notation for a set of all variations with repetition of size k from $d+1$ elements is $V'(d+1, k)$. An element from $V'(d+1, k)$ is a k-tuple of integers $(i_0, \ldots, i_k)$

such that $i_n \in P : 0 \leq n \leq k$. We will refer to such tuples as "*v-index*" and denote them by $\vec{\imath} \in V'(d+1,k)$ or simply $\vec{\imath}_{(d+1,k)}$. We define following operations on v-indices:

1. *Concatenation* of several v-indices is denoted by $\vec{\imath} \mid \vec{\jmath}$. If $\vec{\imath} \in V'(d,k)$ and $\vec{\jmath} \in V'(d,l)$ then $\vec{\imath} \mid \vec{\jmath} \in V'(d,k+l)$.

2. "*Absolute value*" $|\vec{\imath}|$ is defined as $|\vec{\imath}| = i_0 + \ldots + i_{k-1}$.

In order to simplify notation we write a nested iteration as in eq. 4.12 directly in terms of v-indices: for example, $\sum_{i_1}^{d} \ldots \sum_{i_k}^{d}$ becomes $\sum_{\vec{\imath} \in V'(d,k)}$.

4.2.2.2 The composition matrix

The first term behind the summation from eq. 4.12 is a $(k+l)$ fold product of Bernstein polynomials. I.e. we have

$$b_{i_0}^d(t) \cdots b_{j_{k+l}}^d(t) = \sum_{i=0}^{d(k+l)} b_i^{d(k+l)}(t) p_i^{i_0,\ldots,j_{k+l}}$$

where $p_i^{i_0,\ldots,j_{k+l}}$ is the coefficient of the product of Bernstein polynomials, see [41, §4.1, Eq. 4.2d], for example. The coefficients p_i depend on the indices $i_0, \ldots, j_{k+l}$. Denote the coefficients $p_i^{i_0,\ldots,j_{k+l}} : 0 \leq i \leq d(k+l)$ of this product of Bernstein polynomials with indices $i_0, \ldots, j_{k+l} = \vec{\imath}_{(d,k)} | \vec{\jmath}_{(d,l)}$ by a vector

$$\mathbf{p}_{\vec{\imath}_{(d,k)} | \vec{\jmath}_{(d,l)}} = \left[\begin{array}{ccc} p_0^{i_0,\ldots,j_{k+l}} & \cdots & p_{d(k+l)}^{i_0,\ldots,j_{k+l}} \end{array} \right]^T$$

Note that according to product formula for Bernstein polynomials only the entry $p_{|\vec{\imath}_{(d,k)} | \vec{\jmath}_{(d,l)}|}$ is non-zero. Now denote the degree $d(k+l)$ Bezier basis of the product by

$$\mathbf{b}_\pi = \left[\begin{array}{ccc} b_0^{d(k+l)}(t) & \cdots & b_{d(k+l)}^{d(k+l)}(t) \end{array} \right]^T$$

and write the product in matrix format:

$$b_{i_0}^d(t) \cdots b_{j_{k+l}}^d(t) = \mathbf{b}_\pi^T \, \mathbf{p}_{\vec{\imath}_{(d,k)} | \vec{\jmath}_{(d,l)}}$$

Analogously, we write the blossom term from eq. 4.12 in scalar product notation introducing a vector of generalized two-variate basis functions such that the blossom arguments are the variations of control points (u_i, v_i) according to current v-indices. The vector is denoted by $\mathbf{a}_{\vec{\imath}_{(d,k)}; \vec{\jmath}_{(d,l)}}^T$ so that the i-th entry of $\mathbf{a}$ is

$$\mathbf{a}_i = a_i^{k,l}(u_{i_0}, \ldots, u_{i_k}; v_{j_{k+1}}, \ldots, v_{j_{k+l}})$$

This yields

$$f\left(u_{i_0},\ldots,u_{i_k};v_{j_{k+1}},\ldots,v_{j_{k+l}}\right)=\mathbf{a}^T_{\vec{i}_{(d,k)};\vec{j}_{(d,l)}}\mathbf{F}$$

Rewriting eq. 4.12 in terms of these new conventions we get

$$\mathbf{b}^T_\pi\cdot\mathbf{H}=\sum_{\substack{\forall\vec{i}\in V'(d,k)\\ \forall\vec{j}\in V'(d,l)}}\mathbf{b}^T_\pi\underbrace{\mathbf{p}_{\vec{i}|\vec{j}}\mathbf{a}^T_{\vec{i};\vec{j}}}_{\mathbf{A}_{\vec{i}|\vec{j}}}\mathbf{F} \tag{4.13}$$

The under braced term in Eq. 4.13 is an outer product of two vectors with dimensions $d(k+l)\times 1$ and $kl\times 1$. The result of this operation is a matrix with size $d(k+l)\times kl$, denoted by $\mathbf{A}_{\vec{i}|\vec{j}}$. The basis of the product $\mathbf{b}_\pi$ does not change during the iteration, hence, we may take it in front of the summation obtaining

$$\mathbf{b}^T_\pi\mathbf{H}=\mathbf{b}^T_\pi\sum_{\substack{\forall\vec{i}\in V'(d,k)\\ \forall\vec{j}\in V'(d,l)}}\mathbf{A}_{\vec{i}|\vec{j}}\mathbf{F}$$

In each iteration step the non-zero part of current matrix $\mathbf{A}_{\vec{i}|\vec{j}}$ (recall that always only one entry of $\mathbf{p}$ is non-zero) is added to the matrix from previous step. This yields after $(d+1)^{k+l}$ steps:

$$\mathbf{b}^T_\pi\mathbf{H}=\mathbf{b}^T_\pi\mathbf{A}\mathbf{F}$$

with

$$\mathbf{A}=\sum_{\forall\vec{i}|\vec{j}\in V'(d,k+l)}\mathbf{A}_{\vec{i}|\vec{j}}$$

The Bezier bases on both sides of the equation cancel out and we obtain a matrix equation

$$\mathbf{H}=\mathbf{A}\mathbf{F}$$

4.2.3 Unevaluated composition for B-splines

As already pointed out in [18], composing B-splines is considerably more complicated than composing Bezier polynomials. DeRose proposes to convert the B-splines to composite Bezier format and apply the algorithm as described above for each Bezier segment. The result is a 3D composite Bezier curve which can be reduced to B-spline form by removing all unnecessary knots. We could proceed in exactly the same manner and compute composition matrices for each Bezier segment as explained in previous paragraph. Basically, conversion of mutually continuous Bezier segments to a B-spline of the same parametric continuity requires extrapolation between neighboring control points. This way one removes

redundant coefficients from the Bezier representation. This is known as "knot removal", see [52], for example. Though knot removal tends to be numerically unstable for irregular and dense knot vectors it is usually sufficient if the control points of the resulting curve are known. If, however, only the composition matrix is known, the extrapolation has to be performed between the rows of the composition matrix. Here the issue of numerical stability becomes more involved because it enlarges the (albeit small) errors accumulated in composition matrix (numerical analysis of the algorithm for B-splines is postponed to section 4.4) In addition, for efficiency reasons, a careful implementation of sparse matrix products is necessary. On the other hand, the full-size composition matrix (with all Bezier knots unremoved) has approximately d-times (d is the degree of the inner function) more rows than actually required. As will be shown in chapter 7 this has dramatic influence on efficiency when solving the inverse problem. Therefore, we develop the composition of B-splines without this intermediate conversion: we compute the B-spline composition matrix which contains no redundant rows.

4.2.3.1 Preparing the B-spline composition

Let $G(t) = \sum_{i=0}^{p-1} G_i \, b_i^{d,\tau}(t)$ be a B-spline curve in the domain of the B-spline surface $F(u,v) = \sum_{j=0}^{n-1} \sum_{i=0}^{m-1} F_{i,j} b_i^{k,\mu}(u) b_j^{d,\nu}(v)$. Denote the components of $G(t)$ by $(u(t), v(t))$. The starting point for computing the control points of the B-spline curve $H(t) = F(u,v) \circ G(t)$ is to select the intervals I and J for blossoming F and insert the curve into the blossom representation of F:

$$H(t) = F\left(u(t), v(t)\right) = f_{I,J}(\underbrace{u(t), \ldots, u(t)}_{k}; \underbrace{v(t), \ldots, v(t)}_{l})$$

Assume for the moment that the surface consists only of one B-spline patch. Then we may proceed in the same way as in §4.2.2 obtaining a decomposition of the above equation into

$$H(t) = \sum_{i_0}^{d} \ldots \sum_{j_{k+l}}^{d} b_{i_0}^{d,\tau}(t) \cdots b_{j_{k+l}}^{d,\tau}(t) \, f(u_{i_0}, \ldots, u_{i_k}; v_{j_{k+1}}, \ldots, v_{j_{k+l}}) \qquad (4.14)$$

The indices I and J of the surface blossom may be skipped because the two-variate B-spline basis consists only of one segment in each parametric direction. For a general B-spline surface consisting of several polynomial patches, some preprocessing and additional notation are required as will be shown in the following paragraphs.

4.2.3.1.1 Merging the knot vectors of F and G The first subtlety with composing general B-splines is that a knot vector sufficient to represent H must be

constructed from the knot vectors of F and G. Therefore, one cannot write the formulas in the straightforward manner as for Bezier surfaces and curves. Before the composition starts, the knot-vectors of B-splines F and G must be merged as follows: We compute the parameter values t at which $G(t)$ intersects the knot lines of F. Let $\underline{\tau}$ be an ordered set of all real solutions to polynomial equations $\mu_i = u(t) : k-1 \leq i \leq m$ and $\nu_j = v(t) : l-1 \leq i \leq n$. Following De Rose's terminology we refer to these knots as *breakpoint knots*. We introduce new terminology:

- The curve $G'(t)$ obtained by inserting the new knots $\underline{\tau}$ into the original knot vector is called *compatible* with F. Its knot vector is the union of original knots and the breakpoint knots $\tau' = \tau \cup \underline{\tau}$. The compatible knot vector τ' has the format $\tau = \{\ldots, \underline{\tau}_b, \ldots, \underline{\tau}_{b+1}, \ldots\}$, where the dots symbolize that there may be several knots from the original knot vector τ before, after and in-between $\underline{\tau}_b$ and $\underline{\tau}_{b+1}$.
- The segments of $G'(t)$ such that $t \in \langle \underline{\tau}_b, \underline{\tau}_{b+1})$ are called *breakpoint segments*

This has two consequences: First, the breakpoint knots mark locations where G enters a new B-spline patch of F and we need to select blossoming intervals I and J accordingly (see also the example in figure 4.1; here, the intervals will change three times). Second, the restriction to interval $\langle \underline{\tau}_b, \underline{\tau}_{b+1})$ limits the range for iterating over the indices $i_0, \ldots, j_{k+l}$ in eq. 4.14 and it determines the indices of the control points for current breakpoint segment of H.

4.2.3.1.2 Determining the B-spline basis of H The second difficulty is that the result of multiplying the B-spline basis function (first term in eq. 4.14) is a general B-spline function and there is no such simple formula as for Bernstein polynomials to obtain its coefficients. As in the Bezier case, the B-spline basis of the product determines the basis for the composed curve H. Therefore, in order to extract the composition formulae we need to determine the degree and knot vector of the B-spline function resulting from eq. 4.14. Obtaining the coefficients efficiently is more involved and we defer it to §4.3.2. However, the sufficient B-spline basis to represent the product can be determined a-priori: The degree of the B-spline is the product of degree of G and sum of degrees of F in each parametric direction: $d(k+l)$. The knot vector needs to reflect the fact that the degree of the product is higher then the degree of the original B-spline, yet the continuity is the same. The resulting knot vector will contain only knots from the merged basis of G' with multiplicities determined as follows: let m_i be the multiplicity of i-th

distinct knot in τ'. The continuity at a knot is degree minus multiplicity, hence, following identity must hold for the new multiplicity n_i:

$$d - m_i = d.(k + l) - n_i \Longrightarrow n_i = m_i + d.(k - l - 1)$$

This yields the smallest possible knot vector with sufficient support of the product B-spline, in the following denoted by "π". The size of the product B-spline is determined as the size of π minus polynomial order of the product.

Note that π is the "worst-case" knot vector; the actual continuity of H can be larger. Consider figure 4.1: the curve G is linear with no internal knots and F is bi-cubic with two internal knots μ_i and ν_j. Hence, the curve H will have polynomial degree 6 and will be C^2 continuous at both breakpoint knots. However the basis of G' consists of piecewise linear polynomials which are, theoretically, only C^0 as demonstrated by the figure on the right. Since the continuity of the product cannot be higher than the continuity of the operands, the product will be also only C^0 at both breakpoint knots, as the figure demonstrates (the product is shown in bright gray). This means that the multiplicity of knots will be higher than really required. This can be avoided be raising the degree of G' to cubic for the price of higher degree of the resulting curve. Alternatively, one can apply "unevaluated" knot removal to rows of the composition matrix, as mentioned above.

4.2.3.2 The B-spline composition formula

After the B-splines F and G were made compatible and the basis for the B-spline H was constructed we can start with computing the composition of B-splines. We seek the control points of the curve

$$H(t) = F\,(u, v) \circ G'(t),\ \ t \in \langle \underline{\tau}_i, \underline{\tau}_{i+1} \rangle$$

We use the notation from previous paragraph: i.e. π denotes the knot-vector of H and τ' is the merged knot vector of G' which already contains all breakpoint knots:

1. Determine the indices of influenced control points of $H(t)$, $t \in \langle \underline{\tau}_i, \underline{\tau}_{i+1} \rangle$; this is accomplished by locating knots π_P and π_Q such that

   $$\pi_P \leq \underline{\tau}_i < \pi_{P+1} \wedge \pi_Q \leq (\underline{\tau}_{i+1}) < \pi_{Q+1}$$

   Then, by local property of B-splines, the breakpoint segment $t \in \langle \underline{\tau}_i, \underline{\tau}_{i+1} \rangle$ is determined by control points with indices $P - d(k + l)$ to Q

2. Determine the indices of control points of all internal segments of G' which contribute to desired breakpoint segment. Locate knots from τ' with indices R and S such that following relations apply:

   $$\tau'_R \leq \underline{\tau}_i < \tau'_{R+1} \wedge \tau'_S \leq (\underline{\tau}_{i+1}) < \tau'_{S+1}$$

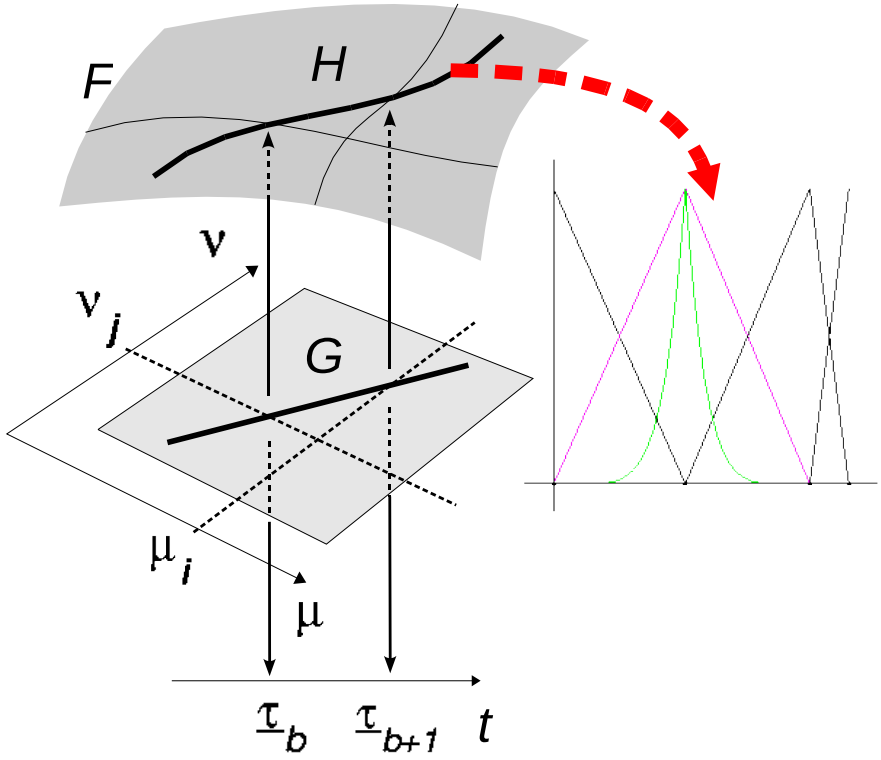


Figure 4.1: Left: the B-splines F and G. Wherever G crosses a knot-line of F a new knot must inserted into G. Right: the symbolic computation of products causes redundant factors in H. The surface is C^2 continuous at knot lines μ_i and ν_j but the worst-case basis for H is apparently only C^0 continuous.

It follows that we consider the control points of G' with indices $R-d$ to S

3. Finally, determine the intervals I and J for blossoming F. Since each break-point segment is completely inside one interval from knot-vectors μ and ν the blossoming intervals I and J must satisfy following relations:

$$\mu_I \leq u(\underline{\tau}_i) < \mu_{I+1} \wedge \nu_J \leq v(\underline{\tau}_i) < \nu_{J+1}$$

With so determined ranges, the composition formula is:

$$\sum_{i=P-d.(k+l)}^{Q} \mathbf{H}_i b_i^{d\cdot(k+l),\pi}(t) = \sum_{s=R-d}^{S} \sum_{i_0=s}^{s+d} \cdots \sum_{j_{k+l}=s}^{s+d} b_{i_0}^{d,\tau}(t) \cdots b_{j_{k+l}}^{d,\tau}(t)\, f_{I,J}(u_{i_0}, \ldots, u_{i_k}; v_{j_{k+1}}, \ldots, v_{j_{k+l}}) \tag{4.15}$$

The iteration ranges follow from the local support properties of B-splines (§A.1.1.3) and can be derived from the B-spline definition and from the de Boor algorithm.

The first "$\sum$" with iteration variable s symbolizes the segment-wise composition of internal segments of G'. Note that it is not necessary to iterate over all p control points of G', since at most $d+1$ subsequent B-spline basis functions can be simultaneously non-zero, see §A.1.1.3.

4.2.3.3 The B-spline composition matrix

Analogously to the Bezier surface-curve composition, the B-spline composition matrix is obtained by rewriting eq. 4.15 in matrix notation. One distinction to the Bezier case is the new iteration variable s which represents the current segment of $G'(t)$. For the s-th segment the variations of $i_0, \ldots, j_{k+l}$ are taken from the ordered set $P_s = \{s, s+1, \ldots, s+d\}$. The set of all possible variations of size k from the set P_s containing $d+1$ elements is denoted by $V'_s(d+1,k)$. This generalization of v-indices to $V'_s(d+1,k)$ is called *shifted v-index* and is denoted by $\vec{\imath}_s \in V'_s(d+1,k)$.

In order to obtain the matrix representation we proceed as follows: Let the size of the $k+l$-fold product of B-splines from eq. 4.15 be q. The product of $k+l$ B-splines is a B-spline of degree $d(k+l)$ with coefficients $p_i : 0 \leq i < q$:

$$b_{i_0}^{d,\tau}(t) \cdots b_{j_{k+l}}^{d,\tau}(t) = \sum_{i=0}^{q-1} p_i^{i_0,\ldots,j_{k+l}}\, b_i^{d(k+l),\pi}(t) \tag{4.16}$$

For a fixed breakpoint segment only the coefficients $p_i^{i_0,\ldots,j_{k+l}}$ with indices $P - d(k+l) \leq i \leq Q$ are considered. Recall that we have restricted the composition to one breakpoint segment, thus only the coefficient according to current range for the control points of H are of importance. Hence we set $p_i^{i_0,\ldots,j_{k+l}} = 0$ for $Q < i < P - d(k+l)$. Denote the size q vector containing these coefficients by $\mathbf{p}_{\vec{\imath}_s|\vec{\jmath}_s}$. Then the product in scalar product notation is

$$b_{i_0}^{d,\tau}(t) \cdots b_{j_{k+l}}^{d,\tau}(t) = \mathbf{b}_\pi^T\, \mathbf{p}_{\vec{\imath}_s|\vec{\jmath}_s} \tag{4.17}$$

The B-spline blossom $f_{I,J}(u_{i_0}, \ldots, u_{i_k}; v_{j_{k+1}}, \ldots, v_{j_{k+l}})$ is expressed as a scalar product of polarized basis functions and the control points $\mathbf{F}$: We denote the vector of polarized basis functions by $\mathbf{a}^T_{\vec{\imath}_s;\vec{\jmath}_s}$ so that the i-th entry of $\mathbf{a}$ is

$$\mathbf{a}_i = a_i^{k,\tau,l,v}(u_{i_0}, \ldots, u_{i_k}; v_{j_{k+1}}, \ldots, v_{j_{k+l}})$$

The $\mathbf{a}_i$ are computed as in §4.1.5 with respect to blossoming intervals I and J. This yields

$$f_{I,J}(u_{i_0}, \ldots, u_{i_k}; v_{j_{k+1}}, \ldots, v_{j_{k+l}}) = \mathbf{a}^T_{\vec{\imath}_s;\vec{\jmath}_s}\, \mathbf{F} \tag{4.18}$$

Substituting 4.17 and 4.18 into 4.15 we obtain

$$\mathbf{b}_\pi^T \mathbf{H} = \sum_{s=R-d}^{S} \sum_{\substack{\vec{\imath}_s \in V_s'(d+1,k) \\ \vec{\jmath}_s \in V_s'(d+1,l)}} \mathbf{b}_\pi^T \; \mathbf{p}_{\vec{\imath}_s|\vec{\jmath}_s} \, \mathbf{a}_{\vec{\imath}_s;\vec{\jmath}_s}^T \; \mathbf{F} \tag{4.19}$$

Evaluating the outer products of the two vectors yields

$$\mathbf{A}_{\vec{\imath}_s;\vec{\jmath}_s} = \mathbf{p}_{\vec{\imath}_s|\vec{\jmath}_s} \mathbf{a}_{\vec{\imath}_s;\vec{\jmath}_s}^T$$

The composition matrix is obtained by iterating over all $\vec{\imath}_s \mid \vec{\jmath}_s \in V_s'(d+1, k+l)$:

$$\mathbf{A} = \sum_{s=R-d}^{S} \sum_{\substack{\vec{\imath}_s \in V_s'(d+1,k) \\ \vec{\jmath}_s \in V_s'(d+1,l)}} \mathbf{A}_{\vec{\imath}_s;\vec{\jmath}_s} \tag{4.20}$$

Note that only that rows of the composition matrix which correspond to non-zero entries of the current vector $\mathbf{p}_{\vec{\imath}_s|\vec{\jmath}_s}$ will be computed. In order to obtain the *complete* composition matrix one needs to repeat this procedure for all pairs of breakpoint knots in τ. Since C^r continuous segments of a B-spline curve share $r+1$ control points care has to be taken that rows of $\mathbf{A}$ corresponding to these common control points of H are not overwritten.

4.3 Efficiency and Data structures

The procedure described in previous section is well suited for demonstration purposes. However, the iteration over all variations of indices in eq. 4.20 would be extremely inefficient: obtaining the complete composition matrix requires approximately $(p-d).(d+1)^{k+l}$ iterations. For example, if the surface is of degree $k = l = 3$ and the curve G has degree $d = 3$ and $p = 20$ the algorithm terminates after ≈ 69000 iterations. In [18] and [53] several methods were proposed to reduce the complexity of the composition algorithm for Bezier simplices. Therefore, in the following we will concentrate only on composition of B-splines. The objective is to show that similar statements as in the before mentioned papers about “one-permutation” and “optimal” algorithm for Bezier simplices are also possible for unevaluated composition of B-splines.

Paragraph 4.3.1 is an excursion to combinatorics: we show how many unique products and blossoms are necessary to compute the composition of two B-splines. Furthermore in §4.3.2 an efficient procedure to obtain the coefficient of B-spline product function is described which utilizes the local properties of B-splines. Using the results gained in §4.3.1 in §4.3.4 we describe an efficient data-structure for storage of intermediate results of the algorithm.

4.3.1 The combinatorics

Analysis eq. 4.14-4.20 reveals two possibilities for decreasing the number of evaluations:

1. The $(k+l)$-fold product of B-spline basis functions (eq. 4.17) does not depend on the ordering of indices in $\vec{\imath}_s \mid \vec{\jmath}_s$. This follows from the commutativity and associativity of products: $a \cdot b \cdot c = a \cdot c \cdot b = b \cdot a \cdot c = \ldots$.

2. Similarly the blossom property 2, §4.1.1 states that inside of one breakpoint segment the value of the blossom term in eq. 4.18 does not depend on the ordering of entries in $\vec{\imath}_s$ and $\vec{\jmath}_s$.

This leads to following consideration: first, compute and store only unique non-zero products and blossoms and second, expand the stored values into the matrix $\mathbf{A}$ such, that the occurance of each particular combination is taken into account appropriately many times. This raises two questions:

- How many unique product and blossom expressions exist?
- How many times each combination appears in the nested iteration in eq. 4.20?

4.3.1.1 The c-index

We need an iteration scheme which ignores the ordering of entries in a v-index. In other words, in all formulae derived in §4.2.2 and §4.2.3 all permutations of fixed is and js yield the same product and, inside of one breakpoint segment, the same value of a blossom. For example, consider all products $b_{i_1}^{k,\tau}(t) \cdots b_{i_5}^{k,\tau}(t)$ such that $2 \leq i_j \leq 4$ for $1 \leq j \leq 5$: The indices in the tuple $(i_1, \ldots, i_5)$ take in values from the set $P_2 = \{2, 3, 4\}$ which we denote by a v-index $\vec{\imath}_2 \in V_2'(3, 5)$. Iterating over all v-indices in $V_2'(3, 5)$ yields all variations of size 5 out of the 3 elements in P_2. However, since the ordering of entries in $\vec{\imath}_2$ is not of importance, many of the products are identical, e.g.:

$$b_2 b_2 b_3 b_3 b_4 = b_2 b_3 b_2 b_3 b_4 = b_4 b_2 b_2 b_3 b_3 = \ldots$$

In order to identify a unique product we only need to know how many times a specific entry appears in $\vec{\imath}_2$. In the above example the indices $2, 3, 4$ appear $(2\times, 2\times, 1\times)$ in different order but for each permutation of two '2's, two '3's, two '4's we obtain the same result. The 3-tuple $(2, 2, 1)$ which contains the number of occurancies of elements from P_2 in $\vec{\imath}_2$ represents a size 3 partitioning of the number $5 = 2 + 2 + 1$. Generally, all v-indices $\vec{\imath}_s \in V_s'(m, n)$ (in this case $m = d + 1$, $n = k + l$) which contain the same number of the same elements from P_s

can be compactly represented in terms of a size-m partition of number n. All such partitionings belong to the class of combinations with repetition from P_s of size n which we denote by $C'_s(m,n)$. We call one such partition a *combinatorial index* or just *c-index* and denote it by $\bar{i}_s \in C'_s(m,n)$, or just $\overline{i_s}$ when m and n are fixed. For fixed s and $P_s = \{s, \ldots, s+m-1\}$ the c-indices correspond to "hyper-indices" used in [18] and [53]. In these papers a recurrence was defined to compute all c-indices given m and n. This allows to write iteration formulae from §4.2.2 and, with slight modifications also the B-spline formulae from §4.2.3, directly in terms of c-indices (for B-splines we need to consider that iteration needs to be performed over all subsequent segments). We define following operations on c-indices:

1. *Component-wise summation* of c-indices $\bar{i}_s \in C'_s(m,n)$ and $\bar{i}_s \in C'_s(m,o)$ yielding $\overline{k}_s = \bar{i}_s + \bar{j}_s$ where $\overline{k}_s \in C'_s(m, n+o)$

2. *Absolute value* of a c-index is $|\bar{i}_s \in C'_s(m,n)| = i_0 + \ldots + i_{n-1} = n$.

In the following a summation over all $\bar{i}_s \in C'_s(d+1, k+l)$ is denoted by $\sum_{\forall \bar{i} \in C'_s(d+1,k+l)}$.

4.3.1.2 The relation between v-indices and c-indices: "The c-number"

The c-index notation considerably reduces the number of necessary iterations. Note that the number of v-indices in $V'_s(m,n)$ is m^n compared to number of c-indices in $C'_s(m,n)$ which is $\binom{m+n-1}{n}$. However, in order to compute the composition matrix, in eq. 4.20 all variations of indices (all v-indices) must taken into account. Thus, we need to know how many times a particular v-index will appear during the iteration. In other words we need to know how many v-indices are mapped to a particular c-index. This is a nice exercise in combinatorics: as above, consider, for example, the c-index $\overline{i_2} \in C'_2(3,5) = (2,2,1)$. Each corresponding v-index $\vec{i}_2 \in V'_2(3,5)$ has five positions containing 5 numbers: $2\times$'2', $2\times$'3' and $1\times$'4'. Take 5 positions of a v-index $V'_2(3,5)$ and place the '2' in two of them. This gives $\binom{5}{2}$ possibilities. Next, place the two '3's in the remaining free positions which is possible in $\binom{5-2}{2}$ ways, since two positions are already occupied by the '2's. At last, place the '4' for which there are $\binom{5-2-2}{1}$ possibilities. This gives a total of

$$\binom{5}{2} \cdot \binom{5-2}{2} \cdot \binom{5-2-2}{1} = \frac{5!}{3! \cdot 2!} \cdot \frac{3!}{1! \cdot 2!} \cdot \frac{1!}{0! \cdot 1!} = \frac{5!}{2! \cdot 2! \cdot 1!} = 60$$

possibilities to expand the c-index $\bar{i}_2 = (2,2,1)$ into v-indices. We call this a *c-number* of a c-index a denote it by $\#\bar{i}_s$. The above procedure can be generalized for any c-index from $C'_s(m,n)$ yielding

$$\#\bar{i}_s \in C'_s(m,n) = \frac{n!}{i_0! \cdots i_{m-1}!} \tag{4.21}$$

In the nominator we have the size of the v-index, the entries of the c-index appears in the denominator. Thus with each c-index there associated a c-number that determines the number of v-indices which can be mapped to a particular c-index. Now we see that in eq. 4.20 it suffices to compute only products and blossoms corresponding to all $\bar{\imath}_s \in C'_s(d+1, k+l)$. If each $\bar{\imath}_s$ is multiplied by its c-number (i.e. it is added $\#\bar{\imath}_s$-times) all variations which are necessary for the composition matrix will be automatically considered.

4.3.1.3 The total number of unique c-indices

Consider eq. 4.15: in order to compute all segments of the curve $H(t)$ we need to iterate over all segments of the inner B-spline $G'(t)$. Denote the number of control points of $G'(t)$ by p and its degree by d. Then the indices $i_1, \ldots, j_{k+l}$ will take in values from $P_s = \{s, s+1, s+d\}$ for $0 \leq s \leq p-d+1$. Hence, we need to determine the total number of unique c-indices for all segments contained in the set $P = \{0, 1, \ldots, p-1\}$.

For simplicity in the following we set $m = d+1$ and $n = k+l$. Since each P_s-set has the same size there will be the same number of c-indices for each segment which is $\binom{m+n-1}{n}$. Recall that although corresponding c-indices from different segments have the same entries they are mapped to different v-indices: for example, $\bar{\imath}_0 \in C'_0(3,3) = (0,1,2)$ is mapped to all v-indices of size 3 from $P_0 = \{0,1,2\}$, $\bar{\imath}_1 = (0,1,2)$ is mapped to v-indices from $P_1 = \{1,2,3\}$ and so on. Note that there are redundant c-indices in neighboring segments, every two neighboring segments overlap in $m-1$ entries. For example, $\bar{\imath}_0 = (0,1,2)$ and $\bar{\imath}_1 = (1,2,0)$ are mapped to the identical set of v-indices:

$$\left.\begin{array}{l} \bar{\imath}_0 = (0,1,2) \\ \\ \bar{\imath}_1 = (1,2,0) \end{array}\right\} (1,2,2), (2,1,2), (2,2,1)$$

In general, all c-indices which have a '0' at the right-most boundary already appear in the previous segment. Therefore, for each segment such that $s > 1$ the size of candidate set P_s is reduced by one and we may skip the right-most entry which appears '0' times. This yields $\binom{m+n-1}{n} - \binom{m+n-2}{n}$ unique c-indices for each segments, except the first one – which has full number of $\binom{m+n-1}{n}$ c-indices. There are $(p-m)$ segments, hence, total number of unique size n combinations of "bandwidth" m from p elements is

$$(p-m+1) \cdot \binom{m+n-1}{n} - (p-m) \cdot \binom{m+n-2}{n} \qquad (4.22)$$

4.3.2 The products

Next we discuss the problem of multiplying B-splines. This operation is required in equations 4.15-4.20. We wish to compute the coefficients p_i given $k+l$-fold product of B-spline basis functions which define the curve $G'(t) = \sum_{i=0}^{p-1} G_i\, b_i^{d,\tau'}(t)$:

$$b_{i_0}^{d,\tau'}(t) \cdots b_{j_{k+l}}^{d,\tau'}(t) = \prod_{j=0}^{k+l} b_{i_j}^{d,\tau'}(t) = \sum_{i=0}^{q-1} b_i^{d(k+l),\pi}\, p_i^{i_0,\ldots,i_{k+l}} \tag{4.23}$$

4.3.2.1 Determining the number of unique products

Armed with the combinatorics from previous paragraph we can easily count the number of unique B-spline products given p, k, l and d: The number of unique non-zero products required for *complete* composition matrix is given by the number of shifted c-indices which is determined by eq. 4.22. For comparison, if $p = 20$, $k + l = 6$ and $d = 2$ the economy-factor compared to the straightforward algorithm is $69000 : 385 \approx 180$. The c-number (equation 4.21) defines how many times a specific combination of is will be encountered during the summation. Hence, re-writing eq. 4.20 in this "economy" c-index notation yields

$$\mathbf{A} = \sum_{s=R-d}^{S} \sum_{\forall \bar{\imath} \in C'_s(d+1,k+l)} \#\bar{\imath} \cdot \mathbf{p}_{\bar{\imath}_s}\, \mathbf{a}^T_{\bar{\imath}_s \to (\overline{u}_s, \overline{v}_s)} \tag{4.24}$$

The subscript $\bar{\imath} \to (\overline{u}, \overline{v})$ denotes a conversion of current c-index to a pair of c-indices which determine arguments of the unevaluated surface blossom. We leave this conversion open until §4.3.3 where an algorithm for obtaining the matching blossom is described.

4.3.2.2 Multiplying B-splines

The vector $\mathbf{p}_{\bar{\imath}_s}$ contains coefficients of the B-spline function from eq. 4.23. There are three possibilities to compute the coefficients:

1. Convert each operand B-spline basis function to composite Bezier form, multiply the corresponding Bezier segments obtaining a composite Bezier function of degree $d(k+l)$. Since it is known in advance which knots are redundant (refer to §4.2.3) these knots may be removed yielding the desired coefficients of the product. This method was used, for example, by Kazinnik and Elber in [45].

2. Use Mørken's symbolic method [58] and recursively multiply each pair of operands

3. Interpolate the product. Determine the degree and knot vector of the product B-spline as explained in §4.2.3, evaluate the basis functions and values of the operands yielding a well-determined linear system of equations in unknowns $\mathbf{p}_{\bar{\imath}_s}$. Then solve the system obtaining the desired values

Although we don't have the ambition to prove that, it appears that the first two methods are equivalent. In [58] Mørken has shown how to express a product of two B-splines recursively as a linear combination of the coefficients of the operands. The recursion is quite complex and we will not rewrite it here. It was designed to compute a product of two general B-spline functions; hence, we would have to multiply the operands of the $k + l$-fold product recursively. With regard to equivalence of the first two methods note that if the minimum support B-spline basis of the product is known in advance then following quantities can be precomputed:

- Knot insertion matrices for B-spline→Bezier conversion, §4.1.5
- Factors resulting from multiplying Bernstein polynomials, see [41, §4.1, eq. 4.2d]
- Knot removal matrices (for the Bezier→B-Spline conversion, see §4.1.5)

Then, using matrix multiplications, a concatenation of following operations yields a closed formula for multiplying B-splines:

1. Transform the B-spline operands to Bezier format
2. Multiply the Bezier segments
3. Remove the unnecessary knots

We conclude that if a tough algebraist undergoes that task Mørkens formula is revealed. The amount of work required by Mørken's algorithm is, in general setting, $O\left((d(k+l))^4\right)$ linear combinations per B-spline segment, see also [81]. Similar result should be obtained for the "Bezier-conversion" algorithm. Keeping track of the sparsity of the operands (note that we multiply only B-spline basis functions, not general dense B-splines) and intermediate result should improve the run-time behavior of the algorithm. However, the implementation is very sensitive to programming errors and requires a considerable effort. Therefore we have used the third method which is easy to implement and more efficient – if the properties of the interpolation matrix and the sparsity of the product B-spline are taken into account. Since this is essential for an efficient implementation, we will discuss it in more detail.

4.3.2.3 Determining non-zero coefficients of the product

In the resulting product function (eq. 4.23) the degree of the original B-spline G' is increased by the factor $(k+l)$. Accordingly, the number of coefficients in the product function (and in the resulting B-spline H) increases approximately of the same factor. Fortunately, not all q coefficients of the product function will be non-zero and the sparsity pattern is easy to predict: The non-zero interval of a B-spline basis function $b_i^{d,\tau}(t)$ is $t \in \langle \tau_i, \tau_{i+d+1} \rangle$, see e.g. [41]. Hence, the product $\prod_{j=0}^{k+l} b_{i_j}^{d,\tau}(t)$ will be non-zero only at the intersection of the support intervals of the operands. Since a B-spline function is zero if and only if its coefficient are zero we conclude that the non-zero coefficients of the product function will appear only inside of certain band-width.

Recall that the knot vector of the product function, π, contains only knots contained in the knot vector of the operands, hence a knot from τ will also occur in π. Let $i_{\min}$ and $i_{\max}$ be indices such that $i_{\max} - i_{\min} \leq d$. Then the non-zero interval for the product is $t \in \langle \tau_{i_{\min}}, \tau_{i_{\max}+d+1} \rangle$. It follows that the non-zero indices of the product must be inside of the range $j_{\min} - d(k+l)$ to $j_{\max}$ where $\pi_{j_{\min}} \leq \tau_{i_{\min}} < \pi_{j_{\min}+1}$ and $\pi_{j_{\max}} \leq \tau_{i_{\max}} < \pi_{j_{\max}+1}$.

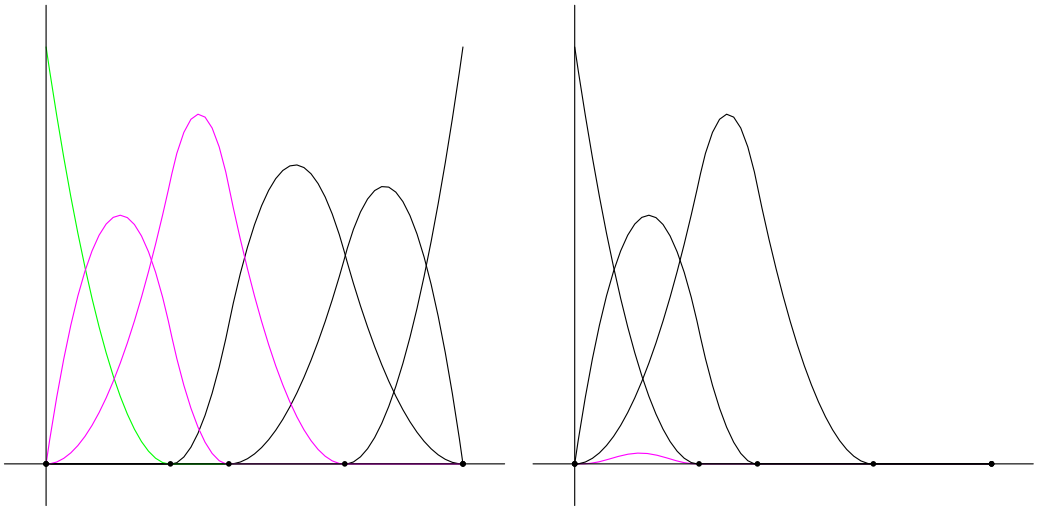
To see the effect of that, consider a B-spline basis $d = 2$, $p = 6$ and knot-vector $\tau = \{0, 0.3, 0.4, 0.7, 1\}$ shown in figure 4.2(a). The knots with multiplicities $m_\tau = \{3, 1, 1, 1, 3\}$ are marked by dots. Consider a 3-fold products $b_0 b_i b_j$ for $i, j \in P = \{0, \ldots, 5\}$. $b_0(t) \neq 0$ if $t \in \langle \tau_0, \tau_3 \rangle = \langle 0, 0.3 \rangle$. The 3-fold product $b_0 b_i b_j \neq 0$ iff $i \leq 2 \wedge j \leq 2$, because for $t \in \langle 0, 0.3 \rangle$ only b_1 and b_2 (colored bright in the figure) are simultaneously non-zero. figure 4.2(b) shows the product $b_0 b_1 b_2$ in bright gray and the operands in black. The product B-spline has $d = 6$, $q = 22$, knot vector π with $m_\pi = \{7, 5, 5, 5, 7\}$ and the non-zero coefficients are aggregated in interval $0 \leq i \leq 6$.

4.3.2.4 Optimizing the product interpolation

The interpolation of products was applied for example by G. Elber in [23] for computing products of two B-splines and can be very simply extended for n-fold products of basis functions as follows: We state the interpolation problem with the system matrix $\mathbf{M}$

$$\mathbf{M}\,\mathbf{p}_{\bar{\imath}_s} = \mathbf{v}_{\bar{\imath}_s}; \forall \bar{\imath}_s \in C'_s(d+1, k+l) \tag{4.25}$$

The rows of the matrix $\mathbf{M}$ contain basis functions of the product basis evaluated at the Greville abscissae of the product B-spline. I.e. the j-th element in the i-th row of is $b_j^{d(k+l),\pi}(\xi_i)$ where $\xi_i = \sum_{i=s}^{s+d(k+l)} \pi_i$. The vector $\mathbf{v}_{\bar{\imath}_s}$ contains values of Gs basis functions evaluated at ξ_i and multiplied according to current c-index, i.e.



(a) (b)

Figure 4.2: Demonstrating the economy of utilizing the predicted bandwidth of a B-spline product: The n-fold product of B-splines is non-zero only at the intersection of support intervals of its operands. Here, we multiply the first three basis functions (colored bright gray in the left figure). The right figure shows the operands and the result (colored bright gray); The resulting B-spline function has only 7 non-zero coefficients out of 22.

the i-th entry of $\mathbf{v}$ is $b_{i_1}^{d,\tau}(\xi_i) \ldots b_{i_{k+l}}^{d,\tau}(\xi_i)$. The solution of the inverse problem

$$\mathbf{M}^{-1}\,\mathbf{v}_{\bar{\imath}_s} = \mathbf{p}_{\bar{\imath}_s} \tag{4.26}$$

yields the coefficients of the product function for the current c-index.

The complexity of this method is influenced by the size and shape of the matrix $\mathbf{M}$ and by the number of right sides in eq. 4.25, i.e. by the number of unique products. Given p, k, l and d as above and q as the size of the product, $\mathbf{M}$ is a banded matrix of size $q \times q$ and with maximum upper and lower bandwidth $l = u = (d+1).(k+l)$. It will be close to be diagonally dominant and, typically, no pivoting will be necessary. We compute banded LU-decomposition of $\mathbf{M}$ and solve the system for all right sides simultaneously. LU decomposition for banded matrices can be computed in circa $O(2.q.l.u)$ operations, see [29]. If no pivoting is applied, the triangular factors of size $q \times q$ inherit the lower ($\mathbf{L}$) and upper ($\mathbf{U}$) bandwidth from $\mathbf{M}$. The solution for each $\mathbf{v}_{\bar{\imath}}$ is obtained by forward and backward substitution in $O(2.q.l) + O(2.q.u)$ operations. We have tested this method with large and high degree B-splines. The performance is satisfactory for middle-sized

data sets (see section 4.4), for large data sets the back-substitution unacceptably slows down the execution of the algorithm. This happens if the number of "right-hand sides" is much larger then the size of the matrix. In this case, it is more efficient to compute the inverse of $\mathbf{P}$ and obtain the coefficients of the product B-spline by means of eq. 4.26. Note that in this case the complexity of computing $\mathbf{M}^{-1}$ is only only as large as computing the banded LU-decomposition of $\mathbf{M}$ plus computing the solutions for q right-hand sides (initially set to unity matrix). We utilize the knowledge about the predicted bandwidth of the result $\mathbf{p}_{\bar{\imath}_s}$ and multiply only a sub matrix of $\mathbf{M}^{-1}$ and subvector of $\mathbf{v}_{\bar{\imath}}$. For very irregular and dense knot vectors pivoting becomes necessary in order to stabilize the LU-decomposition. Then, no prediction on bandwidth of $\mathbf{L}$ can be made; However, the predictions on maximum number of non-zero entries in $\mathbf{M}$s columns still apply.

A great advantage is a simple implementation once a LU decomposition is available; a good implementation of banded LU-decomposition is, for example, GDBLU in LAPACK, see [2].

4.3.3 The blossoms

In this paragraph we discuss efficient evaluation of all unique blossoms required in equation 4.24. When counting the number of unique blossoms we need to consider that new set of blossoms is computed for each breakpoint segment. Also we need to consider that blossoms of a tensor-product B-spline are not invariant under permutation of variables from different parametric directions.

More specifically, in the following we seek unique vectors $\mathbf{a}^T_{\bar{\imath}_s;\bar{\jmath}_s}$ in equation 4.24 for all $\bar{\imath}_s \in C'_s(d+1,k)$ and $\bar{\jmath}_s \in C'_s(d+1,l)$. Furthermore, given a non-zero product corresponding to a concatenation of c-indices $\bar{\imath}_s|\bar{\jmath}_s$ we need to identify all compatible blossom corresponding to a decomposition $\bar{\imath}_s|\bar{\jmath}_s \in C'_s(d+1,k+l) \rightarrow (\overline{u}_s, \overline{v}_s)$ where $\overline{u}_s \in C'_s(d+1,k)$ and $\overline{v}_s \in C'_s(d+1,l)$

4.3.3.1 Determining the number of unique blossoms

The total number of unique blossoms is determined by the number of unique non-zero products. Inside of one breakpoint segment a blossom does not depend on the ordering of arguments, see §4.2.3; hence, the total number of unique blossoms per breakpoint segment of size $r = S-(R-d)+1$ (S and R are defined in §4.2.3) is given by eq. 4.22:

$$\begin{aligned}(r-d+2)\cdot\left[\tbinom{d+k}{k}+\tbinom{d+l}{l}\right]-\\ (r-d+1)\cdot\left[\tbinom{d+k-1}{k}+\tbinom{d+l-1}{l}\right]\end{aligned}$$

4.3.3.2 Obtaining the compatible blossoms

We need to consider a decomposition of c-index which identifies a product into all c-indices which identify all compatible tensor-product blossoms. The subtlety is that one cannot mix the variables throughout the argument bags of tensor-product blossoms. Consider, for example a c-index $\bar{\imath}_2 \in C'_2(3,6) = (3,2,1)$. One v-index corresponding to $\bar{\imath}_2 = (3,2,1)$ is $\vec{\imath}_2 \in V'_s(3,6) = (2,2,2,3,3,3,4)$. This permutation of indices has to be distributed throughout the u and v-argument bags of the blossom. In contrast to product terms which are independent under variations of their arguments the blossoms of a tensor product are not. E.g., we have

$$f(u_2,u_2,u_2;v_3,v_3,v_4) = f(u_2,u_2,u_2;v_4,v_3,v_3)$$

but

$$f(u_2,u_2,u_2;v_3,v_3,v_4) \neq f(u_3,u_3,u_4;v_2,v_2,v_2)$$

From this we conclude that only blossoms corresponding to special combinations of arguments will match one product. All such compatible combinations are obtained from current c-index as follows: given

$$\bar{\imath}_s \in C'_s(d+1,k+l)$$

each decomposition $\bar{\imath}_s \rightarrow \overline{u}_s, \overline{v}_s$ such, that

$$\bar{\imath}_s = \overline{u}_s + \overline{v}_s \wedge \overline{u}_s \in C'_s(d+1,k) \wedge \overline{v}_s \in C'_s(d+1,l) \tag{4.27}$$

delivers a valid blossom. For example, setting k, l and d as above, the c-index $\bar{\imath}_2 = (3,2,1)$ is decomposed into all matching c-indices as follows (the '2' subscripts are omitted):

$\bar{\imath}_2 \in C'_2(3,6) = (3,2,1) \rightarrow \overline{u}, \overline{v}$

$\overline{u} \in C'_2(3,3)$	$\overline{v} \in C'_2(3,3)$
$(0,2,1)$	$(3,0,0)$
$(1,1,1)$	$(2,1,0)$
$(1,2,0)$	$(2,0,1)$
$(2,0,1)$	$(2,0,1)$
$(2,1,0)$	$(1,1,1)$
$(3,0,0)$	$(0,2,1)$

Note, that not all c-indices from $C'_2(3,3)$ appear in the above table: e.g., for $(0,1,2)+(i,j,k) = (3,2,1)$ no compatible i, j and k exist. Since each product which corresponds to c-index $\bar{\imath}_s$ appears only once during the summation, we need to consider all matching blossoms, otherwise, certain blossoms are missed and the

composition matrix would be incomplete. Thus, the curve-surface composition matrix is computed as:

$$\mathbf{A} = \sum_{s=R-d}^{S} \sum_{\forall \bar{\imath}_s \in C'_s(d+1,k+l)} \#\bar{\imath}_s \cdot \mathbf{p}_{\bar{\imath}_s} \mathbf{a}^T_{\bar{\imath}_s \to \overline{u}_s, \overline{v}_s} \tag{4.28}$$

where $\mathbf{a}^T_{\bar{\imath}_s \to \overline{u}_s, \overline{v}_s}$ denotes a sum:

$$\mathbf{a}^T_{\bar{\imath}_s \to \overline{u}_s, \overline{v}_s} = \sum_{\substack{\overline{u}_s \in C'_s(d+1,k) \\ \overline{v}_s \in C'_s(d+1,l) \\ \bar{\imath}_s = \overline{u}_s + \overline{v}_s}} \mathbf{a}^T_{\overline{u}_s, \overline{v}_s}$$

4.3.4 Implementation: the Multi-index tree

On-the-fly computation of blossoms in eq. 4.28 would be extremely inefficient. To see this recall that the tensor-product $\mathbf{a}$-vectors arise from concatenation of $\mathbf{a}$-vectors for each parametric dimension of the tensor product. An inspection of formulas 4.27 and 4.28 reveals that the same $\mathbf{a}$-vectors will appear in the decomposition of different c-indices. If the blossoms were computed on the fly, we would have to compute identical $\mathbf{a}$s several times. The evaluation of an $\mathbf{a}$-vector for a degree d B-spline requires $O(d^2)$ linear combinations for B-spline of degree d. For a tensor-product of B-splines with degrees k and l the concatenation requires $O(k+l)$ multiplications, thus, we have complete cost of building one $\mathbf{a}_{\overline{u},\overline{v}}$ circa $O(k^2+l^2)$. Clearly, the performance is improved if all unique $\mathbf{a}$-vectors are computed only once, stored in a suitable data structure and retrieved on demand. In the next two paragraphs we introduce:
1. The, so-called, Multi-index tree – a data container which allows to insert and retrieve a B-spline blossom (the a-vector) identified by a c-index in d steps where d is the degree of the B-spline.
2. An algorithm using the Multi-index tree which is optimal according to Mann's definition, [53]: among all algorithms it requires the minimal number of linear combinations to compute the value of a blossom

4.3.4.1 Storing the blossoms in a tree

Design of a "suitable" data structure which stores all unique blossoms is a trade-off between storage requirements, fast access and re-usability of the implemented data structure. Given a c-index we must be able to insert or retrieve a precomputed entry fast, possibly without any kind of preprocessing, sorting or ordering. DeRose et al. and S. Mann [18], [53] have used a lexicographic ordering of hyper-indices (c-indices). While this is easy to implement for Bezier polynomials (or for

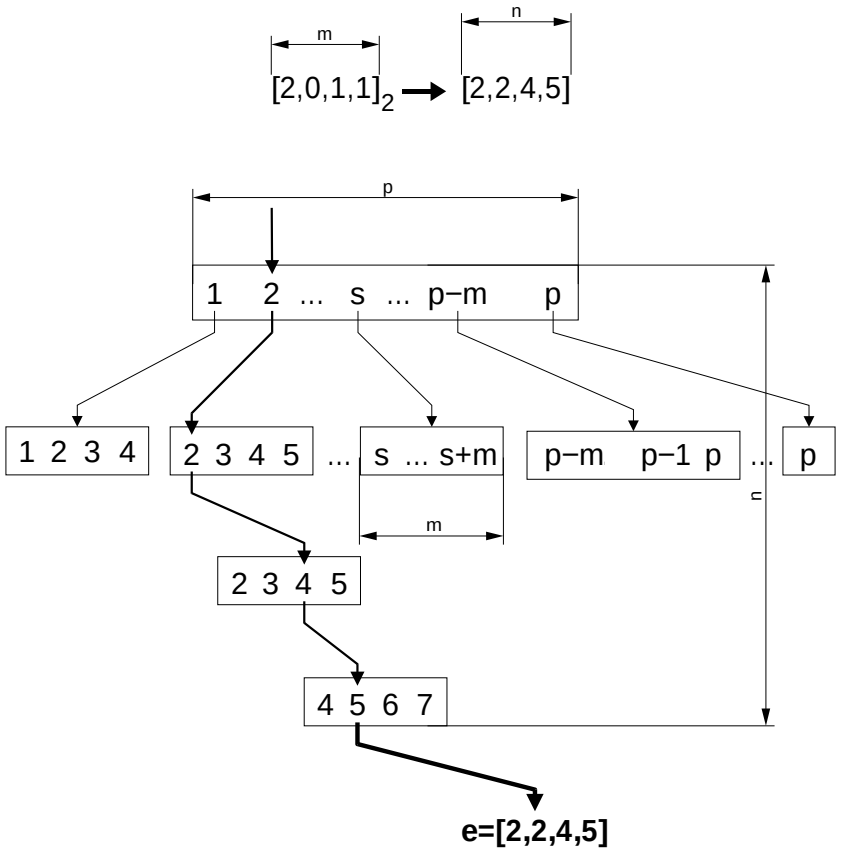


Figure 4.3: Accessing an entry in the Multi-index tree

B-splines converted to Bezier format), it becomes difficult when dealing with B-splines directly. We did not succeed to find an ordering which would reflect the concept of shifted c-indices as described in paragraph 4.3.1. Hence, we propose to store entries which can be uniquely identified by a c-index in a tree structure.

We want to store $\mathbf{a}$-vectors which are uniquely identified by a c-index $\bar{\imath}_s \in C'_s(m,n)$. Recall that a c-index represents unordered combinations of n integers from $P_s = \{s, s+1, \ldots, s+m\}$ such that $|\bar{\imath}_s| = n$. Each c-index is mapped to $\#\bar{\imath}_s$ v-indices of size k. Representatively, one v-index can be obtained from given c-index by counting the number of entries i_r for $1 \leq r \leq m$. This corresponds to component-wise expansion of $\bar{\imath}_s$ into v-index $\vec{\jmath}$ of size n:

$$\bar{\imath}_s = (i_1, \cdots, i_m) \rightarrow \overbrace{(\underbrace{s, \ldots, s}_{i_1\times}, \underbrace{s+1, \ldots, s+1}_{i_2\times}, \cdots, \underbrace{s+m, \ldots, s+m}_{i_m\times})}^{n\times}$$

The so-called *Multi-index tree* maps this conversion to a tree structure organized as in figure 4.3: each node, except of the root node, consists of an array of dimension

m with links to its children nodes. The array of the root node has size of the the total number of control points in the current breakpoint segment, here denoted by p. Leaf nodes contain $\mathbf{a}$-vectors corresponding to given c-index. As a query, c-index $\bar{\imath}_s$ is passed to the tree. Depending on the entries of $\bar{\imath}_s$ the tree is traversed according to v-index expansion as described above. For example: assume that we want to store blossoms of degree 4 B-spline. The c-index $\bar{\imath}_2 \in C_2'(4,4) = (2,0,1,1)$ is expanded into v-index $\vec{\imath}_2 \in V_2'(4,4) = (2,2,4,5)$. We enter the tree at the root node on 2nd position of the array. We descend into the child node which represents the entry "2". Proceeding this way after four steps we arrive at the leaf node which corresponds to c-index $(2,0,1,1)$, or equivalently, the sorted v-index $(2,2,4,5)$. At the leaf nodes, the $\mathbf{a}$-vector that corresponds to the blossom with arguments (u_2,u_2,u_4,u_5) is found and returned as an answer to the query. Note that the tree is initialized by simply allocating the root node for p entries. The figure 4.3 shows the traversal of the C-Tree for $m = n = 4$ and $s = 2$, the solid arrows mark the way along the tree.

The storage requirements for a Multi-index tree are as follows: we need an array of size p at the root node, which has p children each of size m. All subsequent nodes have m children, i.e. there are roughly $p + pm + pm^2 + pm^3 + \ldots + pm^n$ entries, thus, there are circa $O\left(p.m^{n-1}\right)$ auxiliary "administration" elements in addition to the intrinsic entries. This applies if we pre-compute and store all unique products and blossoms ever required. However, we only need to store products and blossoms for two neighboring segments at a time: this reduces the memory cost to approximately $O\left(m^{n-1}\right)$ which is acceptable for small numbers m and n. Recall that m equals the degree of the inner B-spline curve G and n is the sum of the polynomial degrees of the surface. These values are typically small, such as $m = 3$ and $n = 6$, for example.

4.3.4.2 Optimal utilization of Multi-index trees

We use the internal (administration) nodes of the Multi-index tree to speed up the computation of blossoms: Assume that we wish to compute and store all unique blossoms $f_I(u_{i_1},\ldots,u_{i_4})$, such that $0 \le i_j \le m$ for $1 \le j \le 4$. Further assume, without loss of generality, that the blossom arguments are sorted in ascending order i.e. $u_{i_1} \le u_{i_2} \le u_{i_3} \le u_{i_4}$. Consider the triangular scheme for evaluation of $\mathbf{a}$-vectors containing the generalized B-spline basis functions, §4.1.5, eq. 4.10:

$$
\begin{array}{lll}
a^{n-1}_{I-n-1}(u_{i_1}, u_{i_2}, u_{i_x}, u_{i_y}) & & \\
& \ddots & \\
\vdots & & a^1_{I-1}(u_{i_1}, u_{i_2}) \\
& & \\
a^{n-1}_{I}(u_{i_1}, u_{i_2}, u_{i_x}, u_{i_y}) \quad \cdots & a^1_I(u_{i_1}, u_{i_2}) & a^0_I(u_{i_1}) = 1 \\
\downarrow & \downarrow & \downarrow \\
\mathbf{a}^{n-1} & \mathbf{a}^1 & \mathbf{a}^0
\end{array}
$$

Generally, the entries at $(j+1)$-th level are computed as a linear combination of coefficients from j-th level. At each level an intermediate $\mathbf{a}$-vector is computed; its entries are the a-coefficients in the current column of the triangle. Since we iterate over all combinations of the blossom arguments $(u_{i_1}, \ldots, u_{i_4})$ many identical intermediate a-coefficients are computed more than once. For example the coefficients in the first two columns of the triangle are required for all blossom $f(u_{i_1}, u_{i_2}, u_{i_x}, u_{i_y})$ where $1 \leq x, y \leq 4$. Therefore, we store the intermediate a-coefficients at the corresponding internal nodes of the Multi-index tree. All children of an internal node re-use the precomputed coefficient from their parent node.

We conclude that this leads to the same result as Mann's optimal version of the composition algorithm for Bezier polynomials, see [53]. Translated to our terminology, he used a specific ordering of c-indices in order to achieve an optimal utilization of intermediate $\mathbf{a}$-vectors. The term "optimal" was interpreted in the sense that there is no other algorithm which would compute all required blossoms with less linear combinations.

4.4 Practical notes and some results

The formula 4.28 can be directly converted into computer code. Although in both cases the products can be computed on the fly, it is not always the best choice. It depends on the product computation method: if "direct" methods (1. and 2. from section 4.3.2) are applied then there is no need for a temporary storage of the products. If the 3rd method is used, we need to choose from the inverse-matrix method and the back-substitution method. The former case requires only storage of the inverse matrix, the values of the products can be computed on the fly. In the latter case, the LU back-substitution must be applied to all right-sides simultaneously; therefore, one has to store all unique products in a column-major matrix along with corresponding v-indices and the bandwidth information.

With regard to blossoms, we compute all unique $\mathbf{a}$-vectors for each parametric direction and store them in two Multi-index trees. According to eq. 4.28, for each

$\mathbf{p}_{\bar{\imath}_s}$, we compute valid pairs of $\overline{u}_s$ and $\overline{v}_s$ for current $\overline{\imath_s}$. Then trees is queried for $\mathbf{a}$-vectors corresponding to current $\overline{u}$ and $\overline{v}$. Finally, the outer product update

$$\mathbf{A} \leftarrow \mathbf{A}_{\bar{\imath}_{old}} + \mathbf{p}_{\bar{\imath}_{new}} \cdot \mathbf{a}^T_{\bar{\imath}_{new} \rightarrow \overline{u},\overline{v}}$$

is performed; note, that the effort of explicit concatenation of $\mathbf{a}_{\overline{u}}$ and $\mathbf{a}_{\overline{v}}$ into $\mathbf{a}_{\overline{u},\overline{v}}$ can be saved if it is built implicitly during the outer product update.

4.4.1 Run-time performance

The algorithm consists of three significant components:

1. computation of products, denoted by T_1
2. computation of blossoms (T_2)
3. the outer products updates (T_3)

In the following we will consider the run-time performance of the curve-surface composition. Let F be a B-spline surface and G the domain curve. The algorithm has following parameters:

variable	meaning
p	size of G'
d	degree of G'
k, l	degrees of F in u and v parametric directions
m, n	sizes of F in u and v

The effort required for T_1 and T_2 can be estimated according to the analysis from section 4.3; Note, that m and n do not appear in the combinatorial formulae – the computation of B-spline blossoms is always local and depends only on the polynomial degree of the surface. The knot-line density, and thus, the size of the surface is indirectly reflected by the factor p: it increases with the number of knot-lines intersections with the domain curve G, as described in §4.2.3. Results for typical values of p, k, l and d are shown in figure 4.4[2]: as expected, the run-time of the algorithm depends almost linearly on p, as eq. 4.22 predicts. The number of unique products per segment is determined by the binomial coefficient $\binom{d+k+l}{k+l}$; i.e. it grows dramatically with growing degrees of the inner and outer functions d and $k + l$, as illustrated by the graphs in fig. 4.4(b)-(c). Fortunately, in practical applications G and F will rarely have degree higher than cubic. On

[2]All performance test in this thesis were executed on a 750 Mhz Athlon PC with 128 MB dynamic memory available.

the other hand, p can grow very rapidly; note that the surface F is refined in order to generate new degrees of freedom for the surface. By the variation diminishing property of planar B-spline curves, in worst case, each new knot-line of F can introduce order of d new knots per segment in G.

Compared to the DeRose's original composition algorithm, the only additional operation is the summation of outer products in equation 4.28. An outer product update involves an amount of work which is quadratic in number of non-zeros in both vectors. The number of required outer product updates is determined by number of unique products. As we have shown, the density (number of non-zero entries compared to the size of the vector) of both argument vectors will be, typically, small: The density of $\mathbf{a}$ is proportional to the ratio $\frac{k.l}{m.n}$. Similarly density of $\mathbf{p}$ is proportional to $\frac{d}{p}$. We stress that using this sparsity information is essential, otherwise the cost for component T_3 become intractable. An interesting insight into the run-time behavior of the algorithm is given by comparing the ratio of time spent for each component. Figure 4.5 demonstrates the dependency of $T_1 : T_2 : T_3$ on growing p. T_1 (product computation) is the most expensive component, followed by the blossom computation (T_2). In all run-time test we have used the interpolation method to compute the products. Large p implicates large size of the product B-spline and thus a large (but sparse) interpolation problem. Thus, the most time is spent by LAPACKs GDBLU which delivers the LU decomposition and performs the back-substitution.

If G is converted to Bezier format before the composition starts, evaluation of one product is actually a constant time operation – under the assumption that the factors required to compute n-fold product of Bernstein polynomials are precomputed. However, there will be approximately factor d more blossom and outer-product updates necessary. In addition, as mentioned in §4.3.2, in this case the unevaluated H is obtained in composite Bezier format. The consequence is an factor d higher composition matrix which considerable slows down solving of the inverse problem, see chapter 7 for more details.

4.4.2 Numerical stability and shape of the composition matrix

The numerical stability of the algorithm is influenced by the errors caused in its three components. The error in the coefficients of the product $\mathbf{p}$ depends on the method chosen for products computation; since the former two methods include knot removal, their numerical stability will depend on the density and regularity of the merged knot vector of G. Basically the same applies to the third method – the overall precision of the interpolation matrix is also influenced by the regularity of the knot vector. The blossom evaluation is as stable as the de Boor algorithm itself; there is a danger of number cancellation when computing the factors α in formula 4.7 for irregular knot vectors. The third component are the outer product

updates. This operation accumulates the product of errors in p_i and a_j in the element c_{ij}. Though the errors are small, consider, that there are thousands of outer product updates even for moderate values of p, d, k and l. In general, the error in each entry in the composition matrix will be proportional to how many times it was updated.

It follows from the above considerations that the numerical precision is primarily influenced by the parametrization and shape of the "merged" curve G. Recall that each intersection of G with a knot-line of F generates a breakpoint knot in G. Since G may cross the knot lines of F in arbitrary ways it is not possible to avoid irregular knot intervals. To see the influence on the numerical stability of the algorithm, let us review the concept of "composition matrix" from a different perspective:

Recall that the linear transformation $\mathbf{H} = \mathbf{AF}$ delivers the control points of the curve $F(G(t)) = F(u(t), v(t))$ for arbitrary values of $\mathbf{F}$. As above, let the sizes of the surface be m and n. Introduce $m \times n$ vectors $\mathbf{f}_I$ of size $m \times n$ such, that I-th vector has zero entries everywhere except at position I where $f_I = 1$. Then each $\mathbf{f}_I$ corresponds to coefficients of a 2-dimensional basis function of $F(u, v)$:

$$b_i^{k,\tau}(u)\, b_j^{l,\upsilon}(v) = b_I^{k,\tau,l,\upsilon}(u, v) = \mathbf{b}_{u,v}^T \mathbf{f}_I;\ I = i + jm$$

The multiplication of $\mathbf{A}$ by vector $\mathbf{f}_I$ yields $\mathbf{a}_I$ – the I-th column of $\mathbf{A}$. It follows that each column of $\mathbf{A}$ contains control points of the B-spline:

$$b_I^{k,\tau,l,\upsilon}(u(t), v(t)) = \mathbf{b}_\pi^T \mathbf{a}_I$$

In other words, each column of matrix $\mathbf{A}$ represents a decomposition of G onto basis functions of F. Each basis function is non-zero only in certain interval of the domain rectangle. From this we conclude that the number of outer-product updates involving one $\mathbf{a}_I$ will be proportional to the number of Gs segments which lie in the non-zero interval of $b_i^{k,\tau}(u)\, b_j^{l,\upsilon}(v)$. Therefore, in order to avoid accumulation of errors one should choose the curve G as simple as possible. Figure 4.6 on the right shows the graph of the error function

$$\epsilon_{ij}(t) = \int_t \left| b_i^{k,\tau}(u(t))\, b_j^{l,\upsilon}(v(t)) - \mathbf{b}_\pi^T \mathbf{a}_I \right| dt;\ I = i + jm \qquad (4.29)$$

for the domain curve G shown in figure 4.6 on the left; We see that even for such "unreasonable" curves the errors stay inside of acceptable limits. The sparsity pattern of the matrix can be predicted from this consideration. If G crosses the region where a particular basis function has influence, the corresponding column will have non-zero elements. Note that the non-zeros in each column will always occur in strips of certain length (G can "leave and enter" non-zero interval of a basis function several times).

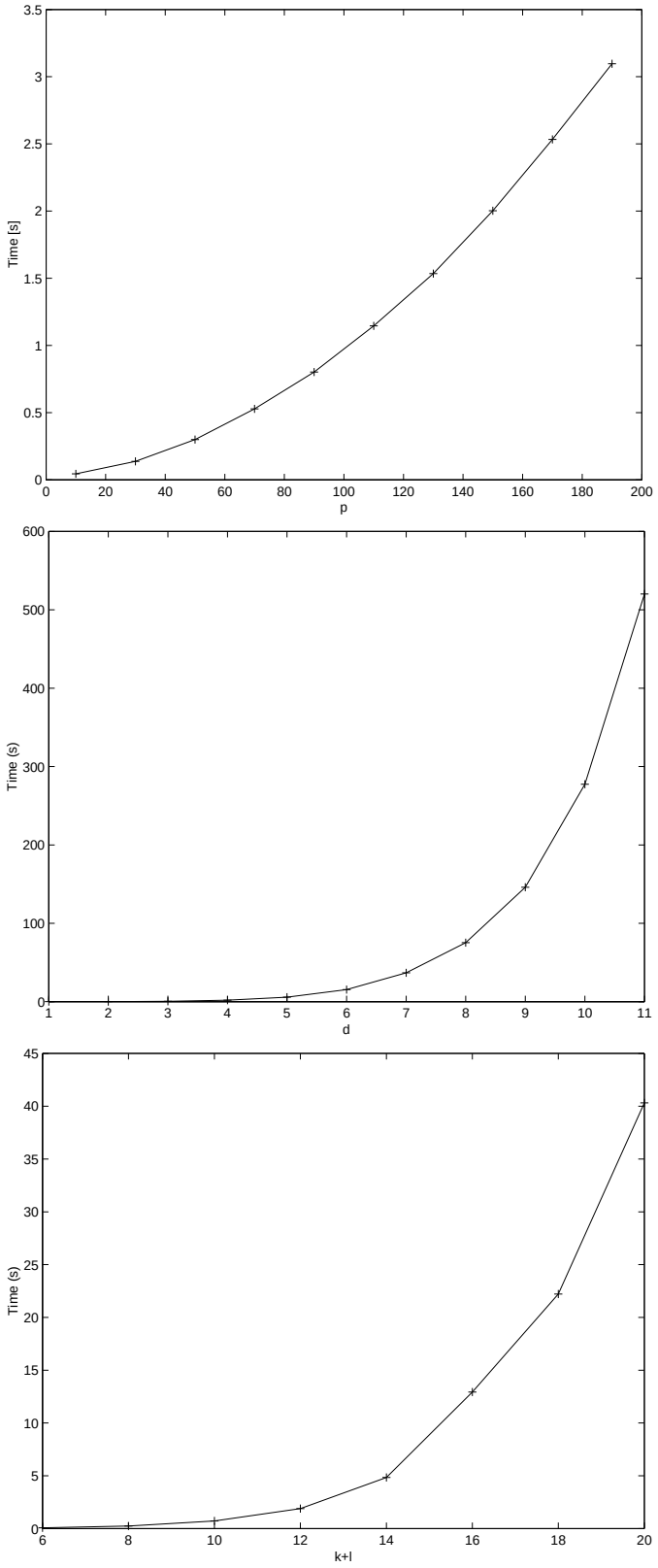


Figure 4.4: The time in seconds spent for computing the composition matrix in dependency on factors (top figure) p, $(d = 2, k + l = 6)$, (middle) d, $(p = 11, k + l = 6)$, (bottom) $k + l$, $(d = 2, p = 10)$.

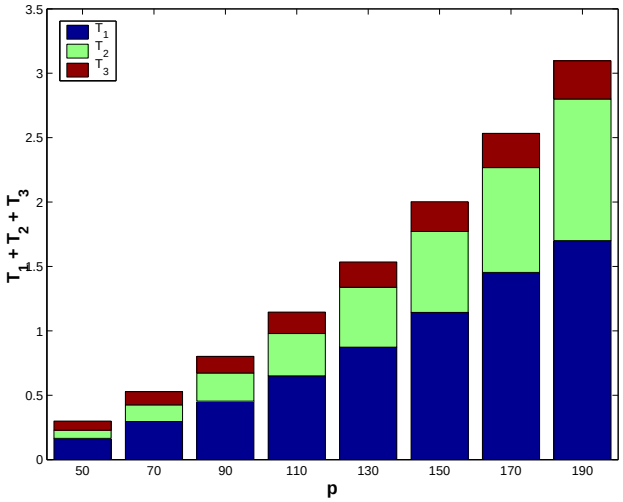


Figure 4.5: Comparison of run-times spent for computation of products (T_1), blossoms (T_2), and outer product updates (T_3).

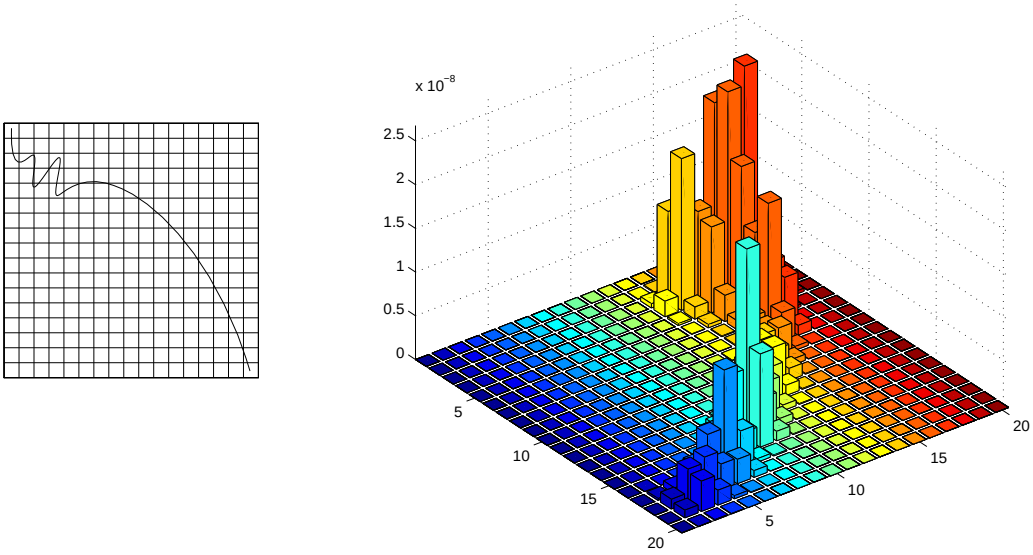


Figure 4.6: Element-wise error plot of the function $\epsilon_{ij}(t); \forall i, j$, as defined in eq. 4.29, for a cubic curve in the domain of a bi-cubic regularly parametrized B-spline surface with 20×20 control points. The upper Figures shows the curve G in the domain of the surface; the grid symbolizes the knot-lines of the surface. The errors are larger (but acceptable) at positions which are influenced by the rapidly varying part of the curve.

Chapter 5

Tangency constraints

This chapter covers tangency constraints with emphasis on efficient algorithmic implementation. We show how to enforce the tangent plane continuity of two surfaces meeting at an arbitrary curve incident on both surfaces. The coefficients of the system of linear equations in the unknown degrees of freedom of the surface are obtained using the curve-surface composition matrix.

5.1 Problem definition

The problem is illustrated in figure 5.1: given are the curve $G(t) = (u(t), v(t))$ in the domain of the B-spline surface $F(u, v)$, and the vector field curve $\vec{N}(t)$ which represents the required direction of the surface normals. The task is to determine the dependent control points of the surface F such that the surface normals along $F(G(t))$ are collinear with $\vec{N}(t)$.

The sufficient condition for a surface to interpolate the direction of given normals is that the scalar product of the following two curves is zero:

$$\left\langle \frac{d}{dt} F(G(t)), \vec{N}(t) \right\rangle = 0; \forall t \tag{5.1}$$

Here $\frac{d}{dt} F(G(t))$ denotes the differentiation w.r.t t; The first derivative function of a parametric curve the, so-called, hodograph and is obtained by applying the chain rule for differentiation to $F(G(t))$:

$$\frac{d}{dt} F(G(t)) = \frac{d}{dt} u(t) \cdot \frac{\partial}{\partial u} F_{u,v} + \frac{d}{dt} v(t) \cdot \frac{\partial}{\partial v} F(u, v)$$

The hodograph curve a linear combinations of partial derivatives of F which constitute the tangent plane at that position. Therefore, each vector perpendicular to the hodograph of $F(G(t))$ will also be perpendicular to the tangent plane of F.

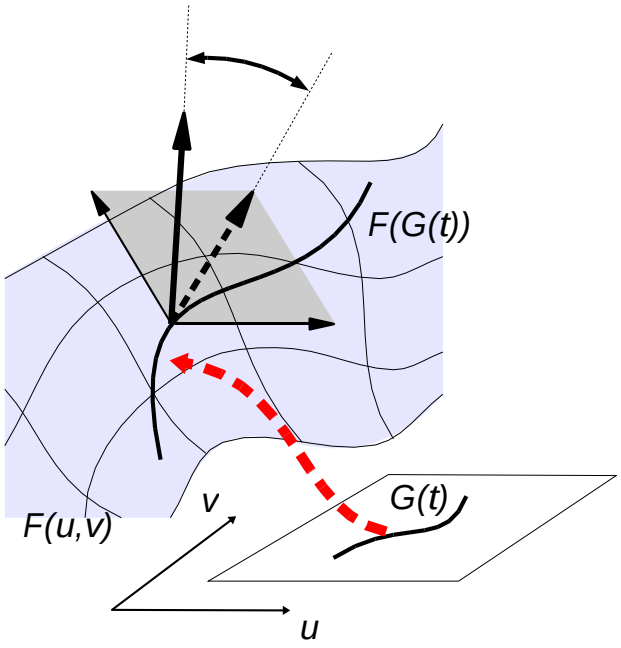


Figure 5.1: The tangency constraint: The task is to enforce orthogonality of the surface normals (this arrow) and the tangent of the surface curve $F(G(t))$ shown as dotted thick arrow.

Two vectors are perpendicular if their scalar product is zero, thus, the above condition will enforce the surface to take in desired direction along $F\left(G(t)\right)$. Scalar product of two B-spline curves is a B-spline function, see [23] or [24], for example. A B-spline function is zero everywhere, if all its coefficients are zero, hence, on the right-hand side of eq. 5.1 we must have a B-spline polynomial with all coefficients equal zero. We need to extract the relationship between thee control points of F and the zero coefficients of the scalar product function. Thus, we are looking for a matrix $\mathbf{B}$ such, that

$$\left\langle \frac{d}{dt} F\left(G(t)\right), \vec{N}(t) \right\rangle = 0 \Longleftrightarrow \mathbf{BF} = \mathbf{0}$$

5.2 Differentiation operator in matrix form

The first task is to express the hodograph curve $\frac{d}{dt} F\left(G(t)\right)$ as a function of the control points of surface F. We have shown in previous chapter that given the composition matrix for F and G the control points of $H = F\left(G(t)\right)$ can be found by $\mathbf{H} = \mathbf{AF}$. B-splines are closed under differentiation, i.e. a derivative of a (non-rational) B-spline curve or is again a B-spline curve of lower degree. Let be given

a B-spline basis $\mathbf{b}_t = \{b_i^{d,\tau}(t) : 0 \le i < n\}$ for the curve $H(t) = \sum_{i=0}^{n} b_i^{d,\tau}(t)\, H_i$. After few manipulations of the de Boor formula [41] the following recurrence yields the control points $H_i^{(r)}$ of the order-r hodograph of H:

$$H_i^{(r)} = \delta_i^r \cdot \left(H_{i+1}^{(r-1)} - H_i^{(r-1)}\right);\ 0 \le i < n - r$$
$$\delta_i^r = \tfrac{d-r+1}{\tau_{i+d+1} - \tau_{i+r}} \tag{5.2}$$

We write the first differentiation step (i.e. $r = 1$) in matrix format as follows: we set up a $(n-1) \times n$ matrix $\mathbf{D}^{(1)}$ containing the coefficients $-\delta_i^1$ and δ_i^1 in the i-th row at positions $i-1$ and i and apply it to the control points $\mathbf{H}$. This is done recursively r times, using the result from previous step yielding a sequence of matrices such that

$$\mathbf{H}^{(r)} = \underbrace{\mathbf{D}^{(r)} \cdots \mathbf{D}^{(1)}}_{\mathbf{D}^{(r)}} \mathbf{H}$$

For univariate B-splines the differentiation matrix $\mathbf{D}^{(r)}$ has $r+1$ entries in each row.

Matrices $\mathbf{D}^{(r)}$ and $\mathbf{A}$ have compatible sizes; hence we can write:

$$\mathbf{H}^{(r)} = \mathbf{D}^{(r)}\mathbf{H} = \mathbf{D}^{(r)}\mathbf{A}\mathbf{F}$$

which yields the required dependency between the r-th derivative of a surface curve $H = F\,(G(t))$ and DOFs of the surface. The degree and the size of the derivative curve reduces by one with each differentiation, i.e. the resulting B-spline has the the degree $d-r$ and size $n-r$. The knot vector of r-order derivative is obtained by truncating r knots at the beginning and at the end of the original knot vector. Hence the basis for the r-th order hodograph of the B-spline is $\mathbf{b}_t^{(r)} = \{b_i^{d-r,\tau'}(t) : 0 \le i < n - r\}$ where τ' denotes the "truncated" knot vector.

Differentiation in matrix form is also possible for B-spline surfaces. One proceeds exactly as in univariate case by differentiating each parametric direction separately. Two-variate differentiation matrices will be required in chapter 6.

5.3 Computing the scalar product

The symbolic scalar product of two curves, one in "evaluated" the other in "unevaluated" format is more difficult: given the vector fields curves $\vec{N}(t)$ and $\vec{T}(t) = \frac{d}{dt}F\,(G(t)) = \mathbf{D}^{(1)}\mathbf{A}\mathbf{F}$ we need to extract the dependency of their scalar product on the control points of the surface F. In the following, we assume without loss of generality, that both curves are defined on the same B-spline basis, i.e.

they have the same size, degree and knot vector.[1] The basis is denoted by the set of basis function $\{b_i^{d,\tau}(t) : 0 \leq i < n\}$, or in vector format $\mathbf{b}$ such that $\mathbf{b}_i = b_i^{d,\tau}(t) : 0 \leq i < n$.

First, we split both operands into their components in each spatial dimension; this yields in matrix notation

$$\begin{aligned} \vec{N}(t) &= \left[\begin{array}{ccc} n_x(t) & n_y(t) & n_z(t) \end{array} \right] \\ &= \mathbf{b}^T \left[\begin{array}{ccc} \mathbf{n}_x & \mathbf{n}_y & \mathbf{n}_z \end{array} \right] \end{aligned}$$

and

$$\begin{aligned} \vec{T}(t) = \tfrac{d}{dt} F\left(G(t)\right) &= \left[\begin{array}{ccc} t_x(t) & t_y(t) & t_z(t) \end{array} \right] \\ &= \mathbf{b}^T \left[\begin{array}{ccc} \mathbf{t}_x & \mathbf{t}_y & \mathbf{t}_z \end{array} \right] \\ &= \mathbf{b}^T \mathbf{D}^{(1)} \mathbf{A} \left[\begin{array}{ccc} \mathbf{f}_x & \mathbf{f}_y & \mathbf{f}_z \end{array} \right] \end{aligned}$$

According to previous paragraph the above formula applies the differential operator $\mathbf{D}^{(1)}$ to establish the connection between the derivative curve and the DOFs of the surface. The inner product of two curves is in this expanded notation:

$$\left\langle \vec{N}(t), \vec{T}(t) \right\rangle = \underbrace{n_x(t) \cdot t_x(t)}_{s_x(t)} + n_y(t) \cdot t_y(t) + n_z(t) \cdot t_z(t) \tag{5.3}$$

Since we already have the dependency between the control points of the hodograph curve and the control points of the surface, the next step will be to express a product of two B-spline functions in unevaluated format - as a function of $\mathbf{t}_x$, $\mathbf{t}_y$ and $\mathbf{t}_z$. We will consider only the under braced term $s_x(t)$ from the above equation. Note that, formally, a curve in matrix notation represents a product of two matrices. We can transpose this matrix product which yields

$$\left(\mathbf{b}^T \mathbf{n}_x\right)^T = \mathbf{n}_x^T \mathbf{b}$$

This allows us to write the term $s_x(t)$ in tensor product form which nicely separates the parameters $\mathbf{n}_x$ and the unknowns $\mathbf{t}_x$:

$$\begin{aligned} s_x(t) &= \mathbf{n}_x^T \left(\mathbf{b}\mathbf{b}^T\right) \mathbf{t}_x \\ &= \mathbf{n}_x^T \mathbf{P}_t \mathbf{t}_x \end{aligned} \tag{5.4}$$

[1] In principle, the procedure described here also works for different B-spline bases of $\vec{N}(t)$ and $\vec{T}(t)$, with a slightly more complicated notation.

The term $\mathbf{P}$ is a square symmetric matrix[2] with polynomial entries $p_{ij}(t) = b_i^{d,\tau}(t) \cdot b_j^{d,\tau}(t) : 0 \leq i,j < n$; i.e. the element $p_{ij}(t)$ is the product of i-th and j-th basis function of $\vec{N}(t)$ and $\vec{T}(t)$. Since entries of the vector $\mathbf{n}_x$ are constant scalar values we carry out the multiplication $\mathbf{n}_x^T \mathbf{P}_t$. This requires additional notation: the product of two B-spline basis function is a B-spline function of polynomial degree $2d$. Denote the coefficients of ij-th product by $\mathbf{p}^{ij}$ and its B-spline basis by $\mathbf{b}_\pi^T$. The degree of the product B-spline is $2d$. In the following we denote the knot vector of the product by π; it is determined as described in §4.2.3. Then the ij-th entry of $\mathbf{P}_t$ is a B-spline $\sum_{k=0}^{m_1} p_k^{ij} b_k^{2d,\pi}(t) = \mathbf{b}_\pi^T \mathbf{p}^{ij}$. Let m_1 be the size of the product B-spline $p_{ij}(t)$ and denote the size of the operand B-splines $\vec{N}(t)$ and $\vec{T}(t)$ by m_2. The product expands into

$$\begin{aligned} \mathbf{n}_x^T \mathbf{P}_t &= \mathbf{n}_x^T \begin{bmatrix} \mathbf{b}_\pi^T \mathbf{p}^{11} & \cdots & \mathbf{b}_\pi^T \mathbf{p}^{1m_1} \\ \vdots & \ddots & \\ \mathbf{b}_\pi^T \mathbf{p}^{m_1 1} & & \mathbf{b}_\pi^T \mathbf{p}^{m_1 m_1} \end{bmatrix} \\ &= \mathbf{b}_\pi^T \begin{bmatrix} \underbrace{\sum_{i=1}^{m_2} n_i^x \mathbf{p}^{i1}}_{\mathbf{x}_1} & \cdots & \underbrace{\sum_{i=1}^{m_2} n_i^x \mathbf{p}^{im_1}}_{\mathbf{x}_{m_2}} \end{bmatrix} \\ &= \mathbf{b}_\pi^T \mathbf{X} \end{aligned} \tag{5.5}$$

The matrix $\mathbf{X}$ consists of m_2 column vectors $\mathbf{x}_i : 1 \leq i \leq m_2$. Inserting the above result into eq. 5.4 and comparing compatible terms yields the control points of the x-component of the scalar product:

$$\begin{gathered} S_x(t) = \mathbf{b}_\pi^T \mathbf{s}_x = \mathbf{b}_\pi^T \mathbf{X} \mathbf{t}_x \Rightarrow \\ \mathbf{s}_x = \mathbf{X} \mathbf{t}_x \end{gathered}$$

The terms $s_y(t)$ and $s_z(t)$ in eq. 5.3 are obtained in exactly the same manner; we only replace $\mathbf{n}_x$ by $\mathbf{n}_y$ and $\mathbf{n}_z$ which yields the matrices $\mathbf{Y}$ and $\mathbf{Z}$. $s_x(t)$, and the corresponding terms $s_y(t)$ and $s_z(t)$, are B-spline polynomials on basis $\mathbf{b}_\pi$ with scalar control points $\mathbf{s}_x$, $\mathbf{s}_y$ and $\mathbf{s}_z$. Hence, we can write eq. 5.3 in matrix notation

$$\left\langle \vec{N}_t, \vec{T}_t \right\rangle = \mathbf{b}_\pi^T \cdot [\mathbf{s}_x + \mathbf{s}_y + \mathbf{s}_z] = \mathbf{b}_\pi^T \cdot \mathbf{s} \tag{5.6}$$

According to the stated orthogonality condition the vector $\mathbf{s}$ must have all entries equal zero. Thus, the equations which connect the control points of the tangent

[2] If $\vec{N}_t$ and $\vec{T}_t$ have different B-Spline bases the matrix $\mathbf{B}_t$ is neither square nor symmetric

curve $\mathbf{T}$ and the scalar product are:

$$\begin{aligned} \mathbf{0} = \mathbf{s} &= [\mathbf{s}_x + \mathbf{s}_y + \mathbf{s}_z] \\ &= [\mathbf{X}\mathbf{t}_x + \mathbf{Y}\mathbf{t}_x + \mathbf{Z}\mathbf{t}_x] \\ &= \begin{bmatrix} \mathbf{X} & \mathbf{Y} & \mathbf{Z} \end{bmatrix} \cdot \begin{bmatrix} \mathbf{t}_x \\ \mathbf{t}_y \\ \mathbf{t}_z \end{bmatrix} \end{aligned}$$

After replacing $\mathbf{t}_x$ (and, accordingly, $\mathbf{t}_y$ and $\mathbf{t}_z$) in the above equation by $\mathbf{t}_x = \mathbf{D}^{(1)}\mathbf{A}\mathbf{f}_x$ we obtain the relationship between control points of the surface and the scalar product:

$$\mathbf{0} = \begin{bmatrix} \mathbf{X} & \mathbf{Y} & \mathbf{Z} \end{bmatrix} \begin{bmatrix} \mathbf{D}^{(1)}\mathbf{A} & & \\ & \mathbf{D}^{(1)}\mathbf{A} & \\ & & \mathbf{D}^{(1)}\mathbf{A} \end{bmatrix} \begin{bmatrix} \mathbf{f}_x \\ \mathbf{f}_y \\ \mathbf{f}_z \end{bmatrix} \tag{5.7}$$

5.4 Practical notes on implementation

We see that the mathematical tools used in this section perfectly match our previous efforts. What remains, are, mainly, matrix operations. This part should not be ignored. Consider that the matrices $\mathbf{X}$, $\mathbf{Y}$, $\mathbf{Z}$ and consequently the matrix of the resulting linear system can be quite large: the sizes of exact curves $\vec{N}(t)$ and $\vec{T}(t)$ are in the order of the "height" of the composition matrix. The size of their product (reflected by the size of matrix $\mathbf{P}$) is order of $2d$ larger, where d is the degree of $\vec{N}$ and $\vec{T}$. The height of composition matrix $\mathbf{A}$ usually varies in order of several hundred rows, the number of columns is determined by the size of the surface. In eq. 5.7, the degrees of freedom of the surface in each spacial dimension are no longer independent – the height of the resulting matrix is, in general, factor $3.\,(2d)$ larger than the "usual" sizes of the composition matrix. If incidence and normal constraints are stated simultaneously – and in the most cases they are – the coordinates in each spacial dimensions have to be separated in both constraints. The full size of such systems quickly reaches thousand and more rows for one curve constraint. In order to speed up the computation we exploit the following properties:

Properties of B-spline products: When evaluating matrices $\mathbf{P}_t$ and $\mathbf{X}$, $\mathbf{Y}$, $\mathbf{Z}$ we utilize the properties of B-spline products analyzed in §4.3.2. If d is the polynomial degree of $\vec{N}(t)$ and $\vec{T}(t)$ then the matrix P_t is symmetric with upper bandwidth d – all products $p_{ij}(t)$ such, that $|i-j| \geq d+1$ are zero. The vectors $\mathbf{p}^{ij}$ are also sparse with predictable bandwidth which considerable accelerates the evaluation of linear combinations in eq. 5.5.

Approximation of $\vec{N}(t)$ by a lower degree curve: The polynomial degree of the exact normal curve $\vec{N}$ is quite high, in general, higher than the degree of exact $\vec{T}$. In practice, $\vec{N}$ can be usually approximated by a lower degree curve with less knots without significant loss of accuracy. Therefore, we apply degree reduction and knot removal to $\vec{N}(t)$ before entering the algorithm.

Faster implicit evaluation of auxiliary matrices: The matrices $\mathbf{A}_x$, $\mathbf{A}_y$, $\mathbf{A}_z$ and $\mathbf{D}^{(1)}$ (eq. 5.7) need not to be computed explicitly. Instead we have hard coded the evaluation of matrix products

$$\left(\mathbf{X}\mathbf{D}^{(1)}\right)\mathbf{A}$$

from eq. 5.7 such, that all three blocks of the final matrix are obtained simultaneously.

Chapter 6

Variational constraints

In this chapter we discuss efficient methods to obtain the Gaussian normal equations resulting from the, so-called, quadratic surface functionals. The most frequently used surface functionals are listed. The structure of these expressions is analyzed which results in a unified algorithmic approach to obtaining the Gaussian normal equations for all types of functionals in an elegant and efficient manner. In particular, we discuss hierarchical decomposition of derivatives of B-splines and fast methods for integrating products of B-splines. Results regarding the run-time performance are presented.

6.1 Quadratic error functionals for surfaces

The problem is stated as follows: given is a B-spline surface

$$S(u,v) : \Omega \rightarrow R^3,\ \Omega \subset R^2$$

determined by its control points $\mathbf{S} \in R^{n\times 3}$. One considers an objective function

$$f : R^3 \rightarrow R^1,\ f = f\left(S(u,v)\right)$$

and seeks values for the control points such that f takes in minimal value over the entire domain of the surface. This is achieved by minimizing a surface functional

$$\Phi(\mathbf{S}) = \int_\Omega f\left(S(u,v)\right) \partial u \partial v$$

with respect to degrees of freedom of the surface. Generally, one proceeds as follows:

- Select an appropriate type of function f (see below)

- Set up the equations which restrict the functional Φ to take in minimal value; We require that the derivatives of Φ with respect to the degrees of freedom of the surface must equal zero. This yields n equations in n unknown control points:
$$\frac{\partial}{\partial S_i}\Phi(S_0,\ldots,S_n) = 0,\ 0 \leq i < n \tag{6.1}$$
- Find the zero set of 6.1 yielding the required control points of the surface.

A frequently used approach is to minimize functionals consisting of squared sums of mixed surface derivatives, see [32], for example. These so-called *quadratic surface functionals* have the advantage that the normal equations 6.1 are linear in the control points of the surface.

The most frequently used functionals are shown in table 6.1. In the table and throughout the following text we use a simplified notation for partial derivatives of a B-spline surfaces. We define a differentiation operator

$$D^{r,s} : \Omega \rightarrow R^3,\ D^{r,s}S(u,v) = \frac{\partial^{r+s}}{\partial u^r \partial v^s} S(u,v)$$

with the convention that for $r = s = 0$ the differential operator $D^{0,0}$ equals the identity transformation, i.e. it has no effect on $S(u,v)$. Using this notation, the left column of table 6.1 shows the orders of required surface derivatives. The rightmost column describes the property of the surface which is minimized by the expression in the middle column.

6.2 Matrix notation for surface functionals

Consider, for example, the functional which minimizes the thin plate energy of a surface $S(u,v)$. The expression to be integrated is:

$$f(u,v) = \left(D^{2,0}S(u,v)\right)^2 + \left(D^{1,1}S(u,v)\right)^2 + \left(D^{0,2}S(u,v)\right)^2 \tag{6.2}$$

Other quadratic functionals are variations of the above expression: they arise from the summation of squared partial derivatives of orders 1, 2 or 3. Often, linear combinations of these expressions are used.

We break down all quadratic functionals of the this type into summations of terms

$$\int_u \int_v \left(D^{r,s}S(u,v)\right)^2 \partial u \partial v;\ 0 \leq r \leq 3,\ 0 \leq s \leq 3 \tag{6.3}$$

We need to compute definite integral of the square of $D^{r,s}S(u,v)$ and extract the relation between its value and the control points of the surface. This can be accomplished easily if we re-write equation 6.3 in matrix format:

r, s	$\Phi = \int_\Omega f\left(S(u,v)\right)\, \partial u \partial v$	Property
$r, s = \{(1,0),(0,1)\}$	$f = \left(D^{1,0}S(u,v)\right)^2 + \left(D^{0,1}S(u,v)\right)^2$	Area
$r, s = \{(2,0),(1,1),(0,2)\}$	$f = \left(D^{2,0}S(u,v)\right)^2 + \left(D^{1,1}S(u,v)\right)^2 \left(D^{0,2}S(u,v)\right)^2$	Thin plate energy
$r, s = \{(2,0),(0,2)\}$	$f = \left(D^{2,0}S(u,v) + D^{0,2}S(u,v)\right)^2$	Mean curvature
$r, s = \{(3,0),(2,1),(1,2),(0,3)\}$	$f = \left(D^{3,0}S(u,v) + D^{2,1}S(u,v)\right)^2 + \left(D^{0,3}S(u,v) + D^{1,2}S(u,v)\right)^2$	Variation of curvature

Table 6.1: Frequently used quadratic surface functionals which can be easily obtained by the presented algorithm.

Let $\{b_i^{k,\tau}(u)b_j^{l,\upsilon}(v) : 0 \le i < m,\, 0 \le j < n\}$ be the set of 2-variate basis functions of the surface $S(u,v)$. The partial derivative of a B-spline surface is obtained by differentiating each basis function with respect to variables u and v:

$$D^{r,s}S(u,v) = D^{r,s}\sum_{j=0}^{n}\sum_{i=0}^{m} S_{ij}b_i^{k,\tau}(u)b_j^{l,\upsilon}(v) = \sum_{j=0}^{n}\sum_{i=0}^{m} S_{ij}D^r b_i^{k,\tau}(u)D^s b_j^{l,\upsilon}(v)$$

Next, we define a vector $\mathbf{l}_{u,v}^{(r,s)}$ as follows:

$$l_I^{r,s}(u,v) = D^r b_i^{k,\tau}(u)D^s b_j^{l,\upsilon}(v) : I = i + mj,\, 0 \le i < m,\, 0 \le j < n$$

The square of the surface derivative becomes in matrix format:

$$\left(D^{r,s}S(u,v)\right)^2 = \mathbf{S}^T \underbrace{\mathbf{l}_{u,v}^{r,s}\left(\mathbf{l}_{u,v}^{r,s}\right)^T}\mathbf{S}$$

The under braced term in the above equation represents an outer product of two vectors, which yields a square matrix of the same size. We denote it by

$$\mathbf{L}_{u,v}^{r,s} = \mathbf{l}_{u,v}^{r,s}\left(\mathbf{l}_{u,v}^{r,s}\right)^T$$

The entries of $\mathbf{L}_{u,v}^{r,s}$ are two-variate B-spline functions

$$l_{IJ}^{r,s}(u,v) = \left(l_I^{r,s}(u,v)\right) \cdot \left(l_J^{r,s}(u,v)\right), \; 0 \leq I < mn, \; 0 \leq J < mn \tag{6.4}$$

Substituting into eq. 6.3 yields

$$\begin{aligned} &\int_u \int_v \left(D^{r,s}S(u,v)\right)^2 \partial u \partial v = \\ &\quad \int_u \int_v \left(\mathbf{S}^T \mathbf{L}_{u,v}^{r,s} \mathbf{S}\right) \partial u \partial v = \\ &\quad \mathbf{S}^T \left(\int_u \int_v \mathbf{L}_{u,v}^{r,s} \partial u \partial v\right) \mathbf{S} = \mathbf{S}^T \left(\mathbf{L}^{r,s}\right) \mathbf{S} \end{aligned} \tag{6.5}$$

The matrix $\mathbf{L}^{r,s} \in R^{mn \times mn}$ contains the values of definite integrals from eq. 6.4 evaluated over the domain of the surface. I.e. the IJ-th entry of $\mathbf{L}^{r,s}$ is obtained by computing:

$$l_{IJ}^{r,s} = \int_u \int_v l_{IJ}^{r,s}(u,v) \partial u \partial v \tag{6.6}$$

Equation 6.5 is a quadratic form in $\mathbf{S}$, hence, its normal equations are:

$$\mathbf{L}^{r,s}\mathbf{S} = \mathbf{0}$$

Setting $r, s = \{(2,0),(1,1),(0,2)\}$ yields the terms which are necessary to assemble the normal equations for the thin plate energy (TPE) minimizing functional, eq. 6.2:

$$\begin{aligned} &\mathbf{L}^{2,0}\mathbf{S} + \mathbf{L}^{1,1}\mathbf{S} + \mathbf{L}^{0,2}\mathbf{S} = \\ &\qquad \left(\mathbf{L}^{2,0} + \mathbf{L}^{1,1} + \mathbf{L}^{0,2}\right)\mathbf{S} = \mathbf{L}_{TPE}\mathbf{S} = \mathbf{0} \end{aligned} \tag{6.7}$$

The other surface functionals are obtained analogously: we just set the values r and s according to table 6.1, obtaining the matrices $\mathbf{L}^{r,s}$. Hence, an efficient implementation of this algorithm is required for different values of r and s.

6.3 Implementation

Our implementation is based on following ideas:

- We compute all combinations of products of B-splines required in eq. 6.4. We integrate in each variable separately and multiply the values of univariate integrals according to formula 6.4
- We use the recursive definition of the derivative of a B-spline basis function which defines a derivative of a B-spline as a linear combinations of B-splines of lower degree. We decompose each B-spline that way obtaining a closed formula which delivers the product of B-spline derivatives of required order.

- We integrate products of B-splines symbolically: we compute the B-spline function which represents the product of two B-splines and determine its antiderivative function. This is not the fastest method, but it is numerically stable and sufficiently fast for our application.

The advantage of this approach is that the computation all quadratic functionals is put on a common basis; basically, we compute the matrices $\mathbf{L}^{r,s}$ for required order of derivatives, see eq. 6.5. For example, derivatives of orders $0 \leq r \leq 2$ and $0 \leq s \leq 2$ are required for the TPE-minimizing functional. Then, the normal equations of a specific functional are obtained simply by matrix addition, see, for example, equation 6.7 in previous paragraph.

6.3.1 Computing two-variate integrals of B-splines

In order to compute the two-variate integrals needed in eq. 6.4, we first compute the univariate products of the basis functions in each variable. That is, eq. 6.6 decomposes into:

$$\int_u \int_v l_{IJ}^{r,s}(u,v)\partial u \partial v = \left(\int_u D^r b_{i_1}^{k,\tau}(u) D^r b_{i_2}^{k,\tau}(u) du \right) \left(\int_v D^s b_{j_1}^{l,\upsilon}(v) D^s b_{j_2}^{l,\upsilon}(v) dv \right) \tag{6.8}$$

Thus given the orders of derivatives in each parametric direction the operands of the above product are computed and stored for all combinations of indices $[i_1, i_2] : 0 \leq i_1 < m$, $0 \leq i_2 < m$, and $[j_1, j_2] : 0 \leq j_1 < n$, $0 \leq j_2 < n$. The value of the two-variate integral, i.e. the IJ-entry of the matrix $\mathbf{L}^{r,s}$ is obtained by retrieving and multiplying the univariate terms corresponding to indices $I = i_1 + mj_1$ and $J = i_2 + nj_2$.

6.3.2 Hierarchical decomposition of B-spline derivatives

Next, we will show how to compute the coefficients of a r-times differentiated B-spline basis function which is required in formula 6.8. The r-th derivative of a B-spline basis function of degree d is defined by the following recurrence:

$$D^r b_i^{d,\tau}(t) = \delta_{i-1}^r \cdot D^{r-1} b_{i-1}^{d-1,\tau}(t) - \delta_i^r \cdot D^{r-1} b_i^{d-1,\tau}(t),\ 0 \leq i < n - r \tag{6.9}$$

where

$$\delta_i^r = \frac{d - r + 1}{\tau_{i+d+1} - \tau_{i+r}}$$

with the convention that for $i - 1 < 0$, $\delta_{i-1}^r = 0$. This can be shown easily by means of eq. 5.2. To compute $D^r b_i^{d,\tau}(t)$ one sets:

$$b_i^{d,\tau}(t) = \sum_{j=0}^{n} h_j b_j^{d,\tau}(t), \quad h_j = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

The differentiation formula for B-splines (eq. 5.2) is applied which immediately yields eq. 6.9. The knot-vector of the differentiated B-spline is determined as explained in §5.2. Thus a derivative of a B-spline basis function is a B-spline function with coefficients determined from linear combinations of the δ-factors; For example, the first derivative becomes

$$D^1 b_i^{d,\tau}(t) = \delta_{i-1}^1 b_{i-1}^{d-1,\tau}(t) - \delta_i^1 b_i^{d-1,\tau}(t) = \sum_{j=i-1}^{i} \gamma_{j,i}^1 b_j^{d-1,\tau}(t)$$

which is a B-spline function from $\mathcal{P}_{d-1,\tau}$ with coefficients

$$\gamma_{j,i}^1 : 0 \leq j < n-1 = \begin{cases} \delta_{i-1}^1 \text{ if } j = i-1 \\ -\delta_{i-1}^1 \text{ if } j = i \\ 0 \quad \text{otherwise} \end{cases}$$

The task is to compute the coefficients $\gamma_{j,i}^r : 0 \leq j < n-r$ for all basis functions $b_i^{d,\tau}(t) : 0 \leq i < n$. The following consideration makes this very efficient: The decomposition advised by the formula 6.9 yields the a triangular scheme with r-levels:

$$\begin{array}{ccccc} & & & & \delta_{i-r}^1 b_{i-r}^{d-r,\tau} \\ & & & \delta_{i-2}^{r-1} D^{r-2} b_{i-2}^{d-2,\tau} & \\ & & & & \cdots \\ & & \delta_{i-1}^r D^{r-1} b_{i-1}^{d-1,\tau} & \delta_{i-1}^{r-1} D^{r-2} b_{i-1}^{d-2,\tau} & \\ & & & & \\ D^r b_i^{d,\tau} & \delta_i^r D^{r-1} b_i^{d-1,\tau} & & \delta_i^{r-1} D^{r-2} b_i^{d-2,\tau} & \delta_i^1 b_i^{d-r,\tau} \end{array}$$

The decomposition proceeds from left to right: the left-most vertex of the triangle is the required B-spline derivative. It decomposes into linear combinations of B-splines of lower degree as prescribed by eq. 6.9. We proceed in this manner until the differentiation stops on the right side of the triangle where the B-spline basis functions which constitute the basis of $D^r b_i^{d,\tau}(t)$ are found. We see that the terms from lower levels are re-used for computing several neighboring derivatives. For example, if one draws the triangle for $D^r b_{i-1}^{d,\tau}$ the term $\delta_{i-1}^r D^{r-1} b_{i-1}^{d-1,\tau}$ and its ancestors will also be contained in its decomposition. Traversing the triangle in the other direction (from right to left) we obtain the coefficients of the derivative B-spline by collecting the appropriate δ terms. This yields the coefficients of the derivative B-spline $D^r b_i^{d,\tau}(t)$ defined in terms of basis functions of degree $d-r$:

$$D^r b_i^d(t) = \sum_{j=i-r}^{i} \gamma_{j,i}^r b_j^{d-r}(t)$$

where coefficients $\gamma_{j,i}^r$ are determined for given i, d and r by collecting the corresponding factors δ during the traversal. Finally, the substitution into eq. 6.8 yields:

$$\begin{aligned}
&\int_t \left(D^r b_{i_1}^{d,\tau}(t) \cdot D^r b_{i_2}^{d,\tau}(t) \right) dt = \\
&\int_t \left(\sum_{k=i_1-r}^{i_1} \gamma_{k,i_1}^r b_k^{d-r,\tau}(t) \right) \left(\sum_{l=i_2-r}^{i_2} \gamma_{l,i_2}^r b_l^{d-r,\tau}(t) \right) dt = \\
&\sum_{k=i_1-r}^{i_1} \sum_{l=i_2-r}^{i_2} \gamma_{k,i_1}^r \gamma_{l,i_2}^r \left(\int_t b_k^{d-r,\tau}(t) b_l^{d-r,\tau}(t) dt \right)
\end{aligned} \tag{6.10}$$

6.3.3 Integrating products of B-splines

It remains to mention how to integrate products of B-splines. This is profoundly discussed in [81], for example. Basically, there are three possibilities:

1. Evaluate the product using one of the three methods from section 4.3.2, compute the antiderivative function of the resulting B-spline [68, 23], and evaluate the definite integral.

2. Use the recurrence for integrating a product of B-splines as described by De Boor, Lyche and Schumaker in [6].

3. Integrate by parts, as proposed by authors of [81]. This method is the fastest of the three but it suffers from numerical instability for B-splines with high degree and dense and uneven knot vectors.

The discussion in the abovementioned papers concludes that the first and second methods are numerically stable but slow. Our experiments with the third method (introduced in that work) have shown that the integration by parts tends to be numerically unstable already for B-splines of relatively low degree (such as 4 or 5) and for B-splines with irregularly spaced knot vectors. Since such B-spline occur very frequently in our application we have decided to use the first method.

The 1-st method is the easiest from the implementation point of view – if efficient symbolic computation of B-spline products is available. We apply the apparatus developed in chapter 4, §4.3.2: the product of B-splines contained in the last term of eq. 6.10 has following properties:

1. it is symmetric, i.e. $\left(\int_t b_k^{d-r,\tau}(t) b_l^{d-r,\tau}(t) dt \right) = \left(\int_t b_l^{d-r,\tau}(t) b_k^{d-r,\tau}(t) dt \right)$

2. it is local: $\int_t b_k^{d-r,\tau}(t) b_l^{d-r,\tau}(t) dt \begin{cases} \neq 0 \text{ if } \max k,l - \min k,l \leq d-r \\ = 0 \text{ otherwise} \end{cases}$

Thus only a fraction of all possible combinations of products will need to be computed. We obtain the product B-spline functions, compute their anti-derivative function according to formula for integration of B-splines (see [41, 25, 23], for example) and evaluate the definite integrals. Finally, the matrix representing the normal equations of a specific surface functional is obtained by back-substitution into eq. 6.8 and addition of the intermediate matrices $\mathbf{L}^{r,s}$.

6.4 Results and practical notes

We conclude with an example of the run-time performance of the algorithm for typical input parameters. Although, theoretically, our method is not the fastest one using the properties of combinations of B-spline products it performs very well even for very dense surfaces. Typically, the degrees of the involved B-splines will be low. The important criterion is how the increasing number of control points (dimension of the spline spaces $\{b_i^{k,\tau}(u)b_j^{l,v}(v) : 0 \leq i < m, 0 \leq j < n\}$) will influence the performance of the algorithm; this is shown in fig. 6.1 on the top. The dependency on the degree of the B-spline basis (the univariate case) is shown in the lower figure 6.1 – we see that the consumed time grows very rapidly.

Note: The expression $\int_t b_i^{d,\tau}(t).b_j^{d,\tau}(t)dt$ represent the algebraic scalar product of two B-spline functions. In certain applications it is useful to have an efficient implementation of this operation at hand. For example, in [45], Kazinnik and Elber have used wavelet-based decomposition of B-splines in order to get a multi-resolution representation of a curve or surface, for which scalar products of B-Splines are required. Further applications are e.g. continuous approximation or symbolic evaluation of continuous L_2 norms of B-Spline functions.

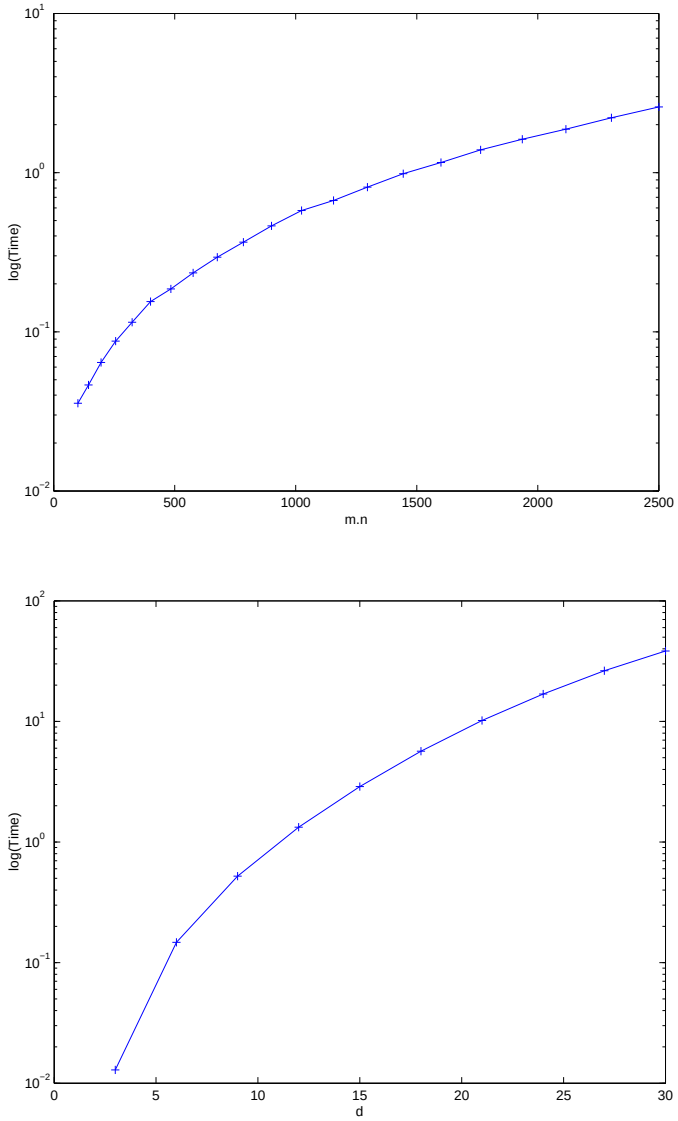


Figure 6.1: The upper figure shows the run-time cost for computing the matrix $\mathbf{L}_{TPE}$ (eq. 6.2) in dependency on the size of a b-cubic B-spline surface. The lower displays the time spent for computing the integrals of univariate products (eq. 6.8) in dependency on the degree of the B-spline basis with constant size $n = 30$. The graphs are plotted in semi-logarithmic scale on the y-axis.

Chapter 7

Linear constraint solving I

In this chapter we discuss the problem of solving the linear constraint problems defined in terms of equations obtained in chapters 5, 6 and 7. The problem to determine a surface from one or several arbitrary curves generally belongs to the category of, so-called, *ill-posed* inverse linear problems. Although the theory of ill-posed problems is well developed in the literature, it is rarely applied for solving surface interpolation problems. One rather avoids to state such problems at all – which leads to considerable restrictions on surface topology and the shape of permitted curve constraints. This chapter demonstrates that under certain assumptions ill-posed surface interpolation problems can be safely solved. Section 7.2 briefly reviews the theory of ill-posed problems and introduces the idea of "regularization". We have used a powerful algebraic tool for solving ill-conditioned system of equations, the Singular Value Decomposition (SVD). In section 7.3 we describe a SVD-based method which safely reveals the rank of the ill-conditioned system. Here, the, so-called, *surface aliasing effect* is addressed which occurs whenever a B-spline surface is deformed along general curves. Two "anti-aliasing methods" which suppress or completely remove the aliasing are described in chapter 8.

7.1 Notation

In this and the following chapters a slightly different notation for linear systems of equations will be used. A system of linear equations is defined by a mapping:

$$\mathbf{A} : R^m \rightarrow R^n,\ \mathbf{Af} = \mathbf{h},\ \mathbf{A} \in R^{m \times n},\ \mathbf{f} \in R^m,\ \mathbf{h} \in R^n \tag{7.1}$$

A solution of the linear system is a defined as inverse mapping:

$$\mathbf{A}^{-1} : R^n \rightarrow R^m,\ \mathbf{A}^{-1}\mathbf{h} = \mathbf{f},\ \mathbf{A} \in R^{n \times m} \tag{7.2}$$

The change consists in using solely vectors to denote the unknowns and the parameters of a linear system. The reason for this change is to have a unified notation for all types of constraints: generally, we simultaneously solve systems of equations consisting of incidence, tangency and variational constraints. Moreover we wish to solve several curve constraints for *one* surface simultaneously. Thus given a surface and constraints affecting it the actual constraint solving is done in three steps:

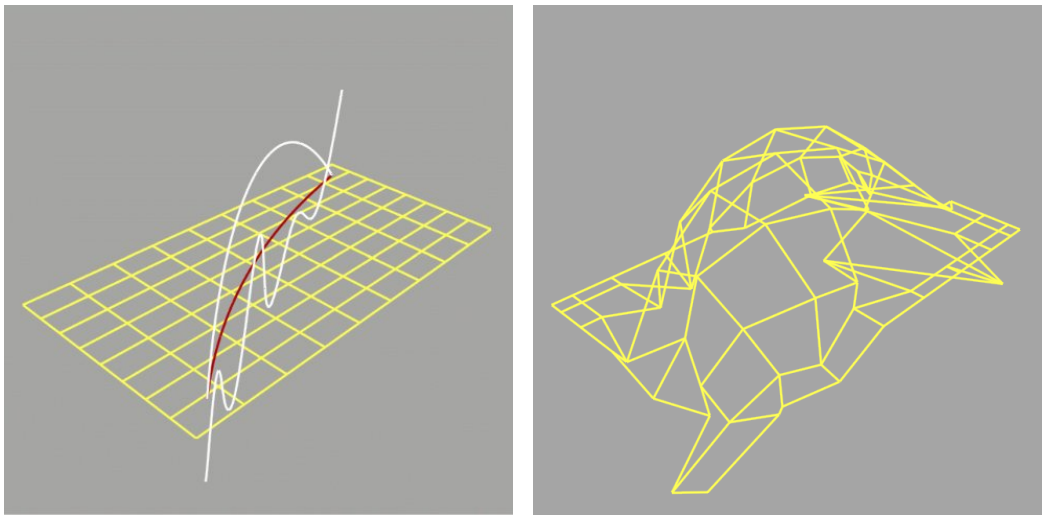
1. Compute the matrices for required constraints.
2. Concatenate the matrices and parameters vertically yielding the global matrix $\mathbf{A}$ and the global rights hand side vector $\mathbf{h}$ in eq. 7.1.
3. Solve the linear system (eq. 7.2) obtaining the control points of the required surface $\mathbf{f}$.

If only incidence constraints are used, the equation system can be solved independently for each spacial dimension. I.e., in eq. 7.1 the vectors $\mathbf{f}, \mathbf{h}$ are simply replaced by the matrices $\mathbf{F}, \mathbf{H}$ (or the vertical concatenation of $\mathbf{H}$s in case of several incidence constraints) in our usual notation: each column corresponds to coordinates in one spacial dimension and the system can be solved for several parameter vectors (corresponding to x, y and z coordinates) simultaneously. If tangency and variational constraints are to be solved the spacial dimensions are no longer independent. Then the vector $\mathbf{f}$ contains the x, y and z coordinates of the surface control points concatenated vertically as explained in chapter 5.

7.2 Ill-posed problems

The properties of an *ill-posed inverse linear problem* and the difficulties associated with solving of such problem are easily demonstrated by an example: Consider a bi-cubic B-spline surface as shown in figure 7.1 with one curve constraint. The original shape of the curve is shown in dark color. The curve is modified two times (the bright curves). We seek the control points of the surface such that the modified curves are interpolated.

In this example, the surface has 8×10 DOFs, the curve constraint is defined by a quadratic B-spline curve with 8 control points in the domain of the surface yielding an exact surface curve of degree 12 and with 72 control points. The equation system which defines the curve-surface incidence consists of 72 equations with 80 unknowns; The unknowns are the control points of the surface and the parameters are the control points of the modified curve.



(a) (b)

Figure 7.1: Demonstrating the ill-posedness of a single curve constraint on a bi-cubic B-spline surface with 80 DOFs. (a): Initial shape of surface, the surface curve H (dark) and two request curves shown in bright gray. One of the curve is varying too rapidly and does not satisfy the discrete Picard condition. (b): The un-regularized solution. Although the surface satisfies the constraint (the smooth bright curve in the left figure) from the shown control mesh we conclude that this surface is useless.

The task to solve the associated inverse linear problem. Clearly, the problem to compute a surface from this curve will be under-determined regardless of the actual size of the system matrix. Note that we could have inserted arbitrary many control points into the domain curve which would have increased the "height" of the system matrix. These additional equations are superfluous; there is only a certain number of variables which are determined by this constraint. Thus, the tasks will be:
(1) To identify the subset of equations which safely render the dependency between the control points of the curve and the surface
(2) to extract those variables (control points) which can be safely determined.

Thus assume we have set up a linear system 7.1 and wish to obtain its solution defined by eq. 7.2. The simplest possibility is to apply pivoted Gaussian elimination to the normal equations of the above system obtaining a generalized solution

$$\mathbf{f} \quad = \quad \mathbf{f}_p + \mathbf{f}_n \tag{7.3}$$

where the first term denotes the subset of those variables (control points) which are uniquely determined by the specified curve constraint and the second term denotes an arbitrary translation factor from the null-space of $\mathbf{A}$. In practice, the surface editing process is an ordered sequence of steps with an intermediate surface in each, thus the term $\mathbf{f}_n$ can be set according to surface from previous editing step. The solution resulting from Gaussian elimination with pivoting threshold 10^{-4} (which is a very "tolerant" value, much greater than the precision provided by the floating point unit of the computer) is shown in fig. 7.1(b): Although only the control mesh of the surface is shown, we may conclude that this surface is worthless: obviously, we have missed the right moment to stop the elimination process – the selected pivoting threshold was too small which has caused an unproportional growth of certain components in $\mathbf{f}$.

Although one can experiment with different pivoting strategies or with setting the pivoting threshold to larger values, in general, Gaussian elimination is not well suited to solve such sets of equations. Apparently, certain equations are close to be linearly dependent, but the actual rank of the matrix cannot be reliably determined. The equation system exhibits the typical features of an *ill-posed* problem:

- *The condition number is high*. Algebraically, the condition of a linear equation system is expressed as amount of change in the normed solution $\|\mathbf{f}\|$ in dependence of the changes in parameters $\mathbf{h}$ [80, Part III]: an inverse linear problem eq. 7.2 is well-conditioned if $\|\mathbf{A}^{-1}\|$ is a continuous function of $\mathbf{h}$ – which means that small perturbations of $\mathbf{h}$ lead to small changes of $\|\mathbf{f}\|$. Otherwise it is ill-conditioned, or ill-posed. Figure 7.1 demonstrates that: even a small deformation of the constrained curve, in this case, shown as the smoother bright curve in figure 7.1(a), causes randomly occurring "wiggles" in the control mesh of the surface.

- *The rank of the system matrix is a "noisy" number*. This property makes the difference between just ill-conditioned and an ill-posed problem. E.g., when applying pivoted Gaussian elimination one will not encounter a pivot which is particularly smaller than the pivots in previous elimination steps (in most cases this would happen when the problem is "only" ill-conditioned). The values of pivots gradually decay to a small number; hence, there is no obvious criterion when to stop the elimination.

7.2.1 The Picard condition

The ill-posedness of the linear system is caused by nearly linearly dependent rows (and columns) in the system matrix; in other words, the chosen linear model, here obtained from the composition of the B-splines spaces of the surface and

the curve, is not able to safely determine the dependency between the parameters and the unknown variables. There is a more precise statement than this, the so-called "Picard condition", see, e.g. [34], [36]: The modified right-hand side of the equations system

$$\mathbf{A}\mathbf{f} = \mathbf{h}', \;\; \mathbf{h}' = \mathbf{h} + \Delta\mathbf{h}$$

must be "smooth enough to survive the inversion to $\mathbf{f}$" [36]. It follows that we cannot expect to obtain a meaningful solution for arbitrary shape of the curve (determined by its control points $\mathbf{h}'$). For example, it is very unlikely that a reasonable surface (with given parametrization and number of control points) which interpolates the rapidly varying curve shown in figure 7.1(a) will exist. We will demonstrate the difference between these two cases in more detail in § 7.4.

7.2.2 Regularization of ill-posed problems

The branch of linear algebra which deals with solving of ill-posed problems is called *regularization*. Ill-posed problems occur in many areas of science and engineering; the typical example of anticipated ill-posedness are the Fredholm integral equations [34] – the vast majority of available literature concentrates on methods to solve this type of equations. A book which provides a lot practical examples associated with practical problems in physics is [69]; Many researchers have sought for efficient and secure methods to obtain a solution of ill-conditioned problems; we name only few: Golub, Stewart, O'Leary et al. [28, 26, 27] and Hansen et al., [35, 12, 38, 37, 10]. One rarely finds a paper which does not deal with Fredholm equations, nevertheless, the methods are (to certain extent) applicable to arbitrary ill-posed problems. One reference with connection to CAGD and B-spline surface interpolation, is Schumaker's paper [72]. A fabulous reference is Hansen's survey on regularization methods [36].

Essentially, ill-posed problems are under-determined due to the uncertainty about the exact rank of the matrix. The consequence is that that standard methods for solving inverse linear problems (such as Gaussian elimination or QR factorization) cannot be used in straightforward manner to obtain a solution. The basic idea of regularization is to remedy this uncertainty by stating an additional condition on properties of the solution in order to improve the rank of the system matrix. Technically, one selects a so-called "side constraint" which represents some expected properties of the solution. Then we seek a compromise between minimizing the residual of the ill-posed equation system and an "optimal" value of the side constraint. Clearly, once a problem is identified as ill-posed and a side constraint is introduced, we have to give up the goal to solve the problem exactly, i.e., we will, in general, not be able to make the residual equal zero. Instead, we hope to obtain a *regularized* solution which is not too far away from the actual

requirement that the residual equals zero.

The main difficulty of regularization is to find a good compromise between the error in the residual and value of the side constraint. In this thesis we will apply discrete regularization methods based on *Singular Value Decomposition* of the model matrix.

7.3 The truncated SVD

The superior tool for analysis of ill-posed problems is the Singular Value Decomposition (SVD) [80, 29, 46, 84]. In this section we will briefly review the properties of SVD. We will explain how the truncation of the SVD helps to reveal the rank of the degenerate system of linear equations.

7.3.1 The SVD

Let be given a system of linear equations:

$$\mathbf{A} : R^m \rightarrow R^n,\ \mathbf{Af} = \mathbf{h},\ \mathbf{A} \in R^{m \times n},\ \mathbf{f} \in R^m,\ \mathbf{h} \in R^n$$

The SVD of system matrix $\mathbf{A}$ has the format $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$. The matrices $\mathbf{U}$, $\mathbf{V}$ are orthonormal and $\mathbf{\Sigma}$ is a diagonal matrix such that

$$\text{diag}(\Sigma) = \sigma_i \geq \sigma_{i+1} \ldots \sigma_{\min(m,n)} \geq 0$$

For well-conditioned non-singular problems all singular values σ_i are non-zero and the condition number $\kappa = \sigma_1 / \sigma_{\min(m,n)}$ is small. In case of well-conditioned singular problem, there is a particular k such that $\sigma_k \gg \sigma_{k+1}$ and $\sigma_i \approx 0$ for $i \geq k+1$. The "truncation" parameter k corresponds to the rank of the singular problem. Consider the following partitioning of the SVD matrices:

$$\mathbf{U}_k = [\mathbf{u}_1 \ldots \mathbf{u}_k] \quad \mathbf{\Sigma}_k = \text{diag}\,(\sigma_1, \ldots, \sigma_k) \quad \mathbf{V}_k = [\mathbf{v}_1 \ldots \mathbf{v}_k]$$

Accordingly, introduce the matrix $\mathbf{V}_{k+1}$ which consists of all $(m-k)$ remaining columns of $\mathbf{V}$. The matrix

$$\mathbf{A}_k = \mathbf{U}_k \mathbf{\Sigma}_k \mathbf{V}_k^T \tag{7.4}$$

represents the closest rank k approximation of the original matrix $\mathbf{A}$, see, the before mentioned publications for further details. Thus, instead of solving the original ill-conditioned problem $\mathbf{Af} = \mathbf{h}$ we solve $\mathbf{A}_k \mathbf{f}_k = \mathbf{h}$ by computing

$$\mathbf{f}_k = \mathbf{V}_k \mathbf{\Sigma}_k^{-1} \mathbf{U}_k^T \mathbf{h} \tag{7.5}$$

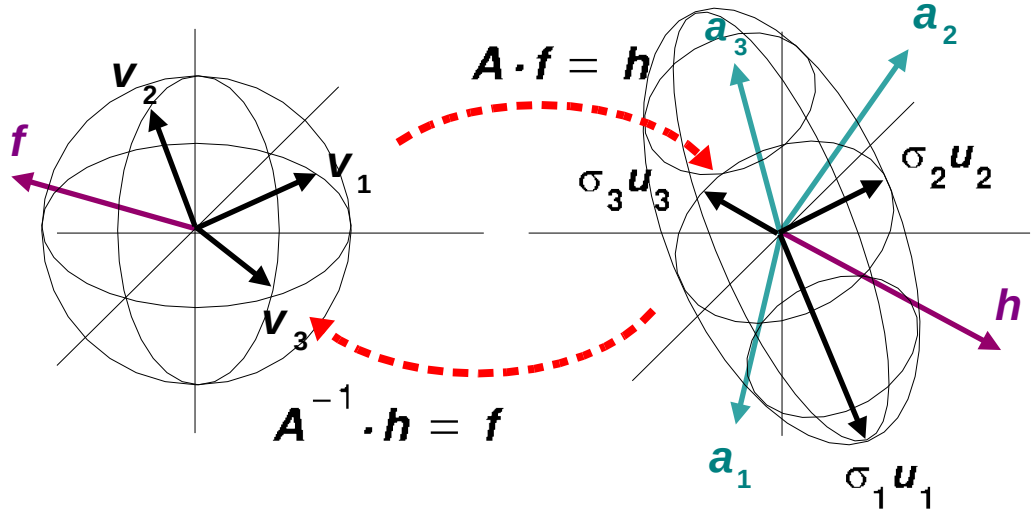


Figure 7.2: Geometric interpretation of Singular Value Decomposition $\mathbf{AV} = \mathbf{U\Sigma}$

for a selected truncation parameter k. Note that $\mathbf{f}_k$ has no component in the null-space of $\mathbf{A}$; the generalized solution space (for cases $k < \min(m, n)$) is obtained by adding a translation factor from $\mathbf{A}$s null-space

$$\mathbf{f} = \mathbf{f}_k + \mathbf{f}_n = \mathbf{f}_k + \mathbf{V}_{k+1}\mathbf{f}_{k+1} \tag{7.6}$$

where $\mathbf{f}_{k+1}$ are the $(m - k)$ variables which are not determined by given constraints. Setting $\mathbf{f}_{k+1} = \mathbf{V}_{k+1}^T\mathbf{f}_0$ (where $\mathbf{f}_0$ denotes a previously known solution of the system (in our application the control points of the previously known surface) and inserting $\mathbf{f}_{k+1}$ into above equation one obtains a solution which:

1. exactly solves $\mathbf{A}_k\mathbf{f}_k = \mathbf{U}_k^T\mathbf{h}$
2. minimizes the residual $\rho_k = \|\mathbf{A}_k\mathbf{f} - \mathbf{h}\|$ and
3. minimizes the normed difference $\eta_k = \|\mathbf{f} - \mathbf{f}_0\|$

7.3.2 The rank revealing effect of SVD

The rank revealing effect is the most valuable property of the SVD. We utilize the knowledge that for any number $k < \min(m, n)$, $k > 0$ among all size $m \times n$ matrices with rank k the matrix $\mathbf{A}_k$ is the closest possible approximation of the original matrix $\mathbf{A}$. The SVD has a geometric interpretation; This is visualized for $m, n = 3$ in figure 7.2. Again we have a linear transformation

$\mathbf{Af} = \mathbf{h}$, $\mathbf{A} \in R^{m \times n}$, $m = n = 3$. The matrix $\mathbf{A}$ transforms a vector $\mathbf{f}$ (left part of the figure) into a vector $\mathbf{h}$ (right part of the figure). The SVD delivers orthonormal bases for R^n and R^m such that the unit (hyper)-sphere in R^n spanned by vectors $\mathbf{v}_1$ to $\mathbf{v}_n$ is transformed by matrix $\mathbf{A}$ to an (hyper)-ellipsoid in R^m with principal axes determined by vectors $\sigma_i \mathbf{u}_i$ which is expressed by SVD $\mathbf{AV} = \mathbf{\Sigma U}$. The geometric interpretation of SVD helps to understand the effect of zero singular values σ_i for $i < \min(m, n)$: if, for example, σ_3 were zero the ellipsoid in R^3 degenerates to an ellipse with principal axes $\sigma_1 \mathbf{u}_1$ and $\sigma_2 \mathbf{u}_2$. Due to the relationship among the columns of $\mathbf{A}$ and the SVD, we may conclude that the matrix $\mathbf{A}$ is degenerate: $\mathbf{A}$ transforms one of the vectors $\mathbf{v}_i$ to a zero vector. In other words, its columns only span a basis for R^2, or equivalently, one of $\mathbf{A}$s columns is linearly dependent from the others.

We conclude by noting that truncating the SVD can be interpreted in three different ways:

1. *Geometric*: we ignore the "collapsed" dimensions of the ellipsoid and consider only the (orthogonal) projection of $\mathbf{h}$ onto $\mathbf{U}_k \in R^k \subset R^m$.

2. *Algorithmic*: we shift a corresponding vector $\mathbf{v}_i$ from $\mathbf{V}_k$ to $\mathbf{V}_{k+1}$ (from the first term in Eq. 7.6 to the second term)

3. *Algebraic*: by setting a singular value equal zero we enlarge the dimension of the null-space of $\mathbf{A}$ by one and collapse the range-space of $\mathbf{A}$ of the same amount.

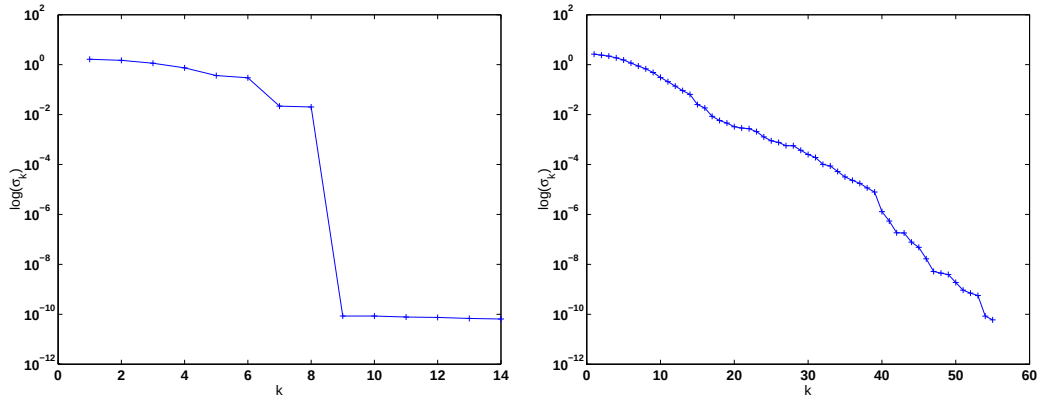
7.4 The L-curve method

The central topic of this section is the algorithm to obtain the optimal truncation parameter given the Singular value decomposition of a linear systems of equation. We will refer to the example from section 7.2, fig. 7.1.

7.4.1 The singular values plot

The benefits of SVD become obvious when comparing a singular problem with well-defined rank and an ill-posed problem. Figures 7.3(a)-(b) show the so-called singular value plots of two curve-surface incidence constraints:

(a) the pre-image of the 3D curve is a part of iso-parametric line for $u = const.$ and $v \in \langle 0.25, 0.75 \rangle$ treated as a general curve (i.e. equations were set up by the composition method)



(a) (b)

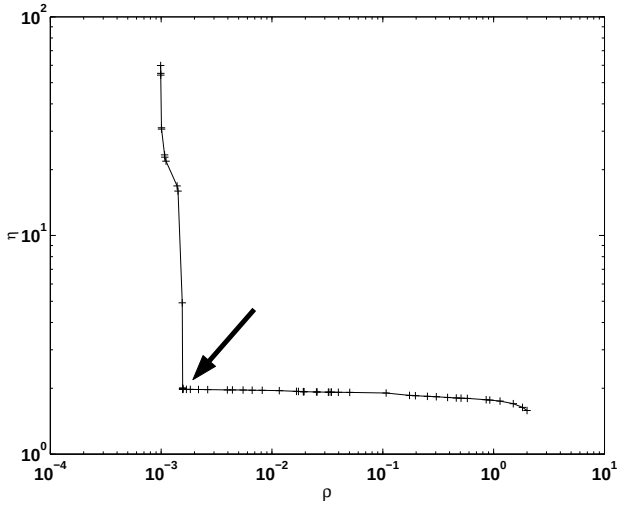
Figure 7.3: The σ-plot of a singular curve constraint with well-determined rank (a), and an ill-posed curve constraint (b). The σ-plots show the distinct jump in the former case but gradually decreasing singular values for the latter case.

(b) the constraint from example in figure 7.1, section7.2 (i.e. the pre-image curve is a general cubic curve)

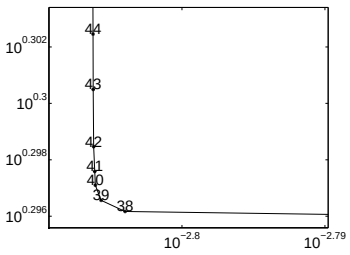
We draw the dependency of the logarithm of singular value σ_k (vertical axis) on the truncation parameter k (horizontal axis). In both cases the surface is bi-cubic with 8×10 control points. For the iso-parametric curve the problem is under-determined but "exactly" singular: the jump between σ_8 and σ_9 has the magnitude of several exponents, therefore we can safely obtain a solution by means of eq. 7.6 by setting the truncation parameter $k = 8$. Of course, for this particular case, it is not necessary to employ the composition method. The equations can be set up directly obtaining a full-rank matrix of size 8×80 which confirms the result from the SVD. On the other hand, the singular values plot of the other constraint shown in fig. 7.3(b) decays gradually to zero without a significant jump in magnitude at a specific σ_k. The problem is ill-posed according to definition in 7.2: there is no obvious choice for the rank k.

7.4.2 Determining the optimal truncation parameter

Determining truncation parameter k is the central difficulty when a linear problem is ill-posed. The choice of the truncation parameter k is apparently not arbitrary: Choosing too small k will cause large error in ρ_k and too large k will increase η_k, since computing $\mathbf{\Sigma}^{-1}$ in eq. 7.5 involves divisions by small numbers. Therefore,



(a)



(b)

Figure 7.4: The L-curve (a) with the magnified "corner" region (b) for the smoothly varying curve constraint shown in figure 7.1.

one seeks a solution which satisfies the criteria defined in §7.3.2: it must minimize the residual η_k and it must possess the minimal norm ρ_k.

It was observed by several researchers that if the residual of the truncated problem ρ_k and the norm of the so obtained solution η_k are plotted versus each other in logarithmic scale for several ks a curve with characteristic "L-shape" almost always results, see [37] for historical overview and further background of L-curves.

For the ill-posed constraint – corresponding to the example in fig. 7.1 and to singular value plot in fig. 7.3(b) – we obtain the L-curve shown in figure 7.4(a): The optimal truncation parameter corresponds to the sharp concave corner em-

phasized by an arrow: at this point, an attempt to determine another unknown variable (i.e. increasing the truncation parameter k) only increases the norm of the solution without a significant improvement of the residual. Thus it is reasonable to conclude that given a linear system of equations $\mathbf{Af} = \mathbf{h}$, the matrix $\mathbf{A}_k$ (equation 7.4) is the best possible approximation of the matrix $\mathbf{A}$ using the criteria from §7.3.1. It follows that among all surfaces the surface with control points $\mathbf{f}_k$ (obtained by means of eq. 7.5 applied to each spacial coordinate) is optimal in the sense of criteria stated on ρ and η.

We detect the corner of the L-curve as follows: A magnification of the corner region shown in figure 7.4(b) reveals that the "corner" consists of a cluster of rapidly varying values (ρ_k, η_k). This leads to an idea to approximate the discrete points of the L-curve by a sufficiently smooth B-spline curve $L(t)$ and to locate a parameter value t_k where the curve exhibits largest negative curvature. Then the point associated with the closest smaller k is chosen. Thus, theoretically, the optimal truncation parameter k is found by solving $\max_t(\kappa)$ and $\min_k \|L(t_k) - (\rho_k, \eta_k)\|$. In order to detect such points, we start with a piecewise linear approximation of the discrete point set $(\rho_k, \eta_k) : k_{\min} \leq i \leq k_{\max}$. We increase the polynomial degree of the piecewise linear curve to cubic and apply a slightly modified version of Lyche-Mørken algorithm [52] to remove as many knots as possible. We build a statistics on errors ϵ_r caused by removal of a r-th knot and increase the threshold which regulates the Lyche-Mørken recursion until the statistics contains only errors such that

$$\|\max(\log \epsilon) - \min(\log \epsilon)\| \leq 1$$

This heuristics relies on the observation that the L-curve has one (in the optimal case), or several, distinct regions of large curvature and is relatively smooth elsewhere. At the output, we obtain the B-spline representation of the L-curve which has one (in the optimal case) or several C^0 discontinuities in the knot vector. The curve points corresponding to these discontinuities are (or are close to) the corner of the L-curve.

The range for tested truncation parameters $(k_{\min}, k_{\max})$ is determined based on the following heuristics: Consider, for example, the L-curve shown in figure 7.4 on the left. Figure 7.3(b) shows the singular value plot of the associated linear problem: there is a jump near σ_{38} but there are also several jumps of approximately the same (or even larger) magnitude between σ_{40} and σ_{50} and between σ_{50} and σ_{60}. Thus, one should select such range for $k_{\min}$ and $k_{\max}$ that no large jumps occur outside of $\sigma_{\min}$ to $\sigma_{\max}$ with a sufficiently large buffer zones at both ends. The comparison of the two figures 7.3(b) and 7.4 also demonstrates how dangerous it is to rely exclusively on the σ-plot: the jump at σ_{53} is larger than the jump at σ_{38} but the solution becomes unstable already at $k > 40$, see also the magnified L-curve corner in fig. 7.4(b).

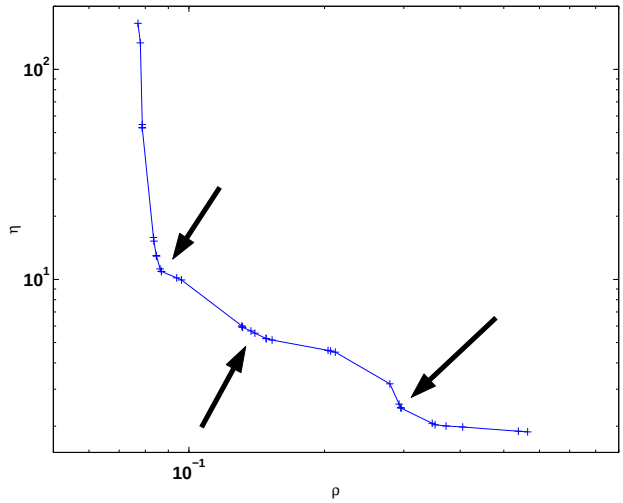


Figure 7.5: The L-curve for the rapidly varying curve from figure 7.1: the "cascade" shape of the L-curve is characteristic for data which do not satisfy the discrete Picard condition.

7.4.3 Demonstrating the Picard condition

It is interesting to compare the L-curves which result when shape of the constrained curve is modified. Again, consider the curve-surface constraint from fig. 7.1: The figure 7.4(a) shows the L-curve for the smoothly changing curve from fig. 7.1 while in figure 7.5 the L-curve is shown which results from the rapidly oscillating curve.

If the constrained curve varies too rapidly the L-curve possesses several distinct corners as shown in figure 7.5: it moves from one to the next corner in a "cascade"-like manner. In such cases it is reasonable to select the corner with smallest η even if the residual does not reach the required threshold. However, such cascades in the L-curve generally indicate that the data to be interpolated does not satisfy the discrete Picard condition, see §7.2.2. Such input data (i.e. such shapes of the constrained curve) should be avoided right from the start. Usually, none of the corners (truncation parameters) will deliver a meaningful solution. Rather, the presence of several sharp corners signalizes that there is no unique choice for a closest lower rank approximation of the problem. This is the limitation of all SVD based regularization methods – the parameters of the ill-posed linear problem must satisfy the Picard condition.

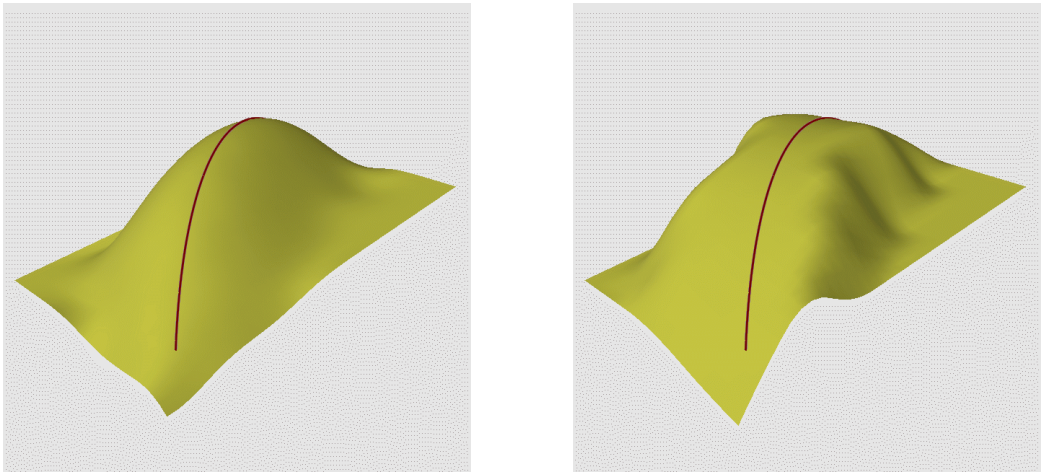


Figure 7.6: The TSVD solution corresponding to the corner of the L-curve from Fig. 7.4. In order to emphasize the "aliasing" effect the right figure shows a bi-quadratic surface with the same curve constraint and the same number of DOFs.

7.4.4 The "aliasing effect" of the truncated SVD solution

The procedure described above is known as "Truncated SVD regularization" (TSVD), see [35]. The solution surface corresponding to the L-curve corner at $k = 38$ is shown in figure 7.6: observe, that the surface deforms along the curve in a "staircase" like manner. It is less apparent in the bi-cubic surface shown on the left and stronger in bi-quadratic surface on the right. We call this the *surface aliasing effect*, cf. [55], because of the similarity with aliasing as known in computer graphics. The resulting surfaces although in algebraic sense correct (the example in fig. 7.6 on the left renders the residual of $\rho_{38} = 10^{-2.95}$, cf. to figure 7.4, the more "aliased" bi-quadratic surface on the right has $\rho_{49} = 10^{-6}$) will surely not be accepted by the designers. Before we approach suitable "anti-aliasing" methods, in the next paragraph we analyze the reasons for such unwanted deformations of the surface in more detail.

7.5 The surface aliasing effect

Figure 7.7 shows the result of constraining the incidence of a diagonal, vertical and horizontal line (in the domain of the surface) on a 20×20 bi-quadratic B-spline surface. The distribution of the dependent control points of the surface is shown in the bottom part of the figure: each bright square represents a control point safely determined by the L-curve method. The aliasing effect in example on the left is strong – the surface exhibits undesired bumps or wiggles – whereas in

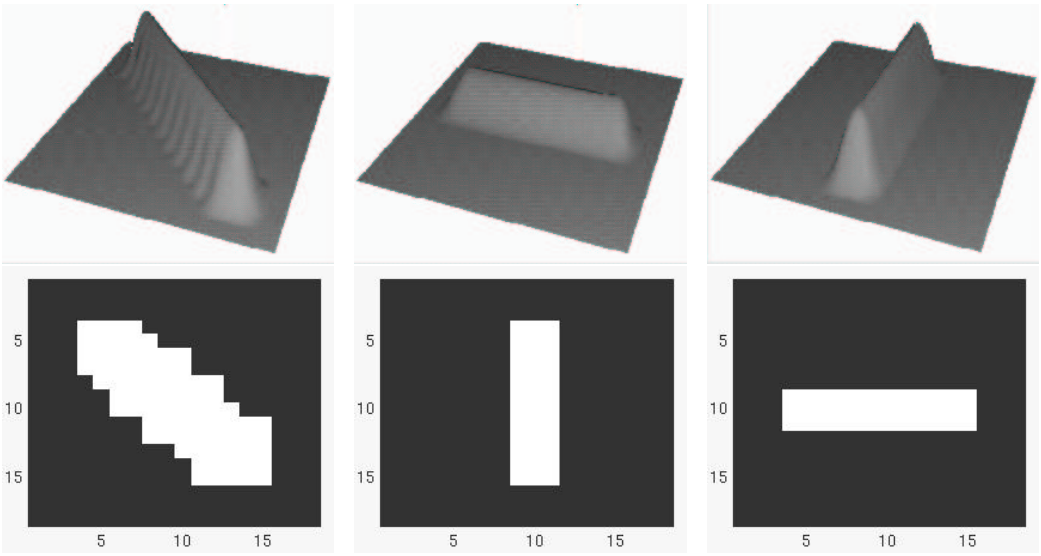


Figure 7.7: An attempt to constrain a diagonal, vertical and horizontal line on a 20×20 bi-quadratic B-spline surface. The lower part of the figure shows the distribution of control points determined by the L-curve method

the middle and right examples, no bumps can be observed. A comparison with the staircase effect when drawing a line on a screen by assigning color values to a discrete grid of pixels comes to mind immediately. What is the reason for this "aliasing" and is there a way to remedy it?

The first question can be easily answered: the control points of a tensor product surface are aligned on a rectangular grid, the size and density of which depend on the parametrization of the surface (compare with pixel-grid of a monitor screen). The TSVD method as described in previous sections defines an algebraically stable solution by minimizing the norm $\|\mathbf{f} - \mathbf{f}_0\|$. It discards certain variables for the prize of this stability. However, this does not mean that the discarded variables are really independent – we just cannot reliably determine the "amount" of the dependency. Apparently, the transition among dependent and independent control points is too abrupt which causes undesirable bumps in the shape of the surface.

The frequency of the bumps depends on the order of polynomial continuity among the surface patches. The higher order of polynomial continuity, the lower frequency of the bumps is observed. With growing polynomial degree (and order of polynomial continuity) the change of a constrained curve propagates throughout a larger region of the surface and is distributed across more surface patches. This gives the surface an opportunity to deform more slowly, and thus, more equally.

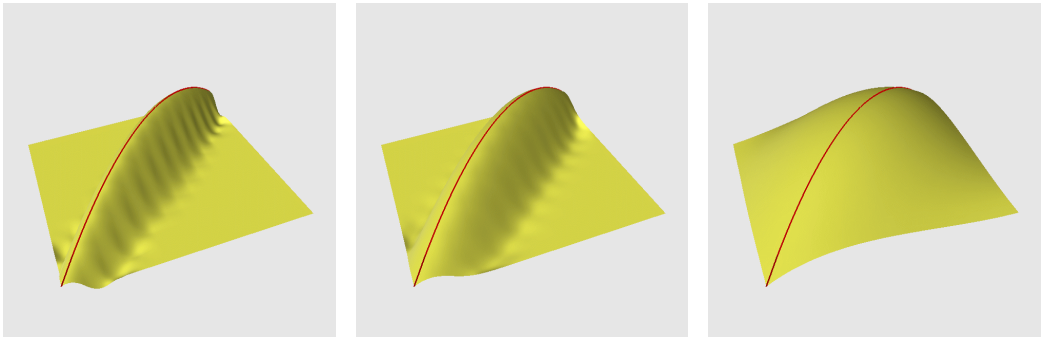


Figure 7.8: Influence of polynomial degree and order of continuity to the aliasing effect: left figure shows a bi-quadratic C^1-continuous B-splines surface, the surface in the middle figure is bi-cubic and C^2-continuous and the right-most surface is bi-quintic C^4-continuous. Each surface has 15×15 control points. In all cases the same curve constraint and the same deformation was applied.

The effect is illustrated in figure 7.8: a change is applied to the same curve on bi-quadratic and bi-cubic and bi-quintic B-spline surfaces with 15×15 control points. The aliasing becomes less distinct with growing order of polynomial continuity of the surfaces. However, increasing the degree of used surfaces is not the optimal solution since the high order of polynomial continuity restricts the allowed changes of the surface. In other words, the surface becomes more sensitive to changes of constrained curves. For example, the L-curve for the bi-quadratic surface in fig. 7.8 on the left has only one distinct corner and satisfies the constraint with residual error $\rho_k = 10^{-6}$ for $k = 80$. I.e. from the numerical point of view the solution is perfect. The L-curve for bi-cubic surface is already more scattered and we have to truncate the SVD at $\rho_{45} = 10^{-4.7}$. The L-curve for the bi-quintic surface exhibits the first sharp corner already at $\rho_6 = 10^{-1.3}$. Thus by increasing the polynomial order of the surface we damp the aliasing effect for the prize of increasing error in the residual. The flexibility of the surface can be increased by generating new degrees of freedom (i.e., by refinement of the control mesh of the surface) at positions where the error between the required curve and the surface exceeds a prescribed limit. However, then the composition matrix has to be recomputed from scratch which becomes more expensive with growing polynomial degrees of the surface, see also section 4.4.

Chapter 8

Linear constraint solving II

In this chapter two methods are proposed to improve the shape of the constrained surfaces. We describe two *anti-aliasing* methods: a geometric method using local reparametrization the of the surface (section 8.2) and an algebraic method which applies regularization with a side-constraint in general form, section 8.3. The section 8.3.3 contains a brief overview about alternatives to SVD and points out directions for future improvements of the algebraic solver.

8.1 Introduction: Anti-aliasing

The aliasing effect always occurs when using piecewise bi-polynomial tensor-product surfaces, whenever the constrained curve does not match the rectangular arrangement of the control points. The problem seems to be known in the field of data interpolation (cf. [14]). In [39], Hayes introduced piecewise composite surfaces with *curved knot lines* which cope better with an arbitrary curve. The domain of the surface is defined as a curvilinear mesh of knot lines. The parametrization of the surface can then be better adjusted to match a given curve. Although very powerful and conceptually simple, in practice, elementary algorithms for traditional B-splines (for example knot insertion and removal, degree raising and lowering) become very complicated with Hayes splines, which may be the reason for low acceptance of this type of surfaces. For example, it is not clear if the polynomial composition can be formulated in similar manner as for "traditional" B-splines, or even, if the blossoming principle applies at all. Though this could be an interesting topic to investigate, the practical usability of Hayes-splines in CAD is questionable – we have not encountered a modeling system which uses this kind of surfaces, or a scientific publication which further investigates the usage of Hayes-splines in geometric and solid modeling.

In our effort to remove the "aliasing" effect" we have successfully applied

two approaches: in the first approach we reparametrize the surface such that the pre-image of an arbitrary curve becomes an iso-parametric line. We convert an arbitrary shaped ill-conditioned curve constraint to a well-conditioned iso-curve constraint which can be safely solved with no aliasing. The second method is comparable to traditional "anti-aliasing" as applied in computer graphics: we find such values for the remaining degrees of freedom of the surface (i.e. those which could not be determined by TSVD) that the aliasing effect becomes less apparent or completely disappears. We define new constraints, working against the aliasing, in connection with the primary incidence constraint. The following paragraphs describe these two methods in more detail.

8.2 Surface reparametrization

It follows from the discussion in section 7.5 that the only case of curve constraints a tensor-product surface can handle without aliasing are iso-parametric lines. In this case, the influenced control points of the surface lie on (or inside) an axis-aligned rectangle, cf. figure 7.7. The question is now, given a surface with one or several arbitrarily positioned curve constraints, is a conversion to this case possible, without destroying the appearance of the input data?

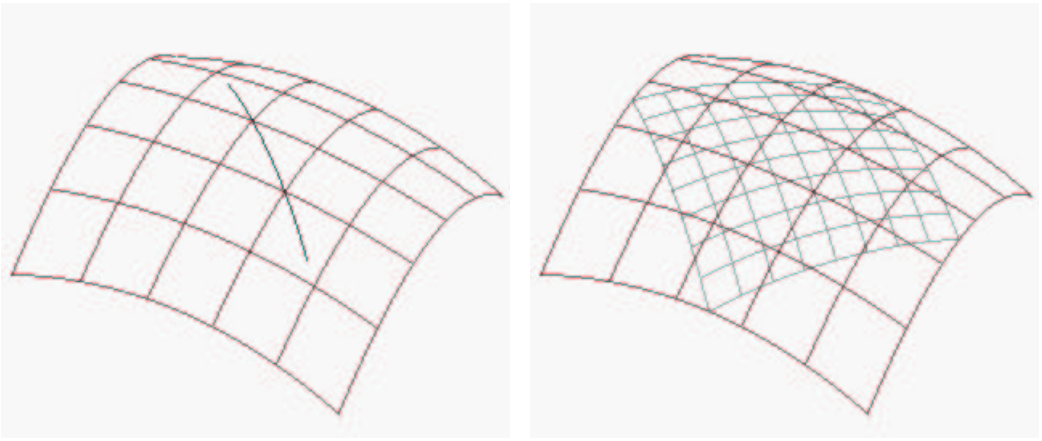
8.2.1 Reparametrization of tensor-product B-splines

Suppose the designer wishes to deform the surface $F(u, v)$ aligned along the curve shown in figure 8.1(a). We are looking for a surface in the domain of which this curve can be represented as an iso-parametric line and which is locally identical to the original surface in the 3D space, see figure 8.1(b). Obviously, this can only be done by some kind of re-parameterization of the original surface, as shown in the figure 8.1(c): The lower left figure shows the curve projected into the domain of the original surface $F(u, v)$. We have to find a surface $G(s, t)$ in the domain of F, such that the given curve is a line in the domain of G, as shown in figure 8.1(c) on the right. If the surface $G(s, t) = (u(s, t), v(s, t))$ is known, we can locally replace the surface F by a new one

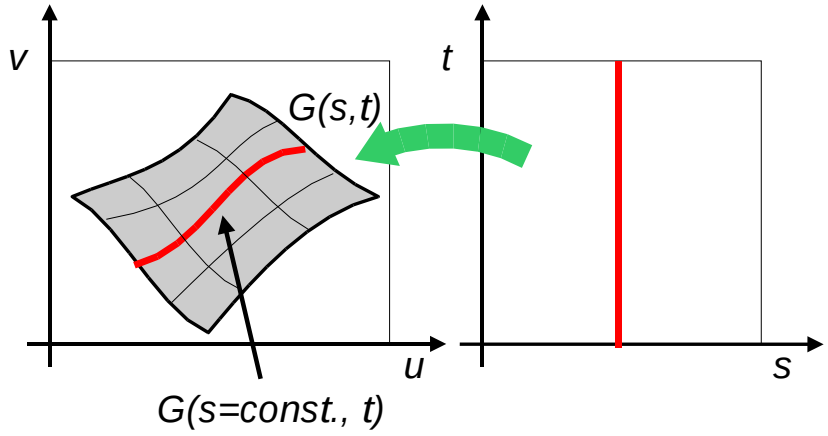
$$H(s, t) = F(\underbrace{u(s, t), v(s, t)}_{G(s,t)}) \tag{8.1}$$

yielding the reparametrized surface H with required properties. There are two problems to be solved:

1. obtaining the surface G



(a) (b)



(c)

Figure 8.1: The surface is reparametrized such that a domain curve (shown as a thick line in the left lower figure) corresponding to the 3D curve shown in figure (a) becomes an iso-parametric line (right lower figure). For that a new surface $G(s,t)$ satisfying this property is computed. The (approximate) composition of the two surfaces yields a surface $H(s,t) = F(G(s,t))$ locally identical with the original surface which is shown in figure (b).

2. computing the reparametrization surface H from G and F

The surface G can be obtained by letting the designer sketch the four boundary curves of the new feature (Fig. 8.1, left), project them into the domain of

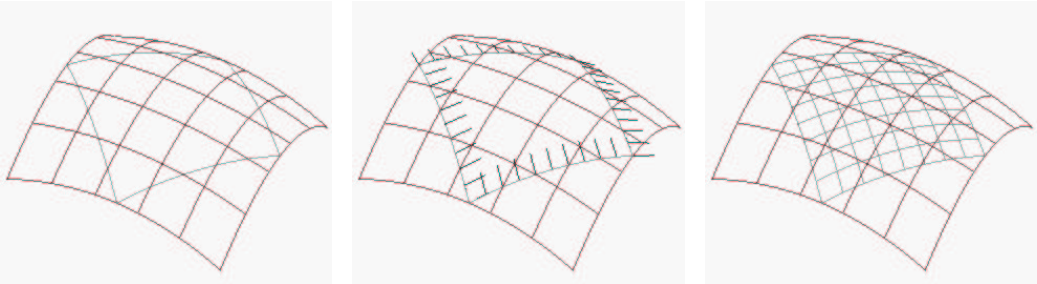


Figure 8.2: Left figure: boundary curves of the new reparametrized surface, middle: derivatives along the boundary curves assuring C^1 continuity to the original surface. Right figure shows the resulting surface obtained from interpolating the four boundary curves and derivatives along them.

the surface F and compute a 2D boolean-sum surface. Another possibility is a heuristics utilizing the sketched curve: the curve is projected into the domain of F, where two offset curves at user-defined distances are computed, which serve as the boundary curves in one parametric direction. The boundaries in the other direction are chosen to be linear. The expression 8.1 is a polynomial surface-surface composition. Unfortunately, if the outer operand of the composition, the surface F, is a tensor-product B-spline the result of the composition does not have a tensor-product representation anymore, as it was already noted by DeRose et al. in [18]. Consequently, there is no algebraic relationship between the control points of both surfaces in the style of one-dimensional composition constraints. To represent H would require a surface definition on a non-rectangular domain. We have not further investigated this alternative; for example, one could try to formulate H in terms of Seidel's B-Patches, see e.g. [77] or Loop's generalization of B-splines to arbitrary domains [51]. Instead, we propose an efficient method which computes an approximation of H in tensor-product B-spline format up to arbitrary user defined tolerance. The algorithm is described in the next paragraph.

8.2.2 Solving a constrained curve network interpolation problem

Consider figures 8.1 and 8.2: We can compute arbitrarily many curves representing G such that s or $t = const.$ and compute their exact representation on the surface F obtaining a network of 3D curves. In the second step we apply Gordon's method to interpolate a tensor product surface through a topologically orthogonal network of 3D curves, see [30]: A set of parallel curves from surface $G(s,t)$ are scanned at suitable values s_i and t_j. In the following we denote them by $G(s_i,t) = G_i(t)$ and $G(s,t_j) = G_j(s)$. Such curves intersect at points

$G(s_i, t_j) = G_{i,j}$. In addition, the vector field curves $\partial G_i(t)/\partial s = D_i(t)$ and $\partial G_j(s)/\partial t = D_j(s)$ are computed. Given this data the algorithm to interpolate the surface $H(s,t)$ looks as follows:

1. Estimate the numbers p, q which determine the number of curves $G_i(t)$ and $G_j(s)$ such that the shape of the surface $G(s,t)$ is fully determined by that curve network. After computing $F(G_i(t)),\ 0 \leq i < p$ and $F(G_j(t)),\ 0 \leq j < q$ we obtain a network of 3D curves incident on F and meeting at points $F(G_{ij})$. Figure 8.2(a) shows the curve network for the case $0 \leq i, j \leq 1$. The derivative curves D_i and D_j transformed to 3D are vector field curves representing directional derivatives of F with respect to s and t, $\partial F(D_i(t)/\partial s)$ and $\partial F(D_j(s)/\partial t)$, see fig. 8.2(b).

2. We now have enough information to carry out a cubic interpolation among the curves $F(G_i(s)),\ 0 \leq i < p$ and $F(G_j(t)),\ 0 \leq j < q$, using the respective derivative conditions $D_i(t)$ and $D_j(s)$. Using surface skinning (c.f. paragraphs 2.2.2 and 2.2.2.1), the surfaces H_1 and H_2 are computed from this data. The surface H_3 is obtained as a result of tensor product interpolation of the values $F(G_{ij}),\ 0 \leq i < p,\ 0 \leq j < q$ and the derivatives at corner vertices and at the intersection points of the scanned curves.

3. According to [30], the surface $H(s,t) = H_1 + H_2 - H_3$ interpolates the given network of curves and points at which they intersect, fig. 8.2(c).

The surface $H(s,t)$ is exactly identical to the original surface $F(u,v)$ along the scanned curves and points and it approximates the original surface in between. Moreover, due to the derivative information inherited from F, there is at least a G^1 continuous connection of H to F at prescribed curves. The quality of the approximation is determined by measuring the maximum of $\epsilon(s,t) = |F(G(s,t)) - H(s,t)|$. Whenever ϵ is larger than a prescribed value, the curve network is refined and the whole process is repeated. The refinement is done by recursively inserting a new curve in the middle of each interval of the surface G in each parametric direction. The curve is then composed with the original surface F and added to the interpolation equations for H. We recommend to start with four boundary curves and corner points and refine the curve network iteratively. Proceeding this way, the example from figure 8.2 succeeds after 1 step with $\max(\epsilon(s,t)) < 10^{-6}$. A sculpted example which succeeds after 3 such refinement steps is shown in figure 8.3.

The degree and knot density of the resulting surface depend on:

- the degree and parameterization chosen for the initial surface G and
- the degree and knot density of the original surface.

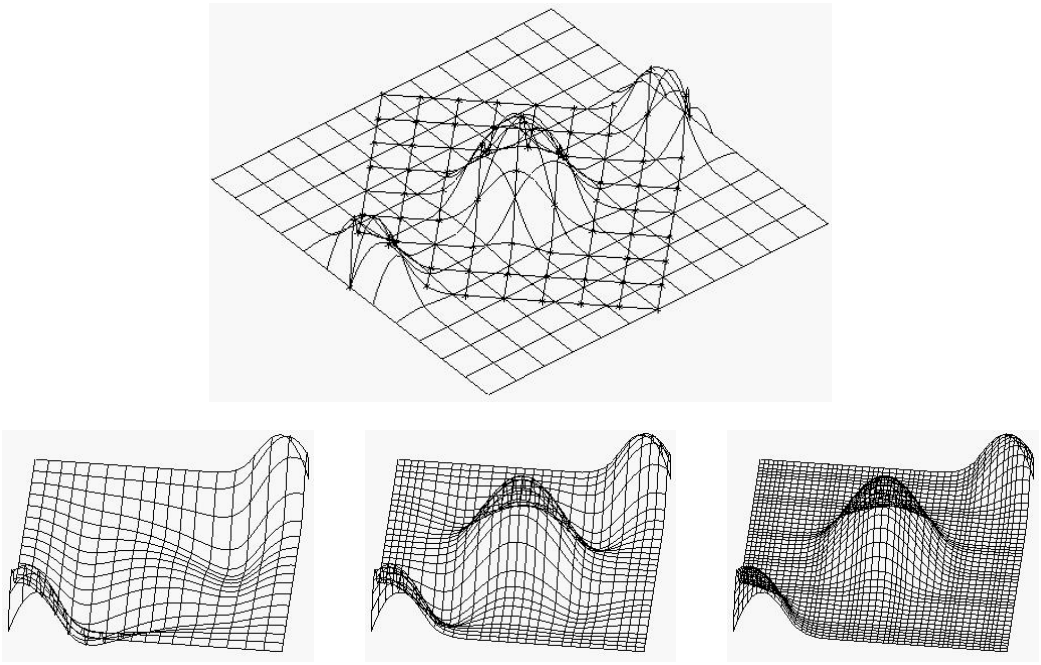


Figure 8.3: The left-most figure shows the surface H after the first interpolation step. The approximation error falls below 10^{-8} after twice inserting a curve and derivatives in the middle of each interval. Interpolation of the curve network in upper figure leads to the result shown in right-most figure.

The degree of a curve resulting from curve-surface composition is given by $d.\,(k+l)$ where d is the degree of the domain curve and k and l are the degrees in both parametric directions of F. The curves are evaluated using the methods from chapter 4 and the skinning problems are solved efficiently with the aid of algorithms for solving banded linear systems. For further details, examples and a run time analysis see [55].

8.2.3 A design example

After the reparametrization, the constrained curve is exactly incident on the reparametrized surface and its pre-image in the domain of the new surface is an iso-parametric line. Incidence and tangency constraints are stated on the boundaries of the new surface and their current values are fixed. Any change of the curve is transfered to the new surface without an interference with the "underlying" original surface.

Figure 8.4 demonstrates a design application of the presented method. Here, the designer wants to add a "crater" shaped feature to the surface shown in figure 8.4 on the left: Two closed curves are sketched on the surface. The system

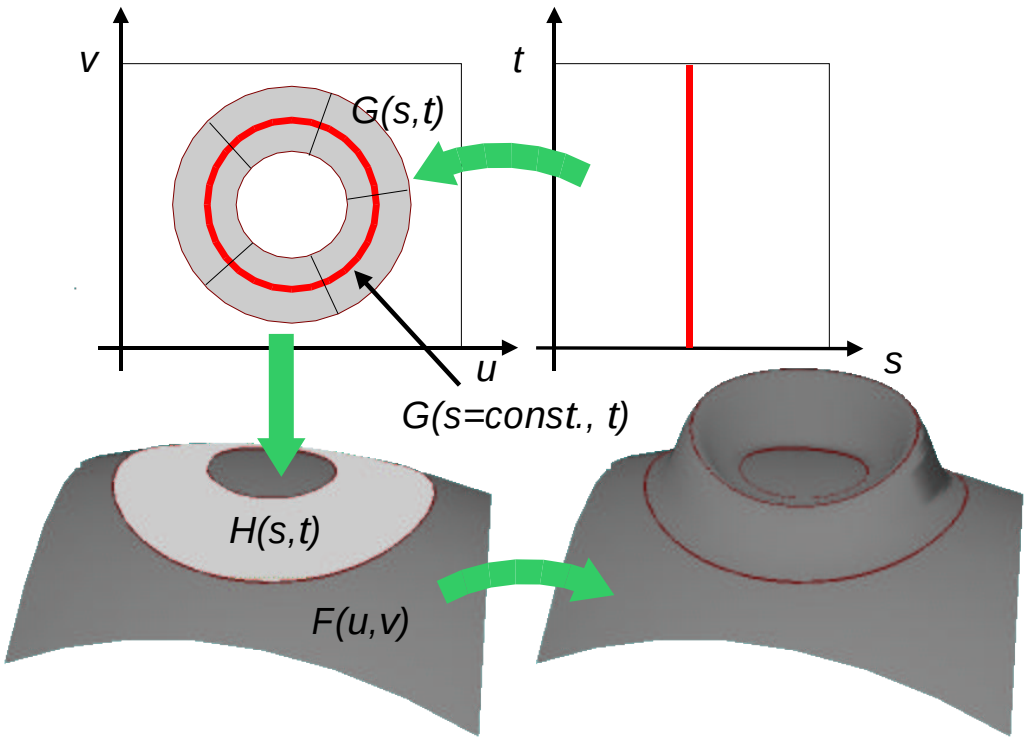


Figure 8.4: The reparametrization method completely circumvents ill-conditioning and aliasing: the surface between the two curve shown in the left part of the figure is replaced by a new surface which can be safely determined from the fixed boundary curves, derivatives along boundary curves and a user specified curve "handle"

projects the curves into the domain of the surface and computes their exact representation on the surface. They represent the boundaries of the new feature. The designer can choose a continuity of the crater feature along the boundary curves. Here incidence and G^1 continuity along both boundaries are specified. The system computes a reparametrization surface from surface curves as described in previous paragraph. Two tangency and two incidence constraints along the boundary curves are generated between the new and the original surface. The area covered by the new surface is trimmed away from the original surface, see fig. 8.4. The manipulation tool of the designer will be any iso-parametric curve in either direction on the crater surface, which can now be selected by choosing a direction and picking a point anywhere on the surface, as demonstrated in figure 8.4 on the right: the surface reacts to changes of any iso-parametric curve as expected: the incidence and tangency constraints along the boundary and feature curves assure the proper connection of the new feature to the original surface. No aliasing ef-

fects occur, since all constrained curves are iso-parametric lines in the domain of the new surface.

8.2.4 Summary of the method

The disadvantage of this method is that one can adjust the surface only for one selected curve constraint. All other constraints need to be recomputed to match the new topology of the model which considerably complicates the data-management. Furthermore, this method guarantees only aliasing-free editing of the reparametrized surface: if we haven't fixed the boundaries of new surface feature, then the underlying original surface would have to follow the changes of new reparametrized surface. However, then the new shape of the original surface needed to be computed from general curve constraints which are not aliasing-free.

8.3 Constrained least squares

The idea behind the second approach is to better utilize the degrees of freedom of the surface which were discarded by the L-curve method. We refer to the notation used in sections 7.3.1-7.4: we solve a general linear system of equations $\mathbf{Af} = \mathbf{h}$ with knowns and unknowns set as explained in §7.3.1.

The L-curve algorithm separates the degrees of freedom into two sets: the dependent ones and the discarded or free ones. Referring to eq. 7.6, we have $(m-k)$ free parameters and the solution space obtained from the TSVD at our disposal. Thus, we may use these free parameters for stating additional constraints which help against the "aliasing" effect and improve the shape of the surface without destroying the already minimal residual of the TSVD solution. Well-suited for this purpose are the quadratic surface functionals frequently used in variational surface design, see [32], for example. We have discussed efficient methods to obtain the normal equations for all frequently used functionals in chapter 6. In the following we will consider a linear combination of matrices

$$\mathbf{L} = \sum_{i=1}^{3} \alpha_i \mathbf{L}_i \quad \text{for} \quad \sum_{i=1}^{3} \alpha_i = 1 \tag{8.2}$$

where $\mathbf{L}_i$ denote the normal equations of area, thin-plate energy and variation of curvature minimizing functionals for $i = 1, 2, 3$, please refer to table 6.1.

8.3.1 The constrained regularization

Formally the so-called regularization methods with general side constraint and constrained variational problems are equivalent. In both cases a constrained least

square problem is solved [29, Section 12.1]: an objective function $f = \mathbf{f}^T\mathbf{L}\mathbf{f}$ is minimized subject to a set of linear constraints $\mathbf{A}\mathbf{f} = \mathbf{h}$ which is usually denoted by

$$\min_{\mathbf{f}} f \quad sub \quad \min_{\mathbf{f}} \|\mathbf{A}\mathbf{f} - \mathbf{h}\| \tag{8.3}$$

In this context $\min_{\mathbf{f}} f = \mathbf{L}\mathbf{f}$. There are two possible ways to solve problems of the type 8.3:

1. Extract a solution space which satisfies the constraints $\mathbf{A}\mathbf{f} = \mathbf{h}$ obtaining a generalized solution in terms of hyperplane equation $\mathbf{f} = \mathbf{f}_k + \mathbf{f}_n$, see also §7.3.1. Substituting the generalized solution for $\mathbf{f}$ in the side constraint yields an unconstrained minimization problem

   $$\mathbf{L}\left(\mathbf{f}_k + \mathbf{f}_n\right) = \mathbf{0}$$

   which is solved for $\mathbf{f}_n$. The solution of the original constrained optimization is obtained by inserting $\mathbf{f}_n$ back into $\mathbf{f} = \mathbf{f}_k + \mathbf{f}_n$.

2. Use Lagrange method (see e.g. [69]): introduce auxiliary variables, the so-called, Lagrange multipliers $\mathbf{\Lambda}$ and solve the system:

   $$\begin{bmatrix} \mathbf{H} & \mathbf{A}^T \\ \mathbf{A} & \mathbf{0} \end{bmatrix} \cdot \begin{bmatrix} \mathbf{f} \\ \mathbf{\Lambda} \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ \mathbf{h} \end{bmatrix}$$

This is easy, if the system matrix $\mathbf{A}$ has full rank: such variational equations are commonly solved in the context of multi-patch surface interpolation or scattered data interpolation, see e.g. [9, 5, 31, 48, 32, 19, 40, 20]. There are efficient numerical methods to solve (1) and (2) if the matrix $\mathbf{A}$ has full rank, e.g. the Generalized QR-decomposition [29]. A practitioner obtains the solution by calling LAPACK [2, 4] functions DGGLSE or DGGGLM. If, however, the matrix $\mathbf{A}$ is rank deficient or even the problem is ill-posed – as it is the case with our generalized curve-surface constraints – one has to turn to more sophisticated methods.

8.3.2 Modified truncated SVD

If the SVD of $\mathbf{A}$ is available, an efficient numerical method to solve such constrained approximation problem is the so-called, Modified TSVD (MTSVD) [38]. It allows to define more general side constraints. Instead of $\|\mathbf{f} - \mathbf{f}_0\|$ one minimizes $\|\mathbf{L}\left(\mathbf{f} - \mathbf{f}_0\right)\|$ for some regularization matrix $\mathbf{L}$. In our case the selection of regularization matrix is obvious: $\mathbf{L}$ is set as described above. We know that the linear combination of the surface functionals is minimal at

$$\mathbf{L}\left(\mathbf{f} - \mathbf{f}_0\right) = \mathbf{0}$$

with the previously known solution $\mathbf{f}_0$ before the deformation. Substituting eq. 7.6 for $\mathbf{f}$ in above equation yields

$$\begin{aligned} \mathbf{L}\left(\mathbf{f}_k + \mathbf{V}_{k+1}\mathbf{f}_{k+1} - \mathbf{f}_0\right) &= \mathbf{0} \\ \mathbf{L}\mathbf{V}_{k+1}\mathbf{f}_{k+1} &= \mathbf{L}\left(\mathbf{f}_0 - \mathbf{f}_k\right) \end{aligned} \tag{8.4}$$

Solving eq. 8.4 for $\mathbf{f}_{k+1}$ and substituting back to eq. 7.6 we obtain a "corrected" MTSVD solution

$$\mathbf{f}_{L,k} = \mathbf{f}_k + \mathbf{V}_{k+1}\mathbf{f}_{k+1}$$

which in addition to properties 1-2 from §7.3.1 minimizes the norm $\|\mathbf{L}\left(\mathbf{f} - \mathbf{f}_0\right)\|$.

In [38], Hansen proposes to compute the values of $\rho_k = \|\mathbf{A}\mathbf{f}_{L,k} - \mathbf{h}\|$ and $\eta_k = \|\mathbf{f}_{L,k}\|$ for each k and locate the corner of the "weighted" L-curve as in the case of TSVD. Consider, that there may be several hundred up to several thousand truncation parameters, thus, we must be able to obtain the "penalty" term $\mathbf{f}_{k+1}$ fast. The linear system in Eq. 8.4 can be solved in a particularly efficient way by pre-computing the worst case QR-decomposition of $\mathbf{L}\mathbf{V}_{k_{\min}+1}$, i.e., for the smallest k ever used and update it for the each $k_{\min} < k$ accordingly, see [38] and [29, §12.6] for details. Thus obtaining the proper "correction" term $\mathbf{f}_{k+1}$ for each k is computationally cheap. Note that the pre-condition for this procedure is that $\mathbf{L}$ or at least $\mathbf{L}\mathbf{V}_{k_{\min}+1}$ must be non-singular. Otherwise, the "trick" of updating the QR-decomposition does not work and Eq. 8.4 must be solved for each k "from scratch". Fortunately, if $\mathbf{L}$ is the linear combination of matrices as described above, the matrix $\mathbf{L}\mathbf{V}_{k_{\min}+1}$ tends to be non-singular for at least one $\alpha_i > 0$, reasonable parametrization of the B-spline surface F and sufficiently large $k_{\min}$, see also [19, 40, 20].

Figure 8.5 shows three different modifications of the same surface and the same constraint as in figure 7.1 after the MTSVD correction was applied. No aliasing can be observed, the error in the residual is less than 10^{-3}. Since the system matrix in Eq. 8.4 is not particularly well conditioned (the typical condition numbers obtained in our experiments were between 10^3-10^4), the discrete L-curve points are considerably more "scattered". Generally, the truncation parameter obtained by above method is smaller than if no "smoothing" side constraint is applied. We have found that the "$\mathbf{L}$ weighted" L-curve almost always possess one more corner compared to the L-curve obtained by TSVD method. Typically, the corner occurs earlier, hence, the usage of a generalized side constraint causes larger error in the residual but delivers smoother solutions, which is in accordance with results described in [38]. Alternatively, one can compute k by the TSVD method and solve for $\mathbf{f}_{k+1}$ only once. However, in some cases the L-curve corners which occur earlier if the weighted norm $\|\mathbf{L}\mathbf{f}\|$ is considered could be missed.

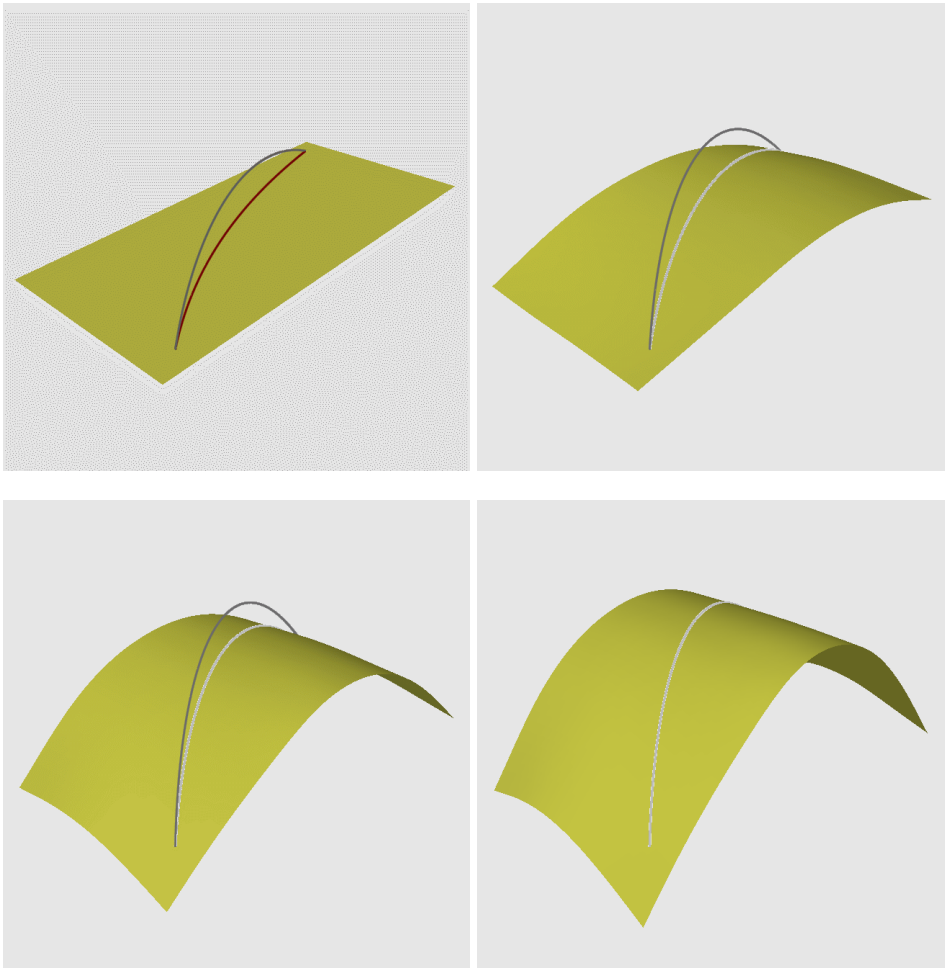


Figure 8.5: Bi-cubic surface obtained by the MTSVD method. The modifications of the surface proceed from left to right, the "next" sketched curve is shown in darker color. Adding the "smoothing term" to the TSVD solution removes the aliasing artifacts and delivers an optically and geometrically smooth surface.

8.3.3 More results and selected problems

Based on numerous experiments with the methods described in previous Paragraphs we conclude that the MSTVD method reliably removes the aliasing effects and, if SVD of the system matrix is available, efficiently locates the compromise between constraints satisfaction and the smoothness of the surface. Of course, the procedure is not limited to single curve constraints. For example, figure 8.6 shows the creation of the "dome" shaped surface such as we have used for demonstration purposes in §2.3.1. The lower row of figures show a dome "feature": the loop is

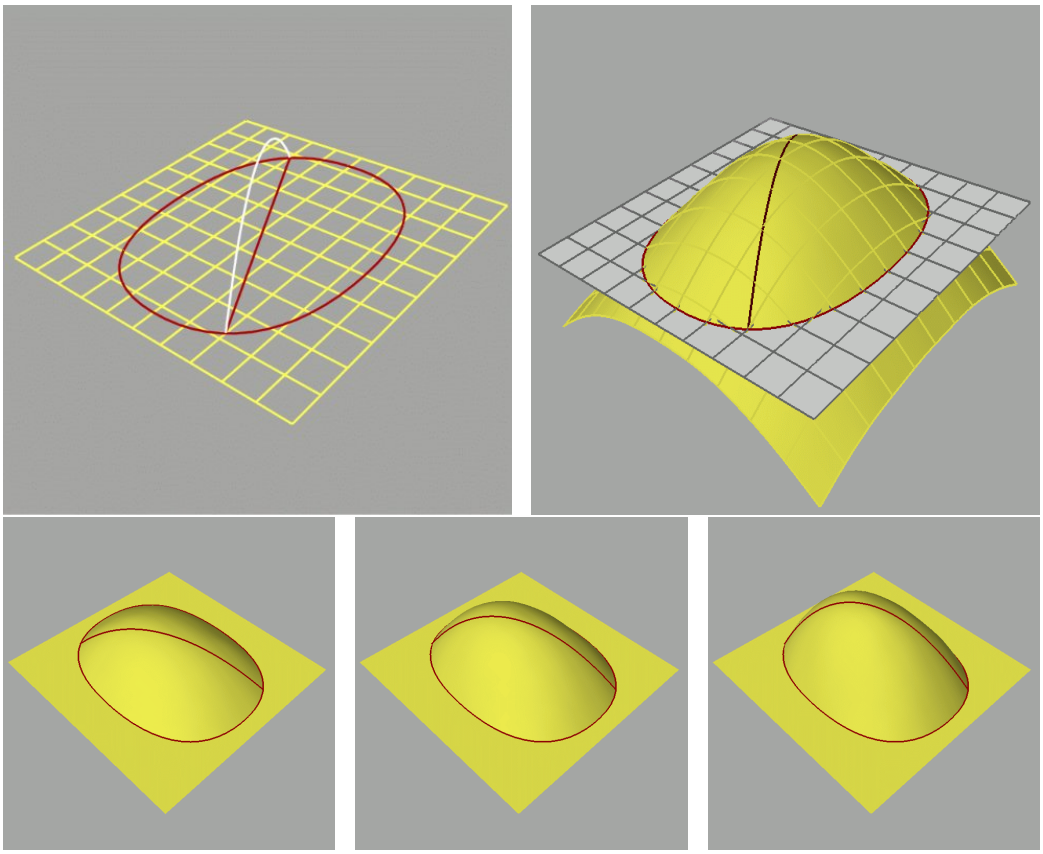


Figure 8.6: The example shows the design of a "dome" surface with two curve constraints. The closed loop remains "fixed", the other curve is modified by subsequent pen-strokes. In the lower figures the bi-cubic surface with 12×12 DOFs was trimmed outside of the fixed loop. In this example the MTSVD method delivers a solution with residuals between 10^{-5}-10^{-4} depending on the change applied to the straight curve.

constrained on both surfaces. Since its shape and position is fixed no changes are carried over to the flat surface which allows to create local features inside of a marked region on the surface, refer also to §2.3.2.

An attractive application is the creation of "n-sided" surface patches for filling n-sided holes in solid models the, so-called, surface fillets, see figure 8.7: The "traditional" approach is to fill an n-sided region with n four-sided patches introducing a midpoint and n splitting curves in a "star" constellation, see e.g. [62]. The patches are then computed from the n boundary and splitting curves with selected order of geometric continuity. The problem is that it is not clear how to determine the midpoint and the splitting curves, and that complicated "compati-

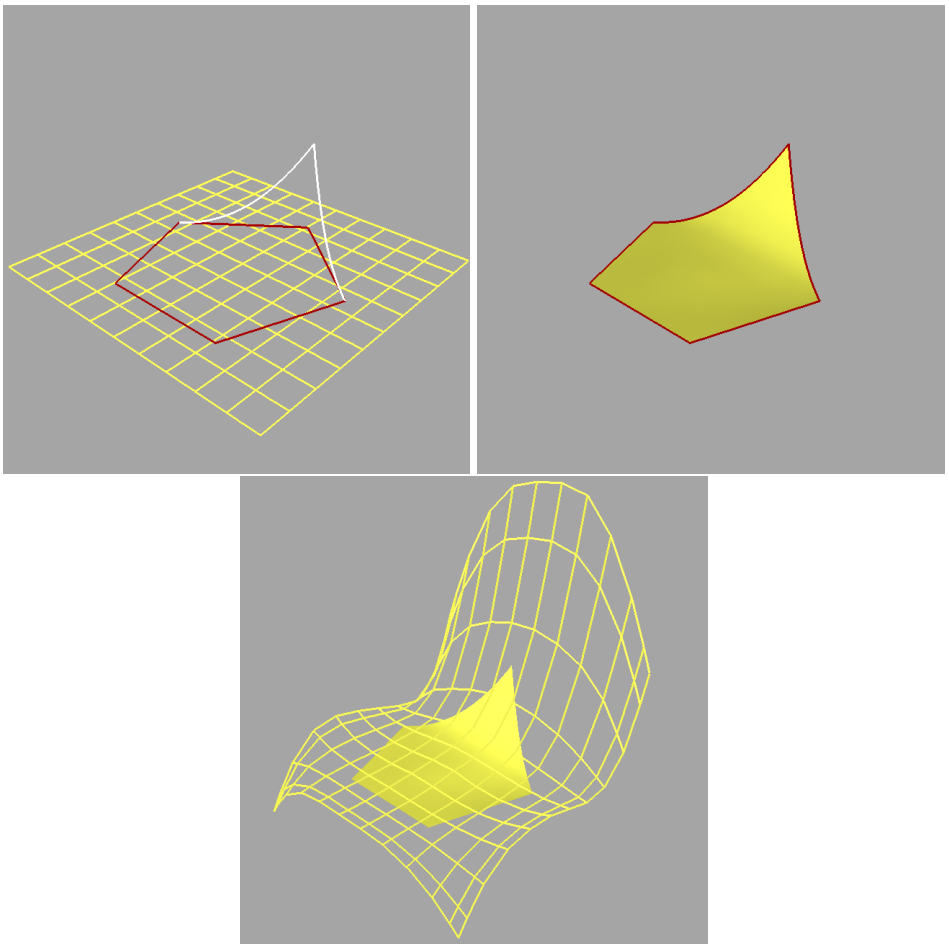


Figure 8.7: A ”5-sided” surface patch simulated by trimming a bi-cubic B-spline surface with 12×12 DOFs computed from 5 curve constraints. The lower figure shows the control mesh of the surface.

bility” pre-conditions for the curves arise, see e.g. [60]. Usage of trimmed Bezier patches was examined by Cohen and Elber in [24] and an approach similar to ours was described by Dietz in [20]: here, the boundary curves are given as strips of points and the B-spline surface is iteratively “shifted” towards the compromise between minimum thin-plate energy and constraint satisfaction. However, the paper does not mention ill-conditioning of the point-incidence equations which occurs if many point constraints are used for discrete approximation of a curve.

The advantage of using generalized curve constraints on arbitrary B-splines is that a low degree tensor-product B-spline surface which interpolates the boundary curves with sufficient accuracy can be obtained. Since usage of trimmed B-spline patches is a de-facto “standard” in almost all modeling systems the method is

more universal than usage of high-degree Bezier patches or non-rectangular surface patches. Furthermore, there is no need to deal with G-continuity conditions: the B-spline surface deforms smoothly inside of the trimmed region, see Fig. 8.7 on the right. In addition, the normals along the boundary curves can be prescribed by simply adding the equations set up as described in section 5.1. The bottom figure shows the control mesh of the surface: in this case a bi-cubic surface with 12×12 DOFs was used. 89 DOFs can be safely determined from the curve constraints, the rendered residual is $\rho_{89} \approx 10^{-7}$. The figure demonstrates that the smoothing constraint enforces an overall smooth surface without fast changes in the curvature. As in the all examples, we have used weight coefficients $\alpha_1 = 0.5$, $\alpha_2 = 0.5$, $\alpha_3 = 0$ in Eq. 8.2.

It is reasonable to assume that the specified curves do not represent contradictory requirements. For example, the endpoints of the modified curve in the "dome" model in figure 8.6 must remain fixed. For "n-sided" constellation of curves such as in figure 8.7 all subsequent curves must share a common endpoint. If more than two curves intersect in one point they must be locally co-planar, and so on. These are necessary conditions for the theoretical existence of the solution and have to be derived from the constraint graph depending on current distribution of DOFs as sketched in section 2.3.

8.3.4 Summary of the method

In conclusion, with regard to surface interpolation as proposed above we summarize the problems and subjects to future work as follows:

8.3.4.1 Efficiency

An important topic of future work will be to improve the efficiency of solving the equation system: Although the SVD provides a perfect insight into an ill-conditioned system obtaining the SVD for a general matrix is a computationally expensive procedure. The amount of work required by currently known symbolic algorithms (the Golub-Kahan method [29, Sec. 5.4 and 8.3]) to perform the SVD of a $m \times n$ matrix is approximately $O\left(m^2 + n^3\right)$. Note that the SVD needs to be computed only once in a "life-time" of a system of linear constraints – computing a surface after a curve modification from available SVD is fast and delivers a new surface shape in real time. Nevertheless, for large data sets computing the SVD may cause quite unpleasant delay-times during the design work: for moderate sizes of m and n, such as used in above examples, i.e. few hundred columns and rows the LAPACK functions GESVD or GESDD require about 1-3 seconds on an 750 MHz PC. This is acceptable; however, consider that the model may consist of several surfaces and many curve constraint. Also, the run-time grows rapidly

for larger examples: for surfaces with about 500 DOFs and more than thousand constraint equations the SVD already requires prohibitive 15-20 seconds. In addition, if we consider the "normal" constraints (fixed normal along a surface curve, see section 5.1 the spacial dimensions of the surface are no longer independent and need to be treated as separate variables. This further blows up the size of the system by factor 3 and computing the SVD for even simple examples becomes unrealistic for usage in an interactive design system. Unfortunately, the alternatives to SVD are by far not that straightforward and reliable. Other, cheaper to compute, rank-revealing factorization is for the so-called Rank-revealing QR-factorization (RR-QR) as proposed by Chan and Hansen in [12]. However, in order to apply this method, at least approximate guess on the location of the optimal truncation parameter is required. Otherwise, the effort becomes comparable to computing the SVD.

An attractive approach appears to be using iterative linear solvers for large and sparse systems of equations (see [70, 82] or [29, chapter 10]) since the matrices in this context are sparse with predictable structure (which is not utilized when computing SVD). In this case one uses the Lagrangian formulation of the constrained problem as explained in section 8.3. If the solution converges, the gain on run-time performance is considerable. Moreover, the so-called "Krylov subspace" methods exhibit similar behavior as the SVD: they tend to filter out the stable components of the solution first, thus, the discrete L-curve method can be applied to obtain the optimal truncation parameter. However, for ill-posed problems, the convergence of most iterative methods is poor. We conclude by noting that the selection of the proper iterative method and the suitable "preconditioned" will be investigated in the scope of future work.

8.3.4.2 Over-smoothing

The "smoothed" MTSVD solution tends to generate "too stiff" surfaces whenever there are not sufficiently many constraint which restrict the shape of the surface, for example, such as shown in figures 7.6 and 7.8. Considering the previously known surface $\mathbf{f}_0$ when we solve for the penalty term $\mathbf{f}'$ in Eq. 8.4 prevents the surface from collapsing into a narrow strip somewhere around the curve constraint. However, for certain curves, the surface functionals used in this work and the MTSVD method generate almost "ruled" surfaces and may smooth out "desired" features of the original surface. This is rarely the shape a designer would expect. One possibility is to use so-called "data-dependent" functionals introduced by Greiner in [31]. They take the shape of a reference surface (assumed to be known) stronger into account than Eq. 8.4 does. The surface sculpting is an iterative process, thus, a surface from previous design step is always available and should be used as such "reference". If no reference surface is available a-priori

it can be computed from sketched curves as a "raw" lest square fit to a bi-linear surface, for example. Another possibility to regulate the influence of the smoothing constraint, is to use incomplete QR-factorization when solving Eq. 8.4 for the penalty term: this allows to regulate the amount of smoothing by considering only a "lower energy" factor $\mathbf{f}'$. The disadvantage is that the QR-factorization has to be carried out with pivoting and hence, has to be computed from scratch for each new shape of the constrained curve.

8.3.4.3 Robust-estimation

The objective of robust estimation is to recognize and filter noise in measurements based on a-priori known characteristic properties of the input data, see e.g. [63, Sec. 15.7]. In our context, the constrained curves can be interpreted as measurements and the DOFs of the surface as the parameters of the mathematical model. The "noise" corresponds to amount of change required from each curve constraint. The L-curve method already implements a "filter" which automatically discards solutions with too large euclidian norm. However, the L-curve corners for too "noisy" curves (refer to §7.4) are difficult to locate. Thus filtering the changes of constrained curves before the L-curve algorithm is entered would result in more robust estimation of the optimal truncation parameter. Mathematically, the prediction is realized by introducing "a-priori covariances", [63, Chap. 15], which express the uncertainty about the exactness of the "measured" value for each constraint. Assigning an uncertainty factor to each constraint equation reflects the fact that an ill-conditioned linear system can be safely solved if the discrete Picard condition is satisfied: i.e., if the perturbation, or noise, component $\Delta\mathbf{h}$ in the parameter vector $\mathbf{h}' = \mathbf{h} + \Delta\mathbf{h}$ is reasonably smooth, see section 7.2. Thus, one could try to determine the uncertainty for each control point h_i (and for the respective constraint equation) according to the "amount" of noise Δh_i induced on h_i. However, it is not yet clear how to measure the quantity "amount of noise": intuitively, the covariances should be set proportional to the change of curvature in the difference curve $\Delta\mathbf{h}$ – if the edited curve exhibits much more changes than the initial one, the probability of obtaining a stable solution sinks; hence, a high uncertainty factors should be used. In [67], Rappoport et. al have used robust estimation of B-spline surfaces from "uncertain" (or noisy) point data. In their work it is assumed that the uncertainty is assigned to each point constraint by the user.

Chapter 9

Conclusions and Acknowledgments

In this chapter a summary of the results and contributions of this thesis is given. Open questions and topics which we have not addressed before are pointed out. The contributions of this thesis are two-fold: first, in the first part consisting of chapters 4-6 we have proposed efficient methods to set up equations for linear constraints between curves and surfaces. In the second, part (chapters 7 and 8) we have described several methods for solving of such constraints. The results gained in both parts are briefly summarized below.

The first part: obtaining the constraint equations

The core of the this part is the extension of the blossom based polynomial composition algorithm to unevaluated form: given a B-spline surface and a B-spline curve in the domain of that surface, we extract a linear transformation (a matrix) which, applied to the control points of the surface, yields the control points of the curve exactly incident on the surface as prescribed by the domain space curve. This allows to formulate linear constraints, especially, incidence of arbitrary curve on a free-form surface in a very elegant manner. Compared to variational methods used previously [83, 11] or discrete methods [20] it is more efficient, numerically more stable and does not unnecessarily increase the condition number of the matrix. We have shown how the result of the curve-surface composition can be directly expressed in B-spline format. The proposed efficient data structures (Multi-Index Trees, data structures utilizing the sparsity of the intermediate results) and careful analysis of the complexity and combinatorial properties of the algorithm allow to perform the composition in unevaluated form very fast – for practical sizes of surface and curve we obtain the matrix in fraction of a second on a standard PC.

The matrices for tangency and variational constraints can be easily obtained

re-using the methods elaborated for composition constraints: We have demonstrated how to prescribe normals along an arbitrary surface curve using the symbolic unevaluated curve representation; this constraint is used to ensure tangency of two surfaces meeting along an arbitrary curve. We have also shown how to obtain the Gaussian normal equations of the quadratic surface functionals. They are used to optimize certain differential surface properties such as curvature, variation of curvature, area and to compute thin plate energy surfaces.

The practical result of this part of the thesis is a software implementation of a powerful two-level kernel: The first, low-level level implements basic symbolic and numeric methods for dealing with B-spline polynomials. The central operation computes a B-spline blossom in unevaluated format. Based on blossoming, other B-spline operations, e.g. degree raising, knot-insertion and removal are expressed in unevaluated format. Also various kinds of B-spline interpolation and approximation are integrated. In addition the kernel provides very efficient procedures for computing derivatives, products, integrals, or integrals of products of B-splines. We have used the matrix based approach in the kernel, which allows to express most of the abovementioned operations as matrix operations without the necessity to know the values of the respective control points or the dimension of a free-form geometry. The second, high-level level implements the procedures which deliver matrices for composition, tangency and variational constraints. The interface between our mathematical kernel and the geometric 3D world is built on top of IRIT – a powerful geometric modeling kernel which contains a lot of functionality required when dealing with free-form curves and surfaces, see [22]. Wherever possible, we have avoided re-implementation of functionality already available from IRIT; therefore, in many parts, our kernel depends on IRIT-data structures and IRIT-libraries.

The second part: numerical methods for constraint--solving

The central difficulty is that the inverse linear problem associated with generalized curve-surface incidence constraints is ill-posed. Hence, the usual methods such as Gaussian elimination or QR-decomposition cannot be applied in straightforward manner. We have proposed to use regularization based on Singular Value Decomposition (SVD). The so-called L-curve method discussed here provides a generic and reliable tool to single out a numerically stable solution such that best possible satisfaction of defined constraints is achieved. However, even the SVD along with the L-curve method cannot be applied blindly. Tensor-product surfaces deformed along arbitrary incident curves exhibit unwanted deformations due to the

rectangular structure of the model space. We have discussed a geometric and an algebraic method to remove this, so-called, “Surface aliasing effect”. The former method reparametrizes the surface such that a general curve constraint is converted to iso-parametric curve constraint which can be easily solved by standard linear algebra methods without aliasing. The reparametrized surface is computed by means of the approximated surface-surface composition algorithm. While this is not possible symbolically, an arbitrary accurate approximation can be obtained using curve network interpolation. This method is used to constrain incidence of a rectangular region on a surface – the user can add editable features to a free-form surface. The disadvantage of this approach is that the surface can be “adjusted” only with regard to one curve constraint. Nevertheless it is an interesting alternative whenever a surface should be sculpted inside a topologically rectangular region. The latter method circumvents this limitation by stating additional constraints which suppress or completely remove the aliasing. Formally we solve a constrained least square problem which minimizes a surface objective function (expressed by a linear combination of surface smoothing functionals) subject to defined curve constraints. Because of the inherent ill-posedness of the problem, sophisticated numerical methods has to be applied in order to obtain a set of degrees of freedom which are sufficient to satisfy given constraints. The remaining free variables are used to enforce an optically pleasing shape of the surface. We have applied the Modified Truncated SVD algorithm which determines a compromise between an optically pleasant shape of the surface and constraint satisfaction in a particularly efficient manner. Compared to methods proposed by Gossard, Welch and Witkin in [83, 11] where pivoted Gaussian elimination was used to single out a numerical rank of the matrix the MTSVD algorithm is much more reliable. Furthermore, the built in coherence between previously known and current solution prevents the surface from collapsing to a point or curve whenever the constraints do not span a sufficiently large solution space. A disadvantage of the method is that a modification of one usually curve influences the whole surface – thus, introducing local modifications is only possible when the surface is split into several trimmed parts connected by curve constraints.

In our implementation, we have used the Matrix Template Library (MTL, see [78]) for storage and manipulation of matrices in both parts. MTL is a template-based software package based on principles of Generic programming [3]. In its philosophy it is similar to the famous Standard Template Library: it provides a set of template data-types and procedures for dealing with matrices and vectors of various formats (e.g. banded, symmetric, general sparse, etc.). Our overall experience with MTL is good, although the version we have used (2.1.2-19) was quite buggy. The robust linear solver builds on top of MTL and LAPACK [1], [2]. The part of the software which analyzes the L-curve is quite independent of the algebraic method chosen to solve the equation system. Hence, the algorithms for

L-curve interrogation can still be used even if the SVD is replaced by other factorizations (e.g. the faster Rank Revealing QR-decomposition) or by an iterative method.

Bibliography

[1] D. C. Anderson and R. H. Crawford. Knowledge management for preliminary computer aided mechanical design. In T. Sata, editor, *Organization of engineering knowledge for product modelling in computer integrated manufacturing*, pages 15–34. IFIP, Elsevier, 1989.

[2] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide Third Edition*. Software, Environments, and Tools 9. Society for Industrial and Applied Mathematics, SIAM, 3. edition, 1999.

[3] Matthew H. Austern. *Generic Programming and the STL: Using and Extending the C++ Standard Template Library*. Professional Computing. Addison-Wesley, 1999.

[4] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R.Pozo, C. Romine, and H. Van der Vorst. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Edition*. SIAM, Philadelphia, PA, 1994.

[5] G. P. Bonneau, H. Hagen, and S. Hahmann. Variational Surface Design and Surface Interrogation. *Computer Graphics Forum*, 3(12):447–459, 1993.

[6] C. De Boor, L. Lyche, and L. L. Schumaker. On calculating with B-splines: Integration. In L. Collatz, H. Werner, and G. Meinardus, editors, *Numerische Methoden der Approximationstheorie*, pages 123–146. Birkhhäuser, Basel, 1976.

[7] B. Brüderlin, U. Döring, R. Klein, and P. Michalik. Declarative geometric modeling with constraints. In A. Iwainsky, editor, *Conference proceedings CAD 2000*, Berlin, March 2000. GFAI.

[8] B. Brüderlin, U. Döring, P. Michalik, D. Beier, and H. Oestreich. Programmable Features in CAD. In *Proceedings, ISATA*, Dublin, Ireland, 2000.

[9] G. Brunnett, H. Hagen, and P. Santarelli. Variational Design of Curves and Surfaces. *Surv. Math. Industry*, 3(27), 1993.

[10] D. Calvetti, P. C. Hansen, and L. Reichel. L-curve curvature bounds via Lanczos bidiagonalization. *Electronic Transactions on Numerical Analysis*, May 2001. To obtain under `www.imm.dtu.dk`.

[11] G. Celniker and W. Welch. Linear constraints for deformable B–spline surfaces. *Computer Graphics*, 25(2):171–174, March 1992.

[12] T. Chan and P. C. Hansen. Computing Truncated Singular Value Decomposition Least Square Solutions by rank revealing QR-Factorizations. *SIAM J. Sci. Stat. Comput.*, 11(3):519–530, May 1990.

[13] S. Coquillart. Extended free–form deformation: a sculpturing tool for 3D geometric modeling. *Computer Graphics*, 24(4):187–196, August 1990.

[14] M. Cox. Algorithms for spline curves and surfaces. In L. A. Piegl, editor, *Fundamental Developments of Computer–Aided Geometric Modeling*, pages 51–75. Academic Press Limited, 1993.

[15] P. de Casteljau. *Formes à Pôles*. Hermes, Paris, 1985.

[16] P. de Casteljau. *Le Lissage*. Hermes, Paris, 1990.

[17] T. DeRose. Composing Bézier Simplices. *ACM Trans. Graph.*, 7(3):198–221, July 1988.

[18] T. DeRose, R. Goldman, H. Hagen, and S. Mann. Functional composition via blossoming. *ACM Transactions on Graphics*, 12(2), April 1993.

[19] U. Dietz. B-Spline Approximation with Energy Constraints. In *Reverse Engineering*. B.G. Teubner, Stuttgart, 1996.

[20] U. Dietz. Creation of Fair B-Spline Surface Fillets. In *Creating Fair and Shape Preserving Curves and Surfaces*. B.G. Teubner, Stuttgart, 1998.

[21] U. Doering, P. Michalik, and B. Brüderlin. A constraint–based shape modeling system. In *Geometric Constraint Solving & Applications*. Springer Verlag, June 1998.

[22] G. Elber. *Users' Manual – IRIT, A solid modeling Program*. Technion institute of Technology, Haifa, Israel, 1990–1996.

[23] G. Elber. *Free form surface analysis using a hybrid of symbolic and numerical computations*. PhD thesis, University of Utah, 1992.

[24] G. Elber and E. Cohen. Filleting and rounding using trimmed tensor product surfaces. In *Proceedings The fourth ACM/IEEE Symposium on Solid Modeling and Applications*, pages 201–216, May 1997.

[25] G. Farin. *Curves and Surfaces for CAGD*. Computer Science and Scientific Computing. Academic Press, Inc., 3. edition, 1992.

[26] R. Fierro, G. H. Golub, P. C. Hansen, and D. P. O'Leary. Regularization by Truncated Total Least Squares. In J.G. Lewis, editor, *Proceedings of the Fifth SIAM Conference on Applied Linear Algebra*, pages 250–254, Philadelphia, PA, 1994. SIAM Press.

[27] G. H. Golub, P. C. Hansen, and D. P. O'Leary. Tikhonov Regularization and Total Least Squares. Journal on Matrix Analysis and Applications, to appear.

[28] G. H. Golub, V. Klema, and G. W. Stewart. Rank Degeneracy and Least Squares Problems. Technical Report 456, Department of Computer Science, Institute for Advanced Computer Studies (UMIACS), University of Maryland at College Park, `www.cs.umd.edu`, 1976.

[29] G. H. Golub and C. E. van Loan. *Matrix Computations*. John Hopkins Series in the Mathematical Sciences. The John Hopkins University Press, 2 edition, 1989.

[30] W. J. Gordon. Sculptured surface definition via blending–function methods. In L. A. Piegl, editor, *Fundamental Developments of Computer–Aided Geometric Modeling*, pages 117–134. Academic Press Limited, 1993.

[31] G. Greiner. Variational design and fairing of spline surfaces. *Computer Graphics Forum (Proc. Eurographics '94)*, (3):143–154, 1994.

[32] G. Greiner and H.-P. Seidel. Automatic modeling of smooth spline surfaces. In V. Skala N. Magnenat-Thalmann, editor, *Proc. WSCG '97*, pages 665–675, 1997.

[33] J. Griessmair and W. Purgathofer. Deformation of solids with trivariate b-splines. In W. Hanmann, F.R.A. Hopgood, and W. Strasser, editors, *Eurographics '89*, pages 137–148. Elsevier Science Publishers (North Holland), 1989.

[34] C. W. Groetsch, editor. *The Theory of Tikhonov Regularization for Fredholm Equations of the First Kind*. Pitman, Boston, 1984.

[35] P. C. Hansen. Truncated Singular Value Decomposition Solutions to Discrete Ill–posed Problems With Ill–determined Numerical Rank. *SIAM J. of Scientific Computing*, 11(3):519–530, May 1990.

[36] P. C. Hansen. *Regularization Tools*. Department of Mathematical Modeling, Technical University of Denmark, `www.imm.dtu.dk`, March 1998.

[37] P. C. Hansen. The L-curve and its use in the numerical treatment of inverse problems. In invited paper for P. Johnston, editor, *Computational Inverse Problems in Electrocardiology*, pages 119–142, Southampton, 2001. WIT Press.

[38] P. C. Hansen, T. Sekh, and H. Shibahashi. The Modified Truncated SVD Method for Regularization in General Form. *SIAM J. of Scientific Computing*, 13(5):1142–1150, September 1992.

[39] J. Hayes. Nag algorithms for the approximation of functions and data. In J. Mason and M. Cox, editors, *Algorithms for Approximation*, pages 653–668. Clarendon Press, Oxford, 1988.

[40] J. Hoschek and U. Dietz. Smooth B-Spline Surface Approximation to Scattered Data. In *Advanced Course on FAIRSHAPE*. B.G. Teubner, Stuttgart, 1996.

[41] J. Hoschek and D. Lasser. *Fundamentals of Computer Aided Geometric Design*. A K Peters Ltd., 2. edition, 1989.

[42] C. Hsu, G. Alt, Z. Huang, E. Beier, and B. Brüderlin. A Constraint–based Manipulator Toolset for Editing 3D Objects. In *Solid Modeling 1997*, Atlanta, Georgia, 1997. ACM Press.

[43] C. Hsu and B. Brüderlin. A graph-based degree of freedom analysis algorithm to solve geometric constraint problems. In W. Strasser and J. Rossignac, editors, *Theory and Practice of Geometric Modeling*. Springer Verlag, 1997.

[44] C. Hsu and B. Brüderlin. A hybrid constraint solver using exact and iterative geometric constructions. In D. Roller and P. Brunet, editors, *CAD Systems Development – Tools and Methods*. Springer Verlag, 1997.

[45] Roman Kazinnik and Gershon Elber. Orthogonal Decomposition of Non-Uniform Bspline Spaces using Wavelets. *Computer Graphics forum*, 16(3):27–38, September 1997.

[46] A. Kielbasinsky and H.Schwetlick. *Numerische lineare Algebra, eine computerorientierte Einführung*. Mathematik für Naturwissenschaft und Technik. Deutscher Verlag der Wissenschaften, Berlin, 1988.

[47] D. H. Kim, P. Michalik, and B. Brüderlin. Sketching B-spline curves and surfaces. In P. Slusallek D. Saupe, editor, *Graphiktag 2001*, pages 68–76, Tübingen, November 2001.

[48] A. Kolb, H. Pottmann, and H.-P. Seidel. Fair surface reconstruction using quadratic functionals. In *Eurographics Proc. '95*, pages 469–479. Eurographics, Blackwell Publishers, 1995.

[49] F. Lazarus, S. Coquillart, and P. Jancéne. Axial deformations: An intuitive deformation technique. *Computer–Aided Design*, 26(8):607–613, August 1994.

[50] W. Liu. Programming support for blossoming. Master's thesis, University of Waterloo, Waterloo, 1995.

[51] C. T. Loop. *Generalized B-spline Surfaces of Arbitrary Topological Type*. PhD thesis, University of Washington, 1992.

[52] T. Lyche and K. Mørken. A Discrete Approach to Knot Removal and Degree Reduction Algorithms for Splines. In J. Mason and M. Cox, editors, *Algorithms for Approximation*, pages 67–82. Oxford University Press, 1987.

[53] S. Mann and W. Liu. An analysis of polynomial composition algorithms. Technical Report CS–95–24, University of Waterloo, Computer Science Department, 1995.

[54] P. Michalik and B. Brüderlin. Computing curve–surface incidence constraints efficiently. In *Proceedings Swiss Conference on CAD/CAM*, February 1999.

[55] P. Michalik and B. Brüderlin. A constraint-based method for sculpting freeform surfaces. In G. Brunnett and H. P. Bieri, editors, *Computing, special issue of the Dagstuhl Seminar on Geometric Modelling*. Springer Verlag, 2000.

[56] P. Michalik and B. Bruederlin. Introducing parametrization in surface models by means of geometric constraints. In *Proceedings Scanning and Human Body Modeling*, Paris, France, May. 4-5 2001.

[57] P. Michalik, D. Kim, and B. Bruderlin. Sketch- and Constraint-based Design of B-spline Surfaces. In *Proceedings of International Conference on Solid Modeling*, Saarbrücken, Germany, 2002.

[58] K. Mørken. Some identities for products and degree raising of splines. *Constructive Approximation*, 7:195–208, 1991.

[59] On-Line Computer Graphics Notes. Free-form deformations. `http://graphics.cs.ucdavis.edu/GraphicsNotes`.

[60] J. Peters. Geometric continuity. Dept C.I.S.E, University of Florida, `www.cise.ufl.edu/research/SurfLab/papers`, February 2001.

[61] L. Piegl and W. Tiller. *The Nurbs Book*. Springer Verlag, 1995.

[62] L. Piegl and W. Tiller. Filling n-sided regions with NURBS patches. *The Visual Computer*, (15):77–89, 1999.

[63] W. Press, S. Teukolsky, W. Vetterling, and B. Flannery. *Numerical Recipies in C: The art of scientific computing*. Cambridge Unversity Press, 1992.

[64] L. Ramshaw. Beziers and B-splines as multiaffine maps. In R. A. Earnshaw, editor, *Theoretical Foundations of Computer Graphics and CAD*, number 40 in NATO ASI Series F: Computer and Systems Sciences, pages 757–776. Springer-Verlag, 1987.

[65] L. Ramshaw. Blossoming: A connect–the–dots approach to splines. Technical Report 19, Digital Systems Reaearch Center, Palo Alto CA, June 1987.

[66] L. Ramshaw. Blossoms are polar forms. *Computer Aided Geometric Design*, 6:323–358, January 1989.

[67] A. Rappoport, Y. Hel-Or, and M. Werman. Interactive design of smooth objects with probabilistic point constraints. *ACM Transactions on Graphics*, 13(2):156–176, April 1994.

[68] J. J. Risler. *Mathematical Methods for CAD*. Press Syndicate of the University of Cambridge, 1991.

[69] B. W. Rust and W. R. Burrus. *Mathematical Programing and the Numerical Solutions of Linear Equations*. Modern Analytic and computational methods in science and mathematics. American Elsevier Publishing Company, New York, N.Y., 1972.

[70] Y. Saad. *Iterative Methods For Sparse Linear Systems*. The PWS Series in Computer Science. PWS Publishing Company, Boston, MA, 1996.

[71] O. W. Salomons. *Computer support in the design of mechanichal products*. PhD thesis, Universiteit Twente, Groeningen, 1995.

[72] L. L. Schumaker and F. I. Utreras. On Generalized Cross validation for Tensor Smoothing Splines. *SIAM J. Sci. STAT. Comput.*, 11(4):713–731, July 1990.

[73] T. Sederberg and S. Parry. Free–form deformation of solid geometric models. In *Proceedings SIGGRAPH '86*, pages 151–160, 1986.

[74] H. P. Seidel. Knot insertion from a blossoming point of view. *Computer Aided Geometric Design*, 5(1):81–86, 1988.

[75] H. P. Seidel. Computing B-Spline Control Points. In W. Strasser and H. P. Seidel, editors, *Theory and Practice of Geometric Modeling*, pages 17–32. Springer Verlag, 1989.

[76] H. P. Seidel. An introduction to polar forms. *IEEE Comp. Graph. Appl.*, 1(13), 1993.

[77] H. P. Seidel, W. Dalmen, and C. A. Micchelli. Blossoming begets B-splines built better by B-patches. *Math. Comp*, (59), 1993.

[78] J. G. Siek. A Modern Framework for High Performance Numerical Linear Algebra. Master's thesis, University of Notre Dame, Notre Dame, Indiana, April 1999. `www.lsc.nd.edu/research/mtl`.

[79] K. Singh and E. Fiume. Wires: A geometric deformation techinque. In *Proceedings SIGGRAPH '98*, 1998.

[80] L. N. Trefethen and D. Bau. *Numerical Linear Algebra*. Sociaty for Industrial and Applied Mathematics, Philadelphia, PA, 1997.

[81] A. H. Vermeulen, R. H. Bartels, and G. R. Heppler. Integrating Product of B-Splines. *SIAM J. Sci. Stat. Computing*, 13(4):1025–1038, July 1992.

[82] R. Weiss. *Parameter-Free Iterative Linear Solvers*. Akademie Verlag Gmbh., Berlin, 1996.

[83] W. Welch and A. Witkin. Variational surface modeling. *Computer Graphics*, 26(2):157–165, July 1992.

[84] Todd Will. Introduction to the Singular Value Decomposition. `http://www.davidson.edu`.

Appendix A

Notations and Definitions

The geometric elements we deal with are B-Spline curves and surfaces in 3-dimensional Euclidian space (denoted by E^3). Each geometric element possesses a defined and fixed number of *degrees of freedom*: Degrees of freedom of a geometric element X are $n \times 3$ real scalar variables which uniquely determine X. This will be abbreviated by the relationship $dofs(X)$ (read: "degrees of freedom of element X") defined as:

$$dofs(X) : X \rightarrow R^{n \times 3}$$

Degrees of freedom for each spacial dimension are an element of vector space over real numbers R^n; we denote vectors in R^n by lower case bold letters, $\mathbf{x}$ and matrices in $R^{m \times n}$ by upper case bold letters, $\mathbf{X}$:

$$dofs(X) = \begin{bmatrix} x_{11} & x_{12} & x_{13} \\ & \vdots & \\ x_{n1} & x_{n2} & x_{n3} \end{bmatrix} = \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \mathbf{x}_3 \end{bmatrix} = \mathbf{X},\ \mathbf{X} \in R^{n \times 3}$$

We say that a geometric element X possesses n degrees of freedom $\mathbf{x}_i$ for each spacial dimension, or simply, it has the size $n \times 3$. An element X is determined if the values of all $dofs(X)$ are known. An element X is unknown if the values of $dofs(X)$ are not known. We will use corresponding symbols for geometric elements and their degrees of freedom, i.e.: $dofs(X) = \mathbf{X}$.

A.1 B-Splines

The term "B-Splines" is frequently used to denote certain category of free-form curves and surfaces. However, from mathematical point of view, B-Splines (Basis Splines) denote a specific basis for a vector space of piecewise polynomial functions. Therefore, we distinguish between B-Splines and functions, curves and

surfaces defined in terms of the B-Spline basis. In the following three paragraph we will define B-Splines and univariate and two-variate B-Spline functions. The latter two paragraphs extend the definition to[1] B-Spline curves and surfaces and establish the notation for denoting the degrees of freedom of B-spline curves and surfaces.

A.1.1 Definition

A.1.1.1 B-Splines and B-Spline functions

Let $\langle a, b\rangle \subseteq R^1$ be a given interval of R^1. Let also be given integers $d > 0$, $n \geq 1$ and a sequence τ of $n + d + 1$ real numbers τ_i such that

1. $\tau_i \leq \tau_{i+1}$, $0 \leq i \leq n + d$
2. $\tau_j \leq a$, $0 \leq j \leq d$
3. $\tau_j \geq b$, $n \leq j \leq n + d$

Such τ_i are called *knots* and the sequence τ is called *knot vector*. We associate with each knot τ_i an integer value m_i, $0 \leq m_i \leq d$, called *multiplicity* of τ_i, which counts the number of occurances of τ_i in the sequence τ.

By $\mathcal{P}_{d,\tau,n}$ we will denote a n-dimensional vector space of univariate piecewise polynomial functions of polynomial degree d on interval $\langle a, b\rangle$ which are C^{d-m_i} continuous at knots τ_i. The dimension of $\mathcal{P}_{d,\tau,n}$ is n, if $m_i \leq d$, $0 \leq i \leq n + d$, see [68]. Elements of $\mathcal{P}_{d,\tau}$ are piecewise polynomial functions:

$$c : \Omega_t \rightarrow R^1,\ c = c(t),\ t \in \Omega_t \subseteq R^1,\ \Omega_t = \langle a, b\rangle$$

A set of n linearly independent piecewise polynomial functions $\{b_i^{d,\tau}(t) : 0 \leq i < n\}$, called ***B-Splines*** (Basis Splines), defined by recursion

$$\begin{aligned} b_i^{0,\tau}(t) &= \begin{cases} 1 & \text{if } \tau_i \leq t < \tau_{i+1} \\ 0 & \text{otherwise} \end{cases} \\ b_i^{d,\tau}(t) &= \frac{t-\tau_i}{\tau_{i+d}-\tau_i} b_i^{d-1,\tau}(t) + \frac{\tau_{i+d+1}-t}{\tau_{i+d+1}-\tau_{i+1}} b_{i+1}^{d-1,\tau}(t) \end{aligned} \tag{A.1}$$

constitutes a basis for $\mathcal{P}_{d,\tau}$, see [68], [25] ,[41]. Therefore any $c(t) \in \mathcal{P}_{d,\tau}$ may be written as a linear combination of n B-Splines:

$$c(t) = \sum_{i=0}^{n-1} c_i b_i^{d,\tau}(t)$$

[1] In fact, the dimension is not limited to three. Curves and surfaces in arbitrary dimension are defined in exactly the same way, by simply considering all dimensions independently.

Each $c(t) \in \mathcal{P}_{d,\tau,n}$ is uniquely determined by its coefficients $\{c_i \in R^1 : 0 \leq i < n\}$ and a fixed set of B-Splines $\{b_i^{d,\tau}(t) : 0 \leq i < n\}$. The coefficients c_i are the degrees of freedom of a B-Spline function $c(t)$.

A.1.1.2 Scalar product notation

In order to simplify notation, we denote a basis (a set of basis functions) by a vector

$$\mathbf{b}_t^T = \left[\begin{array}{ccc} b_0^{d,\tau}(t) & \cdots & b_{n-1}^{d,\tau}(t) \end{array} \right]$$

and write $c(t)$ as a *scalar product* of the basis and the coefficient vector:

$$c(t) = \left[\begin{array}{ccc} b_0^{d,\tau}(t) & \cdots & b_{n-1}^{d,\tau}(t) \end{array} \right] \left[\begin{array}{c} c_0 \\ \vdots \\ c_{n-1} \end{array} \right] = \mathbf{b}_t^T \mathbf{c}$$

It follows that

$$dofs\left(c(t)\right) = \left[\begin{array}{c} c_0 \\ \vdots \\ c_{n-1} \end{array} \right] = \mathbf{c},\ \mathbf{c} \in R^n$$

A.1.1.3 Properties of B-Splines

With this notation B-Splines have following properties (see e.g. [68] for detailed proofs):

1. $b_i^{d,\tau}(t)$ consists of polynomial segments of degree d for $\tau_j \leq t < \tau_{j+1},\ d \leq j \leq n$
2. $b_i^{d,\tau}(t)$ is a piecewise polynomial of degree d for $\tau_d \leq t < \tau_n$
3. $b_i^{d,\tau}(t) \begin{cases} \geq 0; \text{ if } \tau_i \leq t < \tau_{i+d+1};\ d \leq i \leq n \\ 0 \qquad \text{otherwise} \end{cases}$
4. $\sum_{i=0}^{n-1} b_i^{d,\tau}(t) = 1$

These properties will be of importance in the scope of the thesis. Furthermore, we will make use of following properties, induced by properties 1 and 2: Let $x(t) \in \mathcal{P}_{d,\tau,m}$ and $y(t) \in \mathcal{P}_{l,v,n}$ be two B-Spline functions. Then the following propositions applies:

1. B-Spline functions are closed under multiplication, i.e. a product of two B-Spline functions is also a B-Spline function:

$$x(t) \cdot y(t) = z(t);\ z(t) \in \mathcal{P}_{d+l,\pi,p}$$

2. B-Spline functions are closed under differentiation and integration:

$$\frac{d}{dt}x(t) = z(t);\ z(t) \in \mathcal{P}_{d-1,\pi,n-1} \Longleftrightarrow \int z(t)dt = x(t) + C$$

The advantage of using the B-Spline basis compared, for example, to the power basis for piecewise polynomials is that the coefficients c_i have a geometric representation: Let (ξ_i, c_i) with $\xi_i = \frac{1}{d}\sum_{j=i}^{i+d-1}\tau_i$ be the coordinates of a point in E^2. The set of points $\{(\xi_i, c_i) : 0 \leq i < n\}$, called *control points*, constitute the so-called, *control polygon* of a B-spline function $c(t)$. The graph of the function $c(t)$ approximates the shape of a control polygon which allows to conclude on the shape of a B-Spline function by only considering the shape of its control polygon. This is an important property in the context of designing B-Spline curves and surfaces, defined in §A.1.2 and §A.1.3.

A.1.1.4 Two-variate B-Splines and B-Spline functions

Let $\mathcal{P}_{k,\tau}$ and $\mathcal{P}_{l,v}$ be two B-Spline spaces of dimensions m and n. Denote the B-Spline bases for $\mathcal{P}_{k,\tau}$ and $\mathcal{P}_{l,v}$ by $\mathbf{b}_u^T$ and $\mathbf{b}_v^T$. Furthermore, define a set of n B-Spline functions $\{s_j(u) \in \mathcal{P}_{k,\tau} : 0 \leq j < n\}$ with coefficients $s_{i,j}; 0 \leq i < m$. Then a two-variate function

$$s : \Omega_{u,v} \rightarrow R^1,\ s = s(u,v),\ (u,v) \in \Omega_{u,v} \subseteq R^2,\ \Omega_{u,v} = \Omega_u \times \Omega_v = \langle a, b\rangle \times \langle c, d\rangle$$

defined by

$$\begin{aligned} s(u,v) &= \textstyle\sum_{j=0}^{n-1} s_j(u) b_j^{l,v}(v) \\ &= \textstyle\sum_{j=0}^{n-1} \left(\sum_{i=0}^{m-1} s_{i,j} b_i^{k,\tau}(u)\right) b_j^{l,v}(v) \\ &= \textstyle\sum_{j=0}^{n-1}\sum_{i=0}^{m-1} s_{i,j} b_i^{k,\tau}(u) b_j^{l,v}(v) \end{aligned} \tag{A.2}$$

is called a two-variate *tensor-product* B-Spline function of polynomial degree k in u-direction and polynomial degree l in v-direction. Setting $dofs(s_j) = \mathbf{s}_j$, and using the scalar product notation for univariate B-Spline function yields a 2-tensor in variables u and v:

$$\begin{aligned} s(u,v) &= \textstyle\sum_{j=0}^{n-1} \left(\mathbf{b}_u^T \mathbf{s}_j\right) b_j^{l,v}(v) = \mathbf{b}_u^T \sum_{j=0}^{n-1} \mathbf{s}_j b_j^{l,v}(v) \\ &= \mathbf{b}_u^T \begin{bmatrix} \mathbf{s}_0 & \cdots & \mathbf{s}_{n-1} \end{bmatrix} \mathbf{b}_v \\ &= \mathbf{b}_u^T \mathbf{S} \mathbf{b}_v \end{aligned}$$

Each $s(u,v)$ is uniquely determined by its coefficients $s_{i,j}$ and fixed sets of B-Splines $\mathbf{b}_u^T$ and $\mathbf{b}_v^T$ in variables u and v.

A.1.1.5 Scalar product notation for tensor-products

The scalar product notation is established by introducing two-variate B-spline basis functions $b_I^{k,\tau,l,\nu}(u,v) = b_i^{k,\tau}(u)b_j^{l,\upsilon}(v)$ with integer indices I such that

$$I = i + mj,\ 0 \leq i < m \wedge 0 \leq j < n \Rightarrow 0 \leq I \leq (m-1)(n-1)$$

The linear subspace formed by the products $b_i^{k,\tau}(u)b_j^{l,\upsilon}(v)$ is called a tensor product of $\mathcal{P}_{k,\tau}$ and $\mathcal{P}_{l,\upsilon}$, and will be denoted by $\mathcal{P}_{k,\tau} \times \mathcal{P}_{l,\upsilon}$. Similarly as in the univariate case we simplify the notation for two-variate basis functions by introducing a vector notation:

$$\mathbf{b}_{u,v}^T = \left[\ b_0^{k,\tau,l,\upsilon}(u,v) \quad \cdots \quad b_{(m-1)(n-1)}^{k,\tau,l,\upsilon}(u,v)\ \right]$$

We expand $\mathbf{S}$ in row-major order yielding a vector $\mathbf{s}$ of size mn:

$$\mathbf{s} = \left[\ s_{0,0} \quad \cdots \quad s_{m-1,0} \quad \cdots \quad s_{0,n-1} \quad \cdots \quad s_{m-1,n-1}\ \right]^T$$

Using the vector notation for two-variate B-Spline basis yields

$$s(u,v) = \sum_{I=0}^{(m-1)(n-1)} s_I b_I^{k,\tau,l,\upsilon}(u,v) = \mathbf{b}_{u,v}^T \mathbf{s}$$

and

$$dofs\left(s(u,v)\right) = \mathbf{s},\ \mathbf{s} \in R^{mn}$$

A.1.1.6 Properties of two-variate B-Spline functions

The properties of univariate B-Spline functions extend to two-variate B-Spline functions by considering each variable separately. In particular, the *control mesh* of a two-variate B-Spline function is defined as follows: Let $(\mu_i, \nu_j, s_{i,j})$ be the coordinates of a point in E^3 with $\mu_i = \frac{1}{k}\sum_{p=i}^{i+k-1} \tau_p$ and $\nu_j = \frac{1}{l}\sum_{p=i}^{j+l-1} \upsilon_p$. The set of points $\{(\mu_i, \nu_j, s_{i,j}) : 0 \leq i < m \wedge 0 \leq j < n\}$, called *control points* of $s(u,v)$, constitute the so-called, *control mesh* of a B-spline function $s(u,v)$. As in the univariate case the graph of the function $s(u,v)$, a surface in E^3, approximates the shape of the control mesh.

The 2-tensors of that kind have the property that setting $u = const.$ or $v = const.$ yields a univariate B-Spline function in the other non-constant variable: For example, setting $v = const. = \alpha$ in equation A.2 and substituting $b_j^{l,\upsilon}(\alpha) = \beta_j,\ 0 \leq j < n$ yields

$$s(u,\alpha) = \sum_{j=0}^{n-1} s_j(u)\beta_j,\ \ s(u,\alpha) \in \mathcal{P}_{\tau,k}$$

The graph of $s(u, v = const.)$ is a curve in E^3 incident on $s(u,v)$ called an *isoparametric* curve in u-direction.

A.1.2 B-Spline curves

Let $c_j(t) \in \mathcal{P}_{d,\tau},\ 0 \leq j < 3$ be three B-Spline functions of dimension n as defined in §A.1.1. A parametric curve in E^3 defined by

$$C : \Omega_t \rightarrow E^3,\ C = C(t) = (c_0(t), c_1(t), c_2(t)),\ t \in \Omega_t$$

is called a B-Spline curve of degree d. In the following we denote points and vectors in Euclidian spaces by upper italic letters and use $(\cdots)$ to list their coordinates. Denote by $c_{i,j}$ the i-th coefficient of $c_j(t)$. The coefficients $c_{i,j}$ for fixed i and $1 \leq j \leq 3$ define a point in E^3 with coordinates $(c_{i,1}, c_{i,2}, c_{i,3})$, called *control point* of a B-Spline curve. The set of n control points $\{C_i = (c_{i,1}, c_{i,2}, c_{i,3}) : 0 \leq i < n\}$ constitutes a *control polygon* of a B-Spline curve. The curve $C(t)$ may be then written as a linear combination of control points C_i and basis functions:

$$C(t) = \sum_{i=0}^{n-1} C_i b_i^{d,\tau}(t)$$

Since each $c_j(t)$ is fully determined by its coefficients, see §A.1.1, a B-Spline curve is fully determined by its control points. In order to obtain the $dofs(C)$ relation we switch to scalar product notation for each $c_j(t)$. We define a matrix of control points $\mathbf{C}$:

$$\mathbf{C} = \begin{bmatrix} C_i \\ \vdots \\ C_{n-1} \end{bmatrix}$$

which yields

$$\begin{aligned} C(t) &= \begin{bmatrix} c_1(t) & c_2(t) & c_3(t) \end{bmatrix} \\ &= \begin{bmatrix} b_0^{d,\tau}(t) & \cdots & b_{n-1}^{d,\tau}(t) \end{bmatrix} \begin{bmatrix} C_i \\ \vdots \\ C_{n-1} \end{bmatrix} \\ &= \mathbf{b}_t^T \mathbf{C} \end{aligned}$$

Hence, the coordinates of the control points are the degrees of freedom of a free-form curve:

$$dofs\,(C(t)) = \mathbf{C},\ \mathbf{C} \in R^{n \times 3}$$

A.1.3 B-Spline surfaces

Let $s_d(u, v) \in \mathcal{P}_{k,\tau} \times \mathcal{P}_{l,v},\ 0 \leq d < 3$ be three 2-variate B-Spline functions of dimension mn as defined in §A.1.1. A parametric surface in E^3 defined by

$$S : \Omega_{u,v} \rightarrow E^3,\ S = S(u, v) = (s_0(u, v), s_1(u, v), s_2(u, v)),\ (u, v) \in \Omega_{u,v}$$

is called a tensor-product B-Spline surface of polynomial degree k in u and degree l in v. Denote by $s_{i,j,d}$ the coefficient of $s_d(u,v)$ with indices i,j. The coefficients $s_{i,j,d}$ for fixed i,j and $0 \leq d < 2$ define a point in E^3 called *control point* of a B-Spline surface. The set of mn control points $\{S_{i,j} = (s_{i,j,0}, s_{i,j,1}, s_{i,j,2}) : 0 \leq i < m, 0 \leq j < n\}$ constitutes a *control mesh* of a tensor-product B-Spline surface. By convention, the size of the control mesh is m in u-direction and n in v-direction. The surface $S(u,v)$ may be then written as a linear combination of its control points and basis functions for $\mathcal{P}_{k,\tau}$ and $\mathcal{P}_{l,\upsilon}$:

$$S(u,v) = \sum_{j=0}^{n-1} \sum_{i=0}^{m-1} S_{i,j} b_i^{k,\tau}(u) b_j^{l,\upsilon}(v)$$

Alternatively, we will use the compact notation with two-variate B-spline basis functions $b_I^{k,\tau,l,\nu}(u,v) = b_i^{k,\tau}(u) b_j^{l,\upsilon}(v)$ and control points S_I with integer indices I such that

$$I = i + mj,\ 0 \leq i < m \wedge 0 \leq j < n \Rightarrow 0 \leq I \leq (m-1)(n-1)$$

The surface $S(u,v)$ may be then written as a linear combination of its control points and two-variate basis functions:

$$S(u,v) = \sum_{I=0}^{(m-1)(n-1)} S_I b_I^{k,\tau,l,\upsilon}(u,v)$$

Since each $s_d(u,v)$ is fully determined by its coefficients a B-Spline surface is fully determined by the values of its control points. Analogously to a B-Spline curve we define a matrix of control points:

$$\mathbf{S} = \begin{bmatrix} S_0 \\ \vdots \\ S_{(m-1)(n-1)} \end{bmatrix}$$

which yields:

$$\begin{aligned} S(u,v) &= \begin{bmatrix} s_0(u,v) & s_1(u,v) & s_2(u,v) \end{bmatrix} \\ &= \mathbf{b}_{u,v}^T \begin{bmatrix} S_0 \\ \vdots \\ S_{(m-1)(n-1)} \end{bmatrix} \\ &= \mathbf{b}_{u,v}^T \mathbf{S} \end{aligned}$$

The coordinates of the control points are the degrees of freedom of a B-Spline surface:

$$dofs\,(S(u,v)) = \mathbf{S},\ \mathbf{S} \in R^{mn \times 3}$$

A.1.3.1 Iso-parametric curves of a B-Spline surface

Let $v = const.$ be a line in the domain space of a B-Spline surface. Set $v = \alpha$. Then a curve

$$E(u) = F(u, \alpha) = \sum_{j=0}^{n-1} \sum_{i=0}^{m-1} S_{i,j} b_i^{k,\tau}(u) b_j^{l,\upsilon}(\alpha)$$

is called an *iso-parametric* curve in u-direction of the B-Spline surface F. By setting $\beta_j = b_j^{l,\upsilon}(\alpha)$ it follows immediately that $E(u) = \sum_i E_i b_i^{k,\tau}(u)$ with $E_i = \sum_{j=0}^{n-1} S_{i,j}\beta_j$. Similarly, one can set $u = const.$ obtaining an iso-parametric curve in v-direction. Iso-parametric curves play an important role in the design and analysis of tensor-product surfaces, as will be described in §2.2.2.

A.2 Bezier basis as special case of B-Spline basis

We will introduce the, so-called, Bezier basis as a special case of the B-Spline basis: Let $\langle a, b\rangle \subseteq R^1$ be a given interval of R^1. As in §A.1.1 let also be given $d > 0$, $n = d + 1$ and a knot vector τ of size $n + d + 1 = 2d + 2$ such that

1. $\tau_0 = \cdots = \tau_d = a$
2. $\tau_{d+1} = \cdots = \tau_{2d+1} = b$

Set $a = 0$, $b = 1$. For this special case, formula A.1 degenerates into

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

Denote by $\mathcal{P}_d$ a vector space of univariate polynomials of degree $\leq d$. The set of $d + 1$ functions $\{b_i^d(t) : 0 \leq i \leq d\}$, called *Bernstein* polynomials constitutes a basis for $\mathcal{P}_d$, see e.g. [68]. Traditionally, the Bernstein polynomials are called Bezier basis functions. We will use the notation $b_i^d(t)$ (without the super index τ) for Bezier basis functions to denote the difference to general B-Splines. The domain space for Bezier basis functions is not restricted to $t \in \langle 0, 1\rangle$. Setting the knot vector as defined above yields a set of generalized Bernstein polynomials $\{b_i^d(t) : 0 \leq i \leq d,\ t \in \langle a, b\rangle\}$:

$$b_i^d(t) = \left(\frac{t-a}{b-a}\right) b_i^{d-1} + \left(\frac{b-t}{b-a}\right) b_{i+1}^{d-1} = \binom{d+1}{i} \left(\frac{b-t}{b-a}\right)^{d+1-i} \left(\frac{t-a}{b-a}\right)^i$$

Bezier functions, curves and surfaces are defined in exactly the same way as B-Splines in §A.1.1-§A.1.3, therefore we will not make further distinction in the notation. Whenever the type of used basis functions is of importance we will point that out in advance.