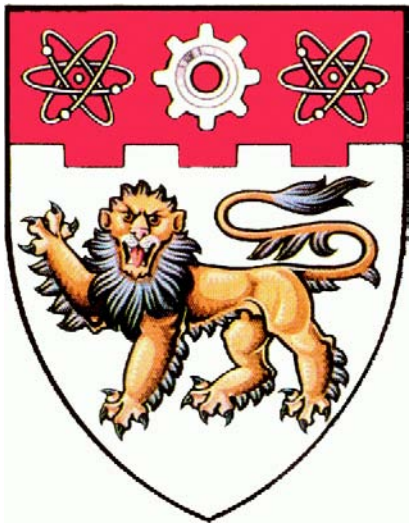
NANYANG TECHNOLOGICAL UNIVERSITY



FREE-FORM SURFACE REPRESENTATION AND APPROXIMATION USING T-SPLINES

BY

WANG YIMIN

A THESIS SUBMITTED TO THE NANYANG TECHNOLOGICAL UNIVERSITY IN FULFILMENT OF THE REQUIREMENT FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

SCHOOL OF COMPUTER ENGINEERING

2009

Acknowledgements

At the first place, I would like to express my most heartfelt gratitude to my supervisor Dr. Jianmin Zheng, for his constant guidance and unconditional help during the past few years of my Ph.D. study. Without his patient and insightful instructions, I can hardly survive this research program. In our numerous discussions, Dr. Zheng has graciously shared many invaluable experiences, which benefit me both technically and philosophically. For all these kindnesses I am forever thankful.

I want to thank Nanyang Technological University and the School of Computer Engineering for granting me this study opportunity. Besides, my appreciation goes to the gameLAB which provides me with the excellent working environment and the high performance equipments.

I feel extremely fortunate to work with many excellent current and former members in the Computational Arts Group. Especially, I would like to thank Dr. Kemao Qian and Dr. Zhongke Wu for their help and concern. I also wish to thank my friends Qi Liu, Chen Chen, Yu Yu, Xiaoyi Huang, Jie Qiu, Quan Chen, Xuexiang Xie, Xian Xiao, Hailing Zhou, Kai Wang and Yuewen Ma, with whom I have shared some great time in Singapore.

Finally, I would like to take this opportunity to express my profound gratefulness to my parents who taught me the value of hard work by their own experiences. Their longtime encouragement and support mean a lot to me.

Contents

List of Figures

Abstract

The problem of describing the surface information of a complex shaped object with a computer is often encountered in the fields of computer aided design and computer graphics. The surface of such an object could have arbitrary shape and is usually called the free-form surface. T-splines are a recently developed generalization of non-uniform rational B-splines (NURBS) technology enabling local refinement, which can overcome many drawbacks of NURBS and offer a more flexible representation for free-form surfaces. T-splines have a potential of becoming the next industry standard for free-form surfaces in high end animation and computer aided design industry. It is apparent that to permit this potential to be fully realized, there must be developed solid mathematical theory and a comprehensive set of supporting algorithms. The research of this thesis therefore aims to investigate some fundamental algorithms of T-splines and to explore their use in surface approximation.

First, a very fundamental problem of T-splines, which is to remove control points from a T-spline control grid while keeping the surface unchanged, is studied in the thesis. An algorithm is presented to detect whether a specified control point can be removed and to compute the new T-spline representation if the point is removable. The algorithm can be viewed as a reverse process of the T-spline local knot insertion algorithm. The complexity of removing more than one control point is also analyzed and the extensions of the algorithm to remove many points are discussed. Compared to the B-spline knot removal in which a whole row or column of control points needs to be removed, the presented T-spline control point removal algorithm usually causes only local change to the T-spline control grid.

Second, the use of T-splines in surface approximation is investigated. Triangular meshes and spline surfaces are currently two main representations for free-form surfaces. There is a need to convert triangular meshes into spline surfaces. However, such conversion is never a simple task especially when the shape is complicated. This thesis introduces a new framework for adaptive surface fitting which achieves the conversion from a triangular mesh that is topologically homeomorphic to a plane region to a spline surface. The new

framework improves the conventional least squares approximation method by incorporating several novel ideas and components such as the adopting of T-splines for fitting, the use of curvature to guide fitting, the re-placement of initial T-spline structure and a faithful re-parameterization. As a result, the reconstructed T-spline surface can well respect geometric features of the original input model and is thus more compact compared to that created by the traditional B-spline methods. Many examples are provided to demonstrate the effectiveness of the proposed framework and algorithm.

Third, to better handle tubular meshes that have the same topology as a cylinder, periodic T-splines are proposed and a convenient representation for periodic T-splines is presented. To parameterize a tubular triangular mesh, an edge based method is proposed, in which the edges rather than the vertices of the mesh are treated as the target for parameterization. This method improves conventional cutting-based algorithms, which cut the mesh to make it a disk topologically, and overcomes the problems of cutting paths that are the zigzag paths leading to suboptimal parameterizations and the difficulty in finding good cutting paths. After that, an adaptive surface fitting algorithm using periodic T-splines is developed, which can effectively approximate tubular triangular meshes.

Fourth, the conversion between T-splines and hierarchical NURBS is considered. Hierarchical B-splines is another generalization of NURBS with local refinement. They are especially suitable for multiresolution editing. In the thesis, the concept of hierarchical B-splines is extended to hierarchical NURBS and then two algorithms are constructed, by which a T-spline surface can be converted into a hierarchical NURBS surface, and vice versa. These algorithms take the structure of the original surface into consideration and yield compact representations. With these algorithms, the user can flatten a hierarchical NURBS surface to create a T-spline for interactive sculpturing or extract hierarchies from a T-spline surface for multiresolution analysis.

Chapter 1

Introduction

1.1 Problem statement and motivation

The task of representing and constructing smooth surfaces is ubiquitous in geometric modeling and computer graphics. However, how to efficiently describe, create and manipulate complicated free-form surfaces even with advanced geometric modeling systems is still a challenging work. T-splines [129, 128] are a recently developed surface modeling technology which is a generalization of non-uniform rational B-splines (NURBS) by permitting T-junction points in the control grid. T-splines permit truly local refinement—a very important feature required by many theoretical and practical problems. The T-spline representation is fully compatible with NURBS, making it easier to use in conjunction with existing commercial software systems such as ACIS modelers, Parasolid modelers, Rhino3D, and Autodesk Maya. The T-spline representation also allows designers to focus on the regions where more detail is called for and to model complicated shapes faster with fewer control points. All these features indicate that T-splines have a potential of becoming the next industry standard for free-form surfaces in high end animation and computer aided design (CAD) industry. It is clear that to permit this potential to be fully realized, both solid mathematical theory and a comprehensive set of supporting algorithms must be developed. Many

researches on the use of T-splines have been studied, examples of which include NURBS surface simplification [128], trimmed NURBS surfaces merging [130], free-form deformation [143] and 3D mesh morphing [160]. This thesis aims to investigate some fundamental problems of T-splines and explore their use in representing and approximating free-form surfaces. Specifically, the research focuses on the following three problems:

The first problem is T-spline knot removal. In NURBS theory, knot insertion and knot removal are two of the fundamental algorithms that can be used as mathematical tools for understanding and analyzing B-splines and also as practical tools for manipulating and rendering B-spline curves and surfaces [50]. Many applications can be developed based on them. For example, knot insertion provides tools for straightforward evaluation of points on the curves or surfaces, base conversion, and adding extra degrees of freedom for shape modification. Knot removal provides tools for data reduction, shape fairing, base conversion, and wavelet decomposition. There exists several algorithms for B-spline knot insertion and knot removal. For T-splines, one important feature is the permission of local refinement. A knot insertion algorithm was first proposed in [129] and later a more efficient algorithm was developed in [128]. Compared to B-spline knot insertion, T-spline knot insertion is much more sophisticated. As the inverse of knot insertion, T-spline knot removal, which is a process of removing a knot from a given T-spline surface, is not known yet. Therefore there is a need to develop T-spline knot removal algorithm to enrich T-spline theory. Just as T-spline knot insertion, it is expected that T-spline knot removal would be much more complicated than B-spline knot removal.

The second problem is surface approximation. Though advanced modeling systems have provided many tools to model highly detailed models, it is still difficult with these systems to directly create such surfaces as human faces and car-body panels. The advance of laser range scanners or other 3D data acquisition equipments offers a promising alternative means for capturing those surfaces which are generally difficult to create. Laser range scanners usually produce large collections of points or triangular meshes. While scattered data

or triangular meshes are an adequate representation for some applications such as visualization, animation and interactive computer games, splines are the major format in computer aided design and manufacture industry. Spline surfaces have parametric equations, which can be used to define differential structure for shape analysis, and have a more compact representation, facilitating the application of surface manipulation. Therefore there is a need to convert scattered data or triangular meshes into spline surfaces. The problem of converting scattered data or triangular meshes into spline surfaces is often referred to as *surface approximation*. By today's technology, this kind of approximation is time consuming and computationally expensive. Especially for surfaces with complicated geometry or topology, the conversion is very challenging. Since T-splines is a generalization of NURBS and allows truly local refinement, it is believed that the flexibility of T-splines would facilitate the surface approximation process.

The third problem is the conversion between T-splines and hierarchical B-splines. In CAD industry, NURBS are the dominant mathematical technique for modeling free-form surfaces, but NURBS surfaces do not support local knot insertion, which is important for locally editing and modifying the fine scale details of the surfaces. T-splines are one generalization of NURBS. Another well established method supporting local refinement is hierarchical B-splines [48] which consist of a collection of B-splines at different levels and enable local refinement of a free-form surface by representing it in a hierarchy. Hierarchical B-splines have become a powerful multiresolution representation. The main difference between hierarchical B-splines and T-splines is really between a hierarchy and T-junctions. With hierarchical B-splines, one could have a hierarchy which can be desirable for some applications, especially for multiresolution modeling. With T-splines, one could work on a single layer mesh, which contains substantially fewer points compared to the B-spline control mesh. Both hierarchical B-splines and T-splines have their own strength. Therefore it is useful in practice to be able to convert one representation into another. In particular, flattening a hierarchical B-spline surface into a T-spline surface can avoid forcing a hierar-

chy on the designer and provide a more direct interface, enabling the designer to focus on the regions where more detail is called for. On the other hand, extracting hierarchies from a T-spline surface can allow the user to perform multiresolution analysis.

1.2 Objectives and contributions

T-splines have shown to be a powerful and flexible free-form surface representation that offers unique computational advantages over NURBS. This research aims to investigate some fundamental issues of T-splines in representing and approximating free-form surfaces and to develop some efficient techniques to enrich T-spline technology.

The first objective of the research is to deliver some fundamental algorithms for T-splines. Since the local knot insertion algorithm plays a very important role in T-spline theory, we first examine the algorithm, aiming to gain insights. We expect that the insights may shed light on developing other fundamental algorithms. In particular, we expect to develop a T-spline knot removal algorithm and T-spline/hierarchical NURBS conversion algorithms in the same fashion of T-spline knot insertion. These algorithms should consider the local geometric characteristics of the surface and the tasks can be done locally, directly and quickly.

The second objective of the research is to present algorithms for converting triangular meshes into T-splines. This could be achieved by surface approximation. In particular, we focus on developing efficient techniques that fit a single T-spline surface to a triangular mesh that is homeomorphic to a plane region or that has the same topology as a cylinder. Solutions to the problem of fitting surfaces to an arbitrarily topological mesh are also proposed. Indeed, the research effort reported in this thesis initially aimed at the fitting problem for an arbitrarily topological model. However, it was soon realized that even the situation of simple topology had many open questions, such as how to make the algorithm well fit the geometric features, how to properly place initial T-spline surface structure, and

how to re-parameterize the mesh faithfully. The algorithms developed for the situation of simple topology will also serve as a base for designing algorithms for the arbitrary topology situation.

To achieve these objectives, thorough research has been conducted and several novel techniques have been proposed and developed. Briefly, contributions of the thesis include:

- the presentation of an algorithm to detect whether or not a specified control point can be removed and also to compute the new T-spline representation if the point is removable. Several possible extensions of the algorithm to remove more than one control point are given. These algorithms perform in the fashion of T-spline knot insertion and usually cause only local change to the T-spline control grid. It is also found that T-spline knot removal is much more complicated than B-spline knot removal.

- the introduction of a new framework for adaptive surface fitting which accomplishes the conversion from a triangular mesh to a spline surface. The new framework incorporates several new ideas and components into the conventional least squares approximation method. The new ideas or components include: 1) T-splines are used for adaptive fitting; 2) Geometric feature (i.e. curvature) is used to guide fitting to improve the fitting results; 3) An approach is given to set the placement of initial T-spline structure; and 4) A faithful re-parameterization is proposed. These components are efficiently integrated, forming a curvature guided adaptive T-spline surface fitting algorithm which achieves high performance for surface fitting.

- the introduction of the concept of periodic T-splines and the proposal of a convenient representation for them, in order to better handle tubular models that have the same topology as a cylinder. A geometrically intrinsic method is proposed to parameterize a tubular triangular mesh. The method improves conventional cutting-based algorithms and overcomes the problems of cutting paths that are the zigzag paths leading to suboptimal parameterizations and the difficulty in finding good cutting paths. Also,

an adaptive surface fitting algorithm using periodic T-splines is developed, which can effectively approximate tubular triangular meshes.

- the extension of the concept of hierarchical B-splines to hierarchical NURBS and the construction of two algorithms, by which a T-spline surface can be converted into a hierarchical NURBS surface, and vice versa. The geometric characteristics of the surfaces are considered in these algorithms and they generally yield compact representations.

1.3 Thesis organization

The rest of the thesis is organized as follows:

- Chapter 2 reviews related background and the state-of-the-arts of freeform surface representation and approximation with emphasis on adaptive surface approximation.

- Chapter 3 first re-examines T-spline local knot insertion algorithm and then presents an algorithms for T-spline knot removal. The complexity of removing more than one control point from a T-spline surface is analyzed and possible extensions of the algorithm to remove more than one control point are suggested.

- Chapter 4 proposes to use T-splines for surface fitting. A new framework is introduced to perform adaptive T-spline surface fitting that achieves the conversion from a triangular mesh to a spline surface. The framework includes several novel contributions such as curvature-guided adaptive T-spline fitting, initial T-mesh re-placement, and faithful re-parameterization.

- Chapter 5 first gives the definition and representation of periodic T-splines. Then an edge-based parameterization algorithm is presented. Finally an adaptive algorithm is proposed to construct periodic T-splines for tubular triangular meshes.

1.3. Thesis organization

- Chapter 6 extends the concept of hierarchical B-spline surfaces to hierarchical NURBS surfaces and presents algorithms for the conversion between a T-spline surface and a hierarchical NURBS surface.
- Chapter 7 concludes the thesis and proposes potential future work of this research.

Chapter 2

Background and Prior Work

2.1 Overview

This chapter provides some backgrounds for surface representations and reviews the state-of-the-art methods for surface approximation.

The problem of describing the surface information of an object within the computer is often encountered in the fields of computer aided design (CAD) and computer graphics (CG). Surfaces that have an arbitrary shape are usually called free-form surfaces. Free-form surfaces have vast applications in different areas, including computer games, computer graphic animations, special effects in movies, virtual reality, CAD/CAM, engineering simulation, scientific visualization, medical visualization, geographic information system (GIS) and digital art, etc. It is important to have a suitable representation for freeform surfaces in a particular application.

There have existed several major representations for free-form surfaces [77], such as polygonal meshes, parametric surfaces [39], subdivision surfaces [18, 35], point based surfaces [4, 60] and implicit surfaces [12]. Each of these surface representations has its own strengths and weaknesses. Basically, a polygonal mesh is defined by a collection of polygon faces with shared edges and vertices. When all the faces in a polygon mesh

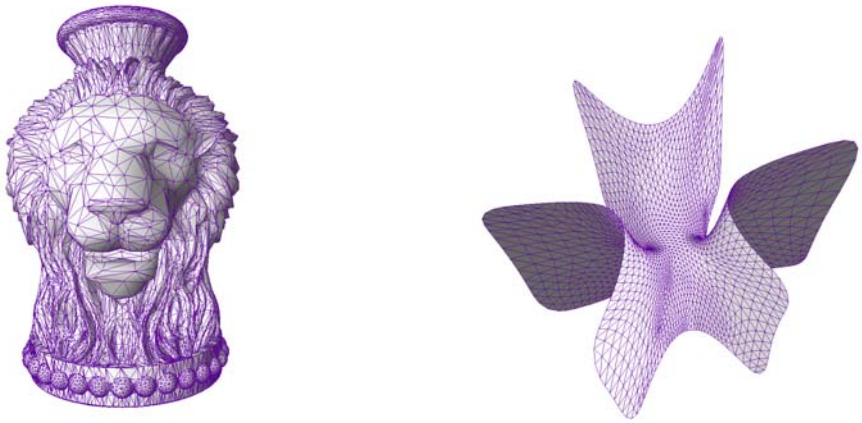
are triangles, the mesh is called a triangular mesh. A parametric surface is defined by a mapping from R^2 to R^3: each point $P = (x, y, z)$ on the parametric surface is written as $(x, y, z) = (f_1(u, v), f_2(u, v), f_3(u, v))$, where (u, v) are the two parameters of the point. A subdivision surface is defined by a coarse control mesh and a set of subdividing rules. The rules are used to recursively refine the mesh and finally a smooth surface is obtained. Different rules give subdivision schemes with different properties [168]. A point based surface is represented by a set of discrete points and the surface is defined by local approximations of these points. An implicit surface is defined by the iso-surface (usually the zero set surface) of an implicit function $f(x, y, z)$, or in another word, $f(x, y, z) = c$ for a constant c. It is easy to determine whether a point is inside , outside or on an implicit surface by its function value.

This thesis focuses on a special parametric surface representation, called T-splines, and the conversion from other representations such as triangular meshes to T-splines. Therefore, in the rest of this chapter, we first discuss triangular meshes in Section 2.2. Next, in Section 2.3 we explain non-uniform rational B-splines (NURBS), which is important for understanding T-splines. Then, T-splines are reviewed in Section 2.4. Finally, a survey on the related work of surface approximation is given in Section 2.5.

2.2 Triangular meshes

A triangular mesh consists of vertices, edges and faces. Each vertex is a point in the R^3 space. Each edge is a line segment with non-zero length that is bounded by two distinct vertices. The two vertices connected by an edge are called neighbors. Each face is a triangle comprising three distinct vertices and three edges connecting any two of them. The vertices specify the geometric information (i.e., locations) of the triangular mesh, while the edges and faces specify the topological information (i.e., how the vertices are connected). A triangular mesh can be closed or have boundaries. An edge is a boundary edge if it belongs

to only one face. The end points of a boundary edge are boundary vertices. Figure 2.1 shows a closed triangular mesh on the left and an open triangular mesh with boundary on the right.

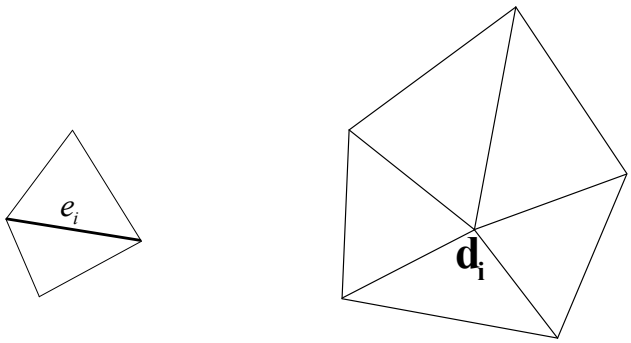


(a) A closed triangular mesh

(b) An open triangular mesh with boundary

Figure 2.1: Examples of triangular meshes.

In a manifold triangular mesh, each edge is shared by two adjacent faces (see Figure 2.2(a)), except for any boundary edge that resides only in one face. Each vertex can be shared by several faces and edges (see Figure 2.2(b)). The faces that share the same vertex should form a closed loop, or a single fan for a vertex on the boundary.



(a) (b)

Figure 2.2: Edge and vertex sharing.

A triangular mesh may have some irregularities. For example, irregularities occur if a mesh includes a degenerated face whose three vertices are collinear (see Figure 2.3(a)), a

self intersecting part (see the highlighted face in Figure 2.3(b)), an isolated edge or vertex (see edge e_i in Figure 2.3(c)), or a non-manifold structure where an edge is shared by more than two faces (see edge e_i in Figure 2.3(d)) or a loop is broken around a vertex (see vertex $\mathbf{d_i}$ in Figure 2.3(e)). These irregular structures often compromise the integrity of triangular meshes, complicate the problem and bring unnecessary troubles. In this thesis we assume that the triangular meshes we discuss are regular.

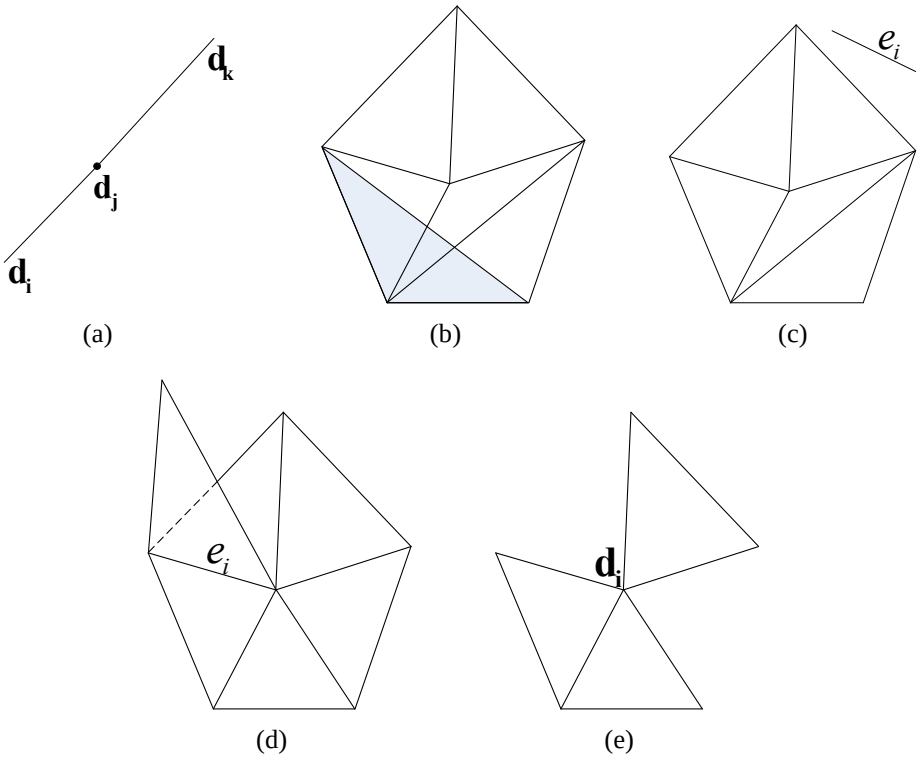


Figure 2.3: The irregularities in a triangular mesh.

Triangular meshes are now one of the most frequently used surface representations. They are flexible to represent a very complicated shape and are supported by graphics processing units. Due to the advances in the 3D laser scanners [11], triangular meshes can be conveniently obtained for real world objects. There are excellent online model repositories such as the AIM@SHAPE shape repository [1] where a large number of models are provided. Because of their popularity, researches have been actively carried out regarding various applications based on triangular meshes in recent years [16].

However, triangular meshes have certain drawbacks. Quite often, hundreds of thousands of triangles are required for a model in order to achieve a relatively accurate representation, making it expensive in both data storage and transmission. Another problem is that a surface represented by a triangular mesh is piecewise linear, giving only C^0 continuity among the triangle faces. Thus it is inconvenient for a triangular mesh to represent a smooth shape and to achieve high degrees of continuity, which are rather desirable in some industrial applications where high precision is required.

There are several different approaches for storing and encoding a triangular mesh with a computer. Among them, the simplest approach is the indexed-face list, in which the coordinates of all the vertices are listed in a table and all the faces are listed in another table. In the indexed-face list, the edge information is not explicitly included. Alternatively, there are more advanced data structures to represent a triangular mesh, such as the winged-edge [7], quad-edge [62] and widely adopted half-edge [155] data structures. In these data structures, the edge information is explicitly maintained and therefore they are referred to as edge based representations. A comparison for these edge based representations is available in [85]. There are some open source Libraries in C++ that provide robust implementations for the half-edge data structure such as CGAL [3] and OpenMesh [15], upon which the user can build his/her own mesh processing algorithms. The choice of the data structure really depends on the requirements of the application. For example, the indexed-face list is efficient for the rendering of triangular meshes. If certain routines such as accessing the neighboring edges or faces are frequently invoked, complex structures such as the half-edge data structure would be preferred.

2.3 NURBS: Non-uniform rational B-splines

The term “spline” initially refers to a wooden batten which is used to draw smooth curves. During the designing process, a spline can be bent at will and hold in place by several lead

weights called ducks.

Adapting such concepts, Schoenberg first studied the mathematic theory of B-splines [127] in 1940s and Riesenfeld introduced B-splines to geometric design community [120] in early 1970s. Later, Versprille discussed the rational B-splines for geometric modeling [152]. Since then, researches in B-splines have been actively carried out and by the 1990s, NURBS had already become an industry standard for free-form curves and surfaces.

Unlike triangular meshes, NURBS are piecewise polynomials or rational functions that offer great flexibility and precision for modeling applications and define truly smooth shapes. Another strength of NURBS is that there exist many efficient and numerically stable algorithms and elegant theory for NURBS. For example, the de Boor algorithm can be used for evaluating the NURBS at any parameter value. Polar forms or blossoming can be used to explain or derive many operations of NURBS [117]. All these properties make NURBS popular in the design and manufacturing industries where mathematically precise equations are required. Thus NURBS are included in many industry standards, such as IGES [80] and STEP [2]. Also, NURBS tools are incorporated into various commercial 3D modeling softwares such as Maya and Rhino3D.

In the following we briefly review B-splines and NURBS with emphasis on the features that are related to the research of this thesis.

2.3.1 B-spline basis functions

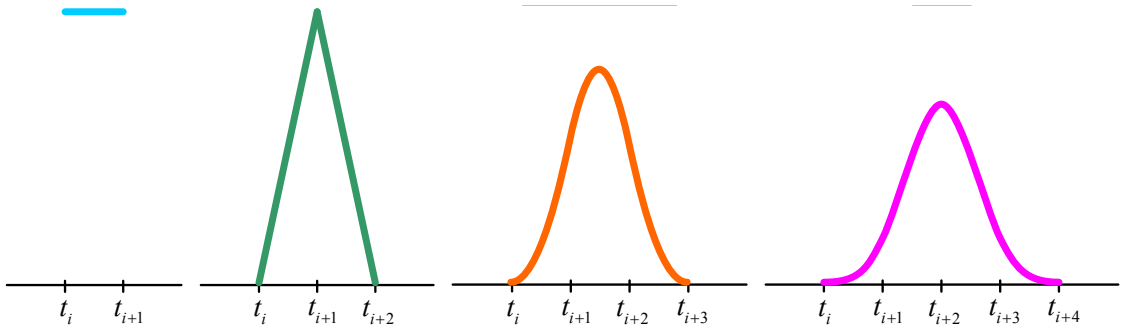
B-spline basis functions are important in B-spline theory and they are constructed from a knot vector. A knot vector is a non-decreasing sequence of coordinates in the parameter space, where each coordinate is called a *knot* and the interval bounded by two adjacent coordinates is called a *knot span*. To define B-spline basis functions of degree p, a knot vector $T = \{t_1, t_2, \cdots, t_{n+p+1}\}$ is required. Denoted by $N_i^k(t)$ the i-th B-spline basis function of degree k, which can be defined using the Cox-de Boor recursive formula [25,

29]:

$$N_i^0(t) = \begin{cases} 1 & \text{if } t \in [t_i, t_{i+1}) \\ 0 & \text{otherwise} \end{cases} \tag{2.1}$$

$$N_i^k(t) = \frac{t - t_i}{t_{i+p} - t_i} N_i^{k-1}(t) + \frac{t_{i+p+1} - t}{t_{i+p+1} - t_{i+1}} N_{i+1}^{k-1}(t), \quad k = 1, \ldots, p$$

In Figure 2.4, we show some B-spline basis functions from degree 0 to degree 3.



(a) Degree 0 (b) Degree 1 (c) Degree 2 (d) Degree 3

Figure 2.4: Some B-spline basis functions.

From the Cox-de Boor recursive formula, it is easy to derive some important properties of the B-spline basis functions:

- **Partition of unity**: $\sum_{i=1}^{n} N_i^p(t) \equiv 1$.
- **Positivity**: $N_i^p(t) > 0$ if $t \in (t_i, t_{i+p+1})$.
- **Compact support**: $N_i^p(t) = 0$ if $t \notin (t_i, t_{i+p+1})$.
- **Continuity**: $N_i^p(t)$ is a piecewise polynomial of degree p and is C^{p-1} continuous. (provided that $\{t_i, t_{i+1}, \cdots, t_{i+p+1}\}$ is strictly increasing.)

We can see that the i-th B-spline basis function $N_i^p(t)$ just depends on knots $t_i, t_{i+1}, \cdots, t_{i+p+1}$. Thus to emphasize this dependence, the B-spline basis function $N_i^p(t)$ is also denoted by $N^p[t_i, t_{i+1}, \cdots, t_{i+p+1}](t)$ in the thesis.

2.3.2 NURBS curves

Here we begin with a discussion on B-spline curves and then extend it to NURBS curves. Though any degree B-spline curves can be defined, we restrict our discussion to cubic B-splines or NURBS for simplicity.

Given n points $\{P_i\}_{i=1}^{i\leq n}$ and a knot vector $T = \{t_1, t_2, \cdots, t_{n+4}\}$, a cubic B-spline curve is defined as follows:

$$\mathbf{C}(t) = \sum_{i=1}^{n} P_i N_i^3(t), \quad t \in [t_4, t_{n+1}] \tag{2.2}$$

where $P_i = (x_i, y_i, z_i)$ are called the *control points*. The line segments connecting each two adjacent control points form the *control polygon*. According to Equation (2.2), each control point P_i is associated with a B-spline basis function N_i^3. It is also understood that each P_i corresponds to knot t_{i+2} and each line segment P_iP_{i+1} (except the first and last ones) in the control polygon corresponds to a segment of $\mathbf{C}(t)$ in $[t_{i+2}, t_{i+3}]$.

When all the knot spans are of the same length, the knot vector T is called a uniform knot vector and the B-spline curve is called a *uniform B-spline curve*. One example of a uniform cubic B-spline curve defined by four control points, together with its control polygon, is given in Figure 2.5(a). This curve has only one segment. The knot vector and the associated basis functions of the control points are illustrated in Figure 2.5(b). The parameter range for the curve is $[t_4, t_5]$ as highlighted in green color, where all the four basis functions have influence. When T is not a uniform knot vector, a B-spline curve defined over it is called a *non-uniform B-spline curve*. For example, if we have multiple knots: $t_0 = t_1 = t_2 = t_3$ and $t_{n+1} = t_{n+2} = t_{n+3} = t_{n+4}$ in T, T is a non-uniform knot vector. See Figure 2.6 for a non-uniform cubic B-spline curve defined over such a knot vector T. Each curve segment and its corresponding line segment are shown in an different color. Note that the B-spline curve interpolates the two end points due to the triple knots in T.

B-spline curves have many important properties which are appreciated in free-form

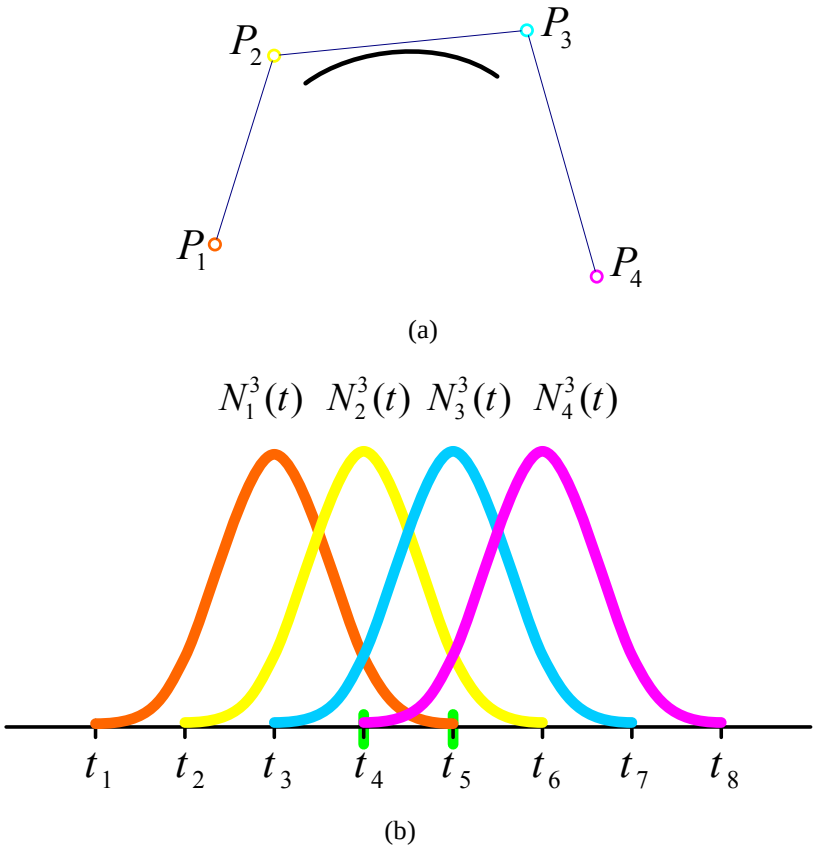


Figure 2.5: A uniform B-spline curve and its basis functions.

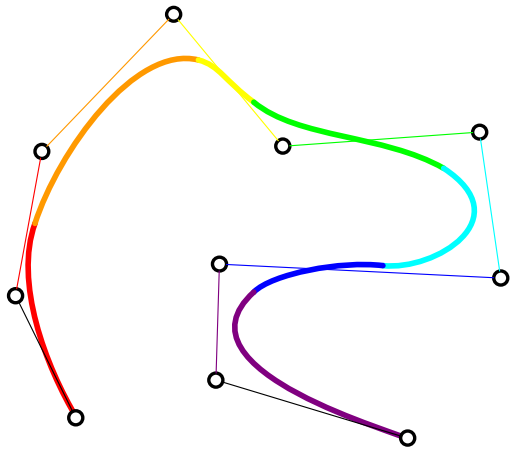


Figure 2.6: A non-uniform B-spline curve.

surface modeling and design. For example,

- **Local modification**: Moving one control point P_i only affects the curve segments in knot span $[t_i, t_{i+4}]$.
- **Convex hull**: The curve segment in knot span $[t_i, t_{i+1}]$ lies within the convex hull of control points $P_{i-3}, P_{i-2}, P_{i-1}, P_i$.
- **Affine invariance**: Applying an affine transformation to a B-spline curve is equivalent to applying the same affine transformation to the control points.
- **Continuity**: The cubic B-spline curve is C^2 continuous provided that T is strictly increasing.

However, there are geometric entities that cannot be modeled exactly by piecewise polynomials. Circles and hyperbolas are examples. This motivated the introduction of rational curves. More generally, B-splines were extended to the non-uniform rational B-splines (NURBS) [110, 122]. NURBS have better capability in shape representation and design than polynomial B-splines. For example, we can use a NURBS curve to precisely define a conic section. A cubic NURBS curve has the following equation:

$$\mathbf{R}(t) = \frac{\sum_{i=1}^{n} w_i P_i N_i^3(t)}{\sum_{i=1}^{n} w_i N_i^3(t)}, \quad t \in [t_4, t_{n+1}] \tag{2.3}$$

where w_i is the control point weight and it controls the impact of P_i on the shape of the curve. When all the w_i take the same value, $\mathbf{R}(t)$ degenerates to a polynomial B-spline curve. Let $Q_i = (w_i x_i, w_i y_i, w_i z_i, w_i)$ be the homogeneous coordinates for control points P_i, the NURBS curve of (2.3) can be concisely written in the homogeneous space:

$$\mathbf{R}^H(t) = \sum_{i=1}^{n} Q_i N_i^3(t) \tag{2.4}$$

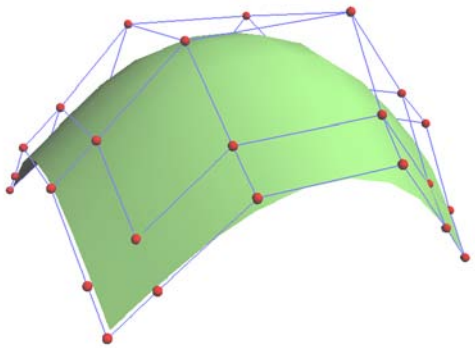


Figure 2.7: A NURBS Surface.

which is actually a polynomial B-spline curve in 4D space.

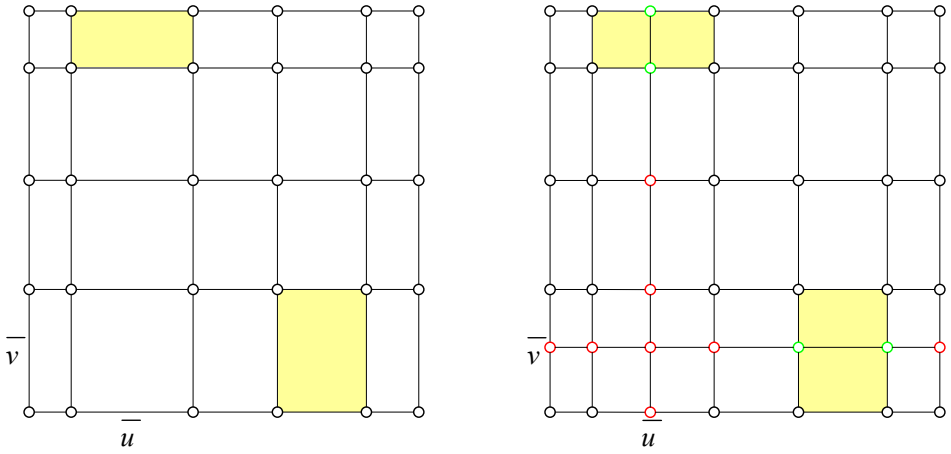
B-spline curves or NURBS curves allow an operation of knot insertion. The basic concept of knot insertion for a NURBS curve is to add one or more knots into the current knot vector T, while keeping the curve unchanged. The increase of knots results in the increase of control points and thus provides more degrees of freedom. In [14], Boehm presented an algorithm for inserting the knots into T sequentially, one knot at a time. The new control points are computed by the linear combination of those old control points. Knot insertion can also be achieved by Oslo algorithm [23] which allows to insert more than one knot into T simultaneously.

2.3.3 NURBS surfaces

The concept and construction of NURBS curves can be extended to NURBS surfaces. Given a set of control points $\{P_{ij}\}_{i=1,j=1}^{i\leq m,j\leq n}$ arranged in m rows and n columns topologically, and knot vectors $U = \{u_1, u_2, \cdots, u_{m+4}\}$, $V = \{v_1, v_2, \cdots, v_{n+4}\}$, a bicubic NURBS surface is defined by

$$\mathbf{S}(u,v) = \frac{\sum_{i=1}^{m}\sum_{j=1}^{n} w_{ij}P_{ij}N_i^3(u)N_j^3(v)}{\sum_{i=1}^{m}\sum_{j=1}^{n} w_{ij}N_i^3(u)N_j^3(v)}, \quad (u,v) \in \Omega = [u_4, u_{m+1}] \times [v_4, v_{n+1}] \tag{2.5}$$

where w_{ij} is the weight for control point P_{ij}, $N_i^3(u)$ and $N_j^3(v)$ are the cubic B-spline basis functions in parameters u and v defined over U and V, respectively. Ω is the parameter domain. The $m \times n$ control points are connected to form a *control grid*. Since the basis functions $N_i^3(u)N_j^3(v)$ are the product of two univariate B-spline bases, sometimes NURBS surfaces are also called tensor-product surfaces. Figure 2.7 shows a NURBS surface with a control grid of 6×6 control points.



(a) Before insertion (b) After insertion

Figure 2.8: Control point insertion for a NURBS surface.

NURBS surfaces have analogous properties and operations to NURBS curves. However, NURBS surfaces do not support local refinement which NURBS curves do. Refer to Figure 2.8 for an example, where the control grid layout of a NURBS surface in the parameter domain is shown. Suppose we want to increase the control ability of the surface in the highlighted areas in Figure 2.8(a). We add two new knots $\bar{u}$ and $\bar{v}$ to the respective knot vectors. This results in a refined control grid shown in Figure 2.8(b), where an entire row and an entire column of control points are added. Note that among the newly added control points, only the ones in green color that reside inside of the highlighted areas are truly required and those control points in red color are not really needed from an application's perspective and they are added just due to the need to maintain the tensor product structure of the NURBS surface control grid. Usually, the more complicated the surface is, the more

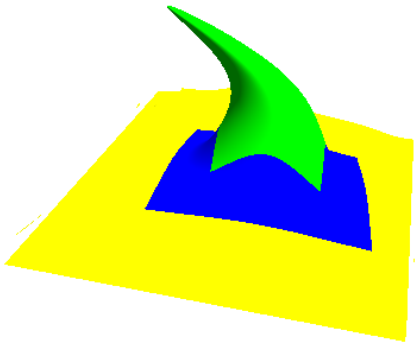


Figure 2.9: A hierarchical B-spline surface.

unnecessary control points are introduced during the knot insertion. This is quite annoying for applications. Besides this, NURBS surfaces have deficiencies that gaps and overlaps at intersections of surfaces cannot be avoided, which complicates mesh generation, and it is difficult to represent most shapes using a single, watertight NURBS surface.

2.3.4 Hierarchical B-spline surfaces

As shown above, for a tensor-product B-spline (or NURBS) surface, knot insertion is not a local process. This is because when a knot is inserted into a B-spline surface, new control points must be added row-wise or column wise. To overcome this drawback, Forsey and Bartels introduced the concept of hierarchical B-splines [48]. A hierarchical B-spline surface represents the surface in a fashion of hierarchy and it is comprised of a series of levels, each of which has a collection of B-spline patches. Figure 2.9 shows an example of a hierarchical B-spline surface. It has 3 levels and uses only 37 control points. The hierarchy provides the capability for local refinement of surfaces and multiresolution surface editing. Hierarchical B-splines have been successfully used in Forsey's interactive modeller called "Dragon" and facial animation. There is a gallery containing a number of models and animations created by hierarchical B-splines [144]. A modeler based on hierarchical B-spline

surfaces [68] is implemented and the result can be outputted into POV-Ray for rendering. Hierarchical B-splines have also been integrated into some modeling environments such as Softimage 3D [142, 36].

2.4 T-splines

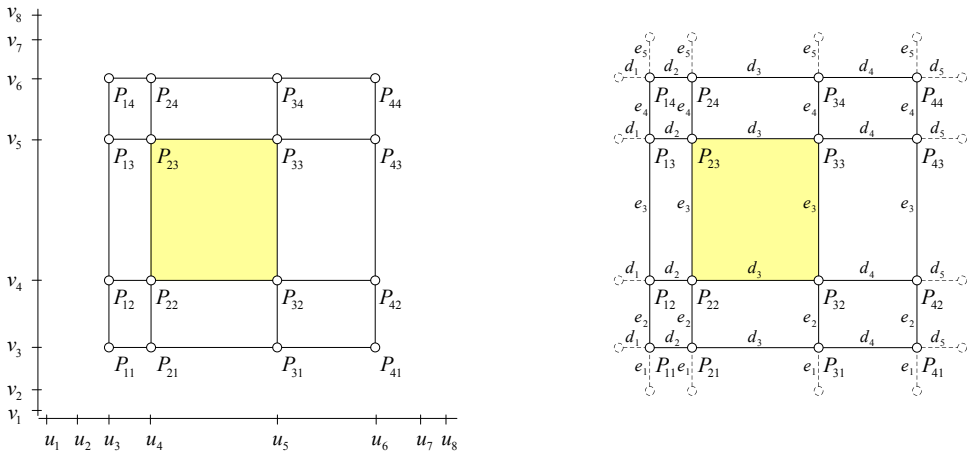
T-splines are a recently developed freeform surface technology [129, 128], which generalizes NURBS to a more flexible control grid and corrects the deficiencies of NURBS. While T-splines inherit most of the properties of NURBS, one important feature of T-splines is that they support local refinement, which means adding a new control point into the T-mesh usually would not cause the insertion of too many extra points. As a result, T-splines provide an more intuitive and natural way for users to design and edit the shape of a surface by focusing on the areas where detailed are really needed. By using the T-spline representation, the number of control points needed for defining a surface is substantially reduced compared to that of an equivalent NURBS surface. This makes the T-spline representation more compact. In addition, a T-spline surface can be losslessly converted into a NURBS surface, which makes T-splines fully compatible with the current CAD/CAM industry. In the following some basic concepts of T-splines are reviewed. The review is restricted to cubic T-splines although T-splines can be extended to any degree [40].

2.4.1 Knot vectors and knot intervals

In the definition of a NURBS surface, the knot information is given by two knot vectors in the $u-$ and $v-$directions. It can be understood that each control point is associated with a pair of knot values. The pair of knot values comes from those two knot vectors. For example, in Figure 2.10(a) where the pre-image of the 4×4 control grid of a B-spline patch in the parameter domain is shown, the control point P_{32} has a knot pair (u_5, v_4).

For a T-spline surface, the structure of the control grid is much flexible. So it would be

more convenient and also more intuitive to use knot intervals to convey the knot information of the surface. A knot interval is a non-negative number assigned to each edge of the T-mesh. The notion of knot intervals was initially introduced in [131] and expanded in [132, 129]. For a cubic B-spline curve, the knot interval is simply the difference of two consecutive knots in a knot vector and thus a parameter length of one of the curve segments that comprise the B-spline curve. Thus the knot interval of a cubic B-spline curve segment is attached as a label to its corresponding control polygon edge. In this way, knot intervals provide a method of conveying knot information for a B-spline curve and knot interval representation is independent of the knot origin in the parameter space.



(a) Knot vectors representation (b) Knot interval representation

Figure 2.10: Knot vectors and knot intervals.

The relationship between knot vectors and knot intervals for a B-spline surface is illustrated in Figure 2.10. In Figure 2.10(a), the knot information is given by knot vectors and in Figure 2.10(b), the knot information is given by knot intervals. The knot interval d_i is for an edge that maps to a horizontal edge in the pre-image of the control grid and the knot interval e_i is for an edge that maps to a vertical edge in the pre-image. They can be

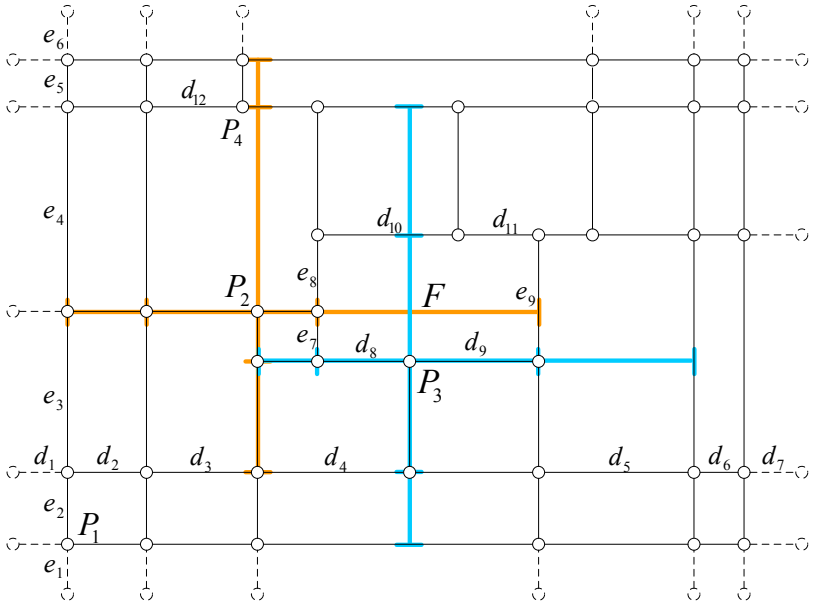


Figure 2.11: The pre-image of a T-mesh (in black color).

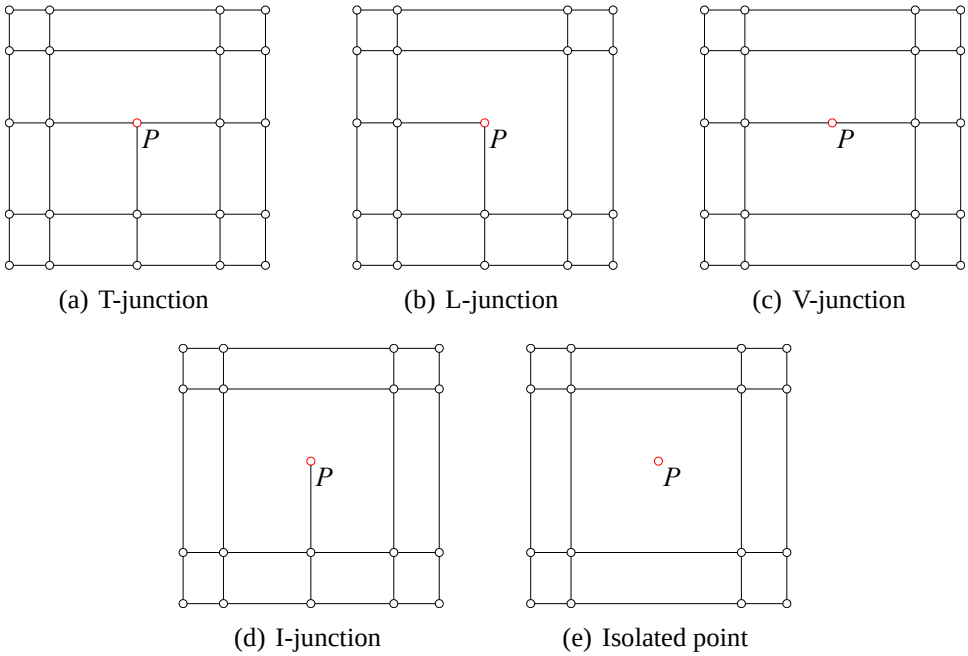
computed from the knot vectors:

$$\begin{aligned} d_i &= u_{i+2} - u_{i+1} \\ e_i &= v_{i+2} - v_{i+1} \end{aligned} \tag{2.6}$$

On the other hand, if the knot intervals are given and a knot origin is chosen, we can also infer knot vectors from (2.6).

2.4.2 T-spline surface equation

A T-spline surface is defined by a control grid called T-mesh. The T-mesh is similar to a NURBS control grid except that in a T-mesh a partial row or column of control points is permitted. The permission of existence of partial rows or columns makes it possible to add a single control point to a T-mesh without propagating an entire row or column of control points and without altering the surface. The final control point(s) in a partial row or column are called *T-junctions*. For example, P_2 and P_3 in the pre-image of a T-mesh shown in Figure 2.11 are T-junctions.



(a) T-junction (b) L-junction (c) V-junction

(d) I-junction (e) Isolated point

Figure 2.12: Different types of T-junctions.

A T-mesh may contain different types of T-junctions which can be distinguished according to their connectivity: rigorous T-junctions which are connected to exactly three edges (see Figure 2.12(a)), L-junctions which are connected to one horizontal edge and one vertical edge (see Figure 2.12(b)), V-junction which are connected by two horizontal edges or two vertical edges (see Figure 2.12(c)), I-junctions which are only connected to one edge (see Figure 2.12(d)), and isolated points which are not connected to any edge (see Figure 2.12(e)). If a T-mesh does not contain T-junctions, it is a tensor product mesh and the T-spline surface degenerates to a NURBS surface.

The knot information of a T-spline is expressed using knot intervals indicating the difference between two knots and assigned to the edges of the T-mesh. The assignment of the knot intervals to a T-mesh edge is subject to some constraints:

Rule 1: *The sum of the knot intervals on opposing edges of any face must be equal.*

Rule 2: *If two T-junction on opposing edges of a face can be connected without violating the previous rule, that edge must be included in the T-mesh.*

For example, in face F of Figure 2.11, $d_8 + d_9 = d_{10} + d_{11}$ and $e_7 + e_8 = e_9$ according to Rule 1. Also, according to Rule 2, there should be an edge connecting P_2 and P_4 if $d_3 = d_{12}$.

In order to derive an analytic equation for T-splines, a knot coordinates system should be imposed and then the knot coordinates are inferred from the knot intervals. First, an arbitrary point in the pre-image is designated to be the knot origin and is given coordinates of $(0, 0)$. Then, we assign an u knot value to each vertical edge in the pre-image of the T-mesh and a v knot value to each horizontal edge using the knot interval information. Finally, each control point gets a pair of knot coordinates from the knot values of the vertical and horizontal edges it lies on. For example in Figure 2.11, if we let P_1 be $(0, 0)$, then the coordinates for P_2 would be $(d_2 + d_3, e_2 + e_3)$ and the coordinates for P_3 would be $(d_2 + d_3 + d_4, e_2 + e_3 - e_7)$.

Based on the T-mesh and its knot information, the equation for a T-spline surface is:

$$\mathbf{S}(u, v) = \frac{\sum_{i=1}^{n} w_i P_i B_i(u, v)}{\sum_{i=1}^{n} w_i B_i(u, v)} \tag{2.7}$$

where the P_i are control points and the w_i are control point weights. The T-spline blending function corresponding to control point P_i is $B_i(u, v)$:

$$B_i(u, v) = N^3[\mathbf{u}_i](u) N^3[\mathbf{v}_i](v) \tag{2.8}$$

where $N^3[\mathbf{u}_i](u), N^3[\mathbf{v}_i](v)$ are the cubic B-spline basis functions associated with the local knot quintuples $\mathbf{u}_i = [u_{i0}, u_{i1}, u_{i2}, u_{i3}, u_{i4}]$ and $\mathbf{v}_i = [v_{i0}, v_{i1}, v_{i2}, v_{i3}, v_{i4}]$ respectively. For

example,

$$N^3[\mathbf{u}_i](u) = \begin{cases} \dfrac{(u-u_{i0})^3}{(u_{i1}-u_{i0})(u_{i3}-u_{i0})(u_{i2}-u_{i0})}, & u_{i0} < u \le u_{i1} \\ \dfrac{(u-u_{i0})^2(u_{i2}-u)}{(u_{i2}-u_{i1})(u_{i3}-u_{i0})(u_{i2}-u_{i0})} + \dfrac{(u_{i3}-u)(u-u_{i0})(u-u_{i1})}{(u_{i2}-u_{i1})(u_{i3}-u_{i1})(u_{i3}-u_{i0})} \\ + \dfrac{(u_{i4}-u)(u-u_{i1})^2}{(u_{i2}-u_{i1})(u_{i4}-u_{i1})(u_{i3}-u_{i1})}, & u_{i1} < u \le u_{i2} \\ \dfrac{(u-u_{i0})(u_{i3}-u)^2}{(u_{i3}-u_{i2})(u_{i3}-u_{i1})(u_{i3}-u_{i0})} + \dfrac{(u_{i4}-u)(u_{i3}-u)(u-u_{i1})}{(u_{i3}-u_{i2})(u_{i4}-u_{i1})(u_{i3}-u_{i1})} \\ + \dfrac{(u_{i4}-u)^2(u-u_{i2})}{(u_{i3}-u_{i2})(u_{i4}-u_{i2})(u_{i4}-u_{i1})}, & u_{i2} < u \le u_{i3} \\ \dfrac{(u_{i4}-u)^3}{(u_{i4}-u_{i3})(u_{i4}-u_{i2})(u_{i4}-u_{i1})}, & u_{i3} < u \le u_{i4} \\ 0, & u \le u_{i0} \text{ or } u > u_{i4} \end{cases}$$

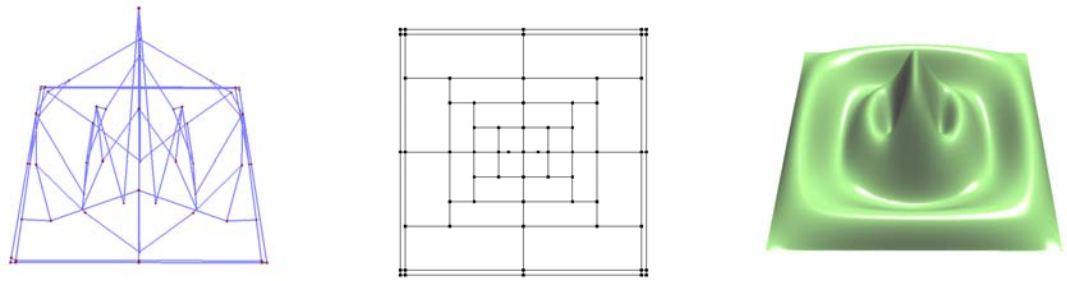
The knot quintuples $\mathbf{u}_i$ and $\mathbf{v}_i$ that determine the blending function $B_i(u,v)$ are extracted from the T-mesh neighborhood of P_i according to the following rule:

Rule 3: *Assume (u_{i2}, v_{i2}) is the knot coordinates of P_i. By casting a ray $R(t) = (u_{i2} + t, v_{i2}),\ t > 0$, u_{i3}, u_{i4} are defined as the knot values of the first two vertical edges or control points that intersect with $R(t)$. Other knots in $\mathbf{u}_i$ and $\mathbf{v}_i$ are found in a similar manner.*

For example, in Figure 2.11, the knot quintuples for $B_2(u,v)$ are $\mathbf{u}_2 = [0, d_2, d_2+d_3, d_2+d_3+d_4-d_8, d_2+d_3+d_4+d_9]$ and $\mathbf{v}_2 = [e_2, e_2+e_3-e_7, e_2+e_3, e_2+e_3+e_4, e_2+e_3+e_4+e_5]$, as visualized using orange color. Similarly, the knot quintuples for $B_3(u,v)$ are $\mathbf{u}_3 = [d_2+d_3, d_2+d_3+d_4-d_8, d_2+d_3+d_4, d_2+d_3+d_4+d_9, d_2+d_3+d_4+d_9+d_5]$ and $\mathbf{v}_3 = [0, e_2, e_2+e_3-e_7, e_2+e_3+e_8, e_2+e_3+e_4]$, as visualized using blue color.

The T-spline equation can also be written in homogeneous form:

$$\mathbf{S^H}(u,v) = \sum_{i=1}^{n} Q_i B_i(u,v) \tag{2.9}$$



(a) (b) (c)

Figure 2.13: An example of a T-spline: the T-mesh, the pre-image, and the surface.

where the $Q_i = (w_i P_i, w_i) = (w_i x_i, w_i y_i, w_i z_i, w_i)$ are homogeneous control points in the 4D space. Figure 2.13 shows an example of a T-spline surface: The left figure shows the T-mesh; the middle one shows the pre-image of the T-mesh in the parameter domain; and the right one is the T-spline surface.

2.4.3 T-spline classification

A T-spline surface would generally be rational except for a few special cases. A standard T-spline surface is one for which all weights $w_i = 1$, then $\sum_{i=1}^{n} w_i B_i(u, v) \equiv 1$. A semi-standard T-spline is one for which $\sum_{i=1}^{n} w_i B_i(u, v) \equiv 1$ and not all $w_i = 1$. A non-standard T-spline is one for which the condition $\sum_{i=1}^{n} w_i B_i(u, v) \equiv 1$ never holds. Both standard and semi-standard T-splines define piecewise polynomial surfaces and non-standard T-splines are rational. An important property of T-splines is that if we perform a local refinement on a standard or semi-standard T-spline surface, the result will always be a standard or semi-standard T-spline surface.

2.5 Surface approximation

Surface approximation, also known as surface fitting, has been widely studied and has applications in a number of fields. This term has different interpretations and here it is regarded

as the process of converting the discrete data such as point clouds or triangular meshes into some smooth surface representations. The problem of surface approximation is usually more complicated, compared to its opposite process that is calculating sampling points or generating a polygonal mesh from a smooth surface [96, 121, 137, 113, 139, 150, 166].

Many approximation methods have been developed to generate various surfaces. For example, some early approach addresses the problem of reconstructing explicit surfaces [138], some other approaches generate implicit surfaces [17, 162, 106, 136], and there are also surface approximation approaches that use subdivision surfaces [66, 70, 94, 102, 93]. This thesis focuses on parametric spline surface approximation that is to find a parametric surface expressed in spline representation.

Parametric spline surface approximation is an important step in reverse engineering [116]. Contrary to the traditional forward engineering which is also known as computer-aided engineering (CAE), reverse engineering is a process of converting an existing real world object to a computer-aided design (CAD) model. Usually, this is done by first obtaining a preliminary data representation of point clouds from the object. Especially with the advances in data acquisition techniques [149], many acquisition devices and systems are available during the last decade [11, 27, 86, 123, 164] and now it is possible to collect huge and high resolution data sets from the objects of large scale [90, 8, 79, 58, 63]. Since the point clouds lack sufficient geometrical and topological information, it is quite often to construct a triangular mesh from the dense point clouds. Many methods have been developed to convert unorganized data points into a triangular mesh [71, 147, 26, 141, 9] and sometimes further processing on triangular meshes, such as mesh fairing, mesh repairing, mesh simplification and re-meshing, is needed [61, 105, 83, 32, 41, 82, 51]. Finally, parametric spline surface approximation is required to convert the triangular models into a parametric surface representation, for example, NURBS which is the standard form in the CAD/CAM industry. Some typical usages of reverse engineering include recovering the digital model from a mechanical product where the original manuscript is lost or unavailable, shortening the

production period for computer animated movies by creating a digital model from a clay sculpture, and protecting the masterpieces of artists by generating a digital copy.

In general, when a surface approximation method is constructed, the following criteria should be considered:

- **Accuracy**: The geometrical error between the reconstructed surface and the original surface should be small. The features of the original surface should be faithfully preserved.
- **Continuity**: The reconstructed surface should have a desirable order of continuity. In general, the surface should achieve at least tangent plane continuity and quite often have curvature continuity as well.
- **Fairness**: The surface should be visually pleasing and avoid unnecessary fluctuations.
- **Conciseness**: The result surface representation should be concise and free of redundancy.
- **Automation**: The approach should be able to find an eligible surface automatically or with few user intervention.

The rest of this section will briefly discuss various surface approximation methods.

2.5.1 Categorization of parametric surface approximation

There is a large body of literatures on the parametric surface approximation. From different points of view, these approaches can be categorized in several ways. The first type of categorization is based on the precision of the resultant surface. If an approach generates a surface whose error compared to the original data are within a given tolerance, it is known as surface approximation or surface fitting [33, 165, 112]. If the result surface is required to exactly pass all the input data, the approach is known as surface interpolation [46, 97, 89]. It can be understood that surface interpolation is a special case of surface approximation

with zero error tolerance. Interpolation approaches ensure the accuracy at the input data points, but they usually result in complicated and redundant surfaces when the input model is densely sampled. Approximation approaches generate surfaces with more efficient representation and they also have better performance in handling the noises in the data.

The second type of categorization can be made based on the form of input data. Some approaches deal with point clouds [140, 103, 100], while the others deal with polygonal meshes [38, 53, 73]. In point clouds, the connectivity information is not specified and there may exist some ambiguities in the data. This could complicate the surface approximation problem. By contrast, the polygonal meshes contain connectivity relations among the data points, which can be exploited in surface approximation.

Also, the approaches can be classified according to the topology of the input data. Specifically, there are surface fitting approaches that deal with the input data of height fields [46, 126, 89], of disc homeomorphic topology [33, 53, 72], or of arbitrary topology [87, 38, 91]. To fit data of height fields, the scalar value spline functions can be used and the parameterization of the data naturally exists. The data that is homeomorphic to a 2D disc can be approximated by a single chart of spline surface defined on a rectangle domain. The approximation has a lot of applications in CAD and computer graphics and it has been extensively studied. It is also the base for the problem of fitting arbitrary topology data, in which the input data are normally segmented at the first stage so that each segment can be approximated by a spline surface. Segmentation of the data can be carried out manually/ semi-automatically [87, 92], automatically [38], or be inferred from the quad re-mesh techniques [34, 146, 84, 76]. When there are several segments, attention must be taken to make sure that the adjacent surfaces are smoothly merged at the common edges and corners. Conventional methods for this include stitching the neighboring surfaces [103, 87, 52, 92] and using some spline schemes for arbitrary topology structure [109, 95, 57, 114, 119, 129, 161, 59, 153] that guarantee the continuity between patches automatically.

The surface approximation approaches can be distinguished by whether they lead to a linear problem [89, 53] or a nonlinear problem [125]. The computational cost for solving a linear problem is generally much lower than that for solving a nonlinear problem. The nonlinear approaches, however, may describe the problem in a more accurate way, but the optimization cost is in general not affordable.

The surface approximation methods can be divided into non-adaptive methods and adaptive methods. In the non-adaptive methods [75, 112, 73], the surface fitting space where the optimal approximate surface is selected from will be globally changed in the surface approximation process. The adaptive methods [49, 38, 53], on the other hand, locally change the surface fitting space based on the previous approximation results. Adaptive methods can generate more meaningful and economic results, which makes them attractive in handling complex free-form models.

Finally, a classification can be made based on the arrangement of the input data. If the data points are arranged in a grid structure in the 3D space, they are called the gridded data; if the data points do not have such a structure and are arbitrarily distributed, they are called the scattered data. The surface approximation approaches that deal with the former data type are known as the gridded data fitting techniques [125, 49, 108, 28] and the approaches for the latter data type are known as the scattered data fitting techniques [53, 157, 73]. These two types of methods usually involve quite different techniques. Therefore, we give more detailed discussions on these two types of methods below.

2.5.2 Gridded data fitting

This subsection looks into the problem of gridded data fitting [6]. The data set is said to be gridded if the data points are arranged in an $M \times N$ grid in the 3D space. Thus each point $\mathbf{D}_{ij}$ could be simply assigned to a parameter pair (u_i, v_j) for $i = 1, 2, \cdots, M$ and $j = 1, 2, \cdots, N$ by relatively simple parameterization schemes such as uniform, chordal length or centripetal parameterization. For the other categories such as scattered data and

triangular meshes, data points are not orderly arranged and their parameterization could not be so straightforward.

In general, the problem of approximating a set of discrete data points can be converted into a least squares problem, which turns out to solve a linear system. Usually when the data set is huge, the size of the linear system would be large, leading to expensive computation. However, if the data is gridded, the situation could be simplified using the surface construction technique known as *surface skinning*, or *surface lofting* [159]. The original idea of surface skinning is to construct a B-spline surface through a collection of section curves. By applying such an idea to surface approximation for gridded data, the problem can be decomposed to a series of curve fitting problems. We first fit a spline curve to each column of gridded data. Such curves can be regarded as the iso-parametric curves on the surface because all the data points in a column have the same v parameter value. Then the control points of the iso-parametric curves are treated as the gridded data points and we perform further curve fittings on these points along the u direction. All the new control points obtained from the above two steps form the control points for the B-spline surface. Since the fitting processes for different columns or rows can be performed independently, several parallel algorithms [19, 20, 21] were developed for surface skinning. When enough processors were available, the method in [20, 21] can manage to reconstruct the surface in a constant time.

The surface skinning technique can also be applied to a certain variant of gridded data, which is called the row-wise data. That is, the data set is organized in rows except that the number of points in each row may differ. Vergeest [151] proposed a method for row-wise data, which fits the data points of each row by as fewest control points as possible, then resamples points on these curves uniformly and approximates the sampling points, and finally constructs a surface through the re-approximated curves using the surface skinning method. Piegl and Tiller [111, 112] construct the knot vectors more carefully and proposed to interpolate the i-th row of data by adopting as many knots that were used in the $(i-1)$-th

curves as possible. In this way, the number of the control points in the final skinned surface decreases significantly compared to that by the normal approach for row-wise data, which just directly merges all the knots together.

2.5.3 Scattered data fitting

Comparing to fitting gridded data, fitting scattered data or triangular meshes using spline surfaces is more challenging and there have been more research on this topic recently. As pointed out in literature [75, 33, 72, 156, 157, 73], it is usually difficult to obtain a satisfactory approximation result by only solving the least squares equations once. Instead, the "approximation-checking-adjustment" cycle needs to be iteratively performed for a number of times before a satisfactory approximation result is obtained. The step "adjustment" refers to parameter correction or control mesh refinement. Sometimes, to make the approximation surface smoother, a surface fairness functional should be taken into account besides the square distance between the surface and the data points. Moreover, if the data could not be approximated by a single surface, segmentation is needed. After that, each segment is approximated by a surface and care must be taken to make sure that the adjacent segments are smoothly connected.

A basic requirement for surface fitting is that the distance from the data points to the reconstructed surface should be within a given tolerance. The traditional methods using B-splines are either to initiate the fitting process with a control mesh that is sufficiently dense or to refine the control mesh globally when the approximation error is large. However, both of these strategies often result in an over-refined control mesh. In fact, the refinement of the control mesh is desired only in the areas where the fitting result is poor, but the B-spline surfaces do not allow such local refinement due to their formulation.

To overcome this drawback, adaptive methods were proposed. Basically there are two categories of adaptive methods. One is the hierarchical approach [49, 53, 89, 72, 165], and the other is the patchwork approach [126, 10, 38]. Since these works are closely related to

our research, below we examine them in details.

As the first category of adaptive approaches, Forsey and Bartels' method [49] fitted a hierarchical B-spline surface to the gridded data set. A B-spline surface with a coarse control mesh was first used to approximate the data set and it was treated as the base surface of the hierarchy. If this surface already fitted the data set very well, the approximation process stopped. Otherwise, the regions where the fitting error exceeded the tolerance were detected and in each of these regions, the residue of the data points was re-approximated by an overlay with a finer control mesh. The construction of the overlays in higher levels continued until the resulting hierarchical B-spline accurately fitted the data set. The limitation of this method was that it was only applicable for gridded data. The idea of fitting a hierarchical B-spline surface to gridded data was also adopted by Krishnamurthy and Levoy [87]. In their approach to dealing with the complex triangular meshes, they first manually segmented the mesh into several quadrilateral patches. By re-sampling of the mesh data in each patch into a grid of points, the method in [49] was then applied.

The problem of adaptive surface fitting for scattered data was discussed by Lee et al. [89], but the scattered data was restricted to be *scalar*. For scalar data, the parameterization problem becomes simple in that we can directly make the X-Y plane be the parameter space. The scalar data approximation could be useful in the areas of geology, oceanography and some experimental science, as shown in [64] and [10]. Its application could also be seen in 3D object reconstruction and even image processing [89]. In [89], a structure called *multilevel B-splines* was defined, in which a hierarchy of control nets was involved and the sum of the B-spline functions induced from these control nets determined the final approximation function. The control net at level k was a refinement of the one at level $k-1$ and all the control nets shared the same domain. The Multilevel B-spline Approximation (MBA) algorithm began by fitting a B-spline surface with a coarse control net to the data set, then it recursively approximated the residues of the data points left by the last fitted surface with finer control net, until the hierarchy of the control net reached the

predefined levels. For every control mesh in the hierarchy, only its non-zero control points were stored in a linear array. Since there were many zero control points in the higher levels, this adaptive approach saved a lot of storage. Based on the ideas in [89], Zhang et al. [165] improved the method by performing the refinement of the control mesh only in the regions where the approximation error was larger than the specified tolerance. This resulted in the representation of the surface in hierarchical B-splines. Lin and Huang [81] made another extension of Lee et al's method in which the non-uniform B-splines was adopted. In a recent approach, Hjelle [69] provided the explicit expressions for control mesh refinement which can result in C^1 or C^2 multilevel B-splines.

In [53] and [72], the approaches for adaptively approximating general scattered 3D data were discussed, with the restriction that an underlying planar parametric surface existed. To deal with such data, it was required to find a way to first assign parameter values to the data points before the fitting process. After that, the methods followed a way similar to [49] to construct a hierarchical B-spline surface. By using these methods, not only the superfluous control points were eliminated, but the complexity of solving the corresponding linear system for surface fitting was also reduced.

In the second category of adaptive methods, the patchwork surface usually consists of a number of Bézier patches with their boundaries smoothly connected to each other. Schmitt et al. [126] proposed an adaptive subdivision approach for surface fitting. In their method, the gridded data set was fitted with a patchwork of Bézier patches which were smoothly connected to achieve G^1 continuity. A patch would be further subdivided into four smaller patches and the corresponding control points were recalculated if the approximation error was not small enough. By this method, the total computational complexity was largely decreased. However, a number of constraints should be imposed to guarantee continuity between the Bézier patches.

Bertram et al. [10] presented a method to approximate a large number of scalar terrain data. The size of data is usually too large to be handled globally. In their method, the data

set was first organized in an initial cluster. After that, the data points in each partition of the cluster was fitted locally by a Bézier patch and the combination of the Bézier patches by means of knot removal became a C^1 continuous B-spline surface which was treated as the base surface of the hierarchical B-splines. Next, the cluster was refined based on the *quadtree* structure, i.e., each partition in the original cluster was further divided into four equal regions. For those partitions in the new cluster whose approximation residue exceeded the tolerance, further approximations were carried out to fit these residues. These patches constituted the overlays of the parent surface. The process continued until a satisfactory result was achieved.

Eck and Hoppe [38] proposed a method to fit arbitrarily topological triangular meshes. They constructed a patchwork of quadrilateral faces from the triangular mesh by using the harmonic maps [37]. Then they approximated the data points of each face by constrained Bézier patches in Peters' scheme [109]. Peters's scheme automatically assured the G^1 continuity on the boundaries of the patches. The Bézier patches might be further combined into a single B-spline patch which was more compact in representation. The adaptivity of this method lies in that, when the resulting surface was not accurate enough, only selected faces were chosen to be split and re-approximated.

Li et al. [91] also developed a method for fitting arbitrarily topological triangular meshes, which was mainly based on their work of global periodical parameterization [118]. They used T-NURCC as the surface scheme and achieved G^1 continuity at the extraordinary points and C^2 continuity elsewhere. In their method, although the construction of the control mesh had certain adaptivity, the initial setting for the control mesh seemed to be over refined and the control points in it were almost uniformly distributed.

Finally, it is worth pointing out that although the methods discussed above in both categories succeed in their specific applications, they are still not good and general enough. They have some disadvantages. For example, the methods of the first category achieve only partial optimization, which means that although the best solution is calculated at each level

of the hierarchy, their sum does not necessarily give the best solution globally. The first category maintains a hierarchy of control meshes which may not be so intuitive for interactive applications. As for the second category, the cost for global optimization is expensive, if all patches are re-approximated even though only a small part is not well fitted. In addition, the disadvantages of the second category also includes the redundancy in the multiple knots and the complication to maintain the continuity in the patchwork.

Chapter 3

T-spline Control Point Removal

3.1 Introduction

It is well known that B-spline basis functions can be refined by linear transformation and this property enables the important operation of B-spline knot insertion [14, 23]. By knot insertion, the number of the knots in a B-spline surface is increased and the shape of the surface can thus be modeled at a finer detail level. A reverse process of B-spline knot insertion is B-spline knot removal [67, 50], which aims to eliminate redundant knots from a B-spline surface without altering its shape. Knot removal is a fundamental operation for spline surfaces. It serves as an underlying tool for many applications such as representation reduction, shape fairing, base conversion and wavelet decomposition. While knot insertion can always be performed without introducing errors, removing a knot without changing the surface is possible only under certain circumstances. In general, approximation algorithms will be used for B-spline knot removal [99, 98]. Due to the restriction on the topology of B-spline surfaces, B-spline surface knot insertion and knot removal have a drawback: knots can only be added or removed in a row-wise or column-wise fashion in order to make the B-spline control mesh a regular grid.

T-splines were proposed to overcome the inflexibility of B-splines. T-splines allow local

refinement or local knot (and control point) insertion. In this chapter, we study the reverse process of inserting control point(s) into a T-spline surface, i.e., T-spline knot removal or control point removal. Two questions will to be tackled: the first one is to detect whether a specified T-spline control point could be removed; and the second one is to compute the updated T-mesh structure and geometry of the T-spline surface after a removable control point is removed. Compared to the B-spline knot removal in which a whole row (or column) of control points needs to be removed, our control point removal for T-splines focuses on the removal of a single control point, which usually causes only local change to the T-spline control grid.

Previous work of T-spline control point removal was reported in [128] where the problem of T-spline surface simplification was considered. The method starts with a simple B-spline surface defined by a 4×4 control grid, and then adaptively refines the grid until the least squares T-spline surface defined over the refined grid approximates the original T-spline surface within the given tolerance. If the tolerance is chosen to be zero, then the control point removal can be achieved. The method is global in nature and is useful for eliminating as many control points as possible. In this chapter, however, we seek local knot removal and try to eliminate a single point or a few points which is/are specified by a user. Compared to the approach in [128], our approach is more intuitive and user controllable. Therefore, it is more required in interactive applications.

Note that the local refinement algorithm is one of the most fundamental algorithms in T-spline theory and technology and it also plays an important role in the research of this thesis. Therefore in Section 3.2 we first re-examine T-spline's local refinement algorithm, aiming to gain some insights, and provide a simple detailed implementation of it. Then in Section 3.3, we present a T-spline control point removal algorithm which removes one control point from a T-spline surface. The possible extension of the algorithm for removing more than one control point is discussed in Section 3.4.

3.2 T-spline local knot insertion

Consider a T-spline surface defined in the homogeneous form:

$$\mathbf{S^H}(u,v) = \sum_{i=1}^{n} \mathbf{Q}_i B_i(u,v) \tag{3.1}$$

where the $\mathbf{Q}_i = (w_i P_i, w_i) = (w_i x_i, w_i y_i, w_i z_i, w_i)$ are control points in homogeneous space, the w_i are control point weights and the P_i are the control points in the 3D space. The $B_i(u,v)$ are T-spline blending functions corresponding to control point $\mathbf{Q}_i$ and are defined by Equation (2.8). The knot quintuples $\mathbf{u}_i$ and $\mathbf{v}_i$ are extracted from the T-mesh based on certain rules given in Chapter 2.

By local knot insertion, we mean to insert one or more control points into a T-mesh without changing the shape of the T-spline surface and without affecting too many existing control points. Generally, if a control point is simply inserted into a T-mesh without any adjustment, the shape of the surface would be likely altered. This is because some T-spline blending functions associated to the control points have been changed since their knot quintuples inferred from the T-mesh are now different (Refer to some detailed examples in Figure 3.3 and Figure 3.4). However, if certain equivalent transformation from the original blending functions to those indicated by the new T-mesh could be found, the geometry of the surface would be correctly preserved. In the following we first discuss the T-spline blending function transformation and then explore T-spline surface's local control point insertion.

3.2.1 Blending function refinement

A T-spline blending function is the product of two univariate B-spline basis functions determined by two knot quintuples. If some knots are added into a knot quintuple, the blending

function can be decomposed into a linear combination of a few new blending functions that have finer knot quintuples. We call such a blending function transformation the blending function refinement.

We now demonstrate how such blending function refinement is achieved. Consider the u direction B-spline basis function. Let k be a new knot to be inserted into the knot quintuple $\mathbf{u} = [u_0, u_1, u_2, u_3, u_4]$ of the basis function $N^3[\mathbf{u}](u)$. Depending on where k is located, we have the following result:
If $u_0 < k \leq u_1$, then

$$N^3[\mathbf{u}](u) = aN^3[u_0, k, u_1, u_2, u_3](u) + N^3[k, u_1, u_2, u_3, u_4](u) \tag{3.2}$$

If $u_1 < k \leq u_2$, then

$$N^3[\mathbf{u}](u) = aN^3[u_0, u_1, k, u_2, u_3](u) + bN^3[u_1, k, u_2, u_3, u_4](u) \tag{3.3}$$

If $u_2 < k \leq u_3$, then

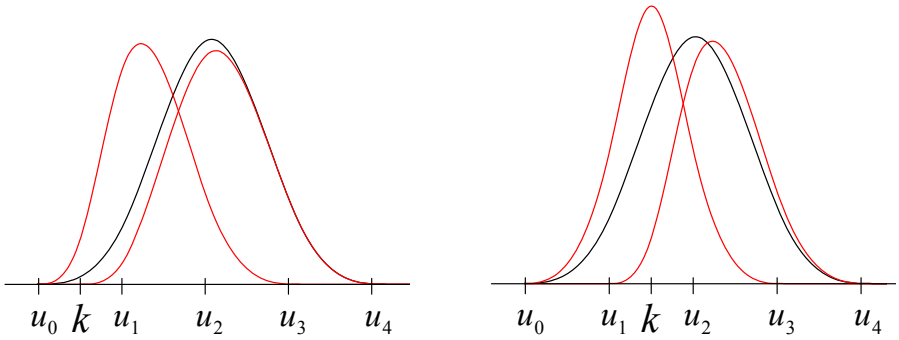
$$N^3[\mathbf{u}](u) = aN^3[u_0, u_1, u_2, k, u_3](u) + bN^3[u_1, u_2, k, u_3, u_4](u) \tag{3.4}$$

If $u_3 < k < u_4$, then

$$N^3[\mathbf{u}](u) = N^3[u_0, u_1, u_2, u_3, k](u) + bN^3[u_1, u_2, u_3, k, u_4](u) \tag{3.5}$$

If $k \leq u_0$ or $k \geq u_4$, $N^3[\mathbf{u}](u)$ will not be refined.
In all the above cases, $a = (k - u_0)/(u_3 - u_0)$ and $b = (u_4 - k)/(u_4 - u_1)$. Figure 3.1 illustrates these situations, where the original basis function is shown in black color and the new basis functions are shown in red color.

If more than one knot are inserted, the refinement can be achieved by repeatedly ap-



(a) $u_0 < k \leq u_1$ (b) $u_1 < k \leq u_2$

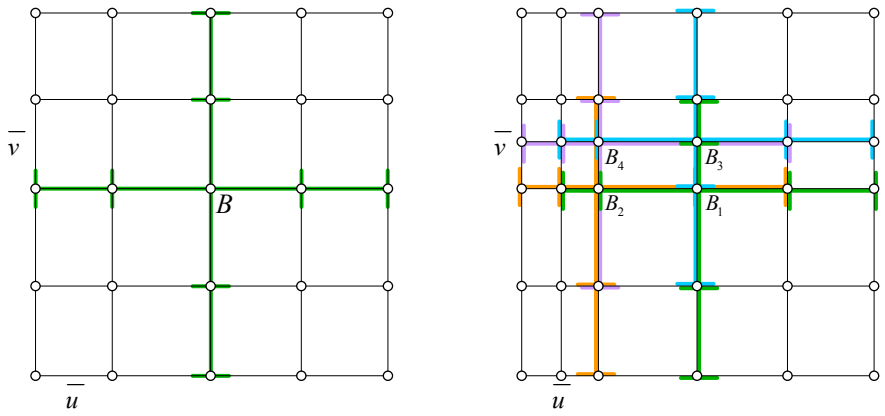
(c) $u_2 < k \leq u_3$ (d) $u_3 < k < u_4$

Figure 3.1: Basis function refinement.

plying the above equations. These results can also be applied to the v direction B-spline basis functions. Therefore, from the transformation of both B-spline basis functions, we can obtain the transformation equations for the T-spline blending functions. For example, Figure 3.2(a) shows the knot quintuples of a blending function $B(u, v)$. If two new knots $\bar{u}$ and $\bar{v}$ are inserted, $B(u, v)$ then becomes the combination of four new blending functions as shown in Figure 3.2(b):

$$B(u, v) = c_1 B_1(u, v) + c_2 B_2(u, v) + c_3 B_3(u, v) + c_4 B_4(u, v) \tag{3.6}$$

where the coefficients c_i are calculated by Equations (3.2)-(3.5).



(a) (b)

Figure 3.2: Blending function refinement.

3.2.2 Local knot insertion

In a T-spline surface, each knot corresponds to a control point. So the local knot insertion is also known as the local control point insertion. The T-spline local control point insertion algorithm was first proposed in [129] and the approach was improved in [128]. The local refinement of T-splines means the capability of adding one or more control points into the T-mesh without altering the surface. The surfaces before and after the refinement are thus the same for any parameter values. That is,

$$\mathbf{S^H}(u,v) = \sum_{i=1}^{n} Q_i B_i(u,v) \equiv \sum_{i=1}^{\tilde{n}} \tilde{Q}_i \tilde{B}_i(u,v) = \tilde{\mathbf{S}}^{\mathbf{H}}(u,v) \tag{3.7}$$

where $\tilde{\mathbf{S}}^{\mathbf{H}}(u,v)$ is the T-spline surface after the control point insertion, and $\tilde{Q}_i, \tilde{n}, \tilde{B}_i(u,v)$ are the new control points, the new control point number and the new blending functions.

The main idea of the T-spline control point insertion algorithm in [128] is that the original T-spline surface equation can be reformulated by decomposing the basis functions and recombining the control points afterwards if they correspond to the same basis function. Eventually we have to maintain the validity of the T-mesh and to ensure that all the blending functions and the control points are properly associated while keeping the equivalence between the new and original surface equations. During the processes of decomposition

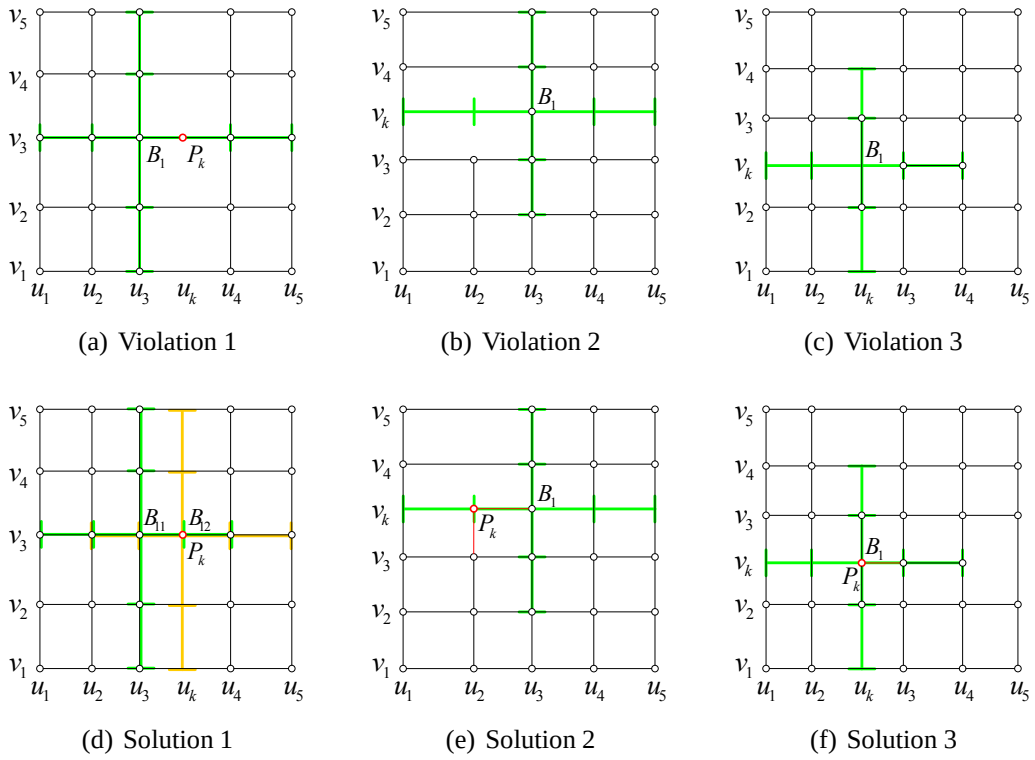
Table 3.1: Violations during T-spline control point insertion.

	Violation	Solution
1.	A blending function is missing a knot inferred from the current T-mesh.	Refine the blending function at the missing knot.
2.	A blending function has a knot that is not indicated in the current T-mesh.	Insert the knot into the T-mesh.
3.	A blending function has no associated control point in the current T-mesh.	Insert an extra control point corresponding to the central knots of the blending function's knot quintuples.

and re-arrangement, temporary discordances between the blending functions and the T-mesh may occur. These mismatches are called the violations. We resolve the violations either by applying blending function refinement or by adding an extra control point at a certain location.

There are three types of violations, as tabulated in Table 3.1. Violation 1 is illustrated in Figure 3.3(a) where P_k is a newly inserted control point at coordinates (u_k, v_3) and $B_1(u, v)$ is an old T-spline blending function with knot quintuples $\mathbf{u}_1 = [u_1, u_2, u_3, u_4, u_5]$ and $\mathbf{v}_1 = [v_1, v_2, v_3, v_4, v_5]$. Note that the knot quintuple $\mathbf{u}_1$ of $B_1(u, v)$ does not include knot u_k which is in the new T-mesh. We call such a situation violation 1. To resolve violation 1, we split $B_1(u, v)$ using blending function refinement and obtain two new blending functions $B_{11}(u, v)$ with knot quintuples $\mathbf{u}_{11} = [u_1, u_2, u_3, u_k, u_4]$ and $\mathbf{v}_{11} = [v_1, v_2, v_3, v_4, v_5]$ and $B_{12}(u, v)$ with knot quintuples $\mathbf{u}_{12} = [u_2, u_3, u_k, u_4, u_5]$ and $\mathbf{v}_{12} = [v_1, v_2, v_3, v_4, v_5]$, as shown in Figure 3.3(d). $B_{11}(u, v)$ and $B_{12}(u, v)$ are compatible to the new T-mesh.

Figure 3.3(b) explains violation 2. The blending function $B_1(u, v)$ has knot quintuples $\mathbf{u}_1 = [u_1, u_2, u_3, u_4, u_5]$ and $\mathbf{v}_1 = [v_2, v_3, v_k, v_4, v_5]$. $B_1(u, v)$ might be the result of the blending function refinement operation of a nearby blending function. While the knot quintuple $\mathbf{u}_1$ contains the knot u_2, the T-mesh has neither a corresponding control point nor a corresponding vertical edge at (u_2, v_k). This results in violation 2. Violation 2 can be resolved by adding an extra control point P_k at (u_2, v_k), as shown in Figure 3.3(e). After that,



(a) Violation 1 (b) Violation 2 (c) Violation 3

(d) Solution 1 (e) Solution 2 (f) Solution 3

Figure 3.3: Three possible violations during knot insertion and their solutions.

$B_1(u, v)$ would be in accordance with the T-mesh.

Violation 3 is explained by Figure 3.3(c). The blending function $B_1(u, v)$ has knot quintuples $\mathbf{u}_1 = [u_1, u_2, u_k, u_3, u_4]$ and $\mathbf{v}_1 = [v_1, v_2, v_k, v_3, v_4]$ and is centered at (u_k, v_k), but there is no corresponding control point in the T-mesh for $B_1(u, v)$. This violation is resolved by inserting a control point P_k at (u_k, v_k), as shown in Figure 3.3(f).

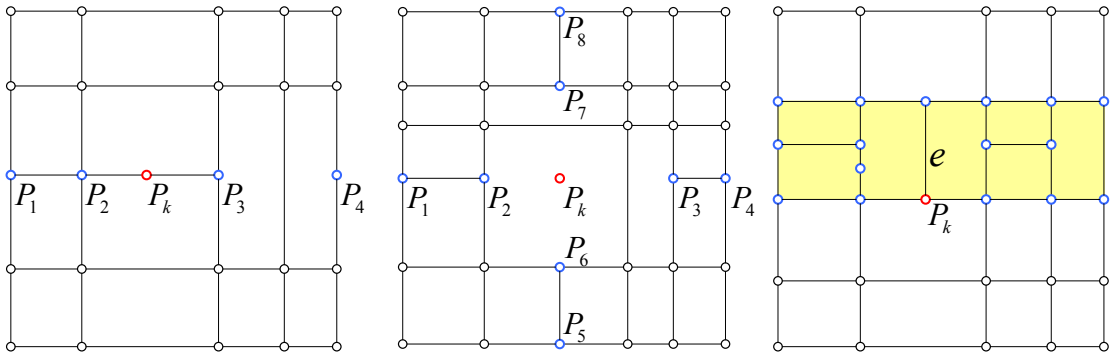
In order to obtain the correct surface representation after the control point insertion, all the violations between the blending functions and the T-mesh should be detected and fixed. This can be done by examining all the blending functions associated to each control point in the T-mesh. In general this is quite time consuming and inefficient especially when the number of the control points is large. Due to the local support property of the blending functions, actually only a few blending functions might be affected by the insertion of a control point. Depending on where and how a new control point is inserted, it can be estimated which blending functions might be affected. Thus it is sufficient to check only

these blending functions for violations in actual implementation of the T-spline local knot insertion algorithm.

Table 3.2: Situations on control point insertion.

	Situation	Possible influence region
1.	A control point is inserted onto an existing edge in the T-mesh.	The four immediate neighbor control points of the inserted point in the same parameter direction of the edge.
2.	A control point is inserted into a face in the T-mesh.	The four immediate neighbor control points of the inserted control point in both parameter directions.
3.	The insertion of a control point causes an edge to be added into the T-mesh.	All the control points in the parameter region that can be orthogonally projected onto the edge.

We propose to consider the following three situations, as listed in Table 3.2. The first situation is illustrated in Figure 3.4(a), where a new control point P_k is being added into a horizontal edge that connects P_2 and P_3. In this case, violations might occur for those blending functions associated to the four immediate neighbor points of P_k (two from either side) in the $u-$parameter direction, i.e. the control points P_1, P_2, P_3, P_4 in the figure. It is possible that some of the suspect blending functions are not affected, such as the one associated to P_4. Only the blending functions associated to P_1, P_2 and P_3 actually contain violations.



(a) Situation 1 (b) Situation 2 (c) Situation 3

Figure 3.4: Different situations of control point insertion.

The second situation is illustrated in Figure 3.4(b), where a control point P_k is being added in a face of the T-mesh. In this situation, we check the blending functions associated to the four immediate neighbors of P_k in both the $u-$ and $v-$directions, namely P_1, P_2, P_3, P_4 and P_5, P_6, P_7, P_8.

The third situation is illustrated in Figure 3.4(c), where the insertion of control point P_k causes an edge e to be added into the T-mesh. In this situation, the blending functions associated to all the control points that can be orthogonally projected onto the edge need to be checked. The suspect area is highlighted in Figure 3.4(c).

Sometimes, the insertion of a knot can be categorized into more than one of these situations. In that case, we should check all the blending functions in the area indicated by each matched situation.

Based on the above discussion, a detailed T-spline control point insertion algorithm is described in Algorithm 3.1. The algorithm is quite self explanatory and we just make a few more clarifications. A stack T is used in the algorithm to store all the blending functions that contain violations, together with the associated control points. We search for the violating blending functions in partial regions of the T-mesh, determined by where the points are inserted. We use Q_i and $B_i(u, v)$ to denote the control points and blending functions in the T-mesh and use R_i and $\hat{B}_i(u, v)$ to denote the control points and blending functions in the stack. These blending functions and their associated control points are put back to the T-mesh when the violations are resolved through the blending function refinement or the insertion of extra control points. The algorithm terminates when the stack T is empty.

Since the procedure of the control point insertion does not change the values of the surface equation and more specifically, we have $\sum w_i B_i(u, v) = \sum \tilde{w}_i \tilde{B}_i(u, v)$, therefore a standard/semi-standard T-spline surface would remain to be standard or semi-standard after the control point insertion and a nonstandard T-spline surface remains to be nonstandard. Also, the algorithm is guaranteed to terminate.

Finally, we use an example to demonstrate the T-spline control point insertion process.

Algorithm 3.1 : T-spline knot insertion algorithm

Given: a T-spline surface $\mathbf{S^H}(u,v)$ and a set of new points that are to be added
Goal: a new T-spline representation $\tilde{\mathbf{S}}^{\mathbf{H}}(u,v)$ of the same surface

1. Create an empty stack T
2. Add new control points at the specified locations and add new edges when necessary
3. Based on different situations, decide the collection of $B_i(u,v)$ to be examined
4. **for** each $B_i(u,v)$ in the collection **do**
5. **if** $B_i(u,v)$ violates the current T-mesh **then**
6. Let $R_i = Q_i$ and $\hat{B}_i(u,v) = B_i(u,v)$
7. Push the pair $(R_i, \hat{B}_i(u,v))$ into T
8. Update the control point in the T-mesh as $Q_i = (0,0,0,0)$
9. Update the blending function $B_i(u,v)$, making it compatible to the T-mesh
10. **end if**
11. **end for**
12. **while** T is not empty **do**
13. Take the top element $(R_j, \hat{B}_j(u,v))$ out of T
14. Find $B_i(u,v)$ in the T-mesh that has the same center knot as $\hat{B}_j(u,v)$
15. **if** there is no such $B_i(u,v)$ in the T-mesh **then** {Violation 3}
16. Push the pair $(R_j, \hat{B}_j(u,v))$ back into T
17. Insert a control point into the T-mesh at the center knot of $\hat{B}_j(u,v)$
18. Add edges if necessary
19. Execute Line 3-11 to update the stack T and the T-mesh
20. **else if** $B_i(u,v)$ has a knot that refines $\hat{B}_j(u,v)$ **then** {Violation 1}
21. Carry out a proper blending function refinement:
$\hat{B}_j(u,v) = c_1\hat{B}_{j1}(u,v) + c_2\hat{B}_{j2}(u,v)$
22. Push the pairs $(c_1R_j, \hat{B}_{j1}(u,v))$ and $(c_2R_j, \hat{B}_{j2}(u,v))$ into T
23. **else if** $\hat{B}_j(u,v)$ has a knot that $B_i(u,v)$ does not have **then** {Violation 2}
24. Push the pair $(R_j, \hat{B}_j(u,v))$ back into T
25. Insert a control point into the T-mesh at that knot
26. Add edges if necessary
27. Execute Line 3-11 to update the stack T and the T-mesh
28. **else** {no violation: $\hat{B}_j(u,v)$ has the same knot quintuples as $B_i(u,v)$}
29. Update the control point associated with $B_i(u,v)$ as: $Q_i = Q_i + R_j$
30. **end if**
31. **end while**

We begin with a T-spline surface whose T-mesh pre-image is shown in Figure 3.5(a). After inserting a point P_1 into the T-mesh, a new T-spline surface is obtained, whose T-mesh pre-image is shown in Figure 3.5(b). We further add a point P_2 into the T-mesh and now the T-mesh pre-image is the one shown in Figure 3.5(c). It can be seen that changes to the T-mesh only take place in a local area, although a few extra control points are needed to be added as well. The T-spline surfaces and the T-meshes for the pre-images in Figure 3.5(a) to 3.5(c) are displayed in Figure 3.5(d) to 3.5(f), respectively. While the T-meshes are now different, the T-spline surfaces are the same.

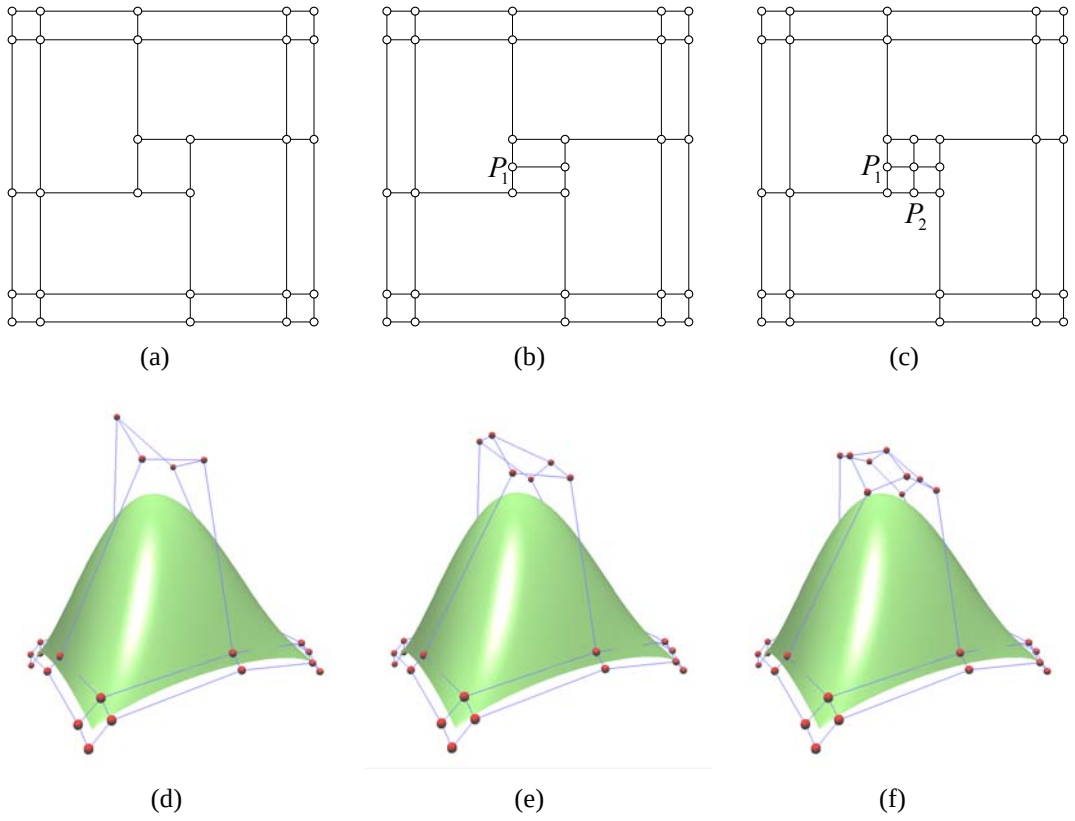


Figure 3.5: An example of knot insertion.

3.3 Removing one control point from a T-spline surface

Now we present an algorithm for T-spline control point (knot) removal. The algorithm is based on two fundamental operations. One is the blending function refinement that has

already been explained in the preceding section. The other is the reverse blending function transformation. In the following, we explain the reverse blending function transformation first and then the T-spline control point removal algorithm.

3.3.1 Reverse blending function transformation

While the blending function refinement is used to split a basis function into two new ones with finer knot quintuples, the reverse blending function transformation presented here works in an opposite way.

Let $\mathbf{u} = [u_0, u_1, u_2, u_3, u_4]$ denote a knot quintuple in which u_2 is the center knot. $N^3[\mathbf{u}](u) = N^3[u_0, u_1, u_2, u_3, u_4](u)$ is the associated B-spline basis function defined on **u**. Now suppose that a new knot quintuple $\mathbf{u}'$ is constructed from **u** by eliminating a knot u_i ($i = 0, 1, 3$,or 4) that is other than the center knot in **u** , inserting another knot u_{add} which satisfies $u_{add} \leq u_0$ or $u_{add} \geq u_4$, and meanwhile keeping the center knot of $\mathbf{u}'$ still to be u_2. Let the B-spline basis function corresponding to $\mathbf{u}'$ be denoted as $N^3[\mathbf{u}'](u)$. $N^3[\mathbf{u}](u)$ can be re-expressed in the form of $N^3[\mathbf{u}](u)$ plus another term. Since the knot span of $\mathbf{u}'$ is larger than that of **u**, such an operation is called the reverse basis function transformation which is essentially derived from the equation of the basis function refinement. There are four different types of reverse basis function transformation, depending on which knot in **u** is replaced.

If $u_{add} \leq u_0$ and $\mathbf{u}' = [u_{add}, u_1, u_2, u_3, u_4]$, then

$$N^3[u_0, u_1, u_2, u_3, u_4](u) = c_0 N^3[u_{add}, u_1, u_2, u_3, u_4](u) + d_0 N^3[u_{add}, u_0, u_1, u_2, u_3](u) \tag{3.8}$$

where $c_0 = 1$ and $d_0 = \frac{u_{add} - u_0}{u_3 - u_{add}}$.

If $u_{add} \leq u_0$ and $\mathbf{u}' = [u_{add}, u_0, u_2, u_3, u_4]$, then

$$N^3[u_0, u_1, u_2, u_3, u_4](u) = c_1 N^3[u_{add}, u_0, u_2, u_3, u_4](u) + d_1 N^3[u_{add}, u_0, u_1, u_2, u_3](u) \tag{3.9}$$

where $c_1 = \frac{u_4 - u_0}{u_4 - u_1}$ and $d_1 = \frac{(u_{add} - u_1)(u_4 - u_0)}{(u_3 - u_{add})(u_4 - u_1)}$.

If $u_{add} \geq u_4$ and $\mathbf{u}' = [u_0, u_1, u_2, u_4, u_{add}]$, then

$$N^3[u_0, u_1, u_2, u_3, u_4](u) = c_2 N^3[u_0, u_1, u_2, u_4, u_{add}](u) + d_2 N^3[u_1, u_2, u_3, u_4, u_{add}](u) \tag{3.10}$$

where $c_2 = \frac{u_4 - u_0}{u_3 - u_0}$ and $d_2 = \frac{(u_{add} - u_3)(u_4 - u_0)}{(u_1 - u_{add})(u_3 - u_0)}$.

If $u_{add} \geq u_4$ and $\mathbf{u}' = [u_0, u_1, u_2, u_3, u_{add}]$, then

$$N^3[u_0, u_1, u_2, u_3, u_4](u) = c_3 N^3[u_0, u_1, u_2, u_3, u_{add}](u) + d_3 N^3[u_1, u_2, u_3, u_4, u_{add}](u) \tag{3.11}$$

where $c_3 = 1$ and $d_3 = \frac{u_{add} - u_4}{u_1 - u_{add}}$.

The reverse blending function transformation for a T-spline blending function $B(u, v)$ can be easily derived from the above four equations. In general, if both $N^3[\mathbf{u}](u)$ and $N^3[\mathbf{v}](v)$ are decomposed, we can rewrite $B(u, v)$ by

$$B(u, v) = N^3[\mathbf{u}](u) N^3[\mathbf{v}](v) = \sum_i c_i N^3[\mathbf{u_i}](u) \cdot \sum_j c_j N^3[\mathbf{v_j}](v) = \sum_k r_k B_k(u, v) \tag{3.12}$$

3.3.2 T-spline control point removal algorithm

Now let us look at how to eliminate a specified control point from the T-mesh without altering the surface. This process is also called T-spline knot removal due to the fact that removing a control point causes the corresponding knot to be removed from the T-spline pre-image in the parameter domain as well.

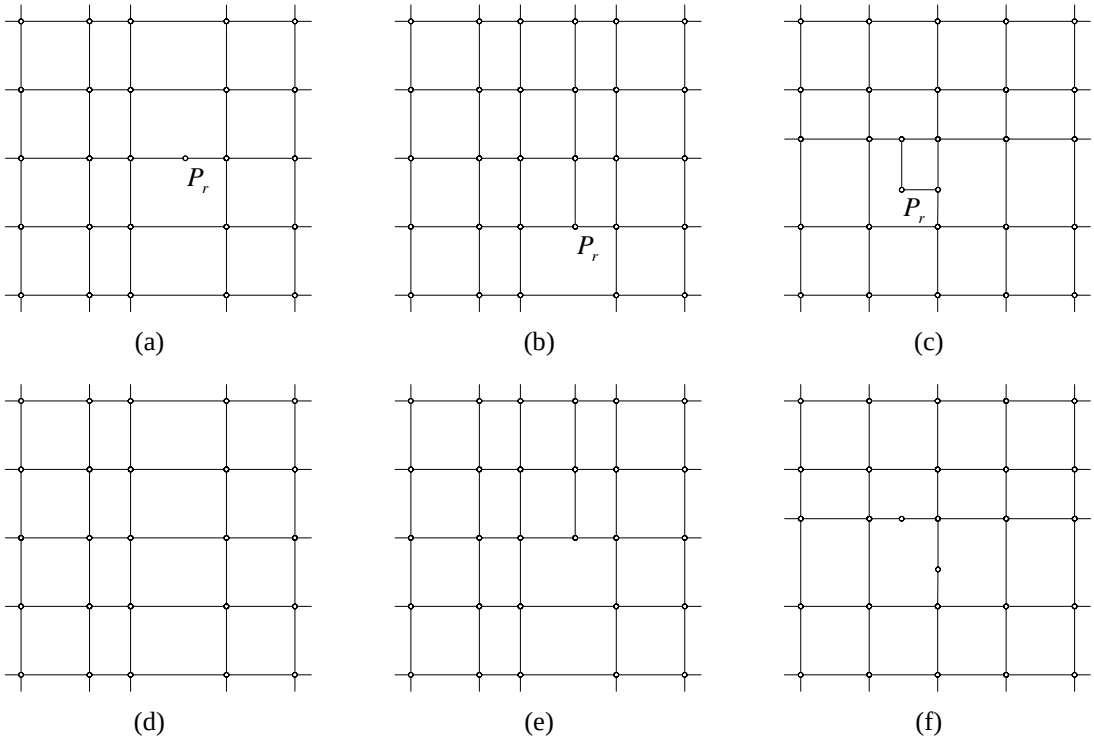
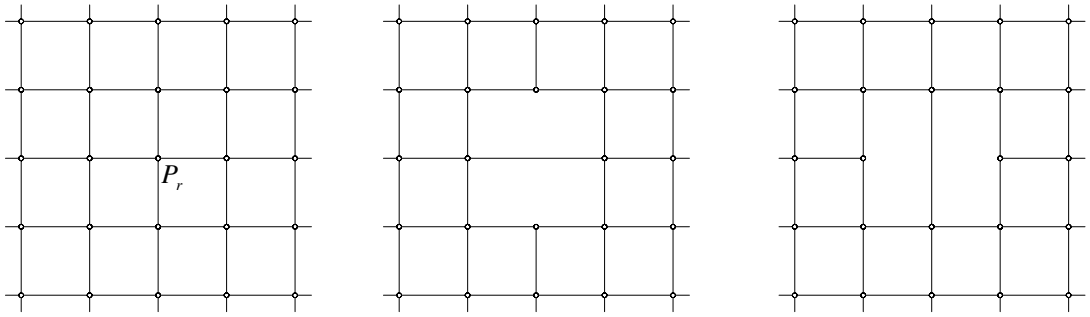


Figure 3.6: T-mesh structure change after removing control point P_r.

An immediate result of removing a control point is the change of the structure of the T-mesh. Such change includes the disappearance of the control point, and possible removing or adding of some edge(s) due to the removal of that point. Figure 3.6 shows three examples of the structure change. Figure 3.6(d) to 3.6(f) are the results of removing P_r from the T-mesh shown in Figure 3.6(a) to 3.6(c), respectively. Sometimes the structure for the new T-mesh is not unique. Refer to Figure 3.7 for a more complicated example, where Figure 3.6(b) and Figure 3.6(c) are two possible topological structures when the control point P_r is removed from the T-mesh shown in Figure 3.7(a). In such a case, both situations could be checked or the user's recommendation may be needed.

Another important component of the T-spline knot removal algorithm is to update the geometry of the control points so as to keep the shape of the T-spline surface unchanged. Assume we want to eliminate the control point P_r which corresponds to knot (u_r, v_r). Our approach begins with the given T-spline surface. The T-spline surface equation $\mathbf{S}^{\mathbf{H}}(u, v) =$



(a) Original T-mesh (b) Removing vertical edges (c) Removing horizontal edges

Figure 3.7: Another T-mesh structure change example.

$\sum_{i=1}^{n} Q_i B_i(u,v)$ is split into two parts: $\sum_{i\neq r} Q_i B_i(u,v)$ and $Q_r B_r(u,v)$ (Q_r is the corresponding point of P_r in the homogeneous space). We call the second part a residue. The first part defines a new T-spline surface whose control points one-to-one correspond to those of the new T-mesh. However, the knot quintuples for the blending functions in $\sum_{i\neq r} Q_i B_i(u,v)$ do not necessarily match those derived from the new T-mesh. It is important to keep in mind that the blending functions and the T-mesh are tightly coupled in a valid T-spline surface. Therefore the main process of our algorithm is to use the reverse blending function transformation and the blending function refinement to update both $\sum_{i\neq r} Q_i B_i(u,v)$ and $Q_r B_r(u,v)$ such that their blending functions gradually match the new T-mesh except that $B_r(u,v)$ has (u_r, v_r) as its center knots in the knot quintuples. During this process, local knot insertion may also be required (see a discussion in the end of this section). As a result, $\sum_{i=1}^{n} Q_i B_i(u,v)$ will eventually be decomposed into a T-spline surface defined over the new T-mesh and a residue term whose blending function has knot quintuples centered at (u_r, v_r). If the residue term becomes zero, a valid new T-spline surface without the control point P_r has been found. Otherwise, the point P_r cannot be removed.

The T-spline control point removal algorithm is thus given in Algorithm 3.2. Note that this algorithm is in the similar fashion of the T-spline knot insertion algorithm proposed in [128]. The main different step is step 3.2) which invokes the operation of reverse blend-

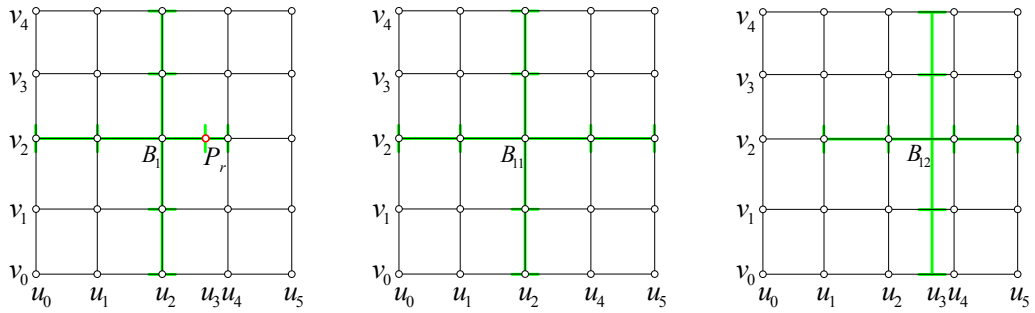
Algorithm 3.2 : T-spline control point removal algorithm

Given: a T-spline surface $\mathbf{S^H}(u, v)$ and a control point P_r corresponding to the knots (u_r, v_r)

Goal: If possible, find a new T-spline representation $\tilde{\mathbf{S}}^{\mathbf{H}}(u, v)$ of the same surface which does not contain P_r

1. Set the current T-spline surface in the homogeneous form to be $\sum_{i \neq r} Q_i B_i(u, v)$ and the residue to be $Q_r B_r(u, v)$
2. **repeat**
3. **if** any $B_i(u, v)$ in the current T-spline surface has the same knot quintuples as the residue's blending function **then**
4. Move it to the residue
5. **else if** any $B_i(u, v)$ in the current T-spline surface contains a knot (u_r, v_r) such that at least one of u_r and v_r is not the center in the respective knot quintuple **then**
6. Perform a proper reverse blending function transformation
7. **else if** any $B_i(u, v)$ in the current T-spline surface is missing a knot inferred from the current T-mesh **then**
8. Perform a proper blending function refinement
9. **else if** any $B_i(u, v)$ in the current T-spline surface has a knot other than (u_r, v_r), which is not indicated in the current T-mesh **then**
10. Add an appropriate control point into the T-mesh
11. **end if**
12. **if** the blending function of the residue is missing a knot inferred from the current T-mesh **then**
13. Perform a proper blending function refinement and move the new generated term whose corresponding knot quintuples are not centered at (u_r, v_r) to the current T-spline surface
14. **end if**
15. **until** there is no new operation
16. **if** the final residue equals to zero **then**
17. The control point P_r is successfully removed
18. **else**
19. The control point P_r cannot be removed
20. **end if**

ing function transformation. Here we use an example to illustrate this step topologically. Suppose we intend to remove the point P_r shown in Figure 3.8(a) from the T-mesh. After removing P_r, the T-mesh pre-image becomes Figure 3.8(b). However, the knot quintuples for the blending function $B_1(u, v)$ shown in Figure 3.8(a) are $\mathbf{u}_1 = [u_0, u_1, u_2, u_3, u_4]$ and $\mathbf{v}_1 = [v_0, v_1, v_2, v_3, v_4]$. It has a knot (u_3, v_2) that corresponds to the removed control point P_r. Therefore, according to step 3.2), a reverse blending function transformation is performed and we obtain two new blending functions: $B_{11}(u, v)$ with knot quintuples $\mathbf{u}_{11} = [u_0, u_1, u_2, u_4, u_5]$ and $\mathbf{v}_{11} = [v_0, v_1, v_2, v_3, v_4]$ and $B_{12}(u, v)$ with knot quintuples $\mathbf{u}_{12} = [u_1, u_2, u_3, u_4, u_5]$ and $\mathbf{v}_{12} = [v_0, v_1, v_2, v_3, v_4]$. as shown in Figure 3.8(b) and Figure 3.8(c). The former conforms with the current T-mesh, and the latter has the same knot quintuple as the residue and thus is moved to the residue.



(a) (b) (c)

Figure 3.8: Invoking the step of reverse blending function transformation.

3.3.3 Validity of the algorithm

For an algorithm described in a recursive manner, it is important that the algorithm terminates after a finite number of steps. We examine the two basic operations in this T-spline control point removal algorithm. Since the knot values involved in this procedure are those that initially exist in the T-mesh, the blending function refinement would be called for only a limited number of times if such a process is needed [128]. For the reverse blending function transformation, it can be seen that each of those four reverse blending function

transformations replaces a blending function by two new functions. One of the new blending functions corresponds to the knot quintuples which do not contain the removed knot, and the other one corresponds to the knot quintuples which are closer to the quintuples of the removed control point. Once the center knot of the knot quintuples becomes the knot to be removed, the reverse blending function transformation is completed. In this way, after a finite number of steps of performing reverse blending function transformation and blending function refinement, the T-spline surface is decomposed into a new T-spline surface which is defined by the new T-mesh without the removed control point plus a residue term whose blending function has knot quintuples centered at (u_r, v_r). If the coefficient of the residue term is zero, then the removal algorithm succeeds and the new T-spline surface is the result. Otherwise, the algorithm returns that the specified point cannot be removed. Therefore, the algorithm for T-spline control point removal is always guaranteed to terminate.

3.3.4 Discussion

In the process of removing a control point, sometimes the algorithm will introduce a few new control points into the T-mesh. The insertion of these control points is to make the blending functions be properly associated with the control points. Figure 3.9 illustrates such a situation. If the point P_1 shown in Figure 3.9(a) is removed from the T-mesh, then a new control point P_4 will automatically be added into the T-mesh by our algorithm (see Figure 3.9(b)), which ensures that the blending function corresponding to P_3 is compatible with the T-mesh.

It should be pointed out that in the situation where a knot removal causes the insertion of extra point(s), the total number of the control points will not be reduced, and thus the user may choose not to do knot removal of that point for applications such as surface simplification. However, if the user's concern is whether a specified point is removable and how to remove it, our algorithm is attractive because the structure of the new T-mesh is automatically determined by the algorithm. Some other possible approaches for knot re-

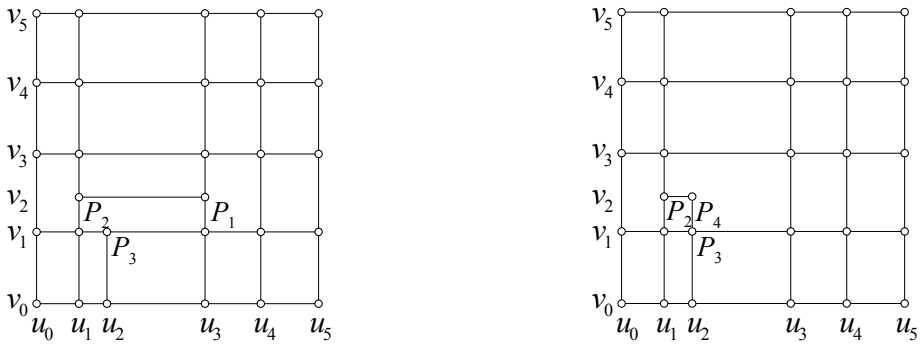


Figure 3.9: Extra control point insertion in the removal process.

moval such as setting up a system of linear equation describing the relationship between the blending functions (or control points) before the removal and after the removal need to know the structure of the new T-mesh in advance.

3.4 Removing more than one control point

This section extends the algorithm developed in Section 3.3 to remove more than one control point.

If a user specifies n control points in a T-mesh, we may extend the algorithm to detect whether these n control points can be removed simultaneously and to compute the new T-mesh if they are removable. The possible modifications include: 1) n control points should be removed in the new T-mesh; and 2) the residue should consist of n terms. However, the structure of the resulting T-mesh after removing several control points could generally have many possibilities. This increases the complexity of the algorithm. In addition, it is unlikely that arbitrarily specified n control points can be removed simultaneously. Therefore, if we want to remove many control points (especially those generated by knot insertion), it is not practical to identify them first and then to apply the extended algorithm.

An alternative approach could be based on the single control point removal algorithm. An unsophisticated attempt is described as follows: for every control point in the current

T-mesh, check its removability; if it is removable, remove it. This method is quite simple. However, the following example shows that this method may fail to remove some control points although they are generated by knot insertion.

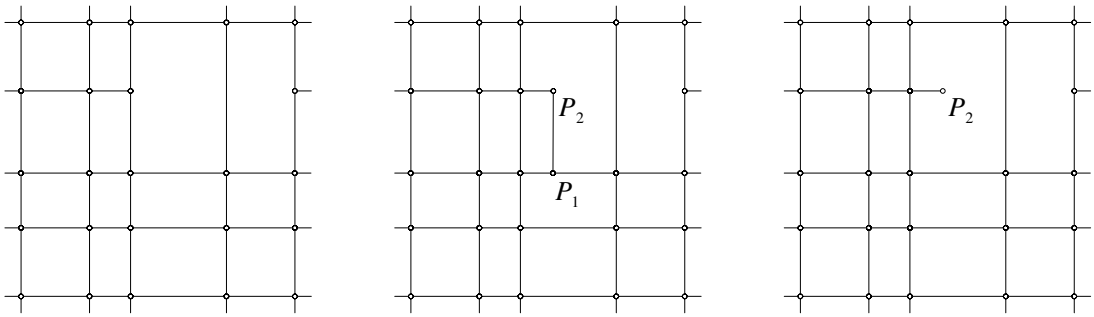


Figure 3.10: Example for identifying the removable control points.

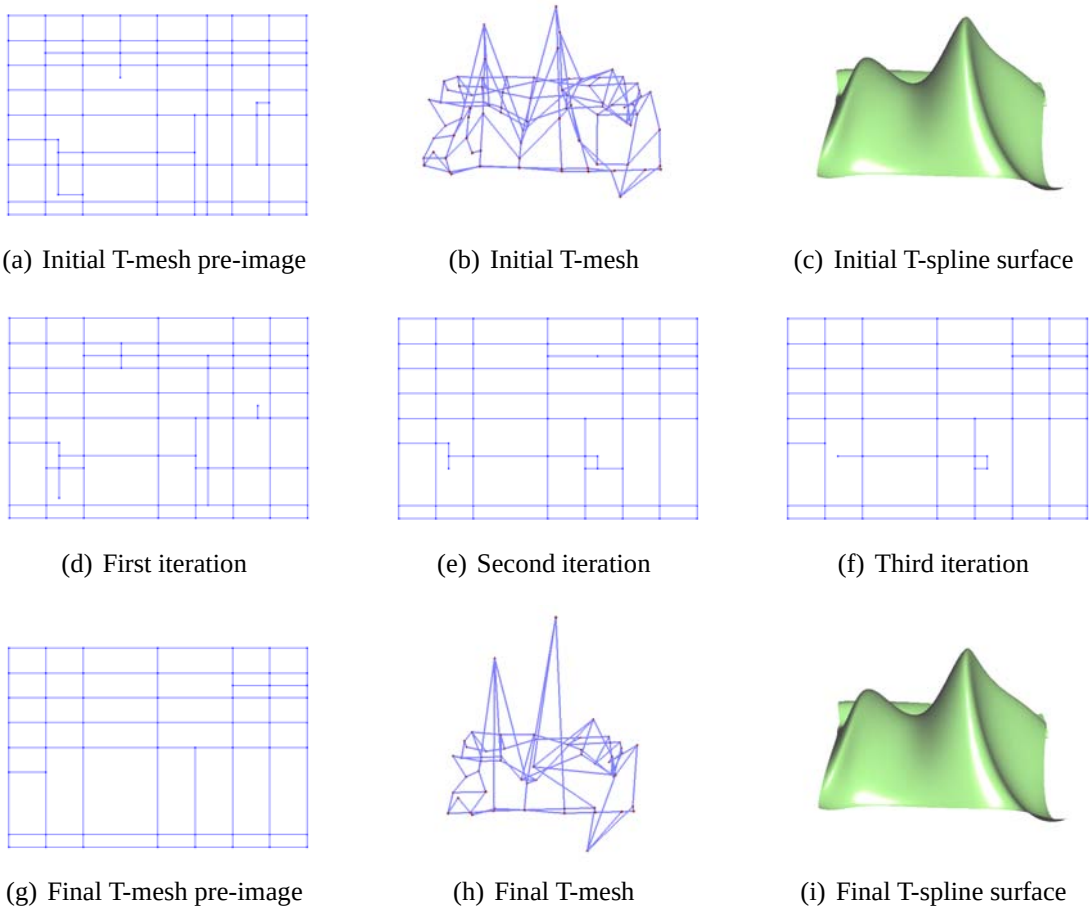
Consider a T-mesh (pre-image) shown in Figure 3.10(b), which is the result of inserting a point P_1 into a T-mesh whose pre-image is shown in Figure 3.10(a). Point P_2 is a control point automatically introduced by the knot insertion algorithm. Suppose no further geometrical change is made to these control points. Obviously, P_1 and P_2 are two redundant control points in the T-mesh and should be removable. However, if we apply the single control point removal algorithm to point P_2, it is *surprising* to find that P_2 cannot be removed from the T-mesh by carefully checking the removal algorithm!

Fortunately, in the above situation, point P_1 can be removed by the single control point removal algorithm, and furthermore after that, point P_2 becomes removable for the single control point removal algorithm (see Figure 3.10(c)).

The above example indicates that one control point may not be removed until some other control points are removed. This observation motivates the following removal strategy for removing as many control points in a T-mesh as possible:

1) Check each control point in the T-mesh. If it is removable, remove it and update the T-mesh.

2) If at least one control point has been removed, execute step 1) again.



(a) Initial T-mesh pre-image (b) Initial T-mesh (c) Initial T-spline surface

(d) First iteration (e) Second iteration (f) Third iteration

(g) Final T-mesh pre-image (h) Final T-mesh (i) Final T-spline surface

Figure 3.11: An example of removing many control points.

An example of removing many control points from a T-spline surface is provided here in Figure 3.11. Figure 3.11(a) to Figure 3.11(c) are the pre-image, the T-mesh, and the surface for the initial T-splines, respectively. The algorithm is then invoked to eliminate the control points. Figure 3.11(d) is the pre-image of the resulting T-mesh after we apply the single control point removal algorithm to all the points of the T-mesh once. We call this process one iteration. We continue this process to the new T-mesh. Figure 3.11(e) and Figure 3.11(f) are the pre-images of the T-mesh after the second and third iterations. It can be seen that the number of control points in the T-spline surface is gradually reduced. The final result is displayed in Figure 3.11(g) to Figure 3.11(i). During this removing process, there are totally 37 control points that are removed. Thus the T-mesh is simplified while the

T-spline surface remains the same.

3.5 Summary

This chapter investigates the problem of removing control points from a T-spline surface. The T-spline knot removal is found to be much more complicated than the B-spline knot removal, since the T-spline knot removal could lead to different result and sometimes the knot removal could cause the insertion of extra control points. A single control point removal algorithm is developed, which is in the style of the T-spline knot insertion algorithm [128]. The algorithm can be used to detect whether a user-specified control point can be removed or not. If the control point is found to be removable, the algorithm returns the new T-mesh with the control point removed. The algorithm can have applications in interactive editing.

The extension of the algorithm to remove more than one control point is also discussed and several approaches are suggested. In many situations, the control points that are added by knot insertion can be completely removed by these approaches. However, there still exist some situations, in which some inserted control points cannot be removed. Therefore developing algorithms that are able to remove all those control points added by knot insertion warrants further investigation.

Another direction for future work is to extend this algorithm to perform approximation knot removal. In this case, the residue does not have to go to zero and can be controlled by a given tolerance. Because usually in a complicated T-spline, it is unlikely to remove a lot of control points exactly. By approximation knot removal, it is possible to remove a lot of of control points, achieving the simplification purpose.

Chapter 4

Curvature-Guided Adaptive T-spline Surface Fitting

4.1 Introduction

This chapter studies the problem of fitting a T-spline surface to a triangular mesh that is topologically homeomorphic to a plane set. Triangular meshes and splines are currently two main representations for free-form surfaces. While triangular meshes are widely used in visualization, animation and interactive computer games, splines are the major format in CAD/CAM industry. Splines have parametric equations and have a more compact representation, facilitating many operations such as shape analysis and surface manipulation. There is a need to convert triangular meshes into spline surface. The conversion can be achieved by surface fitting. Since T-splines are a generalization of NURBS and the conversion from a T-spline surface to a NURBS surface is straightforward, our surface fitting result would be totally compatible to the current industry standards in CAD /CAM.

Basically, the problem is stated as follows: given a triangular mesh with a vertex set $D = \{\mathbf{d_1}, \mathbf{d_2}, \cdots, \mathbf{d_m}\}$, we look for a T-spline surface $\mathbf{S}(u, v)$ such that the distance between the

surface and each vertex $\mathbf{d_i}$ is below a given error tolerance ε

$$\mathrm{dist}(\mathbf{d_i}, \mathbf{S}(u,v)) \leq \varepsilon,\ i = 1, 2, \cdots, m \tag{4.1}$$

Many difficulties lie in this problem such as the enormous size of the data set and the complexity of the underlying shape. Our objectives are to achieve the required fitting precision, to produce visually smooth T-spline surfaces, to preserve geometric features of the input mesh and to output a concise and effective representation. Though some approaches existed for tackling this problem, most of them only achieved the first two objectives and usually resulted in surfaces with highly redundant representations. In this chapter, we propose a new fitting framework to accomplish these objectives. The framework is an iterative, adaptive process that is an integration of several components such as feature detection, parameterization, adaptive least squares surface approximation, initial surface structure replacement and mesh re-parameterization. The novelties in this framework include:

- We propose to use T-splines as the basic tool for surface approximation and to fit a T-spline surface to the triangular mesh using an iterative procedure, where both the topology and the geometry of T-spline surface are gradually improved. Due to the power of the T-spline local refinement, T-splines can easily achieve adaptiveness in that during each iteration step, the target surface is only refined locally at the areas where the approximation error is higher than the tolerance. As a result, we are able to obtain a more compact surface representation compared to those obtained by the traditional methods. In addition, we use standard or semi-standard T-splines for the fitting process and thus the computation cost is kept to be low.

- A good approximation should preserve the geometrical features of the model. This means that the mesh and the surface need to have high resemblance in the feature areas. Feature areas usually refer to the regions where intense geometrical changes take place and they are highly critical in that they reflect the major identities of a geomet-

ric model. In geometry processing, features have become a concern recently. Some approaches have been developed for detecting features on meshes [154, 107, 163]. Here we propose a scheme that examines the feature regions of the input triangular mesh and gives the feature regions more emphases in surface fitting so as to guarantee that the model is faithfully reconstructed. A feature sensitive method was also developed in [88] for surface fitting purpose. While the feature sensitive method considers features in the parameterization phase, we deal with features in the fitting process.

- Since mesh parameterization and initial surface structure placement have important influences on the final fitting surface, it is important to have a good parameterization and initial surface structure placement. However, due to limited prior knowledge about the reconstructed surface, it is difficult to find a very good parameterization and initial surface structure placement at the beginning of the fitting process. Therefore, in our framework, we propose a re-parameterization process and an initial surface structure re-placement process, which use the knowledge obtained during the fitting process to improve the parameterization and initial surface structure placement.

The rest of this chapter is organized as follows: Section 4.2 gives an overview of the algorithm. Each step or component of the algorithm is explained in Sections 4.3-4.10. The experimental results are provided in Section 4.11 and a summary is given in Section 4.12.

4.2 Overview of the algorithm

Given a triangular mesh and an error tolerance, an adaptive surface fitting generally works in the following steps: 1). Parameterize the triangular mesh so that each vertex of the mesh has a pair of parameter values; 2). Initialize the class of the surfaces, from which the optimal surface is searched for; 3). Establish the objective function for the optimization process, which usually combines the least squares distance and the fairness term with a fairness factor, and then solve the optimization problem for the optimal surface; 4). Check

the quality of the resulting surface. If the result is satisfactory, the process stops. Otherwise, proceed to the next step; and 5). Enlarge the class of the surfaces and go back to step 3).

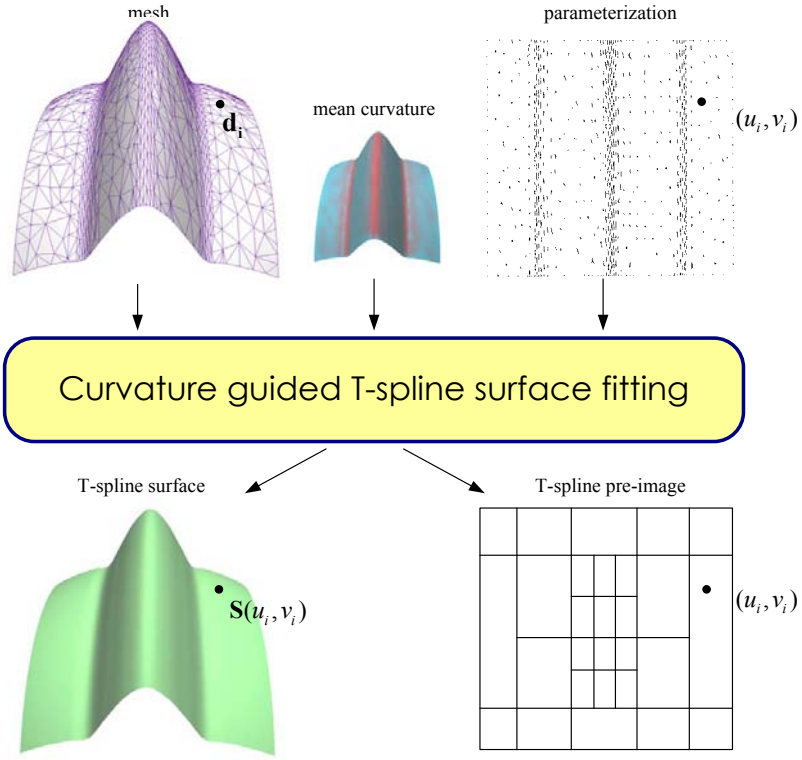


Figure 4.1: Illustration of the curvature guided T-spline surface fitting.

It can be seen that the above fitting process is actually an iterative process. In our case, we use T-splines to perform adaptive fitting. So some steps have to be customized. In step 2), to define the class of the T-spline surfaces, we have to specify the T-mesh topology (i.e., the T-mesh structure or the pre-image of the T-mesh) and the control point weights. This step is usually referred to as the initial T-spline structure placement. In step 3), the least squares objective function is eventually a function of the control point of the T-spline. To simplify computations, we avoid using rational surfaces and thus let the class of the T-spline surfaces be one that consists of polynomial T-splines. This can be implemented by initially choosing standard or semi-standard T-spline structures. In step 5), the enlarging of the class of the T-spline surfaces can be done using T-spline local refinement which produces a finer T-mesh with a new set of control point weights. In this way, we have had an adaptive T-

spline surface fitting method. It could produce a smooth T-spline surface that fits the input mesh within the given tolerance. However, this approach does not reflect the geometric features of the initial mesh. To overcome this deficiency, we propose to improve this method by incorporating features. We modify step 4) and let the features guide the quality checking such that in the feature areas a better approximation is needed. In this research, we use discrete mean curvatures to measure features. Thus we get our first version of adaptive T-spline surface fitting which we call *curvature guided adaptive T-spline surface fitting* (see Algorithm 4.1). An illustration of this approach is shown in Figure 4.1.

Algorithm 4.1 : Curvature guided adaptive T-spline surface fitting

Given: a triangular mesh $\mathbf{T}$, error tolerance ε, fairness factor σ, a parameterization ψ, a set of mean curvatures H

Goal: find a T-spline surface that is close to mesh $\mathbf{T}$ (within ε deviation)

1. Initialize the T-mesh topology and control point weights
2. **loop**
3. Compute an optimal T-spline surface under current settings
4. Check the T-spline surface for approximation error
5. **if** not all the regions on the surface pass the check **then**
6. Refine the T-mesh by adding more control points and update the control point weights
7. **else** {i.e., the surface is acceptable}
8. Exit from the loop
9. **end if**
10. **end loop**
11. **return** the final T-spline surface

In general, the curvature guided adaptive T-spline surface fitting is able to locate more T-spline control points to the areas that have more features and details. However, the performance of the algorithm could be further improved. Notice that both the mesh parameterization and the initial T-spline structure placement have an impact on the fitting result. In Algorithm 4.1, these two steps are done only once at the beginning of the fitting process. At that time, we do not have much knowledge about the reconstructed surface and thus it is difficult to give an optimal parameterization and initial T-spline structure placement. Based on this analysis, this chapter proposes a new framework for adaptive T-spline fitting.

Besides the curvature guidance, in this new framework we also have two new components which are the initial T-spline structure re-placement and the re-parameterization. These two new components are introduced to improve the initial parameterization and placement. They are carried out after Algorithm 4.1 has finished and we have got a reasonable fitting T-spline surface. This fitting T-spline surface is used to provide more knowledge for re-parameterization and re-placement and the surface itself will be improved next. So the new framework includes Algorithm 4.1 within a loop and the initial T-spline structure placement and the mesh parameterization are progressively updated. This results in the second version of our adaptive T-spline fitting which we call curvature guided adaptive T-spline fitting with progressively improved initial settings (see Algorithm 4.2).

Algorithm 4.2 : Curvature guided adaptive T-spline surface fitting with progressively improved initial settings

Given: a triangular mesh $\mathbf{T}$, error tolerance ε, fairness factor σ
Goal: find a T-spline surface that is close to mesh $\mathbf{T}$ (within ε deviation)

1. Compute mean curvature H for all vertices of the triangular mesh
2. Compute a parameterization ψ
3. Invoke Algorithm 4.1 and denote the resulting T-spline surface as S
4. **repeat**
5. $S_0 \Leftarrow S$
6. Re-parameterize the triangular mesh based on S_0, update ψ
7. Re-initialize the T-mesh topology based on S_0 and set the control point weights
8. Invoke Algorithm 4.1 and denote the resulting T-spline surface as S
9. **until** The approximation by S is not improved compared to the approximation by S_0
10. **return** the final T-spline surface

4.3 Feature detection

A free-form model can be rich of geometrical features, which are irregularly distributed over the model. These features include but are not limited to valleys, ridges, creases, corners, spines, and some fine details. Essentially, a feature area can be understood as the region on the model that undergoes more rapid geometrical change than its adjacent regions and hence that draws more attentions than the areas that do not contain features. For example,

on the Stegosaurus model shown in Figure 4.2, the parts of the its head, tail, legs and spikes on the back can be considered as features. We mark these features with red boxes.

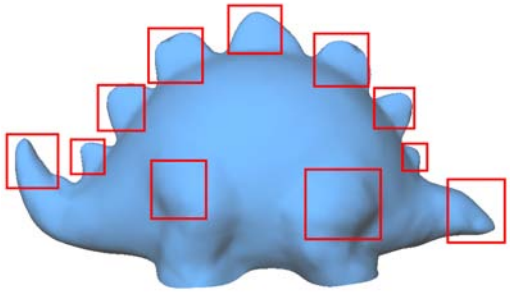


Figure 4.2: Feature areas on a triangular mesh.

For many tasks in geometry processing including surface fitting, the overall performance is to some extent evaluated by the fact that whether the features are properly handled. In order to take care of all the features on a model, it should be first figured out where these features are located.

The curvatures describe the amount by which a model deviates from a plane and is one of the geometrical measures that are closely related to surface features. Generally, a surface region that contains geometrical features would have high curvature values while the one that has a relatively smooth shape would have low curvature values.

At a point on a surface, an infinite number of curvatures in various directions can be defined. The mean curvature h at a given point is the average of the maximal curvature κ_1 and the minimal curvature κ_2 at that point, and can be expressed as $h = \frac{1}{2}(\kappa_1 + \kappa_2)$. Thus the mean curvature can be a good representative of all the curvatures.

Traditionally, curvature is the concept for smooth and differentiable surfaces. In recent years discrete differential geometry [13] was proposed for triangular meshes. In [32], Desbrun et al. derived a formula for computing the mean curvature of a mesh at vertex $\mathbf{d_i}$ using

the information of its 1-ring neighborhood:

$$h_i = \| \frac{1}{4A} \sum_{j=1}^{val_i} (\cot \alpha_j + \cot \beta_j)(\mathbf{d_j} - \mathbf{d_i}) \| \tag{4.2}$$

where A is the sum of the area of the triangle faces adjacent to $\mathbf{d_i}$, α_j, β_j are the two angles opposite to the edge connecting $\mathbf{d_i}$ and $\mathbf{d_j}$ as illustrated in Figure 4.3, and val_i is the valence of $\mathbf{d_i}$.

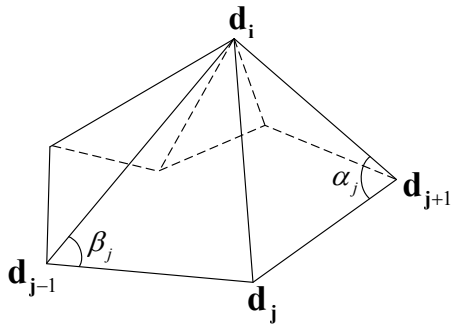


Figure 4.3: The 1-ring neighborhood of vertex $\mathbf{d_i}$.

It is noticed that the above method for computing the mean curvature is only applicable to a vertex that has the complete 1-ring neighbors. For the boundary vertices of an open triangular mesh, the method would not work. Our strategy for a boundary vertex is to simply let its mean curvature equal to that at any one of its one interior neighboring vertices.

For a given mesh, we compute the mean curvature for each vertex. Then based on the curvature values, we can detect feature areas. Figure 4.4 shows the color plot of mean curvatures of the Stegosaurus model, where red color represents high curvature values and blue color represents low curvature values. It can be seen that the feature areas of the Stegosaurus model are identified by having significantly higher curvature values than the non-feature areas.

It is then feasible to associate the features on a model with the corresponding curvature values.

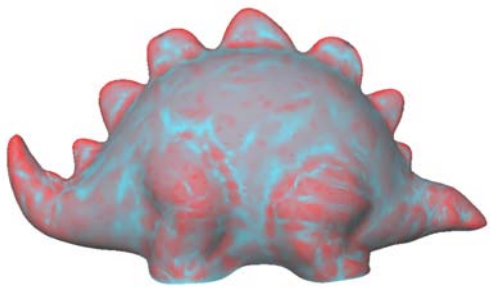


Figure 4.4: Visualization of the mean curvatures on a triangular mesh.

4.4 Parameter generation

To perform parametric surface fitting to a triangular mesh, a process called parameterization which associates each vertex in the triangular mesh with a pair of parameter values is usually needed, especially when the parametric distance is used. In particular, let $\mathbf{T}$ be a triangular mesh in R^3 space with a set of vertices $\{\mathbf{d_1}, \mathbf{d_2}, \cdots, \mathbf{d_m}\}$ and $\Omega \subset R^2$ be a planar domain. A parameterization $\psi : \Omega \rightarrow R^3$ is a bijective mapping from the domain to the surface of the triangular mesh. By the parameterization, each vertex $\mathbf{d_i}$ of the mesh has a pair of parameter values (u_i, v_i), which defines a point $\mathbf{u_i} = (u_i, v_i)$ in the parameter space. The set of points $\{\mathbf{u_1}, \mathbf{u_2}, \cdots, \mathbf{u_m}\}$ forms a planar triangulation $\mathbf{P}$ in the domain by maintaining the same connectivity as $\mathbf{T}$. $\mathbf{P}$ can be regarded as an embedding of $\mathbf{T}$ in Ω and the parameterization can be regarded as a process of flattening a triangular mesh onto a plane, as illustrated in Figure 4.5.

Parameterization is also an important processing in the area of digital geometry processing. Many methods have been developed [43, 44, 135]. Different applications may require different parameterization methods. Since parameterization is a mapping between spaces of different dimensions, generally distortions are inevitably incurred during the process. Therefore most parameterization methods adopt some optimization processes that minimize the distortion in certain criteria which are based on various geometrical proper-

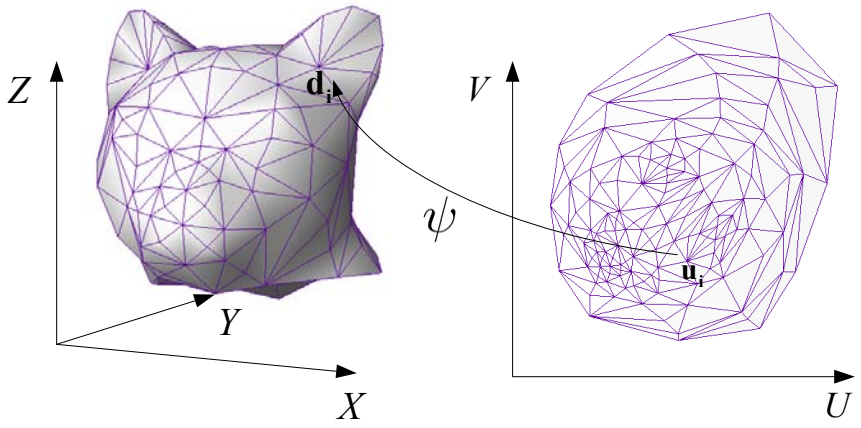


Figure 4.5: Flatten a triangular mesh from R^3 to R^2.

ties. These geometrical properties include not only vertex information, but edge, angle or face information as well. For example, some approaches try to preserve the area of triangle faces after they are mapped to the parameter domain [30, 31] and some approaches are angle-preserving [42, 45, 134] or edge length-preserving [101, 124]. There also exist approaches that consider the optimization of two or more criteria at the same time [22]. In addition, no matter which criterion is chosen, one basic requirement for almost all the parameterization methods is that in the parameter domain the triangulation should not self intersect and no triangle face is degenerated (i.e., all the three vertices of a face should not lie on a same line).

In the rest of this section, we describe a parameterization method that is basically a variant of Floater's mean value coordinates based approach [42] and is expected to provide a good parameterization for the triangular mesh. The obtained parameterization will be used in T-spline surface fitting. The quality of the parameterization result substantially influences surface fitting.

For any triangular mesh that is of an open disk topology, a closed boundary which is comprised of a group of boundary edges can be identified. A boundary edge is characterized by belonging to only one face. The vertices on the boundary are referred to as the boundary vertices and the remaining vertices in the mesh are called interior vertices. Without loss

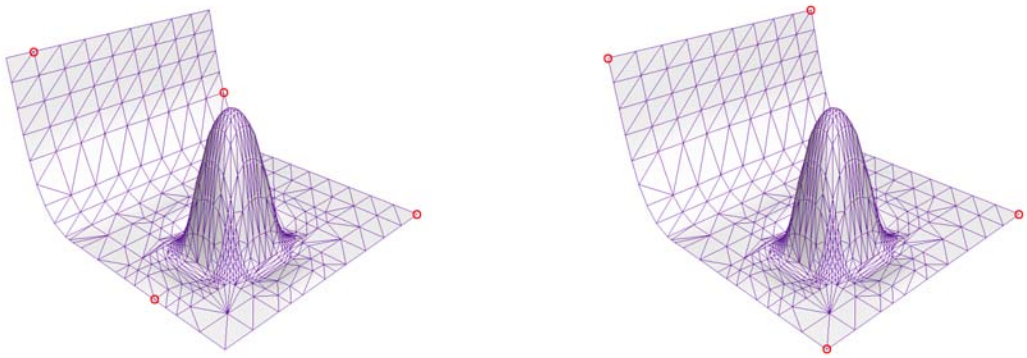
of generality, in the vertex set $\{\mathbf{d_1}, \mathbf{d_2}, \cdots, \mathbf{d_m}\}$ we assume the subset $\{\mathbf{d_1}, \mathbf{d_2}, \cdots, \mathbf{d_l}\}$ are the interior vertices and the subset $\{\mathbf{d_{l+1}}, \cdots, \mathbf{d_m}\}$ are the boundary vertices. We also assume that these boundary vertices are ordered clockwise.

With parameterization, the boundary vertices are mapped onto the boundary of the parameter domain and the interior vertices are mapped onto the interior region of the parameter domain. There are a few options for the shape of the parameter domain. It could be a rectangle, a circle, or a free form polygon. A rectanglar parameter domain would be preferable in our application because that is also the parameter domain of the T-spline surface we are going to construct. More specifically, we define the parameter domain to be a unit square, e.g. $[0, 1] \times [0, 1]$. This setting helps to avoid any extra trimming operation for the fitting surface.

Usually parameterizing an open triangular mesh is performed in two steps. First, the parameters for the boundary vertices are decided; after that, the parameters for the interior vertices are then calculated. In the following, we describe these two steps in detail.

4.4.1 Computing parameters for boundary vertices

It is understandable that the parameters of the boundary vertices influence the result of the overall parameterization. Cursory assignment of the boundary vertex parameters would lead to a big overall distortion. Therefore here we discuss how to devise a reasonable parameterization for the mesh boundary. Since the rectanglar parameter domain is chosen in our case, four corner vertices $\mathbf{d_{b1}}$, $\mathbf{d_{b2}}$, $\mathbf{d_{b3}}$ and $\mathbf{d_{b4}}$, where $b1, b2, b3, b4 \in [l + 1, m]$, should be first designated from the set of the boundary vertices. They are associated with parameters $(0, 0)$, $(0, 1)$, $(1, 1)$ and $(1, 0)$, respectively. Next, the other boundary vertices can be parameterized by approaches that take the geometrical properties of the boundary edges into account. For example, for a boundary vertex $\mathbf{d_j}$ that is between the two corners $\mathbf{d_{b1}}$ and $\mathbf{d_{b2}}$, its parameters are (u_j, v_j), where $u_j = 0$ and v_j is decided using the chordal



(a) (b)

Figure 4.6: Selection of the corner vertices based on different strategies.

length approach:

$$v_i = \frac{\sum_{i=b1}^{j-1} \|\mathbf{d_{i+1}} - \mathbf{d_i}\|}{\sum_{i=b1}^{b2-1} \|\mathbf{d_{i+1}} - \mathbf{d_i}\|} \tag{4.3}$$

Thus the remaining problem is how to carefully pick the four corner vertices so that a good result can be generated. One possible approach is based on the idea that the accumulated edge length between any two adjacent corner vertices should approximately be the same. Let $L_{sum} = \sum_{i=l+1}^{m} \|\mathbf{d_{i+1}} - \mathbf{d_i}\|$ denote the sum of all the lengths of the boundary edges. First, $b1$ is arbitrary selected to be a number that is the subscript of a boundary vertex. For the simplicity of the presentation, we let $b1 = l + 1$. Next, $b2$ is chosen to be the number which satisfies

$$\sum_{i=b1}^{b2-1} \|\mathbf{d_{i+1}} - \mathbf{d_i}\| \approx \frac{1}{4} L_{sum} \tag{4.4}$$

Similarly, $b3$ and $b4$ can also decided in such a way. Although it is widely adopted, sometimes the corner vertices picked by this approach can be undesirable (see the red circles in Figure 4.6(a)).

The second approach is to pick the corner vertices by considering the geometrical feature of the boundary. In fact, many models such as the one shown in Figure 4.6(a) contain some vertices on the boundary that naturally serve as corners. Therefore, we may make the selections based on the angle $\angle \mathbf{d_{i-1}} \mathbf{d_i} \mathbf{d_{i+1}}$ for each boundary vertex $\mathbf{d_i}$ which is formed by

the vertex and its two adjacent boundary vertices. Usually, a vertex with a sharper angle is more suitable to be the corner vertex. Also, in order to reduce the influence caused by the noises in the triangular mesh, the mesh boundary is smoothed first using the following filter several times before the angles are measured:

$$\mathbf{d_i} \leftarrow \frac{1}{6}\mathbf{d_{i-1}} + \frac{2}{3}\mathbf{d_i} + \frac{1}{6}\mathbf{d_{i+1}} \tag{4.5}$$

Finally, we choose the four vertices on the smoothed boundary that have the smallest angles as corners. The corners picked using this approach are highlighted in Figure 4.6(b). This time the result is more desirable.

In practice, both approaches can be attempted and the one that leads to better result can be adopted. Besides, user intervention could also be considered as one option here.

4.4.2 Computing parameters for interior vertices

Denote the index set of all the 1-ring neighboring vertices of $\mathbf{d_i}$ by N_i. For the parameter $\mathbf{u_i}$ of an interior vertex $\mathbf{d_i}$, we let it be a convex combination of the parameters of its neighboring vertices, which can be formulated as:

$$\mathbf{u_i} = \sum_{j \in N_i} \lambda_{ij} \mathbf{u_j}, \quad i = 1, 2, \cdots, l \tag{4.6}$$

where λ_{ij} are some non-negative weights. The choice of these weights would have considerable influence on the parameterization result.

There are l such equations in total, one for each interior vertex. Some of the equations do not involve any boundary vertex and some involve one or more boundary vertices. After reorganizing the above equations by placing the boundary vertices and the interior vertices

on the different sides, we have

$$\mathbf{u_i} + \sum_{j \in N_i, j \leq l} -\lambda_{ij}\mathbf{u_j} = \sum_{j \in N_i, j > l} \lambda_{ij}\mathbf{u_j}, \quad i = 1, 2, \cdots, l \tag{4.7}$$

Combining all these l equation arrives at a linear system:

$$AU = U_0 \tag{4.8}$$

where $U = (\mathbf{u_1}, \cdots, \mathbf{u_l})^T$ is a $1 \times l$ vector of the unknown interior vertices and $U_0 = (\mathbf{u_{0,1}}, \cdots, \mathbf{u_{0,l}})^T$ is a $1 \times l$ constant vector with $\mathbf{u_{0,i}} = \sum_{j \in N_i, j > l} \lambda_{ij}\mathbf{u_j}$. A is an $l \times l$ coefficient matrix consisting of

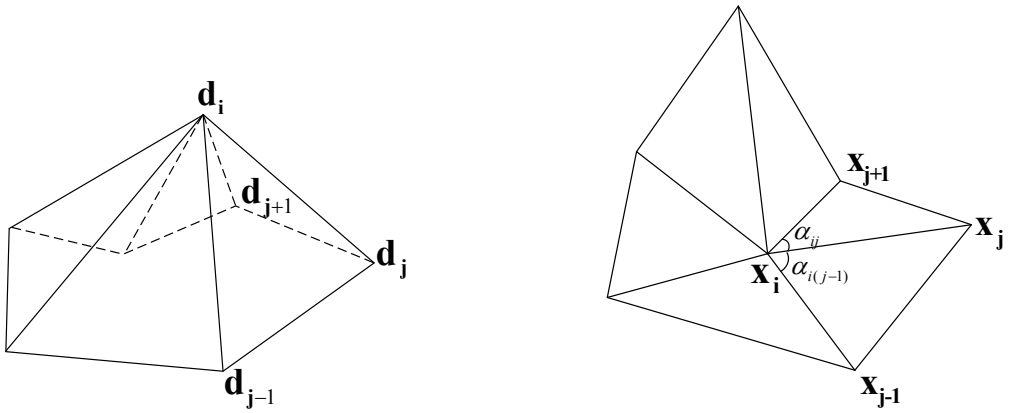
$$a_{ij} = \begin{cases} 1, & i = j \\ -\lambda_{ij}, & i \neq j \ and \ j \in N_i \\ 0, & otherwise \end{cases} \tag{4.9}$$

By solving the linear system, the parameters $\mathbf{u_1}, \mathbf{u_2}, \cdots, \mathbf{u_l}$ for the interior vertices can then be determined and the whole triangular mesh is parameterized.

However, whether the linear system (4.8) has a unique and proper solution depends on certain constraints. Consider the following three conditions:

1. $\lambda_{ij} > 0$, for any i, j;
2. $\sum_{j \in N_i} \lambda_{ij} = 1$, for $i = 1 \cdots l$;
3. $\sum_{j \in N_i} \lambda_{ij}\mathbf{d_j} = \mathbf{d_i}$, for $i = 1 \cdots l$.

If all of these conditions are met, it is guaranteed that the linear system (4.8) is solvable. Furthermore, there is no self-intersecting edges in $\mathbf{P}$, provided that the boundary is mapped to a convex polygon. Also, it has the property that $\mathbf{P}$ would be identical to $\mathbf{T}$ if $\mathbf{T}$ is a planar triangular mesh itself.



(a) The 1-ring neighborhood of vertex $\mathbf{d_i}$

(b) The corresponding planar star-shaped region

Figure 4.7: Compute the mean value coordinates for $\mathbf{d_i}$.

To choose values for λ_{ij} to satisfy the above three conditions, we adopt the mean value coordinates. The mean value coordinates (MVC), proposed by Floater [42], is a set of weights that can be used to represent a vertex in a 2D triangular mesh by convexly combining its neighboring vertices. Mean value coordinates is appreciated for depending not only continuously but also smoothly on the vertices of the mesh. Therefore, it preserves some intrinsic geometric properties of a triangular mesh and has applications in many geometry processes including parametrization.

Suppose $\mathbf{d_i}$ is an interior vertex in the triangular mesh with val_i neighboring vertices (see Figure 4.7(a)). Without loss of generality, suppose the neighboring vertices of $\mathbf{d_i}$ are $\mathbf{d_1}, \mathbf{d_2}, \cdots, \mathbf{d_{val_i}}$. To compute the mean value coordinates λ_{ij} for vertex $\mathbf{d_i}$ with respect to its neighboring vertex $\mathbf{d_j}$ ($1 \leq j \leq val_i$), a planar star-shaped region centering at $\mathbf{x_i}$ is constructed by flattening the 1-ring neighborhood of vertex $\mathbf{d_i}$ into a plane, as shown in Figure 4.7(b). In this flattened star-shaped region, the edges that connect $\mathbf{x_i}$ and its neighbors remain the original length, i.e., $\|\mathbf{x_j} - \mathbf{x_i}\| = \|\mathbf{d_j} - \mathbf{d_i}\|$. The angles α_{ij} between two adjacent such edges are uniformly scaled so that the sum of these scaled angles equals

2π:

$$\alpha_{ij} = \frac{\angle \mathbf{d_j d_i d_{j+1}}}{\sum_{k=1}^{val_i} \angle \mathbf{d_k d_i d_{k+1}}} 2\pi \tag{4.10}$$

Then we compute the initial weights w_{ij} from these geometrical quantities as:

$$w_{ij} = \frac{\tan(\frac{\alpha_{ij}}{2}) + \tan(\frac{\alpha_{i(j-1)}}{2})}{\|\mathbf{x_j} - \mathbf{x_i}\|} \tag{4.11}$$

After that, the mean value coordinates λ_{ij} can be calculated by normalizing w_{ij} using the sum $\sum_k w_{ik}$:

$$\lambda_{ij} = \frac{w_{ij}}{\sum_{k=1}^{val_i} w_{ik}} \tag{4.12}$$

It is apparent that $\lambda_{ij} > 0$ and $\sum_k \lambda_{ik} = 1$. Also, it has been proved in [42] that $\sum_k \lambda_{ik} \mathbf{d_k} = \mathbf{d_i}$.

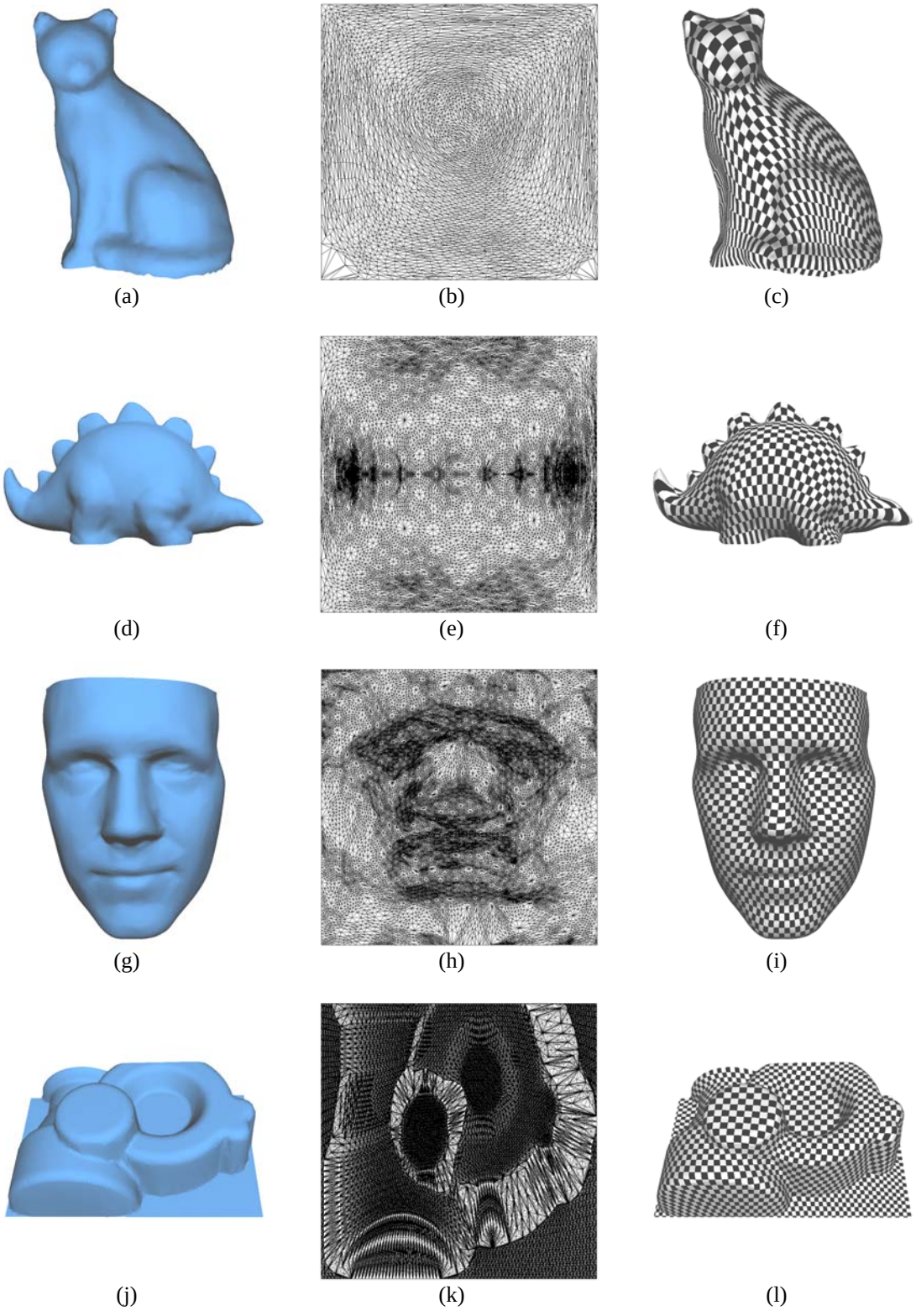
Figure 4.8 shows parameterization using this approach for some triangular mesh models. The models are shown in the first column of the figure. In the second column, the parameterization results onto the unit square domain are shown. For the models on which the natural corners can be identified, we can always correctly pick those vertices to be the corners of the parameter domain. In the third column, the results are further illustrated by mapping the checkboard textures onto the triangular meshes. It can be seen that the checkboard patterns are in general evenly distributed over the models, which indicates the good quality of the parameterization.

4.5 Initial T-spline structure placement

To set up the initial T-spline structure, we have to specify the parameter domain, the T-mesh topology (i.e., structure), and the weights for points on the T-mesh.

The parameter domain of the T-spline surface to be reconstructed is set to be the domain

4.5. Initial T-spline structure placement

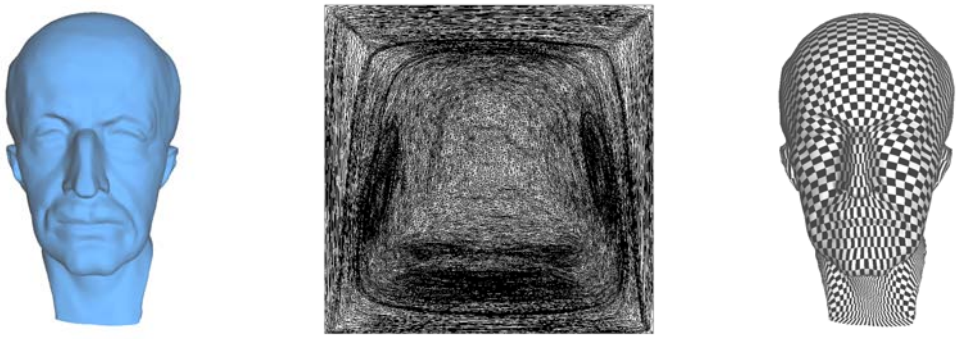


(a) (b) (c)

(d) (e) (f)

(g) (h) (i)

(j) (k) (l)



(m) (n) (o)

Figure 4.8: Parameterization of some triangular meshes using MVC.

that the triangular mesh is parameterized to, which is $[0,1] \times [0,1]$. Since our fitting is an adaptive approach, we begin with a relatively simple T-mesh structure for simplicity. For example, we can choose the initial surface to be a bicubic B-spline patch with 4×4 control points. The knot vectors along the $u-$ and $v-$ directions are both set to be $\{-2\mu, -\mu, 0, 1, 1+\mu, 1+2\mu\}$, where μ is a small positive value (for example, 0.001). In the parameter space the lines corresponding to these knots (except for the first and the last one) form the pre-image of the control grid of the surface. Let us denote this structure (or pre-image) by M. Figure 4.9 shows this initial pre-image. In the figure, the domain of the surface is highlighted in yellow. The dash lines are phantom edges that provide necessary information for the surface definition.

We also let all the control point weights be one. Denote by W the set of all weights corresponding to the control points. Then M and W define a class of T-splines that have the same pre-image M and the same weights W. We denote this class by $S_{M,W}$. Note that each surface in $S_{M,W}$ is a standard T-spline no matter where the control points are geometrically located.

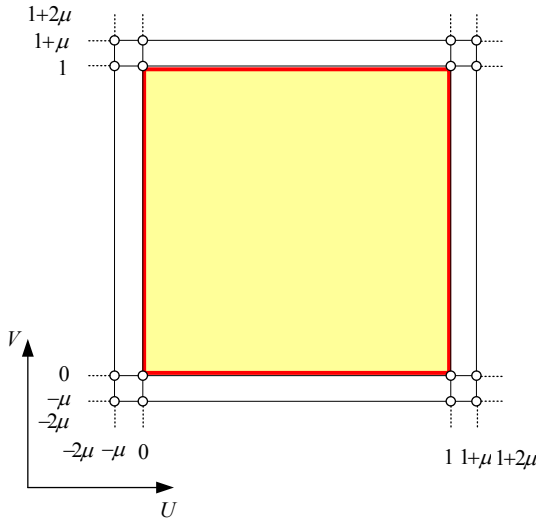


Figure 4.9: The pre-image of an initial T-mesh.

4.6 Least squares T-spline surface approximation

Once the T-mesh topology (i.e., the T-spline pre-image M) and the set of control point weights W are given, they define a class $S_{M,W}$ of T-spline surfaces. The pre-image M and the weight set W are the ones either set in the initialization or produced by the local refinement of the initial M and W. In this class, all the T-spline surfaces have the same T-mesh topology and weights; only their geometry is undecided. Here we seek to find the geometrically optimal T-spline surface in $S_{M,W}$, which best fits the triangular mesh under a certain criterion. Note that any T-spline surface in $S_{M,W}$ is standard or semi-standard due to our special initial T-spline structure placement and the use of T-spline local knot insertion for T-spline structure refinement (see Section 4.8). Therefore the optimal surface we intend to find is actually a polynomial surface, which can be written as follows:

$$\mathbf{S}(u,v) = \sum_{i=1}^{n} w_i P_i B_i(u,v) \tag{4.13}$$

We thus try to find a surface $\mathbf{S}(u,v)$ in the class $S_{M,W}$ that makes the sum of the squared parametric distances between the vertices of the mesh and the surface be minimized. This

is a least squares problem. In general, to generate a fair surface, we also include a fairness functional $J_{fair}(f)$ into the approximation. We choose the thin plate energy as the fairness functional. The details will be described in Section 4.6.1. The optimization problem hence becomes to minimize the following objective function:

$$F(P_1, P_2, \cdots, P_n) = \sum_{j=1}^{m} \parallel \mathbf{S}(u_j, v_j) - \mathbf{d_j} \parallel^2 + \sigma J_{fair}(S) \tag{4.14}$$

where σ is a constant that balances the approximation accuracy and fairness. A larger σ will lead to a smoother surface and a smaller σ will give a more accurate approximation. Note that the objective function is quadratic in control points. To solve the optimization problem, we differentiate the objective function with respect to each control point P_g and let the partial derivative equal zero: $\dfrac{\partial F}{\partial P_g} = 0$. This leads to

$$\sum_{i=1}^{n} w_i \left(\sum_{j=1}^{m} B_i(u_j, v_j) B_g(u_j, v_j) + \sigma m_{ig} \right) P_i = \sum_{j=1}^{m} \mathbf{d_j} B_g(u_j, v_j) \tag{4.15}$$

for $g = 1, \cdots, n$, where m_{ig} come from the fairness functional and their computation is given in Section 4.6.1.

If we introduce a $n \times n$ matrix $A = (a_{gi})$, two $n \times 1$ vectors $B = (b_g)$ and $P = (P_g)$ with

$$a_{gi} = w_i (\sum_{j=1}^{m} B_i(u_j, v_j) B_g(u_j, v_j) + \sigma m_{ig}) \tag{4.16}$$

and

$$b_g = \sum_{j=1}^{m} \mathbf{d_j} B_g(u_j, v_j) \tag{4.17}$$

then the above linear equations can be written as

$$A \cdot P = B \tag{4.18}$$

The solution $P = A^{-1} \cdot B$ of the linear system gives the control points that define the T-spline surface.

The linear system can be solved by standard numerical methods such as the Gaussian elimination if the system is relatively small. However, when the linear system is huge and furthermore the system has a sparse coefficient matrix, simple methods like the Gaussian elimination would be slow and even numerically unstable. Therefore alternative linear system solvers should be used, for example, the preconditioned complex bi-conjugate gradient (PCBCG) solver [115], the bi-conjugate gradient stabilized (Bi-CGSTAB) solver [148] or the sparse direct linear solvers for which an implementation is available in the open source library TAUCS [145]. In general these methods take much less computational time for large scale sparse linear systems and also have stable behaviors.

4.6.1 Fairness functionals

Fairness is an important factor in surface approximation. In order for the reconstructed surface shape to be fair, certain fairness functionals should be incorporated into the optimization. A *fairness functional* $J_{fair}(\mathbf{S})$ is a scalar-valued function of surfaces whose value is the measure of fairness. Quite often, $J_{fair}(\mathbf{S})$ is interpreted in a physical or geometric way, e.g., describing the bending energy or the total curvature contained by a surface. Different types of fairness functionals have been used in prior work [104, 158, 66, 65, 56, 55, 54]. What is in common is that almost all of these fairness functions involve integrating the partial derivatives of the surface and the smaller the value of the fairness functional is, the better the surface shape would be.

In our T-spline surface fitting, we choose the simple thin plate energy as the fairness functional. Physically, thin plate energy represents the bending energy stored in a thin sheet of metal that is being deformed to the shape of the surface. The simple thin plate

energy is formulated as:

$$J_{fair}(\mathbf{S}) = \iint_{\Omega} (\mathbf{S}_{uu}^2(u,v) + 2\mathbf{S}_{uv}^2(u,v) + \mathbf{S}_{vv}^2(u,v))dudv \tag{4.19}$$

The reason why we choose the simple thin plate energy is that it is a good approximation to the measure of fairness of the surface in most cases and that it is quadratic in the T-spline control points and thus can be efficiently computed.

Next, we describe how to compute $J_{fair}(\mathbf{S})$ of (4.19). While it is possible to compute $J_{fair}(\mathbf{S})$ exactly, the computation is rather complicated. An alternative way of computing $J_{fair}(\mathbf{S})$ is to approximate it by sampling the parameter domain Ω with a $step_u \times step_v$ rectanglar grid, which leads to:

$$J'_{fair}(\mathbf{S}) = \sum_{a=1}^{step_u} \sum_{b=1}^{step_v} (\mathbf{S}_{uu}^2(u_a,v_b) + 2\mathbf{S}_{uv}^2(u_a,v_b) + \mathbf{S}_{vv}^2(u_a,v_b))\Delta u_a \Delta v_b \tag{4.20}$$

where u_a and v_b are the sampled knot values at the a-th and b-th steps in u and v direction, respectively, and Δu_a and Δv_b are the corresponding sampling intervals. However, this approach has the drawback that to achieve a reasonable approximation accuracy, we need the sampling steps $step_u$ and $step_v$ to be sufficiently small. This could severely increase the computation time and lower down the overall efficiency of the approach especially when the surface is a T-spline with many locally refined regions. On the other hand, the sampling method may fail to reflect the considerable amount of energies that are contained at the locations between the sampling points and thus may lead to loss of accuracy, if $step_u$ and $step_v$ are big.

To avoid the above problem, we present a discrete method to compute the fairness func-

tional $J_{fair}(\mathbf{S})$ for a T-spline surface. For a polynomial T-spline surface of (4.13), we have

$$\begin{aligned} \mathbf{S}_{uu}(u,v) &= \sum_{i=1}^{n} w_i P_i \frac{\partial^2 B_i(u,v)}{\partial u^2} \\ \mathbf{S}_{uv}(u,v) &= \sum_{i=1}^{n} w_i P_i \frac{\partial^2 B_i(u,v)}{\partial u \partial v} \\ \mathbf{S}_{vv}(u,v) &= \sum_{i=1}^{n} w_i P_i \frac{\partial^2 B_i(u,v)}{\partial v^2} \end{aligned} \tag{4.21}$$

Thus the simple thin plate energy $J_{fair}(\mathbf{S})$ of the T-spline surface can be written

$$J_{fair}(\mathbf{S}) = \begin{bmatrix} P_1 & \cdots & P_i & \cdots & P_n \end{bmatrix} \begin{bmatrix} w_1^2 m_{11} & \cdots & w_1 w_j m_{1j} & \cdots & w_1 w_n m_{1n} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ w_i w_1 m_{k1} & \cdots & w_i w_j m_{ij} & \cdots & w_i w_n m_{in} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ w_n w_1 m_{n1} & \cdots & w_n w_j m_{nj} & \cdots & w_n^2 m_{nn} \end{bmatrix} \begin{bmatrix} P_1 \\ \vdots \\ P_j \\ \vdots \\ P_n \end{bmatrix} \tag{4.22}$$

where

$$m_{ij} = \iint_{\Omega} \left(\frac{\partial^2 B_i(u,v)}{\partial u^2} \frac{\partial^2 B_j(u,v)}{\partial u^2} + 2 \frac{\partial^2 B_i(u,v)}{\partial u \partial v} \frac{\partial^2 B_j(u,v)}{\partial u \partial v} + \frac{\partial^2 B_i(u,v)}{\partial v^2} \frac{\partial^2 B_j(u,v)}{\partial v^2} \right) du dv \tag{4.23}$$

From Equation (4.23), it can be seen that the value for m_{ij} depends on the two B-spline basis functions $B_i(u,v)$ and $B_j(u,v)$. Assume the two knot quintuples $B_i(u,v)$ is associated with are $\mathbf{u}_i = [u_{i0}, u_{i1}, u_{i2}, u_{i3}, u_{i4}]$ and $\mathbf{v}_i = [v_{i0}, v_{i1}, v_{i2}, v_{i3}, v_{i4}]$ where the knots in both $\mathbf{u}_i$ and $\mathbf{v}_i$ are strictly increasing, and the two knot quintuples that $B_j(u,v)$ is associated with are $\mathbf{u}_j$ and $\mathbf{v}_j$. According to the compact support property of B-spline basis functions, we have

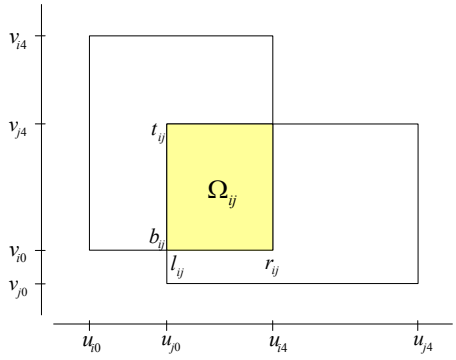
$$B_i(u,v) \begin{cases} > 0 & u_{i0} < u < u_{i4} \; and \; v_{i0} < v < v_{i4} \\ = 0 & otherwise. \end{cases} \tag{4.24}$$

Therefore, Equation (4.23) can be rewritten as

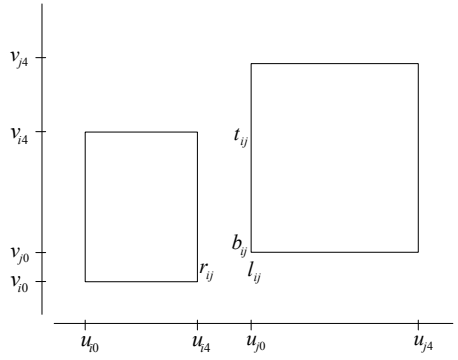
$$m_{ij} = \iint_{\Omega_{ij}} \left(\frac{\partial^2 B_i(u,v)}{\partial u^2}\frac{\partial^2 B_j(u,v)}{\partial u^2} + 2\frac{\partial^2 B_i(u,v)}{\partial u \partial v}\frac{\partial^2 B_j(u,v)}{\partial u \partial v} + \frac{\partial^2 B_i(u,v)}{\partial v^2}\frac{\partial^2 B_j(u,v)}{\partial v^2}\right) du dv \qquad (4.25)$$

where Ω_{ij} is a rectanglar region $[l_{ij}, r_{ij}] \times [b_{ij}, t_{ij}]$ which is the intersection of the nonzero regions of $B_i(u,v)$ and $B_j(u,v)$, as shown in Figure 4.10(a). l_{ij}, r_{ij}, b_{ij} and t_{ij} are computed from $\mathbf{u}_i$, $\mathbf{v}_i$, $\mathbf{u}_j$ and $\mathbf{v}_j$:

$$\begin{aligned} l_{ij} &= \max(u_{i0}, u_{j0}) \\ r_{ij} &= \min(u_{i4}, u_{j4}) \\ b_{ij} &= \max(v_{i0}, v_{j0}) \\ t_{ij} &= \min(v_{i4}, v_{j4}) \end{aligned} \qquad (4.26)$$



(a) The nonzero support of $B_i(u,v)$ and $B_j(u,v)$ are overlapping



(b) The nonzero support of $B_i(u,v)$ and $B_j(u,v)$ are separate

Figure 4.10: Domain for the integral to compute m_{ij}.

If $l_{ij} > r_{ij}$ or $b_{ij} > t_{ij}$, there is no intersection between the nonzero support of $B_i(u,v)$ and $B_j(u,v)$ (Figure 4.10(b)). In that case, $m_{ij} = 0$. Otherwise, we approximately compute m_{ij} by sampling Ω_{ij} with a $a \times b$ grid. To our experience, a value of 5 to 10 is enough for a and b, depending on the size of Ω_{ij}. Thus, rather than uniformly sampling the whole parameter domain Ω, each m_{ij} is computed by adaptively sampling in its own nonzero domain Ω_{ij}. Moreover, due to the symmetry of m_{ij}, we have $m_{ij} = m_{ji}$. Therefore, we only have to compute $\frac{n(n+1)}{2}$ different m_{ij}.

4.7 Curvature guided surface quality check

Once the surface is computed, it should be checked whether the result is satisfactory. A simple way of quality checking is to test whether the parametric distances of each vertex of the triangular mesh to the T-spline surface are below a given tolerance ε. However, using a single (or global) error tolerance for all vertices has drawbacks. It is not easy to choose a suitable global error tolerance for a model with a lot of geometrical features. For the task of surface fitting, it is important for the resulting T-spline surface to include all the features of the input mesh. Features actually reflect details on the model, which may be appropriately approximated only when the error tolerance is considerably low. For non-feature areas, a moderate error tolerance would be sufficient in order to get satisfactory fitting result. So if we choose a small ε to capture features, the surface is likely to end up with a number of unnecessary points in the non-feature areas even if the adaptive approach is used, which is apparently inefficient for the representation. On the other hand, if a big ε is chosen, though the overall geometry of the original mesh can be reproduced, some surface features are simply missed out.

The above discussion, therefore, suggests that different tolerances should be adopted at different areas on the surface. Because it is more difficult to achieve good approximation in feature areas than in non-feature areas, we should pay more attentions to the feature areas. Thus for a feature area, a high fitting precision and accordingly a low error tolerance would be required in order to guarantee faithful reconstruction. On the other hand, for a non-feature area, a moderate fitting precision and accordingly a moderate error tolerance would be enough. In Section 4.3, we identify the features of a model by mean curvature values. Thus here we let mean curvatures serve as a guidance for the error tolerance and introduce curvature-guided individual error tolerance ε_i for vertex $\mathbf{d_i}$:

$$\varepsilon_i = k_i \varepsilon \tag{4.27}$$

where k_i is a *curvature guidance factor* determined by the discrete mean curvature at vertex $\mathbf{d_i}$:

$$k_i = \max(\frac{\bar{h}_{max} - \bar{h}_i}{\bar{h}_{max} - \bar{h}_{min}}, \eta) \tag{4.28}$$

Here $\bar{h}_i = \log(h_i + 1)$ is the logarithm of the mean curvature h_i, the purpose of which is to concentrate the curvature values. $\bar{h}_{max}$ and $\bar{h}_{min}$ are the maximum and minimum values among all the $\bar{h}_j$ ($j = 1, 2, \cdots, m$) which are used to normalize the mean curvature values to the interval of $[0, 1]$. A small number η is used as a threshold to filter the curvatures values that are too big. In our experiment, we choose $\eta = 0.05$. By this adjustment, we would have a smaller tolerance ε_i for the vertex where the mean curvature is high.

So now whether a vertex $\mathbf{d_i}$ passes the test depends on whether $\| \mathbf{S}(u_i, v_i) - \mathbf{d_i} \| \leq \varepsilon_i$. When all the vertices of the mesh pass the test, the T-spline surface is considered acceptable. Otherwise, we have to proceed another iteration. Figure 4.11 shows the result of a checking process. In the figure, the red points represent the vertices that are not passed. It can be seen that most vertices that are not passed are concentrated at the regions with features.

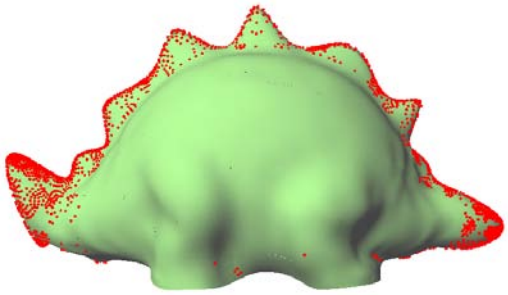


Figure 4.11: Visualization of the vertices that do not pass the checking.

After the above checking process, we also perform other checking for adjusting some parameters we use in the algorithm. We compute the maximum distance between the mesh and the T-spline surface and compare it against the one obtained in the previous iteration. In some situations, though the objective function gets a smaller value, the maximum distance

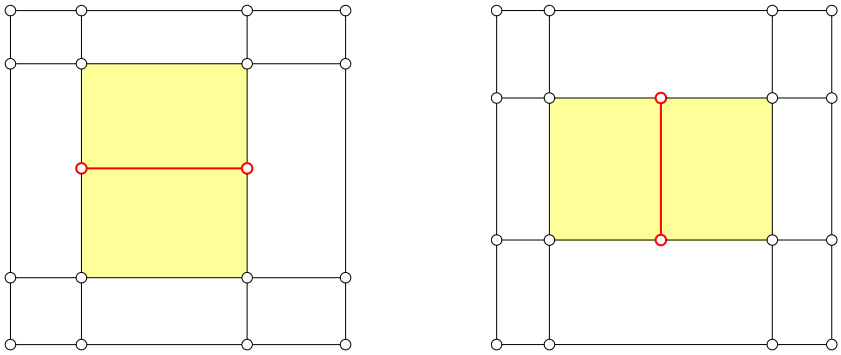
does not decrease substantially due to the complicated distribution of data. If the value is not decreased by a certain extent, we adjust the fairness factor σ by tuning down in order to increase the influence of fitting accuracy in the objective function. Besides, if the maximum distance between the mesh and the surface is already below the global error tolerance ε, that means the general goal for surface fitting is reached. Then, the threshold η for the curvature guidance factors might be slightly raised to 0.2 so as to speed up the fitting process.

4.8 T-spline structure refinement

So far we have described how to find the best approximate T-spline surface from a class of surfaces $S_{M,W}$ by geometrically optimizing the control points of the surface and how to check the quality of the obtained surface. However, if the T-mesh structure of the current T-spline surface class is not appropriate, even the best surface in the class could not well represent the shape of the input triangular mesh. Therefore, here we describe how to improve the T-mesh structure when the preset error tolerance is not met, by adding more control points or edges into the current T-mesh structure.

Generally, we refine the structure of the T-mesh at the offending regions which contain points that fail the previous checking. These points are called the violating points and the regions are called the offending regions. The violating points could be inside a region or on the border of a region. For offending regions, our strategy is to split them. While there are different ways to split a region, we follow the principle that an offending region should, in general, be split in half in the direction where the region has a larger knot difference. Specifically, assume a region is a rectanglar box whose lower-left corner has coordinates (u_{min}, v_{min}) and whose upper-right corner has coordinates (u_{max}, v_{max}). If $u_{max} - u_{min} > v_{max} - v_{min}$, then we split the region vertically at $u = (u_{max} + u_{min})/2$. If $u_{max} - u_{min} < v_{max} - v_{min}$, then we split the region horizontally at $v = (v_{max} + v_{min})/2$. In case both the knot differences are the same, then either direction could be chosen for splitting. Figure 4.12

illustrates the splitting. The offending regions are highlighted in yellow color in the figures. For the offending region in Figure 4.12(a), we split it with a horizontal edge; analogously, for the one in Figure 4.12(b), we split it with a vertical edge.

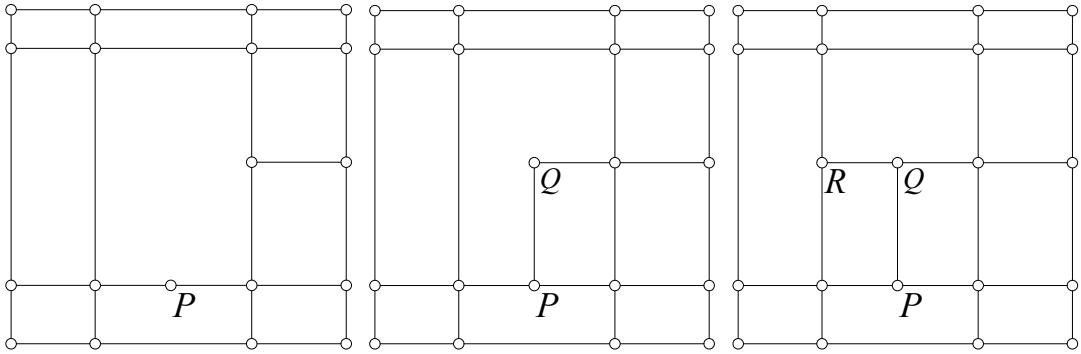


(a) Horizontal split (b) Vertical split

Figure 4.12: Split offending regions.

The actual splitting of a region is accomplished by performing a local refinement in which the two endpoints of the edge used to split the region are inserted into the T-mesh using the T-spline local knot insertion algorithm. The knot insertion algorithm might introduce a few extra points into the T-mesh. Sometimes this might result in L-junctions as shown in Figure 4.13(a) and Figure 4.13(b), where point Q is a newly formed L-junction after we insert a control point P at the demonstrated spot. The L-junctions may irregularize the T-mesh and complicate the problem. Therefore we propose to eliminate L-junctions by extending either one of their two related edges to the nearest knot line. Through experiments, we are inclined to extend them in the direction that will yield a shorter edge in the pre-image, as shown in Figure 4.13(c). As a result, a new control point R is added at the intersection of the knot line and the extended edge.

After the refinement is finished, a new T-mesh structure M and the corresponding weight set W are updated and thus a new class of T-spline surfaces is presented. The new class of T-spline surfaces has more degrees of freedom than the old one. Also, since the new T-mesh structure and weights are obtained by the T-spline local knot insertion algorithm, the new class of T-spline surfaces still consists of standard or semi-standard surfaces. What



(a) P is being added to the T-spline (b) An L-junction Q is introduced (c) The L-junction is eliminated by adding an extra point R

Figure 4.13: Eliminate L-junctions.

remains to define a T-spline surface is to find the control points and thus our approach continues to compute the control points by least squares T-spline surface approximation step described in Section 4.6.

4.9 Initial T-spline structure re-placement

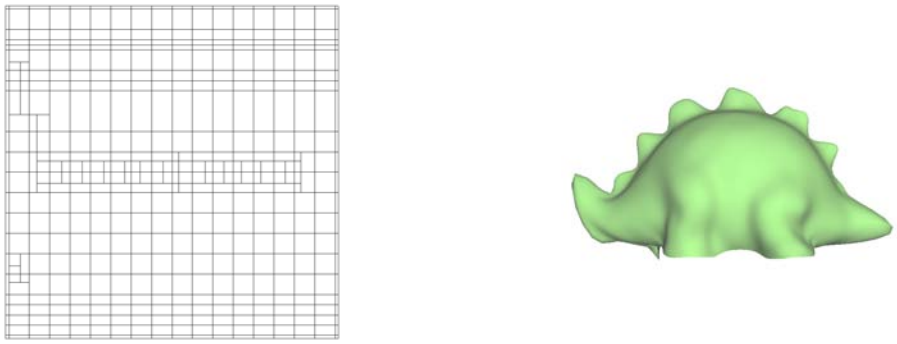
Note that when we start the adaptive T-spline surface fitting, we need to place an initial T-mesh structure. Since there is limited prior knowledge for the surface at the beginning, we simply use a regular control grid without special customization in Section 4.5. However, a carefully designed initial T-spline structure would be expected to give a better approximation result.

In addition, it can be observed that, although we keep adding new control points in a local manner, sometimes the insertion of the control points still spreads to peripheral areas during the process. Especially when the T-spline pre-image is complicated, local refinement at different areas might interact with each other and thus causes unnecessary control points. One straightforward cure to this problem is to narrow the influence of one local refinement operation. This goal can be achieved by having a more reasonable initial T-mesh placement.

In this section we present a method to construct a better initial T-mesh structure, based on the pre-image of the T-mesh of the previously generated T-spline surface. First we try

to construct a tensor-product structure M'. Let M be the pre-image of the control grid of T-spline surface $\mathbf{S}(u, v)$ that approximates the input triangular mesh. In M, each vertical edge corresponding to knot value u_i is examined. Let l_{u_i} be the sum of parameter length of all vertical edges in M with knot value u_i. If l_{u_i} is larger than a certain threshold, say 80% of the total domain span in the vertical direction, we include the entire vertical line corresponding to u_i in M'. The same approach is carried out for horizontal edges. Thus, we obtain a new T-spline pre-image M' that has a tensor product structure. At the same time, all the control point weights corresponding to M' are set to one. In this way, both M' and the weights define a class of T-spline surfaces, which actually consists of standard T-splines. Now M' is ready to be used as a new initial T-spline structure.

Second we further improve M' to better reflect those areas with details. We check all vertices of the input triangular mesh and pick a few vertices (say, n vertices) that have the largest mean curvatures, where n is preferred to be a number between 10 to 30, depending on the scale of the mesh. Then, we insert these n vertices into the initial T-mesh defined by M' at their parameter coordinates, using the T-spline local refinement algorithm to update M'. In order to avoid the occurrence of two almost touching points or two almost touching edges in the new initial T-mesh, we also make the following arrangements. Specify a small threshold ν. If the distance between a selected vertex and a control point is below ν in the pre-image, we choose not to insert that vertex. Also, if the distance between a selected vertex and an edge in M' is below ν, we slightly change the position of knot insertion so that the new point to be inserted resides on that edge. During this process, the corresponding weights are also updated. The updated M' and weights serve as the initial T-spline structure. In general, this approach gives a reasonable initial structure. Figure 4.14(a) shows the pre-image of a new T-mesh. Just based on this initial T-mesh structure without any further refinement, a pretty good T-spline surface can be reconstructed, which is shown on Figure 4.14(b).



(a) (b)

Figure 4.14: A new initial T-mesh structure and a T-spline surface defined from the new structure.

4.10 Faithful re-parameterization

The performance of the surface fitting algorithm is closely related to the parameters assigned to the triangular mesh. A good spline fitting surface usually relies on a decent parameterization. However, due to the limited intrinsic information about the triangular mesh, generally no parameterization method could provide a perfect solution no matter how advanced the method might be, especially when the triangular mesh has complicated geometry. To overcome this weakness, an alternative approach is to improve the initial parameters at a later stage, usually with the help of an intermediate result of a spline surface. This step is often referred to as parameterization improvement or re-parameterization. The modified parameters can then be used in the process of re-approximating the triangular mesh.

In this section, we present a parameterization improvement method, which is carried out after the first qualified T-spline fitting surface is obtained. Based on the current parameterization and the current T-spline surface, we compute for each vertex of the triangular mesh a parameter offset vector and use the offset vector to update the original parameters. In this re-parameterization process, we should keep the new parameterization from having any self intersection. So we call the method *faithful re-parameterization*. The method consists of two steps: the first step is to compute initial parameter correction vectors, and the second

step is to adjust the corrections to make sure that there is no self intersection.

4.10.1 Computing initial parameter corrections

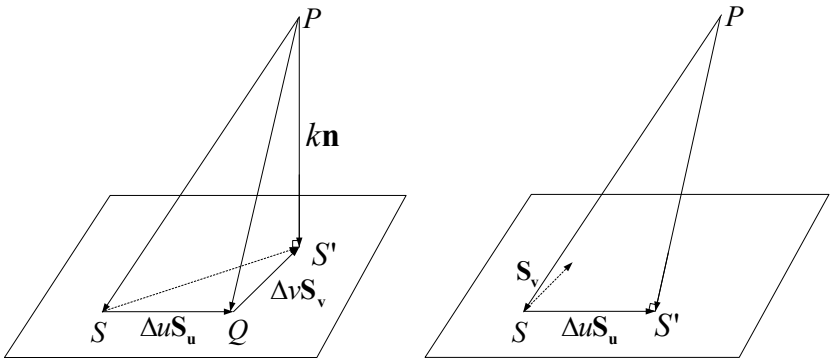
Suppose $\mathbf{S}(u, v)$ is a current T-spline surface that approximates a triangular mesh $\mathbf{T}$. For each vertex $\mathbf{d_i}$ in $\mathbf{T}$, an error vector is defined as the directed edge from $\mathbf{d_i}$ to the corresponding point $\mathbf{S}(u_i, v_i)$ on the T-spline surface, where $\mathbf{u_i} = (u_i, v_i)$ is the associated parameter pair for $\mathbf{d_i}$. For simplicity, we denote $\mathbf{d_i}$ by P and the point $\mathbf{S}(u_i, v_i)$ by S. We examine whether the error vector $\overrightarrow{PS}$ is orthogonal to the T-spline surface. If it is not the case, it implies that the parameter for P (or $\mathbf{d_i}$) could be somehow improved. Generally, there would be a point with parameter (u_i', v_i') on the T-spline surface such that the vector from P to this point is orthogonal to the surface. It can be seen that the approximation error at P can further be lowered down if we correct the parameter of P to (u_i', v_i'). However, locating such a point usually involves a nonlinear computation problem. Thus we linearize the problem and then compute the correction term $\mathbf{\Delta u_i} = (\Delta u_i, \Delta v_i)$ for parameter $\mathbf{u_i}$. Hoschek [74] proposed a similar technique to re-parameterize curves.

Let $\mathbf{S_u}$ and $\mathbf{S_v}$ denote the partial derivatives of the T-spline surface at S. S' is a point on the tangent plane spanned by $\mathbf{S_u}$ and $\mathbf{S_v}$ such that $\overrightarrow{PS'}$ is orthogonal to the tangent plane. Then $\overrightarrow{PS'}$ equals $k\mathbf{n}$, where $\mathbf{n} = \mathbf{S_u} \times \mathbf{S_v}$ is the normal vector of the plane and k is a scalar factor. From the triangle $\triangle PQS'$ shown in Figure 4.15(a), we have the following equation:

$$(\overrightarrow{PS} + \Delta u_i \mathbf{S_u}) \cdot (\mathbf{n} \times \mathbf{S_v}) = 0 \tag{4.29}$$

Solving the equation gives Δu_i:

$$\Delta u_i = -\frac{(\mathbf{S}(u_i, v_i) - \mathbf{d_i}) \cdot (\mathbf{n} \times \mathbf{S_v})}{\mathbf{S_u} \cdot (\mathbf{n} \times \mathbf{S_v})} \tag{4.30}$$



(a) Interior situation (b) Boundary situation

Figure 4.15: Modify the parameter for one vertex.

Similarly, we have

$$\Delta v_i = -\frac{(\mathbf{S}(u_i, v_i) - \mathbf{d_i}) \cdot (\mathbf{n} \times \mathbf{S_u})}{\mathbf{S_v} \cdot (\mathbf{n} \times \mathbf{S_u})} \tag{4.31}$$

If $\mathbf{d_i}$ is on the boundary of the mesh, one parameter should be fixed and thus a special treatment is needed. Suppose a boundary vertex $\mathbf{d_i}$ has parameter $\mathbf{u_i} = (u_i, v_b)$ where v_b is fixed. We only need to correct u_i. This is to ensure that the corrected point is still on the boundary. Refer to Figure 4.15(b). We find Δu_i that makes $\overrightarrow{SS'} \perp \overrightarrow{PS'}$, which leads to the following equation:

$$(\overrightarrow{PS} + \Delta u_i \mathbf{S_u}) \cdot \mathbf{S_u} = 0 \tag{4.32}$$

Thus

$$\begin{aligned} \Delta u_i &= -\frac{(\mathbf{S}(u_i, v_b) - \mathbf{d_i}) \cdot \mathbf{S_u}}{\mathbf{S_u} \cdot \mathbf{S_u}} \\ \Delta v_i &= 0 \end{aligned} \tag{4.33}$$

Analogously, if a boundary vertex $\mathbf{d_i}$ has a parameter $\mathbf{u_i} = (u_b, v_i)$ whose u_b is fixed, $\mathbf{\Delta u_i} = (\Delta u_i, \Delta v_i)$ can be computed by:

$$\begin{aligned} \Delta u_i &= 0 \\ \Delta v_i &= -\frac{(\mathbf{S}(u_b, v_i) - \mathbf{d_i}) \cdot \mathbf{S_v}}{\mathbf{S_v} \cdot \mathbf{S_v}} \end{aligned} \tag{4.34}$$

Once $\mathbf{\Delta u_i}$ is computed, we can then replace the parameter $\mathbf{u_i}$ of $\mathbf{d_i}$ by $\tilde{\mathbf{u_i}} = \mathbf{u_i} + \mathbf{\Delta u_i}$.

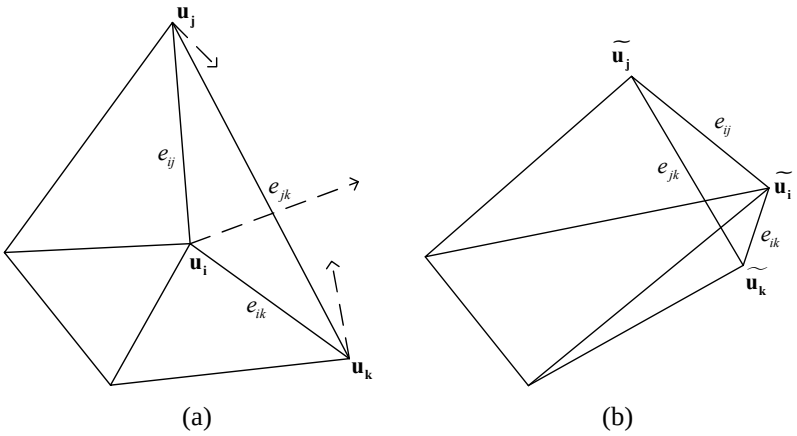


Figure 4.16: Problem for updating the parameterization.

So far we have shown how to update the parameter of a single vertex in the triangular mesh. However, simultaneously updating the parameters of all the vertices in the mesh may cause a problem. Recall that when building up the original parameterization $\mathbf{P}$, it is required that the connectivity relationship in $\mathbf{P}$ should be the same as that in the triangular mesh $\mathbf{T}$. That is, $\mathbf{P}$ is a flattened 2D triangulation whose connectivity is exactly the same as that of $\mathbf{T}$. Now if we update $\mathbf{u_i}$ in $\mathbf{P}$ using the previously calculated correction term $\mathbf{\Delta u_i}$, it might happen that in the new parameterization $\tilde{\mathbf{P}}$ some areas no longer preserve the original connectivity. For example, in Figure 4.16(a) the local region of the parameterization containing vertices $\mathbf{u_i}$, $\mathbf{u_j}$ and $\mathbf{u_k}$ is shown, with three respective correction vectors also marked on the figure. After applying these correction terms to the vertices (see Figure 4.16(b)), it can be seen that new edge $\tilde{e_{jk}}$ undesirably goes across the two other edges. This is unacceptable and we should avoid it. Otherwise, due to the self-intersection, it is hard for $\tilde{\mathbf{P}}$ to continue to faithfully preserve the connectivity as before and it could bring distortions and other unpredictable issues to the T-spline surface fitting if such parameterization is used.

4.10.2 Computing faithful parameter corrections

We have seen that the initial parameter correction terms may cause problems. It can be understood that they give over-corrections and thus we may pull the parameter points back

to avoid self-intersection. This suggests a approach described below to generate a faithful new parameterization $\tilde{\mathbf{P}}$.

We introduce a scalar factor $\beta_i \in [0, 1]$ for each vertex $\mathbf{d_i}$ and let the parameter correction for vertex $\mathbf{d_i}$ be

$$\tilde{\mathbf{u_i}} \leftarrow \mathbf{u_i} + \beta_i \mathbf{\Delta u_i} \tag{4.35}$$

Here, the correction vector $\mathbf{\Delta u_i}$ is adjusted to be $\beta_i \mathbf{\Delta u_i}$. The purpose of introducing β_i is to adjust the over-estimated correction vectors. When $\beta_i = 0$, no correction is actually made to the parameter for vertex $\mathbf{d_i}$ and $\tilde{\mathbf{u_i}} = \mathbf{u_i}$. Since the original parameterization has no self intersection, this implies that if all β_i are small enough, the updated parameterization would be faithful. On the other hand, we want the correction vectors to keep the original magnitude as much as possible in order to have the effect of re-parameterization. Therefore, our goal is to find the largest possible values for $\beta_1, \beta_2, \cdots, \beta_m$ under the condition that there is no self-intersection in the new generated parameterization $\tilde{\mathbf{P}}$ through Equation (4.35).

Let g_i denote an arbitrary triangle face in $\mathbf{P}$ whose vertices are $\mathbf{u_{i1}}$, $\mathbf{u_{i2}}$ and $\mathbf{u_{i3}}$. Let $\tilde{g}_i$ be the corresponding face in $\tilde{\mathbf{P}}$, with vertices $\tilde{\mathbf{u}}_{\mathbf{i1}}$, $\tilde{\mathbf{u}}_{\mathbf{i2}}$ and $\tilde{\mathbf{u}}_{\mathbf{i3}}$. It can be seen that, the occurrence of self intersection can be avoided if we make sure that for any face $\tilde{g}_i$ $(i = 1, 2, \cdots, l)$ in the new parameterization $\tilde{\mathbf{P}}$, each vertex in $\tilde{g}_i$ stays in the same side of the opposing edge as it does in g_i. This condition is equivalent to that the direction of the normal vectors of g_i and $\tilde{g}_i$ are the same, which we can formulate as follows:

$$(\mathbf{u_{i1}u_{i2}} \times \mathbf{u_{i1}u_{i3}}) \cdot (\tilde{\mathbf{u}}_{\mathbf{i1}}\tilde{\mathbf{u}}_{\mathbf{i2}} \times \tilde{\mathbf{u}}_{\mathbf{i1}}\tilde{\mathbf{u}}_{\mathbf{i3}}) \geq 0, \quad \text{for} \quad i = 1, 2, \cdots, l. \tag{4.36}$$

where $\mathbf{u_j u_k}$ (or $\tilde{\mathbf{u}}_{\mathbf{j}}\tilde{\mathbf{u}}_{\mathbf{k}}$) is the directional edge from $\mathbf{u_j}$ to $\mathbf{u_k}$ (or from $\tilde{\mathbf{u}}_{\mathbf{j}}$ to $\tilde{\mathbf{u}}_{\mathbf{k}}$). Replacing $\tilde{\mathbf{u}}_{\mathbf{ik}}$ by $\mathbf{u_{ik}} + \beta_{ik}\mathbf{\Delta u_{ik}}$ gives

$$(\mathbf{u_{i1}u_{i2}} \times \mathbf{u_{i1}u_{i3}}) \cdot ((\mathbf{u_{i1}u_{i2}} - \beta_{i1}\mathbf{\Delta u_{i1}} + \beta_{i2}\mathbf{\Delta u_{i2}}) \times (\mathbf{u_{i1}u_{i3}} - \beta_{i1}\mathbf{\Delta u_{i1}} + \beta_{i3}\mathbf{\Delta u_{i3}})) > 0 \tag{4.37}$$

The left side of (4.37) has only three unknowns β_{i1}, β_{i2} and β_{i3} and we denote it by a function $f_i(\beta_{i1}, \beta_{i2}, \beta_{i3})$, which is typically a degree two polynomial in β_{i1}, β_{i2} and β_{i3}. If we solve all the l inequalities $f_i(\beta_{i1}, \beta_{i2}, \beta_{i3}) \geq 0$, we can get the values for $\beta_1, \beta_2, \cdots, \beta_m$. However, it is complicated to solve multi-variable quadratic inequalities. Thus we seek to simplify the problem somehow. Rather than using an independent β_i for each $\mathbf{\Delta u_i}$, we use a global β for all the parameter correction terms. Then $f_i(\beta_{i1}, \beta_{i2}, \beta_{i3})$ becomes a univariate quadratic function $f_i(\beta)$:

$$f_i(\beta) = a_i\beta^2 + b_i\beta + c_i > 0 \tag{4.38}$$

Now our task is to solve $f_i(\beta) > 0$ for each triangle face and find the largest possible β that satisfies all the inequalities within the valid domain of β, i.e., $\beta \in [0, 1]$. The solution to a quadratic inequality of (4.38) is determined by the coefficients a_i, b_i, c_i. Note that when $\beta = 0$, no parameter correction is made and thus the direction of the normal vectors for all the triangle faces would obviously not be changed. That means, $f_i(0)$ is a positive value.

Based on the different values of a_i, b_i, c_i, the parabola of $f_i(\beta)$ would have different shapes as illustrated in Figure 4.17. Since $f_i(0) > 0$, the shapes in Figures 4.17(b), 4.17(d), and 4.17(e) are impossible. To discuss the solution of (4.38), let $\Delta = b_i^2 - 4a_ic_i$, $r_1 = \frac{-b_i-\sqrt{\Delta}}{2a_i}$ be the small root and $r_2 = \frac{-b_i+\sqrt{\Delta}}{2a_i}$ be the large root, provided $\Delta > 0$. The solution of (4.38) could have the following possibilities:

1. $\Delta \leq 0$:

   As shown in Figure 4.17(a) and 4.17(b), if $\Delta \leq 0$, the whole parabola would be above or below the β axis, depending on whether $a_i > 0$ or not. However, since $f_i(0) > 0$, the situation in Figure 4.17(b) would not happen. Therefore, when $\Delta \leq 0$, $f_i(\beta)$ would always be larger than 0, and β can take any value between 0 and 1.

2. $\Delta > 0$, $r_1 \in [0, 1]$:

   This case is illustrated by Figure 4.17(c). In order for $f_i(\beta) > 0$, we have $0 \leq \beta < r_1$.

3. $\Delta > 0$, $r_1 \notin [0, 1]$, $r_2 \in [0, 1]$:

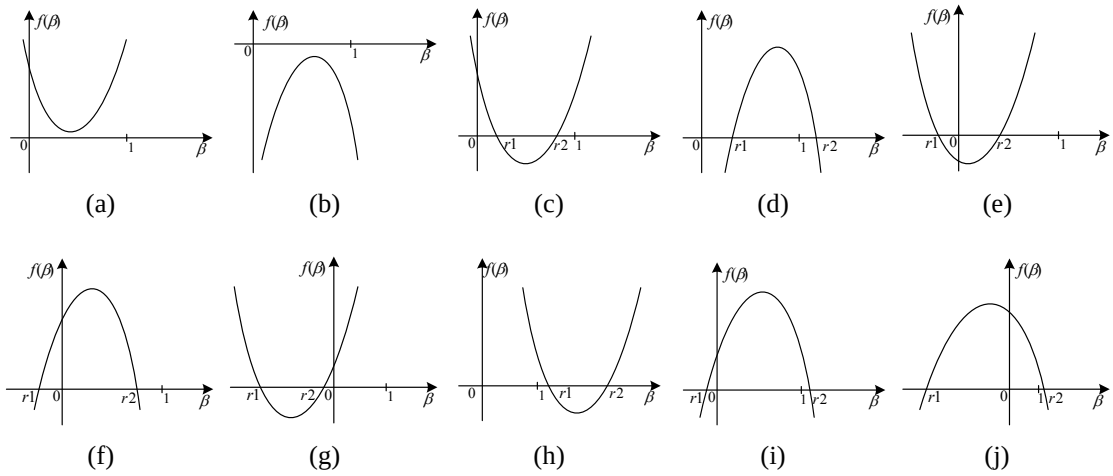


Figure 4.17: Various situations for $f_i(\beta)$.

This case is illustrated by Figure 4.17(f). In order for $f_i(\beta) > 0$, we have $0 \leq \beta < r_2$.

4. $\Delta > 0$, $r_1 \notin [0,1]$, $r_2 \notin [0,1]$:

   This case is illustrated by Figure 4.17(g)-4.17(j). It is obvious that in such case, β can take any value between 0 and 1.

After solving each inequality, we combine the results and obtain a final value for β. Then we update the parameters using $\tilde{\mathbf{u}}_\mathbf{i} = \mathbf{u}_\mathbf{i} + \beta\boldsymbol{\Delta}\mathbf{u}_\mathbf{i}$. With β being a factor for the correction terms, there is now no need to concern about that the parameters would be over-modified. The updated parameterization $\tilde{\mathbf{P}}$ would continue to have the same connectivity as $\mathbf{P}$, without any self intersection.

4.11 Experimental results

In this section, we evaluate our surface fitting method with several practical examples. The geometrical information of the input models is given in Table 4.1.

In our T-spline surface fitting algorithm, the steps of parameterization and least squares T-splines surface approximation need to solve a linear system. These linear systems usually have a sparse matrix. Therefore, a fast and stable linear system solver is sought. In

Table 4.1: The geometric information of the models used in Section 4.11.

Model	Name	Vertex number	Face number
	Cat	3533	6975
	Stegosaurus	12409	24672
	Face	24155	47976
	Stamp	35852	71442
	Max-Planck	25445	50801
	Fandisk	5051	9926
	Bump	8181	16200

our implementation, we adopt the preconditioned complex bi-conjugate gradient (PCBCG) solver [115].

Example 1

In Figure 4.18, a step by step illustration of the process of approximating the Stegosaurus model is given. Figure 4.18(a) shows the input mesh. Using Algorithm 4.1, we begin with a preliminary 4×4 T-mesh topology and Figure 4.18(b) shows the T-mesh topology (or the pre-image of the T-mesh) and the optimal surface fitting result under that T-mesh topology.

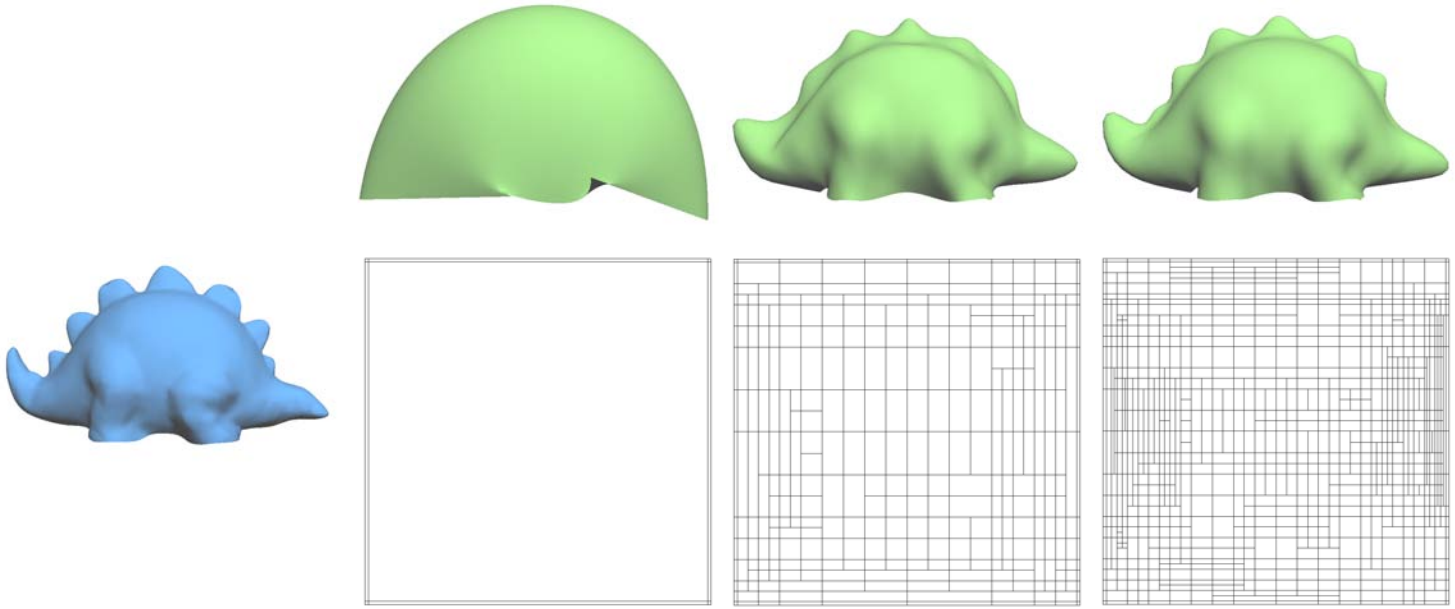
In the calculation of the least squares optimal surface, both the geometric error and the fairness are considered. Unless explicitly specified, the fairness factor σ is set to 10^{-4} in this section. Once the optimal surface is found at each iteration, the surface is checked with the curvature-guided individual error tolerance. For this model, the global error tolerance ε is set to be 0.5% of the scale of the triangular mesh. Obviously, the surface shown in Figure 4.18(b) is not satisfactory and many vertices in the mesh violate the error tolerance. Therefore, the T-mesh is refined accordingly, followed by another iteration of calculating the optimal surface.

Figure 4.18(c) and Figure 4.18(d) show two of the subsequent intermediate results after several fitting iterations. It can be observed that as the T-mesh is being gradually refined, the optimal T-spline surface contains more and more geometrical details and gains more resemblance to the input mesh. Some statistics for these T-spline surfaces are provided in Table 4.2, which includes for each surface the number of elapsed iterations, the number of control points, the number of knots in both parameter directions, the percentage of the violating vertices, and the maximum and average approximation error. It can be seen that the approximation error is gradually reduced while more control point are added to the T-mesh.

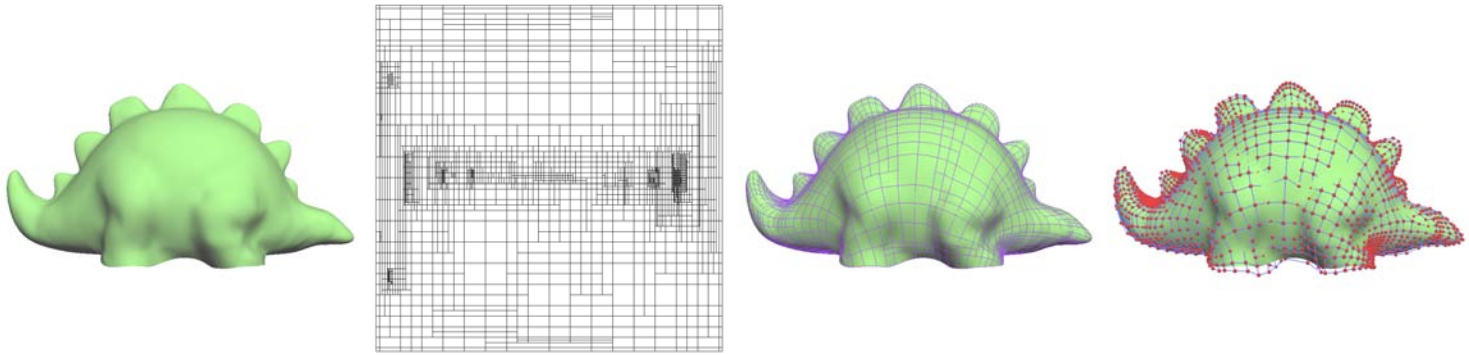
Table 4.2: The statistics for the T-spline surfaces in Figure 4.18.

	Initial	Intermediate 1	Intermediate 2	Final
#iterations	1	5	7	11
#control points	16	389	1094	2554
#knots ($u-$direction)	4	26	55	174
#knots ($v-$direction)	4	24	44	75
violating vertices	90.41%	47.29%	35.15%	0%
ε_{max}	11.30%	4.35%	2.88%	0.37%
ε_{avg}	1.62%	0.49%	0.31%	0.04%

Eventually, Figure 4.18(e) shows the final fitting T-spline surface, of which the maximum error is below 0.37%. Figures 4.18(f)-4.18(h) display its corresponding pre-images in 2D and 3D, and the T-mesh, respectively. The average approximation error is even lower, which is just 0.04%. The surface is obtained after 11 iterations and the T-mesh has 2544 control points. The geometry of the triangular mesh is appropriately represented by the T-spline surface and it can be seen that more control points are located in the regions where surface details are present.



(a) The input mesh (b) The initial result (c) The intermediate result 1 (d) The intermediate result 2



(e) The final T-spline surface (f) The T-mesh pre-image (g) The mapping of the pre-image on the surface (h) The T-mesh

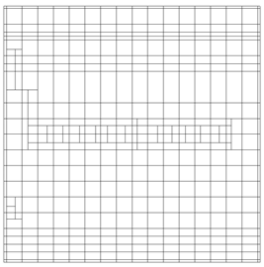
Figure 4.18: The iterative procedure of fitting a T-spline surface to a triangular mesh.

There are 174 and 75 knots in the two knot vectors respectively and any B-spline surface defined by these knot vectors has to host $174 \times 75 = 13050$ control points. The permit of the existence of T-junctions has brought a lot of flexibility into the construction of the T-mesh and enables the resulting T-spline surface to keep only a small set of control points.

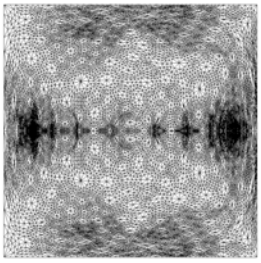
Example 2

In Figure 4.19, the performance of the re-placement of the initial T-mesh structure and the re-parameterization of the triangular mesh is investigated. Based on the T-spline surface fitting result shown in Figure 4.18(e), a new initial T-mesh topology (see Figure 4.19(a)) and a new parameterization of the mesh (see Figure 4.19(b)) are computed, using the approaches described in Section 4.9 and Section 4.10, respectively. Since now the initial T-mesh and the parameterization are computed with the knowledge of an approximation T-spline surface, they are expected to bring better fitting results.

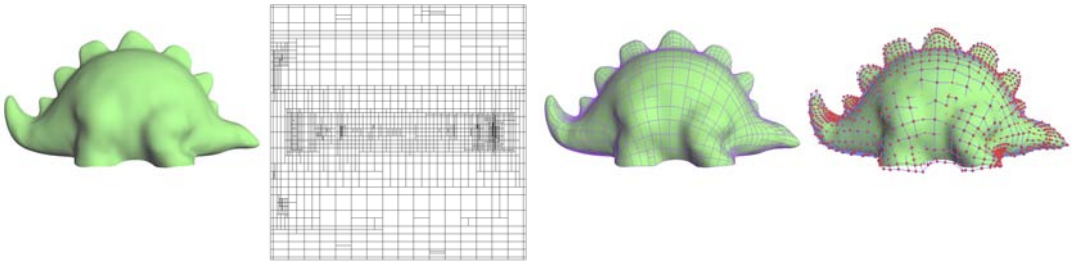
In order to evaluate their effectiveness, we try to use them as the initial settings in surface fitting, both individually and simultaneously. Figure 4.19(c) shows the surface fitting result which uses the T-mesh topology in Figure 4.19(a) as the initialization. From the left to the right in Figure 4.19(c) are the resulting T-spline surface, the T-mesh pre-image, the surface with mapped pre-image and the T-mesh. Similarly, Figure 4.19(d) is the surface fitting result that adopts the parameterization in Figure 4.19(b). Finally, the T-spline surface in Figure 4.19(e) is obtained by using both the new T-mesh initialization and the new parameterization. The information of the number of control points and the approximation errors for these surfaces are given in Table 4.3. While the approximation errors remain roughly at the same level as the T-spline surface shown in Figure 4.18(e), all of the newly obtained surfaces involve fewer control points. When both the new T-mesh initialization and the new parameterization are used, a surface with fewest number of control points is achieved. Since the quality of the surface is not compromised, the new resulting surface can be regarded as having higher efficiency in representation.



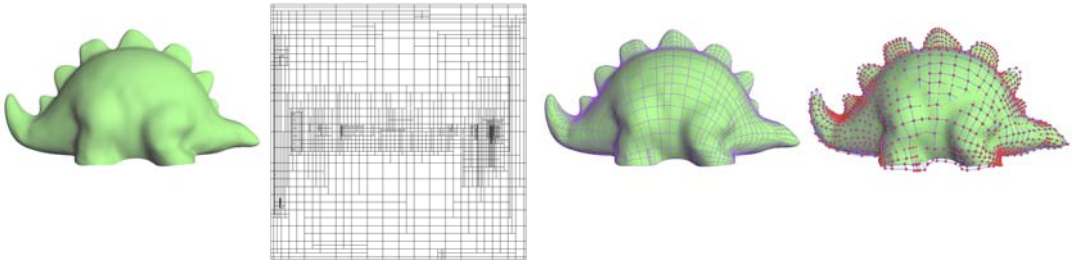
(a) The new initial T-mesh topology



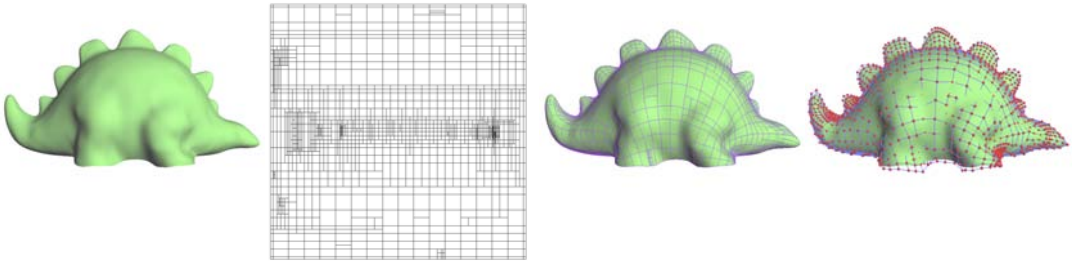
(b) The re-parameterization of the triangular mesh



(c) The surface fitting result with the new initial T-mesh topology



(d) The surface fitting result with the new parameterization



(e) The surface fitting result with both the new initial T-mesh and the new parameterization

Figure 4.19: Re-initialization, re-parameterization and their influence on surface fitting.

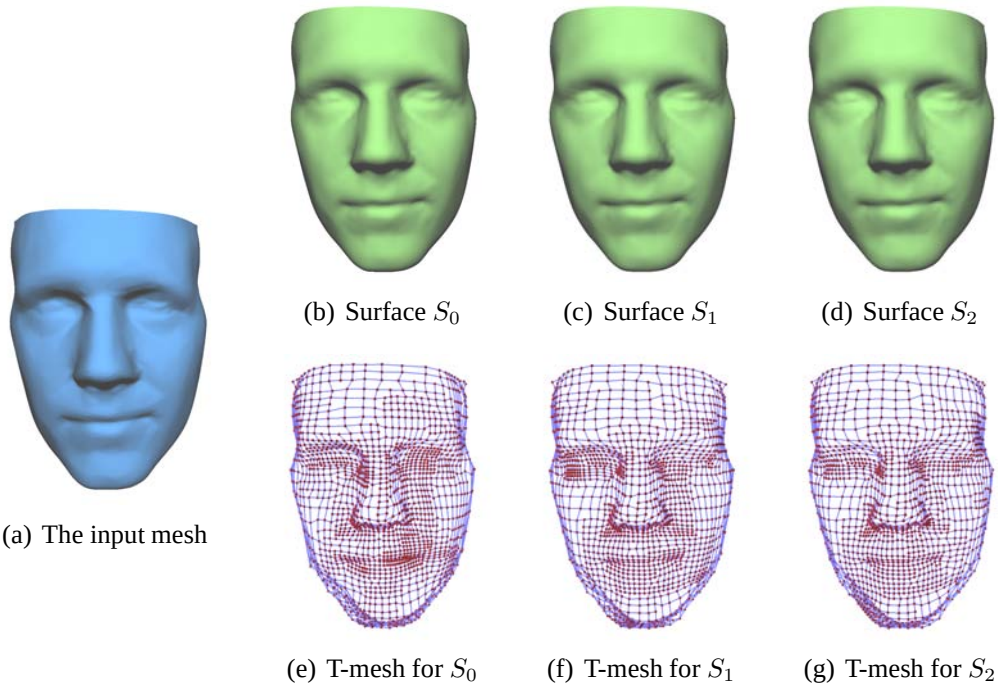
Table 4.3: The statistics for the T-spline surfaces in Figure 4.19.

	with re-initialization	with re-parameterization	with both
#control points	2053	2314	1788
ε_{max}	0.36%	0.35%	0.35%
ε_{avg}	0.04%	0.03%	0.04%

Example 3

Now we demonstrate the performance of Algorithm 4.2. Basically, Algorithm 4.2 is done by iteratively carrying out Algorithm 4.1 and computing new initial T-mesh topology and new parameterization between each two iterations. In this process, a number of qualified T-spline surfaces are generated, which are here denoted by $S_0, S_1, S_2, \ldots$. Specifically, S_0 is the result of directly applying Algorithm 4.1 and S_i is the surface fitting result after the initial T-mesh topology and the parameterization are re-computed i times. Figure 4.20(b), Figure 4.20(c) and Figure 4.20(d) show the surface S_0, S_1 and S_2 that approximate the triangular mesh in Figure 4.20(a). The corresponding T-meshes for them are displayed in Figure 4.20(e) to Figure 4.20(g). Table 4.4 gives the statistics for these surfaces, from which it can be seen that the numbers of control points are being gradually reduced, from S_0 to S_2. Moreover, it can be observed in Figure 4.20 that as the iterations go on, the distribution of the control points in the T-mesh of the surface becomes more reasonable. In the T-mesh for S_2 (shown in Figure 4.20(g)), more control points are concentrated in the eye, nose and mouth areas.

According to our experience, a good result can be obtained in 2-5 iterations of Algorithm 4.2. Unlike Algorithm 4.1, the stop condition for Algorithm 4.2 is somehow different. Here, we demand that the result from current iteration must show a improvement of 3% against the previous one. The improvement can be either in the approximation error or the number of control points. When neither of these two targets are improved, we simply terminate the whole algorithm.



(a) The input mesh (b) Surface S_0 (c) Surface S_1 (d) Surface S_2

(e) T-mesh for S_0 (f) T-mesh for S_1 (g) T-mesh for S_2

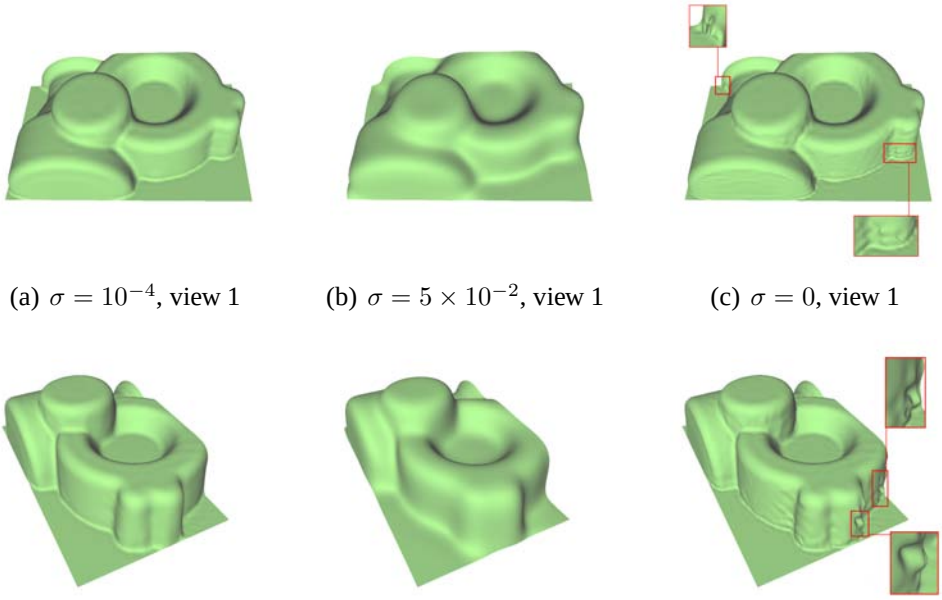
Figure 4.20: The performance of Algorithm 4.2.

Table 4.4: The statistics for the T-spline surfaces in Figure 4.20.

	S_0	S_1	S_2
#control points	1688	1603	1497
ε_{max}	0.14%	0.17%	0.10%
ε_{avg}	0.01%	0.01%	0.01%

Example 4

In Figure 4.21, the influence of the value of the fairness factor is illustrated. The surfaces shown in Figure 4.21(a), Figure 4.21(b) and Figure 4.21(c) are approximated with $\sigma = 10^{-4}$, 5×10^{-2} and 0, respectively. The T-mesh topologies of these surfaces are identical. Figure 4.21(d) to Figure 4.21(f) show these surfaces in a different viewpoint. When $\sigma = 10^{-4}$, the surface has a smooth look and the fitting quality is quite good. When $\sigma = 5 \times 10^{-2}$, the surface is smooth, but the shape is blurred. When $\sigma = 0$, it means the fairness functional is not involved during optimization. Unfortunately, the resulting surface does not have a pleasing look and the surface quality is rather poor in the zoomed areas.



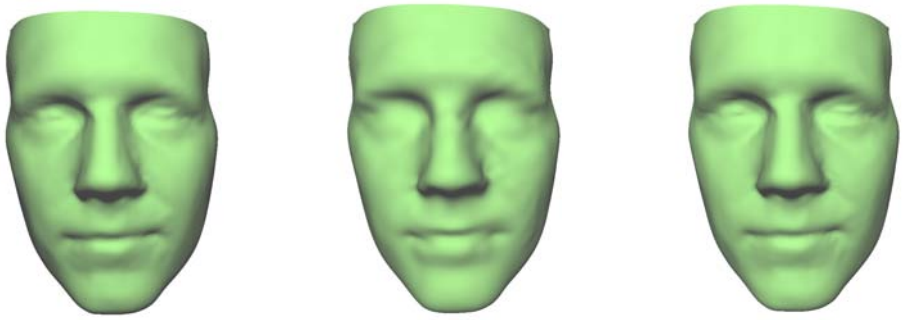
(a) $\sigma = 10^{-4}$, view 1 (b) $\sigma = 5 \times 10^{-2}$, view 1 (c) $\sigma = 0$, view 1

(d) $\sigma = 10^{-4}$, view 2 (e) $\sigma = 5 \times 10^{-2}$, view 2 (f) $\sigma = 0$, view 2

Figure 4.21: The behavior of fairness factors.

Example 5

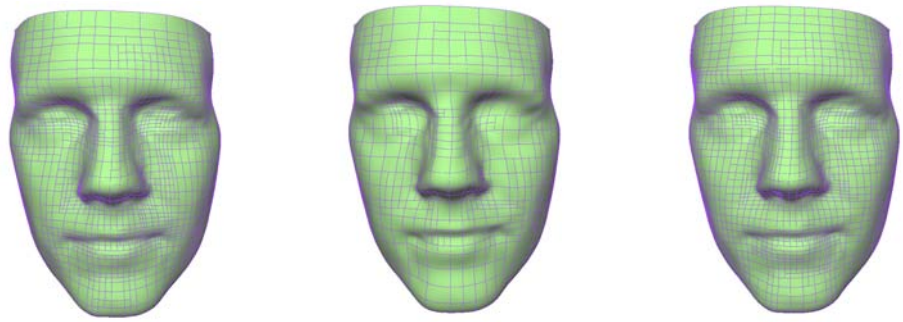
We analyze the effect of the curvature guidance in surface fitting. Surface S_1 in Figure 4.22(a) is obtained with the curvature guidance switched on during the surface fitting process and it has 1688 control point. Surface S_2 in Figure 4.22(b) is computed using the same global error tolerance as S_1, but the curvature guidance is switched off during surface fitting. S_2 has less control points, but the surface quality is quite low, especially in the eye and mouth areas where details are lost. If one wants to achieve the similar surface quality as S_1 without using the curvature guidance in surface fitting, the global error tolerance thus has to be set to a very small value. This leads to surface S_3 in Figure 4.22(c), which has 2024 control points that is 20% more than the control points of S_1. From Figure 4.22(f), it can be seen that the distribution of the control points for S_3 is quite uniform, placing as many control points in the non-feature areas as in the feature areas. This would lower down the efficiency of the surface representation which is quite undesirable.



(a) Surface S_1, with curvature guidance, moderate ε, 1688 control points

(b) Surface S_2, without curvature guidance, moderate ε, 798 control points

(c) Surface S_3, without curvature guidance, small ε, 2024 control points



(d) S_1 mapped with T-spline pre-image

(e) S_2 mapped with T-spline pre-image

(f) S_3 mapped with T-spline pre-image

Figure 4.22: Surface fitting results with and without curvature guidance.

More examples

Finally, more examples of fitting various models are presented in Figure 4.23. From the left column to the right column are the input meshes, the resulting T-spline surfaces, the pre-images and the T-meshes. The number of control points and the approximation errors for these surfaces are given in Table 4.5.

Table 4.5: The statistics for the T-spline surfaces in Figure 4.23.

	Stamp	Max-Planck	Bump	Fandisk
#control points	2506	3510	1081	2093
ε_{max}	0.45%	0.23%	0.30%	0.57%
ε_{avg}	0.02%	0.04%	0.02%	0.04%

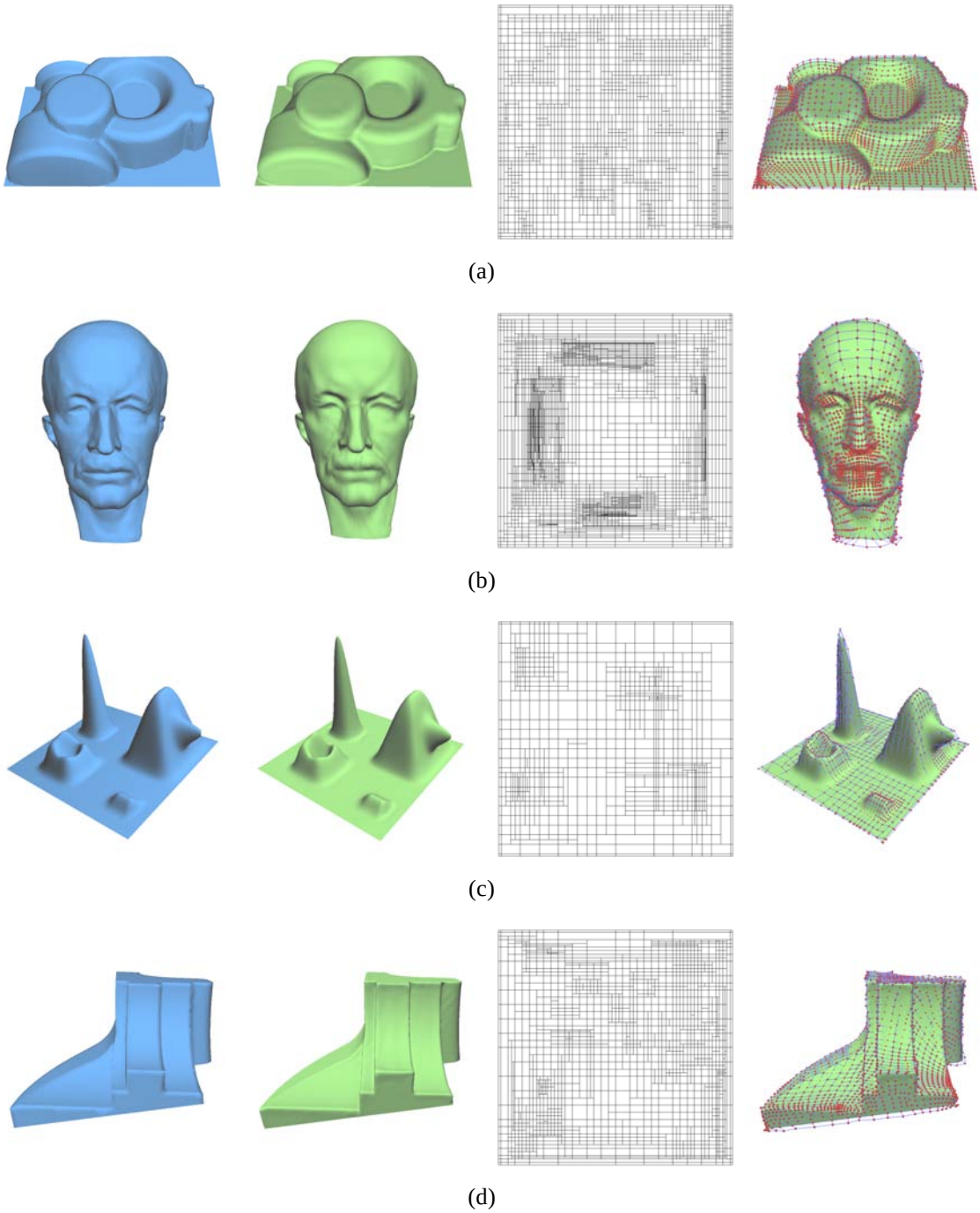


Figure 4.23: More examples on T-spline surface fitting.

4.12 Summary

In this chapter, a new framework for adaptively approximating a triangular mesh using a T-spline surface is proposed, in which conventional surface fitting or adaptive surface fitting methods are enhanced and new components are integrated. The conventional adaptive surface fitting methods are enhanced by T-splines and geometric features. The new components include initial T-spline structure re-placement and faithful re-parameterization. All components or steps of the framework or algorithms are carefully designed to achieve the best performance. In particular, by taking the advantage of the T-spline local refinement property, the algorithm is able to output a surface that gives a good approximation to the triangular mesh. Many examples have demonstrated the effectiveness of the proposed framework and algorithms.

Chapter 5

Periodic T-spline Surface Representation and Approximation

5.1 Introduction

Among various types of triangular meshes, tubular meshes are a special one. A tubular mesh is characterized by having two boundary loops and one lateral surface between them. The lateral surface is open along one direction (the axial direction) and closed along the other direction (the sectional direction). The tubular mesh can be viewed as a deformation of a cylinder. The upper and lower boundaries of the cylinder correspond to the two loops and the interior of the cylinder surface corresponds to the lateral surface (see Figure 5.1). In mechanical engineering and medical engineering, tubular surfaces are quite common. For example, pipes and blood vessels are tubular surfaces. Therefore tubular meshes are useful to represent these shapes. Some examples of tubular meshes are shown in Figure 5.2.

To convert a tubular mesh into a spline surface, one approach is to use the method developed in the preceding chapter. Due to the difference between cylinder topology and disc topology, the traditional way is to perform a pre-process step which cuts the tubular mesh into a disc-like mesh. However, this cutting introduces discontinuities and distortions

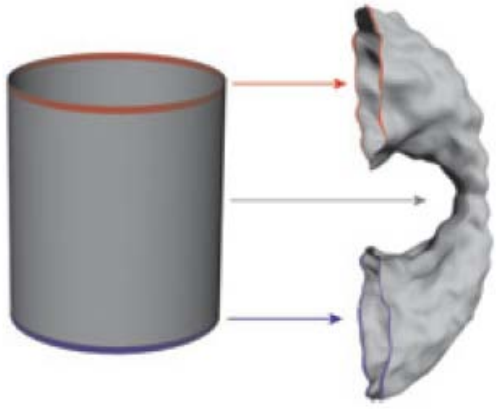


Figure 5.1: The upper and lower boundaries of the cylinder are mapped to two boundary loops of the tubular mesh and the interior of the cylinder is mapped to the lateral surface of the tubular mesh [78].

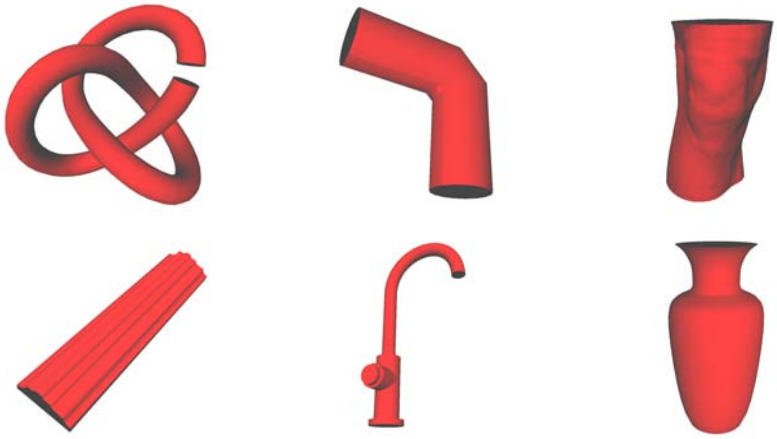


Figure 5.2: Some examples of tubular meshes.

into parameterization and surface fitting. Moreover, the topological structure and semantics of the shape get lost by cutting. In this chapter, we investigate the techniques that directly approximate tubular meshes without cutting. To this end, the following issues have to be considered.

First, we have to choose an appropriate spline representation. Note that conventional T-spline surfaces are defined over an rectanglar domain and tubular meshes have the same topology as a cylinder. It might be more appropriate or natural to introduce periodic T-splines to approximate tubular meshes. While T-splines are the generalization of NURBS,

periodic T-splines generalize periodic NURBS by allowing existence of T-junction points. This chapter presents the formulation for one type of periodic T-splines that are periodic in one direction.

Second, we consider the problem of parameterizing tubular triangular meshes. Unlike an open mesh that is of plane topological type, a tubular mesh gives rise to some special issues in parameterization due to its mesh structure. In this chapter, we present an edge based parameterization method, in which the edges rather than the vertices of the mesh are treated as the target for parameterization. This approach improves conventional cutting-based algorithms which cut the mesh to make it a disk topologically. The problem of cutting paths is their zigzag shape that leads to suboptimal parameterizations and also finding good cutting paths is very difficult. Our proposed method does not need cutting of the mesh. It first parameterizes the edges on the two boundaries of the tubular mesh, then parameterizes the internal edges based on the mean value coordinates, and finally computes the parameters of the mesh vertices.

Third, we need to adapt our surface fitting method developed in the preceding chapter to fit a periodic T-spline surface to a tubular mesh. The approach is designed in an adaptive manner, which also takes into account the local features of the triangular mesh.

5.2 Periodic T-spline surface representation

5.2.1 Periodic B-splines

B-spline basis functions are piecewise polynomials with finite support. They are not periodic. To deal with closed shapes naturally and seamlessly, it is more appropriate to use periodic functions. Therefore we begin by constructing periodic blending functions from B-spline basis functions. For simplicity, we describe our construction for the degree three case. The extension to any degree is straightforward.

Consider a cubic B-spline basis function $N^3[\mathbf{t}_i](t)$ associated with knot vector $\mathbf{t}_i =$

$[t_{i-2}, t_{i-1}, \cdots, t_{i+2}]$. $N^3[\mathbf{t}_i](t)$ is zero for t outside of interval (t_{i-2}, t_{i+2}). Given a period T, we define

$$\tilde{N}^3[\mathbf{t}_i](t) = \sum_{j=-\infty}^{+\infty} N^3[\mathbf{t}_i](t + jT). \tag{5.1}$$

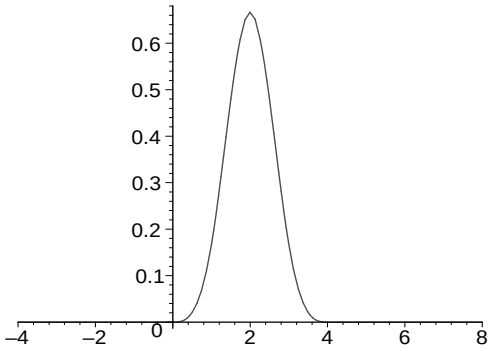
In the above formula, each $N^3[\mathbf{t}_i](t + jT)$ is also a B-spline basis function $N^3[\mathbf{t}_i - jT](t)$ associated with knot vector $[t_{i-2} - jT, t_{i-1} - jT, \cdots, t_{i+2} - jT]$. Obviously, $\tilde{N}^3[\mathbf{t}_i](t)$ is a periodic function with period T. When $T \geq t_{i+2} - t_{i-2}$, $\tilde{N}^3[\mathbf{t}_i](t)$ coincides with $N^3[\mathbf{t}_i - jT](t)$ in $[t_{i-2} - jT, t_{i+2} - jT]$ for any integer j. Refer to Figure 5.3 for an illustration. $N^3[0, 1, 2, 3, 4](t)$ is the same as $\tilde{N}^3[0, 1, 2, 3, 4](t)$ with $T = 4$ in $[0, 4]$. However, when $T = 3$, $N^3[0, 1, 2, 3, 4](t)$ and $\tilde{N}^3[0, 1, 2, 3, 4](t)$ are different.

Periodic B-spline curves

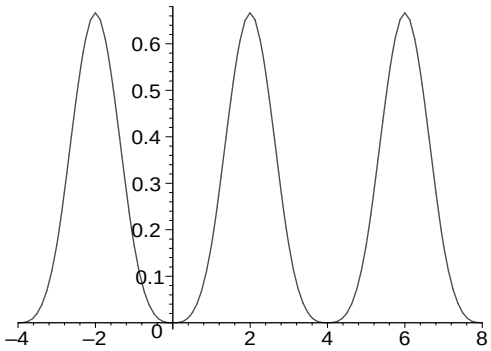
With the periodic function $\tilde{N}^3[\mathbf{t}_i](t)$, it is easy to define periodic B-spline curves.

With reference to Figure 5.4, $(n + 1)$ points $P_0, P_1, \cdots, P_n$ are given, forming a closed control polygon. For each edge $P_iP_{(i+1)\bmod(n+1)} (i = 0, \cdots, n)$ of the polygon, a knot interval d_i is assigned for the purpose of conveying knot information. Knot interval d_i is the difference between two consecutive knots t_i and t_{i+1} in the knot vector and thus the length of parameter range of the curve segment $c_i(t)$ corresponding to the edge $P_iP_{(i+1)\bmod(n+1)}$ in cubic B-splines. Therefore, to generate a knot vector, we simply let $t_0 = 0$ without loss of generality. Then we iteratively compute $t_{i+1} = t_i + d_{i\bmod(n+1)}, i = 0, 1, \cdots$, and $t_{j-1} = t_j - d_{(j-1)\bmod(n+1)}, j = 0, 1, \cdots$. As a result, we get a knot sequence $\{\cdots, t_{-2}, t_{-1}, t_0, t_1, t_2, \cdots\} = \{\cdots, -d_{n-1} - d_n, -d_n, 0, d_0, d_0 + d_1, \cdots\}$, from which we can extract a local knot vector $\mathbf{t_i} = [t_{i-2}, t_{i-1}, t_i, t_{i+1}, t_{i+2}]$ for the B-spline function $N^3[\mathbf{t}_i](t)$ associated with P_i. If we define the period

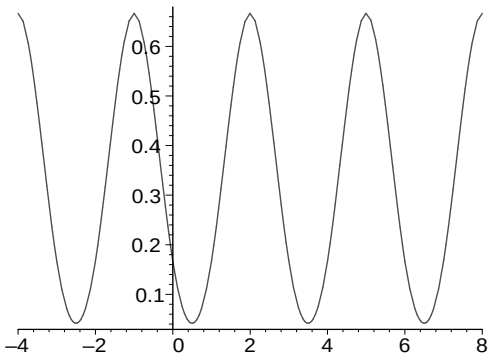
$$T = t_{n+1} - t_0 = d_0 + d_1 + \cdots + d_n,$$



(a) B-spline basis function $N^3[0,1,2,3,4](t)$



(b) Periodic B-spline $\tilde{N}^3[0,1,2,3,4](t)$ with period $T = 4$



(c) Periodic B-spline $\tilde{N}^3[0,1,2,3,4](t)$ with period $T = 3$

Figure 5.3: Illustration of a B-spline basis function and two periodic B-spline functions.

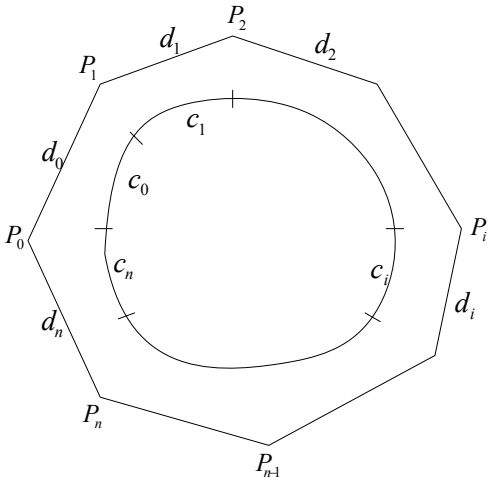


Figure 5.4: A cubic periodic B-spline curve.

then a periodic B-spline curve is defined by

$$C(t) = \sum_{i=0}^{n} P_i \tilde{N}^3[\mathbf{t_i}](t), \qquad t \in (-\infty, +\infty).$$

Obviously, for any t, $C(t+T) = C(t)$. That is, the shape of the curve is completely defined by $C(t)$ within a finite parameter interval $[a, a+T)$ for any number a. Therefore we constrain ourselves to the domain $[t_0, t_{n+1}] = [0, T)$:

$$C(t) = \sum_{i=0}^{n} P_i \tilde{N}^3[\mathbf{t_i}](t), \qquad t \in [0, T). \tag{5.2}$$

It is worthwhile to point out that though $\tilde{N}^3[\mathbf{t_i}](t)$ is a sum of a infinite number of B-spline basis functions, no more than 3 terms do not vanish in domain $[0, T)$. In fact, let us consider $\tilde{N}^3[\mathbf{t_i}](t)$ corresponding to the control point P_i. When the number of the control points is greater than 1 (i.e., $n > 0$), we have $t_{i+2} - t_i \leq T$ and $t_i - t_{i-2} \leq T$ for $\mathbf{t_i} = [t_{i-2}, t_{i-1}, t_i, t_{i+1}, t_{i+2}]$. Then all $N^3[\mathbf{t_i} - jT](t)$ and $N^3[\mathbf{t_i} + jT](t)$ vanish in $[0, T]$ for $j > 1$. Therefore we can write $\tilde{N}^3[\mathbf{t_i}](t) = N^3[\mathbf{t_i} - T](t) + N^3[\mathbf{t_i}](t) + N^3[\mathbf{t_i} + T](t)$.

Periodic NURBS surfaces

Analogously, we can define periodic NURBS surfaces. For example, given $(n+1)\times(m+1)$ points P_{ij}, $i=0,\cdots,n$ and $j=0,\cdots,m$, serving as the control points, and their corresponding weights w_{ij}, we connect adjacent points vertically and horizontally and also connect P_{nj} to P_{0j} for each j. In this way, we form a control mesh which is close in the horizontal direction. Next, a knot interval d_i is assigned to each horizontal edge $P_{ij}P_{(i+1)j}$ for all j and a knot interval e_j is assigned to each vertical edge $P_{ij}P_{i(j+1)}$ for all i. Since the mesh along the vertical direction is open, we need to assign extra knot intervals e_{-1} and e_m past each of the two end control points. From the knot intervals, we can construct two knot sequences $\{\cdots,u_{-2},u_{-1},u_0,u_1,u_2,\cdots\}$ and $\{v_{-1},v_0,v_1,\cdots,v_m,v_{m+1}\}$ as follows:

$$\begin{array}{lll} u_0=0; & u_{i+1}=u_i+d_{i\bmod(n+1)}, i=0,1,\cdots; & u_{j-1}=u_j-u_{(j-1)\bmod(n+1)}, j=0,1,\cdots \\ v_0=0; & v_{i+1}=v_i+e_i, i=0,\cdots,m; & v_{-1}=v_0-e_{-1}. \end{array}$$

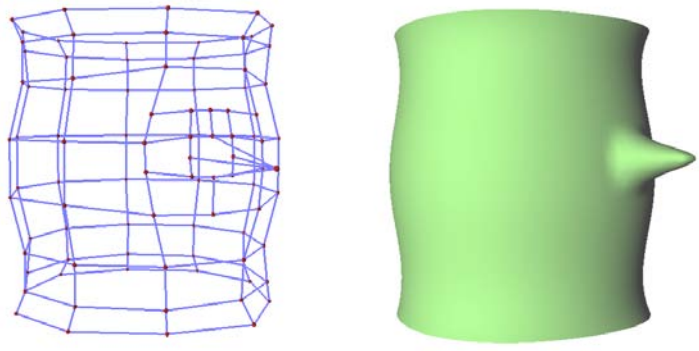
Let $\mathbf{u_i}=[u_{i-2},u_{i-1},u_i,u_{i+1},u_{i+2}]$ and $\mathbf{v_i}=[v_{i-2},v_{i-1},v_i,v_{i+1},v_{i+2}]$ and we define the u-directional period

$$T_u=u_{n+1}-u_0=d_0+d_1+\cdots+d_n,$$

then a periodic NURBS surface is defined by

$$\mathbf{S}(u,v)=\frac{\sum_{i=0}^{n}\sum_{j=0}^{m}w_{ij}P_{ij}\tilde{N}^3[\mathbf{u_i}](u)N^3[\mathbf{v_j}](v)}{\sum_{i=0}^{n}\sum_{j=0}^{m}w_{ij}\tilde{N}^3[\mathbf{u_i}](u)N^3[\mathbf{v_j}](v)},\qquad u\in(-\infty,+\infty),\ v\in[v_1,v_{m-1}].$$

Since the shape of the surface is completely defined by $\mathbf{S}(u,v)$ within a finite parameter interval $[a,a+T)\times[v_1,v_{m-1}]$ for any number a, we can focus on $\mathbf{S}(u,v)$ only in domain $[0,T)\times[v_1,v_{m-1}]$ in practice. Within this domain, $\tilde{N}^3[\mathbf{u_i}](u)=N^3[\mathbf{u_i}-T](u)+N^3[\mathbf{u_i}](t)+N^3[\mathbf{u_i}+T](u)$ as long as $n>0$.



T-mesh Periodic T-spline surface

Figure 5.5: A T-mesh and its periodic T-spline surface.

5.2.2 Periodic T-spline surfaces

Now we are ready to describe the formulation of periodic T-spline surfaces. Although we can define T-spline surfaces that are periodic in either u or v parametric direction (univariate periodic T-splines), or periodic in both u and v directions (bivariate periodic T-splines), in this chapter we focus on discussion of univariate periodic T-splines which are periodic in the u direction. Such periodic T-spline surfaces are close in the u direction and open in the v direction. They are of the same topological type as a cylinder surface. In the remainder of this chapter the term "periodic T-splines" just refers to u-periodic T-splines.

A periodic T-spline surface is defined by a set of control points forming a control grid called a T-mesh. A T-mesh is a basically a topologically cylinder-like rectangle grid that is similar to a periodic NURBS control grid discussed in the preceding sub-section, but allows T-junctions. In a T-mesh, a row or column of control points is permitted to terminate interiorly. The final control point in a partial row or column is a T-junction. An example of a T-mesh and its periodic T-spline surface is shown in Figure 5.5. When a T-mesh does not contain any T-junction, it degenerates to a periodic NURBS control grid and thus the periodic T-spline surface becomes a periodic NURBS surface.

Knot information for T-splines is given by knot intervals assigned to each edge in the T-mesh. Figure 5.6 shows the pre-image of a T-mesh in (u, v) parameter space, with red

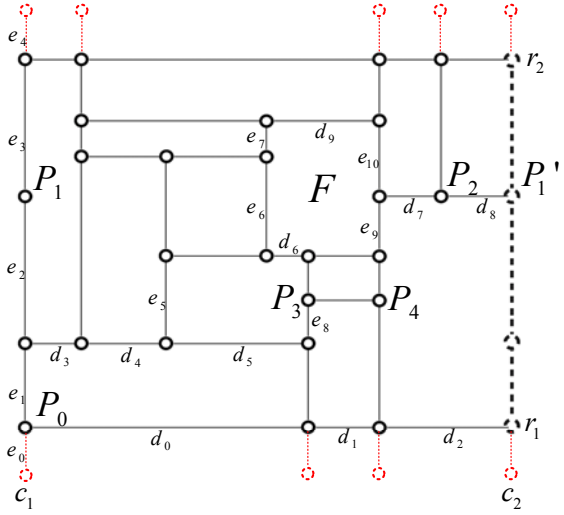


Figure 5.6: An example pre-image for a periodic T-spline surface.

edges containing boundary-condition knot intervals for the v direction. Since the T-mesh is close in the u direction, we arbitrarily choose a column (for example, c_1) as a virtual border and unfold the T-mesh along the virtual border. Column c_2 is a virtual duplicate of column c_1 and P_1, P_1' actually refer to the same control point in the T-mesh. Both c_1 and c_2 are called the left and right virtual borders. d_i and e_j denote the knot intervals. Note that in a periodic NURBS control grid, all the edges between two columns of control points or two rows of control points share a same knot interval due to the topologically tensor-product structure of the mesh. In a periodic T-spline surface, the T-mesh generally does not have such a structure and the assignment of knot intervals is more complicated. To specify an unambiguous T-mesh for a periodic T-spline surface, some rules should be followed, such as Rule 1 and Rule 2 in [129, 128].

Thus according to Rule 1, for face F in Figure 5.6, $d_1 + d_6 = d_9$ and $e_6 + e_7 = e_9 + e_{10}$. It can also be seen that the sum of the knot intervals in row r_1 equals the sum of the knot intervals in row r_2. This sum will be chosen as the period T_u for the u direction. Based on Rule 2, for example, the edge between P_3 and P_4 should be included in the T-mesh. Rule 2 is also applicable to the control point pairs on the leftmost and the rightmost sides of Figure 5.6, which means that edges such as the one between P_1 (or P_1') and P_2 should be

included in the T-mesh.

To introduce knots, we have to impose a knot coordinate system. This can be done by arbitrarily designating the pre-image of a control point (say P_0 in Figure 5.6) to be the knot origin with coordinates $(0, 0)$ and then assigning a u knot value to each vertical edge and a v value to each horizontal edge based on the knot interval information. After that, each control point has knot coordinates. For example, P_1, P_2 and P_3 have knot coordinates $(0, e_1 + e_2)$, $(d_0 + d_1 + d_7, e_1 + e_2)$ and $(d_0, e_1 + e_8)$, respectively.

Now we define the period T_u in the U parameter direction to be the difference of the u knot at the right virtual border and the u knot at the left virtual border. For each control point P_i, we need to extract its two knot vectors $\mathbf{u_i} = [u_{i0}, u_{i1}, u_{i2}, u_{i3}, u_{i4}]$ and $\mathbf{v_i} = [v_{i0}, v_{i1}, v_{i2}, v_{i3}, v_{i4}]$, which are used to define the corresponding cubic B-spline basis functions $N^3[\mathbf{u_i}](u)$ and $N^3[\mathbf{v_i}](v)$ in the u and v directions, respectively. The Rule 3 used in [129, 128] is adapted with modification for inferring these knot vectors in a periodic T-mesh:

Rule 3 Assume (u_{i2}, v_{i2}) are the knot coordinates for P_i. To find u_{i3} and u_{i4}, we cast a ray from P_i in the parameter domain: $R(t) = (u_{i2} + t, v_{i2}),\ t > 0$. Then u_{i3} and u_{i4} are defined as the u coordinates of the first two vertical edges intersected by the ray $R(t)$. Note that when the ray crosses the right virtual border, it continues from the left virtual border and then any knot obtained later should add the period T_u. The other knots in $\mathbf{u_i}$ and $\mathbf{v_i}$ are found likewise.

Thus, the u and v knot vectors for P_1 are $[-(d_7 + d_8), -d_8, 0, d_3, d_3 + d_4]$ and $[0, e_1, e_1 + e_2, e_1 + e_2 + e_3, e_1 + e_2 + e_3 + e_4]$. The u and v knot vector for P_3 are $[d_3, d_3 + d_4, d_0, d_0 + d_1, d_0 + d_1 + d_2]$ and $[0, e_1, e_1 + e_8, e_1 + e_5, e_1 + e_5 + e_6 + e_7]$.

Once the knot vectors for each control point are determined, the parametric equation of

the periodic T-spline surface is written:

$$\mathbf{S}(u,v) = \frac{\sum_{i=0}^{n} w_i P_i B_i^*(u,v)}{\sum_{i=0}^{n} w_i B_i^*(u,v)} \tag{5.3}$$

where $P_i = (x_i, y_i, z_i)$ are the control points in R^3 space, w_i are the control point weights, and $B_i^*(u,v)$ are the periodic T-spline blending functions which are defined by:

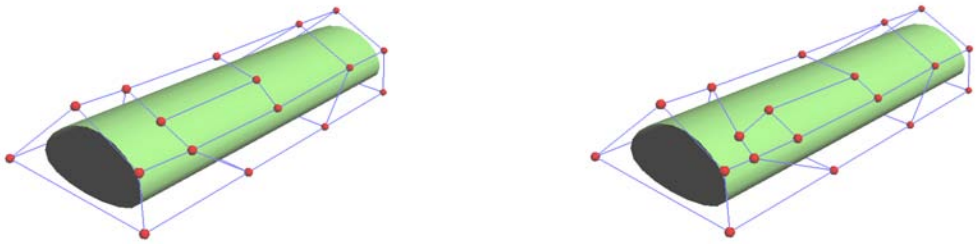
$$B_i^*(u,v) = \tilde{N}^3[\mathbf{u_i}](u) \cdot N^3[\mathbf{v_i}](v).$$

Apparently, this T-spline surface is periodic in the u direction. Similar to periodic NURBS curves or surfaces, if we are just interested in the shape, we can limit the range of parameter u to $[0, T_u)$. Within this domain, most $N^3[\mathbf{u_i} + kT_u](u)$ vanish. Consider $\tilde{N}^3[\mathbf{u_i}](u)$ corresponding to the control point P_i. We have $u_{i5} - u_{i3} \leq 2T_u$ and $u_{i3} - u_{i1} \leq 2T_u$ for $\mathbf{u_i} = [u_{i1}, u_{i2}, u_{i3}, u_{i4}, u_{i5}]$. Thus all $N^3[\mathbf{u_i} - kT_u](u)$ and $N^3[\mathbf{u_i} + kT_u](u)$ vanish in $[0, T_u)$ for $k > 2$ and at most 5 terms do not vanish in domain $[0, T_u)$. Therefore we can write:

$$B_i^*(u,v) = \left(\sum_{k=-2}^{2} N^3[\mathbf{u_i}](u + kT_u)\right) \cdot N^3[\mathbf{v_i}](v) = \left(\sum_{k=-2}^{2} N^3[\mathbf{u_i} + kT_u](u)\right) \cdot N^3[\mathbf{v_i}](v) \tag{5.4}$$

Control point insertion

One important feature of T-splines is local refinement or local knot insertion. This is due to the existence of T-junctions in the T-mesh, which makes it possible to add a single control point to a T-mesh without propagating an entire row or column of control points and without altering the surface. Periodic T-spline surfaces are also able to do local knot insertion. The main idea of the T-spline local refinement algorithm [128] is to maintain the validity of the T-mesh and to ensure that the B-spline basis functions and the control points are properly



(a) A periodic T-spline surface and its T-mesh

(b) The same periodic T-spline surface and its new T-mesh after local insertion of two points

Figure 5.7: Control point insertion for periodic T-spline surfaces.

associated. In the periodic T-spline case, each control point may correspond to several B-spline basis functions. Therefore the original T-spline local refinement algorithm could be adapted to ensure that each control point is properly associated to each corresponding B-spline basis function. Figure 5.7 shows an example of local knot insertion. Figure 5.7(a) is a periodic T-spline surface together with its T-mesh. Figure 5.7(b) is the same periodic T-spline surface but the T-mesh changes due to the local insertion of two points.

5.3 Parameterizing tubular meshes

In this section, we consider the problem of parameterizing triangular meshes that have tubular shapes. The obtained parameterization should be suitable for the process of periodic T-spline surface fitting. Denote a tubular triangular mesh by $\mathbf{T}(V, E)$ where $V = \{\mathbf{d_1}, \mathbf{d_2}, \cdots, \mathbf{d_m}\}$ is a vertex list and $E = \{\mathbf{e_1}, \mathbf{e_2}, \cdots, \mathbf{e_n}\}$ is a directed edge list, and a parameter domain by D corresponding to the lateral surface of a unit cylinder. Our goal is to find a parameterization ψ_v which establishes a mapping between vertex $\mathbf{d_i} = (x_i, y_i, z_i)$ and a parameter pair (u_i, v_i) that forms a point $\mathbf{t_i} = (u_i, v_i)$ in D, $i = 1, 2, \cdots, m$, and thus a mapping between vertex $\mathbf{d_i}$ and a point on the cylinder surface. The edges of $\mathbf{T}$ are correspondingly mapped as well. The mapping of the edges onto the cylinder surface

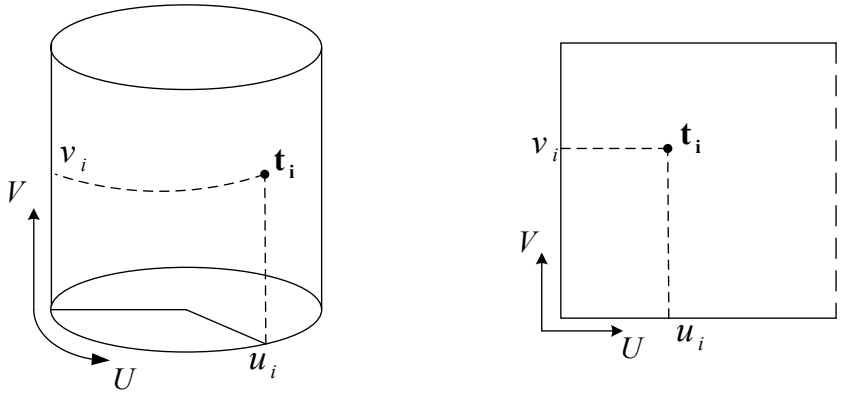


Figure 5.8: The parameter domain for tubular meshes.

forms a curved triangular mesh on the surface. Here, we define the unit cylinder to be a cylinder whose height and perimeter of the cross sectional circle are both 1 and thus $D = [0, 1) \times [0, 1]$. The u direction is closed and the v direction is open, as illustrated on the left of Figure 5.8. Figure 5.8 (right) shows a part of domain D obtained by unfolding the cylinder to a plane. For the parameter pair (u_i, v_i), it can be understood that u_i corresponds to the arc length along the rotational direction of the cylinder and v_i corresponds to the height value along the axial direction of the cylinder. Having two individual closed boundaries is one of the most salient characteristics that differentiate a cylinder domain from an open disk domain. It is essential that the parameterization method should map the vertices on the bottom boundary of the tubular mesh to the boundary $v = 0$ in D and the vertices on the top boundary to the boundary $v = 1$ in D. The rest vertices of the mesh should be mapped to the remainder part of D, namely $[0, 1) \times (0, 1)$.

To parameterize a tubular mesh, one possible solution could be first splitting the tubular mesh along a path consisting of a set of vertices in the axial direction, making it homeomorphic to a topological disk. The cutting path then becomes the boundary of the new mesh. After that, the parameters for the new mesh could be obtained by applying one of the existing parameterization techniques for open meshes. However, such an approach would incur some inconveniences or troubles. The parameters for the vertices on the newly formed boundary should be carefully chosen in order to avoid discontinuity of the param-

eterization. The cutting path should also be properly selected, since some vertices might be more suitable to be boundary vertices than the others. Cursory pick of the cutting path would likely increase the distortion of the result parameterization. Furthermore, no matter which cutting path is finally selected, it is usually mapped onto a straight line in the parameter domain. Given the fact that the cutting path on the mesh is usually in a zigzag shape, it is likely to introduce undesirable effects when the cutting path is mapped to a straight boundary of the parameter domain. There are a few works published for tubular mesh parameterization. Zöckler et al. provide a parameterization method in [167], which involves first cutting the mesh and then stitching it together after a parameterization is calculated. Although the parameters are optimized again after the stitching step, the performance of the method still largely depends on finding a good cutting path. Huysmans et al. report another approch [78], in which no cutting of the mesh is required; however, in order to compute the parameterization, a group of progressive meshes have to be constructed, from which the parameters are gradually optimized.

In addition, to generate a parameterization with low distortion, the vertices on either boundary of the tubular mesh should be assigned with reasonable parameters. There has to be certain correspondence between the parameters for the two boundaries. Both boundaries should have a consistent direction for the parameters, either clockwise or counterclockwise. Moreover, two vertices on the opposite boundaries should be assigned with similar u parameters if they are almost along the same axis on the tubular mesh. As an example, for the vertices $\mathbf{d_i}$ and $\mathbf{d_j}$ of the tubular mesh shown in the left of Figure 5.9, the positions of their mappings $\mathbf{t_i}$ and $\mathbf{t_j}$ should generally be aligned in the parameter domain shown in the middle of Figure 5.9. Otherwise, the result triangulation on the parameter domain would be twisted, as in the example given in the right of Figure 5.9. However, finding correct correspondence between the two boundaries is not straightforward and the situation could become quite complicated.

Another difficulty for tubular mesh parameterization, which the conventional open mesh

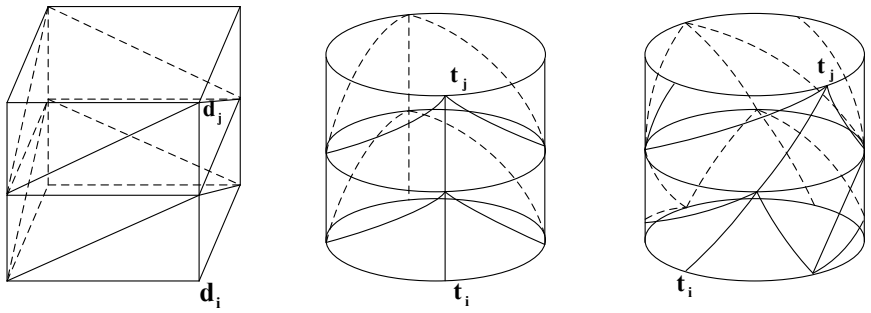


Figure 5.9: A tubular triangular mesh (left), an expected parameterization (middle) and a twisted parameterization (right).

parameterization does not have, is that when we construct equations for determining the parameters of the vertices, the parameters (u_i, v_i) of some vertices should be replaced by $(u_i + 1, v_i)$ or $(u_i - 1, v_i)$ to respect the continuity in a local area. Here 1 is the period for the U parameter direction. However, it is not easy to determine whether (u_i, v_i), $(u_i + 1, v_i)$ or $(u_i - 1, v_i)$ should be used for constructing an equation.

5.3.1 Overview of edge based parameterization

To overcome the above-mentioned difficulties, we here propose a new parameterization method. The main idea of our method is that the edges, rather than the vertices of a mesh, are treated as the target for parameterization. Therefore we call our method *edge based parameterization*. Since each edge is a vector that represents the location offset between two vertices, the offset values are unique no matter where the origin is. Therefore the messy problems such as finding correspondence of vertices on two boundaries and deciding the suitable parameters from a number of possibilities could be avoided. An edge base parameterization ψ_e maps each directed edge $\mathbf{e_i}$ in E to a curve segment on the cylinder and thus a 2D edge vector $\mathbf{u_i}(u_i, v_i)$ in the domain D, $i = 1, 2, \cdots, n$. Note that here u_i and v_i are the u and v components of vector $\mathbf{u_i}$. Once ψ_e is constructed, the conventional vertex parameterization ψ_v for the vertices in V of the tubular mesh can be inferred by specifying an arbitrary boundary vertex as the origin of the (u, v) coordinate system.

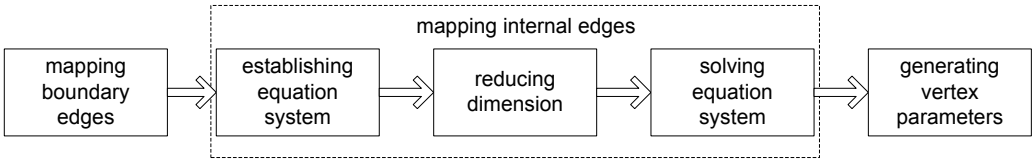


Figure 5.10: A flowchart of our parameterization approach.

A flowchart for our parameterization approach is given in Figure 5.10. The whole parameterization process is divided into three steps. The first step is to determine parameters for the edges on the two boundaries of the mesh. The second step is to calculate parameters of the internal edges. This step involves establishing and solving a linear equation system based on the geometry relations that the internal edges should satisfy. To generate a smooth parameterization that depends on the edges and vertices of the mesh, mean value coordinates [42] are adopted as the weights for the linear combination of adjacent edges. Moreover, strategies are also proposed to reduce the dimension of the equation system to alleviate the computation load. The third step is to convert edge parameterization to vertex parameterization. The details of these steps are elaborated in the next three subsections.

5.3.2 Parameterizing boundary edges

Our method first assigns parameters to the edges on two boundaries of the tubular mesh, i.e. the boundary edges. The edges that are not on any boundary or only have one vertex on the boundary are defined as internal edges. For clarification, in the directed edge list E, we denote $\mathbf{e_1}, \mathbf{e_2}, \cdots, \mathbf{e_d}$ to be internal edges and $\mathbf{e_{d+1}}, \mathbf{e_{d+2}}, \cdots, \mathbf{e_n}$ to be boundary edges without losing any generality. For each directed edge in the mesh, we have two choices on its specific direction. An edge $\mathbf{e_i}$ that connects $\mathbf{d_{i1}}$ and $\mathbf{d_{i2}}$ can originate from either $\mathbf{d_{i1}}$ or $\mathbf{d_{i2}}$. For the internal edges, we let their direction be arbitrarily chosen. For the boundary edges, we make some constraints such that edges belonging to the same boundary have same direction, and edges belonging to different boundaries have opposite directions. These constraints would simplify the description of our method. If certain mesh

data structure such as half-edge [155] is adopted for representation, we can simply achieve the above arrangements by letting the direction of the boundary edges conform with that of the corresponding boundary half edges.

For the boundary edges, the v components would always be zero. We determine the u components of the boundary edges using the chordal length approach. Because the representations of vectors are invariant under translation, edge parameterization can be carried out without specifying an origin for the coordinates system. Therefore, depending on which boundary it belongs to, the boundary edge $\mathbf{e_i}$ is parameterized as either $(l_i/L_1, 0)$ or $(-l_i/L_2, 0)$, where l_i is the length of the edge. L_1 and L_2 are the sum of lengths of the edges on the two respective boundaries and are used to normalize the parameters for boundary edges. This makes the lengths of parameter range for both boundaries the same and equal 1. Note that at the end of this stage, the parameters for either boundary have been fixed but the relative position of the two boundaries is still undecided yet.

5.3.3 Computing internal edge parameters

We now describe how to compute the parameters for the internal edges $\mathbf{e_1}, \mathbf{e_2}, \cdots, \mathbf{e_d}$. We set up a system of linear equations that the edge parameters should satisfy. The edge parameters are then obtained by solving the equations.

First, each triangle face in the tubular mesh corresponds to a triangle in the parameter domain. Suppose $\mathbf{u_i}, \mathbf{u_j}, \mathbf{u_k}$ are the parameters for three directed edges that form a triangle in the tubular mesh. Then $\mathbf{u_i}, \mathbf{u_j}, \mathbf{u_k}$ form a triangle in the parameter domain, as shown on the left part of Figure 5.11. Thus $\mathbf{u_i}, \mathbf{u_j}, \mathbf{u_k}$ should satisfy the following equation, which we call the face related equation:

$$\mathbf{u_i} + [\mathbf{u_j}] + [\mathbf{u_k}] = 0 \tag{5.5}$$

Here, the operator $[\,]$ is introduced, which changes the direction of a vector to its opposite when necessary, to make sure that after the adjustment, the edges corresponding to the three

vectors have different starting vertices.

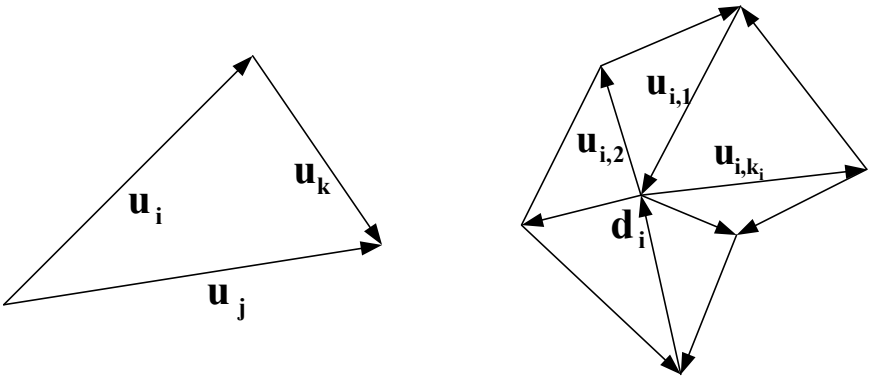


Figure 5.11: Building equations from face relation and vertex relation.

Second, for each internal vertex, we look at its 1-ring neighborhood. To generate a good parameterization, we use mean values coordinates [42]. That is, we let the parameter of the internal vertex be represented as a linear combination of the parameters of its surrounding vertices in the 1-ring neighborhood (See Equation (4.6)). After a simple arrangement, this turns out to be a constraint on the edges incident to $\mathbf{d_i}$ (see the right part of Figure 5.11), which we call the vertex related equation:

$$\sum_{j=1}^{k_i} \lambda_{ij}[\mathbf{u_{i,j}}] = 0 \tag{5.6}$$

where k_i is the valence of the interval vertex $\mathbf{d_i}$ and λ_{ij} is the mean value coordinates of $\mathbf{d_i}$ with respect to its j-th neighboring vertex $\mathbf{d_j}$. Again, here $\mathbf{u_{i,j}}$ is first passed into operator $[\,]$ in order to guarantee that the edge corresponding to $[\mathbf{u_{i,j}}]$ emits from $\mathbf{d_i}$.

So far we have established n_1 face related equations and n_2 vertex related equations, where n_1 is the number of faces and n_2 is the number of internal vertices. These equations contain n_3 unknowns, where n_3 is the number of internal edges. A careful analysis using Euler's formula shows that $n_1 + n_2 = n_3$, which means the number of the equations equals the number of unknowns.

However, it can be found that the face related equations are linearly dependent. This is

because the sums of the edge vectors of two boundaries have the same length but opposite direction. To fix this problem, we select one internal edge, assign appropriate parameters to it, and remove one equation derived from a face that contains the edge. To choose the edge, we compute the angle θ between all the boundary edges and the edges adjacent to them and pick the edge $\mathbf{e}^*$ whose θ is closest to $\frac{1}{2}\pi$. The parameters for $\mathbf{e}^*$ is set to be $(\cos\theta, \sin\theta)$. In this way, we will have $n_3 - 1$ equations with $n_3 - 1$ unknowns. Solving the equations gives the parameters for all the edges.

Dimension reduction

In a triangular mesh, the number of edges is about three times of the number of vertices. When the edges instead of the vertices become the unknown elements for parameterization, the dimension of the optimization problem is inevitably increased. The scale of the equation system accordingly becomes three times larger and the computational cost is therefore higher. Therefore we need a method that can be used to reduce the number of the equations and the unknowns.

It is observed that the established equation system contains a number of equations, each of which corresponds to a face and thus has at most three edges that have non-zero coefficients. Suppose $\mathbf{u_i}$ is the parameter for an unknown edge in Equation (5.5). We can always rearrange such a face related equation into $\mathbf{u_i} = [\mathbf{u_j}] + [\mathbf{u_k}]$. From this expression, $\mathbf{u_i}$ can be understood as depending on $\mathbf{u_j}$ and $\mathbf{u_k}$. This expression makes it possible to substitute the appearance of $\mathbf{u_i}$ in all the other equations of the equation system by $[\mathbf{u_j}]$ and $[\mathbf{u_k}]$. Whenever such an unknown is substituted, the size of the linear system is reduced by one.

Thus, by utilizing all the face related equations for unknown edge substitution, we expect to eliminate some of the unknowns as well as the equations from the original equation system. We tag $\mathbf{u_i}$ (and its corresponding edge) as dependent if it is substituted by other unknowns. Otherwise, $\mathbf{u_i}$ is tagged as independent. After that, we solve a new equation

system which contains only the independent unknowns. Then, the remaining problem is how to tag each unknown edge in a triangular mesh as either dependent or independent. While pure algebraic approaches could be difficult due to the complicated connectivity of the mesh, in the following we present a geometric based approach.

We start from an initial mesh, where the parameters for the boundary edges and one preset edge $\mathbf{e}^*$ are given. All the unknown edges are untagged at this moment. Now we examine all the triangle faces in the mesh. We search for any face that contains only one untagged edge by priority. By reexpressing the face related equation, we tag such an edge as a dependent edge that depends on the other two edges in the face. We establish a dependency table in order to manage the relations among edges, in which we store for each edge its tag and the edges it depends on. Whenever no more faces contain only one untagged edge, we alternatively pick one face that contains two untagged edges. We set one of these edges as independent, and the other edge as dependent. This process continues until each unknown edge in the mesh is given a tag. We clarify this approach by an example given in Figure 5.12, in which a part of a triangular mesh is shown. Suppose $\mathbf{e_1}, \mathbf{e_2}, \mathbf{e_3}$ are already tagged or their parameters are already given. Using the above strategy, $\mathbf{e_4}, \mathbf{e_5}, \mathbf{e_6}, \mathbf{e_7}$ can then be given a tag respectively as well. A possible dependency table for $\mathbf{e_4}, \mathbf{e_5}, \mathbf{e_6}, \mathbf{e_7}$ is given in Table 5.1.

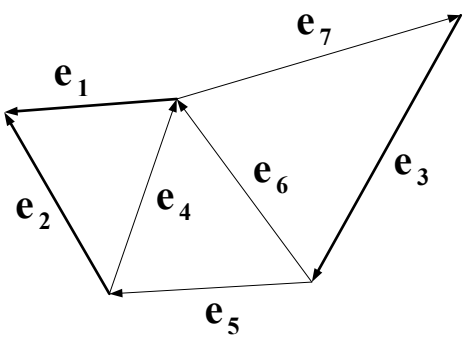


Figure 5.12: A local part of a triangular mesh.

Sometimes, not all the face related equations are used during this process. It could

Table 5.1: The dependency table for some edges in Figure 5.12.

edge	tag	dependency
$\mathbf{e_4}$	dependent	$\mathbf{e_1}$, $\mathbf{e_2}$
$\mathbf{e_5}$	independent	-
$\mathbf{e_6}$	dependent	$\mathbf{e_4}$, $\mathbf{e_5}$
$\mathbf{e_7}$	dependent	$\mathbf{e_3}$, $\mathbf{e_6}$

happen that all the three edges of a face are already tagged before the face is examined. If we leave such faces untreated, the number of independent edges and the number of the equations would be different in the resulting equation system. In order to fix this problem, we iteratively substitute all the dependent unknowns in this face related equation using the dependency table, until the equation no longer contains any dependent unknown. Then, we re-tag an arbitrary independent edge in this equation as dependent and update the dependency table using this equation.

Finally, we remove the face related equations from the equation system and substitute all the dependent edges using the dependency table. Thus we obtain a much smaller linear equation system, whose size is the same as the number of internal vertices. When this equation system is solved, we can then calculate the parameters for those dependent edges. Therefore all the edges in the mesh are parameterized.

5.3.4 Inferring vertex parameterization

Once we have obtained the edge parameterization ψ_e for a triangular mesh of tubular topological type, we can convert the edge parameterization ψ_e to an equivalent vertex parameterization ψ_v, where each vertex $\mathbf{d_i}$ is associated with a pair of parameter values.

First we arbitrarily pick a vertex on either boundary of the mesh and assign to it parameters $(0, 0)$. After that, parameters for the remaining vertices are obtained by traversing the triangular mesh under the breadth-first search algorithm ([24]). We set the previously picked vertex as the root vertex and put it into an empty queue structure. We then keep

retrieving a vertex from the top of the queue and attempt to compute parameters for all of its unattended neighboring vertices. From a vertex $\mathbf{d_i}$, the parameter for its neighboring vertex $\mathbf{d_j}$ is computed by either adding or subtracting to the parameter of $\mathbf{d_i}$ the parameter of the edge that connects $\mathbf{d_i}$ and $\mathbf{d_j}$, depending on the direction of that edge. We require that the u direction parameter for any vertex be in the $[0, 1)$ domain. If u_i is outside $[0, 1)$ after being computed from its neighbor, we try to find an integer k such that $u_i + k \in [0, 1)$. In that case, we use $u_i + k$ as the u direction parameter for $\mathbf{d_i}$ instead.

We put all the newly parameterized vertices into the bottom of the queue and repeatedly deal with the top vertex in the queue. A vertex goes out of the queue when attempts have been made to parameterize all of its neighboring vertices. We continues this process until the queue finally becomes empty, which means that all the vertices of the triangular mesh have been given proper parameters. Finally, after the above procedure stops, we normalize the v coordinates of all the vertices by a translation or/and a scaling to make the range be [0,1]. This yields a vertex parameterization in $[0, 1) \times [0, 1]$ for the tubular mesh.

5.3.5 Performance of the algorithm

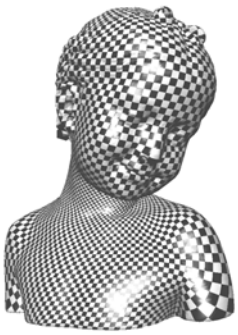
We apply our algorithm to a tubular triangular mesh of the bimba model with two holes as shown in Figure 5.13(a). The parameterization result is given in Figure 5.13(b). By texturing the mesh with a checkboard pattern under the guidance of the resulting parameterization (see Figure 5.14), we can see that the parameters are evenly distributed and have quite low geometric distortion.

We have also tested models of different types to demonstrate the performance of the edge based parameterization. The geometry information of the models is given in Table 5.2. The experimental results are displayed in Figure 5.15. The first column of Figure 5.15 shows the surface view of the models. The second column shows the mesh view of the models. All the triangular meshes used here have the same topological type as a cylinder and there are two closed boundaries for each of these meshes.

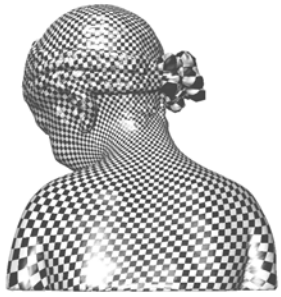
(a) A tubular mesh of the bimba model with two holes

(b) The parameterization result on a cylinder domain

Figure 5.13: An example of the edge based parameterization approach.



(a) Front view



(b) Back view

Figure 5.14: Texture mapping with the checkboard pattern.

We apply the edge based parameterization method to these models, and the results of parameterization are shown in the third column of Figure 5.15. We project the parameterization onto the cylinder through a function: $(u, v) \mapsto (\frac{\cos 2\pi u}{2\pi}, \frac{\sin 2\pi u}{2\pi}, v)$ for a clearer visualization. It can be seen that although we do not make any effort on aligning the parameters of the two boundaries, correct correspondence can be automatically obtained.

Another way of visualizing the result is provided in the fourth column of Figure. 5.15. Here each parameterization is cut open along a path of vertices from one boundary to another and then displayed in a plane. The edges shown in red lines connect a right-end vertex to a left-end one. The edges shown in dash lines form the cutting path. Finally, in the fifth column of Figure. 5.15, we demonstrate the effect of mapping a checkboard texture to the

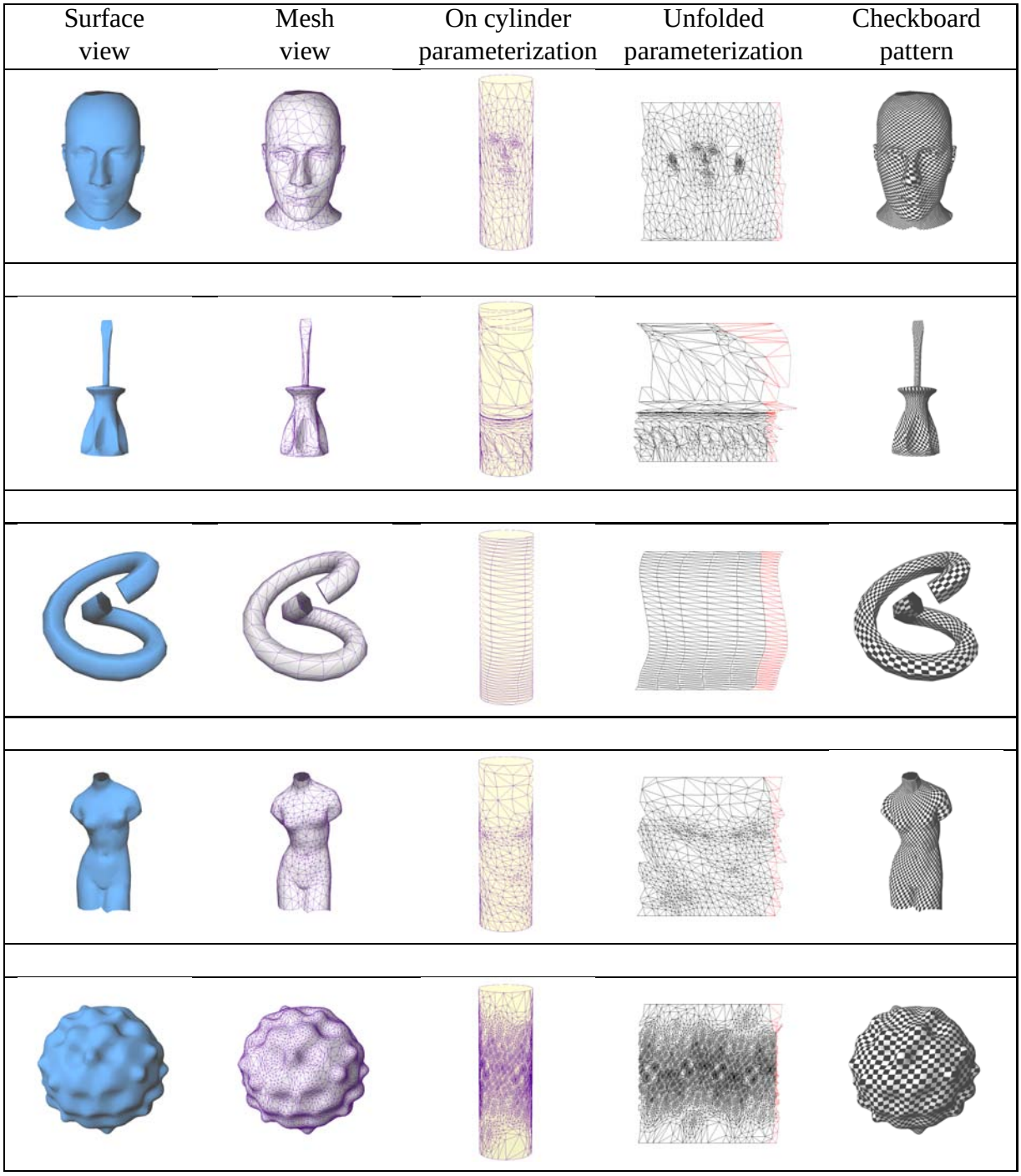


Figure 5.15: Examples of edge based parameterization.

Table 5.2: Information of the models.

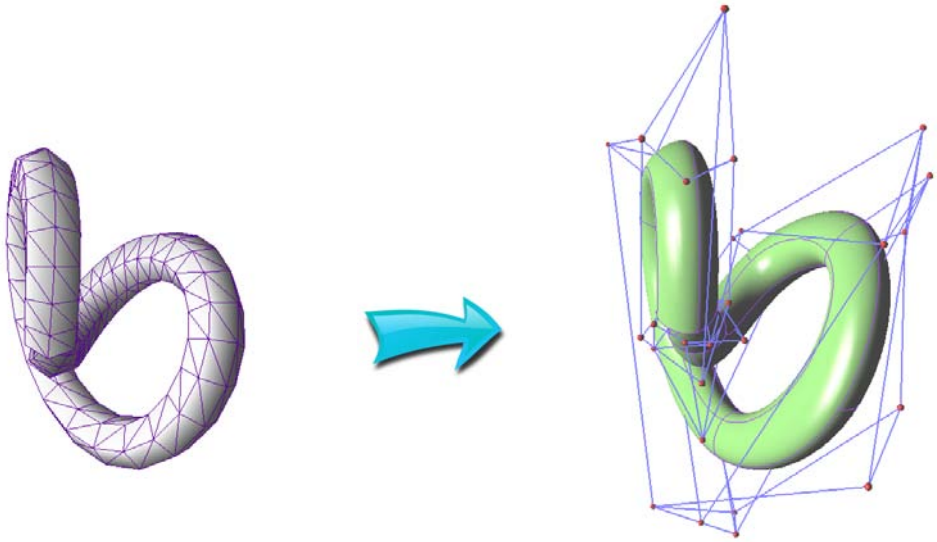
Model	Vertex number	Face number	Edge number
bimba	8414	16705	25119
mannequin	680	1328	2008
screw driver	514	999	1513
knot	234	456	690
Venus	710	1389	2099
bumpy sphere	5610	11181	16791

tubular meshes, using the resulting parameterization. It can be seen that the distortion of the parameterizations is very low.

5.4 Periodic T-spline surface fitting

In this section, we discuss how to automatically fit a periodic T-spline surface to a tubular mesh. Since a tubular mesh is of the same topological type as a periodic T-spline surface defined in Equation (5.3), it is more suitable to approximate the tubular mesh by a periodic T-spline surface. Figure 5.16(a) shows a piecewise linear tubular mesh, comprised of a number of triangle faces. We want to find a smooth surface defined by a periodic T-spline as shown in Figure 5.16(b) together with its T-mesh, which approximates the tubular mesh. Mathematically, this problem can be formulated as follows: given a tubular triangular mesh $\mathbf{T}$ which has a vertex set $V = \{\mathbf{d_1}, \mathbf{d_2}, \cdots, \mathbf{d_m}\}$, we want to find a periodic T-spline surface $\mathbf{S}(u, v)$ that approximates each vertex of the tubular mesh within an error tolerance ε, i.e. $\mathrm{dist}(\mathbf{d_i}, \mathbf{S}(u, v)) \leq \varepsilon$, $i = 1, 2, \cdots, m$.

Here we propose an algorithm to solve the above problem. The algorithm consists of several steps (see Figure 5.17). First, in order to establish a relationship between the tubular mesh and periodic T-spline surfaces, a parameterization step is required to map the tubular mesh to the parameter domain of the surface. Next, an initial T-mesh is constructed to startup the adaptive surface fitting procedure. Then, the algorithm computes a geometrically optimal T-spline surface that best approximates the tubular mesh under the topology



(a) A tubular mesh

(b) A periodic T-spline surface that approximates the mesh

Figure 5.16: Fitting a periodic T-spline surface to a tubular mesh.

setting of the current T-mesh. After that, we check the quality of the obtained periodic T-spline surface and further refine the T-mesh at the locations where the fitting quality is not satisfactory. We keep improving the fitting result by iteratively carrying out the geometry optimization and mesh structure refinement steps until the approximation error evaluated at any vertex is not greater than the prescribed error tolerance. Once the error tolerance is met, the whole process can be terminated at this stage, or we can optionally choose to improve the previously calculated parameterization of the tubular mesh and start over the fitting procedure again in order to probably achieve better periodic T-spline surface fitting.

In this proposed algorithm, we compute the parameterization for a tubular mesh using the edge based parameterization method described in Section 5.3.1 and the reparameterization using a technique described in Chapter 4. Therefore, in the remainder of this section, we just describe the steps for surface initialization, geometry optimization and mesh refinement. In addition, some examples are also provided at the end of the section.

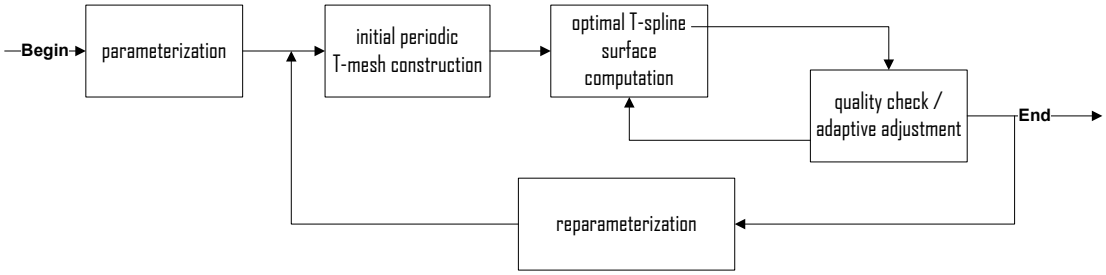


Figure 5.17: The flowchart of periodic T-spline surface fitting.

5.4.1 Initialization

In this step, we construct an initial T-mesh which the adaptive surface fitting process can begin with. First a suitable domain for the surface should be decided. Using the edge based parameterization approach, a mapping is established from the given tubular mesh to a parameter domain $[0,1)\times[0,1]$, which is also the domain of the periodic T-spline surface. Each vertex $\mathbf{d_i}$ corresponds to a parameter pair (u_i, v_i) where $0 \leq u_i < 1, 0 \leq v_i \leq 1$.

Second, we let the pre-image of the T-mesh in the parameter domain be a regular $dim_u \times dim_v$ grid at the first stage. The knot vectors along the $u-$ and $v-$directions associated with the T-mesh are both uniformly divided in the domain. That is, the u knot vector is $\{0, \frac{1}{dim_u-1}, \frac{2}{dim_u-1}, \cdots, \frac{dim_u-2}{dim_u-1}\}$ and the v knot vector is $\{-0.02, -0.01, 0, \frac{1}{dim_v-3}, \frac{2}{dim_v-3}, \cdots, 1, 1.01, 1.02\}$. For simplicity, the sizes of the control grid, dim_u and dim_v, are often set to a number between 4 and 10 depending on the complexity of the input tubular mesh. An example of the pre-image of a 4×4 periodic T-mesh is shown in Figure 5.18. The 4 control points in the rightmost column are virtual duplicates of those in the leftmost column and the domain of the surface is the region inside of the red bounding box.

Finally, we let the control point weight vector $\mathbf{W} = (w_0, w_1, \cdots, w_n)^T$ take the initial setting of $(1, 1, \cdots, 1)^T$. The initial pre-image of the T-mesh and the weight vector together define a class of periodic T-spline surfaces that have the same T-mesh topological structure and the same weight vector. On the other hand, the geometric coordinates of the control points are still unassigned at this moment and are left to be optimized in the next step.

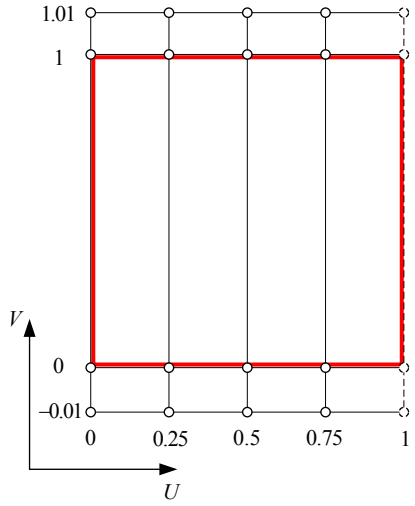


Figure 5.18: The pre-image of a 4×4 periodic T-mesh.

5.4.2 Optimizing the geometry of the surface

When the T-mesh topology (i.e. the layout and the connectivity of the control points in the pre-image) and the control point weights $\mathbf{W}$ are determined, a class of periodic T-spline surfaces is defined. Each individual surface in this class differs from the others only in the geometry of the control points. Therefore the problem is now becoming how to find the surface within the class that best approximates the tubular mesh. This can be done by finding the control points using the least-squares approach.

Note that our initial setting for the T-mesh and the control point weights guarantees that $\sum_{i=0}^{n} w_i B_i^*(u,v) \equiv 1$. In fact, a periodic T-spline surface with such setting is a standard T-spline and thus a parametric polynomial surface. During the fitting process we constrain the surfaces to remain polynomial via the T-spline knot insertion algorithm and thus the surface representation in Equation (5.3) can be simplified in this situation:

$$\mathbf{S}(u,v) = \sum_{i=0}^{n} w_i P_i B_i^*(u,v) \tag{5.7}$$

The following objective function is used as the target for optimization:

$$F(P_0, P_1, \cdots, P_n) = \sum_{j=1}^{m} \parallel \mathbf{S}(u_j, v_j) - \mathbf{d_j} \parallel^2 + \sigma J_{fair}(S) \tag{5.8}$$

This function contains two parts. The first part controls the precision of the surface, which is the squares of sum of the distances between the vertices of the mesh and the periodic T-spline surface. We use the parametric distance $\parallel \mathbf{S}(u_j, v_j) - \mathbf{d_j} \parallel$ to represent $\mathrm{dist}(\mathbf{d_j}, \mathbf{S}(u, v))$. The second part is an energy term that measures the shape fairness of the surface. σ is a factor that balances the two parts: precision and fairness. We choose the simple thin plate energy as $J_{fair}(\mathbf{S})$, which is expressed as:

$$J_{fair}(\mathbf{S}) = \int\int (\mathbf{S}_{uu}^2(u, v) + 2\mathbf{S}_{uv}^2(u, v) + \mathbf{S}_{vv}^2(u, v))dudv \tag{5.9}$$

where $\mathbf{S}_{uu}$, $\mathbf{S}_{uv}$, $\mathbf{S}_{vv}$ are the second order partial derivatives of $\mathbf{S}(u, v)$.

To find $P_0, P_1, \cdots, P_n$ that minimize the objective function, we let all the partial derivatives of the objective function with respective to each control point equal to zero:

$$\frac{\partial F}{\partial P_g} = 0, \quad g = 0, 1, \cdots, n \tag{5.10}$$

which leads to

$$\begin{aligned} &\sum_{i=0}^{n} w_i \left(\begin{array}{l} \sum_{j=1}^{m} B_i^*(u_j, v_j) B_g^*(u_j, v_j) + \\ +\sigma \int\int \left(\begin{array}{l} B_{iuu}^*(u, v) B_{guu}^*(u, v) + \\ +2B_{iuv}^*(u, v) B_{guv}^*(u, v) + \\ +B_{ivv}^*(u, v) B_{gvv}^*(u, v) \end{array} \right) dudv \end{array} \right) P_i \\ &= \sum_{j=1}^{m} \mathbf{d_j} B_g^*(u_j, v_j) \end{aligned} \tag{5.11}$$

Combining these equations together forms a linear system whose solution gives the control

points of the optimal T-spline surface.

5.4.3 T-mesh refinement

After a periodic T-spline surface is computed under the current setting, we need to check whether the approximation meets the requirement. The approximation error, in term of the parametric distance of each vertex $\mathbf{d_i}$ of the mesh to the T-spline surface: $\|\mathbf{S}(u_i, v_i) - \mathbf{d_i}\|$, is evaluated. If all the distances are below the tolerance ε, the T-spline surface is considered qualified and the algorithm moves to the next step. Otherwise, the regions that contain one or more violating vertices whose distances are greater than the tolerance need to be refined.

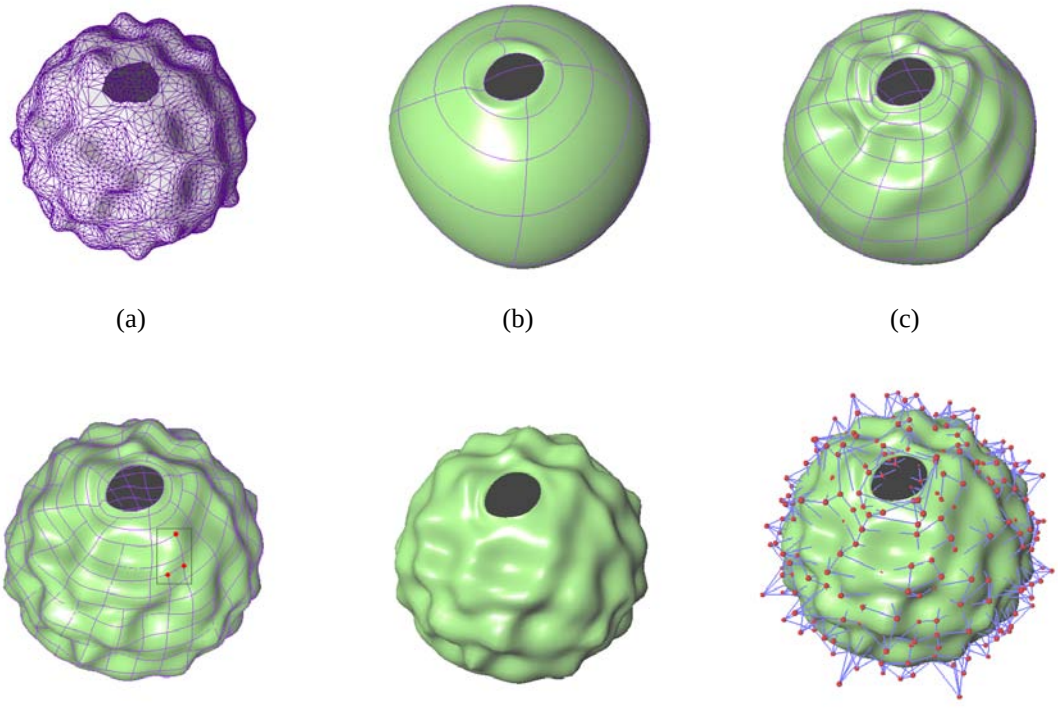
We refine the faces of the T-mesh, which correspond to the regions containing violating vertices, by adding a new edge into that face and splitting the face into two halves of equal size. To add an edge, we simply insert two end points of the edge into the T-mesh, using the local refinement method for periodic T-spline surfaces. The edge could be either horizontal or vertical, depending on whether the height or the width of the region is larger.

After the T-mesh is updated, we obtain a new setting for the T-mesh and the control point weights, which define a new class of T-spline surfaces. Since the new T-mesh structure and the control point weights are generated by the T-spline local refinement algorithm, the resulting class of periodic T-spline surfaces remains to be standard or semi-standard. Therefore, surfaces in this class are still polynomial. The control points of the optimal T-spline in this class can be computed using the geometry optimization step described in the preceding subsection.

5.4.4 Examples and discussions

This section presents two examples to demonstrate the proposed periodic T-spline surface fitting algorithm. The first example is a bumpy model (see Figure 5.19(a)) which is a tubu-

lar triangular mesh and whose basic information is listed in Table 5.3. When we use the fitting algorithm to approximate the bumpy model, the error tolerance for approximation is set to be 0.5% of the size of the model and the fairness factor is set to be 0.00001. Figure 5.19(b) and Figure 5.19(c) show two periodic T-spline surfaces which are the intermediate results after 3 and 6 iterations respectively, with the pre-images mapped onto the surfaces. It can be seen that the structures of the T-meshes are not sufficiently sophisticated to represent all the geometrical features of the original tubular mesh and the surfaces are not well approximated. After 11 iterations, a satisfactory result is obtained, which is shown in Figure 5.19(d) with the pre-image mapped onto the surface. In Figure 5.19(d), a window is drawn to highlight three T-junction points, which are shown in red color, to illustrate the existence of T-junctions. Figure 5.19(e) and Figure 5.19(f) show the final periodic T-spline surface without and with the T-mesh. Refer to Table 5.3 for the detailed information about the final periodic T-spline surface.



(a) (b) (c)

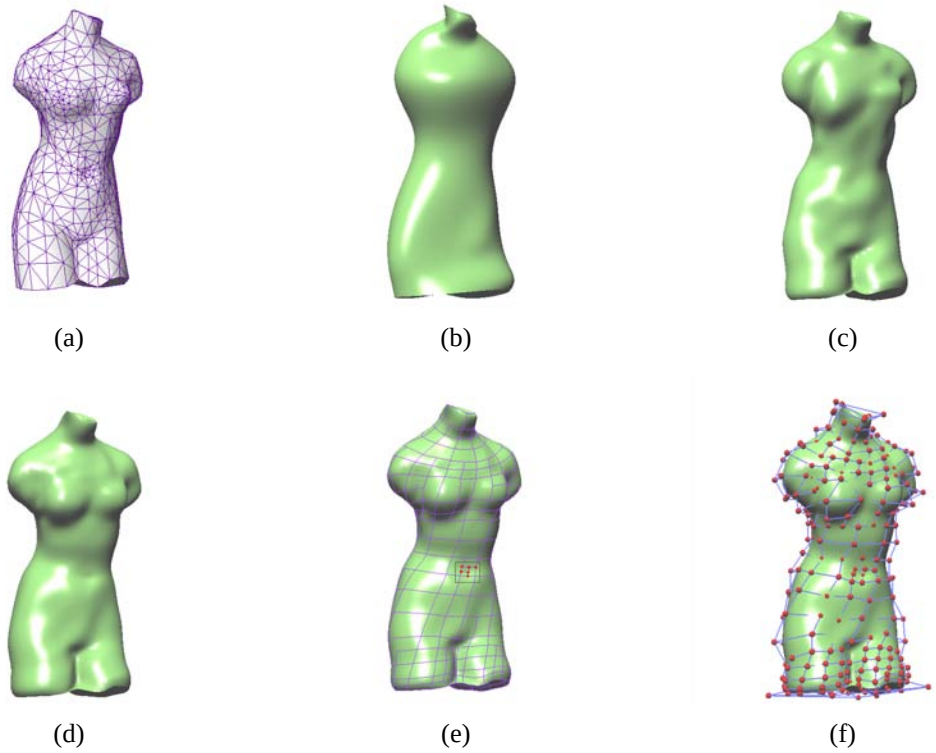
(d) (e) (f)

Figure 5.19: Fitting a bumpy model.

Table 5.3: Statistics of the tubular mesh in Figure 5.19(a) and the final resulting periodic T-spline surface.

Mesh	Number of vertices	5610
	Number of faces	11181
Periodic T-spline surface fitting	Total iterations for fitting	10
	Number of knot in the $u-$direction	33
	Number of knot in the $v-$direction	32
	Number of control points	570
	Maximum approximation error	0.44%
	Average approximation error	0.03%

The second example is to demonstrate the algorithm in approximating tubular meshes that have relatively low quality (i.e., the number of vertices is insufficient compared to the complexity of the geometry they represent). It can be seen that the algorithm is capable of handling these meshes by choosing a smaller error tolerance. The model we use in the second example is the Venus shown in Figure 5.20(a), which has 710 vertices and 1389 faces. If we choose the error tolerance to be 0.5% of the scale of the model, the fitting periodic T-spline surface can be quickly obtained. However, the resulting surface (see Figure 5.20(b)) does not respect the geometrical features of the mesh very well although the error tolerance is already met at all the vertices. This is because merely achieving the approximation error at the vertices of the mesh does not guarantee the overall surface quality, especially when the density of the vertices of the model is not high enough. The resulting surface (shown Figure 5.20(c)) could get better if the error tolerance is changed to 0.05% of the scale of the model, but it is still not good enough. Finally, if we adjust the error tolerance to be 0.02% of the scale of the model, this time we get a visually satisfactory result (see Figure 5.20(d)). The final surface has 342 control points and is obtained in 10 iterations. Figure 5.20(e) and Figure 5.20(f) show the surface with the mapped pre-image and the T-mesh, respectively.



(a) (b) (c)

(d) (e) (f)

Figure 5.20: Fitting the Venus model.

5.5 Summary

This chapter discusses the definitions of periodic B-splines and the representation of periodic T-spline surfaces. Periodic T-spline surfaces are a good geometry representation for defining a tubular-shaped object in a single domain.

We then present an edge based parameterization method that is especially suitable for parameterizing tubular triangular meshes. Instead of parameterizing vertices directly, we first compute the parameters of the edges. The mean value coordinates are used to parameterize the edges. After that, the parameterization for the vertices is derived. Since our method is based on intrinsic geometry of the mesh, there is no need to perform cutting for parameterization and a low distortion can be achieved. We demonstrated our approach by several examples of different types. The effectiveness of the method suggests that the idea of treating edges as the main target may also be applicable in other mesh related problems

in geometry processing. From the parameterization, further applications such as texture mapping and surface fitting of the tubular meshes could be developed.

Finally, we provide an algorithm to convert a tubular triangular mesh into a periodic T-spline surface. In the algorithm, we parameterize the tubular mesh using the edge based parameterization method and build the target periodic T-spline surface in an adaptive manner. We start with an initial periodic T-spline of a simple mesh structure and iteratively refine the T-mesh at locations where the T-spline approximation is poor and re-compute the control points for the optimal T-spline surface. As a result, the output periodic T-spline surface approximates the tubular mesh within the given error tolerance.

Chapter 6

Conversion between T-splines and Hierarchical NURBS

6.1 Introduction

This chapter discusses conversion between two parametric surface representations: T-splines and hierarchical NURBS. T-splines and hierarchical NURBS are both the generalization of NURBS and allow the surface to be refined locally. Hierarchical NURBS is desirable for multiresolution modeling, and it is more intuitive to use T-splines for interactive designing. Since each representation has its own strength, in practice it could be useful to be able to convert one representation into another.

Note that it is always possible to convert a T-spline surface or a hierarchical NURBS surface into a NURBS surface (called the underlying NURBS surface) with finer control mesh. There are also algorithms to construct a hierarchical NURBS surface [47] or a T-spline surface [128] to approximate a given NURBS surface. If we set the approximate error to be zero, the approximate algorithms will lead to exact conversion from a NURBS surface to a T-spline or hierarchical NURBS surface. Using these algorithms, one can achieve the conversion between hierarchical NURBS and T-splines through the underlying

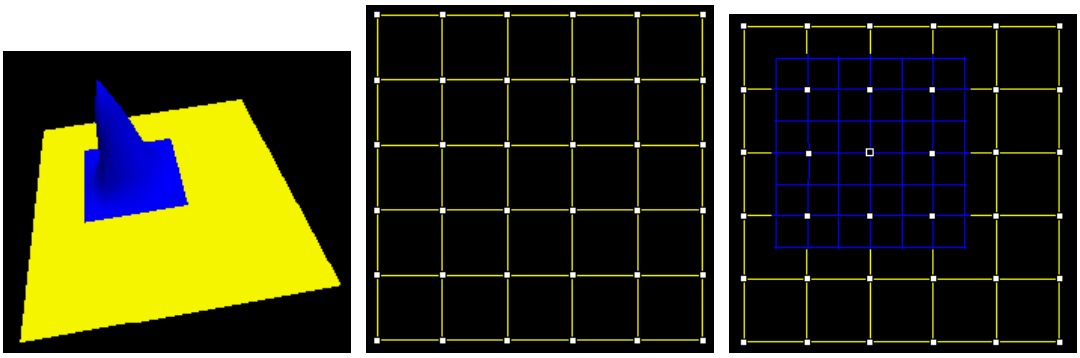
NURBS surface. However, this is an indirect approach and lacks geometric insights. In this chapter, we present direct conversion algorithms, which will take into consideration the structure of the hierarchical NURBS or T-spline surface. The algorithms work in the same fashion as the local knot insertion algorithm of T-splines.

Section 6.2 extends the concept of hierarchical B-splines to the rational case. The algorithms for conversion back and forth of a surface between T-spline and hierarchical B-spline representations are given in Section 6.3 and Section 6.4. Section 6.5 summarizes the chapter and provides some further discussions.

6.2 Hierarchical NURBS

A hierarchical B-spline surface consists of a series of levels, each of which has a collection of B-spline patches. The level 0 has the lowest resolution describing the basic shape of the surface, and its children are called overlays describing some details of the surface. The overlays correspond to refined areas of the parent surface. An overlay at level k is created by designating a patch on its parent at level $k-1$, executing a refinement step and manipulating the control points of the refined patch. Therefore an overlay at level k has twice the resolution of its parent at level $k-1$. At the same level, the overlays do not cross each other. Otherwise, they are made into a larger overlay. Figure 6.1(a) shows a hierarchical B-spline surface consisting of a bicubic B-spline surface and an overlay. The B-spline surface is defined by a 6×6 control grid whose pre-image in the parameter space is shown in Figure 6.1(b). The overlay has a 7×7 control grid whose pre-image is shown in blue in Figure 6.1(c).

Note that to keep C^2 continuity between the overlay and its parent the manipulation of the control points should be restricted to the central control points and the peripheral control points should be kept static. For a bicubic B-spline surface, an overlay has a mesh of at least 7×7 and the control points on the three outer rings must remain unchanged. In



(a) (b) (c)

Figure 6.1: A hierarchical B-spline surface consisting of a bicubic B-spline surface and an overlay.

the hierarchical B-spline scheme, a convenient way to represent the overlay is to use the reference-plus-offset form. That is, each control point $P^{(k)}$ of an overlay at level k is represented in offset form: $P^{(k)} = R^{(k)} + O^{(k)}$ where $R^{(k)}$ is derived from the parent surface at level $k-1$ by a (mid-point) refinement and $O^{(k)}$ is the offset vector. Any change to the control points at level k is represented in the offset vectors. For those control points that remain unchanged, the offset $O^{(k)} = 0$. Offsets are responsible for the difference between the shape of the surface at two levels of representation. In this way, any changes to the surface at the lower level automatically propagate to the higher level and the editing movement of $O^{(k)}$ causes local change of the surface to a restricted region of the surface. Moreover, the reference-plus-offset form improves the economy of the representation because we only need to store the non-zero offsets. For example, an overlay of dimension 7×7 has only one non-zero offset.

Now we extend the concept of hierarchical B-splines into hierarchical NURBS. The generalization is done in two aspects. First, the refinement in the hierarchical B-spline formulation is not necessarily mid-point refinement. Instead, the refinement can be achieved by allowing insertion of knots at general positions. As a result, a knot in the knot sequences of an overlay could be an arbitrary convex combination of two adjacent knots in the parent surface's knot sequences. See Figure 6.2 for an illustration: the left is the pre-image of a

6×6 control grid that defines a bicubic B-spline surface; the middle is the pre-image of a 7×7 control grid that defines an overlay; and the relation of the base and the overlay is shown on the right. Apparently, this increases the ability of hierarchical B-splines for representing the details. Second, the B-spline surfaces in the hierarchy should be allowed to be rational. It is because that the T-spline surfaces are usually rational surfaces and thus cannot be converted into hierarchical B-spline surfaces in the polynomial form. In this case, we apply the idea of original hierarchical B-spline refinement to the homogeneous representation of the rational B-splines. We still use the reference-plus-offset form for the overlays. Thus the offset vectors are 4D vectors. To maintain the continuity between the overlay and its parent, we restrict all the offset vectors corresponding to the three outermost rings of control points of the overlay to be zero.

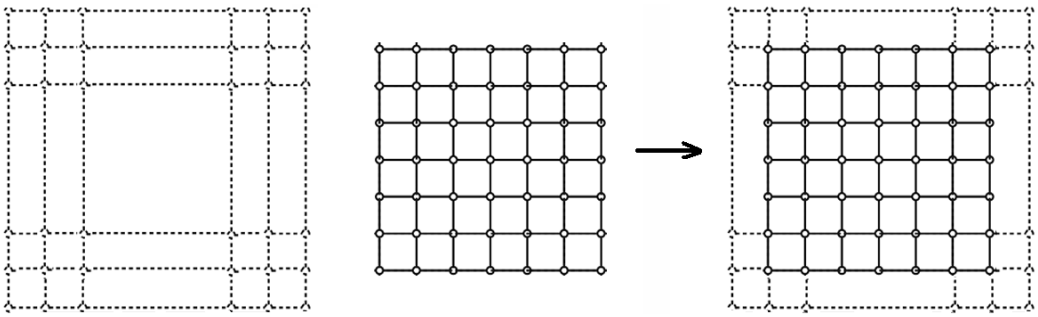


Figure 6.2: The relation between the base mesh and an overlay mesh.

6.3 Algorithm for converting a hierarchical NURBS to a T-spline

Given a hierarchical NURBS surface, one can flatten the hierarchical structure by creating the underlying NURBS surface, which usually contains a number of superfluous control points that serve no purpose other than to satisfy topological requirements of the tensor-product NURBS surface formulation. The conversion of a hierarchical NURBS surface into a T-spline surface is to combine all overlays into a single layer surface with only a few

superfluous points and also to keep the resulting surface identical to the original one. Our approach to such a transformation is to use a recursive way to generate the T-spline surface: we begin with the level 0 NURBS surface and add all the first level overlay(s) to it to form a T-spline surface; then we add the second level overlay(s) into the formed T-spline to create a more complicated T-spline surface; and this process continues until there is no overlay left. Obviously, the key ingredient in this approach is the algorithm that adds an overlay into a T-spline surface. In the following we describe the algorithm.

Now suppose we have a T-spline surface $\mathbf{S}(u,v)$ and an overlay $\mathbf{V}(u,v)$. Assume $\mathbf{S}(u,v)$ is defined by Equation (2.9) and it is the result of combining all the overlays that have lower level than $\mathbf{V}(u,v)$ and some of the overlays that have the same level as $\mathbf{V}(u,v)$ into the level 0 B-spline surface. The overlay $\mathbf{V}(u,v)$ is a bicubic rational B-spline surface, whose equation in homogeneous form is

$$\mathbf{V}(u,v) = \sum_{i=l-1}^{r+1} \sum_{j=b-1}^{t+1} \mathbf{V}_{ij} N_i(u) N_j(v) \tag{6.1}$$

where $\mathbf{V}_{ij}$ are control points in 4D space, $N_i(u)$ and $N_j(v)$ are cubic B-spline basis functions defined over knot sequences $\{u_{l-2}, u_{l-1}, u_l, \cdots, u_r, u_{r+1}, u_{r+2}\}$ and $\{v_{b-2}, v_{b-1}, v_b, \cdots, v_t, v_{t+1}, v_{t+2}\}$. The parameter range of the overlay is $[u_l, u_r] \times [v_b, v_t]$. From the formulation of the hierarchical B-spline, u_l, u_r, v_b and v_t are also the knots of $\mathbf{V}(u,v)$'s parent surface and thus correspond to certain edges in the T-mesh of $\mathbf{S}(u,v)$.

Based on the reference-plus-offset form, we only store the offset for the overlay. In fact, each point $\mathbf{V}_{ij}$ of the overlay is the sum of reference point $\mathbf{R}_{ij}$ and offset vector $\mathbf{O}_{ij}$. All the reference points $\mathbf{R}_{ij}$ are specified from the parent surface. They form a surface $\mathbf{R}(u,v) = \sum_{i=l-1}^{r+1} \sum_{j=b-1}^{t+1} \mathbf{R}_{ij} N_i(u) N_j(v)$ which is an exact re-representation of the parent surface or the T-spline surface $\mathbf{S}(u,v)$ within domain $[u_l, u_r] \times [v_b, v_t]$. If we use all the offset vectors to define a 4D surface $\mathbf{O}(u,v) = \sum_{i=l-1}^{r+1} \sum_{j=b-1}^{t+1} \mathbf{O}_{ij} N_i(u) N_j(v)$, then $\mathbf{V}(u,v) = \mathbf{R}(u,v) + \mathbf{O}(u,v)$. Also note that when $i < l+2$, $i > r-2$, $j < b+2$ or $j > t-2$, $\mathbf{O}_{ij} = 0$.

Therefore the overlay $\mathbf{V}(u,v)$ can be written

$$\mathbf{V}(u,v) = \sum_{i=1}^{n} \mathbf{P}_i B_i(u,v) + \sum_{i=l+2}^{r-2} \sum_{j=b+2}^{t-2} \mathbf{O}_{ij} N_i(u) N_j(v) \tag{6.2}$$

If we treat $\mathbf{O}_{ij}$ as a control point associated with knots (u_i, v_j), Equation (6.2) gives a PB-spline description of the overlay [129]. To make the PB-spline be a T-spline, we may need to perform the necessary knot insertions into the basis functions or add new control points into the mesh to make the mesh be a valid T-mesh. The insertion of a control point $\mathbf{O}_{ij}$ means not only to insert the associated knot (u_i, v_j) into the T-mesh, but to add the corresponding term of $\mathbf{O}_{ij} N_i(u) N_j(v)$ into the T-spline as well. This leads to the following algorithm (**Algorithm 6.1**) for composing an overlay into a T-spline.

Algorithm 6.1 : Composing an overlay into a T-spline surface

1. Insert points $\mathbf{O}_{ij}$ $(i = l+2, \cdots, r-2;\ j = b+2, \cdots, t-2)$ into the T-mesh at the locations inferred by their associated knots;
2. **repeat**
3. **if** (any basis function has a knot that is not indicated in the current T-mesh) **then**
4. Add an appropriate point into the T-mesh;
5. **end if**
6. **if** (any basis function is missing a knot inferred from the current T-mesh) **then**
7. Perform the necessary knot insertions into that basis function;
8. **end if**
9. **if** (any basis function has no control point associated to it) **then**
10. Add an appropriate control point into the T-mesh;
11. **end if**
12. **until** (there is no new operation)
13. **if** (any point has only one edge incident to it) **then**
14. Extend the edge and find the first intersection point of the edge and the T-mesh in the pre-image space;
15. **end if**
16. Insert the intersection point and then go to step 2;

In Line 13-16, we perform knot insertion though the T-mesh is already valid. This is to avoid the hanging edges and thus to preserve the geometrically pleasing shape of the control mesh.

6.3.1 Examples

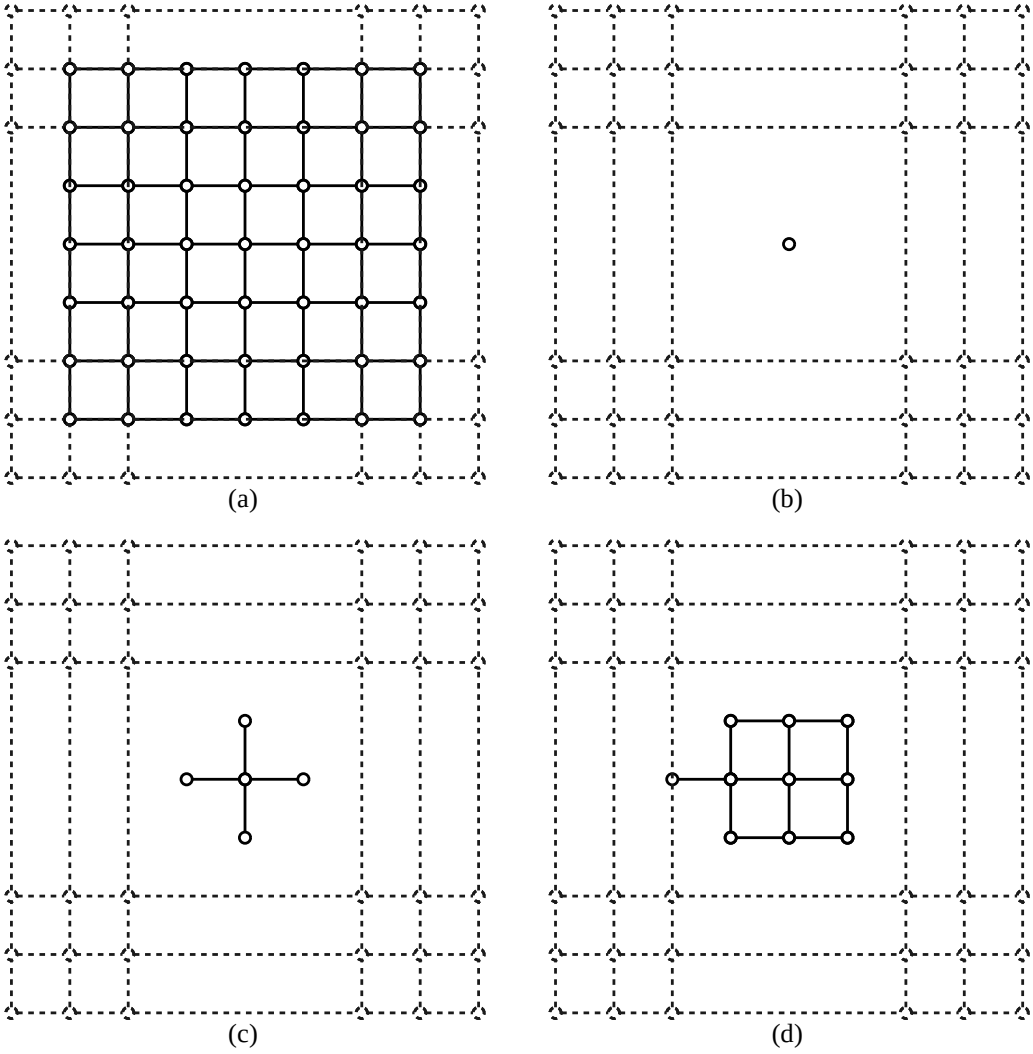


Figure 6.3: Illustration of converting a hierarchical B-spline to a T-spline.

We illustrate the algorithm with an example. Figure 6.3(a) shows the pre-image of a 6×6 control mesh (in dash lines) of a B-spline surface, imposed by the pre-image of a refined 7×7 mesh (in solid lines) that defines an overlay. We first insert the nonzero offset point of the 7×7 mesh (that is the central point) into the 6×6 B-spline mesh, generating a mesh shown in Figure 6.3(b). However, the basis function corresponding to this nonzero offset point has knots that are not indicated in the current mesh. We have to add four control points around the nonzero offset point (see Figure 6.3(c)). Now every basis function

properly associates with a control point in the mesh. But we have four hanging edges in the mesh. To avoid hanging edges, we finally perform Line 13-16 of Algorithm 6.1, inserting another 4 points, to create the T-mesh as shown in Figure 6.3(d). It can be seen that the result T-mesh is quite compact.

Now we present another example of a hierarchical B-spline with its geometry (see Figure 6.4). The hierarchical B-spline has three levels. The level 0 is a part of the $z = 0$ plane, defined by a B-spline with 8×8 control grid (see Figure 6.5(a)). The level 1 consists of 3 B-spline patches (see Figure 6.5(b)): the red and green have 7×7 control grid and only the central point is nonzero; the blue has 8×8 control grid and the central 4 points are non-zero. The level 2 is shown in yellow (see Figure 6.5(c)): it is a B-spline with 7×8 grid and the 2 central points are nonzero.

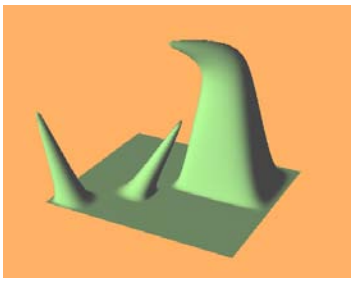
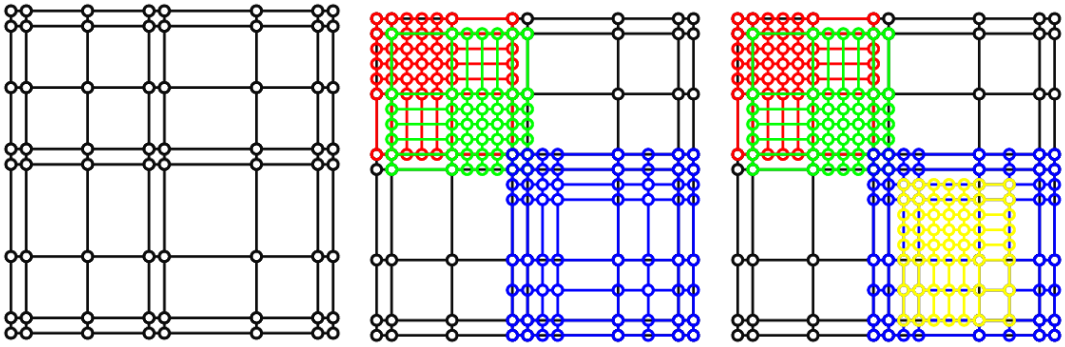


Figure 6.4: A hierarchical B-spline surface.

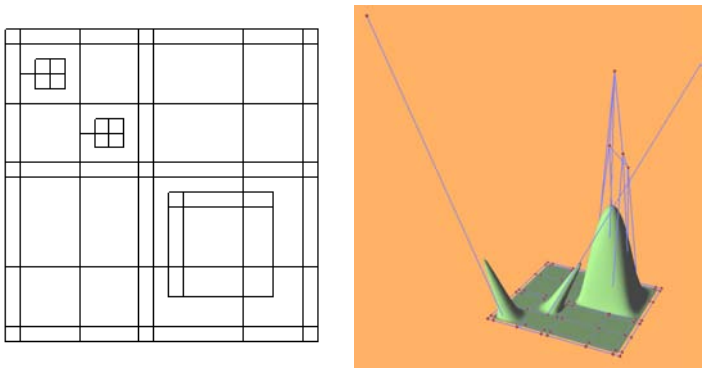


(a) Level 0 (b) Levels 0+1 (c) Levels 0+1+2

Figure 6.5: The pre-image of the control meshes of a hierarchical B-spline surface.

After we apply our algorithm to combine the level 1 and level 0. We get a T-mesh. The pre-image of the T-mesh is shown on Figure 6.6(a) and the surface is shown on Fig-

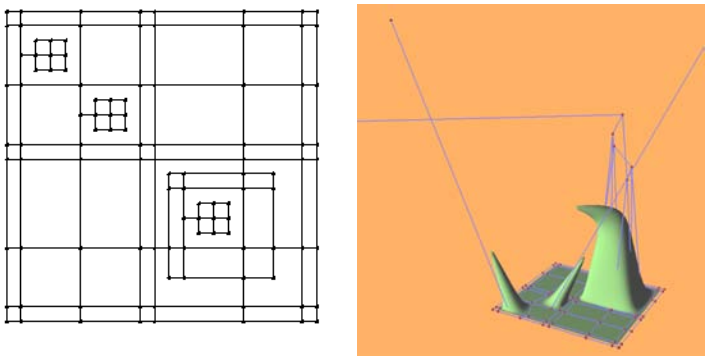
ure 6.6(b). After we also combine the level 2 overlay, the pre-image becomes a new one shown on Figure 6.7(a). The T-mesh and the final surface are shown on Figure 6.7(b).



(a) The pre-image of the T-mesh (b) The T-mesh and the surface

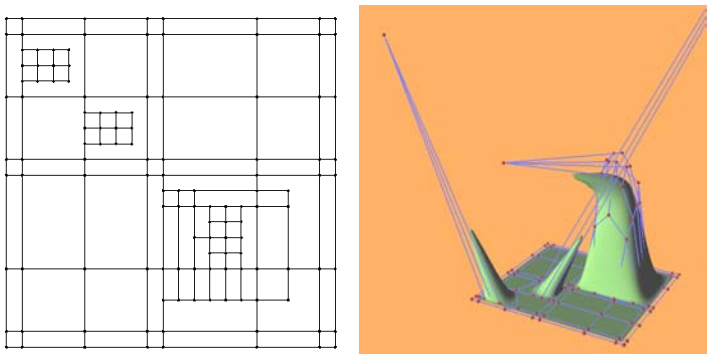
Figure 6.6: Combining level 0 and level 1.

Note that in Figure 6.7, some edges or vertices of the T-mesh coincide. This may cause inconvenience for interactive editing. We can locally insert some vertices to eliminate such coincidence. For example, after inserting some vertices into the T-mesh shown in Figure 6.7, we can get a new T-mesh which defines the same T-spline surface (see Figure 6.8). This new T-mesh would be clearer than the previous one for interactive editing purpose.



(a) The pre-image of the T-mesh (b) The T-mesh and the surface

Figure 6.7: Combining level 0, level 1 and level 2.



(a) The pre-image of the T-mesh (b) The T-mesh and the surface

Figure 6.8: Refined T-mesh for the hierarchical B-spline surface.

6.4 Algorithm for converting a T-spline to a hierarchical NURBS

Compared to the transformation from a hierarchical NURBS surface to a T-spline surface, the reverse transformation that converts a T-spline to a hierarchical NURBS is more challenging since we intend to create a hierarchy from a single layer T-mesh. The basic task here is to generate a series of B-spline surfaces. These B-splines are organized in a hierarchical manner. In the pre-image of the control grid in the parameter space, except for the base B-spline, every B-spline is the refinement of another B-spline in the group. The process for this task involves two aspects: the topological structure and the geometry. The former identifies the control grid structure of the base B-spline surface and the overlays at each level. The latter computes the coordinates of the control points for all the surfaces. Our algorithm can roughly be described as follows in Algorithm 6.2:

The preprocess step is to use the T-spline local knot insertion algorithm to make the three outmost rings of the T-mesh not contain any T-junction points. Refer to Figure 6.9 (left) for a given T-mesh. Figure 6.9 (right) shows the mesh after the preprocess step. The next step just does some initialization. Now we present the details of the remaining steps below.

Algorithm 6.2 : Converting a T-spline to a hierarchical NURBS surface

1. Preprocess the given T-mesh;
2. Set k = 0, $currentMesh$ = the preprocessed T-mesh;
3. Extract a B-spline surface at level k from the $currentMesh$;
4. Determine the offset surfaces for level $lev = k + 1$;
5. **for** (the mesh of each offset surface at level lev) **do**
6. Process the mesh;
7. **if** (the processed mesh is not a B-spline mesh) **then**
8. Set $currentMesh$ = the processed mesh;
9. Set $k = lev$;
10. Recursively call **Algorithm 6.2** starting from step 3;
11. **end if**
12. **end for**

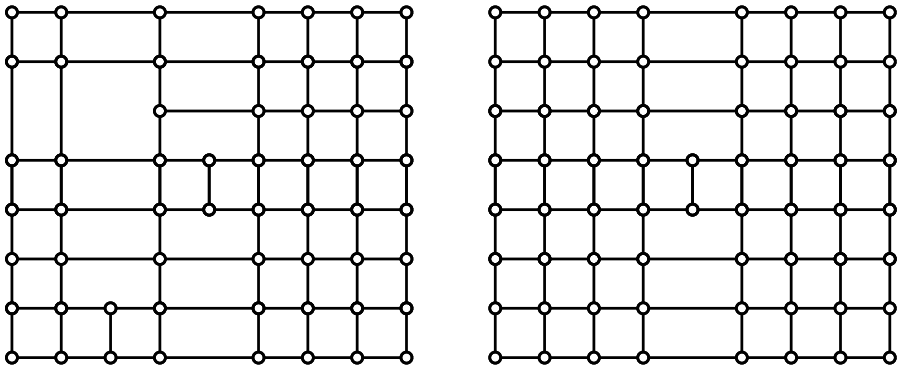


Figure 6.9: Preprocess a given T-mesh.

6.4.1 Extracting a B-spline

While a T-mesh permits T-junctions so that lines of the T-mesh need not traverse the entire control grid, a tensor-product B-spline mesh must be topologically a rectangular grid. Therefore to extract a B-spline mesh from a T-mesh, we eliminate all partial rows and all partial columns in the T-mesh. What remains after the removal of partial rows and columns is an $m \times n$ mesh, which can be used as the topological specification of our B-spline control mesh.

Once the topology of the B-spline mesh is specified, we have to compute the geometry of the B-spline mesh. The formulae for the B-spline control points are derived based on polar form of B-splines [117].

For each control point $\mathbf{P}$ in the extracted B-spline mesh (see Figure 6.10(a)), we com-

pute its polar label $PL_{\mathbf{P}}$. Denote $PL_{\mathbf{P}} = (u_{-1}, u_0, u_1) \times (v_{-1}, v_0, v_1)$. Point $\mathbf{P}$ corresponds to a control point, say $\mathbf{Q}$, in the T-mesh. We also compute $\mathbf{Q}$'s polar label $PL_{\mathbf{Q}}$ in the T-mesh. If $PL_{\mathbf{P}} = PL_{\mathbf{Q}}$, then we just copy $\mathbf{Q}$'s coordinates to $\mathbf{P}$'s. Otherwise, we create a temporary T-mesh by duplicating the existing T-mesh and use T-spline's local knot insertion algorithm to insert points in the temporary T-mesh, forming a 5×5 grid centered around $\mathbf{Q}$. This 5×5 grid is part of the finest control mesh of the underlying B-spline surface. We denote the points by $\mathbf{Q}_{ij}$, $(i, j = -2, \cdots, 2)$ with $\mathbf{Q}_{00}$ corresponding to $\mathbf{Q}$. Refer to Figure 6.10(b) for labels. Then $\mathbf{P}$ can be computed by

$$\mathbf{P} = \sum_{i=-1}^{1} \sum_{j=-1}^{1} c_i d_j \mathbf{Q}_{ij}$$

where

$$\begin{aligned} c_{-1} &= \frac{(a_1 - u_{-1})(a_1 - u_1)}{(a_1 - a_{-2})(a_1 - a_{-1})} \\ c_1 &= \frac{(a_{-1} - u_{-1})(a_{-1} - u_1)}{(a_2 - a_{-1})(a_1 - a_{-1})} \\ c_0 &= 1 - c_{-1} - c_1 \end{aligned}$$

and d_i are similar to c_i where a_j should be replaced by b_j and u_j should be replaced by v_j.

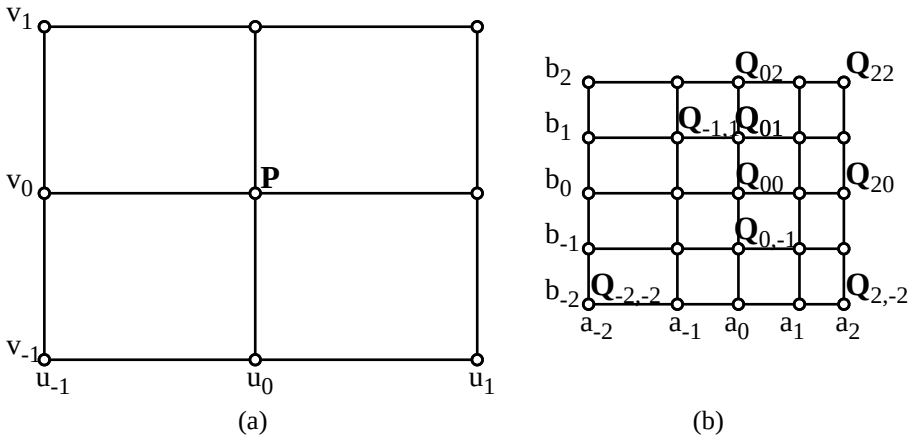


Figure 6.10: Computing B-spline control point $\mathbf{P}$ from $\mathbf{Q}_{ij}$.

6.4.2 Determining higher level offsets

When we have extracted a B-spline mesh from a T-mesh at level k, both the B-spline mesh and the T-mesh define two surfaces which are not necessarily the same. In this case, we need higher level (offset) data to compensate the difference.

We first locate the regions for the higher level offset surfaces. For each point $\mathbf{P}$ in the pre-image of the T-mesh at level k in (u, v) parameter space, we define four numbers u_min, v_min, u_max and v_max such that u_min is the u coordinate of a new point which is obtained by moving point $\mathbf{P}$ left by two bays, u_max is the u coordinate of a point which is obtained by moving $\mathbf{P}$ right by two bays, and v_min, v_max are similarly defined. For each partial row or column in the T-mesh, we define its box-2 $O(u_l, v_b, u_r, v_t)$ to be a rectangle specified by the lower-left corner (u_l, v_b) and the upper-right corner (u_r, v_t) where u_l, v_d are the minima of u_min and v_min of all vertices lying on the partial row or column, and u_r, v_t are the maxima of u_max and v_max of all vertices lying on the partial row or column. Now we compute box-2 for all partial rows and columns. If any two box-2's overlap, we merge them to form a large box bounding them. We also call the large box a box-2. This large box is used to replace the original two small boxes. After this, each box-2 stands for a region of an overlay.

Once the region of an offset surface in the next layer is identified, we have to determine topology and geometry of the offset surface. We extend each box-2 by adding one more ring. The pre-image corresponds one-to-one with the T-mesh. We take the portion of the T-mesh at level k, which corresponds with the extended box-2. It forms a temporary T-mesh at level $k + 1$ for the surface within the identified region. By the construction of box-2, it can be verified that the level $k + 1$ T-mesh has such characteristic that the three outermost rings of the mesh do not contain T-junctions.

The 4D surface defined by the T-mesh at level k within the region is described by

$$\mathbf{S}^{(k)}(u,v) = \sum_{i=1}^{n} \mathbf{P}_i^{(k)} B_i(u,v) \tag{6.3}$$

where $\mathbf{P}_i^{(k)}$ are the control points in the T-mesh at level k or in the temporary T-mesh at level $k+1$. In order to get the offset surface, we restrict the extracted B-spline surface (from the T-mesh at level k) within the region and write its equation to be

$$\mathbf{R}^{(k)}(u,v) = \sum_{i=l-1}^{r+1} \sum_{j=b-1}^{t+1} \mathbf{R}_{ij}^{(k)} N_i(u) N_j(v) \tag{6.4}$$

Then the offset surface is $\mathbf{S}^{(k)}(u,v) - \mathbf{R}^{(k)}(u,v)$. This gives the Algorithm 6.3 for determining the topology and geometry of the offset surface.

Algorithm 6.3 : Determining the topology and geometry of the offset surface

1. Insert points $-\mathbf{R}_{ij}^{(k)}$ $(i = l-1, \cdots, r+1;\ j = b-1, \cdots, t+1)$ into the temporary T-mesh at the locations inferred by $\mathbf{R}_{ij}^{(k)}$'s associated knots;
2. **repeat**
3. **if** (any basis function is missing a knot inferred from the current T-mesh) **then**
4. Perform the necessary knot insertions into that basis function;
5. **end if**
6. **if** (any basis function has a knot that is not indicated in the current T-mesh) **then**
7. Add an appropriate point into the T-mesh;
8. **end if**
9. **if** (any basis function has no control point associated to it) **then**
10. Add an appropriate control point into the T-mesh;
11. **end if**
12. **until** there is no new operation
13. The output mesh is the required T-mesh at level $k+1$ which defines the offset surface;

6.4.3 Processing a layer

Now we have obtained the T-mesh at level $k+1$ for the offset surface. If it happens to be a B-spline mesh, it is a leaf node in the hierarchical B-spline representation at level $k+1$ and we do not need to do further process. Otherwise, some further process is required. We can

either extend all partial rows and columns all the way to create a B-spline or further extract higher level of layers.

We take a strategy to balance the depth of the hierarchy and size of each offset. For each partial row or column in the T-mesh at level $k+1$, we calculate a number, called *deficiency*, which measures how many knots in the underlying B-spline this partial row or column needs to go through to reach the border. If the deficiency is large, the row or column needs to take a long way to reach the border, which might mean that it depicts details and thus would be better to be put in the overlay of the current layer. For this reason, we compute the average of the deficiencies of all partial rows and columns. If any partial row or column has a deficiency less than the average deficiency plus δ (usually we set $\delta = 3$), we extend it all the way to make it a complete row or column. Here, δ is a user's option. Small δ would lead to more levels in the hierarchy, while large δ would result in increased size of the offset surfaces. If there is no T-junction after the extension, then a B-spline is formed and can be treated as a leaf node attached to the parent surface. If there still exist T-junctions, then pass the mesh for further decomposition in the next level.

6.4.4 Illustrations

We illustrate the algorithm topologically with an example. Figure 6.11(a) shows a T-mesh which has four T-junctions. We wish to convert the T-spline to a hierarchical B-spline. We first eliminate the partial row and column, yielding a B-spline mesh in Figure 6.11(b). Of course, the geometry of the B-spline mesh should be computed carefully. Also from the T-mesh, we construct two box-2's for the partial row and column (see Figure 6.11(c)). Combining the overlapped box-2's generates one box as displayed in Figure 6.11(d). This box identifies the range of the next level overlay. Expanding the range bounded by the box one more ring gives a temporary T-mesh. Inserting the *negative* of the extracted B-spline into this temporary T-mesh leads to a T-mesh for the offset, which topologically is the same as the temporary mesh shown in Figure 6.11(e). Then we extend both partial row and partial

column to the border, creating a 11×10 mesh in Figure 6.11(f), which defines the offset of the overlay. The final hierarchical B-spline consists of the level 0 B-spline surface shown in Figure 6.11(b) and the level 1 offset shown in Figure 6.11(f). It can be seen that by this conversion algorithm, we still achieve compactness in the hierarchical B-spline to some extent. Although the size of the level 1 surface is 11×10, the area of non-zero vectors that needs to be stored is only 5×4.

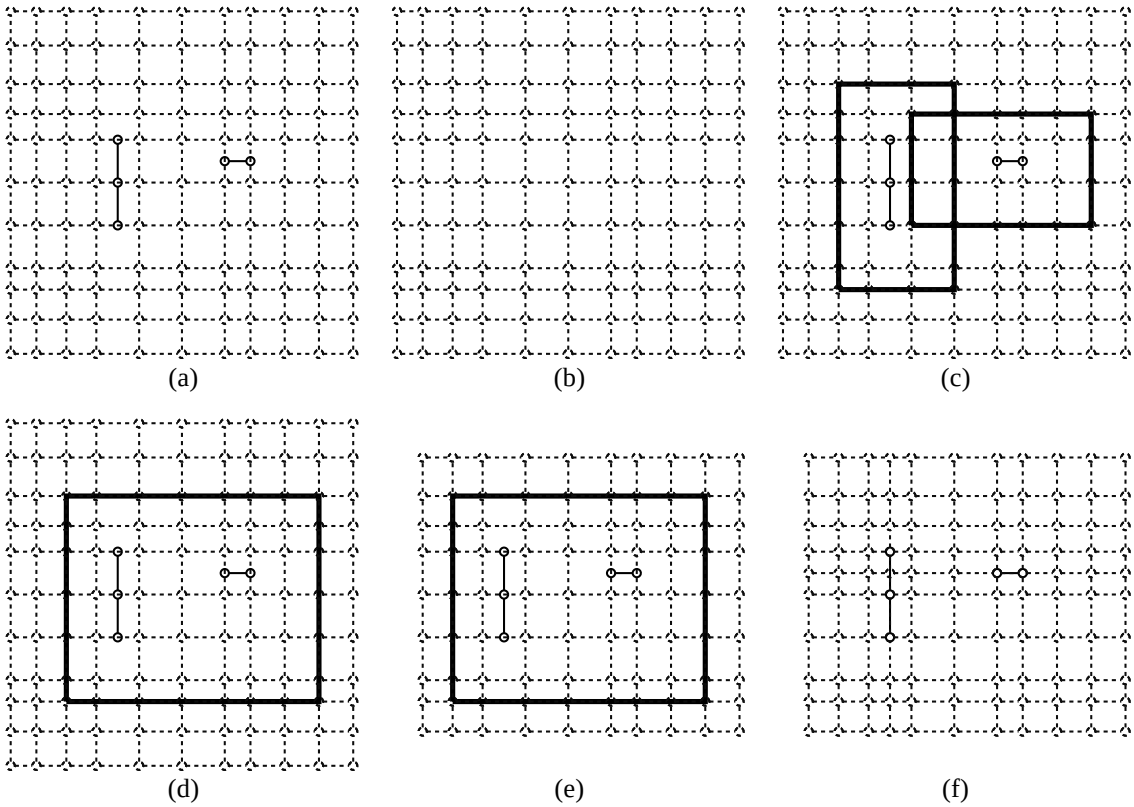


Figure 6.11: A topological illustration.

Another example is given in Figure 6.12. A T-spline surface and its pre-image are shown in Figure 6.12(a) and Figure 6.12(d), respectively. By using the algorithm, the surface can be equivalently converted into a hierarchical NURBS surface. Figure 6.12(b) shows the level 0 base surface of the result hierarchical NURBS, for which the control grid in the parameter domain is displayed in Figure 6.12(e). Figure 6.12(c) is the level 0+1 hierarchical NURBS, where the region of overlay is shown in red color. The red grid in Figure 6.12(f) is the control grid for the overlay.

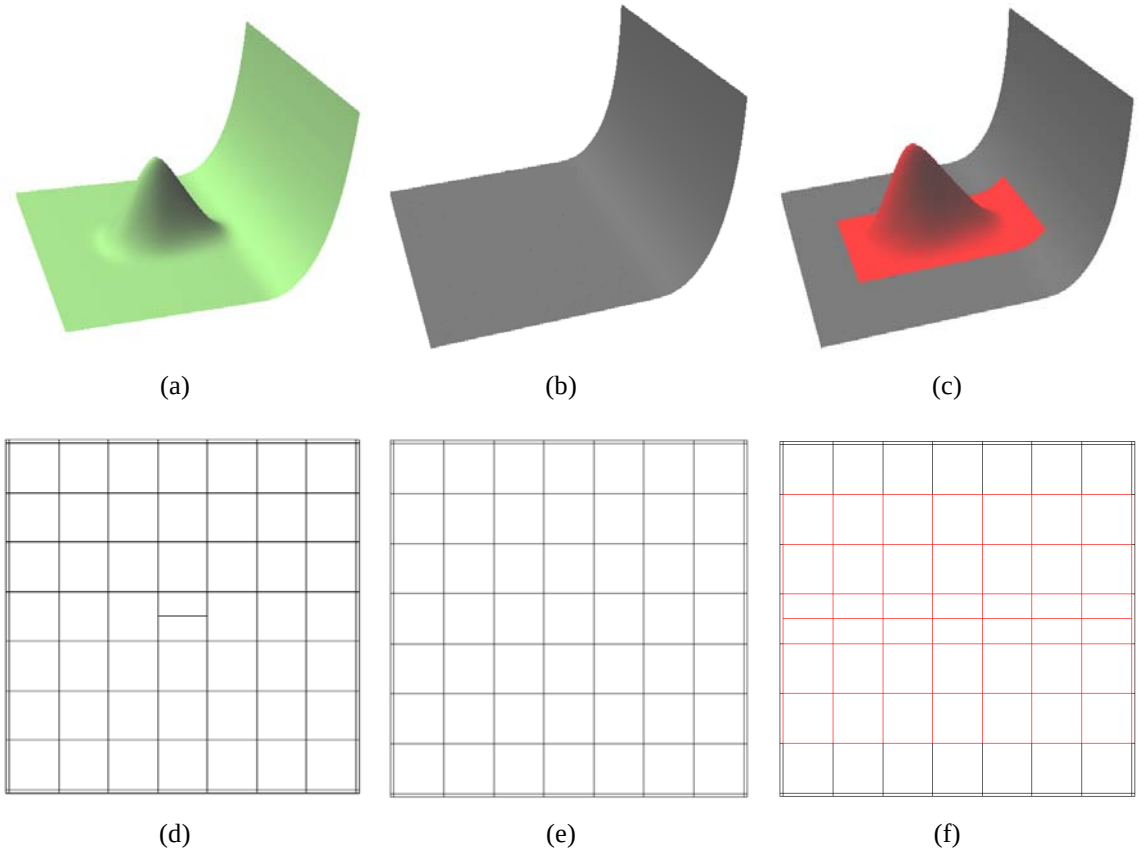


Figure 6.12: Converting a T-spline to a hierarchical NURBS.

6.5 Summary

We have described two algorithms for performing conversion from a hierarchical NURBS surface to a T-spline surface and from a T-spline surface to a hierarchical NURBS surface. The algorithms are intuitive and are directly based on local knot insertion. They can generate quite compact representations. In applications such as free-form shape modeling and design, both hierarchical B-spline surfaces and T-spline surfaces are useful tools. When one of them is available, our algorithms make the other easier to use.

In order to make the conversion algorithms universal, we extend the concept of the hierarchical B-splines to the rational case and we also allow the knot vector for the overlays to be nonuniform.

There are still some remaining problems for future work. For example, so far we just

discussed the surface with simple knots. The conversion for more general cases needs to be extended. In Subsection 6.4.3, we gave a heuristic criterion to decide whether a partial row or column should be extended. Finding an optimal criterion would be an interesting question. Moreover, the current work in this chapter only considers accurate conversion between the two representations. Approximation conversion may be more interesting from practice perspective.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

The research in this thesis investigates the use of T-splines for free-form surface representation and approximation, aiming to gain insights of T-spline theory and to present new efficient algorithms for T-splines. Several fundamental problems have been studied. Novel techniques have been successfully developed, which accomplishes the goal of the research.

First, we have presented an algorithm for T-spline local knot removal that is a very fundamental operation in T-spline theory. The algorithm is able to detect whether a specified control point can be removed from a T-spline surface and to compute the new T-spline representation if the point is removable. It works locally, in the fashion of T-spline local knot insertion. The extension of the algorithm to remove more than one control point is possible and several approaches have been suggested. However, our research has also shown that the T-junction of the T-spline makes the T-spline knot removal problem much more complicated than NURBS knot removal, especially in removing more than one control point from a T-spline surface.

Second, we have studied the problem of surface fitting using T-splines deeply. We have introduced a new framework for adaptive surface fitting which achieves the conversion

from a triangular mesh that is topologically homeomorphic to a plane region to a spline surface. The new framework is quite comprehensive and includes many new ideas and algorithms. Extensive experiments have been conducted to evaluate the performance of the new framework and algorithms. The examples have shown that the proposed adaptive T-spline fitting does produce T-spline control meshes that exhibit a noticeable correspondence between features and control point density. The algorithm is particularly suitable for fitting those triangular meshes whose vertices are geometrically distributed unevenly in the space.

Third, we have presented several techniques for handling tubular models that have the same topology as a cylinder. Periodic T-splines and their simple representations have been proposed. A geometrically intrinsic method has been presented to parameterize tubular triangular meshes. The algorithm overcomes some problems existing in conventional cutting-based algorithms. An adaptive surface fitting algorithm using periodic T-splines has been developed. These techniques can deal with tubular models very well.

Fourth, we have presented algorithms for conversion between a surface represented in T-splines and hierarchical NURBS. These algorithms were well designed by considering the characteristics of both representations. With these algorithms, the user can flatten a hierarchical NURBS surface to create a T-spline which offers a more intuitive interface for interactive sculpturing or extract hierarchies from a T-spline surface for multiresolution analysis.

In summary, T-splines are relatively new and there are many problems that remain to be solved. Our research has provided some new algorithms to the T-spline family. They also demonstrate that T-splines are a flexible and powerful representation for free-form surfaces. In particular, T-splines are an ideal surface representation for applications that have unevenly distributed data or features and need some adaptive processing.

7.2 Future work

The techniques in this research also contain some limitations, which bring some topics for further studies:

- Some algorithms presented in the thesis (for example, the algorithms for the conversion between T-splines and hierarchical NURBS) are heuristic. Although they might not be the optimal solutions, the heuristic algorithms provide a good foundation for future work in designing optimal algorithms that are of interest in both theoretical and practical perspectives.

- We have shown the complexity of T-spline knot removal, especially in removing more than one control point. However, the algorithm only considers exact removal of control points and might not be efficient in handling complicated situations. An interesting problem is to identify various situations and to give a neat solution to knot removal. In addition, from the application's viewpoint, it would be more interesting to consider approximate knot removal. That is to remove the control point(s) such that the surface is changed within a given tolerance.

- The surface fitting problem this thesis focused on is mainly for the models that have relatively simple topology. In general, fitting arbitrarily topological triangular meshes is more challenging, but it is more common in surface reconstruction. Our ultimate goal is to develop techniques that can efficiently convert an arbitrarily topological triangular mesh into spline surfaces. Obviously, the results produced in the thesis provide a good base for the general situation. For example, to approximate a model of arbitrary topology, we may first divide it into a number of disc homeomorphic parts using segmentation [5, 133] techniques. Then, each of these parts can be individually approximated using our proposed method under some continuity constraints on the shared boundaries. Alternatively, we can join these parts together afterwards using surface stitching techniques.

References

[1] "AIM@SHAPE shape repository," http://shapes.aimatshape.net/.

[2] "ISO 10303-1:1994, Industrial Automation and Integration – Product Data Representation and Exchange."

[3] "CGAL, Computational Geometry Algorithms Library," http://www.cgal.org.

[4] M. Alexa, J. Behr, D. Cohen-Or, S. Fleishman, D. Levin, and C. T. Silva, "Point set surfaces," in *Proceedings of the conference on Visualization '01*. San Diego, California: IEEE Computer Society, 2001, pp. 21–28. [Online]. Available: http://portal.acm.org/citation.cfm?id=601673

[5] M. Attene, S. Katz, M. Mortara, G. Patane, M. Spagnuolo, and A. Tal, "Mesh segmentation - a comparative study," in *Proceedings of the IEEE International Conference on Shape Modeling and Applications 2006*. IEEE Computer Society, 2006, p. 7. [Online]. Available: http://portal.acm.org/citation.cfm?id=1136960

[6] B. Barsky and D. Greenberg, "Determining a set of B-Spline control vertices to generate an interpolating surface," vol. 14, no. 3, pp. 203–226, November 1980.

[7] B. Baumgart, "Winged-Edge polyhedron representation for computer vision," in *National Computer Conference*, May 1975. [Online]. Available: http://www.baumgart.org/winged-edge/winged-edge.html

[8] F. Bernardini, H. Rushmeier, I. Martin, J. Mittleman, and G. Taubin, "Building a digital model of michelangelo's florentine pieta," *Computer Graphics and Applications, IEEE*, vol. 22, no. 1, pp. 59–67, 2002.

[9] F. Bernardini and H. Rushmeier, "The 3D model acquisition pipeline," *Computer Graphics Forum*, vol. 21, no. 2, pp. 149–172, 2002.

[10] M. Bertram, X. Tricoche, and H. Hagen, "Adaptive smooth scattered data approximation for large-scale terrain visualization," in *Proceedings of the symposium on Data visualisation 2003*, C. D. H. G.-P. Bonneau, S. Hahmann, Ed. Eurographics Association, 2003, pp. 177–184. [Online]. Available: http://www.eg.org/EG/DL/WS/VisSym03/177-184-tricoche.pdf

[11] F. Blais, "Review of 20 years of range sensor development," *Journal of Electronic Imaging*, vol. 13, no. 1, pp. 231–240, January 2004.

[12] J. Bloomenthal and B. Wyvill, Eds., *Introduction to Implicit Surfaces*. Morgan Kaufmann Publishers Inc., 1997. [Online]. Available: http://portal.acm.org/citation.cfm?id=549676

[13] A. I. Bobenko, P. Schröder, J. M. Sullivan, and G. M. Ziegler, *Discrete Differential Geometry*, 1st ed. Birkhäuser Basel, Mar. 2008.

[14] W. Boehm, "Inserting new knots into B-spline curves," *Computer Aided Design*, vol. 12, pp. 199–201, 1980.

[15] M. Botsch, S. Steinberg, S. Bischoff, and L. Kobbelt, "OpenMesh – a generic and efficient polygon mesh data structure," in *OpenSG Symposium*, 2002.

[16] M. Botsch, M. Pauly, C. Rössl, S. Bischoff, and L. Kobbelt, "Geometric modeling based on triangle meshes," in *ACM SIGGRAPH*

2006 Courses. Boston, Massachusetts: ACM, 2006. [Online]. Available: http://portal.acm.org/citation.cfm?id=1185839

[17] J. C. Carr, R. K. Beatson, J. B. Cherrie, T. J. Mitchell, W. R. Fright, B. C. McCallum, and T. R. Evans, "Reconstruction and representation of 3D objects with radial basis functions," in *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. ACM, 2001, pp. 67–76.

[18] E. Catmull and J. Clark, "Recursively generated b-spline surfaces on arbitrary topological meshes," *Computer-Aided Design*, vol. 10, no. 6, pp. 350–355, Nov. 1978.

[19] F. Cheng and A. Goshtasby, "A parallel B-spline surface fitting algorithm," *ACM Trans. Graph.*, vol. 8, no. 1, pp. 41–50, 1989.

[20] F. Cheng, G. W. Wasilkowski, J. Wang, C. Zhang, and W. Wang, "Parallel B-Spline surface interpolation on a mesh-connected processor array." *J. Parallel Distrib. Comput.*, vol. 24, no. 2, pp. 224–229, 1995.

[21] K.-L. Chung and W.-M. Yan, "Parallel B-Spline surface fitting on mesh-connected computers." *J. Parallel Distrib. Comput.*, vol. 35, no. 2, pp. 205–210, 1996.

[22] U. Clarenz, N. Litke, and M. Rumpf, "Axioms and variational problems in surface parameterization." *Computer Aided Geometric Design*, vol. 21, no. 8, pp. 727–749, 2004.

[23] E. Cohen, T. Lyche, and R. Riesenfeld, "Discrete B-splines and subdivision techniques in computer-aided geometric design and computer graphics," *Computer Graphics and Image Processing*, vol. 14, pp. 87–111, 1980.

[24] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, Second Edition*. The MIT Press, September 2001.

[25] M. Cox, "The numerical evaluation of b-splines," National Physical Laboratory, DNAC 4, August 1971.

[26] B. Curless and M. Levoy, "A volumetric method for building complex models from range images," in *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*. ACM, 1996, pp. 303–312.

[27] Cyberware, Inc, http://www.cyberware.com.

[28] J. Dan and W. Lancheng, "An algorithm of NURBS surface fitting for reverse engineering," *The International Journal of Advanced Manufacturing Technology*, vol. 31, no. 1, pp. 92–97, Nov. 2006.

[29] C. de Boor, "On calculating with b-splines," *Journal of Approximation Theory*, vol. 6, pp. 50–62, 1972.

[30] P. Degener, J. Meseth, and R. Klein, "An adaptable surface parameterization method," *In Proceedings of the 12th International Meshing Roundtable*, pp. 201—213, 2003. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.59.1400

[31] M. Desbrun, M. Meyer, and P. Alliez, "Intrinsic parameterizations of surface meshes," *Computer Graphics Forum*, vol. 21, no. 3, pp. 209–218, 2002. [Online]. Available: http://dx.doi.org/10.1111/1467-8659.00580

[32] M. Desbrun, M. Meyer, P. Schröder, and A. H. Barr, "Implicit fairing of irregular meshes using diffusion and curvature flow," in *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co., 1999, pp. 317–324. [Online]. Available: http://portal.acm.org/citation.cfm?id=311576

[33] U. Dietz, "B-spline approximation with energy constraints," in *Advanced Course on Fairshape. Teubner*, 1996, pp. 229–239.

[34] S. Dong, P. Bremer, M. Garland, V. Pascucci, and J. C. Hart, "Spectral surface quadrangulation," *ACM Trans. Graph.*, vol. 25, no. 3, pp. 1057–1066, 2006.

[35] D. Doo and M. Sabin, "Behaviour of recursive division surfaces near extraordinary points," *Computer-Aided Design*, vol. 10, no. 6, pp. 356–360, Nov. 1978.

[36] J.-L. Duprat, "Integrating Hsplines into Softimage 3D," Master's thesis, The University of British Columbia, 1997.

[37] M. Eck, T. DeRose, T. Duchamp, H. Hoppe, M. Lounsbery, and W. Stuetzle, "Multiresolution analysis of arbitrary meshes," *Computer Graphics*, vol. 29, no. Annual Conference Series, pp. 173–182, 1995. [Online]. Available: citeseer.ist.psu.edu/article/eck95multiresolution.html

[38] M. Eck and H. Hoppe, "Automatic reconstruction of B-spline surfaces of arbitrary topological type," in *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, 1996, pp. 325–334.

[39] G. Farin, *Curves and Surfaces for CAGD: A Practical Guide*, 5th ed. Morgan Kaufmann, Oct. 2001.

[40] G. T. Finnigan, "Arbitrary degree t-splines," Master's thesis, Brigham Young University, 2008.

[41] S. Fleishman, I. Drori, and D. Cohen-Or, "Bilateral mesh denoising," *ACM Trans. Graph.*, vol. 22, no. 3, pp. 950–953, 2003.

[42] M. S. Floater, "Mean value coordinates," *Computer Aided Geometric Design*, vol. 20, pp. 19–27(9), March 2003.

[43] M. S. Floater and K. Hormann, "Parameterization of triangulations and unorganized points," in *Tutorials on Multiresolution in Geometric Modelling*, ser. Mathematics and Visualization, A. Iske, E. Quak, and M. S. Floater, Eds. Springer, 2002, pp. 287–316.

[44] ——, "Surface parameterization: a tutorial and survey," in *Advances in Multiresolution for Geometric Modelling*, ser. Mathematics and Visualization, N. A. Dodgson, M. S. Floater, and M. A. Sabin, Eds. Berlin, Heidelberg: Springer, 2005, pp. 157–186.

[45] M. S. Floater, "Parametrization and smooth approximation of surface triangulations." *Computer Aided Geometric Design*, vol. 14, no. 3, pp. 231–250, 1997.

[46] T. A. Foley, "Scattered data interpolation and approximation with error bounds," *Comput. Aided Geom. Des.*, vol. 3, no. 3, pp. 163–177, 1986. [Online]. Available: http://portal.acm.org/citation.cfm?id=18696

[47] D. Forsey and D. Wong, "Multiresolution surface reconstruction for hierarchical B-splines," in *Graphics Interface*, 1998, pp. 57–64.

[48] D. R. Forsey and R. H. Bartels, "Hierarchical B-spline refinement," in *Proceedings of the 15th annual conference on Computer graphics and interactive techniques*, 1988, pp. 205–212.

[49] ——, "Surface fitting with hierarchical splines," *ACM Transactions on Graphics*, vol. 14, no. 2, pp. 134–161, 1995.

[50] R. Goldman and T. Lyche, *Knot Insertion and Deletion Algorithms for B-Spline Curves and Surfaces*. SIAM, 1993.

[51] C. Gotsman, S. Gumhold, and L. Kobbelt, "Simplification and compression of 3d meshes," *IN PROCEEDINGS OF THE EUROPEAN SUMMER SCHOOL ON PRIN-*

CIPLES OF MULTIRESOLUTION IN GEOMETRIC MODELLING, pp. 319—361, 2002.

[52] B. F. Gregorski, B. Hamann, and K. I. Joy, "Reconstruction of B-Spline surfaces from scattered data points," in *Proceedings of Computer Graphics International 2000*, N. Magnenat-Thalmann and D. Thalmann, Eds., 2000, pp. 163–170.

[53] G. Greiner and K. Hormann, "Interpolating and approximating scattered 3D-data with hierarchical tensor product B-splines," in *Surface Fitting and Multiresolution Methods*, ser. Innovations in Applied Mathematics, A. L. Méhauté, C. Rabut, and L. L. Schumaker, Eds. Nashville: Vanderbilt University Press, 1997, pp. 163–172.

[54] G. Greiner and H.-P. Seidel, "Automatic modeling of smooth spline surfaces," in *Proc. WSCG '97*, 1997, pp. 665–675.

[55] G. Greiner, "Variational design and fairing of spline surfaces," *Computer Graphics Forum*, vol. 13, no. 3, pp. 143–154, 1994.

[56] G. Greiner, J. Loos, and W. Wesselink, "Data dependent thin plate energy and its use in interactive surface modeling," *Computer Graphics Forum*, vol. 15, no. 3, pp. 175–186, August 1996, ISSN 1067-7055.

[57] C. M. Grimm and J. F. Hughes, "Modeling surfaces of arbitrary topology using manifolds," in *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*. ACM, 1995, pp. 359–368.

[58] A. Gruen, F. Remondino, and L. Zhang, "Computer reconstruction and modeling of the great buddha of bamiyan, afghanistan," in *19th CIPA International Symposium 2003*, 2003, pp. 440–445.

[59] X. Gu, Y. He, and H. Qin, "Manifold splines," in *Proceedings of the 2005 ACM symposium on Solid and physical modeling*. Cambridge, Massachusetts: ACM, 2005, pp. 27–38.

[60] G. Guennebaud and M. Gross, "Algebraic point set surfaces," *ACM Trans. Graph.*, vol. 26, no. 3, p. 23, 2007. [Online]. Available: http://portal.acm.org/citation.cfm?id=1276377.1276406

[61] A. Gueziec, G. Taubin, F. Lazarus, and B. Hom, "Cutting and stitching: converting sets of polygons to manifold surfaces," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 7, no. 2, pp. 136–151, 2001.

[62] L. Guibas and J. Stolfi, "Primitives for the manipulation of general subdivisions and the computation of voronoi diagrams," *ACM Trans. Graph.*, vol. 4, no. 2, pp. 74–123, 1985. [Online]. Available: http://portal.acm.org/citation.cfm?id=282923

[63] G. Guidi, L. Micoli, M. Russo, B. Frischer, M. D. Simone, A. Spinetti, and L. Carosso, "3D digitization of a large model of imperial rome," in *Proceedings of the Fifth International Conference on 3-D Digital Imaging and Modeling*. IEEE Computer Society, 2005, pp. 565–572.

[64] J. Haber, F. Zeilfelder, O. Davydov, and H. P. Seidel, "Smooth approximation and rendering of large scattered data sets," in *Proceedings of the conference on Visualization '01*. San Diego, California: IEEE Computer Society, 2001, pp. 341–348.

[65] H. Hagen, S. Hahmann, and G.-P. Bonneau, "Variational surface design and surface interrogation," *Computer Graphics Forum*, vol. 12, no. 3, pp. 447–459, 1993. [Online]. Available: http://dx.doi.org/10.1111/1467-8659.1230447

[66] M. Halstead, M. Kass, and T. DeRose, "Efficient, fair interpolation using catmull-clark surfaces," in *Proceedings of the 20th annual conference on Computer*

graphics and interactive techniques. Anaheim, CA: ACM, 1993, pp. 35–44. [Online]. Available: http://portal.acm.org/citation.cfm?id=166121

[67] D. Handscomb, "Knot elimination: reversal of the oslo algorithm," *International Series of Numerical Mathematics*, vol. 81, pp. 103–111, 1987.

[68] HBSURF, http://www.cs.berkeley.edu/ rcdavis/classes/cs284/.

[69] O. Hjelle, "Approximation of scattered data with multilevel B-splines," SINTEF, Tech. Rep., 2001.

[70] H. Hoppe, T. DeRose, T. Duchamp, M. Halstead, H. Jin, J. McDonald, J. Schweitzer, and W. Stuetzle, "Piecewise smooth surface reconstruction," in *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*. ACM, 1994, pp. 295–302.

[71] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle, "Surface reconstruction from unorganized points," in *Proceedings of the 19th annual conference on Computer graphics and interactive techniques*. ACM, 1992, pp. 71–78.

[72] K. Hormann, "Fitting free form surfaces," in *Principles of 3D Image Analysis and Synthesis*, ser. The Kluwer International Series in Engineering and Computer Science, B. Girod, G. Greiner, and H. Niemann, Eds. Boston: Kluwer Academic Publishers, 2000, ch. 4.7, pp. 192–202.

[73] ——, "From scattered samples to smooth surfaces," in *Proceedings of the Forth Israel-Korea Bi-National Conference on Geometric Modeling and Computer Graphics*, D. Cohen-Or, N. Dyn, G. Elber, and A. Shamir, Eds., Tel Aviv, Israel, February 2003, pp. 1–5.

[74] J. Hoschek, "Intrinsic parametrization for approximation," *Comput. Aided Geom. Des.*, vol. 5, no. 1, pp. 27–31, 1988.

[75] J. Hoschek and U. Dietz, "Smooth B-spline surface approximation to scattered data," in *Reverse Engineering*, 1996, pp. 143–151.

[76] J. Huang, M. Zhang, J. Ma, X. Liu, L. Kobbelt, and H. Bao, "Spectral quadrangulation with orientation and alignment control," *ACM Trans. Graph.*, vol. 27, no. 5, pp. 1–9, 2008.

[77] A. Hubeli and M. H. Gross, "A survey of surface representations for geometric modeling, technical report no. 335," ETH Z urich, Computer Science Department, Tech. Rep., 2000.

[78] T. Huysmans, J. Sijbers, and B. Verdonk, "Parameterization of tubular surfaces on the cylinder," *Journal of the Winter School of Computer Graphics*, vol. 13, no. 3, pp. 97–104, 2005.

[79] K. Ikeuchi, A. Nakazawa, K. Hasegawa, and T. Ohishi, "The great buddha project: modeling cultural heritage for VR systems through observation," in *Mixed and Augmented Reality, 2003. Proceedings. The Second IEEE and ACM International Symposium on*, 2003, pp. 7–16.

[80] Initial Graphics Exchange Specification(IGES), http://ts.nist.gov/standards/iges/.

[81] J.Lin and Z. Huang, "Adaptive scattered data interpolation with multilevel nonuniform B-Splines," in *EUROGRAPHICS '99 Short Papers*, 1999.

[82] T. R. Jones, F. Durand, and M. Desbrun, "Non-iterative, feature-preserving mesh smoothing," in *ACM SIGGRAPH 2003 Papers*. San Diego, California: ACM, 2003, pp. 943–949. [Online]. Available: http://portal.acm.org/citation.cfm?id=882367

[83] T. Ju, "Robust repair of polygonal models," in *ACM SIGGRAPH 2004 Papers*. Los Angeles, California: ACM, 2004, pp. 888–895.

[84] F. Kälberer, M. Nieser, and K. Polthier, "QuadCover - surface parameterization using branched coverings," *Computer Graphics Forum*, vol. 26, no. 3, pp. 375–384, 2007.

[85] L. Kettner, "Using generic programming for designing a data structure for polyhedral surfaces," *Comput. Geom. Theory Appl.*, vol. 13, no. 1, pp. 65–90, 1999. [Online]. Available: http://portal.acm.org/citation.cfm?id=316664

[86] Konica Minolta 3-D Digitizer, http://www.minolta3d.com.

[87] V. Krishnamurthy and M. Levoy, "Fitting smooth surfaces to dense polygon meshes," in *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, 1996, pp. 313–324.

[88] Y. Lai, S. Hu, and H. Pottmann, "Surface fitting based on a feature sensitive parametrization," *Computer-Aided Design*, vol. 38, no. 7, pp. 800–807, Jul. 2006.

[89] S. Lee, G. Wolberg, and S. Y. Shin, "Scattered data interpolation with multilevel B-Splines," *IEEE Transactions on Visualization and Computer Graphics*, vol. 3, no. 3, pp. 228–244, July September 1997.

[90] M. Levoy, K. Pulli, B. Curless, S. Rusinkiewicz, D. Koller, L. Pereira, M. Ginzton, S. Anderson, J. Davis, J. Ginsberg, J. Shade, and D. Fulk, "The digital Michelangelo project: 3D scanning of large statues," in *Siggraph 2000, Computer Graphics Proceedings*, K. Akeley, Ed. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000, pp. 131–144. [Online]. Available: citeseer.csail.mit.edu/levoy00digital.html

[91] W. Li, N. Ray, and B. Lévy, "Automatic and interactive mesh to t-spline conversion," in *Proceedings of the fourth Eurographics symposium on Geometry processing*. Cagliari, Sardinia, Italy: Eurographics Association, 2006, pp. 191–200.

[92] H. Lin, W. Chen, and H. Bao, "Adaptive patch-based mesh fitting for reverse engineering," *Computer-Aided Design*, vol. 39, no. 12, pp. 1134–1142, Dec. 2007.

[93] R. Ling, W. Wang, and D. Yan, "Fitting sharp features with loop subdivision surfaces," *Computer Graphics Forum*, vol. 27, no. 5, pp. 1383–1391, Jul. 2008.

[94] N. Litke, A. Levin, and P. Schröder, "Fitting subdivision surfaces," in *Proceedings of the conference on Visualization '01*. San Diego, California: IEEE Computer Society, 2001, pp. 319–324.

[95] C. Loop, "Smooth spline surfaces over irregular meshes," in *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*. ACM, 1994, pp. 303–310.

[96] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3D surface construction algorithm," *SIGGRAPH Comput. Graph.*, vol. 21, no. 4, pp. 163–169, 1987. [Online]. Available: http://portal.acm.org/citation.cfm?id=37422

[97] M. Lounsbery, S. Mann, and T. DeRose, "Parametric surface interpolation," *IEEE Comput. Graph. Appl.*, vol. 12, no. 5, pp. 45–52, 1992. [Online]. Available: http://portal.acm.org/citation.cfm?id=617779

[98] T. Lyche, *Knot Removal for Spline Curves and Surfaces*. Academic Press, New York, 1992, ch. Approximation Theory VII, pp. 207–226.

[99] T. Lyche and K. Mørken, "Knot removal for parametric B-spline curves and surfaces." *Computer Aided Geometric Design*, vol. 4, no. 3, pp. 217–230, 1987.

[100] W. Ma and J. P. Kruth, "Parameterization of randomly measured points for least squares fitting of B-spline curves and surfaces," *Computer-Aided Design*, vol. 27, no. 9, pp. 663–675, 1995.

[101] J. Maillot, H. Yahia, and A. Verroust, "Interactive texture mapping," in *SIGGRAPH '93: Proceedings of the 20th annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM Press, 1993, pp. 27–34.

[102] M. Marinov and L. Kobbelt, "Optimization methods for scattered data approximation with subdivision surfaces," *Graph. Models*, vol. 67, no. 5, pp. 452–473, 2005.

[103] M. J. Milroy, C. Bradley, G. W. Vickers, and D. J. Weir, "G^1 continuity of B-spline surface patches in reverse engineering." *Computer-Aided Design*, vol. 27, no. 6, pp. 471–478, 1995.

[104] H. P. Moreton and C. H. Séquin, "Functional optimization for fair surface design," in *SIGGRAPH '92: Proceedings of the 19th annual conference on Computer graphics and interactive techniques*. ACM Press, 1992, pp. 167–176.

[105] F. Nooruddin and G. Turk, "Simplification and repair of polygonal models using volumetric techniques," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 9, no. 2, pp. 191–205, 2003.

[106] Y. Ohtake, A. Belyaev, M. Alexa, G. Turk, and H. Seidel, "Multi-level partition of unity implicits," *ACM Trans. Graph.*, vol. 22, no. 3, pp. 463–470, 2003.

[107] Y. Ohtake, A. Belyaev, and H. Seidel, "Ridge-valley lines on meshes via implicit surface fitting," *ACM Trans. Graph.*, vol. 23, no. 3, pp. 609–612, 2004. [Online]. Available: http://portal.acm.org/citation.cfm?id=1015706.1015768

[108] H. Park, "An approximate lofting approach for B-Spline surface fitting to functional surfaces," *The International Journal of Advanced Manufacturing Technology*, vol. 18, no. 7, pp. 474–482, Oct. 2001.

[109] J. Peters, "Constructing c^1 surfaces of arbitrary topology using biquadratic and bicubic splines," in *Designing Fair Curves and Surfaces*, N. Sapidis, Ed. SIAM, 1994, pp. 277–293.

[110] L. Piegl and W. Tiller, *The NURBS Book*. Springer-Verlag, 1997.

[111] L. A. Piegl and W. Tiller, "Reducing the number of control points in surface interpolation," *IEEE Computer Graphics and Applications*, vol. 20, no. 5, pp. 70–74, 2000.

[112] ——, "Surface approximation to scanned data." *The Visual Computer*, vol. 16, no. 7, pp. 386–395, 2000.

[113] L. A. Piegl and A. M. Richard, "Tessellating trimmed surfaces," *Computer-Aided Design*, vol. 27, no. 1, pp. 16–26, 1995.

[114] H. Prautzsch, "Freeform splines," *Computer Aided Geometric Design*, vol. 14, no. 3, pp. 201–206, Apr. 1997.

[115] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes in C: The Art of Scientific Computing*, 2nd ed. Cambridge University Press, January 1993.

[116] V. Raja and K. J. Fernandes, *Reverse Engineering: An Industrial Perspective*, 1st ed. Springer, 2008.

[117] L. Ramshaw, "Blossoms are polar forms." *Computer Aided Geometric Design*, vol. 6, no. 4, pp. 323–358, 1989.

[118] N. Ray, W. C. Li, B. Lévy, A. Sheffer, and P. Alliez, "Periodic global parameterization," *ACM Trans. Graph.*, vol. 25, no. 4, pp. 1460–1485, 2006.

[119] U. Reif, "TURBS – topologically unrestricted rational b-splines," *Constr. Approx*, vol. 14, pp. 57—78, 1998.

[120] R. F. Riesenfeld, "Application of b-spline approximation to geometric problems of computer aided design," Ph.D. dissertation, Syracuse University, Syracuse, NY, 1973.

[121] A. Rockwood, K. Heaton, and T. Davis, "Real-time rendering of trimmed surfaces," in *Proceedings of the 16th annual conference on Computer graphics and interactive techniques*. ACM, 1989, pp. 107–116.

[122] D. F. Rogers, *An introduction to NURBS: with historical perspective*. Morgan Kaufmann Publishers Inc., 2001. [Online]. Available: http://portal.acm.org/citation.cfm?id=347021

[123] F. Sadlo, T. Weyrich, R. Peikert, and M. Gross, "A practical structured light acquisition system for point-based geometry and texture," in *Point-Based Graphics, 2005. Eurographics/IEEE VGTC Symposium Proceedings*, 2005, pp. 89–145.

[124] P. V. Sander, J. Snyder, S. J. Gortler, and H. Hoppe, "Texture mapping progressive meshes," in *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. ACM, 2001, pp. 409–416. [Online]. Available: http://portal.acm.org/citation.cfm?id=383307

[125] B. Sarkar and C.-H. Menq, "Smooth-surface approximation and reverse engineering." *Computer-Aided Design*, vol. 23, no. 9, pp. 623–628, 1991.

[126] F. J. M. Schmitt, B. A. Barsky, and W. hui Du, "An adaptive subdivision method for surface-fitting from sampled data," in *SIGGRAPH '86: Proceedings of the 13th annual conference on Computer graphics and interactive techniques*. ACM Press, 1986, pp. 179–188.

[127] I. Schoenberg, "Contributions to the problem of approximation of equidistant data by analytic functions," *Quarterly of Applied Mathematics*, vol. 4, no. 1,2, pp. 45–99,112–141, 1946.

[128] T. Sederberg, D.Cardon, G.Finnigan, N.North, J. Zheng, and T. Lyche, "T-spline simplification and local refinement," *ACM Transactions on Graphics (SIGGRAPH 2004)*, vol. 23, no. 3, pp. 276–283, 2004.

[129] T. Sederberg, J. Zheng, A. Bakenov, and A. Nasri, "T-splines and T-NURCCs," *ACM Transactions on Graphics (SIGGRAPH 2003)*, vol. 22, no. 3, pp. 477–484, 2003.

[130] T. W. Sederberg, G. T. Finnigan, X. Li, H. Lin, and H. Ipson, "Watertight trimmed NURBS," in *ACM SIGGRAPH 2008 papers*. Los Angeles, California: ACM, 2008, pp. 1–8. [Online]. Available: http://portal.acm.org/citation.cfm?id=1360678

[131] T. W. Sederberg, J. Zheng, D. Sewell, and M. Sabin, "Non-uniform recursive subdivision surfaces," in *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*. ACM, 1998, pp. 387–394. [Online]. Available: http://portal.acm.org/citation.cfm?id=280942

[132] T. W. Sederberg, J. Zheng, and X. Song, "Knot intervals and multi-degree splines," *Computer Aided Geometric Design*, vol. 20, no. 7, pp. 455–468, 2003.

[133] A. Shamir, "A survey on mesh segmentation techniques," *Computer Graphics Forum*, vol. 27, pp. 1539–1556, Sep. 2008. [Online]. Available: http://www.ingentaconnect.com/content/bpl/cgf/2008/00000027/00000006/art00006

[134] A. Sheffer, B. Lévy, M. Mogilnitsky, and A. Bogomyakov, "ABF++: fast and robust angle based flattening," *ACM Trans. Graph.*, vol. 24, no. 2, pp. 311–330, 2005. [Online]. Available: http://portal.acm.org/citation.cfm?id=1061347.1061354

[135] A. Sheffer, E. Praun, and K. Rose, "Mesh parameterization methods and their applications," *Found. Trends. Comput. Graph. Vis.*, vol. 2, no. 2, pp. 105–171, 2006. [Online]. Available: http://portal.acm.org/citation.cfm?id=1295187

[136] C. Shen, J. F. O'Brien, and J. R. Shewchuk, "Interpolating and approximating implicit surfaces from polygon soup," in *ACM SIGGRAPH 2004 Papers*. Los Angeles, California: ACM, 2004, pp. 896–904.

[137] X. Sheng and B. Hirsch, "Triangulation of trimmed surfaces in parametric space," *Computer-Aided Design*, vol. 24, no. 8, pp. 437–444, Aug. 1992.

[138] D. Shepard, "A two-dimensional interpolation function for irregularly-spaced data," in *Proceedings of the 1968 23rd ACM national conference*. ACM Press, 1968, pp. 517–524.

[139] K. Shimada and D. C. Gossard, "Automatic triangular mesh generation of trimmed parametric surfaces for finite element analysis*1," *Computer Aided Geometric Design*, vol. 15, no. 3, pp. 199–222, Mar. 1998.

[140] S. S. Sinha and P. Seneviratne, "Single valuedness, parameterization, and approximating 3D surfaces using b-splines," vol. 2031, Jun. 1993, pp. 193–204.

[141] I. Söderkvist, "Introductory overview of surface reconstruction methods," Department of Mathematics, Lulea University of Technology, Lulea, Sweden., Tech. Rep. 1999-10, 1999, can be retrieved from http://www.sm.luth.se/ inge/publications/-surfrec.ps. [Online]. Available: citeseer.ist.psu.edu/soderkvist99introductory.html

[142] SOFTIMAGE 3D,
www.softimage.com.

[143] W. Song and X. Yang, "Free-form deformation with weighted t-spline," *The Visual Computer*, vol. 21, no. 3, pp. 139–151, Apr. 2005. [Online]. Available: http://dx.doi.org/10.1007/s00371-004-0277-8

[144] The Dragon Wing, http://www.cs.ubc.ca/nest/imager/contributions/forsey/dragon/top.html.

[145] S. Toledo, D. Chen, and V. Rotkin, "TAUCS, a library of sparse linear solvers," http://www.tau.ac.il/ stoledo/taucs/.

[146] Y. Tong, P. Alliez, D. Cohen-Steiner, and M. Desbrun, "Designing quadrangulations with discrete harmonic forms," in *Proceedings of the fourth Eurographics symposium on Geometry processing*. Cagliari, Sardinia, Italy: Eurographics Association, 2006, pp. 201–210.

[147] G. Turk and M. Levoy, "Zippered polygon meshes from range images," in *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*. ACM, 1994, pp. 311–318.

[148] H. A. van der Vorst, "BI-CGSTAB: a fast and smoothly converging variant of bi-cg for the solution of nonsymmetric linear systems," *SIAM J. Sci. Stat. Comput.*, vol. 13, no. 2, pp. 631–644, 1992. [Online]. Available: http://portal.acm.org/citation.cfm?id=131930

[149] T. Várady, R. R. Martin, and J. Cox, "Reverse engineering of geometric models - An introduction," *Computer-Aided Design*, vol. 29, no. 4, pp. 255–268, April 1997.

[150] L. Velho, L. H. de Figueiredo, and J. Gomes, "A unified approach for hierarchical adaptive tesselation of surfaces," *ACM Trans. Graph.*, vol. 18, no. 4, pp. 329–360, 1999.

[151] J. S. M. Vergeest, "Surface fitting for interactive shape design," *Comput. Ind.*, vol. 13, no. 1, pp. 1–13, 1989.

[152] K. Versprille, "Computer-aided design applications of the rational b-spline approximation form," Ph.D. dissertation, Syracuse University, Syracuse, NY, February 1975.

[153] H. Wang, Y. He, X. Li, X. Gu, and H. Qin, "Polycube splines," *Computer-Aided Design*, vol. 40, no. 6, pp. 721–733, Jun. 2008.

[154] K. Watanabe and A. G. Belyaev, "Detection of salient curvature features on polygonal surfaces," *Computer Graphics Forum*, vol. 20, no. 3, pp. 385–392, 2001. [Online]. Available: http://dx.doi.org/10.1111/1467-8659.00531

[155] K. Weiler, "Edge-based data structures for solid modeling in curved-surface environments," *IEEE Computer Graphics and Applications*, vol. 5, no. 1, pp. 21–40, jan 1985.

[156] D. J. Weir, M. J. Milroy, C. Bradley, and G. W. Vickers, "Wrap-around B-spline surface fitting to digitized data with applications to reverse engineering," *Journal of Manufacturing Science and Engineering*, vol. 122, no. 2, pp. 323–330, 2000.

[157] V. Weiss, L. Andor, G. Renner, and T. Várady, "Advanced surface fitting techniques." *Computer Aided Geometric Design*, vol. 19, no. 1, pp. 19–42, 2002.

[158] W. Welch and A. Witkin, "Variational surface modeling," in *SIGGRAPH '92: Proceedings of the 19th annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM Press, 1992, pp. 157–166.

[159] C. D. Woodward, "Skinning techniques for interactive B-spline surface interpolation," *Comput. Aided Des.*, vol. 20, no. 10, pp. 441–451, 1988.

[160] H. Yang and B. Jüttler, "3D shape metamorphosis based on t-spline level sets," *The Visual Computer*, vol. 23, no. 12, pp. 1015–1025, Dec. 2007. [Online]. Available: http://dx.doi.org/10.1007/s00371-007-0168-x

[161] L. Ying and D. Zorin, "A simple manifold-based construction of surfaces of arbitrary smoothness," *ACM Trans. Graph.*, vol. 23, no. 3, pp. 271–275, 2004.

[162] G. Yngve and G. Turk, "Robust creation of implicit surfaces from polygonal meshes," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 8, no. 4, pp. 346–359, 2002.

[163] S. Yoshizawa, A. Belyaev, H. Yokota, and H. Seidel, "Fast, robust, and faithful methods for detecting crest lines on meshes," *Comput. Aided Geom. Des.*, vol. 25, no. 8, pp. 545–560, 2008. [Online]. Available: http://portal.acm.org/citation.cfm?id=1435006.1435164

[164] S. Zhang and S. Yau, "Three-dimensional shape measurement using a structured light system with dual cameras," *Optical Engineering*, vol. 47, no. 1, pp. 013 604–12, 2008. [Online]. Available: http://link.aip.org/link/?JOE/47/013604/1

[165] W. Zhang, Z. Tang, and J. Li, "Adaptive hierarchical B-spline surface approximation of large-scale scattered data." in *Pacific Conference on Computer Graphics and Applications*, 1998, pp. 8–16.

[166] J. Zheng and T. W. Sederberg, "Estimating tessellation parameter intervals for rational curves and surfaces," *ACM Trans. Graph.*, vol. 19, no. 1, pp. 56–77, 2000.

[167] M. Zöckler, D. Stalling, and H.-C. Hege, "Fast and intuitive generation of geometric shape transitions," *The Visual Computer*, vol. 16, no. 5, pp. 241–253, 2000.

[168] D. Zorin and P. Schröder, "Subdivision for Modeling and Animation," SIGGRAPH 2000, Tech. Rep., 2000, course Notes.

Author's Publications

1. Y. Wang, J. Zheng, and H. S. Seah, "Conversion between T-Splines and hierarchical B-Splines," in *Computer Graphics and Imaging*, Hawaii, USA, 2005, pp. 8–13.

2. J. Zheng, Y. Wang, and H. S. Seah, "Adaptive T-spline surface fitting to z-map models," in *GRAPHITE '05*, ACM, 2005, pp. 405–411.

3. Y. Wang and J. Zheng, "Control point removal algorithm for T-spline surfaces," *Lecture Notes in Computer Science (GMP'2006)*, Vol. 4077, 2006, pp. 385–396.

4. Y. Wang and J. Zheng, "Adaptive T-spline surface approximation of triangular meshes," in *Proceedings of the 6th International Conference on Information, Communications & Signal Processing*, 2007.

5. Y. Wang and J. Zheng, "Edge based parameterization for tubular meshes," in *Proceedings of the 7th ACM SIGGRAPH International Conference on Virtual-Reality Continuum and Its Applications in Industry*, ACM, 2008.

6. Y. Wang and J. Zheng, "Tubular triangular mesh parameterization and applications," *Computer Animation and Virtual Worlds*, Vol. 20, 2009.

7. Y. Wang and J. Zheng, "Using periodic T-spline surfaces for virtual reality applications," in *International Conf. of ISAGA*, 2009.

8. Y. Wang and J. Zheng, "Curvature guided adaptive T-spline surface fitting," Tech. Rep., 2009.