Deforming NURBS Surfaces & B-rep Models

A THESIS SUBMITTED TO THE UNIVERSITY OF MANCHESTER

FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

IN THE FACULTY OF SCIENCE AND ENGINEERING

April 2003

YingLiang MA

Department of Computer Science

Contents

List of Figures

Abstract

UNIVERSITY OF MANCHESTER

ABSTRACT OF THESIS submitted by **YingLiang MA** for the Degree of Doctor of Philosophy and entitled **Deforming NURBS Surfaces & B-rep Models.**

Month and Year of Submission: April 2003

Object deformation is an important technique in computer graphics and CAD. In this thesis, a metaball deformation method on NURBS surfaces and B-rep models is presented. This method enables the user to interactively deform the NURBS object by specifying a series of constraints, which may consist of points, lines, curves and surfaces, their effective radii and maximum displacements, and the deformation model creates a generalized metaball for each constraint.

This thesis presents several research contributions relative to metaball deformation models. Point projection for NURBS object is introduced as a method for calculating the minimum distance between the 3D test point and NURBS objects. A new tessellation method for trimmed NURBS surfaces is presented. Finally, an extended method for the deformation of B-rep models is discussed. It provides a flexible method to change the shape of CAD solid models.

Declaration

No portion of the work referred to in this thesis has been submitted in support of an application for another degree of qualification of this or any other university or other institution of learning.

Copyright & Ownership

1. Copyright in text of this thesis rests with the Author. Copies (by any process) either in full, or of extracts, may be made **only** in accordance with instructions given by the Author and lodged in the John Rylands University Library of Manchester. Details may be obtained from the Librarian. This page must form part of any such copies made. Further copies (by any process) of copies made in accordance with such instructions may not be made without the permission (in writing) of the Author.

2. The ownership of any intellectual property rights which may be described in this thesis is vested in the University of Manchester, subject to any prior agreement to the contrary, and may not be made available for use by third parties without the written permission of the University, which will prescribe the terms and conditions of any such agreement.

   Further information on the conditions under which disclosures and exploitation may take place is available from the Head of the Department of Computer Science.

Dedication

To my parents, thanks for all your support.

Acknowledgement

First and foremost, I would express my sincere thanks to my supervisor, Mr. W. T. Hewitt, for his guidance during my research work and during the writing up of this thesis.

I should also like to thank all staffs in the Manchester Visualization Centre, who have given me assistance. I should especially like to thank Collin C. Venters, George Leaver, Yien Kwok, Mark Riding and Mary McDerby.

Thanks are also due to Robert Haines for proof reading my papers and discussing ideas during last two years. Thanks also go to all my friends, particularly, Jie Fang, Zhong Chuan Yu, ZhiMing Pan, XiaoXia Ni,Yang Xu and ShiYuan Shen. They bring such good fun in the last three years.

I would also thank Dr. XiaoGang Chen, the senior lecture in department of textiles (UMIST), created the opportunity to implement my NURBS library into their CAD system. Thanks also go to all members in CAD/CAM lab, textiles department, UMIST, especially to Dr. XinCai Tan and XueGong Ai.

I must also thank my girlfriend Ning Li and my parents for providing for their support and sacrifice during the lengthy production of this thesis.

Finally, I should like to thank the Department of Computer Science for the financial assistance.

Chapter 1

Introduction

1.1 Background

The use of computers to aid the design and manufacture of parts has a history of more than 30 years [75]. The first 3D CAD system appeared in the early 1970's, and it used wire frames to display the 3D model. Since then, the various systems have become better, faster and cheaper. The rendering of 3D objects has been developed from wire frame through wire frame with hidden line removal, two and half dimensions, 3D surfaces to 3D solid models [75]. Nowadays, CAD techniques are widely used in almost every manufacturing industry particularly in automobile, aeronautics and marine industry.

Thus the design and visualization of 3D objects is a major research area for CAD and Computer Graphics. Design refers to the creation and modification process, and the task of the visualization is to show the 3D objects on the 2D computer screen. Much of the research in design process is to create the algorithms for the design, modification and assembling of models. The algorithms for visualizing 3D objects are designed to create as many fast and realistic effects as possible on the 2D computer screen.

In the last few years, surface manipulation methods have been developed which not only professionals but also laymen can use. They do not require a user to have a detailed knowledge of the surface which they are manipulating [75]. This thesis presents a number of issues with respect to surface design and modification method as well as method of trimmed surface visualization. They can be categorised as geometry representation, deformation, surface visualization and reverse engineering.

- **Geometry Representation**

  By the late 1970s, the CAD/CAM industry recognized the need for a modeller that had a common internal method of representing and storing different geometric entities. The solution is to use NURBS (**N**on **U**niform **R**ational **B-S**pline) [1]. NURBS are the best available mathematical form for representation of both analytical shapes and free-form curves or surfaces.

- **Deformation**

  Although much progress has been made in the area of 3D surface modelling, creating complex free-form surfaces is still very difficult and tedious [2]. Deformation provides a more flexible method to construct a surface from a skeleton in an interactive environment. In computer animation, morphing based on the deformation technique presents an amazing way to transform one object to another [2].

- **Surface Visualization**

  The primary purpose of 3D computer graphics is to produce a 2D image of a scene or an object from a description or model of the 3D object. In a CAD system, surface visualization can assist the user to create the model in an interactive way by displaying the wire-frame or rendered objects. The surfaces in the model can have some complex shapes with holes and curved boundaries. For rendering the surface, the surface is usually

tessellated into a set of triangles or quadrilaterals. Another application of surface visualization is to assist the engineer to analyse the surface to find out both geometric and physical properties.

- **Reverse Engineering**

  The need for reverse engineering comes about for a variety of reasons; chief among them is the need to replace a broken or obsolete part that is no longer available from the original manufacturer. In the reverse engineering of an object's form, 3D data can be collected by a touch probe or laser range sensor (laser scanner). Surface fitting is used to construct the surface from the set of sampled point data. If we use NURBS or other types of parametric surface, we need to recover the parameters of the sampled points by applying the point projection method. However, projecting the sampled points to the surface is very expensive and numerical computation is not very stable [57].

This thesis focuses on the research of providing a deformation tool for CAD based on metaballs model [3][4][5][6][7][8]. The author also covers the visualization of a metaballs model and geometric analysis of the metaballs model, as well as one improved traditional algorithm.

1.2 Main Contributions

The main contributions for metaballs deformation applied on the NURBS surface and solid model described by the author are:

1. **Point Inversion and Projection for NURBS Curves and Surface**

   Point inversion and projection for curves or surfaces is a fundamental problem in curve and surface fitting, robotics, animation and interactive systems. The central problem of point inversion and projection is to calculate the minimum distance between the test point and a NURBS curve or surface. The method [17] widely used at the moment uses an iterative method based upon the Newton-Raphson method. Figure 1.1 shows the results of our method.

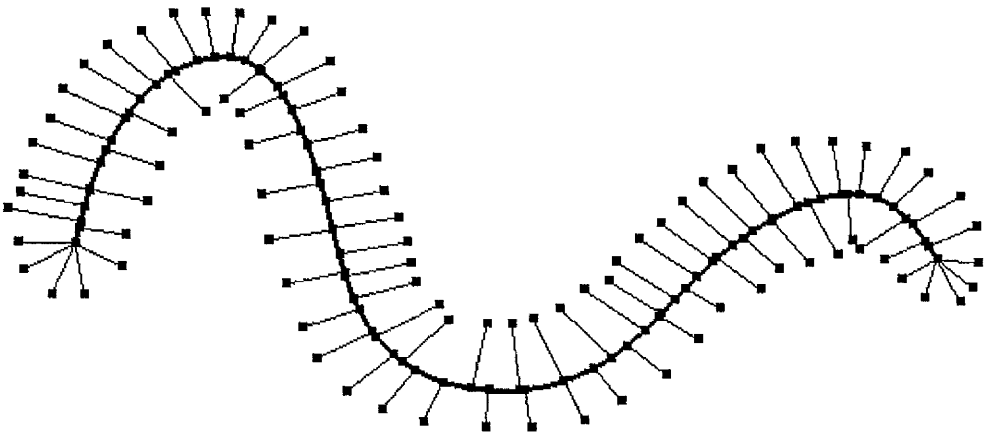


Figure 1.1: Point projection for NURBS curve.

However, good initial values must be given to achieve convergence. It is difficult to get such values due to the complex shape of NURBS curves or surfaces.

The aim of author's approach is to provide the good initial value. The NURBS curve or surface is first subdivided into a set of Bézier curves or

patches. By analysing the relationship between the test point and the control polygon of Bézier curve or the control point net of the Bézier patch, the candidate Bézier curves or patches are extracted and the approximate candidate points are calculated. Finally, by comparing the distances between the test point and candidate points, the closest point is found. The accuracy of the closest point can be improved by applying the Newton-Raphson method to it.

This pre-processing means less iterations and this approach achieves better results both in efficiency and stability than the traditional methods. It also is applied in the metaballs model to calculate the distance function.

2. Adaptive Tessellation for Trimmed NURBS Surface

Trimmed surfaces have a fundamental role in computer-aided design [9][10][11]. Most complex objects are generated by some sort of trimming or scissoring process. Trimmed patches are also the result of Boolean operations on solid objects bounded by NURBS surfaces. In the CAD pipeline, the trimmed patch undergoes a number of processes such as rendering for visualization, cutter patch generation, area computation or rapid prototyping. The simplest method to accomplish all of this is to approximate the trimmed patch by triangular facets to within a user given tolerance.

There are two ways to subdivide the trimmed patch: uniform or non-uniform. Uniform tessellation is to sample the surface at uniform parametric intervals. However this often leads to regions of a surface which are either overcomplicated or undersampled. Non-uniform (adaptive) tessellation focuses more attention in regions of highest curvature. A minimal number of polygons can be generated for a particular subdivision tolerance.

The author presents an adaptive tessellation method for trimmed NURBS surfaces. Based on the subdivision of a NURBS surface by using the knot insertion algorithm, the surface is tessellated into a set of 'flat' enough Bézier patches [24]. A Scanline algorithm is applied to remove the patches inside the inner trimming loop. The result of tessellation is both quadrilaterals and triangles which can be passed into the rendering pipeline (figure 1.2).

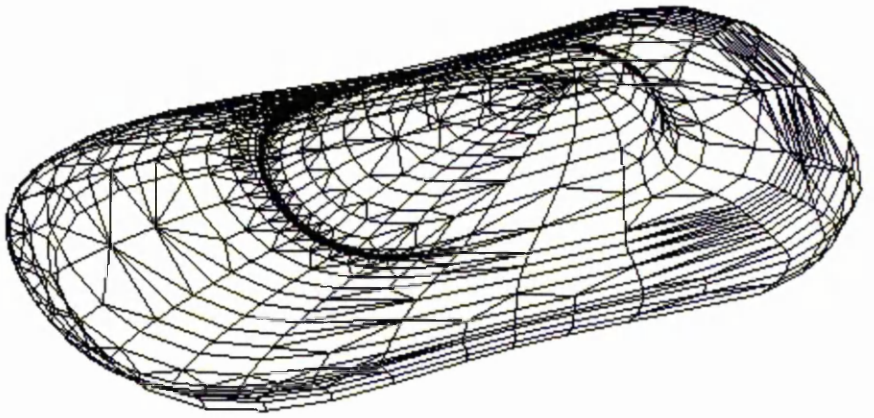


Figure 1.2: The adaptive tessellation for a car model

This method is performed completely in parametric space, and furthermore it does not adopt any complex methods to generate triangles, so that the procedure runs fast and reliably.

3. Generalized Metaballs Deformation on The Solid Model

In an interactive CAD system, the shape of object can be finely controlled by interactively adjusting the positions of its vertices or control vertices. However, to most users, this manipulation is tedious and inefficient.

The most popular deformation method is the free-form deformation (FFD) technique developed by Sederberg and Parry [12]. FFD is typically conducted by embedding an object to be deformed into a parametric space of a trivariate Bézier volume whose control points are organized as a

lattice, the deformation of the object being obtained by moving the control points of the trivariate Bézier volume. However in FFD the user is forced to define some control points in the space to be deformed.

To overcome the disadvantages of FFD-based method, Xiaogang Jin, Youfu Li and Qunsheng Peng [8] developed a constrained deformation model based on generalized metaballs. In their method, constraints are generalized to include points, lines, surfaces and volumes. The user need only define a set of constraints with desired displacements and an effective radius associated with each constraint. However, their method is applied on the mesh model rather than any parametric surface or solid model.

The author extends this method to the trimmed NURBS surface and further to the solid model. The new method modifies the positions of control points on the NURBS surface so that it obtains a more accurate model after deformation than the mesh model. The visualization of geometric properties of the deformed surface is given to assist the user to analyse the quality of the NURBS surface.

1.3 Thesis Outline

The remainder of this thesis is organized as follows.

Chapter 2 gives a brief introduction of the fundamentals of NURBS curves and surfaces, including the definitions and properties of NURBS curves and surfaces, the basic algorithm for NURBS, the fundamental surface construction techniques and multiple trimmed NURBS surfaces (B-Rep model).

The next chapter describes the mathematical background of general constrained deformations based on the generalized metaball. This chapter also reviews other deformation and surface manipulation methods.

The subsequent three chapters devote themselves to the previously described research contributions. For each chapter, some brief background information on the problem is given and the approach to the problem is presented.

Chapter 4 presents a novel solution for point inversion and projection for NURBS curves and surfaces.

Chapter 5 reviews methods for the tessellation of trimmed NURBS surfaces, presents an adaptive method and gives the results and conclusions.

Chapter 6 describes the method for applying the metaball model on the solid model

Chapter 7 summarises the results of the research on the metaball model, along with conclusions and further work.

The appendix presents an overview of the two utility libraries which are used in my testbed software.

All images and geometric figures appearing in this thesis have been produced using the author's prototype surface design and deformation software, except those highlighted.

Chapter 2

Introduction to NURBS

In this Chapter, we first introduce the definitions and properties of NURBS curves and surfaces. Geometric algorithms, common surface construction techniques and the B-Rep model are also introduced in this chapter, which are related to the metaball model described in the next four chapters.

2.1 B-Spline Curves and Surfaces

We start with the B-Spline curves and surfaces. More detailed discussions about B-Spline curves and surfaces can be found in [13][14].

2.1.1 Definition of B-Spline Curves and Surfaces

A p^{th}-*degree* B-Spline curve is defined by

$$\boldsymbol{C}(u) = \sum_{i=0}^{n} N_{i,p}(u)\boldsymbol{P}_i \qquad a \le u \le b \qquad (2.1)$$

where the $\{\boldsymbol{P}_i\}$ are the control points in 3D Euclidean space, and the $\{N_{i,p}(u)\}$ are the pth-degree B-Spline basis functions defined on a knot vector

$$N_{i,0}(u) = \begin{cases} 1 & \text{if } u_i \le u \le u_{i+1} \\ 0 & \text{otherwise} \end{cases}$$

(m+1 knots) $\boldsymbol{U} = \{u_j\}_{j=0}^{m=n+p+1}$ by the Cox-deBoor recurrence relations [13][15].

$\boldsymbol{C}(u)$ is a point on the curve corresponding to the parameter $u \in \boldsymbol{U}$.

$$N_{i,p}(u) = \frac{u-u_i}{u_{i+p}-u_i} N_{i,p-1}(u) + \frac{u_{i+p+1}-u}{u_{i+p+1}-u_{i+1}} N_{i+1,p-1}(u) \qquad (2.2)$$

We now list a number of important properties of the B-Spline basis functions.

- **Local support:** $N_{i,p}(u) = 0$ if u is outside the interval $[u_i, u_{i+p+1})$.
- **Non-negativity:** $N_{i,p}(u) \ge 0$ for all i, p and u.
- **Partition of unity:** $\sum_{i=0}^{n} N_{i,p}(u) = 1$ for all $u \in U$.
- **Differentiability:** in the interior of a knot span $[u_i, u_{i+1})$, $N_{i,p}(u)$ is continuously differentiable. At an interior knot, $N_{i,p}(u)$ is $p-k$ times continuously differentiable where k is the multiplicity of the knot.
- **Extrema:** for $p \ne 0$, $N_{i,p}(u)$ attains exactly one maximum value.

The knot vector of a B-Spline curve has the form

$$\boldsymbol{U} = \{\underbrace{0,\ldots,0}_{p+1}, u_{p+1}, \ldots, u_n, \underbrace{1,\ldots,1}_{p+1}\}$$

which yields the endpoint interpolation, as $\boldsymbol{C}(0) = \boldsymbol{P}_0$ and $\boldsymbol{C}(1) = \boldsymbol{P}_n$.

A B-Spline surface is obtained by taking a bi-directional net of control points, two knot vectors, and the products of the univariate B-Spline functions

$$\boldsymbol{S}(u,v) = \sum_{i=0}^{n}\sum_{j=0}^{m} N_{i,p}(u) N_{j,q}(v) \boldsymbol{P}_{i,j} \qquad (2.3)$$

where, similar to (2.1), the $\{\boldsymbol{P}_{i,j}\}$ are the control net in 3D space, $N_{i,p}(u)$ and $N_{j,q}(v)$ are the normalized B-Spline basis functions of degree p and degree q in the u and v parameter directions, respectively. These basis functions are defined over the knot vectors

$$\boldsymbol{U} = \{\underbrace{0,..,0}_{p+1}, u_{p+1},..., u_n, \underbrace{1,...,1}_{p+1}\}$$

and

$$\boldsymbol{V} = \{\underbrace{0,..,0}_{q+1}, v_{q+1},..., v_m, \underbrace{1,...,1}_{q+1}\}$$

$\boldsymbol{S}(u,v)$ is a position on the surface corresponding to the parameters $u \in \boldsymbol{U}$ and $v \in \boldsymbol{V}$.

2.1.2 Properties of B-Spline Curves and Surfaces

The most significant and useful properties of B-Spline curves and surfaces are:

- **Strong convex hull property:** The whole B-Spline curve or surface is contained within the union of individual convex hulls of its segments or patches (figure 2.1).

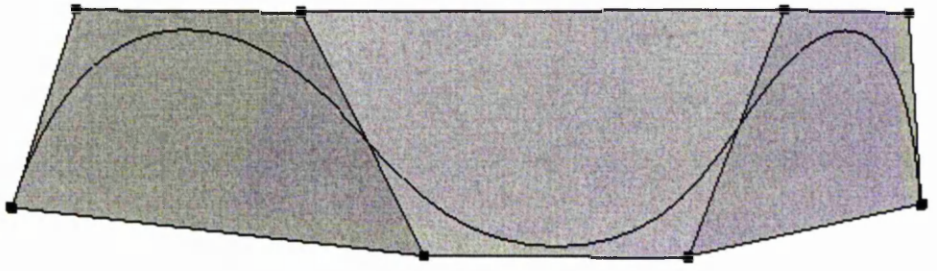


Figure 2.1: Strong convex hull property of B-Spline curve

- **Variation diminishing:** no line (plane) has more intersections with a B-Spline curve than its control polygon. This property can be extended to the surface.
- **Differentiability:** $\boldsymbol{C}(u)$ is infinitely differentiable in a knot span and is $p-k$ times differentiable at a knot (k is the multiplicity of the knot).

$\boldsymbol{S}(u,v)$ is $p-k$ $(q-k)$ times differentiable with respect to u (v) at a u knot (v knot) of multiplicity k.

- **Local modification:** The movement of a control point $\boldsymbol{P}_i$ changes the curve $\boldsymbol{C}(u)$ only in the knot interval $[u_i, u_{i+p+1})$. Figure 2.2 shows the effect of the control point $\boldsymbol{P}_3$ being moved. If a control point $\boldsymbol{P}_{i,j}$ is moved, it affects the surface $\boldsymbol{S}(u,v)$ only in the rectangle $[u_i, u_{i+p+1}) \times [v_j, v_{j+q+1})$.

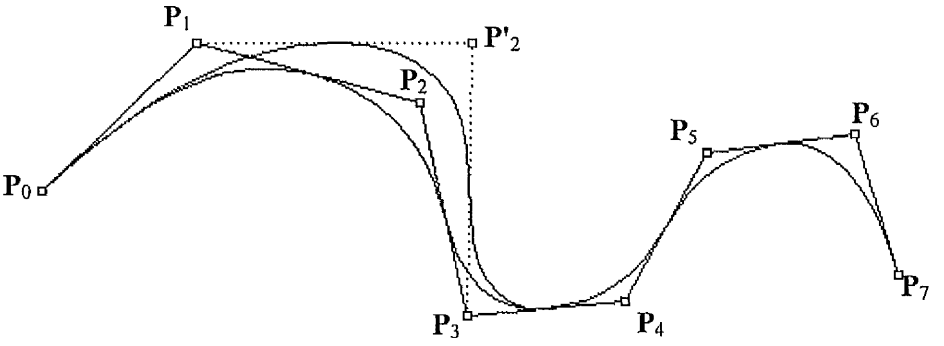


Figure 2.2: Moving a control point to change the shape of B-Spline curve

- **Affine invariance:** Affine transforms including transformations, rotations, scalings and shears can be expressed as $A[\boldsymbol{P}] = \boldsymbol{L}[\boldsymbol{P}] + \boldsymbol{v}$, where $\boldsymbol{L}$ is a 3×3 matrix, vector $\boldsymbol{v}$ and $\boldsymbol{P}$ is a point in 3D space. From (2.1) and *the partition of unity* of the B-Spline basis function, we have:

$$A[\boldsymbol{C}(u)] = A\left[\sum_{i=0}^{n} N_{i,p}(u)\boldsymbol{P}_i\right]$$
$$= \sum_{i=0}^{n} N_{i,p}(u) A[\boldsymbol{P}_i]$$

This result can be extended to a B-Spline surface, which means a B-Spline curve or surface is closed under affine transformations. An affine transformation of the control points will apply an affine transformation to the B-Spline curve or surface.

The local modification property of the B-Spline surface is very important in the metaball model in Chapter 3. The strong convex hull property plays an

important role in the algorithm of point inversion and projection for NURBS curves and surface which will be discussed in Chapter 4.

2.2 NURBS Curves and Surfaces

We can develop the non-rational forms of B-Spline to their rational forms (NURBS) by using *homogeneous coordinates*. A comprehensive discussion about NURBS and its application can be found in [16][17][1][18].

2.2.1 Homogeneous Coordinates

Before defining the NURBS curves and surfaces, we need to define the *homogeneous coordinates*. A homogeneous coordinate simply projects a 3D point into a 4D space. So, a single point $P=(x,y,z)$ is represented by $P^w=(wx,wy,wz,w)$, where w is known as the *weight*. Normally, w is not equal to zero, however in some special cases, w can be zero, which represents an infinite point in 3D space. Figure 2.3 illustrates the relationship between the 3D point $\boldsymbol{P}$ and the 4D P^w.

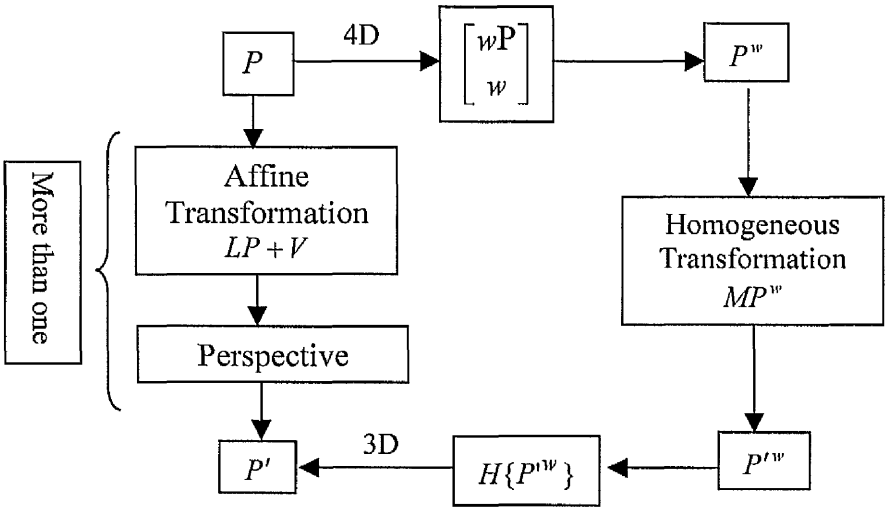


Figure 2.3: Homogeneous Transformation

In order to convert from 4D homogeneous coordinate point back to 3D point, we introduce the notation $\boldsymbol{P} = \boldsymbol{H}\{\boldsymbol{P}^w\}$ where $\boldsymbol{H}$ is a projective map from 4-space to 3-space:

$$\boldsymbol{H}\left\{\begin{pmatrix} wx \\ wy \\ wz \\ w \end{pmatrix}\right\} = \begin{pmatrix} wx/w \\ wy/w \\ wz/w \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

The advantage of representing points in E^3 as homogeneous coordinates is that *homogeneous transformations* may be applied to them. A homogeneous transformation can be expressed as a 4×4 matrix, which can combine a series of affine and perspective transformations.

2.2.2 Definition of NURBS Curves and Surfaces

A pth-degree NURBS curve is defined by:

$$\boldsymbol{C}(u) = \frac{\sum_{i=0}^{n} N_{i,p}(u) w_i \boldsymbol{P}_i}{\sum_{i=0}^{n} N_{i,p}(u) w_i} \qquad a \le u \le b \qquad (2.4)$$

where the $\{\boldsymbol{P}_i\}$ are the *control points* (control polygon), the $\{w_i\}$ are the *weights* and the $\{N_{i,p}(u)\}$ are defined as in equation (2.1).

A NURBS curve can be also expressed in homogeneous coordinates as

$$\boldsymbol{C}^w(u) = \sum_{i=0}^{n} N_{i,p}(u) \boldsymbol{P}_i^w \qquad (2.5)$$

A NURBS surface of degree p in the u direction and degree q in the v direction is defined by

$$\boldsymbol{S}(u,v) = \frac{\sum_{i=0}^{n}\sum_{j=0}^{m} N_{i,p}(u) N_{j,q}(v) w_{w,j} \boldsymbol{P}_{i,j}}{\sum_{i=0}^{n}\sum_{j=0}^{m} N_{i,p}(u) N_{j,q}(v) w_{w,j}} \qquad (2.6)$$

It can also be redefined in the homogeneous coordinates as

$$\boldsymbol{S}^{w}(u,v)=\sum_{i=0}^{n}\sum_{j=0}^{m}N_{i,p}(u)N_{j,q}(v)\boldsymbol{P}_{i,j}^{w} \qquad (2.7)$$

2.2.3 Properties of NURBS Curves and Surfaces

NURBS inherit all of the properties of B-splines. Besides these, the introduction of weight gives NURBS the following further properties:

- **Closed under perspective transformation:** Like B-Spline curves and surfaces, NURBS curves and surfaces are closed under affine transformations. Furthermore, they are also closed under perspective transformation.

- **Local shape control using weight:** The shape of NURBS curves and surfaces can be modified not only via the control points, but also by changing the weights [19][20]. If we change the weight w_i associated with the control point $\boldsymbol{P}_i$, the shape of the NURBS curve is modified only in $p+1$ knot spans $[u_i, u_{i+p+1})$. If w_i increases (decreases), the curve is pulled toward (pulled away from) $\boldsymbol{P}_i$, pushed away from (pulled toward) $\boldsymbol{P}_j (j \neq i)$. An example is shown in figure 2.4.

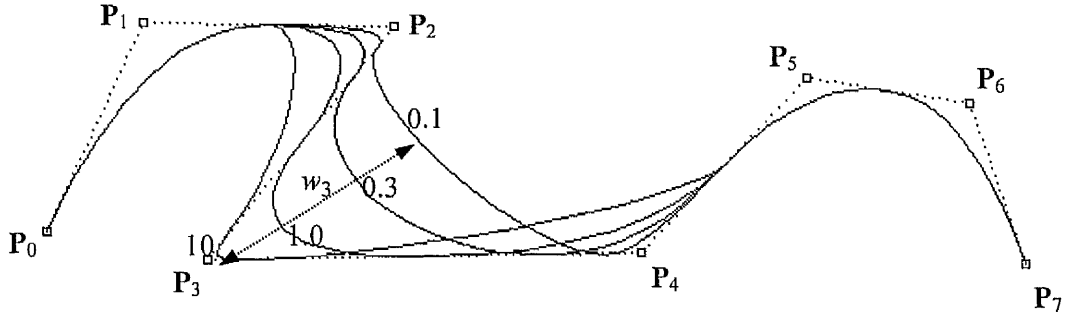
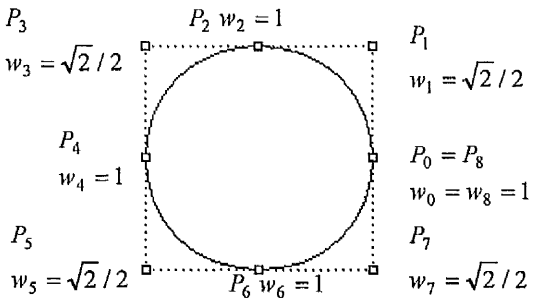


Figure 2.4: Modifying a weight to change a NURBS curve

- **NURBS representation of conic sections is exact:** NURBS can precisely represent not only the conic curves, but also the commonly used quadric surfaces by the introduction of weights. Figure 2.5 shows a NURBS circle with 9 control points.



$$U = [0,0,0,\tfrac{1}{4},\tfrac{1}{4},\tfrac{1}{2},\tfrac{1}{2},\tfrac{3}{4},\tfrac{3}{4},1,1,1]$$

Figure 2.5: NURBS representation of a full circle

2.2.4 Derivatives of NURBS Curves and Surfaces

In general, compared to the computation of derivatives of non-rational basis functions, derivatives of rational functions are complicated to compute, involving denominators with high powers. If $\boldsymbol{C}^w(u)$ is a non-rational curve in four-dimensional space, we can express the derivatives of a NURBS curve $\boldsymbol{C}(u)$ in terms of the derivatives of $\boldsymbol{C}^w(u)$. Let

$$\boldsymbol{C}(u) = \frac{w(u)\boldsymbol{C}(u)}{w(u)} = \frac{\boldsymbol{A}(u)}{w(u)}$$

Where, the numerator of (2.4), $\boldsymbol{A}(u)$ is the function whose coordinates are the first three coordinates of $\boldsymbol{C}^w(u)$.

If $\boldsymbol{C}^{(k)w}(u)$ is the kth derivative of $\boldsymbol{C}^w(u)$, then

$$\boldsymbol{C}^{(k)w}(u) = \sum_{i=0}^{n} N_{i,p}^{(k)}(t)\boldsymbol{P}_i^w \qquad (2.8)$$

where

$$N_{i,p}^{(k)}(u) = p\left(\frac{N_{i,p-1}^{(k-1)}(u)}{u_{i+p} - u_i} - \frac{N_{i+1,p-1}^{(k-1)}(u)}{u_{i+p+1} - u_{i+1}}\right)$$

Then the first derivative of a NURBS curve can be written as

$$\begin{aligned} \boldsymbol{C}'(u) &= \frac{w(u)\boldsymbol{A}'(u) - w'(u)\boldsymbol{A}(u)}{w(u)^2} \\ &= \frac{w(u)\boldsymbol{A}'(u) - w'(u)w(u)\boldsymbol{C}(u)}{w(u)^2} = \frac{\boldsymbol{A}'(u) - w'(u)\boldsymbol{C}(u)}{w(u)} \end{aligned} \quad (2.9)$$

In a similar way the second derivative

$$\boldsymbol{C}''(u) = \frac{w(u)^2\boldsymbol{A}''(u) - 2w(u)w'(u)\boldsymbol{A}'(u) + (2w'(u)^2 - w(u)w''(u))\boldsymbol{A}(u)}{w^3(u)} \quad (2.10)$$

Calculating the derivatives of NURBS surfaces is done in a similar way to the calculation of the derivatives of NURBS curves. We only need to derive formulae for the derivatives of $S(u,v)$ in terms of the derivatives of $S^w(u,v)$ (a non-rational surface). Here the derivatives of non-rational surface are given by

$$\frac{\partial^{k+l}}{\partial^k u \partial^l v}\boldsymbol{S}^w(u,v) = \sum_{i=0}^{n}\sum_{j=0}^{m} N_{i,p}^{(k)} N_{j,q}^{(l)} \boldsymbol{P}_{i,j}^{\ w} \quad (2.11)$$

Then the first and second partial derivatives of a NURBS surface are given by

$$\boldsymbol{S}_u = \frac{w\boldsymbol{A}_u - w_u\boldsymbol{A}}{w^2} \quad (2.12)$$

$$\boldsymbol{S}_v = \frac{w\boldsymbol{A}_v - w_v\boldsymbol{A}}{w^2} \quad (2.13)$$

$$\boldsymbol{S}_{uu} = \frac{w^2\boldsymbol{A}_{uu} - 2ww_u\boldsymbol{A}_u + (2w_u^2 - ww_{uu})\boldsymbol{A}}{w^3} \quad (2.14)$$

$$\boldsymbol{S}_{vv} = \frac{w^2 \boldsymbol{A}_{vv} - 2ww_v \boldsymbol{A}_v + (2w_v^2 - ww_{vv})\boldsymbol{A}}{w^3} \tag{2.15}$$

$$\boldsymbol{S}_{uv} = \frac{w^2 \boldsymbol{A}_{uv} - ww_u \boldsymbol{A}_v - ww_v \boldsymbol{A}_u + (2w_u w_v - ww_{uv})\boldsymbol{A}}{w^3} \tag{2.16}$$

For simplicity the (u,v) parameters are omitted from (2.12) to (2.16).

The surface normal calculation involves the first derivatives in both u and v direction and a cross product. Therefore the normal of a NURBS surface is relatively expensive to compute. The normal gives the direction of displacement for control points in the metaball model, which will be discussed in detail in chapter 3.

2.3 Fundamental Algorithms

A number of fundamental geometric algorithms can be applied to NURBS curves and surface. They are knot insertion, curve or surface decomposition, degree elevation and knot removal.

2.3.1 Knot insertion

Inserting one or several knots into a defined knot can increase the flexibility of a NURBS curve or surface without changing its shape.

For a given NURBS curve $\boldsymbol{C}^w(u)$ defined over the knot vector $U = [u_0,...,u_m]$, a new knot $\overline{u} \in [u_k, u_{k+1})$ is inserted into $\boldsymbol{U}$ to form a new knot vector $\overline{U} = [u_0,....,u_k,\overline{u},u_{k+1},..,u_m]$. By setting up and solving the following system of linear equations:

$$\sum_{i=0}^{n} N_{i,p}(u)\boldsymbol{P}_i^w = \sum_{i=0}^{n+1} \overline{N}_{i,p}(u)\boldsymbol{Q}_i^w$$

where $\overline{N}_{i,p}(u)$ are the pth degree B-spline basis functions defined over $\overline{U}$, [21] develops the solution as:

$$\boldsymbol{Q}_i^w = (1-\alpha_i)\boldsymbol{P}_{i-1}^w + \alpha_i \boldsymbol{P}_i^w \qquad (2.17)$$

where

$$\alpha_i = \begin{cases} 1 & i \le k-p \\ \dfrac{\overline{u}-u_i}{u_{i+p}-u_i} & k-p+1 \le i \le k \\ 0 & i \ge k+1 \end{cases}$$

Only p control points of the control polygon ($n+1$ control points) are recalculated. The above knot insertion equation can be generalized for inserting a knot $\overline{u} \in [u_k, u_{k+1})$ with multiplicity r. Suppose $\overline{u}$ has initial multiplicity s and $s+p \le p$. Denote the ith new control point in the rth insertion step by $\boldsymbol{Q}_{i,r}^w$, with $\boldsymbol{Q}_{i,0}^w = \boldsymbol{P}_i^w$.

Then:

$$\boldsymbol{Q}_{i,r}^w = (1-\alpha_{i,r})\boldsymbol{Q}_{i-1,r-1}^w + \alpha_{i,r}\boldsymbol{Q}_{i,r-1}^w \qquad (2.18)$$

where

$$\alpha_{i,r} = \begin{cases} 1 & i \le k-p+r-1 \\ \dfrac{\overline{u}-u_i}{u_{i+p-r+1}-u_i} & k-p+r \le i \le k-s \\ 0 & i \ge k-s+1 \end{cases}$$

The knot insertion algorithm can be extended to surfaces by applying equation 2.17 or 2.18 to the control points in either the u or v parameter directions. It is often necessary to insert many knots at once; this is called *knot refinement*. The applications of knot insertion and knot refinement include:

- Increasing the flexibility of the NURBS curves or surfaces by adding more control points.
- Decomposing the NURBS curve or surface into a set of Bézier subcurves or Bézier patches – we elaborate this in the next section.

- Obtaining polygonal or polyhedral approximations to curves or surfaces. Refined knot vectors bring the control polygon or net closer to the curve or surface (figure 2.6).

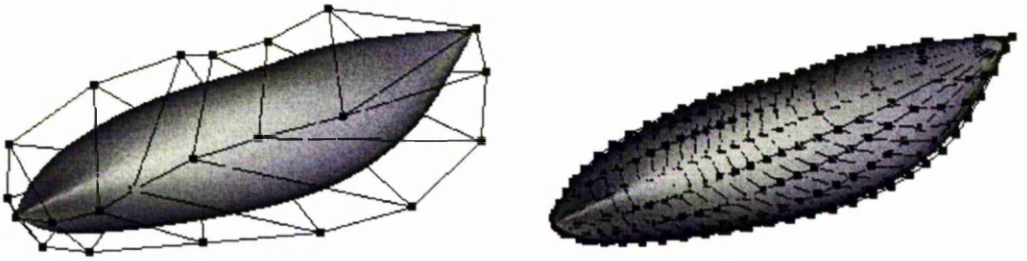


Figure 2.6: Knot insertion to obtain polyhedral approximation to the NURBS surface

2.3.2 Curve & surface decomposition

As discussed in the last section, in some applications a curve or surface needs to be subdivided into a number of Bézier curves or Bézier patches. First, we give the algorithm for subdividing a single curve or surface into two segments.

For a pth degree NURBS curve, inserting multiplicity p knots into the knot vector U will split the curve into two separate parts. For a given NURBS surface of degree p in the u direction and degree q in the v direction, inserting multiplicity p knots into $\boldsymbol{U}$ knot vector, or multiplicity q into the $\boldsymbol{V}$ knot vector will subdivide the surface into two patches. Figure 2.7 gives the example of subdividing a curve and figure 2.8 shows the subdivision of a surface.

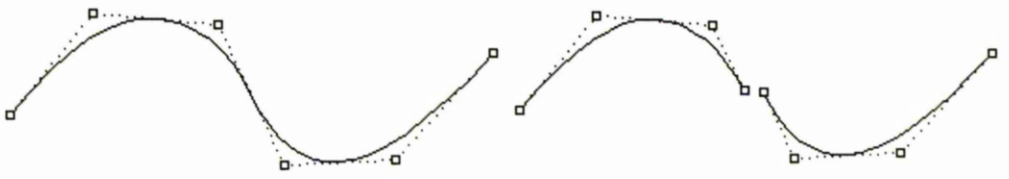


Figure 2.7: subdividing a NURBS curve

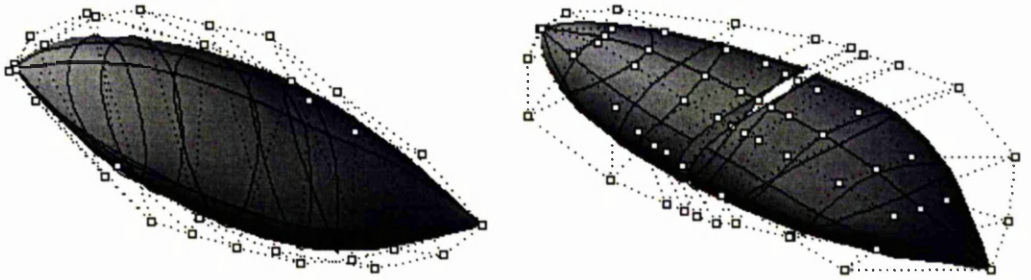


Figure 2.8: The subdivision of a NURBS surface

The decomposition of NURBS curves or surfaces is to subdivide them into their piecewise Bézier form. The algorithms are given by

- For a given NURBS curve, the rational Bézier subcurves are obtained by inserting each interior knot until it has multiplicity p.
- For a given NURBS surface, the rational Bézier patches are obtained by inserting each interior knot in $\boldsymbol{U}$ until it has multiplicity p and then inserting each interior knot in $\boldsymbol{V}$ until it has multiplicity q.
- The resulting piecewise Bézier form can be further converted to power basis form, which is often used for fast evaluation of curves or surface in computer graphics. Figures 2.9 and 2.10 shows the decomposition of a curve and a surface respectively.

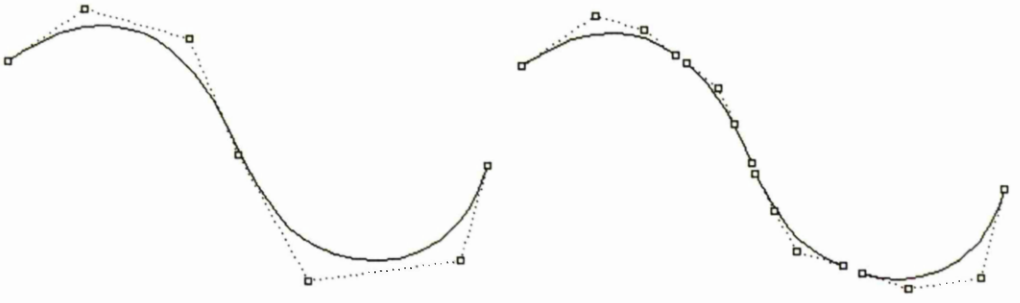


Figure 2.9: The decomposition of a NURBS curve

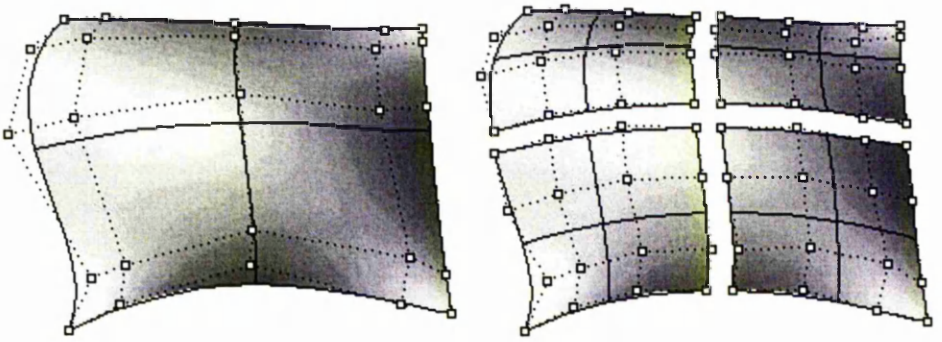


Figure 2.10: The decomposition of a NURBS surface

2.3.3 Degree Elevation

As another basic operation for NURBS, degree elevation increases the degree of a curve or surface whilst keeping the curve or surface unchanged. For a given *p*th degree NURBS curve defined in equation 2.4, it must be possible to elevate its degree from p to $p+1$, because it is a piecewise polynomial curve and rational degree elevation is based on non-rational degree elevation. Denote the elevated curve as

$$\boldsymbol{C}^w_{p+1}(u) = \sum_{i=0}^{\bar{n}} \bar{N}_{i,p+1}(u)\boldsymbol{Q}^w_i$$

$$= \sum_{i=0}^{n} N_{i,p}(u)\boldsymbol{P}^w_i$$

where $\bar{N}_{i,p+1}(u)$ are the $(p+1)$th degree B-spline basis functions defined on the new knot vector $\bar{U}$. Degree elevation of the NURBS curve refers to the algorithm for computing the unknown $\boldsymbol{Q}^w_i$, $\bar{U}$ and $\bar{n}$. Degree elevation is accomplished for a given NURBS surface by applying the curve degree elevation algorithm to the rows and columns of the control net. A good summary of the reference material concerning the degree elevation algorithm is given in [17]. Figure 2.11 shows an ellipsoid before and after degree elevation.

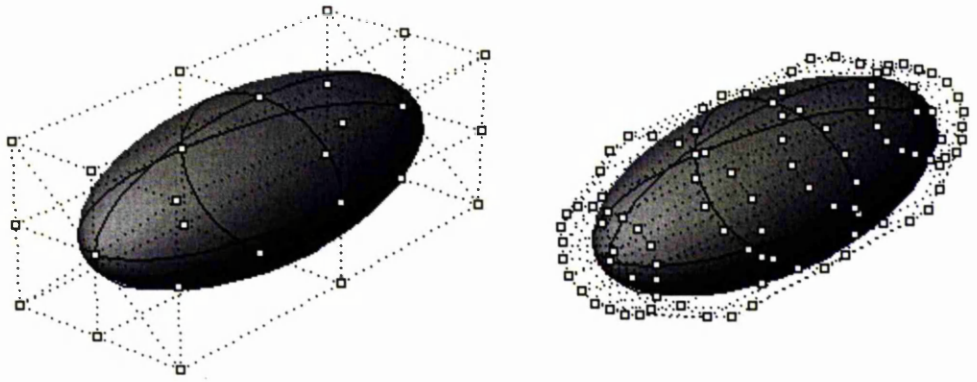


Figure 2.11: An ellipsoid before and after degree elevation

2.3.4 Knot Removal

Knot removal is the reverse process of knot insertion and it is an important utility in several applications. For example, when a NURBS curve or surface is interactively modified, knots are inserted into the knot vector to increase the flexibility of the curve or surface by adding new control points. After the modification, new control points may be removed. Knot removal may be involved to obtain the most compact representation of the curve or surface. Another application is to link several NURBS curves together. Knot removal is used to remove unnecessary knots in the knot vector of the new NURBS curve. After unnecessary (removable) knots are removed, the shape differences between the new curve and the old curve should be within the tolerance, which is specified by the user.

- **Knot-removal routines for curves**

For a given *p*th degree NURBS curve:

$$\boldsymbol{C}^w(u) = \sum_{i=0}^{n} N_{i,p}(u)\boldsymbol{P}_i^w$$

If u_r is an interior knot of multiplicity s in $\boldsymbol{U}$, we remove u_r t times ($1 \le t \le s$). Denote the new knot vector as U_t. We only can get the new curve as

$$\boldsymbol{C}^w(u) = \sum_{i=0}^{n-t} \overline{N}_{i,p}(u)\boldsymbol{Q}_i^w$$

if u_r is t times removable. The algorithm of knot removal must determine if knot is removable (how many times) and compute the new control points $\boldsymbol{Q}_i^w$. A comprehensive discussion about this algorithm can be found in [22].

- **Knot removal from a surface**

Let $\boldsymbol{S}^w(u,v) = \sum_{i=0}^{n}\sum_{j=0}^{m} N_{i,p}(u)N_{j,p}(v)\boldsymbol{P}_{i,j}^w$ be a NURBS surface. A u knot (v knot) is removed from $\boldsymbol{S}^w(u,v)$ by applying the knot removal algorithm to $m+1$ columns ($n+1$ rows) of control points.

Figure 2.12 shows an example of removing a knot from a NURBS ellipse.

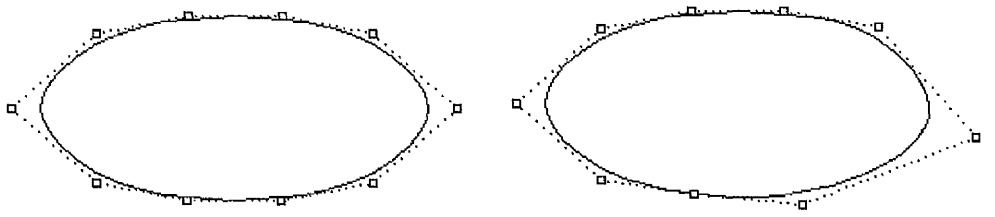


Figure 2.12: Removing a knot from a NURBS ellipse

2.4 Construction of Common Surfaces

There are a number of techniques for building a NURBS surface from curves. The most common NURBS surfaces are bilinear, extruded, ruled and revolved.

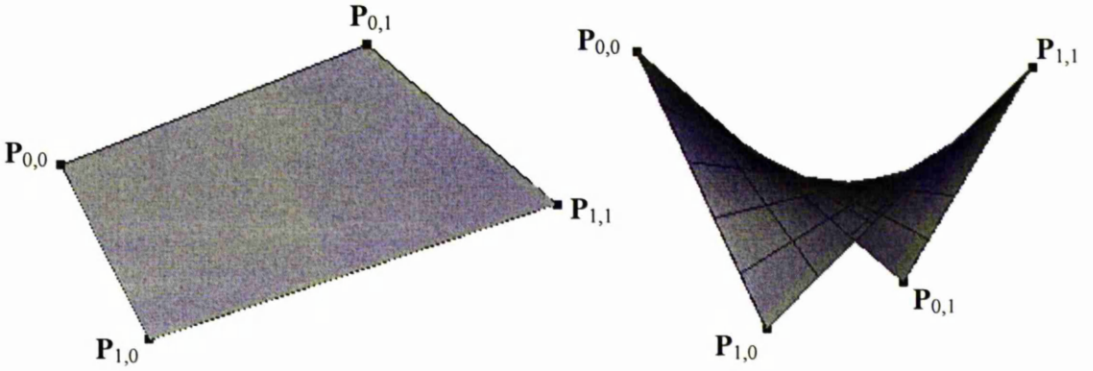
2.4.1 Bilinear Surfaces

Let four points $\mathbf{P}_{0,0}$, $\mathbf{P}_{1,0}$, $\mathbf{P}_{0,1}$ and $\mathbf{P}_{1,1}$ be defined in three-dimension space. Four line segments $\mathbf{P}_{0,0}\mathbf{P}_{1,0}$, $\mathbf{P}_{0,1}\mathbf{P}_{1,1}$, $\mathbf{P}_{0,0}\mathbf{P}_{0,1}$ and $\mathbf{P}_{1,0}\mathbf{P}_{1,1}$ can be formed. A bilinear NURBS surface (non-rational) can be created by a simple linear interpolation between the opposite boundary lines in both u and v directions:

$$\boldsymbol{S}(u,v) = \sum_{i=0}^{1}\sum_{j=0}^{1} N_{i,1}(u)N_{j,1}(v)\boldsymbol{P}_{i,j} \qquad (2.19)$$

with the knot vectors $\boldsymbol{U} = \boldsymbol{V} = \{0,0,1,1\}$.

If four points lie on a common plane, the bilinear NURBS surface represents the planar surface patch whose control points are the corner points of the planar patch (figure 2.13(a)).



(a) Planar line boundaries (b) non-planar line boundaries

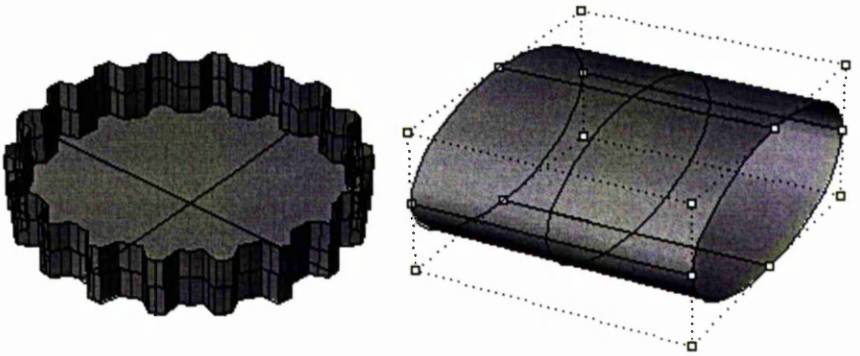
Figure 2.13: Bilinear Surfaces

2.4.2 Extruded Surfaces

Let $\mathbf{E}$ be a vector and $\boldsymbol{C}(u) = \sum_{i=0}^{n} R_{i,p}(u)\boldsymbol{P}_i$ be a pth degree NURBS defined on the knot vector $\boldsymbol{U}$, with weights w_i. Then the extruded surface can be obtained by sweeping $C(u)$ a distance of $\|E\|$ along $\boldsymbol{E}$. The extruded surface has the form

$$\boldsymbol{S}(u,v) = \sum_{i=0}^{n}\sum_{j=0}^{1} R_{(i,p)(j,1)}(u,v)\boldsymbol{P}_{i,j}^{w} \qquad (2.20)$$

Where the knot vector $\boldsymbol{U}$ is the knot vector of $\boldsymbol{C}(u)$ and $\boldsymbol{V} = \{0,0,1,1\}$. The control points are given by $\boldsymbol{P}_{i,0} = \boldsymbol{P}_i$, $\boldsymbol{P}_{i,1} = \boldsymbol{P}_i + E$ and $w_{i,0} = w_{i,1} = w_i$.



(a) Gear (b) Elliptic Cylinder

Figure 2.14: Extruded Surfaces

Figure 2.14(a) shows a gear, which is created by extruding a cross-section. Figure 2.14(b) shows a right elliptic cylinder by extruding a NURBS ellipse normal to the plane of the ellipse.

Another kind of extruded surface is created by extruding the profile curve $C(u)$ along the path curve. Figure 2.15 shows a thread of fabric by extruding a cross-section of ellipse along a path curve.

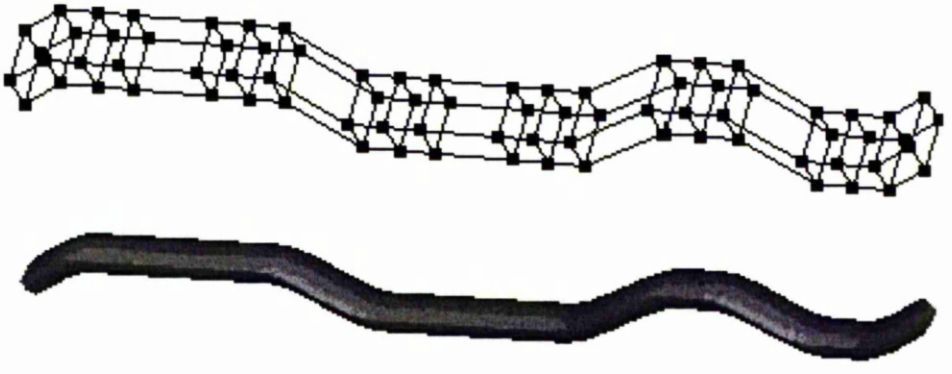


Figure 2.15: Extruding an ellipse along a path curve

2.4.3 Ruled Surfaces

Assume we have two boundary NURBS curves

$$\boldsymbol{C}_k(u) = \sum_{i=0}^{n_k} R_{i,p_k}(u)\boldsymbol{P}_i^k \qquad k = 1,2$$

defined on the knot vectors $\boldsymbol{U_0}$ and $\boldsymbol{U_1}$ respectively.

Because of the tensor product nature of the surface, the two boundary curves $\boldsymbol{C}_k(u)$ must have the same degree and be defined on the same knot vector. The process to make the two curves share the same degree and same knot vector is called the compatibility-proceed [23]. A ruled surface $\boldsymbol{S}(u,v)$ is created by linear interpolation between $\boldsymbol{C}_0(u)$ and $\boldsymbol{C}_1(u)$ in v direction. The desired surface form is

$$\boldsymbol{S}(u,v) = \sum_{i=0}^{n}\sum_{j=0}^{1} R_{(i,p)(j,1)}(u,v)\boldsymbol{P}_{i,j}^{w}$$

where $\boldsymbol{V} = \{0,0,1,1\}$, $\boldsymbol{U} = \boldsymbol{U}_1 \cup \boldsymbol{U}_2$, $p = \max\{p_0, p_1\}$, $\boldsymbol{P}_{i,0}$ and $w_{i,0}$ are taken from the compatibility-processed $\boldsymbol{C}_0(u)$ and, $\boldsymbol{P}_{i,1}$ and $w_{i,1}$ are taken from the compatibility-processed $\boldsymbol{C}_1(u)$.

Figure 2.16 shows a ruled surface constructed from two edge curves.

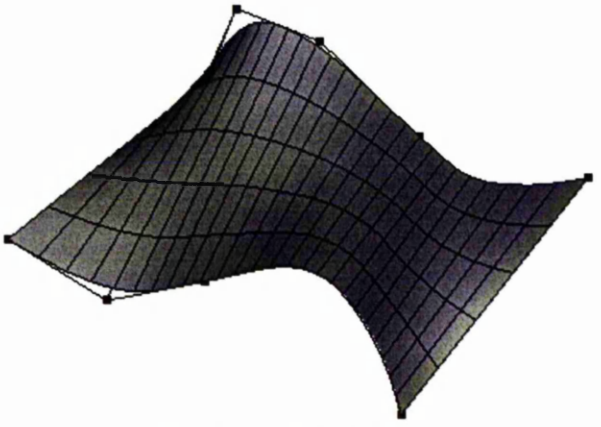


Figure 2.16: A Ruled Surface

2.4.4 Revolved Surface

Considering that a NURBS profile curve (figure 2.17 (a)) lying on the *xz*-plane has the form

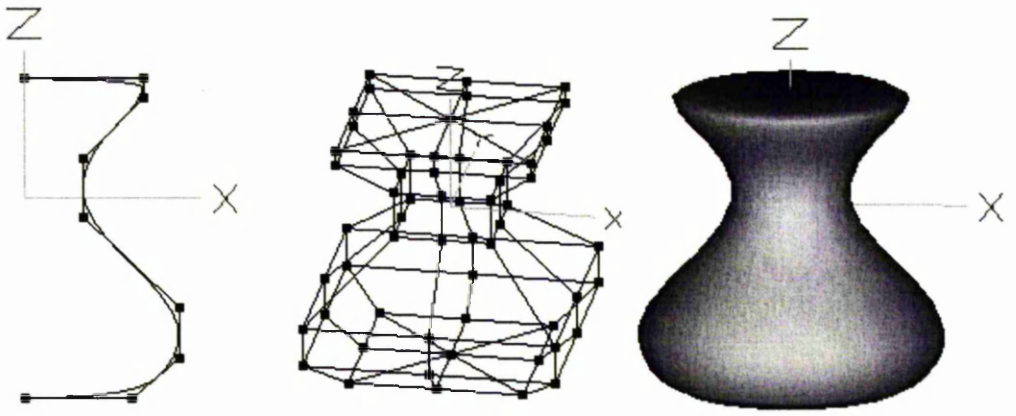
$$\boldsymbol{C}(v) = \sum_{j=0}^{m} R_{j,q}(v)\boldsymbol{P}_j$$

which is defined on the knot vector $\boldsymbol{V}$, then the full revolution surface can be created by revolving the profile curve a full 360° about *z*-axis. Let us use the nine-point circle representation, with $\boldsymbol{U} = \{0,0,0,\frac{1}{4},\frac{1}{4},\frac{1}{2},\frac{1}{2},\frac{3}{4},\frac{3}{4},1,1,1\}$ and weights $w_i = \{1,\frac{\sqrt{2}}{2},1,\frac{\sqrt{2}}{2},1,\frac{\sqrt{2}}{2},1,\frac{\sqrt{2}}{2},1\}$. Then the required surface has the form

$$\boldsymbol{S}(u,v) = \sum_{i=0}^{8}\sum_{j=0}^{m} R_{(i,2)(j,q)}(u,v)\boldsymbol{P}_{i,j}^{w} \qquad (2.21)$$

where

- The knot vector $\boldsymbol{U} = \{0,0,0,\frac{1}{4},\frac{1}{4},\frac{1}{2},\frac{1}{2},\frac{3}{4},\frac{3}{4},1,1,1\}$
- The knot vector $\boldsymbol{V}$ is same as the profile curve.
- For $i = 0$, $\boldsymbol{P}_{i,j} = \boldsymbol{P}_{0,j} = \boldsymbol{P}_j$.
- For fixed j, $\boldsymbol{P}_{i,j}(i = 0,....,8)$ lie on the plane $z = z_j$ forming a nine-point circle square control polygon of width $2x_i$ with centre on the *z*-axis (figure2.17 (b)).
- $w_{i,j} = \{w_j, \frac{\sqrt{2}w_j}{2}, w_j, \frac{\sqrt{2}w_j}{2}, w_j, \frac{\sqrt{2}w_j}{2}, w_j, \frac{\sqrt{2}w_j}{2}, w_j\}$.



(a) Profile curve (b) Control net (c) Rendered surface

Figure 2.17: Revolved Surfaces

2.5 NURBS Solid Model

Arbitrary solid objects can be represented within the computer by several categories of data structures, such as the boundary representation (B-rep), spatial decomposition and constructive solid geometry (CSG). However, B-rep and CSG are exact representations, while the spatial decomposition method is an approximate model description.

In CSG, solids are described as combinations of simple solids (primitives) in a series of Boolean operations. An illustration is given in figure 2.18. The advantages of CSG are its compactness and ability to record Boolean operations and changes of transformation quickly, including undo operations. CSG models can be converted to other representations but it is difficult to convert arbitrary models back to CSG. However, CSG models cannot represent objects with complex surfaces such as the wing of airplane or the body of ship.

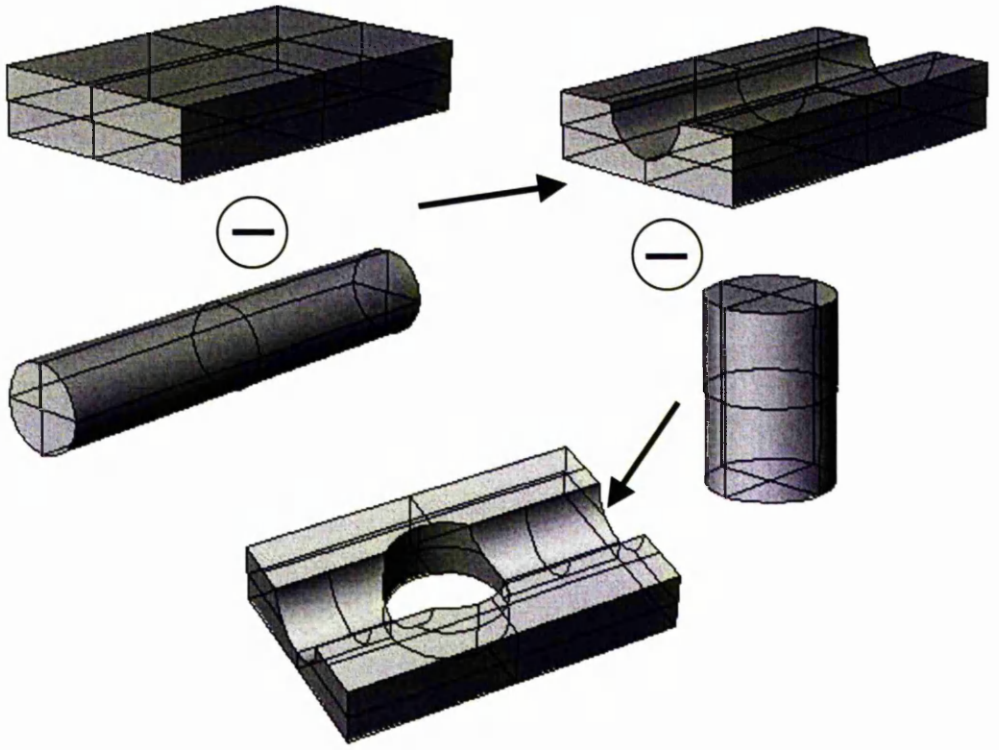


Figure 2.18: A CSG model through two subtraction operations

On the other hand, B-rep models represent a solid indirectly by a representation of its bounding surfaces. A B-rep solid is represented as a volume contained in a set of surfaces together with the topological information which defines the relationships between surfaces. Because B-rep includes such topological information, a solid is represented as a closed space in three-dimension space. Therefore, B-rep can represent a wide class of objects but the data structure is complex, and it requires a large memory space.

In this thesis, we deform only the B-rep model where the deformation takes place in one of the bounding surfaces. All bounding surfaces are converted into the form of trimmed NURBS surfaces. Therefore we call this solid model the NURBS solid model.

2.5.1 Trimmed NURBS Surfaces

Trimmed surfaces have played a fundamental role in Computer Aided Design and computer graphics for many years [9][11]. Most complex geometrical objects are generated from some sort of trimming process such as fillet, blend and chamfer operations. Trimmed surfaces are also the result of a Boolean operation on the solid objects, which is bounded by a set of trimmed NURBS surfaces.

- **The definition of trimmed NURBS surfaces**

A trimmed NURBS surface is a NURBS surface defined by (2.5) and several trimming curves. The trimming curves are normally in NURBS form so that there will be a uniform data structure to describe the whole trimmed surface. Assume that N such curves are defined as

$$\begin{aligned} \boldsymbol{C}_k(t) = (u_k(t), v_k(t)) = \sum_{i=0}^{n} \boldsymbol{P}_i^k N_{i,j}(t) \\ k = 1,2,\ldots, N \end{aligned} \qquad (2.22)$$

These curves form a set of trimming loops: one outer loop and several inner loops. The outer loop corresponds to the outer boundary of the trimming region. The inner loops actually indicate holes in the surface. As shown in figure 2.19, in parametric space (UV space) the solid-line loop is the outer trimming loop and two dash-line ones are the inner trimming loops.

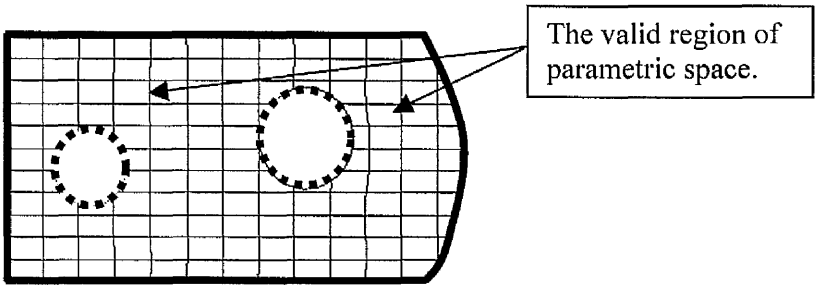


Figure 2.19: The trimming loops of the trimmed NURBS surface

2.5.2 Adaptive Tessellation

Both in Computer Aided Design and Computer Graphics, trimmed surfaces are tessellated into a set of triangles or quadrilaterals for rendering, visualization, area computation and rapid prototyping. There are several tessellating methods which can be classified into two simple categories [24]:

- **Uniform subdivision**. This is the simplest case and involves a user specifying a level at which uniform subdivision of all patches is to terminate.
- **Non-uniform subdivision**. This means stopping the division when the subdivision products meet a patch flatness criterion.

The second category is theoretically preferable as it generates fewer polygons than the first one. More subdivision takes place in the areas of high surface curvature. The methods in second category are also called as *adaptive tessellation*. An example of this is shown in figure 2.20.

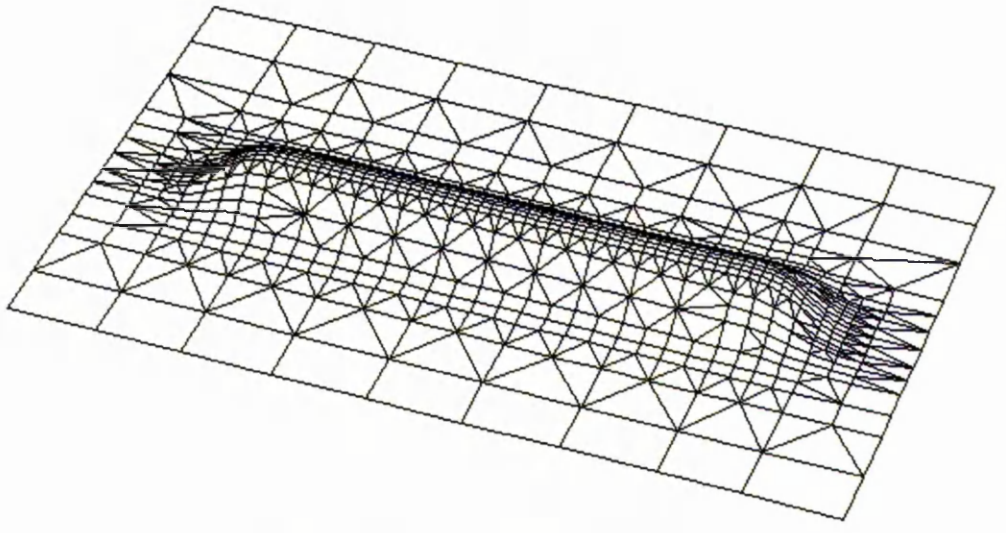


Figure 2.20: Adaptive Tessellation

2.5.3 Curve and Surface Intersection

In the process of creating a solid model, it is often necessary to find the intersection between two curves or surfaces.

- **NURBS Curve Intersection**

A comparison of three Bézier curve intersection algorithms is given by Sederberg [25]. We extend these methods to calculate the intersection of NURBS curves both in 2D and 3D. An algorithm for 2D curve intersection is given as follows (An illustration is given in figure 2.21.):

Algorithm 1. Curve_Intersection (C1, C2)
Input: *The two curves* ***C1*** *and* ***C2***
Output: *The list of intersection points* ***Pts***
begin
find the two bounding rectangles ***RC1*** *and* ***RC2*** *for these two curves*
if *No intersection of* ***RC1*** *and* ***RC2*** ***return*** *No intersection points*
else
if *two curves are flat enough*
then
find the intersection point of two straight lines {approximate two curves}
return *the intersection point*
else *{ two curves are not flat enough}*
subdivide ***C1*** *into two subcurves* ***NC1*** *and* ***SC1.***
subdivide ***C2*** *into two subcurves* ***NC2*** *and* ***SC2.***
Curve_Intersection *(NC1, NC2);*
Curve_Intersection *(SC1, SC2);*
end if
end if
end *of Algorithm 1.*

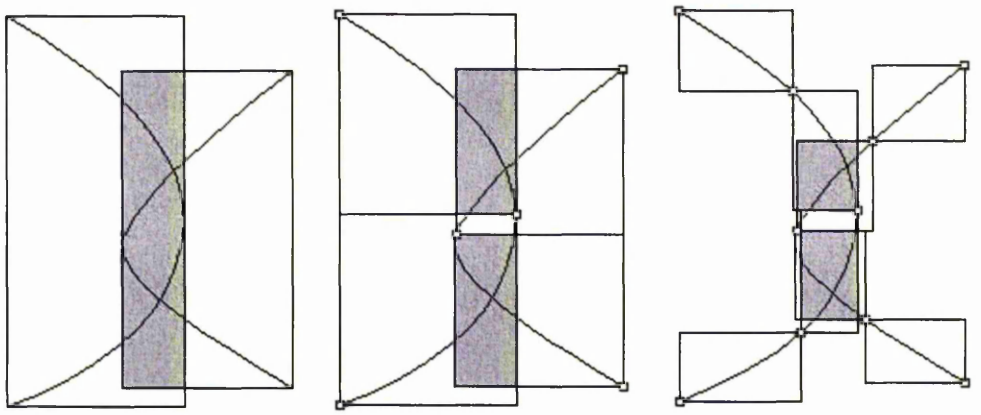


Figure 2.21: 2D Bounding rectangles overlap for curve intersection.

- **NURBS Surface Intersection**

The intersection between two NURBS surfaces is much more difficult to solve than the intersection of two NURBS curves. The result of intersection may be a point, curve or plane. The approach to surface intersection can be classified into four main categories [26]:

- **Analytic**
- **Lattice Evaluation**
- **Marching**
- **Subdivision**

We only discuss the method of subdivision as it gives a robust solution for the majority of cases and it is also the easiest one to implement. In a similar way to the 3D curve case, the overlapping test of bounding boxes is carried out to find out the possibility of intersection. If this is so, the surfaces are subdivided and the test is repeated. This subdivision continues until the surface patch is flat enough, which can be approximated by a plane patch. Then the intersection

between two surfaces can be approximated by calculating the intersection of a set of plane patches.

2.5.4 B-rep Model

Many of the current solid modelling systems are based on Boundary Representation (B-rep). B-rep overcomes the disadvantages of CSG model in that it also can represent sculptured solids, whose boundaries are represented by trimmed rational parametric surfaces. This is a wide family of objects that can represent exactly quadrics, tori and free-form solids [27]. In this section, we give a brief introduction to B-rep. Comprehensive reference materials can be found in [27][28][30][31][32].

The B-rep representation of a solid lends itself to a description in terms of *faces*, *edges* and *vertices*. Each *face* is a trimmed parametric surface patch, which defines the solid boundary. Each of the trimming curves form an *edge*, and is formed by an intersection of two surfaces (usually faces). Finally, endpoints of edges form the vertices. They can be represented as an intersection of three surfaces. Figure 2.25 shows an example solid and the face connectivity structure that we maintain. Each graph vertex represents a patch, with graph edges expressing the adjacency information (i.e., which patches are next to each other). We also maintain the two faces that are adjacent to each edge, and an anticlockwise order of faces around each vertex. We convert all trimmed parametric surface patches and trimming curve into the form of trimmed NURBS surfaces and NURBS curves respectively, so that all data except vertices have the form of NURBS. We call this a NURBS B-rep model.

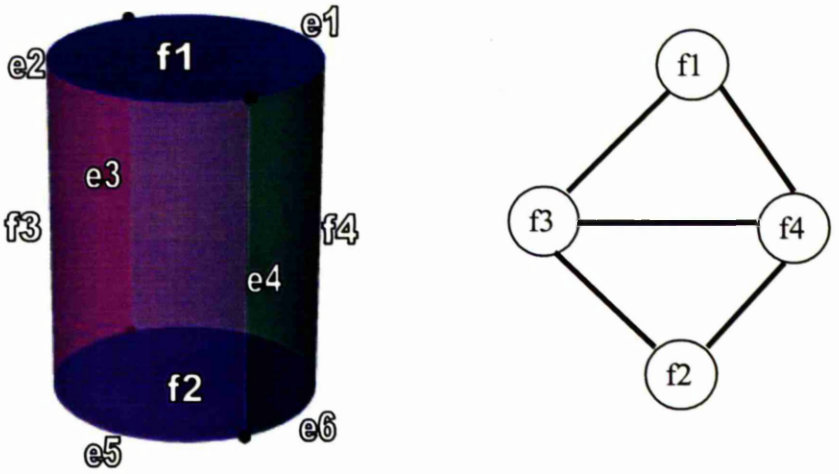


Figure 2.25: A cylinder and its face connectivity structure

2.6 Summary

This chapter has discussed some fundamentals of NURBS and operations on NURBS including NURBS solid B-rep model, which will be necessary to describe the deformation on untrimmed NURBS surface, trimmed NURBS surface and NURBS solid model in rest chapters. The properties and flexibility of NURBS, along with their ability to represent free-form curves and surfaces, handle discontinuities, and accurately model conic sections, make them a good candidate for use in a curve/surface representation schemes. Trimmed NURBS surfaces, as the surface boundary of NURBS B-rep model, play a very important role in CAD and Computer Graphics. The tessellation of trimmed NURBS surfaces will be discussed in chapter five.

Chapter 3

Deformation Model

3.1 Introduction

The generation of complex surfaces is a problem that has been addressed by many researchers and commercial systems [76]. Although much progress has been made in geometric modelling, creating complex free-form surfaces is still very difficult and tedious. One major reason is that the users often have to construct complex models starting with most fundamental elements, such as points, curves or simple primitives. Great effort has been made on improving the modelling efficiency in the last two decades [76]. In the case of NURBS-based modelling, although effective shape editing tools are now available, state-of-the-art technology still does not allow the user intuitive control over the smooth blending of the complex models [34].

In this chapter, we will discuss several deformation models for the NURBS-based modelling system, and we present our approach of generalized metaball modelling for NURBS.

3.2 Geometric Deformation

Three major geometric deformation methods: direction control point manipulation [35][36], Free-form deformation (FFD) [38][40] and its extension (EFFD) [33] have been developed in the last twenty years. All of them can be applied in the NURBS-based modelling system.

3.2.1 Direct Control Point Manipulation

Moving the control point is a fundamental way of modifying the shape of NURBS surface and it is used in every NURBS-based modelling system. A surface editor in such a modelling system provides a method for selecting a single or a group of control points and manipulating them in three dimensions. According to the NURBS property of local shape control, this manipulation will affect only the area around these control points (figure 3.1).

However, this can often be a clumsy and tedious method for surface design, especially for complex surfaces with hundreds of control points. It has several disadvantages:

- The number of control points the user will have to move depends on the size of the deformation region. For example, the design of a large bump may require moving many control points whereas designing small bumps may be impossible.
- The shape of the deformed region (both along its boundary and within its interior) is imposed by the shape of surface isoparametric lines, this is, by the position of neighbouring control points. Designing a bump with a circular boundary, for example, is almost impossible.

- The position of the deformed region on the surface is imposed by the position of the control points since only the control points are moved.

Some of these problems can be partially solved by using refinement techniques (knot refinement). In [35] and [36] Piegl also gave a method of combining control point-based and weight-based modifications. The weight-based technique is a nice solution for the size problem.

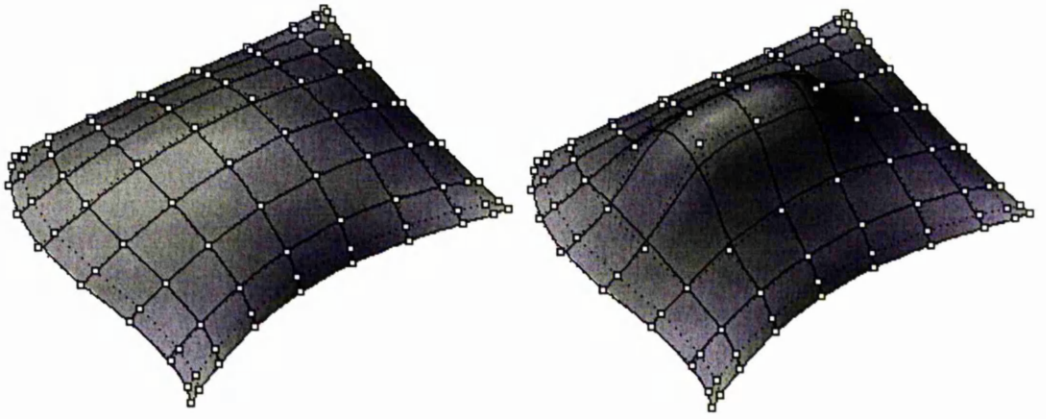


Figure 3.1: Direct control point manipulation

3.2.2 Free-Form Deformations

Another widely used deformation method for surface modelling is free-form deformation (FFD) [38]. The basic concept of FFD is that an object to be deformed is imagined as flexible and embedded in a pliable solid. Deformations applied to the surrounding solid directly affect the embedded geometry. In [37], Barr first presented a set of powerful transformations for a solid object that is the origins of FFD. The transformations include stretching, bending, twisting, and tapering operations. They were applied in a hierarchical manner as a set of multiplied matrices either in global space or the local space of the object.

Sederberg and Parry [38] present a deformation tool in which the representation of the surface is hidden by a FFD lattice embedding the object. The deformations of the FFD lattice are automatically passed to the object. FFD has proved to be an intuitive and efficient modelling technique appreciated by designers [39]. Figure 3.2 has a sample 3×3 FFD of a teapot, which shows how a complicated surface can be transformed with the movement of only a few mesh points.

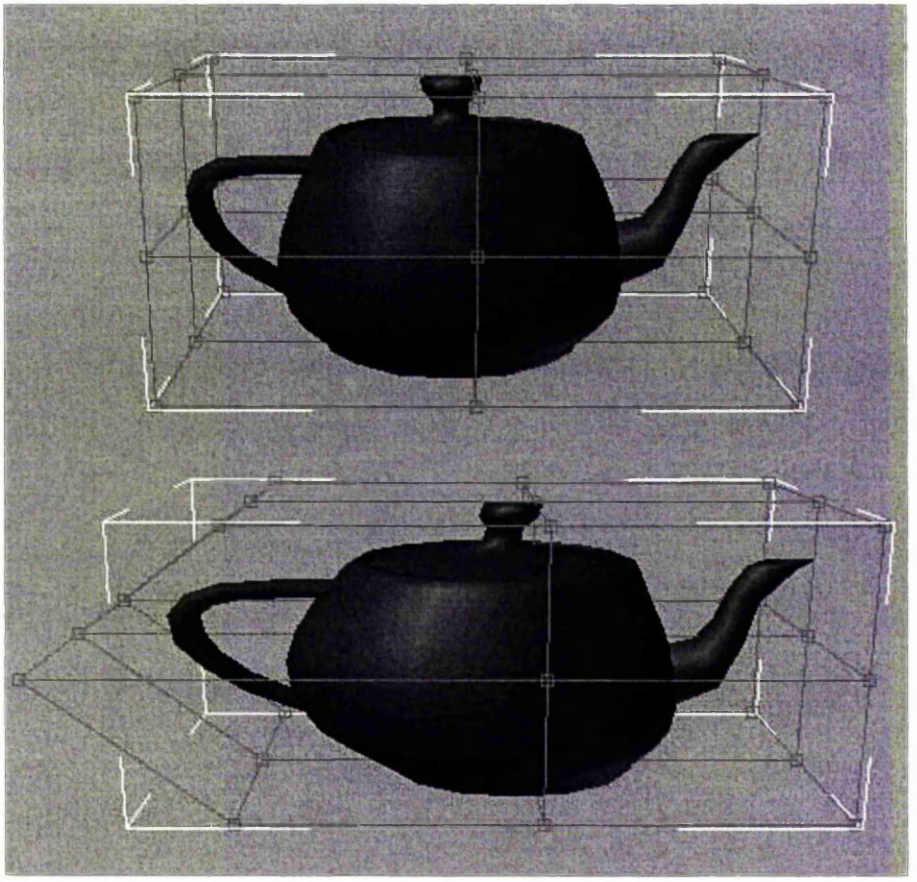


Figure 3.2: A simple 3×3 FFD transformation (Made in 3DS Max)

Here we give a brief introduction of the theory of FFD, more can be found in [37][38][40]. Free-Form Deformation consists of embedding the geometric model or the region of the model that has to be deformed into a parallelepipedical 3D lattice regularly subdivided, as shown in figure 3.3. The

deformations of the FFD lattice are then automatically passed to the model. Let l, m and n be the number of subdivisions along each of the three directions, U, V and W. These numbers can be chosen by the user depending on the deformation he wants to produce (in figure 3.3, $l=2$, $m=1$ and $n=2$).

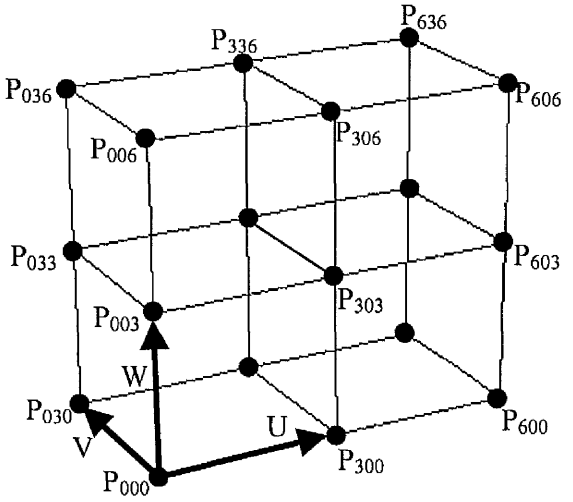


Figure 3.3: A parallelepipedical lattice

The 3D lattice is represented by a tensor product piecewise Bézier volume. This volume is defined by an array of $(3l+1)\times(3m+1)\times(3n+1)$ control points $\boldsymbol{P}_{ijk}$. Each subdivision element, also named "Chunk" by Clark in [41], is defined by:

$$\boldsymbol{L}(u,v,w) = \sum_{i,j,k=0}^{3} B_i(u)B_j(v)B_k(w)\boldsymbol{P}_{i,j,k} \qquad 0 \le u,v,w \le 1 \qquad (4.1)$$

where the $B_i(t)$ are the degree 3 Bernstein polynomials, the $\boldsymbol{P}_{ijk}$ are the chunk control points.

The Free-Form deformation technique is divided into two steps:

- Before deforming the 3D lattice, the coordinates u_s, v_s and w_s, in the lattice parametric space, of each object point are computed. With

parallelepipedical lattices, this step requires only the resolution of three linear equations. For any point X interior to the lattice, $0 < u_s < l$, $0 < v_s < m$ and $0 < w_s < n$.

- After deforming the 3D lattice, the deformed positions of the object points are computed. The deformed position X_{ffd} of an arbitrarily point X with coordinates (u_s, v_s, w_s) in the lattice parameter space is computed in two steps. First, determine the chunk where the point lies by computing the floor values (u_0, v_0, w_0) of (u_s, v_s, w_s). Let $u = u_s - u_0$, $v = v_s - v_0$ and $w = w_s - w_0$ be X coordinates in the chunk parameter space. The second step consists of generating the Cartesian coordinates of X_{ffd} from *u, v, w* and the matrix of 4×4×4 control points $\boldsymbol{P}_{ijk}$ of the chunk, according to (4.1).

The deformation is specified by moving the $(l+1)\times(m+1)\times(n+1)$ control points corresponding to the corner points of the volume elements (or chunks). Only these points are represented on figure 3.3. The tangents at the corner control points can also be modified by the user. The other control points are automatically updated. Two modes exist for the manipulation of corner control points. Constant tangent mode, where the tangents of the point remain constant when the point is moved, and non-constant tangent mode where the tangents of the point are updated according to the position of the neighbour points simulating a C-Spline interaction [41]. These two modes can be chosen independently for each of the three directions.

3.2.3 Extended Free-Form Deformations

Although FFD is a very intuitive method for surface modelling, it is still restricted by the shape of lattice when it applies to the complex sculptured

surfaces. Extended Free-Form Deformations (EFFD) allow the user to modify the complex surface using the non-parallelepipedical lattices. EFFD lattices are equivalent to FFD lattices; only the initial lattice shape is different. The EFFD technique can be described in four steps (notice that the EFFD lattice is defined independently of the surface to which it will be applied.):

1. Editing an EFFD lattice.

An EFFD lattice is defined either from a predefined three-dimensional lattice or from two-dimensional lattices. A very useful non-parallelepipedical lattice is the cylindrical lattice, which is obtained by welding two opposite faces of a parallelepipedical lattice and by merging all the points of the cylinder axis (figure 3.4). EFFD lattices can also be created from two-dimensional lattices in the same way as surfaces are defined from curves (loft, sweep, extrusion...). Traditional modelling methods are employed to define them.

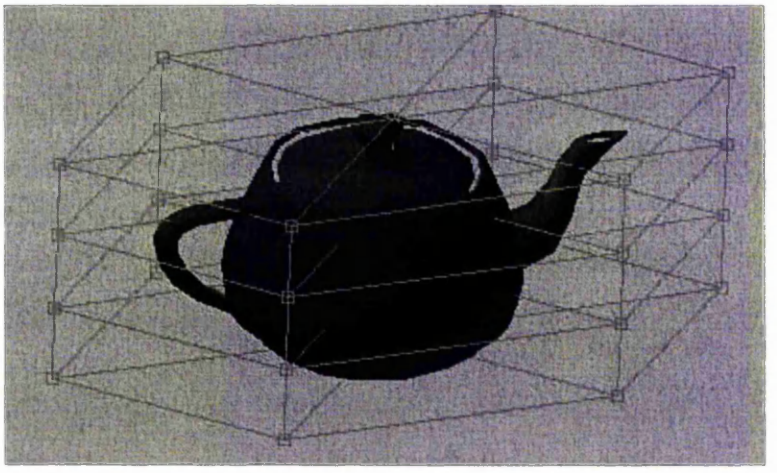


Figure 3.4: Cylindrical lattice

2. Associating an EFFD lattice with the surface.

The next step consists in taking an EFFD lattice out of the library and associating it with the desired surface. While an EFFD lattice is associated with a surface, one can still edit it without deforming the surface. At this

time, an attractive capability is to allow the user to move the lattice to a user specified point on the surface.

3. "Freezing" an EFFD lattice.

Now, we deform the surface. Assuming that several lattices are associated with the surface, the user must first select one of the EFFD lattices and "freeze" it. Freezing a lattice consists of computing the (u_s, v_s, w_s) coordinates of each point of the surface in the EFFD lattice parameter space. For each surface only one EFFD lattice can be frozen at a time. With arbitrarily shaped lattices, finding the (u_s, v_s, w_s) coordinates of the surface points is decomposed into two steps. First, the chunk where the point is supposed to lie is determined by using the convex hull property of Bézier volumes. The (u, v, w) coordinates inside the chunk are then computed using Newton approximation.

4. Deforming the surface.

When an EFFD lattice is frozen, all the transformations applied by the user to the lattice are passed to the surface. Only moving transformations are valid for frozen lattices. The computation of the X_{ffd} coordinate points of the deformed surface is equivalent to the FFD one.

3.3 Metaball Deformation Method

3.3.1 Background

Although FFD-based methods can achieve a variety of deformations, the user is forced to define some control points in the space to be deformed and then move these control points. This indirect interface may be unnatural for some applications. Hsu W. and Hughes J. [42] addressed this problem and proposed a direct interface that involves solving a complex equation system, but its computational cost is high. Borrel and Bechmann [43] developed a general

deformation model in which the deformation is defined by some user-specified point displacement constraints. The desired deformation is obtained by selecting a solution obeying the constraints. Nevertheless, the shape of the resulting deformation in this method is not strongly correlated to the constraints except that the constraints are satisfied. To overcome this, Borrel and Rappoport [44] introduced a local deformation method which they term simple constrained deformation (Scodef). In Scodef, the user defines some constraint points, each of which is associated with a user-defined displacement and an effective radius. The displacement of any point to be deformed is the blend of the local B-Spline basis functions determined by these constraint points. Note that the deformation achieved by Scodef is both local and intuitive and the constrained points can be directly located on the boundary surface of the object to be deformed. To extend the flexibility of the local deformation, however, deformation models based on line, surface and volume constraints are desired. Borrel and Rappoport point out that their model could not be generalized to deal with these kinds of constraints.

Motivated by the concept of a metaball, Xiangang Jin, Youfu Li and Qunsheng Peng [8] present a new constrained deformation model based on the special potential function distribution of generalized metaballs. In this method, constraints are generalized to include point, line, surface and volume constraints. The user need only define a set of constraints with desired displacements and an effective radius associated with each constraint. A generalized metaball is then set up at each constraint with a local potential function centered at the constraint falling to zero for points beyond the effective radius. The displacement of any point within the metaball is a blend of these generalized metaballs. This deformation model produces a local deformation and is independent of representation of the underlying objects to be deformed. The constraints generate some "bump" shapes over the space based on the type of constraint and its associated potential function, and

influence the final shape of the deformed object directly. The location and height of a bump are defined by a constraint's effective radius.

In this thesis, we are going to apply this method to an untrimmed NURBS surface, then to a trimmed NURBS surface, finally to a NURBS solid model.

3.3.2 The definition of metaball model

Metaball modelling has been recognized as a flexible technique for implicit surface modelling. It is very convenient for designing closed surfaces and provides simple solutions for creating blends, ramifications and advanced human character design [44][45][46][47][48]. A good introduction to metaball modelling and implicit surfaces can be found in [49]. The generalized metaballs are defined as an isosurface of a scalar field which is generated from some field generating points. The field value at any point is determined by the distance to generating point calculated through the potential function. The constraints for generalized metaballs include lines, surfaces and volumes [48][49], which are termed skeletons.

The skeleton-based model provides an intuitive way to define the desired shapes with implicit surfaces. Let $\boldsymbol{L}$ be the skeleton, $P(x,y,z)$ be a point in 3D space, $r(\boldsymbol{P},\boldsymbol{L})$ be the minimal distance from $P(x,y,z)$ to the individual point $Q(u,v,w)$ on the skeleton $\boldsymbol{L}$. Then,

$$r(\boldsymbol{P},\boldsymbol{L}) = \|\boldsymbol{P}-\boldsymbol{Q}\| \quad \boldsymbol{Q} \in \boldsymbol{L} \qquad (3.1)$$

Figure 3.5 give an illustration of the metaball model for the disk constraint.

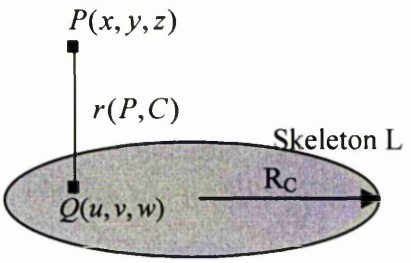


Figure 3.5: An illustration of the metaball model

Then the potential function associated with skeleton $\boldsymbol{L}$ can be defined as the composition of a potential function $f(r,R)$ and a distance function $r(\boldsymbol{P},\boldsymbol{L})$

$$F(r(\boldsymbol{P},\boldsymbol{L}),R) = f(r,R) \circ r(\boldsymbol{P},\boldsymbol{L}) \qquad (3.2)$$

where R is a specified distance called the effective radius. Euclidean space is often adopted as the distance space for calculating $r(\boldsymbol{P},\boldsymbol{L})$ and

$$r(\boldsymbol{P},\boldsymbol{L}) = \sqrt{(x-u)^2 + (y-v)^2 + (z-w)^2} \qquad \boldsymbol{Q} \in \boldsymbol{L} \qquad (3.3)$$

There are several potential functions which can be used for generalized metaballs: Blinn's exponential function [50], Nishimura's piecewise quadric polynomial [50], Murakami's degree four polynomial [50] and Wyvill's degree six polynomial [50]. Wyvill's degree six polynomial is the better one as it blends well and can avoid the calculation of a square root.

$$f(r,R) = \begin{cases} -\frac{4}{9}\left(\frac{r}{R}\right)^6 + \frac{17}{9}\left(\frac{r}{R}\right)^4 - \frac{22}{9}\left(\frac{r}{R}\right)^2 + 1 & 0 \le r \le R \\ 0 & r > R \end{cases} \qquad (3.4)$$

The local space deformation of metaball modelling can be achieved by interactively specifying the constraints and their effective radii. The constraints can either be points, lines, surfaces or volumes.

Let $\boldsymbol{L}$ be a constraint skeleton, R be the effective radius, and $\boldsymbol{S}$ be the corresponding distance surface.

$$\boldsymbol{S} = \{P(x,y,z) \in \boldsymbol{S} \mid r(\boldsymbol{P},\boldsymbol{L}) = R\} \qquad (3.5)$$

We define the tuple $M = \langle \boldsymbol{S}, f(r,R) \rangle$ as a generalized metaball based on the skeleton $\boldsymbol{L}$.

A generalized constrained deformation model based on generalized metaballs can then be defined. Let $\boldsymbol{P} = (x,y,z)$ be a point in 3D space, *Deform(**P**)* be a deformation function which maps $\boldsymbol{P}$ to *Deform(**P**)*. Let $\boldsymbol{L}_i$ be a constraint which consists of points, lines, surfaces and volumes, ΔD_i be its displacement, R_i be the effective radius of $\boldsymbol{L}_i$. Then the deformation function affected by constraint $\boldsymbol{L}_i$ is defined as

$$Deform(\boldsymbol{P}) = \boldsymbol{P} + \Delta D_i F(r(\boldsymbol{P},\boldsymbol{L}_i), R_i) \qquad (3.6)$$

The deformation model (3.6) has some useful properties.

$$\begin{cases} Deform(\boldsymbol{P}) = \boldsymbol{P} + \Delta D_i F(0, R_i) = \boldsymbol{P} + \Delta D_i & \forall \boldsymbol{P} \in \boldsymbol{L}_i \\ Deform(\boldsymbol{P}) = \boldsymbol{P} + \Delta D_i F(R_i, R_i) = \boldsymbol{P} \end{cases}$$

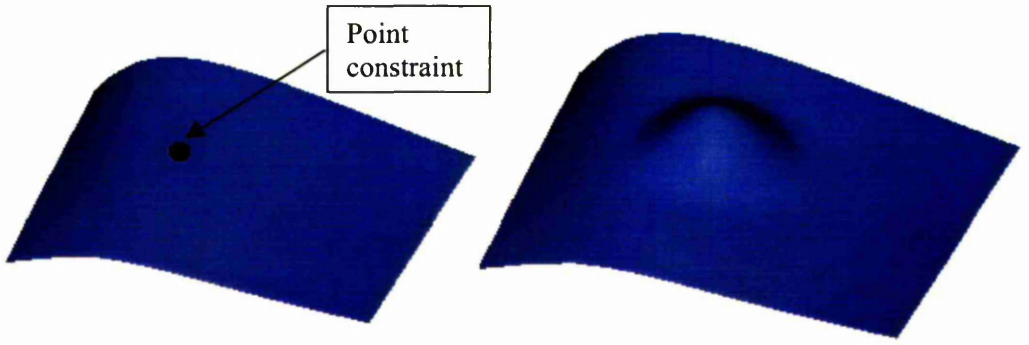
Therefore, if the distance from $\boldsymbol{P}$ to constraint $\boldsymbol{L}_i$ is larger than R, the value of distance function is equal to zero. The deformation function yields a local deformation, which generates a deformation precisely within the effective radius of the user-specified constraint.

The above deformation model can be extended to deal with multiple constraints. The deformation functions for n constraints is defined as

$$Deform(\boldsymbol{P}) = \boldsymbol{P} + \sum_{i=1}^{n} \Delta D_i F(r(\boldsymbol{P},\boldsymbol{L}_i), R_i) \qquad (3.7)$$

3.4 General Constraints

Constraints for a generalized metaball can be a point, a line segment, a piece of surface, or even a volume. We give the computation methods for some typical cases.



(a) Original Surface (b) Deformed Surface

Figure 3.6: Point constraint deformation

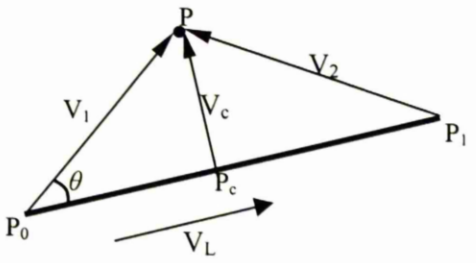
3.4.1 Point constraint

Let $\boldsymbol{L}_i$ be a point constraint. Then $r(\boldsymbol{P},\boldsymbol{L})$ is just the distance from point P to $\boldsymbol{L}_i$ ($r(\boldsymbol{P},\boldsymbol{L}_i) = \|\boldsymbol{P} - \boldsymbol{L}_i\|$). Figure 3.6 gives an illustration.

3.4.2 Line segment constraint

Let $\boldsymbol{L}_i$ be a line segment determined by its two end points P_0 and P_1. The distance $r(\boldsymbol{P},\boldsymbol{L}_i : P_0P_1)$ can be computed through vector calculation.

Let P be a test point in 3D space, V_1 be a vector from P_0 to P, V_2 be a vector from P_1 to P, V_L is a vector from P_0 to P_1 and V_c be a vector from P to P_c (The projection point from P to line segment P_0P_1). Then the projection vector Vc can be calculated:



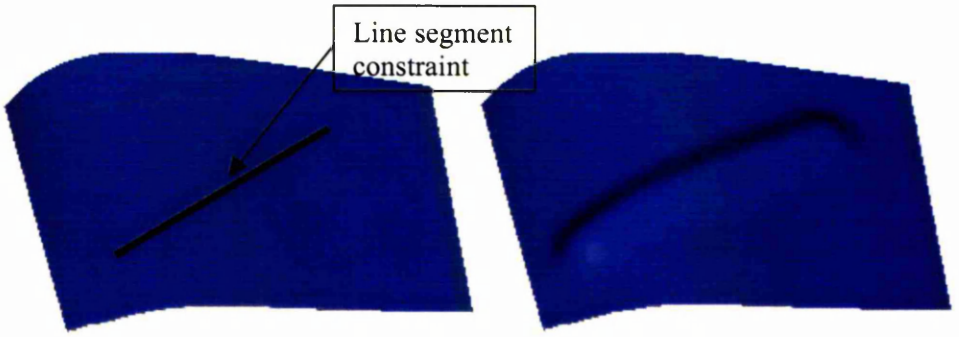
$$\left|\overrightarrow{X}\right| = \left|\overrightarrow{V_1}\right|\cos\theta = \left|\overrightarrow{V_1}\right| \cdot \frac{\overrightarrow{V_1} \bullet \overrightarrow{V_L}}{\left|\overrightarrow{V_1}\right| \cdot \left|\overrightarrow{V_L}\right|} = \frac{\overrightarrow{V_1} \bullet \overrightarrow{V_L}}{\left|\overrightarrow{V_L}\right|}$$

$$\alpha = \frac{\left|\overrightarrow{X}\right|}{\left|\overrightarrow{V_L}\right|} = \frac{\overrightarrow{V_1} \bullet \overrightarrow{V_L}}{\left|\overrightarrow{V_L}\right|^2}$$

$$P_c = (1-\alpha) * P_0 + \alpha * P_1$$

$$V_c = \overrightarrow{P_c P}$$

At last, the distance function $r(\boldsymbol{P}, \boldsymbol{L}_i : P_0P_1) = |\boldsymbol{V}_c|$. The line segment constraint of metaball deformation is given in figure 3.7.



(a) Original Surface (b) Deformed Surface

Figure 3.7: The deformation of line segment constraint

3.4.3 Polyline constraint

Let constraint L_i be a polyline define by $P_0P_1P_2...P_n$. For any line segment $P_{i-1}P_i (i = 1,2,3,..,n)$, we can obtain $r(P, P_{i-1}P_i)$ by the line segment constraint

$$r(P, L_i : P_0P_1P_2...P_n) = \min_{i\in[1,n]}\{r(P, P_{i-1}P_i)\} \qquad (3.8)$$

method as described above. The distance between any space point P and L_i is the minimum of the obtained distances.

An example is shown in figure 3.8.

Figure 3.8: The deformation of polyline constraint

3.4.4 Circle line constraint

Let L_i be a circle line whose radius is R_c. For simplicity, we transform the circle line onto the xz plane, and its centre is transformed into the origin. For any 3D point $\boldsymbol{P}$, we apply the same transformation and obtain $\widetilde{P} = (\widetilde{x}, \widetilde{y}, \widetilde{z})$. From figure 3.9, it is obvious that $OB = R_c, O\widetilde{P} = \sqrt{\widetilde{x}^2 + \widetilde{y}^2 + \widetilde{z}^2}$, thus

$$\widetilde{P}B^2 = AB^2 + \widetilde{y}^2 = (\sqrt{\widetilde{x}^2 + \widetilde{z}^2} - R_c)^2 + \widetilde{y}^2$$

Then

$$r(\boldsymbol{P}, \boldsymbol{L}_i) = \sqrt{R_c^2 + \widetilde{x}^2 + \widetilde{y}^2 + \widetilde{z}^2 - 2R_c\sqrt{\widetilde{x}^2 + \widetilde{z}^2}}$$

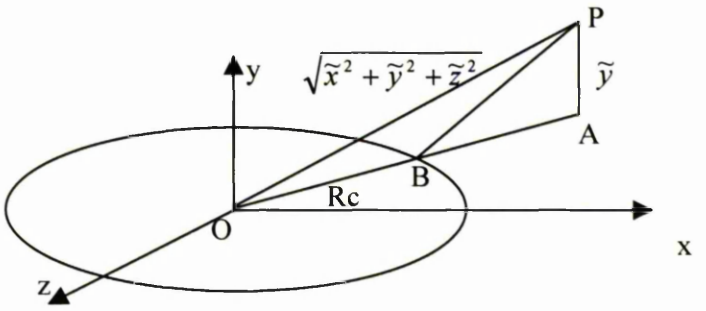


Figure 3.9: Distance calculation for a circle line

Its corresponding metaball is a torus whose major radius equals $R_c + R_i$ and minor radius equals $R_c - R_i$ as illustrated in figure 3.10.

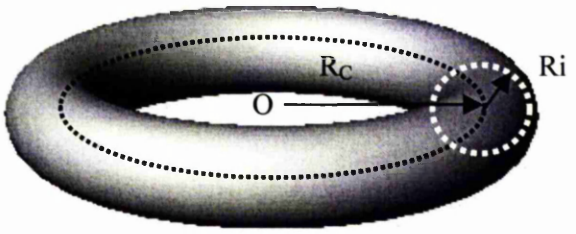


Figure 3.10: Generalized metaball for a circle line

3.4.5 NURBS curve constraint

Let L_i be a NURBS curve $C(u)$ of degree p. The minimum distance from a 3D point P to L_i either lies in its end points, or lies in the points satisfying the equation:

$$(P - C(u)) \cdot C'(u) = 0$$

Chapter 4 will give our algorithm to solve this equation. It is obvious that the corresponding metaball is a generalized cylinder. Figure 3.11 gives a sample of NURBS curve constraint.

Figure 3.11: The deformation of NURBS curve constraint

3.4.6 Disk constraint

Let L_i be a disk whose radius is R_C. We first calculate the distance r_1 from a 3D point P to the plane where the disk lies. If the perpendicular point of P lies within the disk, $r(\boldsymbol{P}, \boldsymbol{L}_i) = r_1$; otherwise we calculate the distance r_2 from P to the circle line, and set $r(\boldsymbol{P}, \boldsymbol{L}_i) = r_2$. The shape of corresponding generalized metaball of a disk is shown in figure 3.12. The result can easily be extended to the planar polygon constraint. Figure 3.13 gives the generalized metaball of a square.

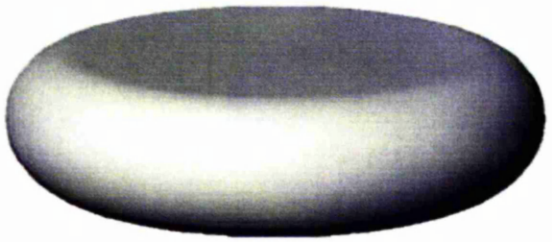


Figure 3.12: The generalized metaball of a disk

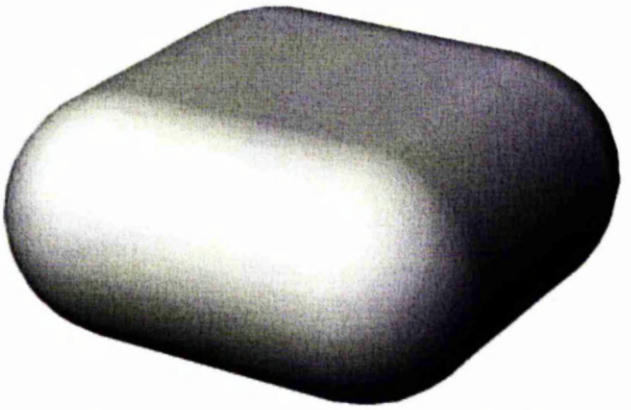


Figure 3.13: The generalized metaball of a square

3.4.7 Sphere constraint

Let L_i be a sphere whose radius is R_C, its centre is located in the origin of coordinate system ($O(x_c, y_c, z_c)$). Then the distance from a 3D point P to the sphere is

$$r(\boldsymbol{P}, \boldsymbol{L}_i) = \left| \sqrt{(x - x_c)^2 + (y - y_c)^2 + (z - z_c)^2} - R_c \right|$$

The cross section for this metaball is shown in figure 3.14.

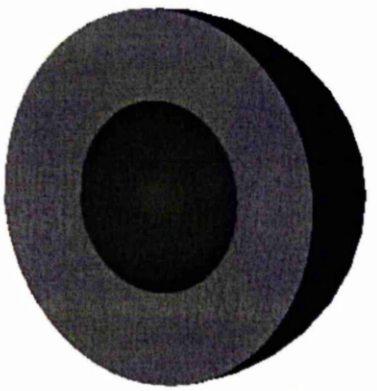


Figure 3.14: The generalized metaball of a sphere.

3.4.8 Cylinder constraint

Let L_i be a cylinder whose radius is R_c, and its height is h. We do some transformations so that its bottom surface lies on the xz plane and its centre line coincides with z axis (figure 3.15). We apply the same transformations to the 3D point P to obtain $\tilde{P} = (\tilde{x}, \tilde{y}, \tilde{z})$. The distance function is

$$\begin{cases} \min(R_c - \sqrt{\tilde{x}^2 + \tilde{z}^2}, \tilde{y}, h - \tilde{y}) & \text{if } \sqrt{\tilde{x}^2 + \tilde{z}^2} \le R_c \text{ and } 0 < \tilde{y} < h \\ -\tilde{y} & \text{if } \sqrt{\tilde{x}^2 + \tilde{z}^2} \le R_c \text{ and } \tilde{y} \le 0 \\ \tilde{y} - h & \text{if } \sqrt{\tilde{x}^2 + \tilde{z}^2} \le R_c \text{ and } \tilde{y} \ge h \\ \sqrt{\tilde{x}^2 + \tilde{z}^2} - R_c & \text{if } \sqrt{\tilde{x}^2 + \tilde{z}^2} > R_c \text{ and } 0 < \tilde{y} < h \\ \sqrt{R_c^2 + \tilde{x}^2 + \tilde{y}^2 + \tilde{z}^2 - 2R_c\sqrt{\tilde{x}^2 + \tilde{z}^2}} & \text{if } \sqrt{\tilde{x}^2 + \tilde{z}^2} > R_c \text{ and } \tilde{y} \le 0 \\ \sqrt{R_c^2 + \tilde{x}^2 + (\tilde{y} - h)^2 + \tilde{z}^2 - 2R\sqrt{\tilde{x}^2 + \tilde{y}^2}} & \text{if } \sqrt{\tilde{x}^2 + \tilde{z}^2} > R_c \text{ and } \tilde{y} \ge h \end{cases}$$

The outer surface of the generalized metaball for a cylinder constraint is shown in figure 3.16.

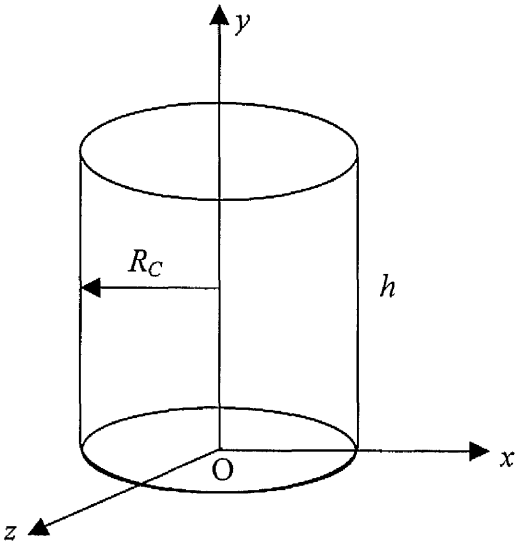


Figure 3.15: Cylinder constraint

Figure 3.16: The outer surface of the generalized metaball for a cylinder constraint

3.4.9 Sphere volume constraint

Let L_i be a sphere volume whose radius is R_C and its centre is $O(x_c, y_c, z_c)$. Let $r(\boldsymbol{P}, \boldsymbol{L}_i)$ equal zero if the 3D point P lies inside the sphere volume. Otherwise, the distance from P to the sphere volume is

$$r(\boldsymbol{P}, \boldsymbol{L}_i) = \sqrt{(x - x_c)^2 + (y - y_c)^2 + (z - z_c)^2} - R_c$$

3.4.10 Cube volume constraint

Let L_i be a cubic volume, whose edge length is *2a*. We apply transformations so that the centre of cube is at the origin and its edges are parallel to the three coordinate axes. Then we apply the same transformations to the 3D point P so that we get $\widetilde{P} = (\widetilde{x}, \widetilde{y}, \widetilde{z})$. If $\widetilde{P}$ lies inside, $r(\boldsymbol{P}, \boldsymbol{L}_i)$ equals to zero. Otherwise the point nearest to $\widetilde{P}$ either lies on the faces of the cubes, or lies on the edges, or lies on the vertices of the cube according to the position of $\widetilde{P}$. The distance can be measured from the nearest point of $\widetilde{P}$ to $\widetilde{P}$. The shape of the generalized metaball for a cube volume is shown in figure 3.17.

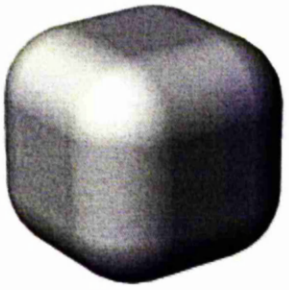
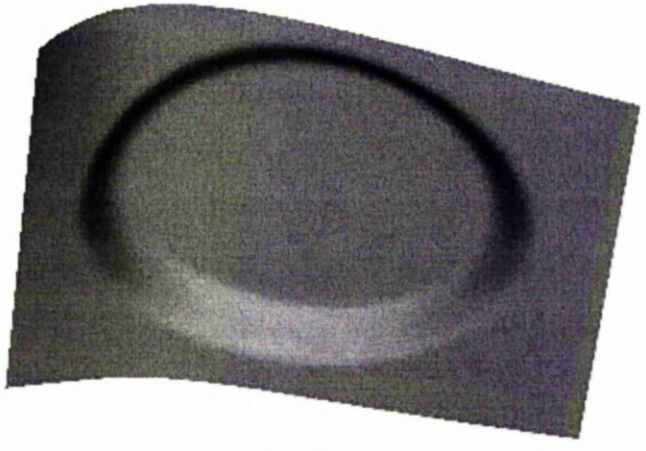


Figure 3.17: Generalized metaball of a cube volume

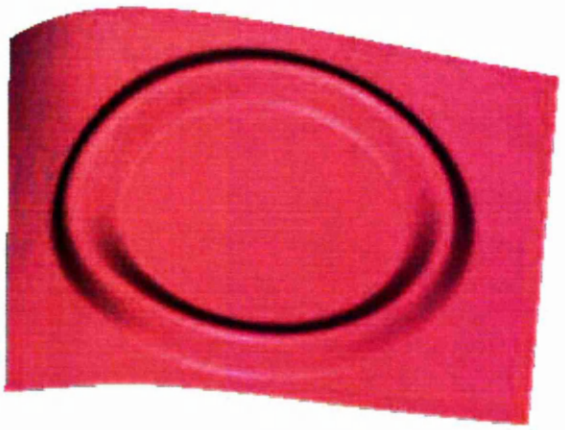
3.4.11 Summary

In this section, we presented the computation methods of the distance function $r(\boldsymbol{P}, \boldsymbol{L}_i)$ for some typical cases. For those constraints which are not listed above, their distance functions can be calculated similarly. Because this deformation model is a local one, the point will not be affected if the distance from it to the constraint is larger than the effective radius. Therefore, we can apply the bounding boxes for the individual constraints to improve the efficiency of the algorithm. If a point does not lie inside the bounding boxes of the constraint of metaball, the distance function does not need be calculated and the constraint has no effect on this point. Finally, some examples of constrained deformation on a single NURBS surface are given in figure 3.18.



(a) The generalized metaball for a disk

Figure 3.18



(b) The generalized metaball for a circle line

(c) The generalized metaball for a rectangle

Figure 3.18: Some general constraints

3.5 Deformations for Metaball Model

The general constraint deformations based on metaballs can be applied to both polygonal meshes and parametric surfaces. For polygonal meshes, it achieves faster computation for the displacement of vertices than in the parametric surface model. However, it may lose accuracy in the geometric representation so it is not suitable for CAD and geometry modelling. On the other hand, it is very useful for computer animation [8].

We will adopt this model for parametric surfaces, more specifically, NURBS surfaces. There are two ways to deform the surface: moving control points or modifying the weights. It is also possible to combine these two methods together to modify the shape of a NURBS surface.

3.5.1 Moving the control points

According to the locality property of a NURBS surface, moving a group of control points will change the shape of NURBS locally. Therefore, we apply general constrained deformation to the control net of a NURBS surface to achieve the metaball deformation on the surface. We divide the procedure of deformation for the NURBS surface into three steps:

- **Refining the control net of the NURBS surface**

  The control net is refined to increase the density of the control points. The density of control points is decided by the effective radius R. The calculation of the density is given by (3.9).

  $$\lambda = \frac{N}{\pi R^2} \qquad (3.9)$$

  where N is the number of control points inside the circle whose radius is R. N can be counted from the distance function $r(\boldsymbol{P}, \boldsymbol{L}_i)$ for the point constraint. The density λ is the parameter controlled by the user.

- **Moving the control points**

  The displacement of an individual control point is calculated via the potential function (3.4). If the value of potential function is greater than zero, the control point is moved along the normal vector of this control point.

- **Removing unnecessary knots (unnecessary control points)**

  After moving the control points, we remove any unnecessary knots (unnecessary control points) to reduce the memory consumption and

simplify the data for the NURBS surface. The algorithm of knot removal for a NURBS surface has been given in section 2.3.4.

The whole procedure of moving a control point for a general constraint is illustrated in figure 3.19.

(a) The original surface (4×4 control points)

(b) The surface after knot refinement (19×19 control points)

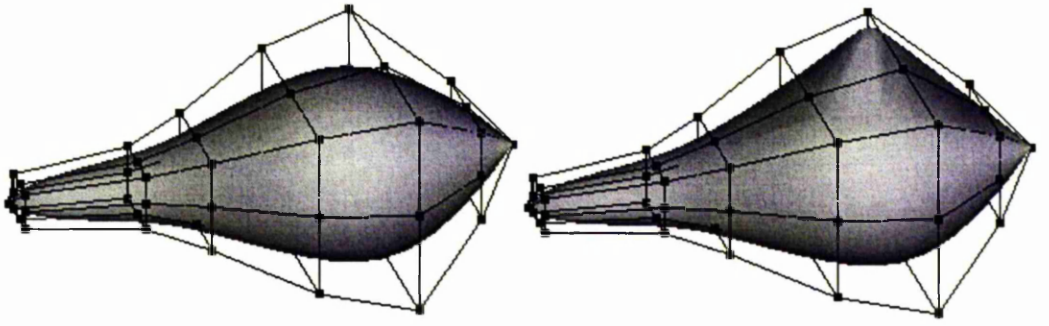
(c) The surface after deformation (19×19 control points)

(d) The surface after knot removal (10×10 control points)

Figure 3.19: The procedure of moving control points for metaball deformation

3.5.2 Modifying the weights

Weight-based shape modifications rely on the geometric meaning of the weights. From section 2.2.3, we know that as a weight w associating a control point P increases / decreases, the curve or surface will be pulled / pushed toward / away from P. From [35], we also know that a moderately larger number (e.g. 100) causes the curve or surface to pass very close to the control point P. Figure 3.20 gives an example.



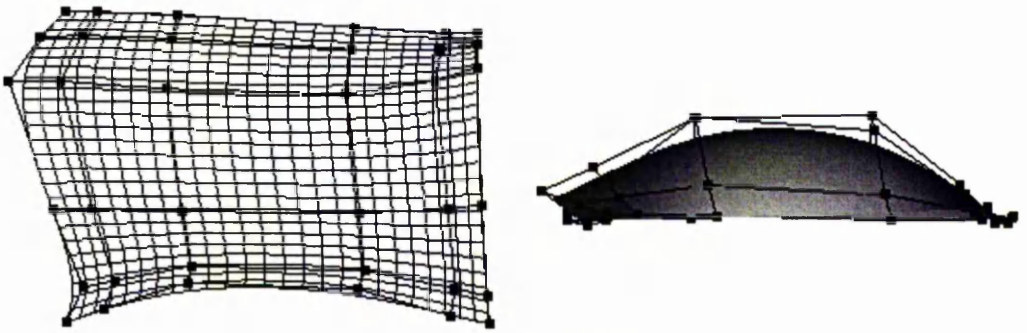
(a) The weight set to 1. (b) The weight set to 100.

Figure 2.20: Modifying a weight for a revolved surface

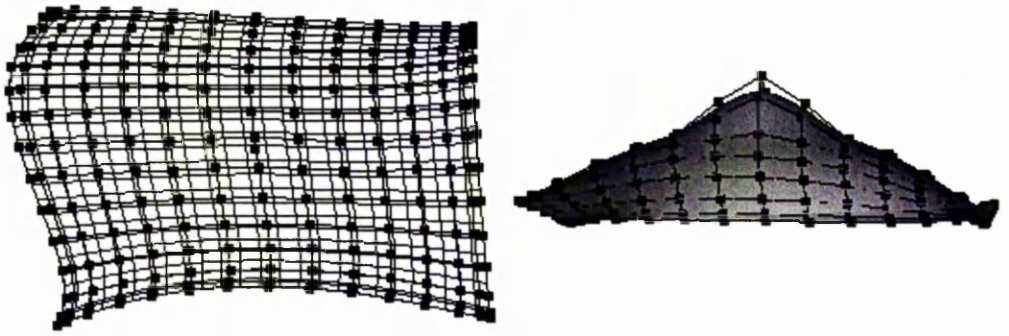
In the metaball deformation model, the control net is refined before applying the general constraints. The knot refinement of a NURBS surface brings the control net closer to the surface and modifying the weights only changes the shape of the NURBS surface within the control net. Therefore, it is not possible practically to use only the weight-based shape modification to achieve the metaball deformation. Together with control point based modification, the weight-based modification can be used as a refined method in the metaball deformation model. We apply equation 3.10 similarly to equation 3.6 to change the weights.

$$Deform(w) = w + \Delta D_i F(r(\boldsymbol{P}, \boldsymbol{L}_i), R_i) \qquad (3.6)$$

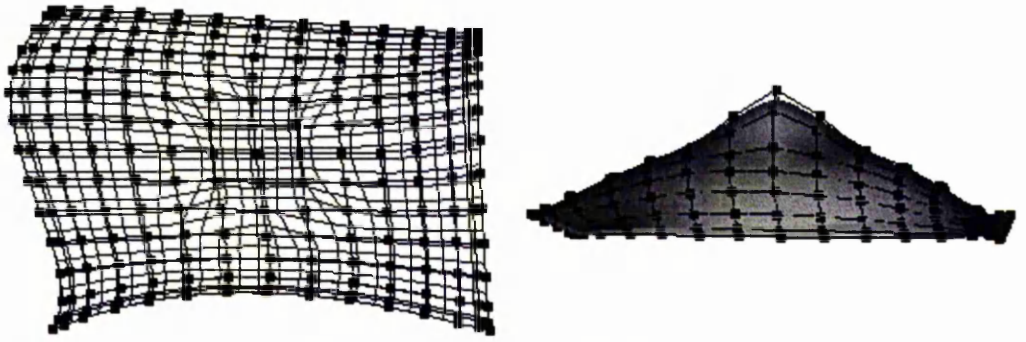
Figure 3.21 gives an illustration which shows the effect of using weight-based modification combined with control point based modification.



(a) The original surface



(b) The control point based modification



(c) Weight based modification combining with control point based modification

Figure 3.21: Applying weight based modification to the metaball deformation

3.6 Closing

This chapter has generalized the method for general constrained deformations based on generalized metaballs. Furthermore we apply it on the NURBS surface by both moving the control points and changing the weights. This chapter also presents a list of pictures to demonstrate the different types of constraints. All these pictures are generated by author's testbed software.

To solve the problem of the minimum distance from a 3D point to the NURBS curve, chapter 4 gives an innovative algorithm and also extends the method for NURBS surfaces. Chapter 5 presents a solution for tessellating the trimmed NURBS surface on which we apply the metaball deformation.

Chapter 4

Point Inversion and Projection

4.1 What and why?

Projecting a test point to a NURBS curve or surface finds the closest point on the curve or surface and point inversion finds the corresponding parameters *(u)* for the curve or (u, v) for the surface for this point. Both point projection and point inversion are common problems for interactively selecting a NURBS curve or surface and curve or surface fitting. When the user clicks somewhere near the curve or surface, we can find out which curve or surface has been selected based on the minimum distance between the nearest point on the curve or surface and the test point. In curve and surface fitting we need to calculate the tolerance, which is the minimum distance from the interpolation point to the desired curve or surface.

In order to solve the distance function of the NURBS curve constraint in the metaball deformation model, we need to find out the minimum distance between the 3D point and the closest point on the curve, which is to solve the problem of point projection for the NURBS curve. Point projection for NURBS

surfaces is the extension of the algorithm for NURBS curves and can also provide the solution of the distance function for the NURBS surface constraint.

4.2 Previous work

Much of the early work in this area comes from the robotics and computational geometry communities [51]. Their work has often focused on use of polygonal models for finding the minimum distance between two geometric objects. Chin [52] and Edelsbrunner [53] both describe O(logN) algorithms for finding the minimum distance between two convex polygons. However, all these algorithms involve presumably large number of tests on polygons and the result for the minimum distance is not accurate enough for the computer graphics and computer aided design communities. In contrast, parametric models provide more accurate solutions.

Using a parametric model, Mortensen [54] gave a numerical approach to this problem. For the calculation of the minimum distance between the test point and the curve, we need to find a vector (p-q) from the point p to the curve $q(u)$ that is perpendicular to the tangent vector p'' at p. Figure 4.1 shows the vector geometry. Mathematically, we express the required conditions as

$$d_{\min} = |p-q| \quad \text{when } (p-q)\cdot p'' = 0 .$$

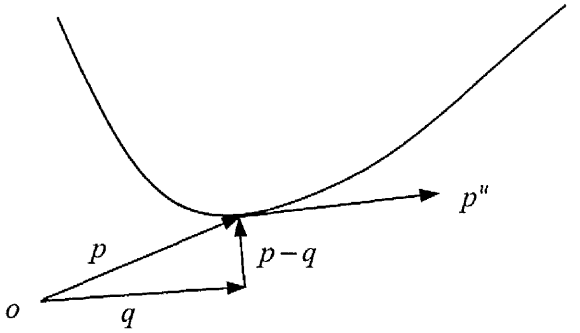
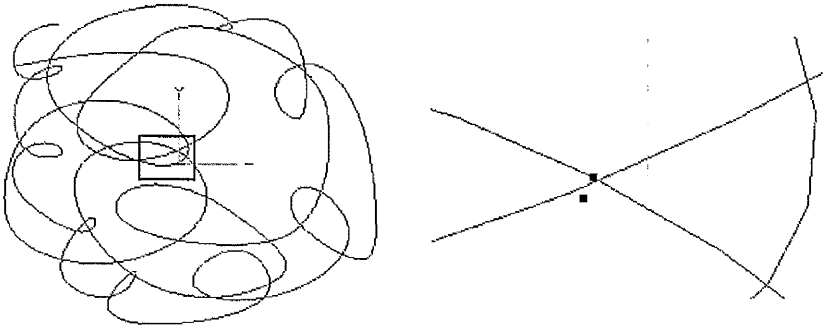


Figure 4.1: Minimum distance between a point and a curve

He chose to use the Newton-Raphson method to the find the roots of the polynomial $(p-q)\cdot p^{u}=0$. However, he needed to find a good initial value for the Newton-Raphson method. Similar to the curve, finding the minimum distance between a point and a surface involves finding the minimum distance between a point and a plane. Let the point be denoted by q and the plane by kn, where n is the unit normal vector. Let p denote the point on the plane closest to q. Then we have $d_{\min}=|p-q|$, where p must satisfy $(p-q)\times n=0$. Finally, we also need the Newton-Raphson method to find the roots. Limaiem [55] has presented another approach to find the minimum distance to convex parametric curves and surfaces. Lin [56] provides the approach for finding the minimum distance for concave surfaces. Both approaches use the Newton-Raphson method to find the roots for some distance equations. Therefore, both finding the minimum distance between a point and a curve and between a point and a surface need a good initial value for achieving the convergence result.

However this initial value is hard to obtain due to the complexity of the NURBS curve or surface shape [57]. Furthermore, the Newton-Raphson method may give the wrong answer when projecting a point near the intersection point of the NURBS curve (figure 4.2) and it fails quite often for the points near the boundaries of the surface [57].



(a) Cubic NURBS curve (73 control points). (b) Wrong point projection.

Figure 4.2: Wrong result for point projection on the complex curve

Piegl and Tiller [57] recently provided another method to solve the point projection problem for a NURBS surface. Their algorithm consists of three steps:

- **Decompose the NURBS surface into quadrilaterals.**
- **Project the test point onto the closest quadrilateral.**
- **Recover parameters from the closest quadrilateral.**

Obviously, decomposing the NURBS surface into quadrilaterals and finding the closest quadrilateral are very expensive.

Our approach for point projection and point inversion provides a good initial value for the Newton-Raphson method, reduces the computation of the algorithm and increases the stability of the recursive function for the Newton-Raphson method.

4.3 Outline of algorithm

4.3.1 Algorithm for NURBS curve

Our approach consists of three stages: *subdivision of the NURBS curve*, *control polygon detection* and *the relationship between the test point and the Bézier subcurve*. In the first stage, we decompose the NURBS curve into its piecewise Bézier form (see section 2.3.2). In section 4.4, we give the algorithm to find the 2D/3D simple and convex control polygon of the Bézier subcurve. In section 4.5, we analyse the relationship between the test point and control polygon and in section 4.6 we give the overall algorithm.

4.3.2 Algorithm for NURBS surface

Similar to the algorithm for NURBS curve, first, we decompose the surface into a set of Bézier patches (section 2.3.2). Then we check every control point net of the Bézier patch is a valid control point net or not, which will be discussed in section 4.4. After obtaining a valid control point net, we analyse the relationship between the test point and the Bézier patch and discard the Bézier patch whose closest point lies on the one of four boundary curves (Figure 4.3). Finally, we project the test point to the candidate Bézier patches to obtain the candidate points and select the closest one as the result. The accuracy of the closest point can be improved by using the Newton-Raphson method.

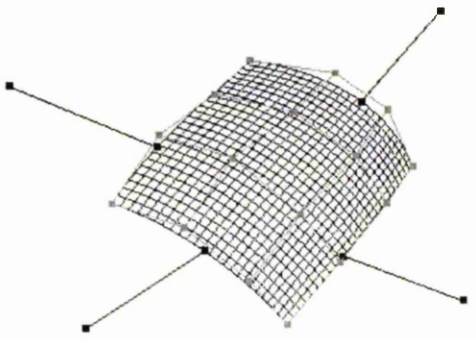


Figure 4.3: Discarded Bézier patch

4.4 Control Polygon and Control Point Net detection

For a NURBS curve, before we can establish the relationship between the test point and the Bézier subcurve, we need to classify the control polygons to simplify the problem. As we know, polygons can be divided into *simple* and *non-simple* polygons. Simple means no crossing edges. As shown in figure 4.4, polygon (5) is a non-simple polygon. Furthermore, simple polygons can be grouped as *convex* and *concave* polygons. Polygons (1), (2), (3) are convex and (4) is concave. Now, we define the *valid* control polygon as the simple and

convex polygon in 2D. In order to generalize the definition to 3D, we give a fast way to check whether the control polygon is valid or not both in 2D and 3D.

For a NURBS surface, we only analyse the relationship between the test point and the *valid* control point net. After the definition of a *valid* control polygon, we can specify the *valid* control point net, as every polygon in both the U and V parameter directions is valid.

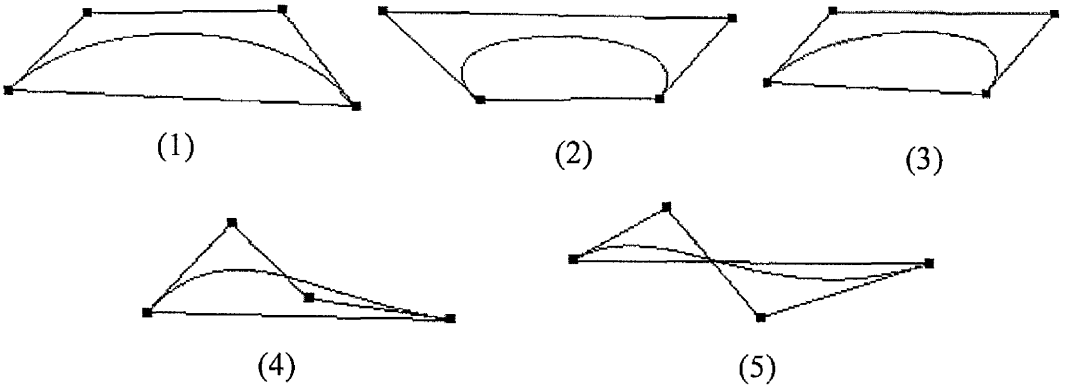
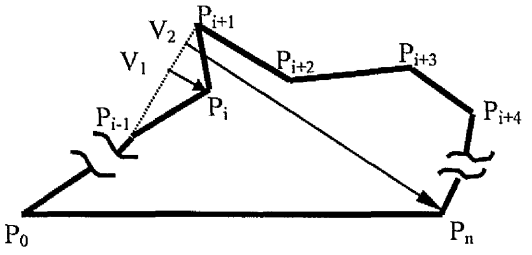


Figure 4.4: Type of control polygons of 2D cubic Bézier subcurve.

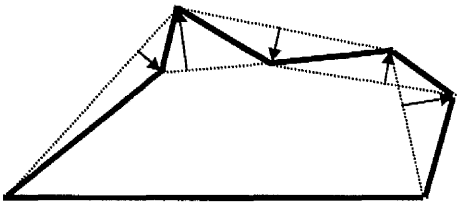
4.4.1 Fast valid control polygon detection algorithm

We give a fast way to detect whether the control polygon is simple and convex, or not, by checking the dot product result of two vectors. As shown in figure 4.5(a), for a given control polygon of a Bézier curve of degree p ($p>2$), we can detect the sign of the dot product $R = \overrightarrow{V_1P_i} \bullet \overrightarrow{V_2P_n}(i < (n/2))$ or $R = \overrightarrow{V_1P_i} \bullet \overrightarrow{V_2P_0}(i \geq (n/2))$ to determine whether vertex P_i is in the "convex" direction or not. If the result of dot product is greater than zero, then this vertex is in "concave" direction. Note that edge P_0P_n is the chord of Bézier subcurve and vertices P_0, P_1 ... P_{n-1}, P_n are the control points. V_1 and V_2 are the points projecting from the vertex P_i and one of endpoints: P_n ($^{i<(n/2)}$) or P_0

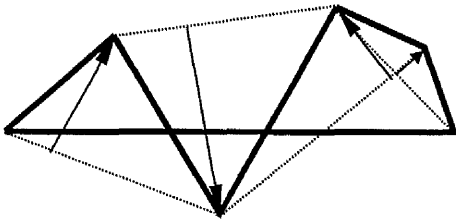
($i \geq (n/2)$) to line segment $P_{i-1}P_{i+1}$, respectively. If the dot product R is negative, we move to the next vertex and repeat this procedure until we find one is positive or finish detecting all vertices except P_0 and P_n. If all results are negative then the control polygon is a simple and convex one.



(a) Dot product of two vectors.



(b) Vertices direction for concave.



(c) Vertices direction for non-simple polygon.

Figure 4.5: Valid polygon detection

This algorithm also works for detecting non-simple polygons. As shown in figure 4.5(c), although this algorithm does not tell you the vertex direction properly, it does tell you it is not a convex polygon no matter if it is a concave or non-simple one. This algorithm can generalise to 3D. We will call the 3D simple and convex polygon which satisfies the dot product conditions as previously discussed a 3D valid control polygon.

4.4.2 Pseudo code

In summary, the simple and convex polygon (valid polygon) detection algorithm is shown as follows:

Algorithm 1. Is_Valid_Polygon *{Detecting the simple and convex polygon}*

Input*: The control polygon P of Bézier subcurve. n is the highest index in control polygon.*

Output*: the result of detection.*

begin

for $i = 1$ *to* $i < n$ ***by*** $i{+}{+}$ ***do***

begin

compute projection vector $\overrightarrow{V_1P_i}$;

if $i < (n/2)$ ***then***

compute projection vector $\overrightarrow{V_1P_n}$;

$R = \overrightarrow{V_1P_i} \bullet \overrightarrow{V_1P_n}$ *;{R is the result of dot product}*

else

compute projection vector $\overrightarrow{V_1P_0}$;

$R = \overrightarrow{V_1P_i} \bullet \overrightarrow{V_1P_0}$ *;{R is the result of dot product}*

end if

if $R > 0$ ***then return*** *FALSE;{The polygon is not a simple or convex one}*

end if

end*{End of the loop for}*

return *TRUE;{It is a simple and convex polygon}*

End *of Algorithm 1.*

Finally, we define the control point net whose polygons in both the U and V parameter directions are valid as the valid control point net. The detection algorithm is given as follows:

Algorithm 2. Is_Vaild_CP_Net *{Detecting the valid control point net}*

Input*: The control point net of Bézier patch. n is the highest index of control points in U direction. m is the highest index of control points in V direction.*

Ouput*: the result of detection.*

```
begin
 for i = 0 to i < m by i++ do
       {Detect every control polygon in U direction}
   begin
     generate a control polygon P;
     if (Is_Valid_Polygon(P) return FALSE) then
       return FALSE; {control point net is not valid.}
     end if
   end {End of loop for}

for i = 0 to i < n by i++ do
   {Detect every control polygon in V direction}
   begin
     generate a control polygon P;
     if (Is_Valid_Polygon(P) return FALSE) then
       return FALSE; {control point net is not valid.}
     end if
   end {End of loop for}

return TRUE;
End of Algorithm 2.
```

4.5 The Relationship between the test point and Bézier curve or Bézier Patch

For a curve, after decomposition, we obtain a set of Bézier subcurves. When detecting the valid control polygon, if we do not get a desired control polygon, we continue to subdivide the Bézier subcurve until we get a valid control polygon or the control polygon is flat enough. For a valid polygon, we analyse the relationship between the test point and the valid control polygon of the Bézier subcurve.

For a surface, after decomposition, we obtain a set of Bézier patches. When detecting the valid control point net, if we do not get a desired control point net, we continue to subdivide the Bézier patch until we get a valid one or the control point net is flat enough.

For a valid control point net, we analyse the relationship between the test point and the valid control point net of the Bézier patch. As shown in figure 4.6, for given a valid control point net, we can extract valid control polygons both in the U direction and V directions. Therefore, we can analyse the relationship between each valid control polygon and the test point to generalize the relationship between the valid control point net and the test point. We analyse the relationship between the valid control polygon (Bézier curve) and the test point as follows:

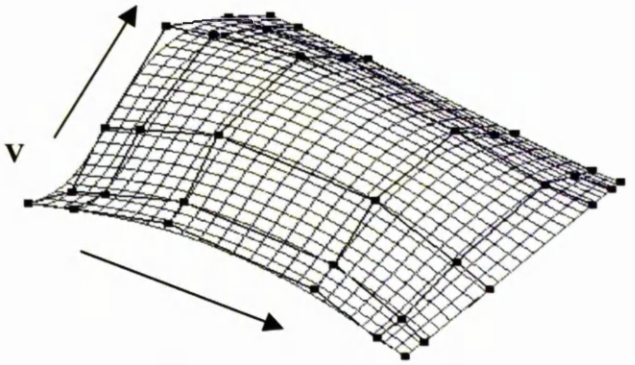


Figure 4.6: Control point net

Lemma 1. As shown in figure 4.7, suppose that an n (n>1) degree 2D Bézier curve has a valid control polygon $P_0,\ldots,P_n$ and a test point P. We have four dot products $R_1 = \overrightarrow{P_0P} \bullet \overrightarrow{P_0P_1}$, $R_2 = \overrightarrow{PP_n} \bullet \overrightarrow{P_{n-1}P_n}$, $R_3 = \overrightarrow{P_nP_0} \bullet \overrightarrow{P_nP}$ and $R_4 = \overrightarrow{P_nP_0} \bullet \overrightarrow{P_0P}$. If $R_1 < 0$ or $R_2 < 0$ and $R_3 * R_4 > 0$, then the nearest point **must** be one of endpoints (P_0 or P_n).

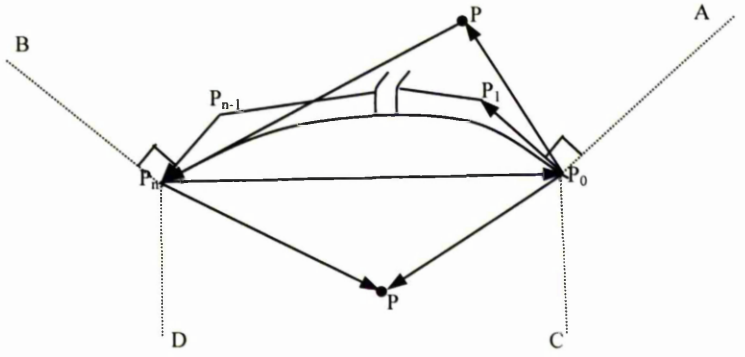


Figure 4.7: Conditions for 2D Bézier curve (satisfied)

Proof. The dot product R_1 and R_2 will be positive if the test point is within the area AP_0P_nB and the multiplication R_3*R_4 will be negative if the test point is within the area CP_0P_nD.

According to the formula (4.1) of the derivative of a Bézier curve, we can obtain the formulas for the end derivatives (4.2).

$$C'(u) = n\sum_{i=0}^{n-1} B_{i,n-1}(u)(P_{i+1} - P_i) \qquad (4.1)$$

$$\begin{cases} C'(0) = n(P_1 - P_0) \\ C'(1) = n(P_n - P_{n-1}) \end{cases} \qquad (4.2)$$

where $B_{i,n-1}(u)$ is the Bézier basis function and $P_0,\ldots,P_n$ are the control points, for an n degree Bézier curve.

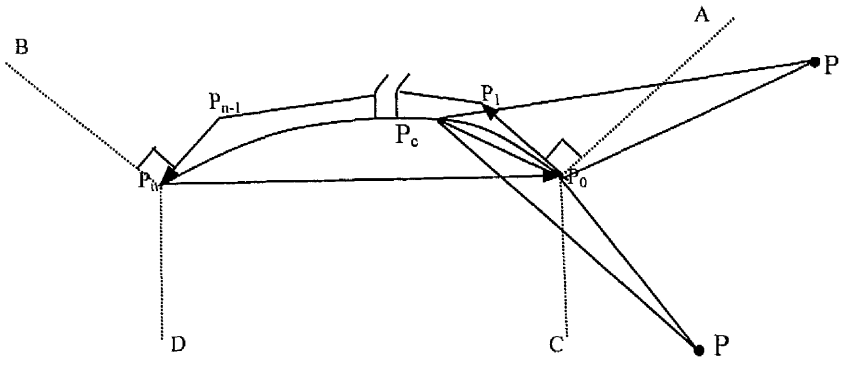


Figure 4.8: Conditions for 2D Bézier curve (unsatisfied)

Notice that, in figure 4.8, $C'(0)$ is the vector $\overrightarrow{P_0P_1}$ and $C'(1)$ is the vector $\overrightarrow{P_{n-1}P_n}$. We assume that P_c is any point on the Bézier subcurve. If the control polygon is simple and convex, according to the strong convex hull property, P_c must be inside the control polygon. Therefore, $\angle PP_0P_c$ must be larger than 90° if P is outside of area AP_0P_nB and area CP_0P_nD. Furthermore, according to the property of the triangle, $\angle PP_0P_c$ is larger than $\angle P_0P_cP$, therefore, $|PP_c|$ is larger than $|PP_0|$. It is obvious that P_0 is the nearest point of P.

We can extend this rule to a 3D valid control polygon of a Bézier curve. Although in 3D space the valid control polygon sometimes does not lie on the same plane, we can set the control polygon plane as the plane constructed from the first two edges of the control polygon. As shown in figure 4.9, we also have

$\left|P_c' P'\right| > \left|P_0 P'\right|$ (P' and P_c' are the projection points from P and P_c to control polygon plane respectively and P_c is a any point on the Bézier curve.)

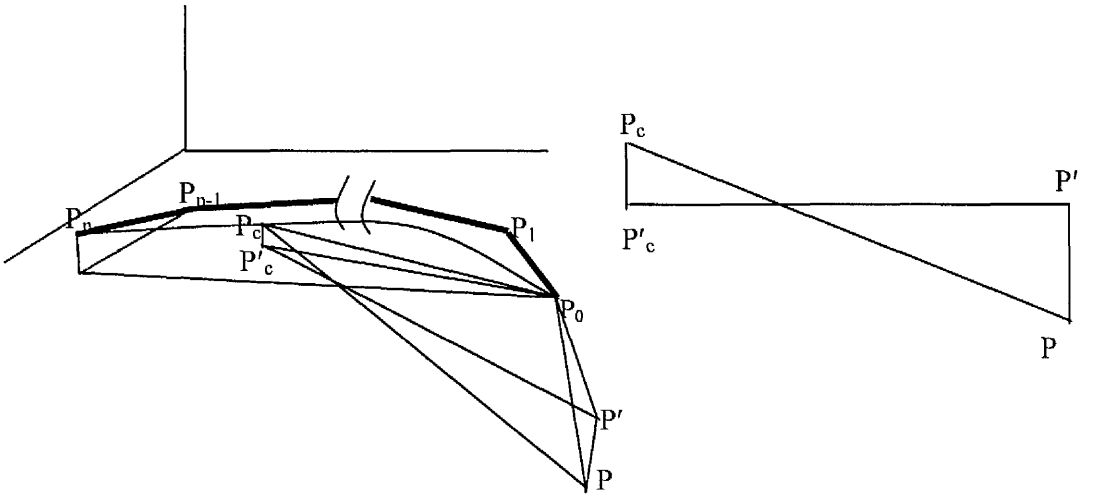


Figure 4.9: Conditions for 3D Bézier curve.

Also

$\left|P_c P\right| = \sqrt{\left|P_c' P'\right|^2 + (\left|PP'\right| + \left|P_c P_c'\right|)^2}$ (Refer to figure 4.9) $\left|P_0 P\right| = \sqrt{\left|P_0 P'\right|^2 + \left|PP'\right|^2}$

Therefore $\left|P_c P\right| > \left|P_0 P\right|$.

Theorem 1. Suppose that an n (n>1) degree 2D/3D Bézier subcurve has a valid control polygon $P_0, \ldots, P_n$ and a test point P is in the 3D space. We have four dot products $R_1 = \overrightarrow{P_0 P} \bullet \overrightarrow{P_0 P_1}$, $R_2 = \overrightarrow{PP_n} \bullet \overrightarrow{P_{n-1} P_n}$, $R_3 = \overrightarrow{P_n P_0} \bullet \overrightarrow{P_n P}$ and $R_4 = \overrightarrow{P_n P_0} \bullet \overrightarrow{P_0 P}$. If $R_1 < 0$ or $R_2 < 0$ and $R_3 * R_4 > 0$, then the nearest point **must** be one of endpoints (P_0 or P_n).

For the curve case, we analyse the relationship between the test point and a valid control polygon. We do four dot products, which are described in Theorem 1. If this case does not satisfy the conditions in Theorem 1, the

nearest point must be one of the endpoints. Furthermore we can discard this Bézier subcurve. In summary, we give the algorithm as follows:

Algorithm 3. Point_Nearest_Bézier_Curve *(Detecting whether the nearest point is one of endpoints or not according to the result of four dot products).*

Input*: The control polygon P of Bézier curve. n is the highest index in control polygon.*

Output*: the result of detection.*

begin

$R_1 = \overrightarrow{P_0P} \bullet \overrightarrow{P_0P_1}$, $R_2 = \overrightarrow{PP_n} \bullet \overrightarrow{P_{n-1}P_n}$

$R_3 = \overrightarrow{P_nP_0} \bullet \overrightarrow{P_nP}$, $R_4 = \overrightarrow{P_nP_0} \bullet \overrightarrow{P_0P}$

if $R_1 < 0$ *or* $R_2 < 0$ *and* $R_3 * R_4 > 0$

then return *FALSE;*

else return *TRUE;*

end if

End *of Algorithm 3.*

Then, for the surface case, we consider the relationship between the test point and a valid control point net. We also do four dot products with every polygon in the U direction and V direction. If every polygon in the U direction or every polygon in the V direction does not satisfy the dot product conditions described in Theorem 1, the nearest point must be on the one of four boundary curves. Furthermore we can discard this Bézier patch.

In summary, we give the algorithm as follows:

Algorithm 4 Point_Nearest_Bézier_Patch

{Detecting whether the nearest point is the point on the boundary curves.}

***Input**: The control point net of Bézier patch. n is the highest index of control points in U direction. m is the highest index of control points in V direction.*

***Output**: the result of detection.*

```
begin
 Flag ← FALSE;
 for  i = 0 to i < m by i++ do
       {Detect every control polygon in U direction}
   begin
     generate a control polygon P;
     if  (Point_Nearest_Bézier_Curve(P) return TRUE) then
        Flag ← TRUE;
       break;
     end if
   end {End of loop for}
 if Flag == FALSE then
    return FALSE; { the nearest point is the point on the boundary curves.}
end if
Flag ← FALSE;
for i = 0 to i < n by i++ do
       {Detect every control polygon in V direction}
   begin
     generate a control polygon P;
     if  (Point_Nearest_Bézier_Curve(P) return TRUE) then
        Flag ← TRUE;
       break;
     end if
   end {End of loop for}
 if Flag == FALSE then
    return FALSE; { the nearest point is the point on the boundary curves.}
end if
return TRUE; { the nearest point is the point on the Bézier patch.}
End of Algorithm 4
```

4.6 Find the closest point on the NURBS curve

By extracting the Bézier subcurves as candidates, we subdivide these candidate subcurves recursively until the control polygon is flat enough or reaching a recursion limit. By "flat enough", we mean that the control polygon is close enough to a straight line so that we can approximate the nearest point (candidate point) in that region by calculating a projection vector from the test point to the straight line. The algorithm for finding the candidate points from the NURBS curve is given as follows.

Algorithm 5. Find the candidate points for projecting a point to a NURBS curve.

```
Nearest_Candidate_Points_Curve(P,n ,U,m,Valid,pt)
      P: array point of control points.
      n: The highest index in the control points.
      U: array pointer of knot vector.
      m: The highest index in the knot vector.
      Valid: flag variable for valid polygon.
            (TRUE is valid. FALSE is invalid)
      pt: 2D/3D test point.
Begin
 if(Curve is a Bezier curve)
   if (Valid) then
      if (Point_Nearest_Bézier_Curve return FALSE) then
         return no candidate point was found;
      else
         if(control polygon is flat enough or recursive limit reached) then
            return the candidate point on the Bezier subcurve;
         end if
      end if
```

```
    else{Valid is FALSE}
        if(Is_Valid_Polygon return TRUE) then
          Valid = TRUE;
        end if
    end if
    Subdivide the curve at midpoint of knot vector;
    Nearest_Candidate_Points_Curve(left half);
    Nearest_Candidate_Points_Curve(right half);
else {The curve is not a Bezier curve.}
  Subdivide the curve at midpoint of knot vector;
end if
end
End of Algorithm 5.
```

We start the recursive function with *Valid* set to FALSE. After the termination of this function, we select the nearest point from candidate points and improve its accuracy by applying the Newton-Raphson method. Section 4.10 gives some examples of projecting a set of points to the NURBS curve both in 2D and 3D.

4.7 Find the closest point on the NURBS surface

By extracting the Bézier patches as candidates, we subdivide these candidate patches recursively until the control point net is flat enough or reaching the recursion limit. By "flat enough", we mean that the control point net is close enough to a plane so that we can approximate the nearest point (candidate point) in that region by projecting the test point to the approximate plane. The algorithm for finding the candidate points from the NURBS surface is given as follows:

Algorithm 6 Find the candidate points for projecting a point to a NURBS surface.

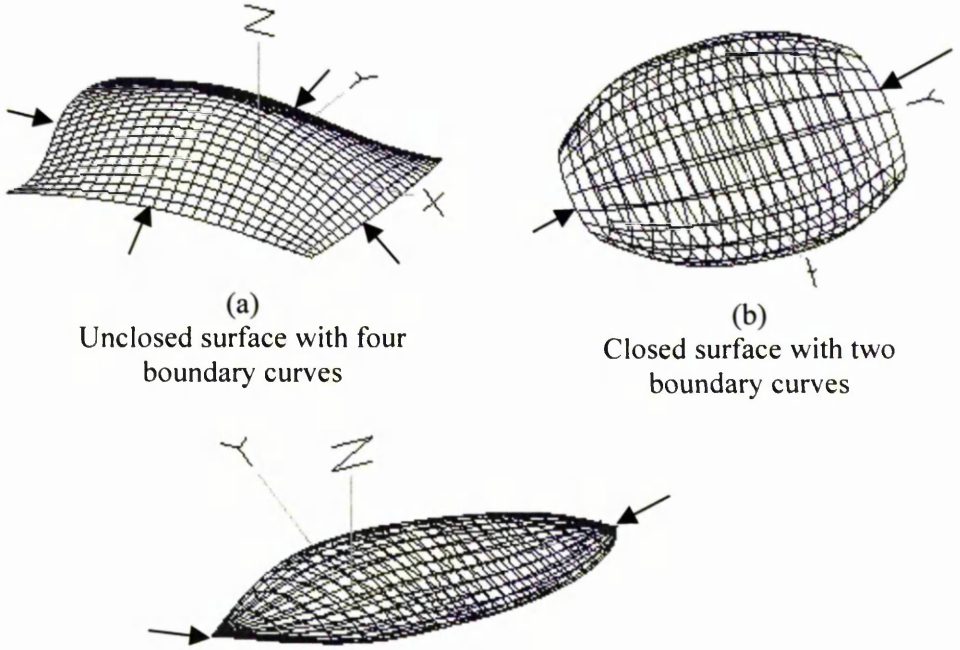
```
Nearest_Candidate_Points_Surface(pSur,Dir,Valid,pt)
    pSur: The pointer of NURBS surface object.
    Dir: The direction for splitting. (TRUE for U direction and FALSE for V direction.)
    Valid: flag variable for valid control point net. (TRUE is valid. FALSE is invalid)
    pt: 3D test point.
Begin
 if(Surface is a Bézier patch)
  if (Valid is TRUE) then
    if (Point_Nearest_Bézier_Patch return FALSE) then
      return no candidate point was found;
    else
      if(control point net is flat enough or recursive limit reached) then
      return the candidate point on the Bézier patch;
      end if
    end if
  else{Valid is FALSE}
    if(Is_Vaild_CP_Net return TRUE) then   Valid = TRUE;
    end if
  end if
  if(Dir is TRUE)
    Split the surface at midpoint of U knot vector;
  else
    Split the surface at midpoint of V knot vector;
  end if
  Nearest_Candidate_Points_Surface(pSur1,Dir,Valid,pt);
  {pSur1 is one half surface in the U or V direction.}
  Nearest_Candidate_Points_Surface(pSur1,Dir,Valid,pt);
  {pSur2 is the other half surface in the U or V direction.}
else {The surface is not a Bézier patch.}
   if (Dir is TRUE)   Split the surface at midpoint of U knot vector;
   else               Split the surface at midpoint of V knot vector;
  end if
End
```

End of Algorithm 6

4.8 Boundary conditions of NURBS surface

After termination of the recursive function described in the last section, we select the closest point from the candidate points as well as the points projecting from the test point to the boundary curves.

If a NURBS surface is not closed both in the U direction and V direction, it has four boundary curves. On the other hand, if the surface is closed in the u direction or v direction, it has two boundary curves. In some cases, it has only two "boundary points" as shown in figure 4.10.



(a)
Unclosed surface with four boundary curves

(b)
Closed surface with two boundary curves

(c)
Closed surface with two boundary points

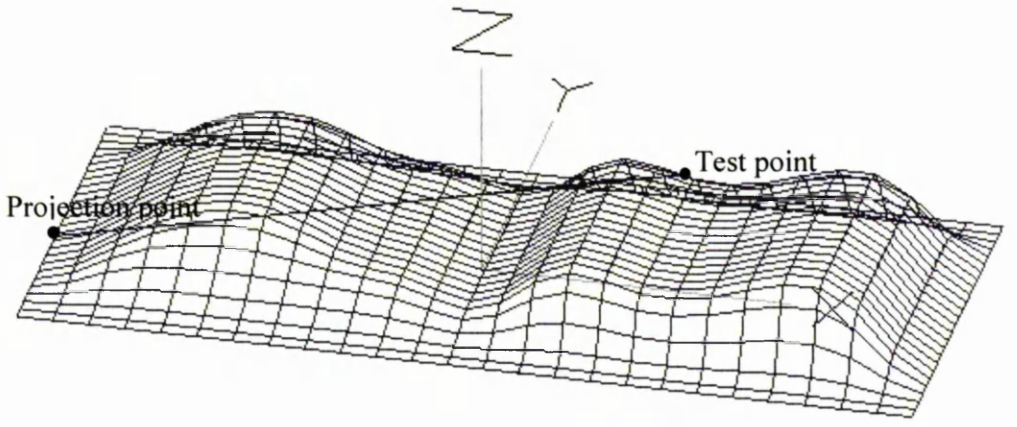
Figure 4.10: Boundary curves of the NURBS surface

For an unclosed NURBS surface, we have four projection points, projected from the test point to the four boundary curves. For a closed NURBS surface,

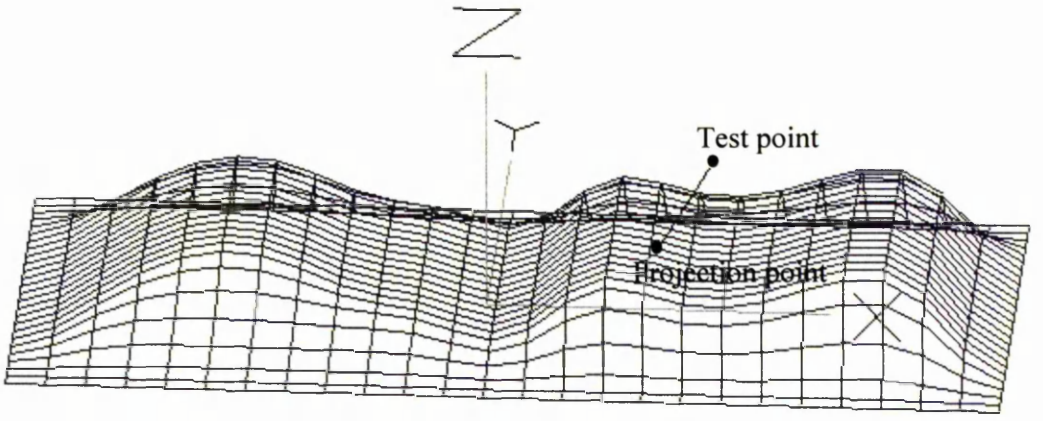
we also have two projection points. From the projection points and candidate points, we can find the closest one as the solution for the point projection for the NURBS surface.

4.9 The Newton-Raphson method for a NURBS surface

The Newton-Raphson method is used to improve the accuracy of the closest point. Piegl and Tiller [17] used the Newton-Raphson method to minimize the distance between the test point and the whole NURBS surface. However after some tests, we find Newton-Raphson occasionally still gives the wrong answer even with a quite good initial value when applied to the whole NURBS surface. Therefore, instead of applying it to the whole NURBS surface, we apply the Newton-Raphson method to the quadrilateral which is a "flat enough" Bézier patch.



(a) Wrong point projection (applying Newton-Raphson method to the whole NURBS surface).



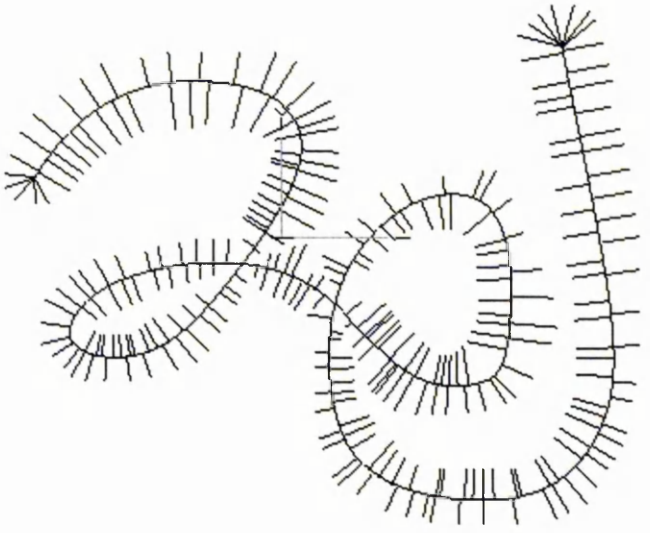
(b) Right point projection (applying Newton-Raphson method to the Bézier patch).

Figure 4.10: The Newton-Raphson method for surface.

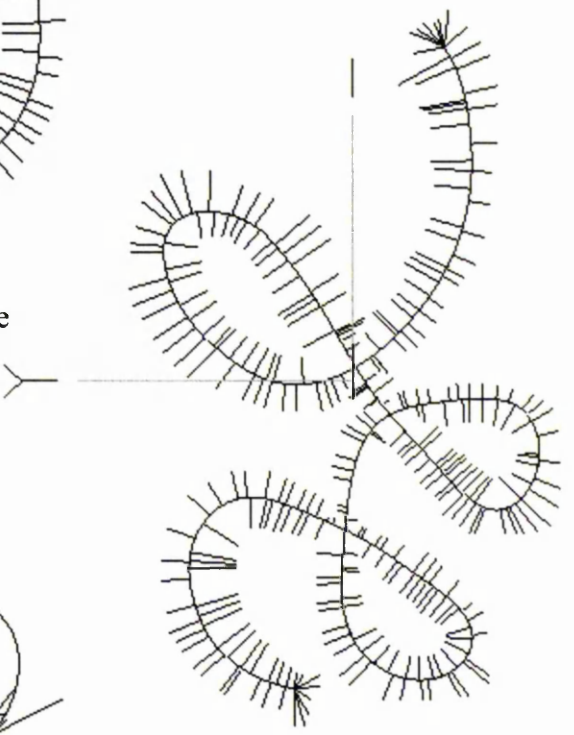
4.10 Examples

For illustrative purposes we present three pictures of points projected to a 2D cubic NURBS curves (figure 4.11(a)), a 2D NURBS curve of degree 5 (figure 4.11(b)) and a 3D cubic NURBS curve (figure 4.11(c)) respectively.

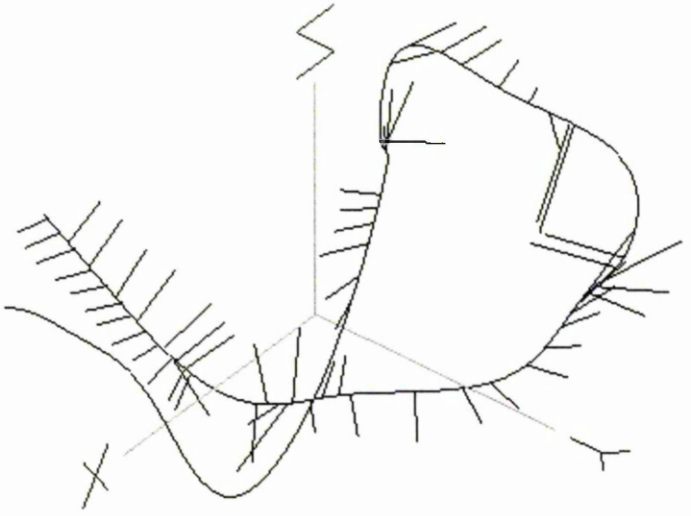
We also present some pictures of points projected to the NURBS surface. Figure 4.12(a) gives the result of projecting a set of points from a straight line to the NURBS surface. Figure 4.12(b), (c) present the results of projecting two isocurves which are created and offset from the NURBS surface in the U direction and V direction respectively. Figure 4.12(d) gives the result of projecting a set of points from a vertical line to the NURBS surface. Finally, Figure 4.12(e) presents the result of projecting a set of points to a closed NURBS surface.



(a) Point projection for a 2D cubic NURBS curve
(degree =3, 26 control points)

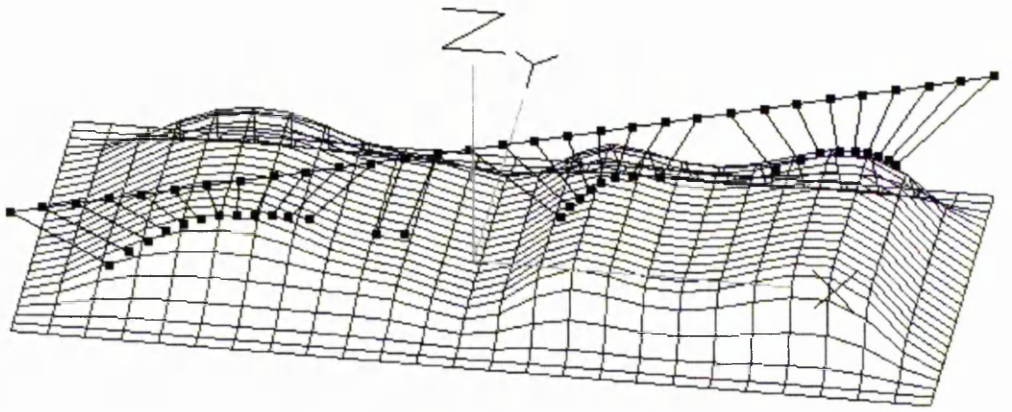


(b) Point projection for a 2D cubic NURBS curve
(degree =5, 40 control points)

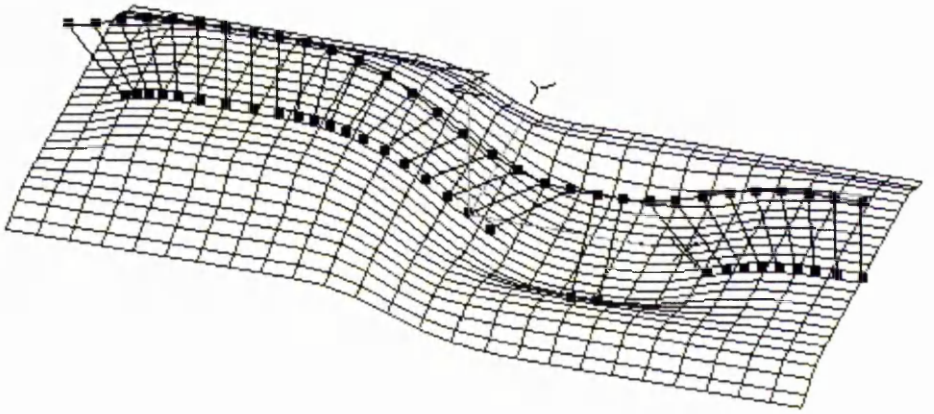


(c) Point projection for a 3D NURBS curve.
(degree =3, 20 control points)

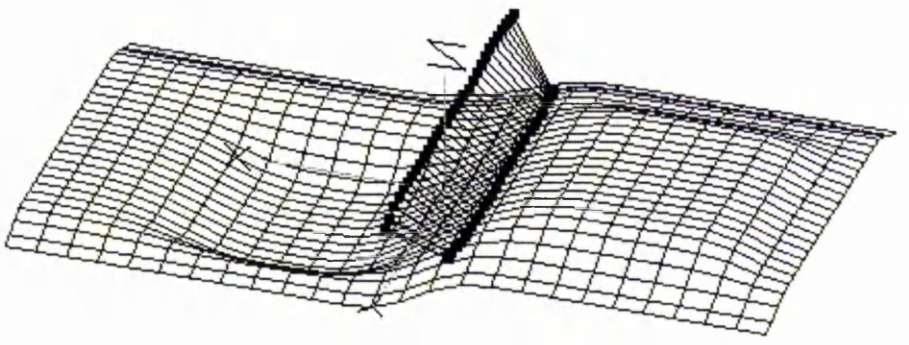
Figure 4.11: Point Projection for NURBS Curves



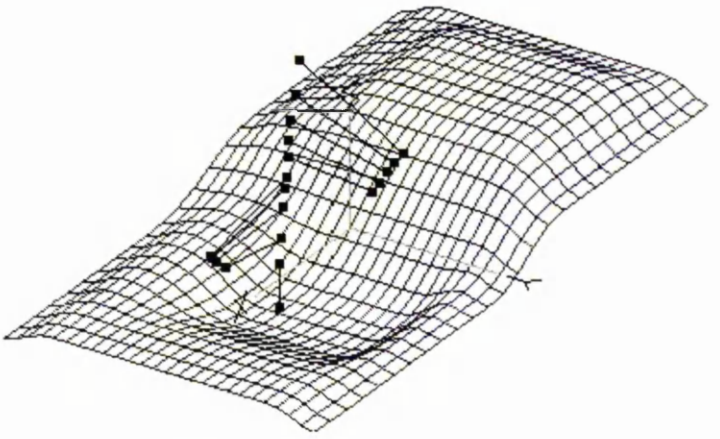
(a) straight line projection



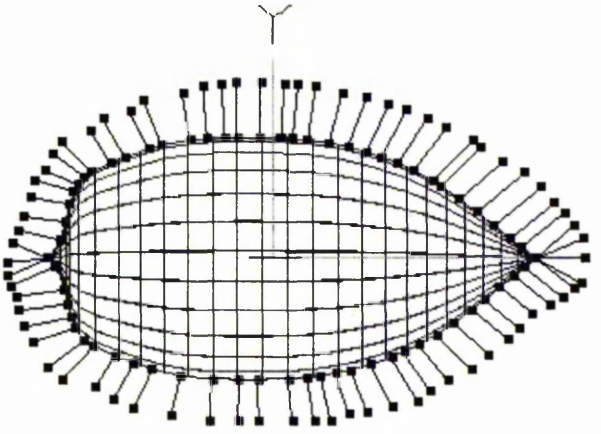
(b) Isocurve projection (U direction)



(c) Isocurve projection (V direction)



(d) Vertical line projection



(e) Point projection for Closed NURBS surface

Figure 4.12: Point Projection for NURBS Surfaces

4.11 Comparison

We give the results of comparing our method with Piegl and Tiller's method [17] for point projection on the NURBS curves both in efficiency and accuracy. Piegl and Tiller's method first evaluates curve points at n equally spaced parameter values for the whole NURBS curve. Then they compute the distance

of each point from the test point P and choose the parameter value u_0 to be the value yielding the point closest to the test point P. Finally, they apply the Newton-Raphson method to the closest point to improve the its accuracy. The accuracy of the result can be evaluated by equation (4.3) (zero cosine).

$$\frac{|C'(u_i)\cdot(C(u_i)-P)|}{|C'(u_i)||C(u_i)-P|}\le\varepsilon \qquad (4.3)$$

where $C(u)$ is the NURBS curve, P is the test point and ε is the tolerance for zero cosine.

All examples are implemented with Visual C++ 6.0 with Windows98 in the same compatible personal computer with Pentium II 400 CPU chip and 64 M memory. Table 4.1 lists iteration times of our method, the number of equally spaced curve parameters for Piegl and Tiller's method (n), the CPU time and the accuracy.

- Example 1 2D Curve in figure 4.11(a) (degree =3, 26 control points)

Index	t (CPU time ms)	Iteration time / n	Accuracy (zero cosine)
Our Method (1)	3.003	59	2.083230e-07
NLib (1)	3.252	100	1.1625887e-07
Our Method (2)	3.491	69	1.13872e-012
NLib (2)	3.481	100	2.560200e-012
Our Method (3)	2.517	61	3.626256e-09
Nlib (3)	3.172	100	5.1356532e-09

- Example 2 2D curve in figure 4.11(b) (degree = 5, 40 control points)

Index	t (CPU time ms)	Iteration time / n	Accuracy (zero cosine)
Our Method (1)	5.195	89	4.6637136e-08
NLib (1)	5.627	200	1.319305e-011
Our Method (2)	4.558	77	1.4516641e-08
NLib (2)	5.701	200	2.693517e-011
Our Method (3)	4.356	87	1.507982e-010
Nlib (3)	5.256	200	4.032916e-08

- Example 3 3D curve in figure 4.11(c) (degree = 3, 20 control points)

Index	t (CPU time ms)	Iteration time / n	Accuracy (zero cosine)
Our Method (1)	2.358	53	9.8645779e-08
NLib (1)	2.643	80	1.2507234e-09
Our Method (2)	2.359	55	1.2857904e-08
NLib (2)	2.892	80	1.0723880e-08
Our Method (3)	3.806	61	2.609440e-013
Nlib (3)	2.997	80	1.187688e-012

- Example 4 complex 2D curve in figure 4.2 (degree = 3, 73 control points)

Index	t (CPU time ms)	Iteration time / n	Accuracy (zero cosine)
Our Method (1)	12.779	239	1.313991e-010
NLib (1)	22.781	1000	4.4269524e-09
Our Method (2)	12.276	229	1.292864e-010
NLib (2)	21.762	1000	5.0203867e-07
Our Method (3)	19.974	307	2.8378485e-09
Nlib (3)	21.446	1000	5.3175793e-07

4.12 Point Inversion

The point inversion problem is very similar to point projection. The only difference is that the candidate Bézier subcurves or patches are extracted based upon the strong convex hull property [17]. Then, for a curve, we apply algorithm 5 to these candidate Bézier subcurves to extract the closest point on the NURBS curve and its corresponding curve parameter values. For a surface, we apply algorithm 6 to these candidate Bézier patches to extract the closest point on the NURBS surface and its corresponding surface parameter values.

4.13 Conclusion

In this chapter we have presented a novel method to solve both point projection and point inversion problems. This method achieves better results both in accuracy and efficiency, especially in dealing with complex NURBS curves. Furthermore, this method extracts all candidate points for point projection so that it avoids selecting the wrong initial value for the Newton-Raphson method in the self-intersecting curve case. It also provides a good initial value to achieve reliable convergence for the Newton-Raphson method.

On the other hand, for NURBS surfaces, this method dramatically decreases the computation of the algorithm compared with the method [57], which decomposes the NURBS into a set of quadrilaterals. We also apply the Newton-Raphson method on the Bézier patch instead of the whole NURBS surface, which improve the stability of the algorithm.

Chapter 5

Adaptive Tessellation

5.1 Previous work

A brief introduction about trimmed NURBS surfaces has been given in section 2.5.1 and also a short introduction to adaptive tessellation methods has been presented in section 2.5.2. In this section, we give an overview of previous work done in the areas of tessellating trimmed NURBS surfaces.

5.1.1 Adaptive Forward Differencing

Shantz and Chang [58] describe a direct hardware rendering technique for a trimmed surface based on the adaptive forward differencing (AFD) method. Similar to the scanline algorithm for rendering a polygon, this method is suitable for special graphics VLSI. However, it is not practical for implementation on variant graphic hardwares, because this technique directly renders surface without going through the intermediate form of triangles. Furthermore, the computation becomes very expensive, when this method

subdivides the surface down to pixel size in order to render a high quality image. The method operates in 5 steps:

1. The NURBS trimming curves are converted to piecewise Bézier curves by knot insertion.
2. The Bézier curves are subdivided so they are monotony in the u parameter direction.
3. The Bézier curves are converted to forward a difference basis.
4. They are sorted in u parameter order by their minimum u value.
5. For each AFD forward step in the u direction (from curve to curve) the active trimming curve sections are forward-stepped down to find intersections with the new curve. The appropriate portions of the surface curve are drawn based on the trim curve-winding rule.

5.1.2 Tessellation Under Highly Varying Transformation

Salim S. Abi-Ezzi and Leon A. Shirman [59] provide a **dynamic** and **uniform** tessellation method for arbitrary degree polynomial and rational Bézier patches. NURBS surfaces are converted into Bézier patches before applying this method. They designed two approximation criteria: a size criterion which uses a threshold on the size of triangles, and a deviation criterion which uses a threshold on the deviation of these triangles from the actual surface. This method involves highly varying the modelling and viewing transformations [59] (between world coordinates and display coordinates) and performing the complex operations of finding derivative bounds, computing norms of transformations, and factoring of views at data creation time. Therefore, it is expensive and not practical for high degree Bézier patches because it will generate more triangles for higher degree surfaces. It is obvious that the uniform tessellation generates more triangles than the non-uniform method.

5.1.3 Fast Dynamic Tessellation of Trimmed NURBS Surface

Salim S. Abi-Ezzi and Srikanth Subramaniam [60] present a **dynamic** and **non-uniform** tessellation method developed from the previous method (section 5.1.2). Similar to the previous one, it converts the NURBS surface into Bézier patches and the Bézier control points are used for further computations. Then the trimming patches are further simplified into monotonic regions, which contain several trimming curves. The next step contains two phases of traversal.

The first phase reduces each trimming NURBS loop into its Bézier components, then processes each trimming curve to determine the maximum and minimum value of U and V on each trimming segment, then computes its intersection with the U/V knot line. The intersection problem can be solved by using a Bézier root-solving algorithm. Finally, it handles some special cases to ensure the stability of the algorithm.

The aim of the second phase is to extract the triangles from both trimmed and untrimmed patches. For untrimmed patches, triangles are generated from two U/V isolines. For trimmed patches, triangles are generated from the U/V isolines and trimming Bézier curves.

5.1.4 Triangulating Trimmed Surfaces for Stereolithography Applications

Sheng and Hirsh [61] presented a method for triangulation of trimmed surfaces in parameter space. This approach first maps the trimmed regions of the surface into parametric space and the trimmed regions are approximated by 2D polygon regions, which are then triangulated by a restricted Delaunay triangulation algorithm. The generated triangles are subdivided further until each edge of the triangles is smaller than the allowed length that results from

the surface definition and the specified tolerance. The detailed algorithm is described as follows:

1. Creation of mapping polygons

To generate the mapping polygons in parametric space, the surface is first split into patches along their common boundaries (see figure 5.1). After splitting, the boundary of a mapping polygon consists of splitting lines and trimming curves. The mapping polygons are connected by these splitting lines and trimming curves. To form the ordinary boundary of a polygon, each trimming curve is subdivided into line segments for a given tolerance.

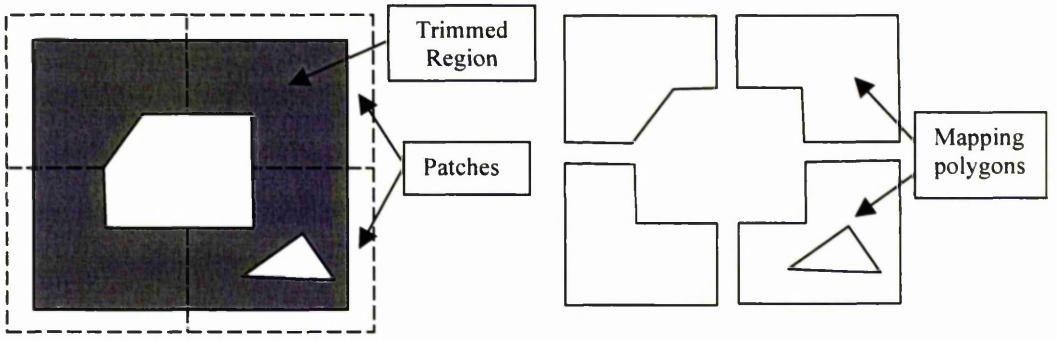


Figure 5.1: Generation of mapping polygons by splitting of trimmed region

2. Evaluation of the flatness of a patch

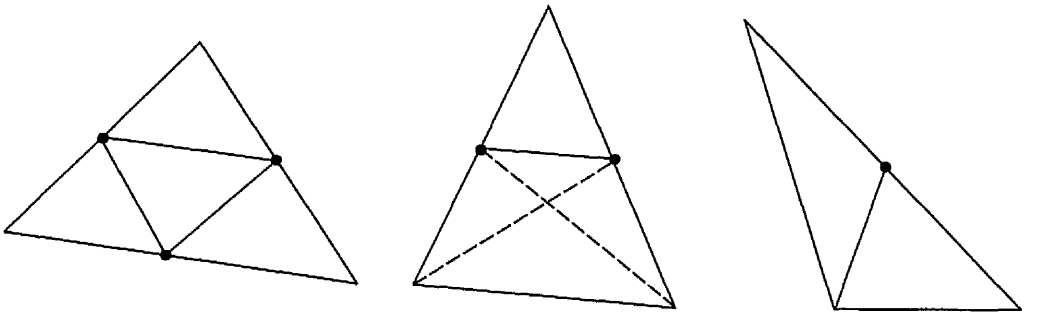
In their method, only the triangulation of the surface is considered, because it is the only one arising in their applications. The evaluation can be formulated as follows: given a parametric patch of a C^2 surface and an arbitrary triangle with its vertices on the surface, determine the maximum distance between the surface and triangle with the same parametric bound. If the distance is smaller than a user-specified tolerance, the patch is said to be sufficiently flat. The goal of this operation is the condition for the termination of subdivision of triangulated patches. Therefore, the triangles that are generated from the

restricted Delaunay triangulation are subdivided further until all the edges fall within the allowed bounds. The allowed bound is defined as: Let Ω be the maximal edge length of the triangle, the distances between the centre of the triangle and its vertices are not greater than (2/3) Ω [62]. If (2/3) Ω is less than the user-specified tolerance, they say the triangulated patch is flat enough.

3. Restricted Delaunay Triangulation

Once, the mapping polygon for each patch has been established, it can be triangulated separately. The goal of the triangulation is to obtain triangles whose edge lengths do not exceed the allowed length Ω determined in the previous section. To minimize the number of triangles produced, it is obviously desirable that each edge length of the triangles be as close to Ω as possible (an equilateral triangle is optimal).

On the basis of the principle of the restricted Delaunay triangulation, each mapping polygon of the patches is first split into a set of triangles. By measuring the edge lengths, one can decide whether a triangle has to be further subdivided. If the edges of the triangle exceed the allowed bound Ω, a refinement procedure divides the triangle into two, three or four subtriangles at the middle points of the edges. The procedure is shown in figure 5.2. While the subdivision in figure 5.2(a) and 5.2(c) are unique, one has to select one of the two possibilities for figure 5.2(b). The refinement procedure runs recursively until the edge lengths of all generated triangle fall short of the maximum.



(a) All three edges larger than Ω. (b) Two edges larger than Ω. (c) One edge larger than Ω.

Figure 5.2: Subdivision of triangles.

5.1.5 Triangulating The Trimmed NURBS Surface in Parameter Domain

Piegl and Richard [63] propose a somewhat similar algorithm to triangulate a trimmed NURBS surface: they use the same criterion for maximum edge length, but the method does not split the surface into several regions representing Bézier patches in parameter space. This tessellation method consists of 5 steps. Step 1 is to compute the longest edge size in the parameter domain. Step 2 obtains a polygonal approximation of the trimming curves. Step 3 selects points inside the valid region. Step 4 triangulates the trimmed region. The final step is to map the triangles onto the surface and build a 3D triangular database for further processing. The details of the algorithm are given as follows.

1. **Computing the longest edge size**

   The 3D triangles, obtained by mapping the 2D ones onto the surface, deviate from the surface by less than ε, where ε is a user specified tolerance. The edge length can be calculated from the upper bounds of the second derivatives, computed over the entire patch [63]. This way of

the calculation of edge length is more adaptive than Sheng and Hirsh 's method [64] (Directional adaptive).

2. Polygonal approximation of trimming curve

In the previous section, they calculate a maximum edge length λ such that triangles in the parameter domain with sides less than λ map onto 3D triangles that are within ε distance from the surface. The task now is to subdivide the trimming curves into polylines, such that no edge is longer than λ.

3. Selection of points inside trimmed region

The selection of points inside the trimmed region is done via a simple scanline-type algorithm used in raster graphics to fill a polygon. Once the trimming curves are approximated, points inside the trimmed region are selected in a similar way to pixel selection in the polygon fill algorithm [65].

4. Triangulating the trimmed region

This step is tied to a specific data structure to store points as well as the boundary edges of the trimming curves. This structure is obtained by putting a uniform grid over the points and processing each point and edge into a grid cell. Each cell then has a list of points lying inside the cell, and a list of edges intersecting the cell. By applying a Delaunay triangulation algorithm, the points in each cell are transformed into a set of Delaunay triangles.

5. Mapping triangles onto the trimmed surface

Once the triangulation of the trimmed domain is completed, the domain triangles are mapped onto the surface forming a tessellation. Although this is a straightforward map, a data structure has to be maintained so that triangles can be passed onto a postprocessor, such as a contouring program or a shader based on polygonal objects.

5.1.6 Summary

The first method is designed for the special graphic hardware for rendering triangles which are generated from the AFD method. It works for cubic Bézier surfaces only and it is not practical for tessellating high degree surfaces. The second method is a uniform tessellation that generates more triangles than non-uniform methods.

The other three methods perform tessellation in parametric space. The first method is both dynamic and non-uniform tessellation. However, it involves some complex algorithms which could reduce the stability of the whole method. The second method takes special care of the edges of the solid that is being subdivided, and guarantees the absence of cracks. This method has two main disadvantages: it is not adaptive (global bounds for second derivatives are found for every patch), and second, it does not care about the shape of the resulting triangles. The consequence of not being adaptive is that the number of triangulation vertices is too large. The third method is adaptive and not sensitive to the complexity of the trimmed patch. Unlike the first method, it calculates the bound for second derivatives locally, therefore it achieve more efficient flatness testing.

All three methods have common advantages and disadvantages as general tessellation methods in the parametric domain.
Advantages:

- Methods that operate on triangles are far easier and numerically more stable than those dealing with freeform geometry.
- The piecewise triangular approximation is a parameter independent representation of the trimmed surface.

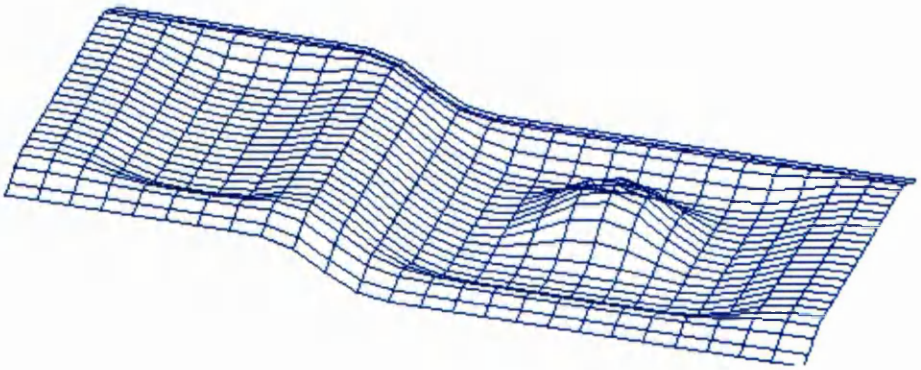
Disadvantages:

- Adequate representation of a trimmed patch with high curvature areas requires large numbers of triangles.
- The triangulation, if not done properly, can result in triangles of different sizes, and, in particular, in long and skinny triangles which, in turn, can cause numerical problems.

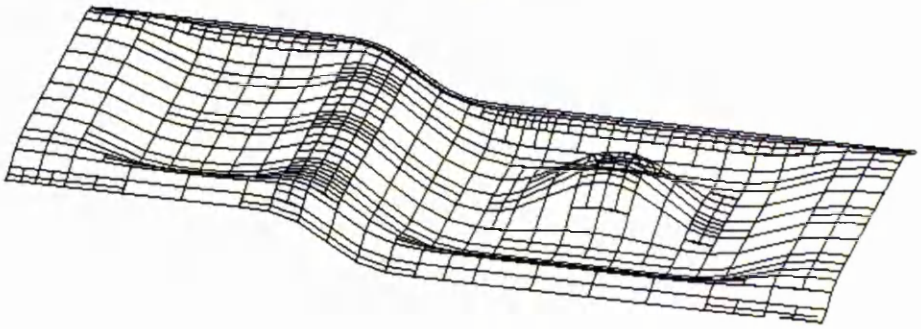
5.2 New Approach

Like the two tessellation methods that are discussed in the previous section, our method has most of the advantages that the other tessellation methods in the parametric domain have. Our new approach differs in the way it checks the flatness of the desired patches, and its way of subdividing the surface. The subdivision and flatness checking have the same methodology as the solution to the problem of point projection for NURBS curves and surfaces. The tessellation is based on the individual Bézier patch that is flat enough. The whole algorithm consists of the following steps:

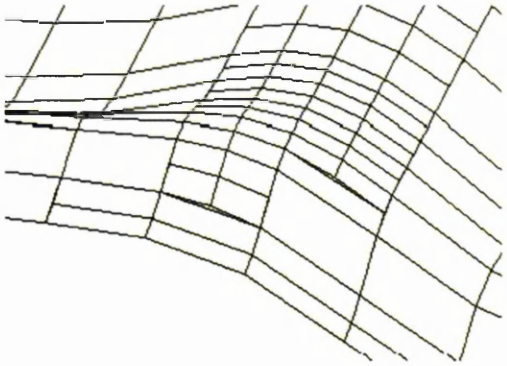
- Finding the bounding box for outer trimming loops and spitting the surface to fit the bounding box. The splitting method uses knot refinement.
- Subdividing the surface into a set of Bézier patches which are flat enough.
- Removing the patches outside the boundary of the outer trimming loop and removing the patches inside the boundary of the inner trimming loops.
- Closing the outer and inner boundary with a set of triangles.

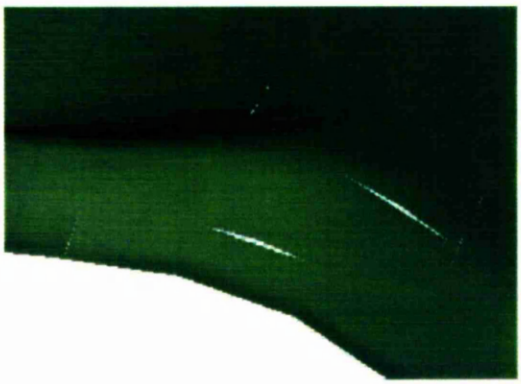


(a) The original NURBS surface.



(b) "flat enough" Bézier patches.





(c) The holes between the Bézier patches. (d) Rendering picture for the holes and patches.

Figure 5.3: Tessellating the untrimmed NURBS surface

5.2.1 Tessellating the untrimmed NURBS surface

Before we start to tessellate the trimmed NURBS surface, we give an introduction to the tessellation of an untrimmed NURBS surface, which is based on the methodology of subdividing the NURBS surface into a set of "flat enough" Bézier patches. If we only accept the set of "flat enough" Bézier patches as the result of tessellation, some holes will appear between patches due to the approximation of a patch boundary by a straight line. An example of this degenerative process is shown in figure 5.3.

The solution for this problem is to use triangles to approximate the Bézier patches if it has midpoints along its boundary. We designed an easy way to generate triangles from the Bézier patches, which have at least one midpoint (See figure 5.4). As the exception, a patch with one midpoint generates three triangles. Generally, the patch with n midpoints generates (n+4) triangles. Finally, figures 5.5 and 5.6 show the tessellation using this solution.

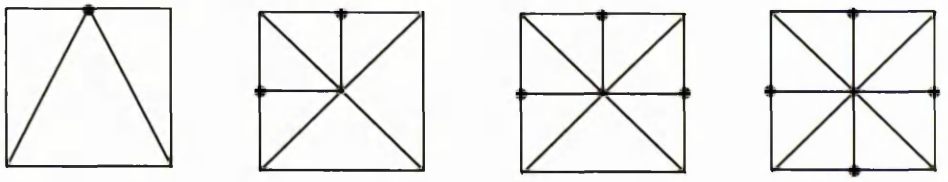


Figure 5.4: Generating triangles from the Bézier patch.

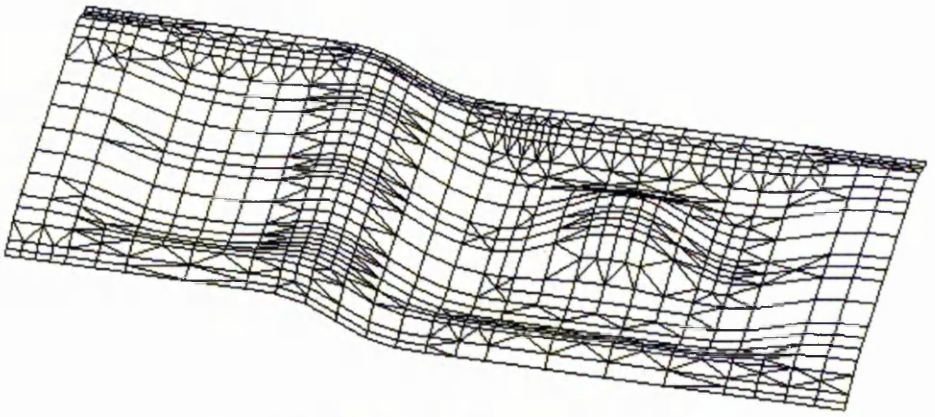


Figure 5.5: The wire frame of tessellating result.

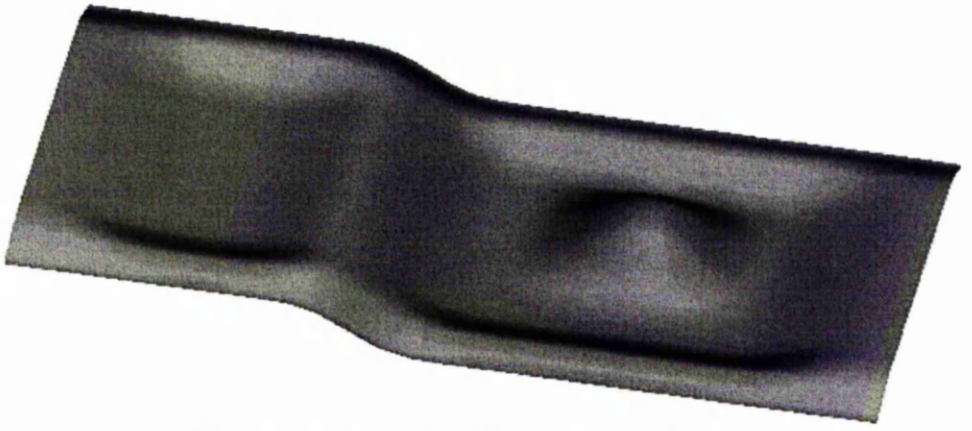


Figure 5.6: Rendered picture of tessellating result.

5.2.2 Finding The Bounding Box and Splitting The Surface

From the definition of trimmed NURBS surface, we know that the trimmed surface has one outer trimming loop and several inner trimming loops. For the outer trimming loop, we define the outer bounding box in the parametric domain as the box containing all trimming curves in the outer trimming loop. On the other hand, the inner bounding box is the box containing all trimming curves in one inner trimming loop. As shown in figure 5.7, the solid-line rectangle is the outer bounding box and the dash-line one is the inner bounding box.

After obtaining the outer bounding box, we split the surface in both U, V directions in the parametric domain to make the remain surface fitting with the outer bounding box (showing in figure 5.8).

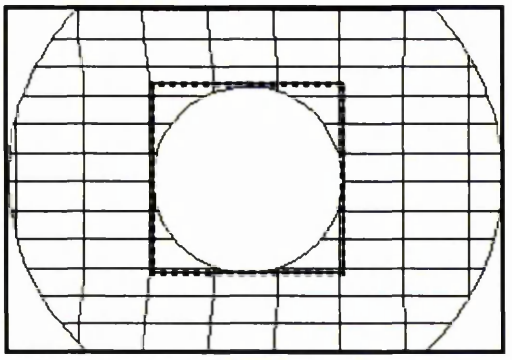


Figure 5.7: Bounding Boxes

(a) Original NURBS surface.

(b) The surface trimmed with the outer bounding box

Figure 5.8: Splitting surface to fit with the bounding box.

5.2.3 Removing The Patches

After we obtain the surface that fits within the outer bounding box, we tessellate it as an untrimmed NURBS surface, which is described in section 5.2.1. We now have a set of "flat enough" Bézier patches. By mapping these Bézier patches into parametric space (U, V), we get a set of small rectangles instead. We also have all the trimming loops in parametric space and separate them as one outer trimming loop and several inner trimming loops. By

applying the scanline algorithm [66][67][68][69], we remove the patches outside the outer trimming loop and inside the inner trimming loops. At the same time, we build a point array for recording all boundary points along inner and outer boundaries. These points are used to generate the triangles for closing the boundaries.

1. **Scanline Algorithm**

   The scanline algorithm provides the tools to determine whether a Bézier patch is inside the trimming loops, intersects with trimming curves or outside the trimming loops. As shown in Figure 5.9, uniformly distributed scanlines are placed in the U direction of parameter space and the density of U scanlines is determined by the tolerance of the tessellation. The Bézier patches are created between the two neighbourhood scanlines. Each scanline may have an odd or even number of intersection points with trimming loops. If we get an odd number of intersection points, we can repeat the tangent point to generate the even number of intersection points (No. 4 scanline in figure 5.9). According to the sequence of the intersection points, we can divide the region into a positive one and a negative one. The positive one is inside the trimming loops and the negative one is outside the loops (figure 5.10). Finally, we can compare the Bézier patch with the positive and negative regions, remove the patches inside the loops, and create a patch to fit with the loop boundary if the patch intersects with a loop.

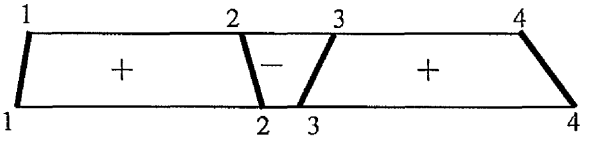


Figure 5.10: Positive and negative regions

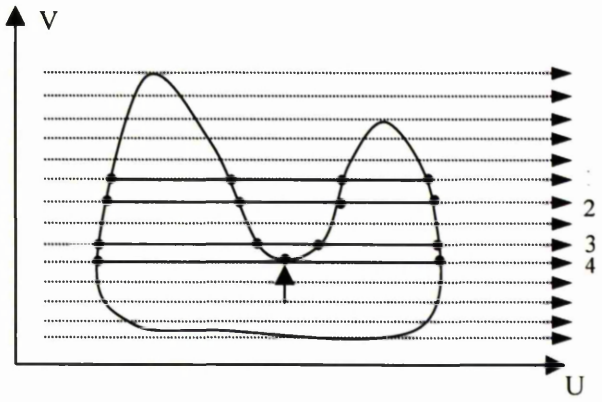


Figure 5.9: U Scanline

2. Summary

In summary, we give the algorithm for detecting the Bézier patches outside the outer boundary.

Algorithm1 *Bézier_Patch_Outside_Outer_Boundary*
Input: *A Bézier patch and outer trimming loop.*
Output: *the result of detection.*
Begin
{m is the highest index of outer trimming curves}
for *i = 0 to i < m by i++ do*
{Detect whether outer trimming curve intersect with the Bézier patch}
begin
generate a polyline to approximate the trimming curve;
if *(the Bézier patch intersects the polyline) then*
Generate a new patch which fit with the outer boundary;
Add the boundary points into array;
return *the result of intersection and the new patch;*
end if
end *{End of loop for}*

Flag ← FALSE; {Flag TRUE: inside; FALSE: outside}
for *i = 0 to i < m by i++ do*
{Detect whether the patch is inside the outer trimming loop or not.}
begin
generate a polyline to approximate the trimming curve;
if *(the Bézier patch is inside the trimming loop)* ***then***
Flag ←TRUE;
end if
end *{End of loop for}*

if *Flag == TRUE* ***then***
return *the patch inside the outer trimming loop;*
else

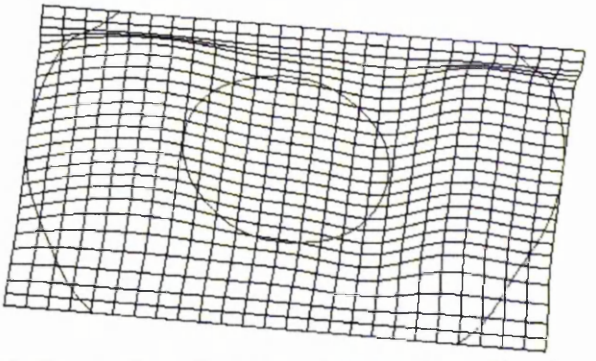
```
  return the patch outside the outer trimming loop; {The patch will be
removed;}
End of Algorithm 1
```

We have a similar algorithm to detect the Bézier patches inside the inner boundary.

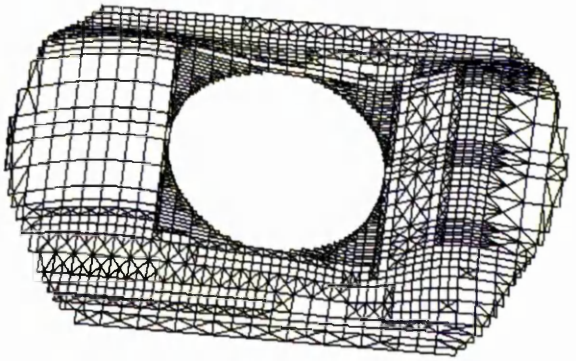
```
Algorithm 2 Bézier_Patch_Inside_inner_Boundary
Input: A Bézier patch and inner trimming loops.
Output: the result of detection.
Begin
   {m is the highest index of inner trimming loops}
  for  i = 0 to i < m by i++ do
       {Detect whether the Bézier patch is inside one of the inner trimming
loops}
    begin
      generate a polygon to approximate the trimming loop;
      if  (the Bézier patch is inside the polygon) then
           return the patch is inside the polygon; {The patch will be removed;}
     else if (the Bézier patch intersects with polygon) then
           generate a new patch fitting with the inner boundary;
           add the boundary points into array;
          return the result of intersection and the new patch;
     end if
  end {End of loop for}

  return the patch outside the inner trimming loops;
 End of Algorithm 2
```

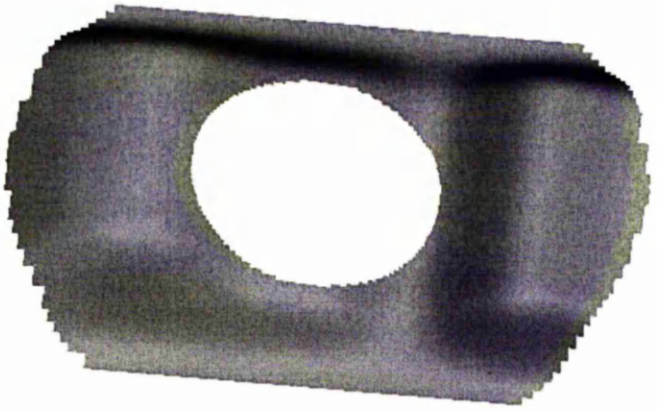
Finally, figure 5.10 gives the illustration of this procedure.



(a) The surface fitting with outer bounding box.



(b) The tessellation result with removing all patches outside the outer boundary and inside the inner boundary.

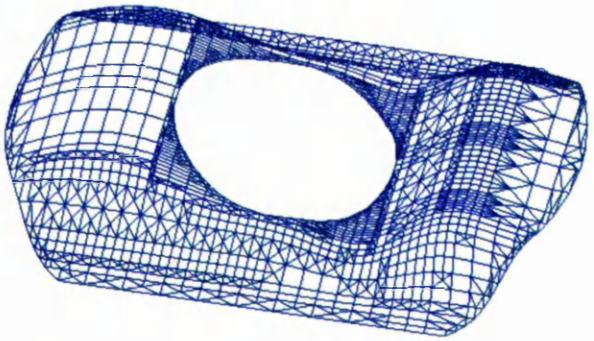


(c) Rendered picture

Figure 5.10: The procedure of removing patches

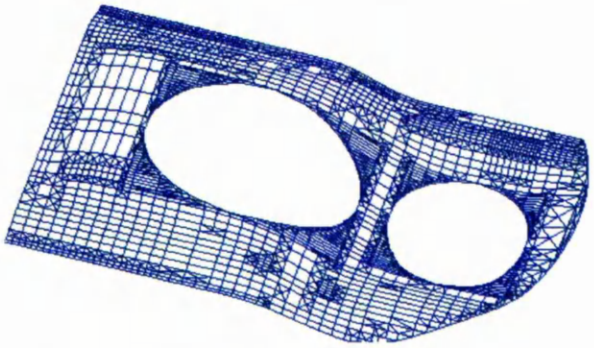
5.2.4 Closing the Outer and Inner Boundary with a Set of Triangles

The aim of this step is to generate smooth boundaries for both the outer and inner trimming loops. The point array that is generated in the last section is used to generate the triangles through the neighbouring points. To obtain the correct rendering effect, we need to set the points in anti-clockwise or clockwise direction in all triangles. Figure 5.11 gives the final result both in wire frame and rendering mode.



(a) Wire frame picture of final result 1.

(b) Rendered picture of final result 1.



(c) Wire frame picture of final result 2.

(d) Rendered picture of final result 2.

Figure 5.11: Final results of tessellation

5.2.5 Summary of the Algorithm

The summary of the whole algorithm is given by the following the pseudo code.

Algorithm 3 *Tessellation_Trimmed_NURBS_Surface*
Input: *A trimmed NURBS surface*
Output: *a set of quadrilaterals and triangles*
Begin
Get the outer bounding box;
Split the NURBS surface to fit with outer bounding box in parametric space;
Tessellate the surface as an untrimmed NURBS surface; {The result is a set of Bézier patches}
{m is the highest index of the Bézier patches.}
for *i = 0 to i < m by i++ do*
{Detect whether to remove, generate a new patch or keep the patch.}
begin
if *(Bézier_Patch_Outside_Outer_Boundary* ***return*** *intersection)* ***then***
Add the new patch into patch array;
else if *(Bézier_Patch_Outside_Outer_Boundary* ***return*** *inside)* ***then***
Add the original patch into patch array;
end if

{n is the highest index of inner trimming loops.}
for *j=0 to j<n by j++ do*
begin
if*(Bézier_Patch_Inside_inner_Boundary* ***return*** *intersection)* ***then***
Add the new patch into patch array;
else if *(Bézier_Patch_Inside_inner_Boundary* ***return*** *outside)* ***then***
Add the original patch into patch array;
end if
end *{End of loop for}*

end *{End of loop for}*

Generate quadrilaterals and triangles from the patch array;
Generate triangles from the boundary point array;
End *of Algorithm 3*

5.3 Conclusions

In this chapter, we have presented an algorithm for tessellating a trimmed NURBS surface in the parametric domain. Based on the flatness test, the method stops the subdivision of the surface and obtains a tessellation within a

user specified tolerance. The tessellation is performed completely in parametric space, and furthermore this method does not adopt any complex method to generate triangles, so that the procedure runs fast and reliably.

One drawback is that the surface subdivision involves high computation which reduces the efficiency of the whole algorithm. The subdivision techniques using knot insertion have been described by Boehm [70] and Cohen and others (the Oslo algorithm) [71]. The efficiency can be improved by computing only part of the control net in the knot insertion algorithm to speed up the procedure of the flatness test for control net of the Bézier patch.

Chapter 6

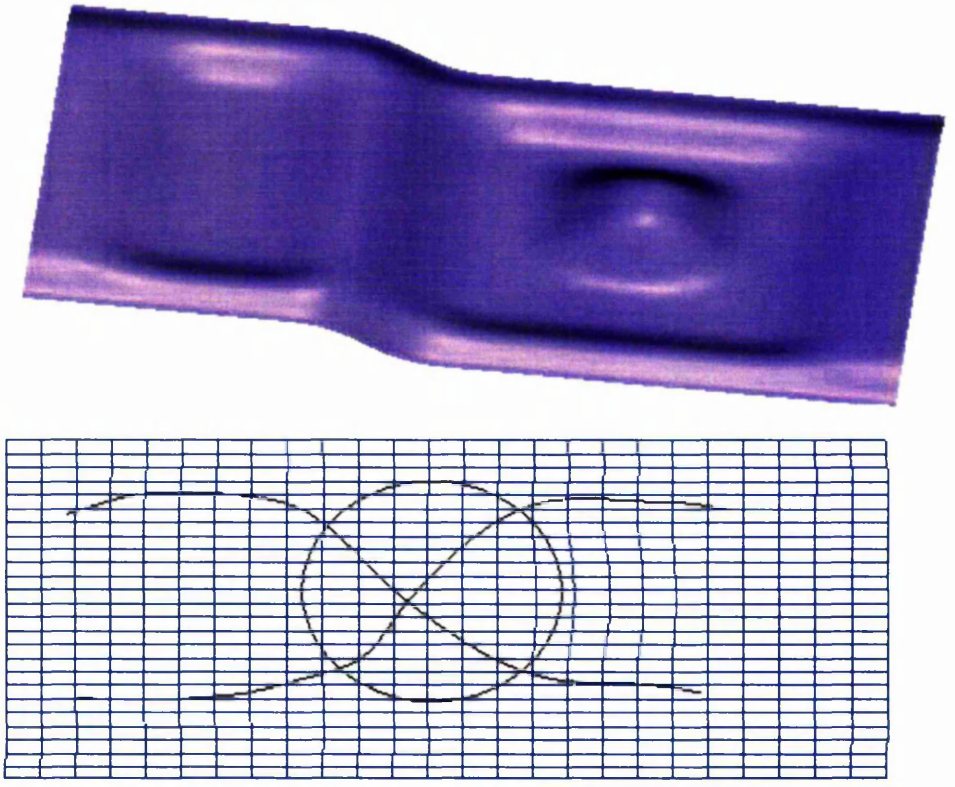
Deforming B-rep Model

6.1 Deformation on a single untrimmed NURBS surface

As described in chapter 3, we apply the metaball deformation model on a single NURBS surface. By moving the control points on the refined control point net of the NURBS surface, we can deform single or multiple general constraints on the surface. Figure 6.1 shows an example of deforming "MVC" on the NURBS surfaces.

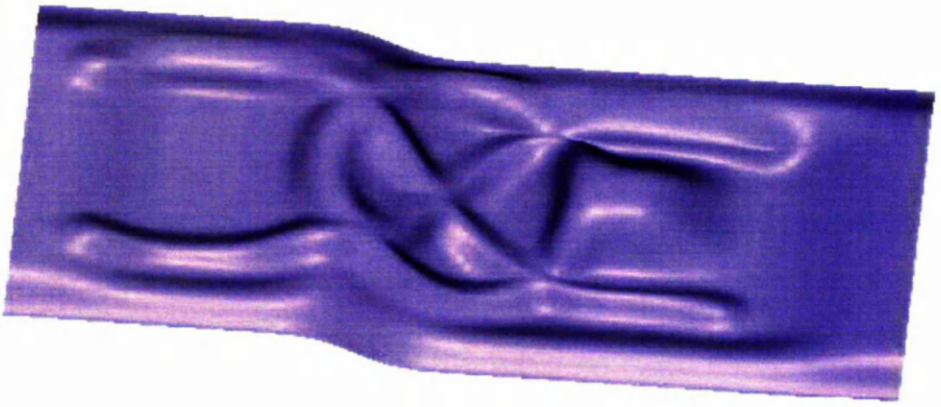
Figure 6.1: deforming "MVC" on the NURBS surfaces.

There are two ways for applying multiple constraints on a single NURBS surface. Method one is to apply them individually so that the constraints will overlap each other. Method two is to calculate the minimum distance for all constraints and apply them together as a group on the surface. Figure 6.2 gives an illustration of deforming multiple constraints on single surface.



(a) Original NURBS Surface and general curve constraints.

(b) Applying method one on the surface.



(c) Applying method two on the surface.

Figure 6.2: Apply different constrained deformation methods on the surface

From figure 6.2(b) and figure 6.3(c), we can see that method one accumulates the displacements for individual curve constraints. On the other hand, method two calculates the distances for all curves instead of just one curve and selects the minimum one which generates the displacement.

6.2 Deformation on a trimmed NURBS surface

According to the definition of the trimmed NURBS surface described in section 2.5.1, we give the data structure of the trimmed surface in a C++ style.

```
class TrimNURBSSurface
{
  NURBSSurface m_Surf;  //untrimmed NURBS surface.
  NURBSCurveArray m_CurArr; //trimming curve array in parameter space.
  TrimmingLoopArray m_LoopArr; //trimming loop array (Both inner and outer).
};
```

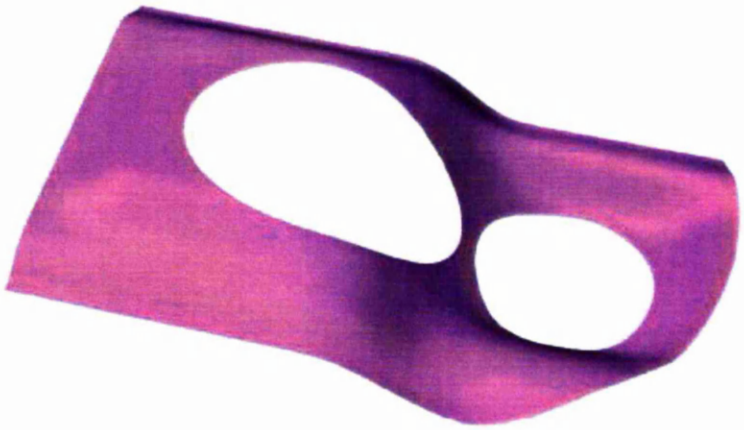
The data structure of *TrimmingLoop* is defined as:

```
Class TrimmingLoop
{
  IntArray m_indexes;    //curves' index for each loop.
  BOOL   m_type;         //TRUE for outer loop; FALSE for inner loop.
};
```

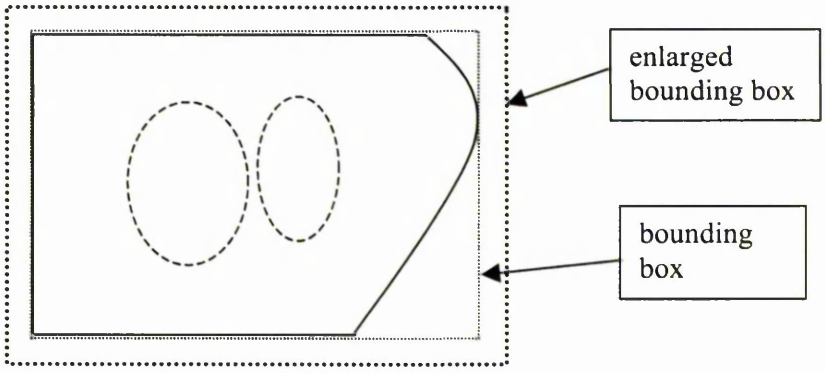
We keep trimming curves (both inner trimming loops and outer trimming loops) unchanged on the parametric domain when deforming single or multiple general constraints on the trimmed NURBS surface. However, the shape of trimming curves in 3D space could be changed after the deformations. In our prototype system, deformation on a trimmed NURBS surface is divided into three steps.

1. **Extract outer trimming loops and calculate its bounding box**

   We only move the control points which are inside the enlarged outer bounding box. The enlarged outer bounding box is box which is offset toward the outer direction by a distance of the effective radius in the metaball model. The goal of only moving points inside the enlarged outer bounding box is to reduce the computation of metaball deformations. Figure 6.3 gives an illustration. The dash-line curves are the inner trimming loops and the solid-line curves are the outer trimming loop. The square-dot rectangle is the bounding box for outer trimming loops and the round-dot one is the enlarged bounding box.



(a) A trimmed NURBS surface.



(b) Outer bounding box and its enlarged bounding box (parametric space).

Figure 6.3: Trimmed NURBS surface and its trimming loops

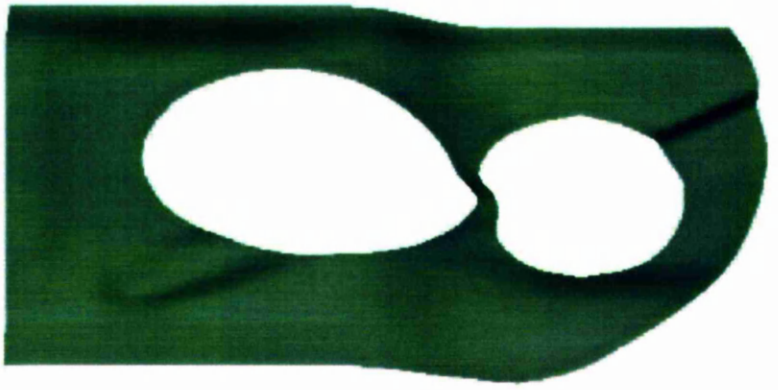
2. Deforming the trimmed NURBS surface

Similarly to deforming the untrimmed surface, we deform the untrimmed NURBS surface (*m_Surf*) in the data structure of the trimmed surface by moving the control points inside the enlarged bounding box.

3. Tessellating the deformed and trimmed NURBS surface

During the deformation process, we only change the untrimmed NURBS surface and keep other data numbers unchanged. Using the tessellation method describe in chapter 5, we subdivide the deformed surface, remove unwanted Bézier patches and close the boundary with a set of triangles. The final result is given in figure 6.4 (using the same surface as figure 6.3).

(a) Deformation on an untrimmed surface



(b) Deformation on a trimmed surface

Figure 6.4: Deforming the trimmed NURBS surface by using metaball model

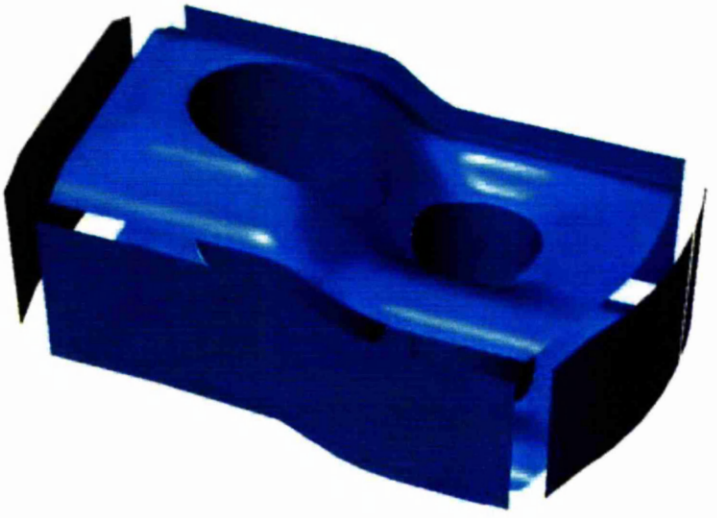
6.3 Deformation on a B-rep Model

A NURBS B-rep model contains several trimmed NURBS surfaces which form the faces of the B-rep model. Each face has several edge curves connecting with other faces. In this thesis, we only discuss the method of deforming one face of a B-rep model. Deforming two or more faces at the same time is much more complicated and is left for future discussion.

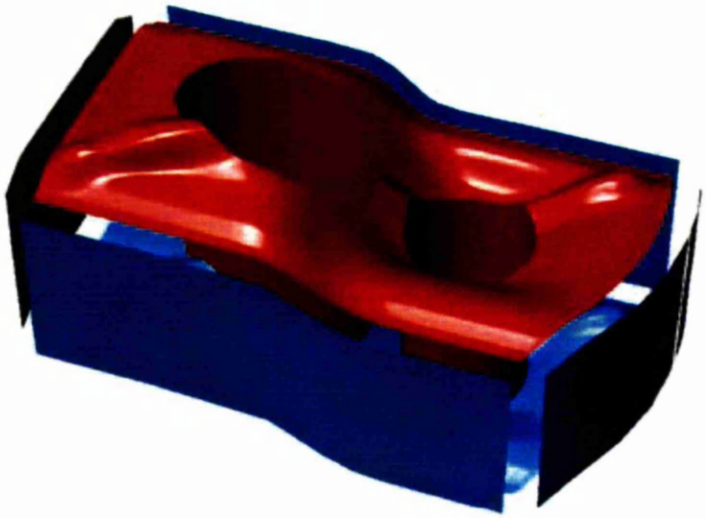
Deforming a face on the B-rep model may change one or more edges on this face. That leads to changes to other faces which are associated with these

edges. There are three methods to solve the modification of other faces in a B-rep model.

Method one first maps these edges (trimming curves) into the parametric space of individual faces (trimmed NURBS surfaces). The mapping involves converting 3D NURBS curves into 2D NURBS curves in the parameter domain. Point inversion for a NURBS surface, described in chapter 5, can find the parameter values (u, v) for reconstructing the 2D trimming NURBS curve in parametric space. Then we regenerate the modified faces by resetting the trimming curve array in the data structure of the trimmed NURBS surface, as described in section 6.2. Finally, we connect all faces, edges and vertices together to regenerate the B-rep NURBS model. An example is given in figure 6.5. We deform a line constraint on the top surface of the B-rep model. That causes changes to two edges connecting with two general cylinder surfaces (holes). The red coloured surfaces are changed.



(a) The broken original B-rep model.



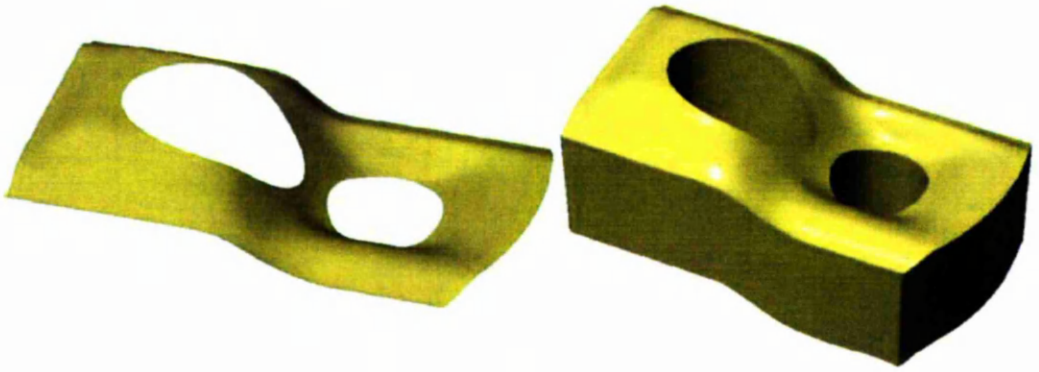
(b) The broken deformed B-rep model (deforming only the top surface; two general cylinder surfaces are also changed).

Figure 6.5: Deforming the B-rep model by using method one (Made in Rhino).

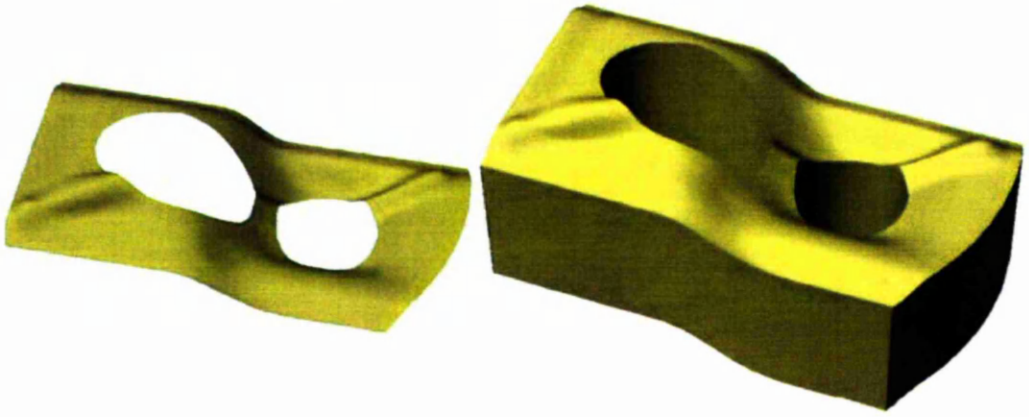
However, when we apply method one to some complex B-rep models, some edges may go beyond the boundary of associated faces and it is also impossible to extend the faces in some degree due to the complex shape of the faces.

Method two overcomes this disadvantage. The only condition is that we must know the history of the construction of this B-rep model so that we can reconstruct the B-rep model from the deformed trimmed NURBS surface. For example, we extrude one trimmed surface to create a B-rep model (figure 6.6a). If we deform a straight-line constraint on this trimmed surface and then extrude the surface again to recreate the B-rep model (figure 6.6b).

Although method two is simple, fast and accurate, it is not suitable for all B-rep models. A simple B-rep solid model does not contain any information about the construction history while a CSG solid model does. The third method can provide a complete solution for the deformation of a B-rep model.



(a) The original trimmed surface and B-rep model



(b) The deformed trimmed surface and reconstructed B-rep model

Figure 6.6: Deforming the B-rep model by using method two (Made in Rhino).

The third method is to create one or more surface patches between the deformed edges and original edges. The patches are the blending surfaces. The blending surface can be created from two boundary NURBS curves. Therefore, the blending surface fits the gap between the original edge and the deformed edge, as shown in figure 6.7. The final result is shown in figure 6.8. The red colour surfaces are the surface patches (blending surface). A comprehensive survey of blending surfaces is given in [72][73][74]. The size of blending surface depends on the effective radius in the metaball deformation model. Because the sizes of blending surfaces are normally much smaller than the whole size of the B-rep model, we can apply the method of creating a ruled surface, described in section 2.4.3, to create the blending surface. It is the simplest blending surface.

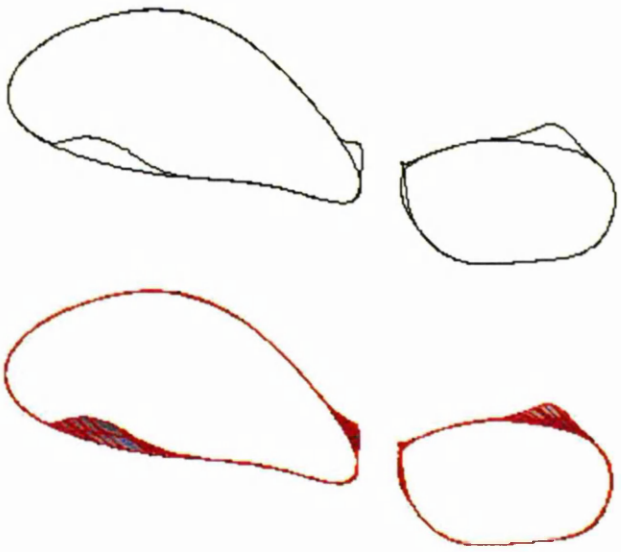


Figure 6.7: original edge curves, deformed edges curves and surface patches

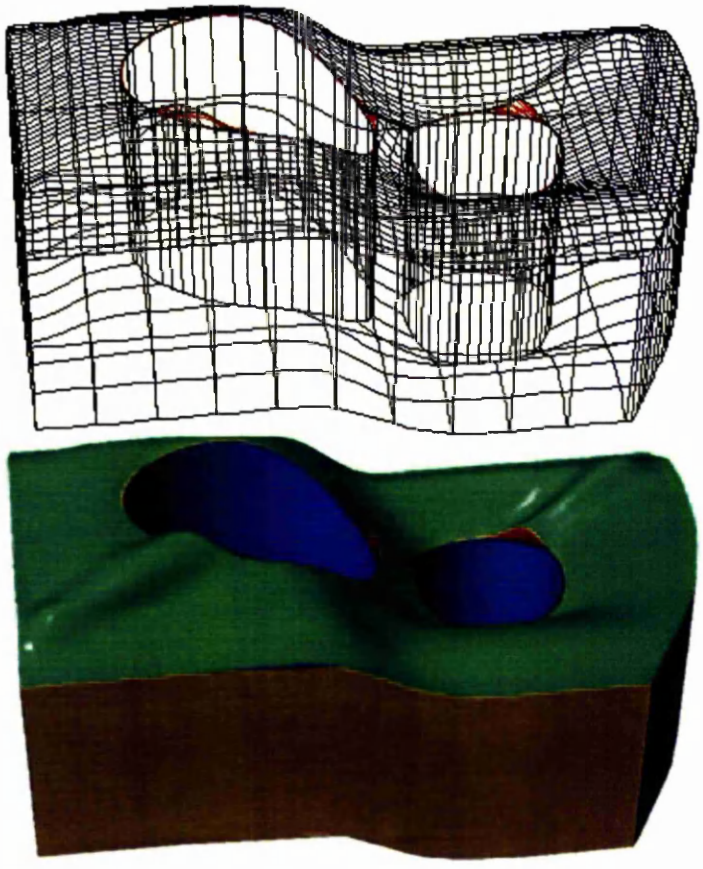


Figure 6.8: The final result by using method three (Made in Rhino).

All three methods have both advantages and disadvantages. Method one involves point inversion and reconstructing a trimmed NURBS surface, so that the computation cost is high. Another problem is that the new trimming curve may go beyond the boundaries of the original NURBS surfaces. In that case, we have to extend the surface to the new boundaries. The advantages of method one are that no additional information is required to reconstruct the B-rep model and no additional surfaces are added. Also the whole structure of the B-rep remains unchanged.

Method two gives a simple solution for reconstructing the B-rep model from the deformed surface. It does not involve any geometric computation and the new B-rep is robust. However, not all B-rep models can be recreated from the single surface.

Method three is the best solution. Blending surfaces are created to fit the gaps between the deformed edges and the original edges. No complicated geometric computation is needed.

6.4 Summary

This chapter has generalized the technique of deforming general constraints on a single untrimmed NURBS surface, trimmed NURBS surface and a NURBS B-rep model by using the metaball deformation model.

Chapter 7

Conclusions

7.1 Summary of Work Done

This thesis describes a number of contributions to CAD and computer graphics. Those contributions are highlighted in this chapter:

1. A novel approach for the point projection and inversion for NURBS curves and surfaces is presented in chapter 4. This method provides a good initial value for Newton-Raphson method to achieve a convergence and that makes the algorithm more reliable. Furthermore, for a NURBS surface, this approach dramatically decreases the computation of the algorithm by decomposing the NURBS surface into a set of quadrilaterals. It also applies the Newton-Raphson method on a Bézier patch instead of the whole NURBS surface, which improves the stability of the algorithm. The algorithm is incorporated into the metaball deformation model (chapter 3 and 6) to calculate the distance between the control points on the NURBS surface to a general constraint (a NURBS curve or surface). The algorithm has been extensively tested

through projecting thousands of control points from the NURBS surface to the constraint to prove its stability and efficiency. The algorithm of point projection for NURBS curves and surfaces has been published in [79].

2. Tessellating trimmed NURBS surfaces is one of the remaining research problems in CAD. A new method has been introduced in Chapter 5. The whole surface is tessellated into a set of both triangles and quadrilaterals, which can be sent directly to the graphics pipeline for rendering. It has the advantages of both optimising the number of polygons (triangles and quadrilaterals) and dynamically subdividing the surface based on the curvature of surface. It does not involve any complex triangle generation algorithm so that the performance of this method is fast and reliable. This research has been published in [80].

3. A new approach to deform the untrimmed NURBS surface, trimmed NURBS surface and NURBS solid model are presented in Chapter 3 and Chapter 6. Previous work on the metaball deformation model is implemented on the mesh model (tessellated solid model) which no longer has accurate geometric information such as boundary surfaces, edge curves, trimming curves and etc. By modifying the position of control points of the NURBS surface, we extend the metaball deformation model to the NURBS objects. The metaball model for NURBS objects can achieve accurate geometric modification, which is suitable for the CAD domain. In the case of NURBS solid models represented by B-Rep, our method deforms one of the boundary surfaces and creates blending surfaces fitting the gap. Deforming the solid model not only keeps more accurate geometric information than deforming the mesh model, but also achieves better rendering results.

7.2 Future work

The ideas presented in this thesis open up several interesting directions for future research:

- **Point Inversion and Projection for NURBS Curves and Surfaces**: The extension of this algorithm to calculate the minimum distance between two NURBS curves or two NURBS surface could be considered. The analysis of the relationship between a single test point and a Bézier subcurve or a Bézier patch can be extended to analyse the relationship between two Bézier subcurves or two Bézier patches. By finding a pair of candidate Bézier subcurves or Bézier patches, we recursively subdivide them until they can be approximated by straight line or a plane. The final minimum distance can be calculated through computing the distance between two straight lines or two planes.

  In a similar way, this algorithm could also be extended to compute the minimum distance between NURBS curves and NURBS surfaces.

- **Adaptive Tessellation for Trimmed NURBS Surface**: Future work on adaptive tessellation techniques will be extended to tessellate the whole NURBS solid model, which contains several trimmed NURBS surfaces forming its boundary surfaces. Tessellating all boundary surfaces separately is not acceptable, because it may lead to holes along the boundary edges. Holes on the edge are caused by the discontinuous vertex connections between two neighbourhood boundary surfaces. Therefore, research into tessellating the whole solid model involves creating unified vertices along edges.

- **Metaball Deformation Model for NURBS Objects**: Initially, we may incorporate different potential functions into the metaball deformation model to create different blending effects. Alternative potential functions are Blinn's exponential function, Nishimura's piece-wise quadric polynomial, and Murakami's degree four polynomial [50]. Further attention should be directed toward incorporating some physical properties into the metaball model also, so that the displacement of individual control points is not only decided by the distance function but also affected by other functions associated with the physical properties. Furthermore, deforming two or more neighbourhood boundary surfaces should be considered, providing a more flexible deformation tool for NURBS solid modelling.

Appendix A

openNURBS Toolkit

The testbed software includes two libraries: o*penNURBS* toolkit and *YLNurbsLib*, which has been developed by the author. This appendix gives an overview of the toolkit. The openNURBS toolkit is available on the openNURBS website (www.opennurbs.org) which is funded by Robert McNeel & Associates.

A.1 Overview of openNURBS toolkit

The openNURBS toolkit can read and write the complete Rhinoceros file format. Rhinoceros is a design software package, developed by Robert McNeel & Associates. Rhinoceros data files contain NURBS curves, surfaces, and solids. These data types can accurately hold all of the 3-D geometry found in most other CAD/CAM file formats including IGES [77], STEP [78], VDA/FS, ACIS, Parasolid, etc.

The openNURBS toolkit is written in standard C++, which can be compiled in Windows, Mac and Linux environments. Object-oriented design makes it possible for the user to understand the relationships between individual

geometric objects that are represented by NURBS. A base geometric object class (*CRhinoObject*) is derived from an abstract class: *CRhinoChunk*, which has a unique ID number and virtual functions for reading and writing geometric data. *CRhinoObject* has some common attributes shared by all geometric objects, such as colour, material, label and bounding box. Figure A.1 shows the hierarchical relationships between all classes inside in the openNURBS toolkit.

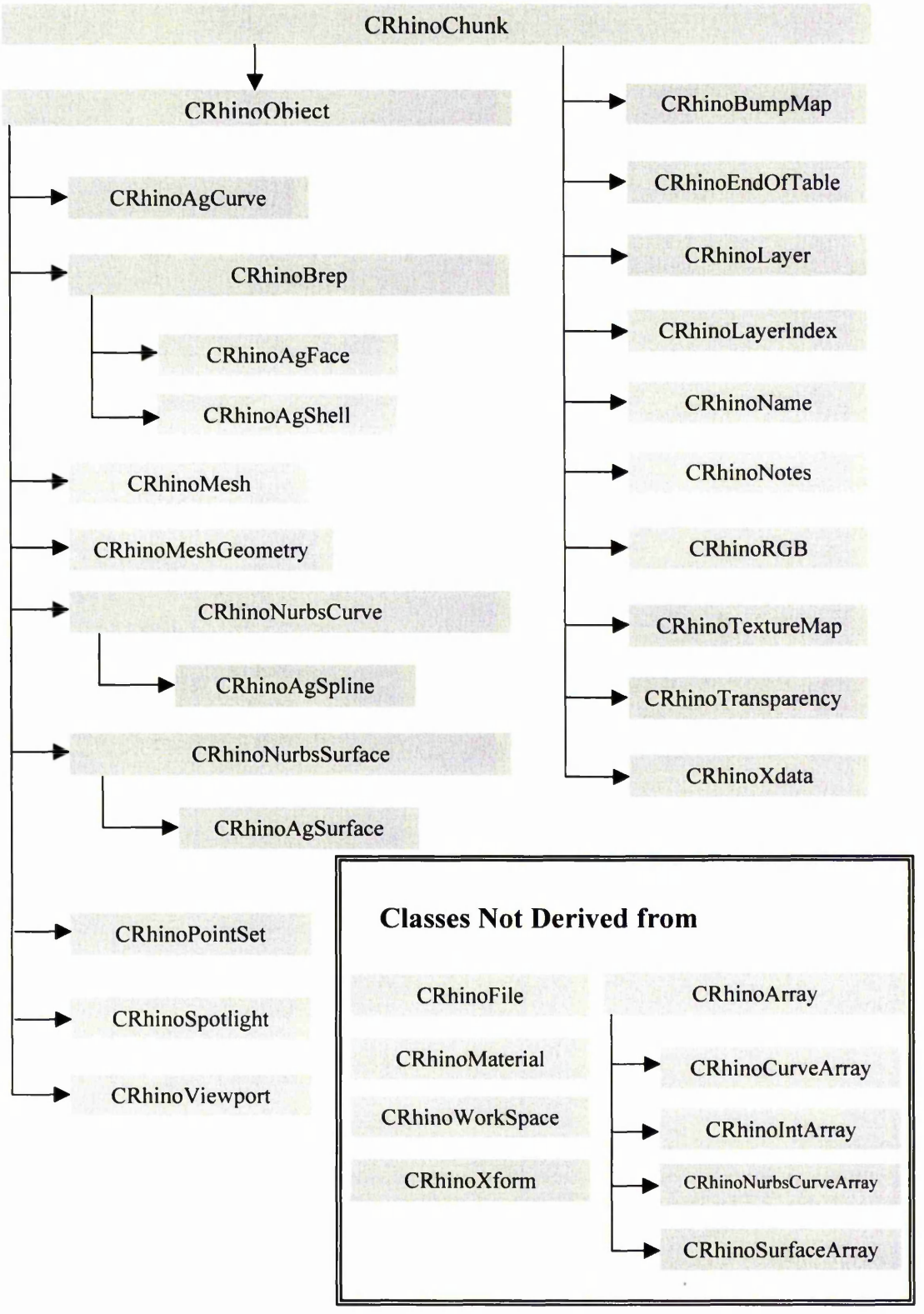


Figure A.1: Hierarchy Chart of openNURBS toolkit libaray

Appendix B

YLNurbsLib

Based on the infrastructure of the openNURBS toolkit, the author developed the YLNurbsLib library, which works as a geometry kernel for his testbed software. It has the object-oriented definitions of NURBS curve, NURBS surface, trimmed NURBS surface and mesh. It also includes methods of point projection and tessellation, which are described in chapter 4 and 5.

B.1 OO definitions of NURBS objects

A simple class: *CYLNLibObject*, an abstract class for the whole library, contains only colour information and several virtual functions. Derived from this abstract class, *CYLNLibNurbsCurve* and *CYLNLibNurbsSurface* have common NURBS data (knot vector, control points and degree), basic functions (evaluation, knot insertion, splitting) and some advanced functions (point projection, tessellation for surface only). Figure B.1 gives the relationships between all classes within the YLNurbsLib library.

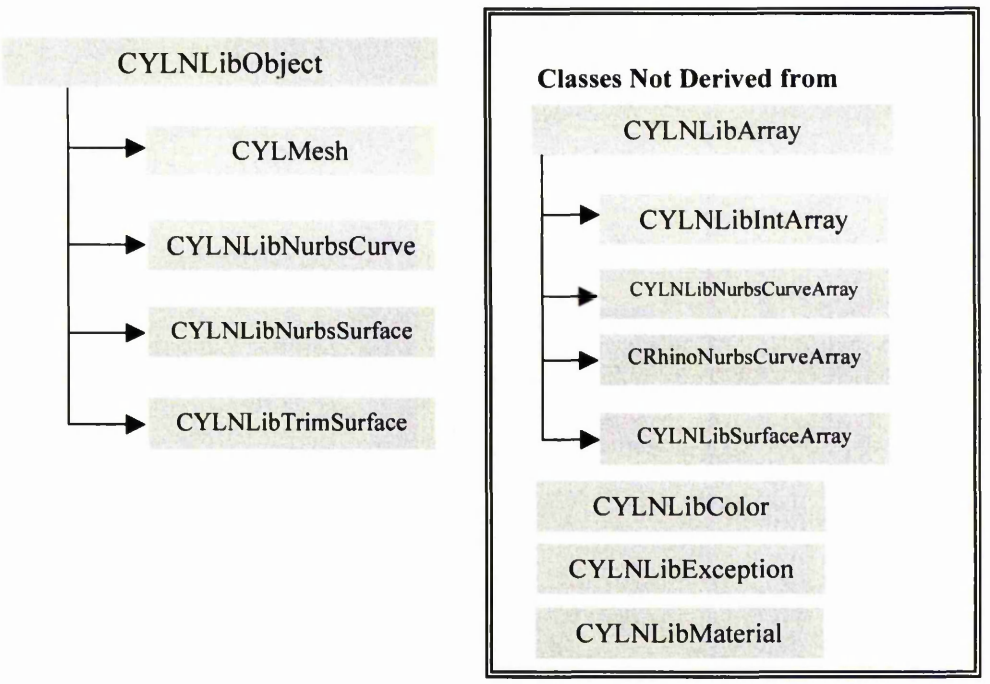


Figure B.1: Hierarchy Chart of YLNurbsLib libaray

To define a NURBS curve in YLNurbsLib, the user needs to give dimension (must be 3D), type (rational or non-rational), degree and number of control points in the NURBS curve construction function:

CYLNLibNurbsCurve (int dimension, BOOL bIsRational, int degree, int cv_count);

The public functions *SetKnot(int index, double k)* and *SetCV(int index, POINT_STYLE style, const double* Point)* in *CYLNLibNurbsCurve* class are used to set up the knot vector and control point array. The length of knot vector can be calculated through equation B.1.

$$length\_knot = degree + 1 + cv\_count \qquad (B.1)$$

Here is an example of a NURBS curve declaration in YLNurbsLib library:

```
Int i , knot_count;
CYLNLibNurbsCurve *NewCurve;
//define a non-rational NURBS cubic curve with 5 control points
NewCurve = new CYLNLibNurbsCurve (3, FALSE,3, 5);

for(i=0; i<5; i++)
   SetCV(i, yl_not_rational, Pi); //Set up non-rational control point array

knot_count = 3+1+5;
For(i=0; i<knot_count; i++)
  SetKnot(i, Ki);
```

Similar to the NURBS curve, a NURBS surface can be defined through the surface construction function:

```
CYLNLibNurbsSurface(
                        int dimension,          //Must be 3
                        BOOL bIsRational,  //True for rational, False for non-rational
                        int degree_u,          //degree in U direction
                        int degree_v,          //degree in V direction
                        int cv_count_u,     //CV count in U direction
                        int cv_count_v      //CV count in V direction
);
```

Also two public functions (*SetKnot* and *SetCV*) are used to set up the U, V knot vectors and control point net.

The class declaration of trimmed NURBS surface has been already described in section 6.2. The trimmed NURBS in YLNurbsLib has some additional private data members such as point arrays for external boundaries and internal boundaries, which are used by the functions inside class and are not accessible from functions outside the class even the derived class.

B.2 Memory Management

Memory management is an important issue in software development. Memory leakage is a serious problem in many applications. If the software has a memory leakage problem and runs for a very long period, the computer may run out of memory and crash. The YLNurbsLib library uses a dynamic array to store all NURBS objects. *CYLNLibArray* is a template dynamic array. Because NURBS objects are a complex class, it is not possible to directly put the object in the dynamic array. Instead pointers to NURBS object classes are stored in the dynamic array. Therefore, to prevent the memory leakage problem, it requires two steps to release all memory used by the NURBS objects.

- Step 1: delete NURBS objects.
- Step 2: delete all pointers allocated in the array.

Here is an example of dynamic NURBS curve array:

- Definition

```
class NurbsCurveArray : public CYLNLibArray<CYLNLibNurbsCurve*>
{
public:
  NurbsCurveArray( size_t = 0 );
  ~NurbsCurveArray();
};
```

- Release all memory

```
size = NurbsCurveArray.GetSize();
for(i=0;i<size;i++)
   delete NurbsCurveArray[i];

NurbsCurveArray.RemoveAll();  //Remove all pointers inside the dynamic array
```

References

[1] L Piegl. On NURBS: A Survey. IEEE Computer Graphics and Applications, 11(1):55-71, 1991.

[2] Xiang Fang, Hujun Bao and Pheng Ann Heng. Continuous field based free-form surface modelling and morphing. *Computer & Graphics*, 25: 235-243, 2001.

[3] Blinn JF. A generalization of algebraic surface drawing. *ACM Transactions on Graphics*, 1(3):235-56, 1982.

[4] Nishimura H, Hirai M and Kawai T. Object modelling by distribution function and a method of image generation. *Transaction on IECE*, 68-D(4):718-25, 1985.

[5] Wyvill G, McPheeters C and Wyvill B. Data structure for soft objects. *The Visual Computer*, 2:227-34, 1986.

[6] Shen J and Thalmann D. Interactive shape design using meta-balls and splines. In: Brian Wyvill, Marie-Paule Gascuel, editors. *Proceedings of implicit surface'95*. France: Grenoble, pp.187-96, 1995.

[7] Bloomenthal J and Wyvill B. Interactive techniques for implicit modelling. *Computer Graphics*, 24(2) 109-16, 1990.

[8] Xiaogang Jin, Youfu Li and Qunsheng Peng, General constrained deformations based on generalized metaballs. *Computer & Graphics* 24:219-231, 2000.

[9] Miller, J R. Sculptured surfaces in solid models: issues and alternative approaches. *IEEE Computer Graphics & Application.* 6(12):37-48, 1986.

[10] Casale, M S. Free-form solid modelling with trimmed surface patches, *IEEE Computer Graphics & Application.* 7(1): 33-43, 1987.

[11] Farouki, R T. Trimmed surface algorithms for the evaluation and interrogation of solid boundary representations. *IBM Journal Research & Development*, 31(3): 314-334, 1987.

[12] Sederberg TW and Parry SR. Free-form deformation of solid geometric models. *ACM Computer Graphics*, 18(3):21-30, 1984.

[13] C De Boor. *A Practical Guide to Splines.* Springer-Verlag, 1978.

[14] G Farin. *Curves and surfaces for Computer Aided Geometric Design: A Practical Guide*. Academic Press, 3rd edition, 1983.

[15] M G Cox. The Numerical Evaluation of B-Splines. *Journal of Institute of Mathematics and its Applications*, 10(2):134-149, 1972.

[16] W Tiller. Rational B-Splines for Curve and Surface Representation. *IEEE Computer Graphics & Applications*, 3(6):61-69, 1983.

[17] L Piegl and W Tiller. *The NURBS Book.* Springer-Verlag, 1995.

[18] L Piegl and W Tiller. Curve and Surface Constructions Using Rational B-Splines. *Computer-Aided Design*, 19(9):485-498, 1987.

[19] L Piegl. Modifying the Shape of Rational B-splines. Part 1: Curves. *Computer Aided Design*, 21(8):509-519, 1989.

[20] L Piegl. Modifying the Shape of Rational B-splines. Part 2: Surfaces. *Computer Aided Design*, 21(9):538-546, 1989.

[21] W Boehm. Inserting New Knots into B-spline Curve. *Computer Aided Design*, 12(4):199-201, 1980.

[22] W Tiller. Knot-removal algorithms for NURBS curves and surfaces. Computer Aided Design, 24(8):445-453, 1992.

[23] F Lin. *NURBS in CAD and Computer Graphics.* Ph.D thesis, University of Manchester, 1996.

[24] Alan Watt. *3D Computer Graphics (Third Edition).* Addison-Wesley, 2000.

[25] T W Sederberg and S R Parry. Comparison of Three Curve Intersection Algorithms. *Computer Aided Design*, 18(1): 58-63, 1986.

[26] N M Patrikalakis. Surface-to-Surface Intersections. *IEEE Computer Graphics and Applications*, 13(1):89-94, 1993.

[27] Keyser, J., Krishnan, S and Manocha, D. Efficient and accurate B-rep generation of low degree sculptured solids using exact arithmetic: I —representations. *Computer Aided Geometric Design* 16:841-859, 1999.

[28] Keyser, J., Krishnan, S and Manocha, D. Efficient and accurate B-rep generation of low degree sculptured solids using exact arithmetic: II—Computation. *Computer Aided Geometric Design* 16:861-892, 1999.

[29] Hoffmann, C.M. *Geometric and Solid Modelling*. Morgan Kaufmann, San Mateo, CA, 1989.

[30] Hoffmann, C.M. How solid is solid modelling, in: M.C. Lin and D. Manocha, eds., *Applied Computational Geometry*, Springer, Berlin, 1-8, 1996.

[31] Krishnan, S. *Efficient and accurate boundary evaluation algorithms for sculptured solids*. Ph.D Thesis, Department of Computer Science, University of N. Carolina at Chapel Hill, 1997.

[32] Mantyla, M. *An Introduction to Solid Modelling*. Computer Science Press, Rockville, Maryland, 1988.

[33] Coquillart, S. Extended Free-form Deformation: A sculpturing Tool for 3D Geometric Modelling. *Proceedings of SIGGRAPH'90*, Computer Graphics 24(4):187-196, 1990.

[34] Xiang Fang, Hujun Bao, Pheng Ann Heng and etc. continuous field based free-form surface modelling and morphing. *Computers & Graphics* 25:235-243, 2001.

[35] L. Piegl. Modifying the Shape of Rational B-Splines. Part 1: Curves. *Computer Aided Design*, 21(8): 509-518, 1989.

[36] L. Piegl. Modifying the Shape of Rational B-Splines. Part 2: Surfaces. *Computer Aided Design*, 21(9):538-546, 1989.

[37] A. H. Barr. Global and Local Deformations of Solid Primitives. *Proceedings of SIGGRAPH'84*, Computer Graphics 18(3):21-30, 1984.

[38] T. W. Sederberg and S. R. Parry. Free-Form Deformation of Solid Geometric Models. *Proceedings of SIGGRAPH'86, Computer Graphics* 20(4):151-160, 1986.

[39] J. E. Chadwick, D. R. Haumann, and R.E. Parent. Layered Construction for Deformable Animated Characters. *Proceedings of SIGGRAPH'89, Computer Graphics* 23:243-252, 1989.

[40] T. W. Sederberg and S. R. Parry. Free-Form Deformation of Polygonal Data. *Second Image Symposium*: 633-639, CESTA, 1986.

[41] J. H. Clark. Parametric curves, surfaces and volumes in computer graphics and computer aided geometric design. *Technical report 221*, Stanford University, 1981.

[42] Hsu W., Hughes J. and Kaufmann H. Direct manipulations of free-form deformations. *Computer Graphics* 26(2): 177-184, 1992.

[43] Borrel P. and Bechmann D. Deformation of N-dimensional Objects. *International Journal of Computational Geometry and Applications* 1(4):427-453, 1991.

[44] Borrel P. and Rappoport A. Simple constrained deformations for geometric modelling and interactive design. *ACM Transactions on Graphics* 13(2):137-55, 1994.

[45] Blinn JF. A generalization of algebraic surface drawing. *ACM Transactions on Graphics* 1(3):235-256, 1982.

[46] Nishimura H., Hirai M. and Kawai T. Object modelling by distribution function and a method of image generation. *Transaction on IECE* 68-D(4):718-725, 1986.

[47] Wyvill B. and Wyvill G. Field functions for implicit surfaces. *The Visual Computer* 5:75-82, 1989.

[48] Bloomenthal J. and Wyvill B. Interactive techniques for implicit modelling. *Computer Graphics* 24(2):109-116, 1990.

[49] Bloomenthal J., Bajaj C., Blinn J., Cani-Gascuel M., Rockwood A., Wyvill B. and Wyvill G. *An introduction to implicit surfaces*. Los Altos, CA: Morgan Kaufmann Publishers, 1997.

[50] Nishita T. and Nakamae E. A method for displaying metaballs by using Bézier clipping. Computer Graphics Forum 13(3):271-280, 1994.

[51] David E. Johnson and Elaine Cohen. A framework for efficient minimum distance computation. *Proc. IEEE Intl. Conf. Robotics & Automation*, Leuven, Belgium, 16-21:3678-3684, 1998.

[52] Chin, Francis and Wang, Cao. An Optimal algorithms for the intersection and the minimum distance problems between planar polygons. *IEEE transactions on Computers*, C-32(12):1203-1207, 1983.

[53] Edelsbrunner, H. On computing the extreme distances between two convex polygons. Tu Graz, *Tech. Rep.* F96, 1982.

[54] Michael E. Mortenson. *Geometric Modeling*. John Wiley & Sons, New York, pp 305-317, 1985.

[55] Limaiem, Anis and Trochu, Francois. Geometric algorithms for the intersection of curves and surfaces. *Computer & Graphics*, 19(3):391-403, 1995.

[56] Lin, Ming and Manocha, Dinesh. Fast interference detection between geometric models. *The Visual Computer*: 541-561, 1995.

[57] Piegl L. and Tiller W. Parametrization for surface fitting in reverse engineering. *Computer Aided Design*, 33:593-603, 2001.

[58] Michael Shantz and Sheue-Ling Chang. Rendering trimmed NURBS with adaptive forward differencing. *Computer Graphics Proceedings of Siggraph'88*, 1988.

[59] Salim S. Abi-Ezzi and Leon A. Shirman. Tessellation of curved surfaces under highly varying transformation. *Proceedings of EUROGRAPHICS '93*, pp385-397, 1993.

[60] Salim S. Abi-Ezzi and Srikanth Subramaniam. Fast tessellation of trimmed NURBS surface. *Proceedings of EUROGRAPHICS '94*, 1994.

[61] Sheng, X and Hirsh, B.E. Triangulation of trimmed surfaces in parametric space. *Computer-Aided Design* 24(8):437-444, 1992.

[62] Filip, D, Magedson, R and Markot, R. Surface algorithm using bounds on derivatives. *Computer-Aided Geometric Design* 3:295-311, 1986.

[63] Piegl, L.A. and Richard, A.M. Tessellating trimmed NURBS surface. *Computer-Aided Design* 27(1):16-26, 1995.

[64] Vigo, M. Directional adaptive surface triangulation. *Computer-Aided Design* 16:107-126, 1999.

[65] Alan Watt. *3D Computer Graphics (Third Edition)*. Addison-Wesley, pp 128 –129.

[66] Jonathan E. Steinhart and James Arvo. *Graphics Gems II*. ISBN 0-12-064480-0, Academic Press, Inc, 1991.

[67] Hanan Samet. *Applications of Spatial Data Structures*. ISBN 0-201-50300-X, Addison-Wesley, Reading, MA. (I.9 Scanline Coherent Shape Algebra; IV.7 Quadtree /Octree-to-Boundary Conversion), 1990.

[68] Hanan Samet. *The Design and Analysis of Spatial Data Structures*. ISBN 0-201-50255-0, Addison-Wesley, Reading, MA (I.9 Scanline Coherent Shape Algebra), 1990.

[69] William D. Atkinson *Method and Apparatus for Image Compression and Manipulation*. United States Patent Number 4,622,545. (I.9 Scanline Coherent Shape Algebra), 1986.

[70] Wolfgang Boehm. Inserting New Knots into B-Spline Curves. *Computer Aided Design* 12:99-201, 1980.

[71] Elaine Cohen, Tom Lyche, and Richard Riesenfeld. Discrete B-Splines and Subdivision Techniques in Computer-Aided Geometric Design and Computer Graphics. *Computer Graphics and Image Processing* 14, 1980.

[72] J. Vida, R. R. Martin, and T. Varady. A Survey of Blending Methods That Use Parametric Surfaces. *Computer Aided Design*, 26(5):341-363, 1994.

[73] L Bardis and N. M. Patrikalakis. Blending Rational B-Spline Surfaces. In *Proceedings of Eurographics '89*, 1989.

[74] P C Filkins, S. T. Tuohy, and N. M. Patrikalakis. Computational Methods for Blending Surface Approximation. *Engineering with Computers*, (9):49-62, 1993.

[75] Marian Bozdoc and Auckland NZ. *History of Computer Aided Design*. URL: http://www.thocp.net/software/cad.htm.

[76] W Welch and A Witkin. Variational Surface Modelling. In *Proceedings of SIGGRAPH'92*, pp 157-167,1992.

[77] ANSI. *The Initial Graphics Exchange Specification (IGES) Version 5.2*, 1993. ANSI Y14.26M.

[78] ISO. *Standard for Exchange of Product Model Data (STEP)*, 1994. ISO 10303.

[79] Y. Ma and W. T. Hewitt. Point inversion and projection for NURBS curve and surface. *Computer Aided Geometric Design*. (Article in press)

[80] Y. Ma and W. T. Hewitt. Adaptive Tessellation for Trimmed NURBS Surface. Short presentation, *Proceedings of EUROGRAPHICS '2002*, Germany, 2002.