FREE FORM SURFACE ANALYSIS USING A HYBRID OF SYMBOLIC AND NUMERIC COMPUTATION

by

Gershon Elber

A dissertation submitted to the faculty of
The University of Utah
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science

The University of Utah

December 1992

THE UNIVERSITY OF UTAH GRADUATE SCHOOL

SUPERVISORY COMMITTEE APPROVAL

of a dissertation submitted by

Gershon Elber

This dissertation has been read by each member of the following supervisory committee and by majority vote has been found to be satisfactory.

	Chair: Elaine Cohen
	Rich Riesenfeld
	Sam Drake

THE UNIVERSITY OF UTAH GRADUATE SCHOOL

FINAL READING APPROVAL

To the Graduate Council of The University of Utah:

I have read the dissertation of Gershon Elber in its final form and have found that (1) its format, citations, and bibliographic style are consistent and acceptable; (2) its illustrative materials including figures, tables, and charts are in place; and (3) the final manuscript is satisfactory to the Supervisory Committee and is ready for submission to the Graduate School.

Date

Elaine Cohen
Chair, Supervisory Committee

Approved for the Major Department

Tom Henderson
Chair/Dean

Approved for the Graduate Council

B. Gale Dick
Dean of The Graduate School

ABSTRACT

Detailed analysis of many mathematical properties of sculptured models has been hindered by the fact that the properties do not have the same representation as the surface. For example, unit tangents, surface normals, and principal curvatures are typically computed at predefined discrete sets of points on the surface. As such, aliasing can occur and features between samples can be missed. Synthesizing information about the shape of an object and operating on the model, whether by physical machining tools, graphics display programs, or mathematical analysis, has been treated as either a discrete or local problem in general. The research being reported on here has focused on another approach, that of creating algorithms that construct the mathematical properties in closed form, or construct approximations to those mathematical properties through symbolic computation. Global analysis can then be applied while an accurate error bound is obtained.

Basic tools required for such symbolic computation are presented and their usage in a broad range of applications from offset approximations through curvature analysis to generation of machining toolpaths are demonstrated. The combination is not only shown to be powerful but it also provides a novel approach to problem solving in an elegant and robust way.

This thesis is dedicated to my wife

Irit

and my two daughters

Lotem and Sivan.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to thank all the people who explicitly or implicitly made the completion of this work a possibility.

Elaine Cohen, with her unwillingness to accept the unclear, pushed me into better and well-defined algorithms. Her mathematicals skill, combined with the practical view of the problems the CAGD field is facing, were a tremendous help for this work.

Rich Riesenfeld and Elaine Cohen provided generous funding that allowed me to devote my time to interesting research. IBM generously supported me for two years on a fellowship.

Sam Drake, the third member of my committee, helped in testing all the NC machining related algorithms.

Mike Milochik provided the picture developing services and made all the color enlargements of this document. Hank Driskill volunteered his scanner and scanned some of the noncomputer generated images back into electronic form.

It was impossible to perform this research without the excellent environment that the Alpha_1 group has created. I could not have hoped for a better environment in which to work. Special thanks go to Beth Cobb and Robert Mecklenburg for their willingness to try and help with problems that arose.

Finally, this work would never have started, had not my wife Irit been willing to make this huge sacrifice. Her support and control on our growing family allowed me to devote more time to this research than I could ever hoped. My gratitude extends to my parents, who encouraged me to apply and study at the best school I could find, not necessarily the closest.

CHAPTER 1

INTRODUCTION

"Where shall I begin, please your Majesty?" he asked. "Begin at the beginning," the King said, gravely, "and go on till you come to the end: then stop."

Alice's Adventures in Wonderland, Lewis Carroll

The field of CAGD has evolved significantly in the last decade. The B-spline representation, introduced to the field of CAGD during the 70s, has become a dominant representation in mechanical design. Techniques to manipulate, evaluate, mill, render, and analyze freeform surface based models have undergone extensive research. It is common for surfaces/curves to be approximated by a set of polygons/polylines for milling, rendering, and analysis purposes. Other properties are computed at discrete locations and interpolation is used to provide the information over the entire domain.

This thesis applies symbolic computation to some of these problems. The power of symbolic computation for freeform curves and surfaces will be demonstrated throughout this document. Symbolic computation provides the ability to compute properties with exact precision. It virtually eliminates the fundamental problems that rise from discrete sampling. If the domain being sampled contains information in higher frequencies than that of the samples, the original data cannot be reconstructed precisely (Nyquist theorem). Unfortunately, many important problems have that characteristic.

The use of symbolic computation opens the door for solving problems using a global approach. It may be useful to demonstrate the differences between global and

local methods using an example from computer graphics. Two common methods are used to render realistic scenes: ray tracing and radiosity. The first "fires" a single ray at a time and samples the world along that particular line. This is clearly a local approach. The second looks over the problem globally and finds all the energy distribution simultaneously in the equilibrium state. Given a scene, this second technique is ideally global because it takes into account *all* data. We say "ideally" because several approximations are commonly performed in this method to obtain faster results. It is not surprising then, that ray tracing methods face severe aliasing problems. Major research efforts in the computer graphics community are devoted to overcoming the ray sampling problems. The ideal radiosity method does not introduce aliasing problems, although the scene subdivision and polygonization stage does, simply because most of the subdivision techniques are local.

Another way to distinguish global techniques from local ones is that global techniques can arrive at all their results at the same time while local methods provide the viewer with sequential information. A global method may be used to analyze a whole surface at once. The ray tracing technique from the above example processes one sampled ray at a time, whereas the radiosity method computes and returns the light distribution information for all elements in the scene simultaneously.

Greedy algorithms are local, as shown by the coin based example in [1], pp. 321 emphasizing their global inefficiency. Taylor approximations are another example of exploiting local information. The Newton Raphson curve root finding is clearly a local technique in that it converges to roots in a neighborhood of the initial guess. Recall that it does not ensure finding all the roots. On the other hand, if an algorithm uses properties of the *entire* surface and makes decisions based on both local and global information, it is referred to as a *global* algorithm. An ideal method should find all the solutions quickly and so must be global.

We use derived surfaces, called *property surfaces*, whose definitions are derived

from different attributes of the original surface as auxiliary surfaces to help analyze the original surface. For example, $\frac{\partial F}{\partial u}(u,v)$ is a property surface of F, and so is $n(u,v)$, the surface of unit normals. The two surfaces of principal curvatures, $\kappa_n^1(u,v)$ and $\kappa_n^2(u,v)$ are also property surfaces.

> **Definition 1.1** *Suppose $\mathcal{S}_1$ and $\mathcal{S}_2$ are vector spaces of surfaces. An operator $\mathcal{P} : F \rightarrow p \in \mathcal{S}_2$, for all $F \in \mathcal{S}_1$, is called a* property operator *if the image surface, p, is associated with a property of F, the domain surface. In that case, p is called a* property surface.

Some property surfaces are of the same "type" as the original surface whereas others are not. If F is a tensor product NURBs surface, then $\frac{\partial F}{\partial u}(u,v)$ is a property surface which is also a tensor product NURBs surface with the same knot vectors, but with different (lower) order and continuity properties. $\frac{\partial F}{\partial u}(u,v) \times \frac{\partial F}{\partial v}(u,v)$ is also a property surface, that is, a tensor product NURBs surface, but with different knot vectors, different (higher) order, and different (lower) continuity. These two property surfaces share the trait with F that they are NURBs surfaces, but $n(u,v)$, $\kappa_n^1(u,v)$ and $\kappa_n^2(u,v)$ are not NURBs surfaces, in general. They cannot be represented as a piecewise parametric rational functions, as we shall later see, and hence, cannot be represented as NURBs surfaces.

We restrict ourselves (definition 1.1) to using property surfaces that are either representable as NURBs or to property surfaces for which we can derive approximations that are representable as NURBs. This restriction enables us to apply any algorithmic approach developed for the NURBs representation to the property surfaces. Not every property is representable as a NURBs surface. A unit normal surface has a square root in the denominator of its normalization which is not representable as a NURBs.

Contouring techniques [4, 45, 59] developed for freeform surfaces can be applied immediately to a NURBs representation of a property surface. Because both the

original and the property surfaces share the same parametric domain, one can easily create a trimmed surface consisting of those regions in the original surface having the desired property values. In other cases the zero set of certain property surfaces may be required. For example, let $\hat{n}_z(u,v)$ be the z component of $\hat{n}(u,v)$, where $\hat{n}(u,v) = \frac{\partial F(u,v)}{\partial u} \times \frac{\partial F(u,v)}{\partial v}$ is orthogonal to the parametric surface $F(u,v)$ at (u,v). Then the set of zeros of $\hat{n}_z(u,v)$ is simply the parameter values along the silhouettes of the original surface when F is being viewed from $(0,0,\infty)$. Hence, the silhouette extraction problem is equivalent to a root finding problem (contouring), which is usually simpler. Trimmed surfaces [9, 47] are the natural way to represent the regions defined by the contouring operator. In fact, the parameter values of the contours of certain property surfaces can serve as the parameter values of a trimming curve for the original surface.

In this thesis, we apply global techniques based on symbolic computation to a broad range of problems. In Chapter 2, we develop the tools, some of which are described above, that will be used throughout this dissertation. It may be a surprise how small the number of required tools is.

As is discussed in Chapter 3, offsets of freeform piecewise polynomial/rational curves and surfaces are not, in general, representable in the same domain. Approximation techniques are used instead. However, we use a symbolic method to compute the error function of these approximations. We use this error function in two ways. First its extrema serve as a bound on the error. In addition, isolation of the regions of the error function with large error allow us to automatically improve the approximation until a prescribed tolerance is achieved.

In Chapter 4, surface curvature analysis is performed globally using symbolic computation of curvature property surfaces. Curvature analysis has applications in modeling as well as in manufacturing. Symbolic computation provides the ability to trichotomize a surface into three regions, convex, concave, and saddlelike regions.

This trichotomy, which can dramatically improve milling algorithms efficiency, is almost impossible without a global approach.

In Chapter 5, the questions of toolpath generation for NC machining is dealt with. Optimal toolpaths for freeform surfaces is known as a difficult problem and current approaches fails in extreme cases. We will present a symbolic algorithm that generates a toolpath for machining freeform surfaces that performs much better than current local schemes. This algorithm is then enhanced so it can automatically generate machining toolpaths for real models consisting of several trimmed surfaces while avoiding any gouging. Because the algorithm provides a bound on the redundancy in the generated toolpath, it was successfully modified and used as a rendering tool. The toolpath curves are rendered using curve rendering techniques to form the image.

Finally in Chapter 6, several other applications are addressed. Approximation of higher order curves using lower order ones is the first. We also add a composition operator to the set of operators we defined in Chapter 2 and discuss its potential. Other surface properties such as speed, and slope are defined and considered, and twist is considered as part of a global symbolic approach.

CHAPTER 2

SYMBOLIC AND NUMERIC COMPUTATION

Equations are more important to me, because politics is for the present, but an equation is something for eternity.

Albert Einstein

This Chapter develops the symbolic representational and numeric computational tools required to carry out the analysis performed throughout this document. The derivations of the tools for the Bézier representation are presented while the appropriate references to derivations for the equivalent tools for the NURBs representation are made.

2.1 Symbolic Representation

The following basic symbolic representations will be required for Bézier and NURBs curves and surfaces:

- derivative representation.
- sum/difference representation.
- product representation.

This quite minimal set of representations is extremely powerful tool as is demonstrated by the following Chapters. Because a division by a scalar entity can always be defined as a rational expression, we never have to compute such an operation

explicitly: $x/y \equiv \frac{x}{y}$. It should be carefully noted that these representations represent the result of symbolic operators. That is, they represent the result by means of symbols instead of evaluating it numerically at a single point. More practically, the result is represented in the same Bézier or NURBs domain as a curve or a surface, so a closure is formed. This closure enables arbitrary composition of these operators.

2.1.1 Derivatives Representation

Representing the derivatives of Bézier and NURBs curves and surface is straightforward [27]. Differentiation of a single Bézier basis function may be expressed as a linear combination of two lower order Bézier basis functions:

$$\begin{aligned}
\mathbf{B}_i^{n(1)}(t) &= \binom{i}{n} i t^{i-1}(1-t)^{n-i} - \binom{i}{n}(n-i)t^i(1-t)^{n-i-1} \\
&= n\left(\frac{(n-1)!}{(i-1)!(n-i)!}t^{i-1}(1-t)^{n-i} - \frac{(n-1)!}{i!(n-i-1)!}t^i(1-t)^{n-i-1}\right) \\
&= n\left(\binom{i-1}{n-1}t^{i-1}(1-t)^{n-i} - \binom{i}{n-1}t^i(1-t)^{n-i-1}\right) \\
&= n(\mathbf{B}_{i-1}^{n-1}(t) - \mathbf{B}_i^{n-1}(t)). \qquad (2.1)
\end{aligned}$$

For curves it immediately follows that:

$$\begin{aligned}
\frac{dC(t)}{dt} &= \sum_{i=0}^{n} P_i \mathbf{B}_i^{n(1)}(t) \\
&= \sum_{i=0}^{n} nP_i(\mathbf{B}_{i-1}^{n-1} - \mathbf{B}_i^{n-1}) \\
&= n(\sum_{i=0}^{n} P_i\mathbf{B}_{i-1}^{n-1} - \sum_{i=0}^{n} P_i\mathbf{B}_i^{n-1}) \\
&= n(\sum_{i=1}^{n} P_i\mathbf{B}_{i-1}^{n-1} - \sum_{i=0}^{n-1} P_i\mathbf{B}_i^{n-1}) \\
&= n\sum_{i=0}^{n-1}(P_{i+1} - P_i)\mathbf{B}_i^{n-1}. \qquad (2.2)
\end{aligned}$$

Extension to tensor product surfaces is straightforward because there is no dependency between the two surface parameters (m is the degree).

$$\begin{aligned}\frac{\partial S(u,v)}{\partial u} &= \sum_{i=0}^{m}\sum_{j=0}^{n} P_{ij}\mathbf{B}_i^m(u)^{(1)}\mathbf{B}_j^n(v) \\ &= m\sum_{i=0}^{m-1}\sum_{j=0}^{n}(P_{(i+1)j}-P_{ij})\mathbf{B}_i^{m-1}(u)\mathbf{B}_j^n(v), \end{aligned} \tag{2.3}$$

and similarly for $\frac{\partial S(u,v)}{\partial v}$.

Differentiation in the NURBs domain follows the same procedure [27] (k is the degree):

$$\begin{aligned}\frac{dC(t)}{dt} &= \sum_{i=0}^{n-1} P_i\mathbf{B}_{i,\tau}^k(t)^{(1)}(t) \\ &= k\sum_{i=0}^{n-2}\frac{(P_{i+1}-P_i)}{t_{i+k}-t_i}\mathbf{B}_{i,\tau}^{k-1}(t), \end{aligned} \tag{2.4}$$

and for surfaces:

$$\begin{aligned}\frac{\partial S(u,v)}{\partial u} &= \sum_{i=0}^{m}\sum_{j=0}^{n} P_{ij}\mathbf{B}_{i,\tau}^k(u)^{(1)}\mathbf{B}_{j,\xi}^l(v) \\ &= k\sum_{i=0}^{m-1}\sum_{j=0}^{n}\frac{P_{(i+1)j}-P_{ij}}{t_{i+k}^u-t_i^u}\mathbf{B}_{i,\tau}^{k-1}(u)\mathbf{B}_{j,\xi}^l(v). \end{aligned} \tag{2.5}$$

2.1.2 Representation of Sum/Difference

Finding a representation for the sum or difference of two Bézier or NURBs curves or surfaces can be achieved by bringing them to a common representation. If the two curves do not share the same parametric domain, their knot vectors can always be affinely transformed without modifying the curves, so their parametric domains will match. If the two curves or surfaces are not of the same polynomial order, the lower one should be degree raised [16, 17] to the higher order. If internal knots (of a B-spline curve) have different multiplicities in the two curves, at each knot, the curve with the lower multiplicity should be refined [7, 15] to match the higher

multiplicity. Once both curves are transformed to have a common order and knot vector, their control polygons can simply be summed or differenced because:

$$\begin{aligned} C_1(t) \pm C_2(t) &= \sum_{i=0}^{m} P_i \mathbf{B}_{i,\tau}^m(t) \pm \sum_{i=0}^{m} Q_i \mathbf{B}_{i,\tau}^m(t) \\ &= \sum_{i=0}^{m} \left(P_i \pm Q_i\right) \mathbf{B}_{i,\tau}^k(u) \end{aligned} \tag{2.6}$$

This condition also holds for surfaces. Once the two surfaces share the same orders and knot vectors, their meshes can be simply subtracted or added:

$$\begin{aligned} S_1(u,v) \pm S_2(u,v) &= \sum_{i=0}^{k}\sum_{j=0}^{l} P_{ij} \mathbf{B}_{i,\tau^u}^m(u) \mathbf{B}_{j,\tau^v}^n(v) \pm \sum_{i=0}^{k}\sum_{j=0}^{l} Q_{kl} \mathbf{B}_{i,\tau^u}^m(u) \mathbf{B}_{j,\tau^v}^n(v) \\ &= \sum_{i=0}^{k}\sum_{j=0}^{l} (P_{ij} \pm Q_{kl}) \mathbf{B}_{i,\tau^u}^m(u) \mathbf{B}_{j,\tau^v}^n(v) \end{aligned} \tag{2.7}$$

Bringing two Bézier curves to a common domain only requires elevating the degree of the lower one.

$$\begin{aligned} (1-t)\mathbf{B}_i^n(t) &= \frac{n!}{i!(n-i)!} t^i (1-t)^{n-i+1} \\ &= \frac{n-i+1}{n+1} \binom{i}{n+1} t^i (1-t)^{n-i+1} \\ &= \frac{n-i+1}{n+1} \mathbf{B}_i^{n+1}(t) \end{aligned}$$

and

$$\begin{aligned} t\mathbf{B}_i^n(t) &= \frac{n!}{i!(n-i)!} t^{i+1} (1-t)^{n-i} \\ &= \frac{i+1}{n+1} \binom{i+1}{n+1} t^{i+1} (1-t)^{n-i} \\ &= \frac{i+1}{n+1} \mathbf{B}_{i+1}^{n+1}(t). \end{aligned}$$

Therefore a Bézier basis function of degree n may be represented as a convex combination of two Bézier basis functions of degree $n+1$:

$$\mathbf{B}_i^n(t) = (1-t)\mathbf{B}_i^n(t) + t\mathbf{B}_i^n(t)$$

$$= \frac{n-i+1}{n+1}\mathbf{B}_i^{n+1}(t) + \frac{i+1}{n+1}\mathbf{B}_{i+1}^{n+1}(t). \tag{2.8}$$

Given a Bézier curve of degree n, raising it to degree $n+1$ involves:

$$\begin{aligned} & C(t) \\ = & \sum_{i=0}^{n} P_i \mathbf{B}_i^n(t) \\ = & \sum_{i=0}^{n} P_i \left(\frac{n+1-i}{n+1}\mathbf{B}_i^{n+1}(t) + \frac{i+1}{n+1}\mathbf{B}_{i+1}^{n+1}(t) \right) \\ = & P_0\mathbf{B}_0^{n+1}(t) + \sum_{i=1}^{n} \left(\frac{n+1-i}{n+1}P_i + \frac{i}{n+1}P_{i-1} \right) \mathbf{B}_i^{n+1}(t) + P_n\mathbf{B}_{n+1}^{n+1}(t). \end{aligned} \tag{2.9}$$

Once again, similar procedures can be followed for Bézier surfaces.

It is desired to be able to subtract or add a constant to a freeform surface or curve. Because $\sum_{i=0}^{m} \mathbf{B}_i^m \equiv 1$, a K constant curve may be represented as the following Bézier curve:

$$K = K\sum_{i=0}^{m} \mathbf{B}_i^m(t) = \sum_{i=0}^{m} K\mathbf{B}_i^m(t) \tag{2.10}$$

and hence, using equations (2.6) and (2.10) constant subtraction or addition is equivalent to subtracting or adding this constant from all curve coefficients. An equivalent formulation holds for surfaces, and the NURBs representation.

A simple formulation can be defined for adding and subtracting rational curves $C_i(t) = \left(\frac{c_i^x(t)}{w_i(t)}, \frac{c_i^y(t)}{w_i(t)}, \frac{c_i^z(t)}{w_i(t)}\right) = \frac{(c_i^x(t), c_i^y(t), c_i^z(t))}{w_i(t)}$:

$$\begin{aligned} C_1(t) \pm C_2(t) &= \frac{(c_1^x(t), c_1^y(t), c_1^z(t))}{w_1(t)} \pm \frac{(c_2^x(t), c_2^y(t), c_2^z(t))}{w_2(t)} \\ &= \frac{(c_1^x(t), c_1^y(t), c_1^z(t))w_2(t) \pm (c_2^x(t), c_2^y(t), c_2^z(t))w_1(t)}{w_1(t)w_2(t)}. \end{aligned} \tag{2.11}$$

Addition of rational curves requires the capability to find products. Rational surface addition and/or subtraction may be represented in a similar way. See [27, 30] for additional details.

2.1.3 Product Representation

Although finding the symbolic derivative, sum and difference of Bézier or NURBs curves and surfaces is straightforward, finding products of curves and surfaces is more difficult. We start by considering the product of two Bézier curves and then derive the analogous formulation for surfaces.

Given two Bézier basis functions $\mathbf{B}_i^m$ and $\mathbf{B}_j^n$, their product [30, 62] is equal to:

$$\begin{aligned}\mathbf{B}_i^m(t)\mathbf{B}_j^n(t) &= \binom{m}{i}t^i(1-t)^{m-i}\binom{n}{j}t^j(1-t)^{n-j} \\ &= \binom{m}{i}\binom{n}{j}t^{i+j}(1-t)^{m+n-i-j}.\end{aligned}$$

On the other hand:

$$\mathbf{B}_{i+j}^{m+n} = \binom{m+n}{i+j}t^{i+j}(1-t)^{m+n-i-j},$$

therefore:

$$\mathbf{B}_i^m(t)\mathbf{B}_j^n(t) = \frac{\binom{m}{i}\binom{n}{j}}{\binom{m+n}{i+j}}\mathbf{B}_{i+j}^{m+n}(t). \tag{2.12}$$

Using equation (2.12), one can easily derive product formulas for curves and surfaces:

$$\begin{aligned}C_1(t)C_2(t) &= \sum_{i=0}^{m} P_i\mathbf{B}_i^m(t)\sum_{j=0}^{n} Q_j\mathbf{B}_j^n(t) \\ &= \sum_{i=0}^{m}\sum_{j=0}^{n} P_iQ_j\mathbf{B}_i^m(t)\mathbf{B}_j^n(t) \\ &= \sum_{i=0}^{m}\sum_{j=0}^{n} P_iQ_j\frac{\binom{m}{i}\binom{n}{j}}{\binom{m+n}{i+j}}\mathbf{B}_{i+j}^{m+n}(t) \\ &= \sum_{k=0}^{m+n} R_k\mathbf{B}_k^{m+n}(t)\end{aligned} \tag{2.13}$$

where:

$$R_k = \sum_{i=\max(0,k-n)}^{\min(k,m)} P_iQ_{k-i}\frac{\binom{m}{i}\binom{n}{k-i}}{\binom{m+n}{k}}.$$

The formulation for product surface follows much the same derivation:

$$S_1(u,v)S_2(u,v) = \sum_{i=0}^{m}\sum_{j=0}^{n} P_{ij}\mathbf{B}_i^m(u)\mathbf{B}_j^n(v)\sum_{k=0}^{p}\sum_{l=0}^{q} Q_{kl}\mathbf{B}_k^p(u)\mathbf{B}_l^q(v)$$

$$
\begin{aligned}
&= \sum_{i=0}^{m}\sum_{j=0}^{n}\sum_{k=0}^{p}\sum_{l=0}^{q} P_{ij}Q_{kl}\mathbf{B}_i^m(u)\mathbf{B}_k^p(u)\mathbf{B}_j^n(v)\mathbf{B}_l^q(v) \\
&= \sum_{i=0}^{m}\sum_{j=0}^{n}\sum_{k=0}^{p}\sum_{l=0}^{q} P_{ij}Q_{kl}\frac{\binom{m}{i}\binom{p}{k}}{\binom{m+p}{i+k}}\mathbf{B}_{i+k}^{m+p}(u)\frac{\binom{n}{j}\binom{q}{l}}{\binom{n+q}{j+l}}\mathbf{B}_{j+l}^{n+q}(v) \\
&= \sum_{r=0}^{m+p}\sum_{s=0}^{n+q} R_{rs}\mathbf{B}_{i+k}^{m+p}(u)\mathbf{B}_{j+l}^{n+q}(v) \qquad (2.14)
\end{aligned}
$$

where:

$$
R_{rs} = \sum_{i=\max(0,r-p)}^{\min(r,m)}\ \sum_{j=\max(0,s-q)}^{\min(s,n)} P_{i,j}Q_{r-i,s-j}\frac{\binom{m}{i}\binom{p}{r-i}}{\binom{m+p}{r}}\frac{\binom{n}{j}\binom{q}{s-j}}{\binom{n+q}{s}}.
$$

Finding the products of polynomial B-spline and NURBs is far more difficult. A direct approach has recently been developed in [49] which supports symbolic computation of the coefficients of the product after finding the knot vector. However, because it is computationally expensive and complex to implement, one might choose to exploit the uniqueness property of the process and compute the coefficients of the product by solving an interpolation problem. First, one would form the knot vector of the product, which can be derived from the knot vectors and orders of the factors. The order of the product curve, $C(t) = C^1(t)C^2(t)$, is equal to $O = O^1 + O^2 - 1$, where O^1 and O^2 are the orders of $C^1(t)$ and $C^2(t)$, respectively. The knot values and the continuity of the product curve at its knots are determined by the factor curve with the lower degree of continuity at that knot. Let ν^i be a vector holding all distinct values in τ^i, the knot vector of C^i, arranged in ascending order. At each knot value λ_j of ν^i, let $\mu^i(\lambda_j)$ be multiplicity of λ_j in C^i. Then the continuity of C^i at this knot is equal to $\mathcal{C}_j^i = O^i - \mu^i(\lambda_j) - 1$. τ^i can be decomposed into two vectors: ν^i holding all distinct values and $\mathcal{C}^i$ holding their continuities, i.e., $\tau^1 \Leftrightarrow \{\nu^1, \mathcal{C}^1\}$ and similarly $\tau^2 \Leftrightarrow \{\nu^2, \mathcal{C}^2\}$. Let ν be the merged ordered set of the distinct values from both ν^1 and ν^2, and let $\mathcal{C}$ be defined so that the mth knot $\mathcal{C}_m = min(\mathcal{C}_j^1, \mathcal{C}_k^2)$ if $\nu_j^1 = \nu_k^2$. The resulting knot vector τ will contain all the distinct values in ν with multiplicity μ equal to $\mu(\lambda_m) = O - \mathcal{C}_m - 1, \forall \lambda_m \in \nu$. The resulting knot vector, τ, is minimal in the sense that any curve $C(t)$ representing

the product $C^1(t)C^2(t)$ will have a knot vector which contains τ. One can find the *unique* B-spline curve defined over τ that interpolates $C^1(\xi_i)C^2(\xi_i)$ for all ξ_i, the node values of τ. From uniqueness, such a curve interpolates $C^1(t)C^2(t)$. This process transforms the problem into an interpolation problem, producing a set of linear equations that must be solved for the control polygon points of $C(t)$. The matrix formed is banded.

The resulting curve is unique in the sense that it minimizes the loss of continuity. Each interval between two adjacent distinct knots of τ may be represented as a polynomial, and can be represented as a Bézier segment. However, such an approach guarantees only C^0 continuity at the knots.

2.2 Numeric Computation

Knowing the zero set of a property surface and/or knowing all regions in which the property surface values are larger than some threshold or even equal to some specified value is frequently useful in extracting shape information from given curve(s) and surface(s). The ability to slice the given curve/surface with a plane is called contouring and is equivalent to finding the intersection of a curve and a line or a surface and a plane.

Contouring is used extensively to extract data from property surfaces (see Chapter 1). Once the contours are computed, their domain values in the parametric space can serve as bounding trimming curves for the original surface, $S(u, v)$, so that the trimmed surface will hold all regions in $S(u, v)$ known to have property values larger (or smaller) than the contouring level, or will contain regions bounded between two property values.

2.2.1 Contouring in E^3

This contouring process is closely related to the surface-surface intersection and ray-surface intersection [41] problems, with their inherent numerical complexities

and instabilities.

Let $F(u,v) = (\frac{x(u,v)}{w(u,v)}, \frac{y(u,v)}{w(u,v)}, \frac{z(u,v)}{w(u,v)})$ and $P = Ax + By + Cz + D = 0$ be a property surface and a contouring plane, respectively. By substituting the coordinate functions of $F(u,v)$ into P one can solve for all the values of u and v in the domain for which $F(u,v) \cap P \neq \emptyset$.

$$\begin{aligned} S(u,v) &= A\frac{x(u,v)}{w(u,v)} + B\frac{y(u,v)}{w(u,v)} + C\frac{z(u,v)}{w(u,v)} + D \\ &= \frac{Ax(u,v) + By(u,v) + Cz(u,v) + Dw(u,v)}{w(u,v)}. \end{aligned} \tag{2.15}$$

A single NURBs surface representation for equation (2.15) can be found using the operators defined in 2.1.2, 2.1.3, namely surface addition and surface multiplication. The zero set of the surface $S(u,v)$ is the set of parametric values for the required intersection. Because both $F(u,v)$ and $S(u,v)$ share the same parametric domain, mapping the parametric domain information to $F(u,v)$ is trivial. $S(u,v)$ is a *scalar* surface, which leads to a simpler and faster computation. Assuming $w(u,v) \neq 0$, the zero set of $S(u,v)$ is computed using only the numerator of $S(u,v)$. Thus, even when $F(u,v)$ is a rational surface, the contouring computation can be performed on a scalar polynomial surface.

To find the contours, the scalar surface resulting from equation (2.15) is recursively subdivided so subsurfaces intersecting the contouring plane are isolated. At each stage, the scalar surface coefficients are classified into three categories:

1. all coefficients are positive.

2. all coefficients are negative.

3. coefficients with different signs exists.

Using the convex hull property of Bézier and NURBs surfaces, it is clear the first two cases have no intersection with the contouring plane. The third case

Algorithm 2.1

Input:
$S(u,v)$, input surface.
P, the contouring plane
τ, tolerance of subdivision used in termination criteria.

Output:
Subsurfaces of $S(u,v)$, intersecting the contouring plane.

Algorithm:

```
IntersectingSubSrf( S )
begin
  M ⇐ S control mesh.
  If termination criteria hold with τ
    return { S }.
  else if M control points are all positive or all negative
    return φ.
  else
  begin
    Subdivide S into two subsurfaces S^1 and S^2.
    return IntersectingSubSrf( S^1 ) ∪ IntersectingSubSrf( S^2 ).
  end
end
```

suggests the surfaces *may* intersect and further investigation is in order, so only the third type needs further subdivision. These steps are similar to the one presented in [23, 59].

Figure 2.1 shows this first stage of isolating the subsurfaces crossing the contouring plane.

Termination criteria obviously relate to flatness testing. However, it is also required that the cross section of the patch with the contouring plane be *simple*, where

Definition 2.1 *A* simple *patch during the contouring process is a patch which intersects the contouring plane along one and only one connected*

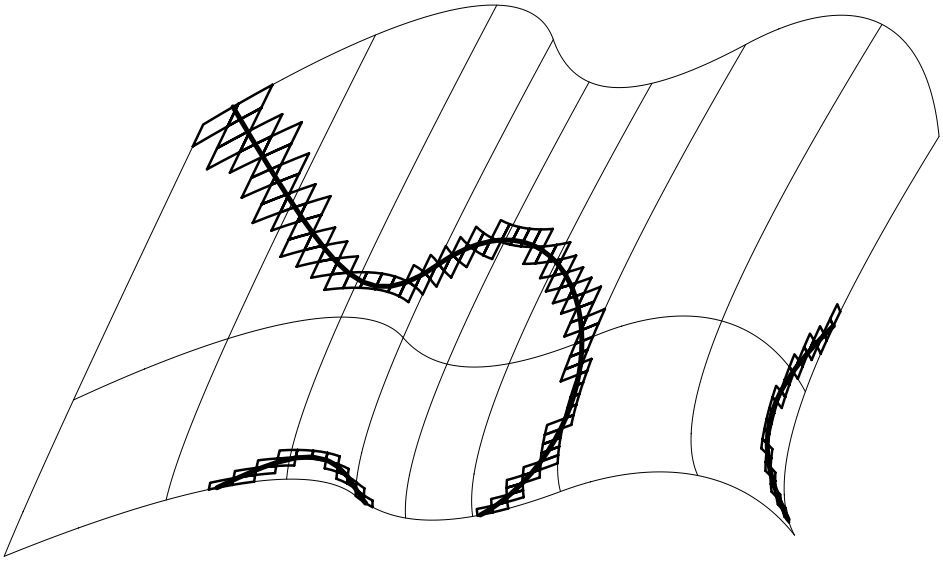


Figure 2.1. Subsurfaces intersecting the XY parallel contouring plane.

curve. Furthermore, this curve must start and end on two different boundaries of the patch.

It is clear from Figure 2.2 that tracing the patches to form a piecewise linear approximation of the contour may be ambiguous, because a single patch may have more than two (one in and one out) neighbors.

Coercing the termination condition to allow only simple patches at the lowest level simplifies the task of disambiguating and connecting the patches into a piecewise linear approximation. This termination condition also simplifies the problem of correctly identifying the contours at degenerate points such as saddles. These points can never satisfy the simplicity condition (definition 2.1), because at a saddle point four contour curves meet. The flatness criteria will terminate the subdivision, and will mark such points so they can be treated in a special manner, depending on the application.

Once traced into a a list of patches, one can pick the middle of each patch to form the piecewise linear approximation of the contouring curve. However, by using a higher order approximation on the intersection of the patch boundary and the

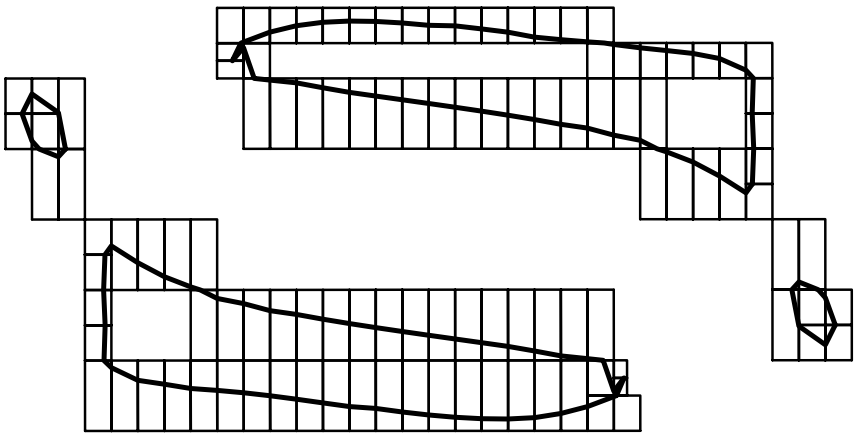


Figure 2.2. Subsurfaces chaining into piecewise linear approx. may be ambiguous.

contouring plane one can obtain a much better result (Figures 2.1 and 2.2). Because the patches are simple, a boundary crossing the contouring plane must have one of its ends above the contouring plane and the other below it. The use of a first order approximation (a linear segment connecting the two end points) to find the intersection with the contouring plane was found to be inadequate because it provided no information about the interior of the boundary curve. The subdivision approach described in 2.2.2 can be used. Numeric methods may be used as well considering that the single solution is bounded by the boundary curve end points. In practice, as in Figure 2.1, the secant method was used, which is derived in any introductory numerical analysis book, and which guarantees convergence because the patch (and boundary) are simple.

One can, at this stage, attempt to numerically improve the curve by "marching" along the surface as suggested in [23]. Furthermore, middle points may be introduced and improved as well. We found that for most purposes, as used in later Chapters, this stage was unnecessary and acceptable accuracy could be extracted in reasonable time using only subdivision. The robustness of this "marching" process has not been proven to be reliable enough.

Algorithm 2.2

Input:
$C(u)$, input curve.
τ, tolerance of output.

Output:
Zero set of $C(u)$.

Algorithm:

```
zeroSet( C )
begin
  P ⇐ C control polygon.

  If length of P, ‖P‖, is smaller than τ
    return { middle point of P }.
  else if P control points are all positive or all negative
    return φ.
  else
  begin
    Subdivide C into two subcurves C^1 and C^2.
    return zeroSet( C^1 ) ∪ zeroSet( C^2 )
  end
end
```

2.2.2 Contouring in E^2

Computation of zero sets of curves is, in general, a much simpler task because the result, for nondegenerate curves, is a finite set of points. A simple subdivision based algorithm exploiting the Convex Hull property of Bézier and NURBs curves can be easily formulated in a similar fashion [44]:

As pointed out in 2.2.1, numeric improvement is possible, but for our purposes subdivision what found more robust and sufficiently fast.

One can easily extend algorithm 2.2 to find the solution(s) of $C(u) = K$ for some constant K by defining a new curve $\hat{C}(u)$ as follows:

$$\begin{aligned} \hat{C}(u) &= C(u) - K \\ &= \sum_{i=0}^{m} P_i B_i^k(u) - \sum_{i=0}^{m} K B_i^k(u) \\ &= \sum_{i=0}^{m} (P_i - K) B_i^k(u) \end{aligned} \tag{2.16}$$

using equation (2.6) and equation (2.10).

CHAPTER 3

OFFSETS

When it is dark enough you can see the stars.

Ralph Waldo Emerson

Offset curves and surfaces are very important in manufacturing, Therefore, computation and approximation of offset curves and surfaces have undergone extensive research. For curves, the offset is an intuitive operation and has been mathematically known for more than a hundred years [6, 58, 65]. The offset operation is closed for arcs and lines, i.e., an offset of an arc and a line are an arc and a line, respectively. This is not so, in general, for Bézier and NURBs curves, so approximations are usually derived.

Two methods for finding approximations to offset curves are commonly used. The first approximates the curve using piecewise lines and arcs and then finds the representation of the exact offset to the arc and line approximation. That approach was introduced [52] and used successfully in [12]. The second method attempts to approximate the offset by directly transforming the curve representation, in particular the control points [13, 18, 28, 36, 37, 50]. To improve the accuracy of the approximation in the second method, the original curve is subdivided [36, 37, 50] or manually refined [13] when the error is above a prespecified tolerance level. The same offset technique is then applied to each of the subdivided pieces. The original curve is usually subdivided in the middle of its parametric domain [36, 37, 50], although in general, that is not the optimal location. Curve inflection points have also been considered as splitting points for offsets [37].

Both approaches do not bound the offset error globally. To bound the error introduced by the piecewise arcs and lines approximation, a curve-line and a curve-arc maximum global distance computation is required. Such computation is traditionally performed using a finite set of samples. A bound on the maximum error over the entire curve region cannot be guaranteed using such a technique. In the second method, a finite number of samples are examined to estimate the error for the entire curve region (typically one, in the middle of the parametric domain), which again cannot insure global error bound. Both methods usually result in a piecewise representation of the approximation to the offset, a more difficult representation to use in further applications if the offset is to be used as a modeling tool. Only the use of B-spline refinement [13, 15] results in a single curve. Approximations to offsets of freeform surfaces are more difficult to determine because the subdivided components are subsurfaces. Piecewise bicubic patches have been used to approximate a surface offset of a given freeform surface [29]. This method loses continuity across patches, unlike the refinement technique [13], which can be adapted for surfaces and which maintains the original continuity.

Because of the advantages of the curve/surface B-spline refinement technique, we have used this method as the basis of this implementation for bounding the global error. However, the method presented here for bounding the error is *not* limited to this type of representation.

Trimming the loops formed by the self-intersection curves of the offset is considered a difficult problem [12]. An attempt has been made to make the calculation using numerical techniques and to perform a direct search for cusps as a mean of detecting and identifying self-intersections [35]. However, an approximation to the offset may have no cusps simply because it is just an approximation. For surfaces, unidimensional successive searches have been used to isolate self-intersection points by minimizing the ratio of the Euclidean space distance (which goes to zero at

a self-intersection point) over the parametric space distance (which should be nonzero at such point) [2]. Because this method converges to a local minimum, the initial guess location is crucial but is picked at *random*. Thus, robustness is not guaranteed. Self-intersection curves have been traced using surface "walking" techniques [2] that can also be combined with the detection methods developed here.

Section 3.1 develops the method for bounding the error and then shows how to use that information to isolate the regions with maximum error. Then, we show how to apply local improvement steps iteratively, so convergence to a prespecified tolerance is assured. In section 3.2, we attempt to improve offset approximations by perturbing control points using an analysis of the error function. Section 3.3 extends this method to support a variable offset operator that can be used as a modeling tool. Section 3.4 shows how to use the tools developed in section 3.1 to robustly detect and trim loops formed by self-intersections of the offset.

3.1 A Global Bound for the Offset Operator

Let $C(t)$ be a planar regular parameterized curve, which without loss of generality, is assumed to be in the $x-y$ plane. An offset curve for $C(t)$ by an amount d is defined mathematically as:

$$\hat{\mathcal{C}}_d(t) = C(t) + N(t)d \tag{3.1}$$

where $N(t)$ is the unit normal to the curve at t. Becuase $N(t)$ flips its direction by 180^o at inflection points, a different definition for $N(t)$ should be used to define a manufacturing or design offset:

Definition 3.1 *The* offset binormal, $B_o(t)$, *to a planar curve in the* $x-y$ *plane is a unit vector in* $+z$ *direction. Then the* offset normal, $N_o(t)$, *is defined as* $N_o(t) = B_o(t) \times T(t)$, *where* $T(t)$ *is the unit tangent to the curve.*

Throughout this Chapter, and unless otherwise specified, only the offset normal, $N_o(t)$, is used:

$$\mathcal{C}_d(t) = C(t) + N_o(t)d \tag{3.2}$$

Similarly for surfaces, an offset surface for surface $S(u,v)$ by an amount d is mathematically defined as:

$$\mathcal{S}_d(u,v) = S(u,v) + n(u,v)d \tag{3.3}$$

where $n(u,v)$ is the surface unit normal to the surface at parameter values (u,v).

In this Chapter, we will concentrate on characterizing methods for the NURBs representation because the Bézier representation is a subset of it. Given two freeform NURBs curves $C_1(t)$, $C_2(t)$, their sum, difference (equation (2.6)), and product (equation (2.13)) is also a NURBs curve as seen in Chapter 2. Derivatives of NURBs curves are also NURBs curves (equation (2.4)), as are constant functions (i.e., equation 2.10).

Therefore, if $N_o(t)$ $(n(u,v))$ could be computed and represented as a NURBs, so could $\mathcal{C}_d(t)$ $(\mathcal{S}_d(u,v))$, respectively. Unfortunately, however, the general form of a normal involves a square root which is usually not representable as either a polynomial or a piecewise polynomial. Thus, offsets of freeform curves and surfaces will, in general, be approximations.

Let $\mathcal{C}_d^a(t)$ be an approximation to the offset curve of $C(t)$ by an amount d (equation (3.2)), and let $\delta(t) = \mathcal{C}_d^a(t) - C(t)$ be the difference curve. Ideally, if $\mathcal{C}_d^a(t) \equiv \mathcal{C}_d(t)$, then $\delta(t) \equiv N_o(t)d$.

Two tests could be applied to $\delta(t)$ to determine the accuracy of the offset approximation. First, the deviation of $\delta(t)$ from the direction of $N_o(t)$ could be measured, by testing whether $\delta(t)$ is orthogonal to the curve tangent. If $\hat{T}(t) = C'(t)$, then $T(t) = \frac{\hat{T}(t)}{||\hat{T}(t)||}$ is the unit tangent of $C(t)$. $\left\langle \frac{\hat{T}(t)}{||\hat{T}(t)||}, \frac{\delta(t)}{||\delta(t)||} \right\rangle$ measures the cosine of the angle between $\hat{T}(t)$ and $\delta(t)$, and is equal to zero everywhere along the exact offset

curve. However, finding $T(t)$ and $\|\delta(t)\|$ requires representing square roots, and therefore is impractical when using a piecewise rational representation. However, the square of this inner product,

$$\frac{\left\langle \hat{T}(t), \delta(t) \right\rangle \left\langle \hat{T}(t), \delta(t) \right\rangle}{\|\hat{T}(t)\|^2 \|\delta(t)\|^2}, \tag{3.4}$$

can be represented.

Although equation (3.4) is representable as a piecewise rational, it is a complex process. The equation requires at least six curve products (more if the curves are rational), each of which doubles the degree.

Instead, a second test that measures the magnitude of $\delta(t)$ can be applied to determine the accuracy of $\mathcal{C}_d^a(t)$. Computationally, it is much more attractive. Current offset techniques usually test accuracy by evaluating this magnitude on a set of sampled points. Direct representation of $\|\delta(t)\|$ would require the representation of a square root, so $\psi(t) = \|\delta(t)\|^2$ is used instead and compared with d^2:

$$\psi(t) = \|\delta(t)\|^2 = \delta_x(t)^2 + \delta_y(t)^2 + \delta_z(t)^2 \tag{3.5}$$

where $\delta_x(t)$, $\delta_y(t)$ and $\delta_z(t)$ are the components of $\delta(t)$.

Equation (3.5) can be directly represented using multiplication and addition which are computable for rationals and piecewise rationals. Hereafter, assume $\psi(t)$ can be computed and represented as a scalar NURBs curve. For exact offsets, ψ is a constant value curve equal to d^2. By subtracting d^2 from ψ, the difference curve is obtained.

$$\epsilon(t) = \psi(t) - d^2. \tag{3.6}$$

The extremal values of the coefficients of ϵ provide a global error measure. It is important to examine the consequences for computing $\epsilon(t)$ instead of $\varepsilon(t) = \|\delta(t)\| - d$, the Euclidean error between the exact offset curve and its approximation:

$$\epsilon(t) = \psi(t) - d^2 = \|\delta(t)\|^2 - d^2 = (\varepsilon(t) + d)^2 - d^2 = \varepsilon(t)^2 + 2d\varepsilon(t) \approx 2d\varepsilon(t) \tag{3.7}$$

In other words, by computing the differences of the squared magnitude, the resulting error bound is scaled by the magnitude of twice the offset distance, $2d$, which is a constant and therefore easy to control. $\varepsilon(t)^2$ has been ignored because it is much smaller than $2d\varepsilon(t)$, when the error converges to zero.

The problem of finding the global offset error has been reduced to a problem of finding the extrema of a freeform explicit curve. Because the values of a scalar B-spline curve over an interval lie between the maximum and minimum values of the coefficients of the nonzero B-spline functions, a simple and computationally efficient way of locally bounding the curve is immediately available.

The error between a C^2 continuous function and its Schoenberg variation diminishing spline approximation over a knot vector $\{t_i\}$ is $O(|\{t_i\}|^2)$, where $|\{t_i\}| = max_i\{t_{i+1} - t_i\}$. By using a sequence of Schoenberg variation diminishing spline approximations to $N_o(t)$, each one based on a knot vector that is a refinement of the previous one, and a sequence, $\{C_i(t)\}$, of refined representations to C, based on the same sequence of knot vectors, we form a convergent sequence of approximations to C_d. If the approximation is close over one interval, it is unnecessary to refine over that interval just to make the mesh norm smaller, because the approximation error is based on maximum error bounds over local regions. Hence, we need only refine over intervals where the error is large, as determined by the extrema of ϵ.

We derive an iterative algorithm in which each step uses the direct polygon transformation method [13] to compute offset approximations. The criterion for proceeding to the next step uses the magnitude of the extrema of $\epsilon(t)$. Then, the locations of the extrema are used to refine $C(t)$ (going from $C_i(t)$ to $C_{i+1}(t)$) and to create a new approximation to the offset. The process terminates when the magnitudes of the extrema of ϵ are within the tolerance.

Algorithm 3.1 retains its curve refinement history in the $C_i(t)$ sequence. The last curve in the sequence can be offset to within a provided tolerance by an amount

Algorithm 3.1

Input:
τ, required offset curve tolerance.
$C(t)$, input curve.
d, offset distance.

Output:
$C_d^a(t)$, offset curve approximation within τ accuracy.

Algorithm:
$C_0(t) \Leftarrow C(t)$.
$i \Leftarrow 0$.
Do
  Compute offset approximation $C_d^a(t)$ for $C_i(t)$.
  Compute offset error $\epsilon(t)$ for $C_i(t)$, $C_d^a(t)$.
  $C_{i+1}(t) \Leftarrow C_i(t)$ refined at $\epsilon(t)$ highest error region(s).
  $i \Leftarrow i + 1$.
While ($\epsilon(t)$ highest error $> \tau$).

d. Because the algorithm "knows" more about the curve, improvements can be applied in a more optimal way than simply subdividing the curve at its midpoint as has been done in the past. Even for polynomial representations such as Bézier curves, it is common to split the curve at the middle of the parametric domain if the accuracy of the offset is not good enough. Using the global error measure, one can now split the curve near the parameter value with the highest error. This will usually result in requiring fewer subdivisions to achieve a given tolerance.

One can compute and refine the curve at the maxima of $\epsilon(t)$ only in each iteration. However, simultaneous refinement of all regions whose respective errors were larger than allowable was found to be much faster. The computation of $\epsilon(t)$ is much more demanding than single knot insertion. By using simultaneous refinement, this computation is fully exploited.

Figure 3.1 shows four stages of algorithm 3.1, using global refinement, operating on a chess pawn cross section. Single knots have been inserted in all parametric regions whose error was above the tolerance level. The number of control points and the respective error function $\epsilon(t)$ for each iteration are also provided in Figure 3.1. The error is improved by almost an order of magnitude on each iteration up to the required tolerance of 0.0001.

Finding approximations to offsets of surfaces are usually more difficult, but the above method can be applied to finding errors of offset surfaces as well. $\delta(t)$, $\psi(t)$ and $\epsilon(t)$ would be simply explicit surfaces instead of explicit curves, i.e., $\delta(u,v)$, $\psi(u,v)$ and $\epsilon(u,v)$. In Figure 3.2, this error bounding extension surface is used to automatically iterate, refine, and improve an offset B-spline surface to a specified tolerance. It is interesting to compare the two offset surfaces in Figure 3.2. They both have the same tolerance but the offset distance is different. The offset error increases as d becomes larger and therefore more refinements are required to achieve the same accuracy.

3.2 Better Approximation of Offsets

In section 3.1, a technique was developed to provide a global bound using a global error function. This error function can be used to attempt to reduce the maximum error by perturbing the control points instead of refinement, as in section 3.1. Ideally, for each control point, the gradient direction that maximizes the change in the error function would be computed and the control point would be moved in that direction. Such a computation is extremely expensive and slow and a compromise must be made. By refining and offsetting in the normal direction, it is known the offset approximation converges to the exact offset. Therefore, the normal direction is a simple candidate for a preferred direction to use. We will also see that this

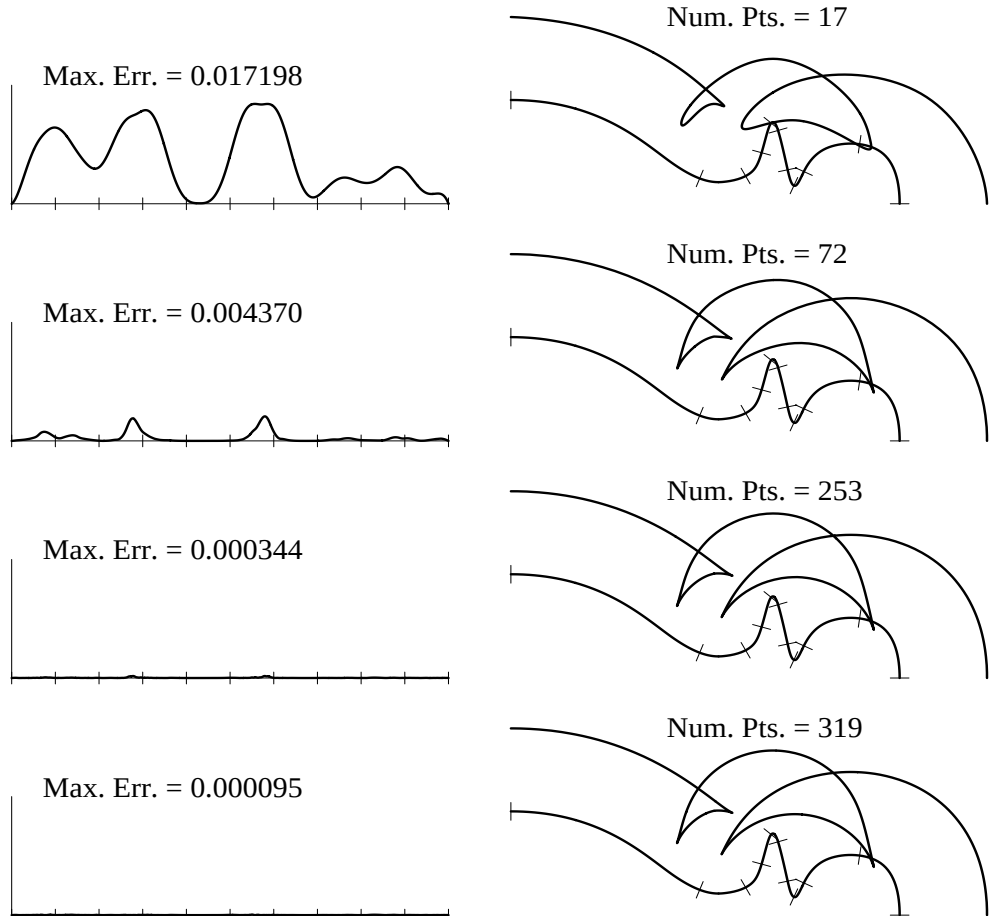


Figure 3.1. Four stages in global error bounding $\epsilon(t)$ and simultaneous refinement.

direction allows an exact representation of offset of quadratic circular curves with no refinement at all. The iterative process follows in algorithm 3.2.

In each iteration, the error function is computed and each control point is moved in the normal direction by the error amount at the node parameter value associated with this control point. This process repeats itself until no improvement is gained in the maximum error (i.e., no convergence) or the required tolerance is being achieved.

Figure 3.3 shows a unit circle composed of four 90 degree quadratic arcs. The first offset is obviously underestimated, but it converges quite quickly to the exact offset by moving only the corner points. These points have nonzero error, as can be seen from Figure 3.4 which also shows the respective error function as the process

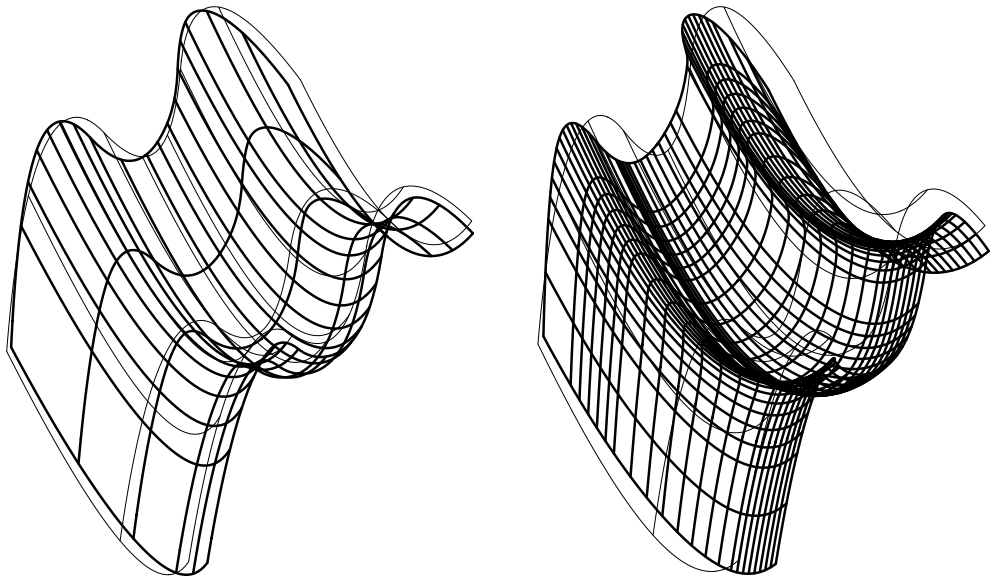


Figure 3.2. Error bounded offset surface, using simultaneous auto refinement.

converges. The points in the error function in Figure 3.4 at which the error is always zero correspond to the end points of the four 90 degrees Bézier segments forming the circle. Because the normals for the corner control point node values are in the direction (vectors $(\pm a, \pm a)$) pointing to the corner control points of a larger similarly represented circle, this process converges to an exact offset circle, with no refinement.

In Figure 3.5, the quadratic curve consists of three arcs of 120 degrees and three lines. The offset error along the line is zero and no improvement is applied there. The arcs can be improved to the exact representation. The required tolerance of 0.01 terminated this process at that accuracy as can be seen in Table 3.1.

Figure 3.6 is a case in which exact representation of the offset as a NURBs does not exist. Control points perturbation can improve the result, but refinement is still necessary to meet the required tolerance of 0.04 as can be seen from Table 3.2.

Figure 3.7 shows the same process applied to a unit sphere. This time the process does not converge to the exact representation because the normals at the node values of the corner points are not in the exact direction (vectors $(\pm a, \pm a, \pm a)$).

Algorithm 3.2

Input:
τ, required offset curve tolerance.
$C(t)$, input curve.
$C_d^a(t)$, offset approximation to input curve.
d, offset distance.

Output:
$\hat{C}_d^a(t)$, improved curve approximation.

Algorithm:
$C_0^a(t) \Leftarrow C_d^a(t)$.
$i \Leftarrow 0$.
$MaxErr \Leftarrow Infinity$.
Do
  Compute offset error $\epsilon(t)$ for $C(t)$, $C_i^a(t)$.
  $C_{i+1}(t) \Leftarrow C_i(t)$ perturbed according to $\epsilon(t)$ at node values.
  $LastMaxErr \Leftarrow MaxErr$.
  $MaxErr \Leftarrow \mathbf{min}(LastMaxErr, \epsilon(t)$ highest error$)$.
  $i \Leftarrow i + 1$.
While $(MaxErr > \tau$ and $MaxErr < LastMaxErr)$.

Even so, the improvement gained is quite significant. The right side of Figure 3.7 is the regular offset while the left side shows the same surface (and same number of control points) after perturbing it. Table 3.3 provides the convergence steps for this case, up to the prespecified tolerance of 0.01. Figure 3.7 right is stage 1 of Table 3.3 while Figure 3.7 left is stage 6.

3.3 The Offset Operator as a Modeling Tool

The offset operator can be used as a modeling tool. In fact, one can extend the global error finding method developed in section 3.1 and allow variable distance offsets as well. Given a parameter value, t, one needs to specify the offset distance required at that location. A scalar explicit distance function $d(t)$ (or $d(u, v)$ for

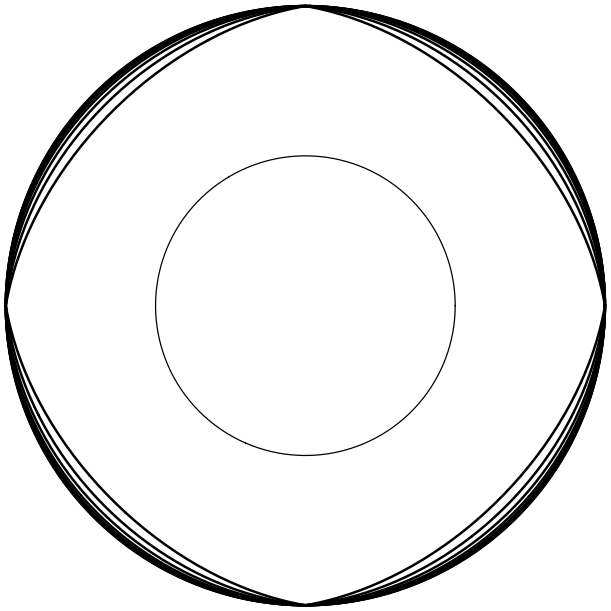


Figure 3.3. Control points perturbation converges to exact offset circle.

Figure 3.4. The error function convergence to zero, of the circle in Figure 3.3.

surfaces) having the same domain as $C(t)$ ($S(u,v)$) can be used. The only change that must be made to the method developed in section 3.1 is that equation (3.6) should now read:

$$\epsilon(t) = \psi(t) - d^2(t), \tag{3.8}$$

where d, which used to be constant, is now a distance function. In equation (3.7), it was shown that the global error bound depends on d, so now the extrema of $d(t)$ are used to bound the error. Algorithm 3.1 described in section 3.1 is identical to the one that should be used here. Figures 3.8 and 3.9 show some simple examples of the operator's power, for both curves and surfaces.

Table 3.1. Convergence errors of Figure 3.5 offset curve using perturbation.

Step	Error	Comments
1	0.49	
2	0.387	
3	0.291	
4	0.211	
5	0.149	
6	0.104	
7	0.071	
8	0.048	
9	0.033	
10	0.022	
11	0.015	
12	< 0.01	Tolerance is met.

Table 3.2. Convergence errors of Figure 3.6 offset curve using perturbation.

Step	Error	Comments
1	0.22	
2	0.197	
3	0.187	
4	0.182	
5	0.179	
$\vdots$	$\vdots$	
16	0.178	No improvement - refinement stage
21	0.083	
22	0.040	
23	< 0.04	Tolerance is met.

3.4 Trimming Self-Intersection Loops

Two types of loops are sometimes created in $C_d^a(t)$ when $C(t)$ is a C^1 continuous curve. If $\kappa(t)$, the curvature of $C(t)$, is larger than $\frac{1}{d}$, where d is the offset distance, a loop will be formed (see Figure 3.10). Because this loop is local to a region in which the curvature is too high, this type of loops will be referred to as a *local loop*. However, not all loops resulting from offset operations are of this kind. Some of the loops formed, as can be seen in Figure 3.11, are the result of two separate regions

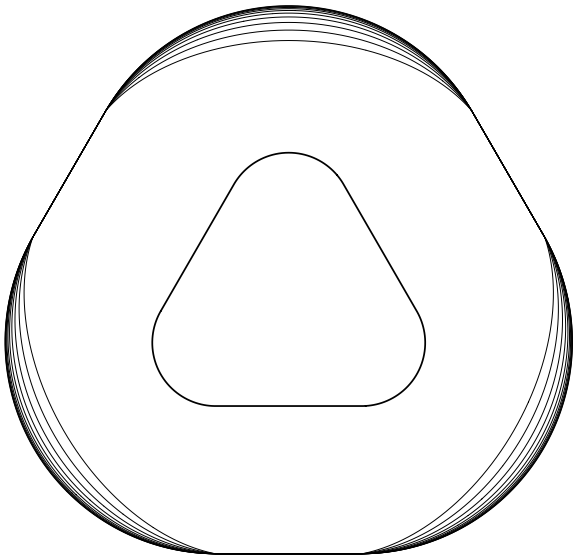


Figure 3.5. Error function convergence to zero, for three 120 degrees arcs in curve.

in $C(t)$ so close that the offset curve in those regions intersects itself. This type of loop is referred to as a *global loop*.

Detection of these loops is a difficult problem. A search for cusps was suggested as a method to detect local loops [35]. However, because $\mathcal{C}_d^a(t)$ is only an approximation, it is possible that no cusps will be formed (see first (top) stage of Figure 3.1). Moreover, the cusps, when detected, must be grouped in pairs, which is not a natural process using this technique. We use a more robust method to correctly detect all loops.

Let $\mathcal{T}(t)$ be the tangent vector to $\mathcal{C}_d(t)$ and let $\kappa(t)$ be the curvature of $C(t)$. Luckily, local loops have a distinct characteristic that when $\kappa(t_0) = \frac{1}{d}$, $\|\mathcal{T}(t_0)\| = 0$, and $\mathcal{C}_d(t)$ has a cusp at t_0 (see [28] and appendix 1). So, if $C(t)$ is curvature continuous, each time $\kappa(t) = \frac{1}{d}$ and $N(t) = N_o(t)$, $\|\mathcal{T}(t)\| = 0$. If $\kappa(t) > \frac{1}{d}$ and the normals coincide, $\mathcal{T}(t)$ flips its direction 180^o. When $\kappa(t)$ continuously changes from $< \frac{1}{d}$ to $> \frac{1}{d}$ and then back to $< \frac{1}{d}$ and the normals coincide, two cusps will be formed in $\mathcal{C}_d(t)$ at the places where $\kappa(t) = \frac{1}{d}$.

Using this characteristic, the cusp pairs can be identified by finding the zero set

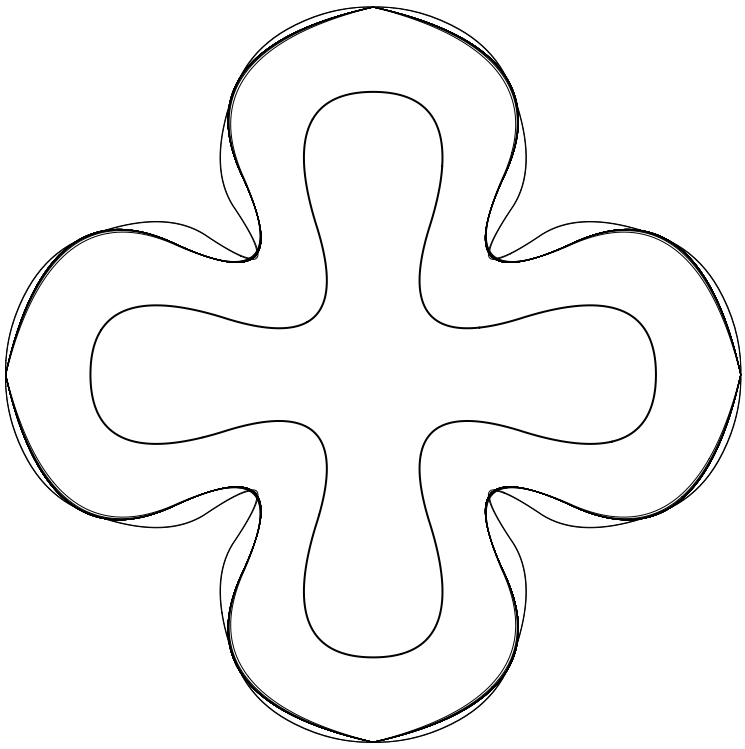


Figure 3.6. The error function does not convergence to zero, for general curves.

of $\tau(t) = \langle \mathcal{T}(t), T(t) \rangle$. The regions where $\tau(t)$ is negative are the regions where $\mathcal{T}(t)$ flips its direction (i.e., normals coincide and $\kappa(t) > \frac{1}{d}$). Figure 3.12 demonstrates this process on the pawn cross section in Figure 3.1. The tangent curves $T(t)$ ((a) in Figure 3.12) and $\mathcal{T}(t)$ ((b) in Figure 3.12) have been derived. Their dot product ((c) in Figure 3.12), $\tau(t) = \langle T(t), \mathcal{T}(t) \rangle$, is computed and used to identify the two local loops in the resulting offset approximation in its two negative regions ((d) in Figure 3.12). Once the two loops have been identified, they can be trimmed away ((e) in Figure 3.12).

The usage of $\tau(t)$ to identify local loops make this process more robust, even if no cusps are formed in the offset approximation. The tangent vector, $\mathcal{T}(t)$, still flips its direction and still makes $\tau(t)$ negative (Figure 3.12 (c)). Furthermore, by detecting the negative regions of $\tau(t)$ the cusps are virtually paired because each cusp pair is the negative $\tau(t)$ region boundary.

Table 3.3. Convergence errors of Figure 3.7 offset sphere using perturbation.

Step	Error	Comments
1	2.574	
2	1.43	
3	0.964	
4	0.804	
5	0.733	
6	0.691	
⋮	⋮	
20	0.616	No improvement - refinement stage
21	0.296	
22	0.238	
23	0.186	
24	0.146	
25	0.115	
26	< 0.01	Tolerance is met.

Once a local loop has been identified using $\tau(t)$, the algorithm splits the curve into three parts, the region before the first cusp, the region after the second cusp, and the region between the two cusps. The third part, between the cusps, must be deleted. The first two should then be intersected against each other to find the self intersection point using standard curve-curve intersection algorithms [15, 43, 60], trimmed properly to the intersection point, and then merged back. See Figures 3.10 and 3.11 for some examples.

Global loops have no such characteristic and are therefore more difficult to isolate. It is necessary to find all the self-intersections of a curve. However, a curve which is monotone in one dimension can never intersect itself. Therefore, one way to approach this problem is to split the curve into monotone subcurves, intersect all the subcurves against each other using curve-curve intersection algorithms, and isolate all the self-intersection points if any. Loops can now be formed by tracing the self-intersection points along the parameter space. Given an intersection point P_i, when $C(t_i^1) = C(t_i^2)$, the sign of the dot product $\langle T(t_i^1), N_o(t_i^2) \rangle$ can be used

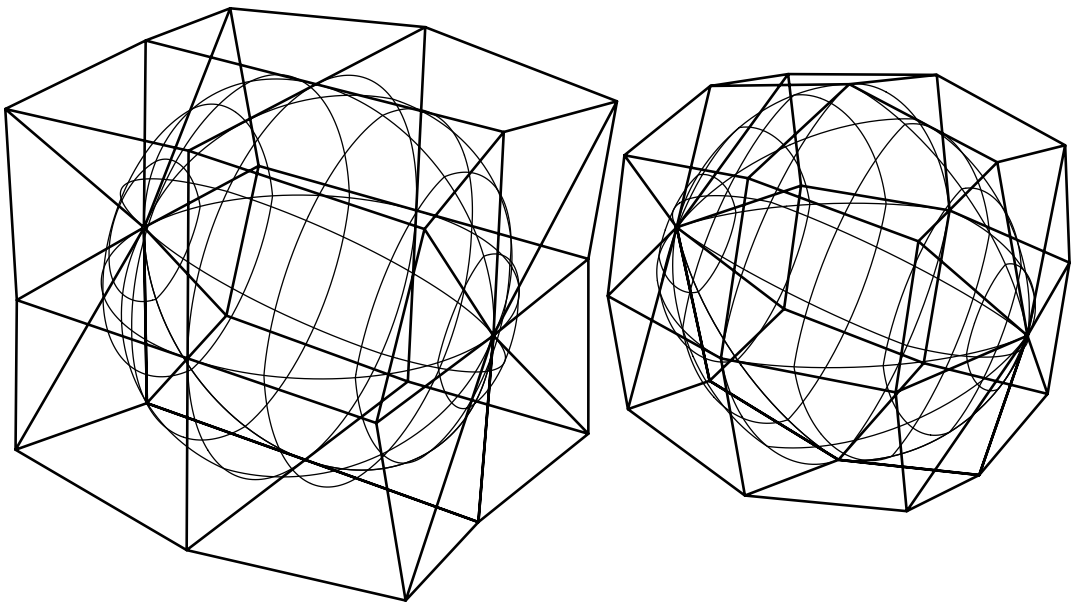


Figure 3.7. Control points perturbation can also improve offset surface accuracy.

to determine if a loop is to be purged or not. Given P_i, the normal $N_o(t_i^2)$ defines the relative position of the original and offset curve. If the dot product is negative, it means the intersecting curve (with tangent $T(t_i^1)$) in P_i is closer locally (P_4 in Figure 3.13) to the original curve than the offset amount. Because curves are continuous, it implies the whole loop is closer than the offset amount and therefore should be removed (loop 4 in Figure 3.13). Similarly, the dot product is found to be positive in P_5 in Figure 3.13 so in the neighborhood of P_5, loop 5 distance to the offset curve in the N_5 direction is larger than the offset amount and therefore loop 5 is locally (and globally) valid. The loops are tested while following the parameter values of the curve from its beginning to its end. For each intersection of an untested loop i, the tangent T_i of the current curve parameter is computed along with the offset normal N_i of the *other* curve at the intersection point i. Using the example in Figure 3.11, loop 1 is tested first. $\langle T_1, N_1 \rangle$ is found to be negative and therefore loop 1 is purged. Because $\langle T_2, N_2 \rangle$ is positive loop 2 should not be purged, etc.

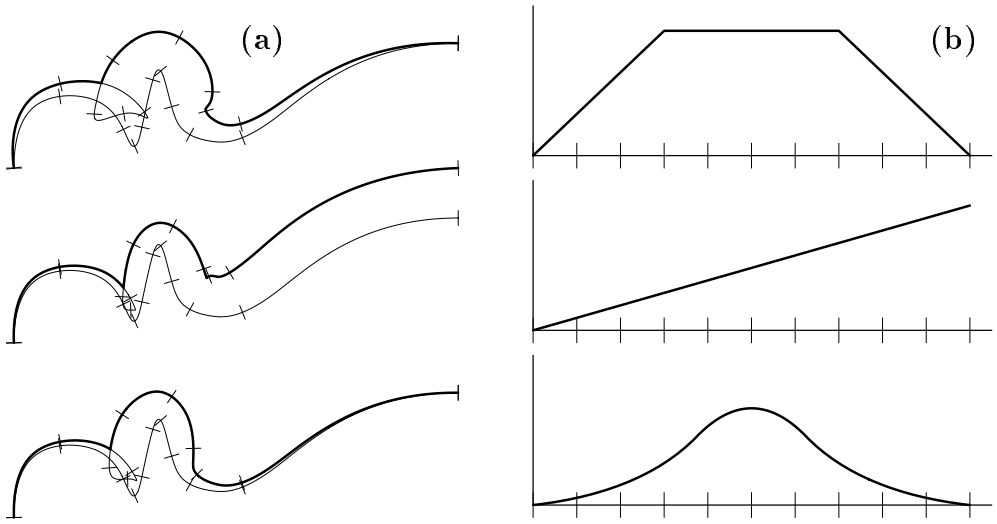


Figure 3.8. Variable distance offset (a) using a scalar distance function (b).

This approach has been used to trim out the global loops of Figure 3.11.

The curve offset local loop detection method may be extended to surfaces as well. If the surface radius is smaller than the offset distance, the normal of the offset surface may flips its direction. If both principal curvatures are the same and equal to κ (i.e., an oblique point which is locally a sphere of radius $\frac{1}{\kappa}$), then an offset by more than $\frac{1}{\kappa}$ will cause both tangents in the isoparametric directions to be flipped or the normal of the offset will point to the same direction. If, however, the two principal curvatures are different (say $\kappa_1 > \kappa_2$), the normal to the surface will be flipped when the offset distances passes $\frac{1}{\kappa_1}$, and flipped back when later the distance grows beyond $\frac{1}{\kappa_2}$. Because exact spherical shapes are fairly rare and simple to deal with, the normal flipping may be a useful tool in *detection* of self-intersections. Let $N(u,v)$ be the normal surface to the original surface $S(u,v)$ and $\mathcal{N}(u,v)$ be the normal surface to the offset surface $\mathcal{S}(u,v)$, and define

$$\nu(u,v) = \langle N(u,v), \mathcal{N}(u,v) \rangle \,. \tag{3.9}$$

Equation (3.9) can be used to detect self-intersections. If $\nu(u,v) < 0$, there must be a self-intersection. In Figure 3.14 the apex of $S(u,v)$ is an oblique point and near

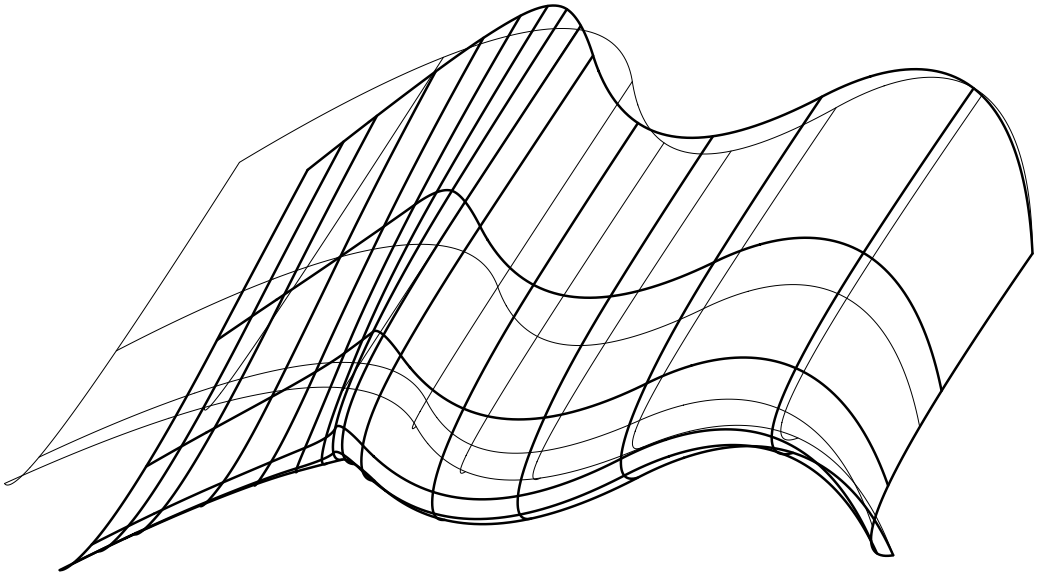


Figure 3.9. Variable distance surface offset (u direction linear, v constant).

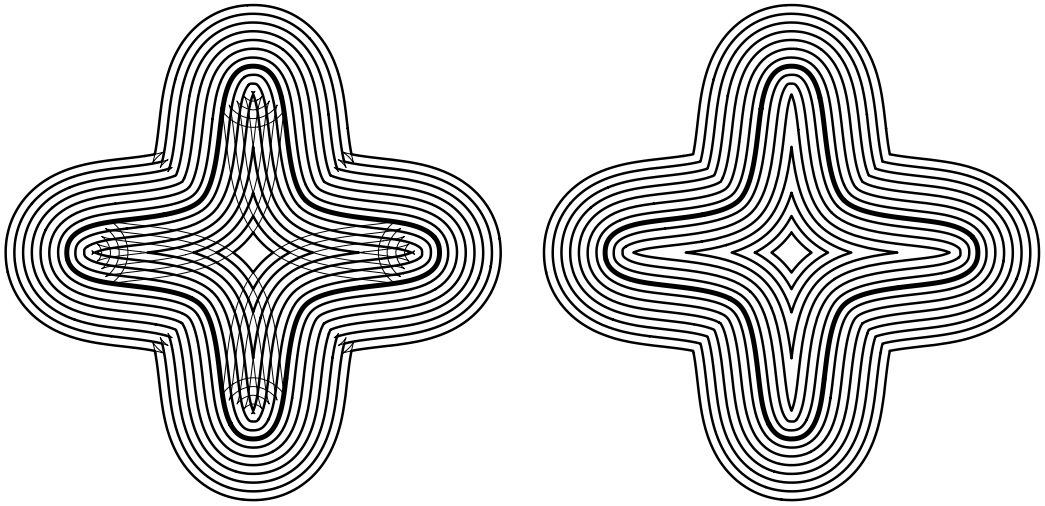


Figure 3.10. Offset operation local loops are trimmed using a distinct characteristic.

it both $N(u,v)$ and $\mathcal{N}(u,v)$ point to the same direction because both $\frac{1}{\kappa_1} < d$ and $\frac{1}{\kappa_2} < d$ or both tangents to the $\mathcal{S}(u,v)$ in the isoparametric directions are flipped. However, in the intermediate region of $\mathcal{S}(u,v)$ the u direction (surface of revolution circular direction) curvature has reached the offset distance and so the tangents in the u isoparametric direction are flipped, while the tangents in the v isoparametric direction are not. In that region, obviously $\frac{1}{\kappa_1} > d > \frac{1}{\kappa_2}$ and ν is negative, which

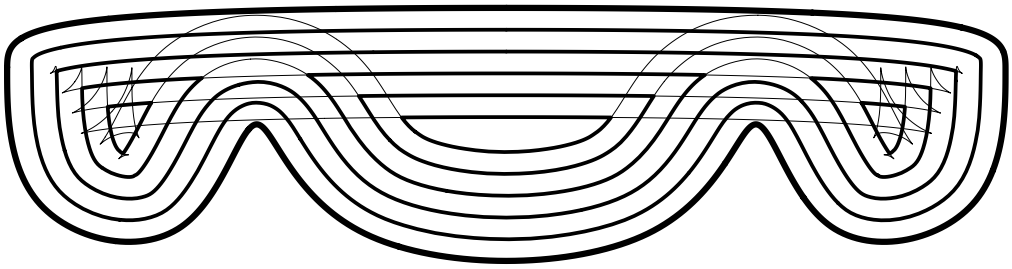


Figure 3.11. Global loop are being trimmed using numerical techniques.

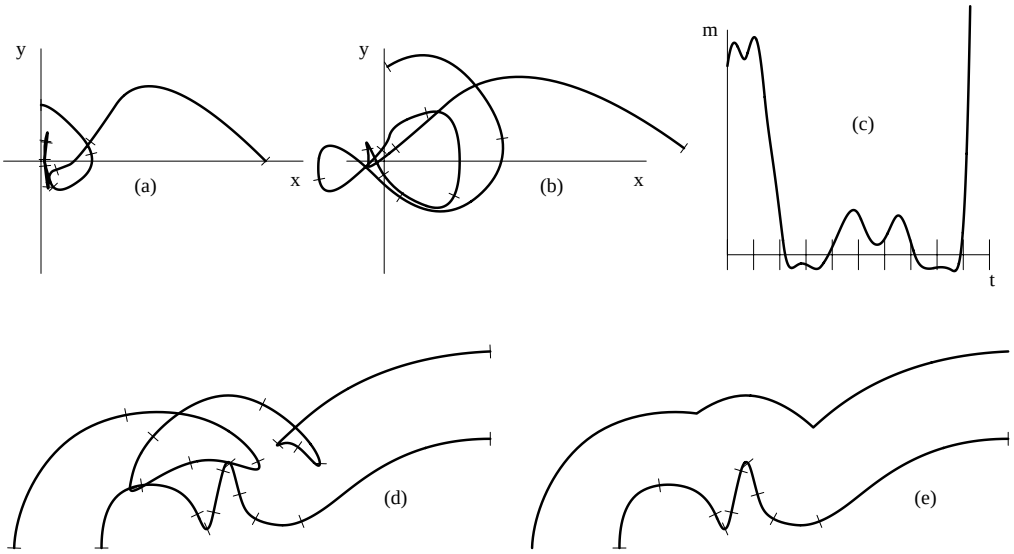


Figure 3.12. Product of a curve and its offset tangents used to identify local loops.

together signal the self-intersection.

Trimming surface loops are much more difficult because, in general, they are not isoparametric. Because an analytic approach was not feasible, an approach which subdivided the surface into polygons and detected self-intersections on this approximation was used. To simplify the process, it was assumed the the original surface was completely visible from the z direction (envisioning a 3 axis pocket for NC applications). The z was used as the sweeping axis, in algorithm 3.3, to minimize the number of polygon-polygon intersection tests, in the detecting of possible self-intersections in the offset. **minimumZ** and **maximumZ**, in algorithm 3.3,

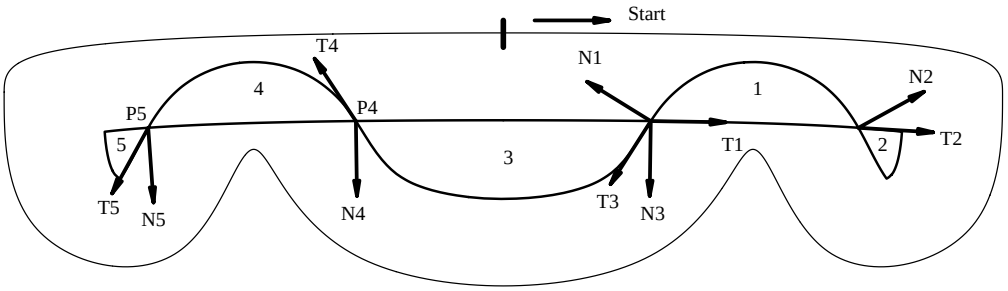


Figure 3.13. Global loop classification is based on $\langle N_i(t_i^1), T_i(t_i^2)\rangle$ sign.

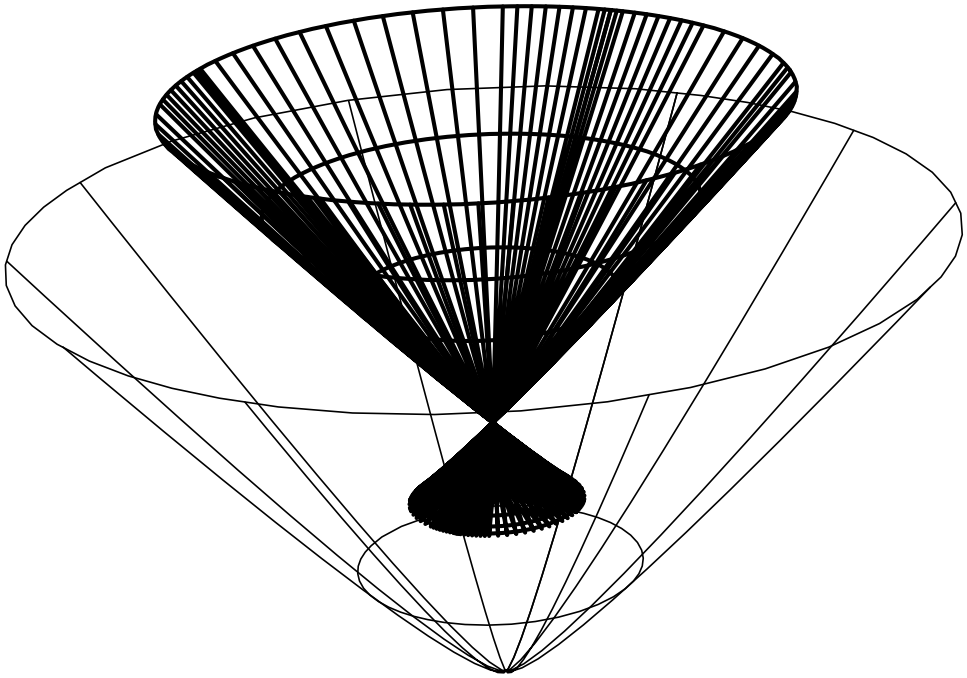


Figure 3.14. Offset surface self-inter. may be detected using $\langle N(u,v), \mathcal{N}(u,v)\rangle$ sign.

return the z extrema of given object, **IntersectPolyPoly** finds the linear segment of two intersecting polygons, and finally **isSurface** and **isFlat** are two predicates.

As if these difficulties are not enough, the topology of the self-intersection can be extremely complex. Figure 3.15 left shows an offset surface of a simple surface. The centered region of the surface has large enough curvature to cause the offset surface to intersect itself. Extremely complex self-intersection loops are generated as can be seen on the right of Figure 3.15 which shows the self-intersection curves

Algorithm 3.3

Input:
τ, tolerance for subdivision control.
$\mathcal{S}(u,v)$, an offset surface, possibly self-intersecting.

Output:
$\mathcal{L}$, a piecewise linear representation of the self intersection curves.

Algorithm:
$\mathcal{Q} \Leftarrow \mathcal{S}(u,v)$, a priority queue holding sorted data in z according to minimum z of elements.
$\mathcal{P} \Leftarrow \emptyset$, a set of all active polygons.
$\mathcal{L} \Leftarrow \emptyset$.
While ($\mathcal{Q} \neq \emptyset$)
begin
  $Obj \Leftarrow first(\mathcal{Q})$.
  $z \Leftarrow$ **minimumZ**(Obj).
  if (**isSurface**(Obj))
    if (**isFlat**(Obj, τ))
      Convert to polygons, and for each polygon P_i Do
        $\mathcal{L} \Leftarrow \mathcal{L} \cup$ **InterActiveList**$(P_i, z, \mathcal{P}, \mathcal{Q})$.
    else
      Subdivide into two subsurfaces and insert both to $\mathcal{Q}$.
  else /* Its a polygon */
    $\mathcal{L} \Leftarrow \mathcal{L} \cup$ **InterActiveList**$(Obj, z, \mathcal{P}, \mathcal{Q})$.
end

in the parameter space of the surface.

Removal of self-intersections in surface offsets, is not totally solved and should be further investigated. A complete study of the complex topology of the self-intersection curves may provide some leads.

Algorithm 3.3 continued

InterActiveList$(P, z, \mathcal{P}, \mathcal{Q})$
```
begin
  M ⇐ ∅, holding all self-intersections with polygon P.
  if ( minimumZ( P ) > z )
    Insert P to Q.
  else
  begin
    For each polygon P_i in P do
    begin
      if ( maximumZ( P_i ) < z )
        remove( P_i, P ).
      else
        M ⇐ M ∪ IntersectPolyPoly(P, P_i).
    end
    Insert P to P.
  end
  return M.
end
```

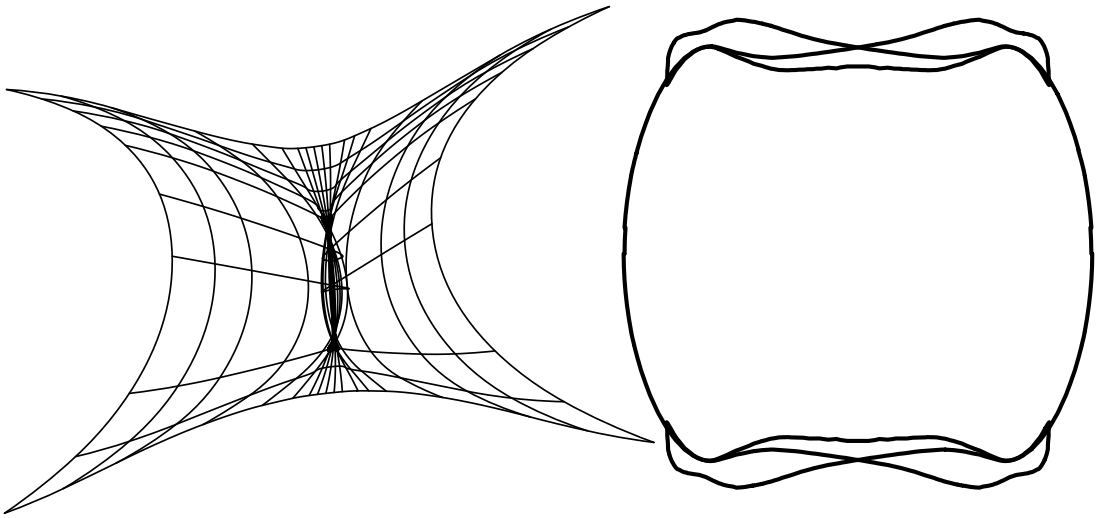


Figure 3.15. Offset surface self-intersection can be topologically complex.

CHAPTER 4

SECOND ORDER SURFACE ANALYSIS

It is our purpose to give a presentation of geometry, as it stands today, in its visual, intuitive aspects. With the aid of visual imagination we can illuminate the manifold facts and problems of geometry, and beyond that, it is possible in many cases to depict the geometric outline of the methods of investigation and proof, without necessarily entering into the details connected with the strict definions of concepts and with the actual calculations.

D. Hilbert, in "Geometry and the Imagination," 1932.

A critical characteristic for many applications in computer graphics and in CAD is the shape of the model's bounding surfaces. Second order surface analysis can be used to understand curvature characteristics, and thus shape, and to improve the implementation, efficiency and effectiveness of manufacturing and analysis processes. Fundamental operations, such as adaptive subdivision and refinement, use shape information to decide where and how many knots to add. Algorithms for the creation of tool paths for NC (Numerically Controlled) code generation for freeform surfaces are usually based on ball end cutters with their spherical centers following an (approximate) offset surface of the original surface. Flat end cutters can remove material faster and have a better finish; however, flat end cutters can be used only with 5 axis milling in convex regions (see Figure 4.1).

Definition 4.1 *A* surface trichotomy *is a partition of a surface into three types of regions: convex, concave and saddle shapes (Figure 4.1).*

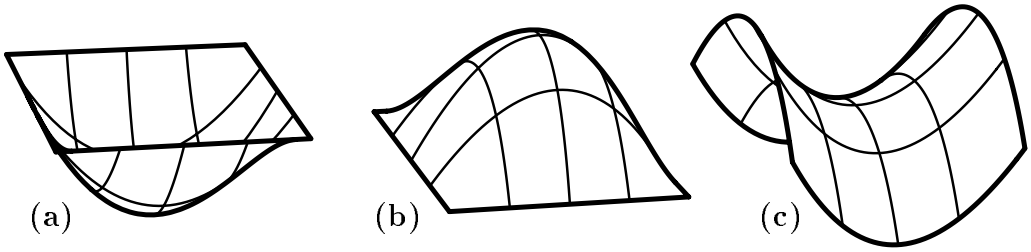


Figure 4.1. Mainly concave (a), convex (b), and saddle (c) regions.

The ability to trichotomize sculptured surfaces into convex, concave or saddle regions (Figure 4.1) is thus essential to the use of flat end cutters in milling freeform surfaces. Also, regions with small curvature can be accurately milled faster with larger ball end cutters. Tool changes should be minimized because the are time consuming operations. Such minimization can be achieved by subdividing the surface into regions with different curvature bounds, each of which can be milled using tools appropriate to that region.

Methods in use do not support the separation of original surfaces into trimmed surfaces each of which with only one of the three characteristics throughout. That is, each trimmed surface is either convex everywhere, concave everywhere, or saddle everywhere. Second order surface properties are usually estimated locally by numerically evaluating them at a grid of points or, in manufacturing, at a finite set of sampled points along a planned milling tool path. Research into the computation of curvature has been done in the context of offset operator approximations with cubic B-spline curves [66] and bicubic patches [29].

There have been attempts [3, 4, 20, 33] to understand and compute second order surface properties as well as twist by evaluation on a predefined grid. The methods use the Gaussian curvature $K(u,v) = \kappa_n^1(u,v)\kappa_n^2(u,v)$ and mean curvature $H(u,v) = \frac{\kappa_n^1(u,v)+\kappa_n^2(u,v)}{2}$, where $\kappa_n^1(u,v)$ and $\kappa_n^2(u,v)$ are the *principal curvatures* at the parameter value (u,v), in an attempt to provide a bound on the surface angularity. However, if the surface is a saddle at (u,v), then κ_n^1 and κ_n^2 have

different signs so the magnitude of H is not a useful measure of such a bound. In the extreme condition when the surface is minimal [21], $H \equiv 0$ regardless of the surface angularity. The magnitude of K can also be ineffective. Even if κ_n^1 is large, K may be small because κ_n^2 is small. Therefore, neither K nor H by itself can provide sufficient shape information for subdivision and/or efficient NC applications. This problem has been recognized by some of the authors cited above. These curvature estimation techniques are *local*, because they make use of local surface information only. More surface information might improve an algorithm or change a decision. Local information is inferior to global information in complex settings. Symbolic techniques can be used to help make decisions based upon the entire aspect of a surface rather than a limited number of local samples.

In this Chapter, a hybrid approach using both symbolic and numeric operations for computing curvature properties is developed. We use *property surfaces* (see definition 1.1) whose definitions are derived from different attributes of the original surface, as auxiliary surfaces to help analyze the original surface.

Section 4.1 briefly develops the differential geometry used in the analysis. In section 4.2, we compute second order properties, and use visualization to better understand the shape of a given surface.

4.1 Differential Geometry

Surface curvature is well understood mathematically and the theory behind it is developed in most introductory differential geometry books [21, 48, 63]. The set of analysis equations that are based on the second fundamental form are used extensively in locally evaluating surface curvature. Because these equations are crucial to our discussion, they are briefly stated here.

Let $F(u,v)$ be a $C^{(2)}$ regular parametric surface. Let the *unnormalized normal* to a surface $F(u,v)$, $\hat{n}(u,v)$, be defined as

$$\hat{n}(u,v) = \frac{\partial F}{\partial u} \times \frac{\partial F}{\partial v}, \tag{4.1}$$

and define the surface unit normal, $n(u,v)$, to be

$$n(u,v) = \frac{\frac{\partial F}{\partial u} \times \frac{\partial F}{\partial v}}{\left\| \frac{\partial F}{\partial u} \times \frac{\partial F}{\partial v} \right\|}. \tag{4.2}$$

Because $F(u,v)$ is regular, $\|\hat{n}(u,v)\| \neq 0$ and $n(u,v)$ is well defined.

Let $C(t) = F(u(t), v(t))$ be a regular curve on F, that is $\left\| \frac{dC(t)}{dt} \right\| \neq 0$. The rate of change of the arc length of C with respect to its parameter, t, is $\frac{ds}{dt} = \left\| \frac{dC(t)}{dt} \right\|$ where s is arc length. Because $\frac{dC(t)}{dt} = \left(\frac{\partial F}{\partial u}\frac{du}{dt} + \frac{\partial F}{\partial v}\frac{dv}{dt} \right)$, one can show [32, 48, 63] that

$$\left(\frac{ds}{dt}\right)^2 = \begin{bmatrix} \frac{du}{dt} & \frac{dv}{dt} \end{bmatrix} G \begin{bmatrix} \frac{du}{dt} & \frac{dv}{dt} \end{bmatrix}^T = I\left(\frac{du}{dt}, \frac{dv}{dt}\right).$$

I is known as the first fundamental form, with matrix G equal to:

$$G = (g_{ij}) = \begin{bmatrix} \left\langle \frac{\partial F}{\partial u}, \frac{\partial F}{\partial u} \right\rangle & \left\langle \frac{\partial F}{\partial u}, \frac{\partial F}{\partial v} \right\rangle \\ \\ \left\langle \frac{\partial F}{\partial v}, \frac{\partial F}{\partial u} \right\rangle & \left\langle \frac{\partial F}{\partial v}, \frac{\partial F}{\partial v} \right\rangle \end{bmatrix}. \tag{4.3}$$

By considering all such curves, $C(t)$, through a point (u,v) and differentiating twice, one can extract second order properties of the surface F at (u,v). The second order derivatives of $C(t)$ contain terms with $\frac{\partial F}{\partial u}$ and $\frac{\partial F}{\partial v}$ as factors. However, the inner product of these terms with n is always zero because the partials are in the tangent plane of $F(u,v)$. Therefore, $\left\langle n(u,v), \frac{d^2C(t)}{dt^2} \right\rangle$, the component of $\frac{d^2C(t)}{dt^2}$ pointing in the direction perpendicular to the surface is composed of second order derivatives only.

$$\begin{aligned} & \left\langle n(u,v), \frac{d^2C(t)}{dt^2} \right\rangle \\ = \; & \left\langle n(u,v), \frac{\partial^2 F}{\partial u^2} \right\rangle \left(\frac{du}{dt}\right)^2 + 2\left\langle n(u,v), \frac{\partial^2 F}{\partial u \partial v} \right\rangle \frac{du}{dt}\frac{dv}{dt} + \left\langle n(u,v), \frac{\partial^2 F}{\partial v^2} \right\rangle \left(\frac{dv}{dt}\right)^2 \\ = \; & \begin{bmatrix} \frac{du}{dt} & \frac{dv}{dt} \end{bmatrix} L \begin{bmatrix} \frac{du}{dt} & \frac{dv}{dt} \end{bmatrix}^T \end{aligned}$$

$$= \quad II\left(\frac{du}{dt},\ \ \frac{dv}{dt}\right). \tag{4.4}$$

II is known as the second fundamental form, with matrix L equal to:

$$L = (l_{ij}) = \begin{bmatrix} \left\langle n, \frac{\partial^2 F}{\partial u^2}\right\rangle & \left\langle n, \frac{\partial^2 F}{\partial u \partial v}\right\rangle \\ \\ \left\langle n, \frac{\partial^2 F}{\partial u \partial v}\right\rangle & \left\langle n, \frac{\partial^2 F}{\partial v^2}\right\rangle \end{bmatrix}. \tag{4.5}$$

Let $\hat{l}_{ij}$ denote the inner product with the unnormalized normal $\hat{n}(u,v)$. For example, $\hat{l}_{11} = \left\langle \hat{n}, \frac{\partial^2 F}{\partial u^2}\right\rangle$.

The normal curvature on the surface $F(u,v)$ in some tangent direction Δ, where $\Delta = \left\langle \delta, \left(\frac{dF}{du},\ \ \frac{dF}{dv}\right)\right\rangle$, and $\delta = \left(\frac{du}{dt},\ \ \frac{dv}{dt}\right)$, is defined [21, 32, 48, 63] as:

$$\kappa_n = \frac{II(\frac{du}{dt}, \frac{dv}{dt})}{I(\frac{du}{dt}, \frac{dv}{dt})} = \frac{\delta L \delta^T}{\delta G \delta^T}. \tag{4.6}$$

The normal curvature depends on the surface tangent direction Δ, and is equal to the curvature of the osculating circle to the intersection curve between $F(u,v)$ and the plane through $n(u,v)$ and Δ at (u,v) (Figure 4.2). The extremal values of the normal curvature serve as bounds on the components of curvature not in the tangent plane.

The normal curvature is an *intrinsic property* [48, 63] of the surface. By differentiating (4.6) with respect to δ, the problem of finding extrema of κ_n is transformed [21, 32, 48, 63] into the problem of solving for the roots of

$$|G|\,\kappa_n^2 + (g_{11}l_{22} + l_{11}g_{22} - 2g_{12}l_{12})\kappa_n + |L| = a\kappa_n^2 + b\kappa_n + c = 0, \tag{4.7}$$

where $|G|$ and $|L|$ denotes the determinants of G and L, respectively.

The Gaussian curvature is a scalar value and is defined as the product of the two roots of (4.7), κ_n^1 and κ_n^2,

$$K = \kappa_n^1 \kappa_n^2 = \frac{|L|}{|G|}. \tag{4.8}$$

The mean curvature is defined as their arithmetic average,

$$H = \frac{\kappa_n^1 + \kappa_n^2}{2} = -\frac{(g_{11}l_{22} + l_{11}g_{22} - 2g_{12}l_{12})}{2\,|G|}. \tag{4.9}$$

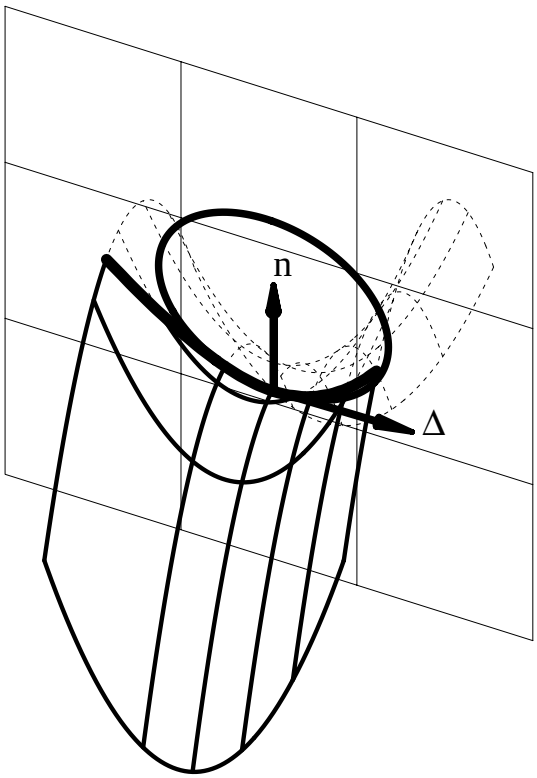


Figure 4.2. Normal curvature κ_n (circle) of $F(u,v)$ at (u,v) in direction Δ.

4.2 The approach

The tools defined in Chapter 2 are used symbolically to compute the second order properties of a given surface as described in Section 4.1. NURBs property surfaces are derived whenever possible so that the method can take advantage of the computational characteristics of NURBs.

4.2.1 Surface Trichotomy

Use of the curvature trichotomy of a surface can result in a more optimal freeform surface milling process. Only convex regions (see Figure 4.1) are millable using flat end cutters and 5 axis milling. Flat end cutters, as opposed to ball end cutters, can mill faster and remove more material per time unit. Furthermore, the surface finish of flat end cutters is usually better. Using the trichotomy operator, convex

regions within surfaces can be detected and milled in more efficient way and with a better finish.

The determinant of L, $|L|$, in (4.7) is the key to this second order surface analysis. If $|L| = 0$, one of the normal curvature extrema κ_n^i must be zero. Assuming the surface is curvature continuous, adjacent regions for which κ_n^i has a different sign must be separated by a curve, C_s, for which $|L| = 0$, that is, one of the $\kappa_n^i = 0$. Furthermore, if $|L| > 0$ at some point p on the surface F, the surface is either convex or concave at p, while if $|L| < 0$ the surface locally is a saddle. In order to compute a property surface representing $|L|$ using (4.5), it is necessary to find a square root to compute $n(u, v)$, which cannot be represented, in general, as a polynomial or as a piecewise rational. However, by reordering the operations to use the unnormalized surface normal $\hat{n}(u, v)$ and noting $n(u, v)$ appears twice as a factor in each term of $|L|$, $|L|$ can be represented exactly as a rational function and with no square roots,

$$|L| = \frac{\hat{l}_{11}\hat{l}_{22} - \hat{l}_{21}\hat{l}_{12}}{\|\hat{n}\|^2}. \tag{4.10}$$

This equation is representable as a NURBs using only operations from Chapter 2. $\hat{n}$ is a cross product of two surface partials $\frac{\partial F}{\partial u}$ and $\frac{\partial F}{\partial v}$. The components of L, $\hat{l}_{ij}$, are inner products of $\hat{n}$ with second order partials of F. Because only the zero set is of interest, and F is assumed to be a regular surface, it is necessary to examine only the numerator of (4.10). Once the zero set of $|L|$ has been computed, trimmed surfaces are created, each of which is completely convex, concave or saddle. The sign of $|L|$ at a single point on each trimmed surface is then used to classify the saddle regions while convex and concave regions are distinguished from each other by simply evaluating the sign of $\hat{l}_{11}$, for example, at that single point. Whereas the saddle region is an intrinsic surface characteristic, the convex/concave classification is parameterization dependent. Flipping the u or v (but not both) surface param-

eterization direction will flip the normal direction $n(u, v)$ and therefore the sign of $\hat{l}_{11}$.

Figures 4.3 through 4.7 show some examples. Figure 4.3 is a biquadratic B-spline surface with three internal knots in each direction (patches of a B-spline surface are counted as how many Bézier patches would result from subdividing the NURBs surface at each interior knot, so this surface yields 16 polynomial patches), while Figure 4.4 is a single biquadratic patch. The bicubic surfaces in Figures 4.5 and 4.6 have two internal knots in each direction, yielding 9 polynomial patches. Figure 4.7 top is a bicubic NURBs surface with a single internal knot in each direction, yielding four Bézier patches. All figures have been colored consistently, with yellow marking the saddle regions, red representing a convex region and green representing a concave region.

The biquadratic surface of Figure 4.3 is not C^2 along each internal knot, and the surface trichotomy is isoparametric along the internal knots lines.

However, in general, this behavior should not be expected, or even anticipated, for biquadratic surfaces, because even a single biquadratic patch may contain both convex and saddle regions simultaneously as shown in Figure 4.4.

The surface in Figure 4.5 uses the same control mesh as the one in Figure 4.3 but is bicubic. Both surfaces in Figure 4.3 and Figure 4.5 use appropriate uniform open end condition knot vectors. A comparison of these two Figures graphically demonstrates the influence of the *order* of the tensor product spline surface on the shape, as shown by comparing the shapes and locations of the convex and concave regions. This phenomenon is somewhat counterintuitive to the common belief that two NURBs surfaces with the same mesh but different order are very similar, except that the one with higher order is a smoother version. The curvature characteristics have actually been changed. Figure 4.3 has one concave region, one convex region and two flat regions, all of which have isoparametric boundaries.

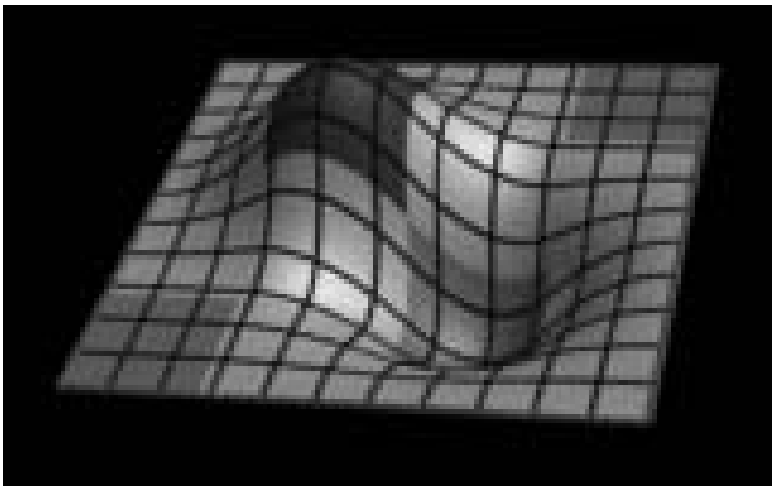


Figure 4.3. Biquadratic surface trichotomy with 16 polynomial patches.

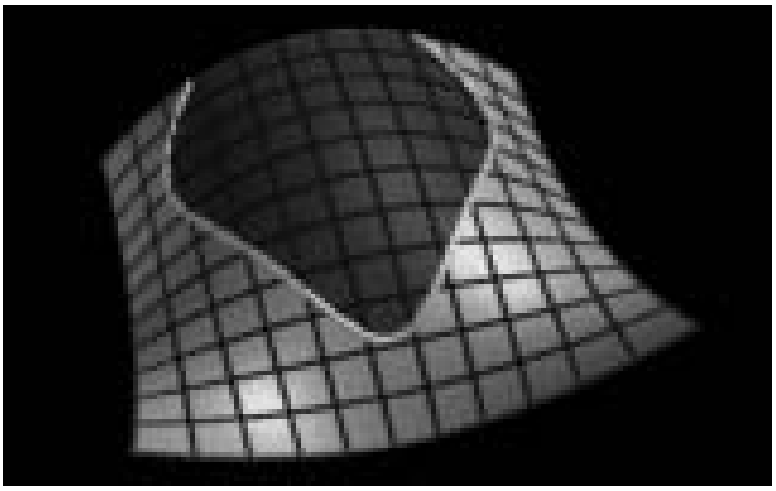


Figure 4.4. Biquadratic polynomial trichotomy.

Figure 4.5, however, has only one concave region and one convex region. The union of the two regions has a figure eight boundary, where convex and concave change at a single point. The curved boundaries of those regions are different from the straight line boundaries in Figure 4.3.

Figure 4.6 shows that the combination of symbolic computation (of $|L|$ as a property surface) with numeric analysis (contouring the property surface) can detect widely separated and isolated regions. In addition, it demonstrates the robustness of this methodology by accurately detecting two very shallow concave

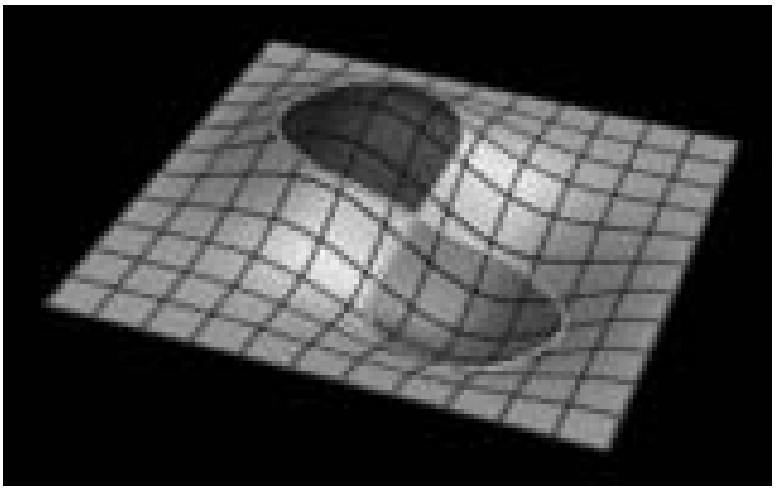


Figure 4.5. Bicubic surface trichotomy: same control mesh as Figure 4.3.

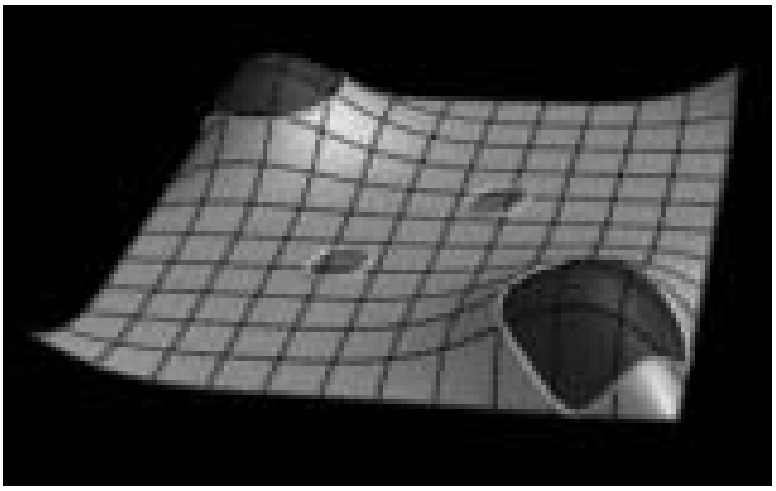


Figure 4.6. Bicubic with isolated convex and concave regions in a saddle region.

regions in the middle of the surface. In Figures 4.5 and 4.7, another ill conditioned case is shown in which several convex and concave regions meet at a single point. Because trimmed surfaces are formed, it was necessary that the boundaries be completely and correctly defined. The points where the three regions meet are correctly detected and determined and the topology of the regions is correctly maintained, which also demonstrates another type of robustness.

To provide a better sense of the process, the bottom of Figure 4.7 also shows the scalar property surface of the determinant of the second fundamental form, $|L|$,

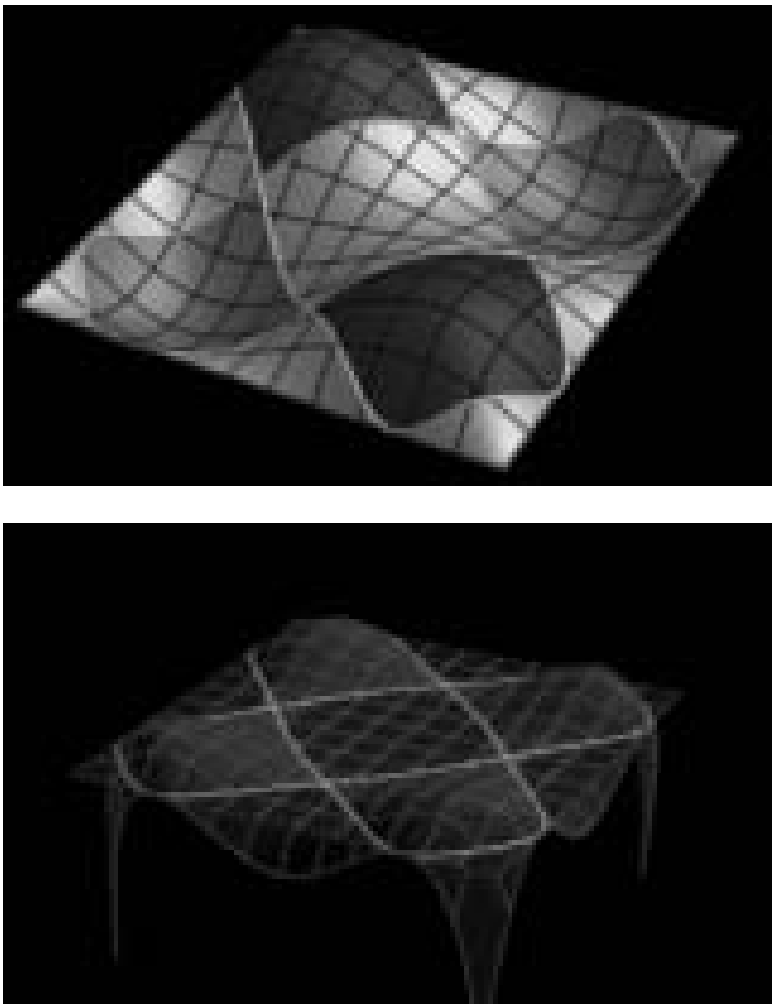


Figure 4.7. Bicubic surface with convex and concave regions meet at a single point (top). The surface second fundamental form property surface and its zero set (bottom).

with its zero set, as a function of u and v.

Figure 4.8 demonstrates this method on a more realistic object. The Utah teapot trichotomy degenerated into a dichotomy because no concave regions exist in the teapot model.

It is interesting to note that a sufficient condition for a surface to be developable [32] is that its Gaussian curvature is always zero: $K(u,v) \equiv 0$. Because $K(u,v) = \frac{|L|}{|G|}$, this condition is equivalent to the condition that $|L| \equiv 0$ for regular surfaces were $|G| \neq 0$. Hereafter, a simple practical test that can answer whether

Figure 4.8. Teapot trichotomy degenerates into a ditochomy (no concave regions).

a surface is developable or not can be derived by symbolically computing $|L|$ and comparing all its coefficients to zero. Figure 4.9 show two developable NURBs surfaces. The top one is ruled surface along an isoparametric direction while the bottom one was bent along *nonisoparametric* direction.

4.2.2 Bounding the Curvature

The extrema of the surface curvature are important for analyzing the curvature of a given surface. Normal curvature extrema occur in the principal directions [32, 48, 63], but the direct application of quadratic equation solution for equation (4.7) would require finding a square root. However, because the surface has been subdivided into convex, concave, and saddle regions, each region carries the following property:

- If the region has a saddle shape, then one of the principal curvatures, κ_n^1, is positive while the other, κ_n^2, is negative.
- If the region is convex both principal curvatures are negative.
- If the region is concave both principal curvatures are positive.

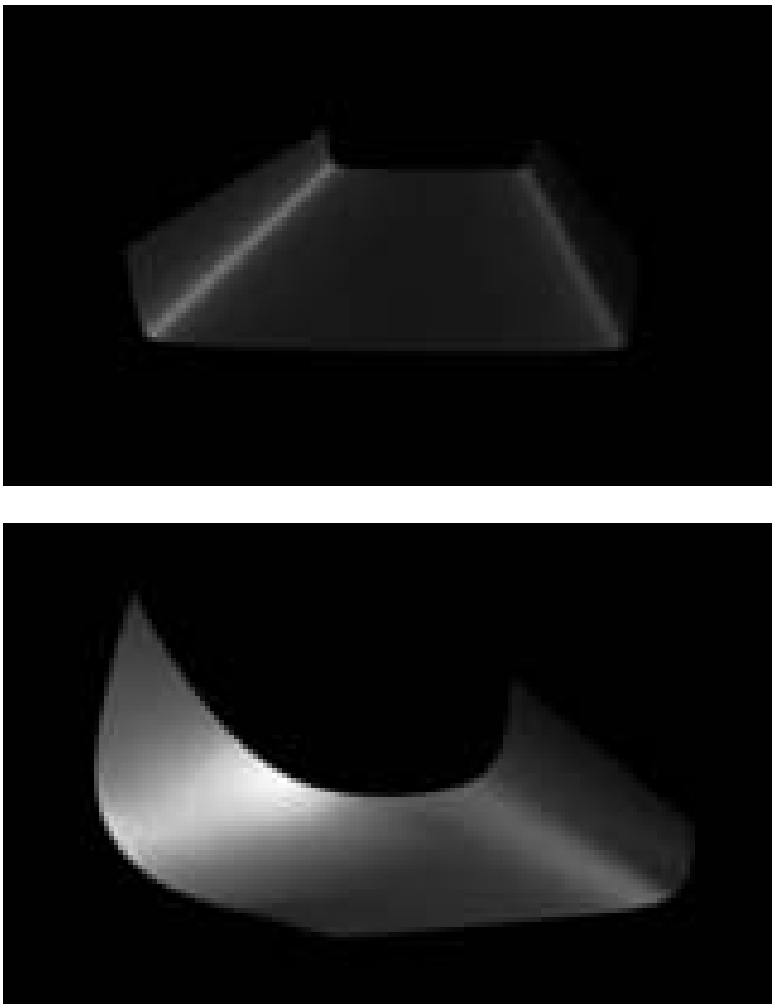


Figure 4.9. Two ruled surface examples.

Using quadratic equation properties for equation (4.7), it can easily be shown that:

$$\begin{aligned}
\psi &= (2H)^2 \\
&= \left(\kappa_n^1 + \kappa_n^2\right)^2 \\
&= \left(-\frac{b}{a}\right)^2 \\
&= \left(-\frac{g_{11}l_{22} + l_{11}g_{22} - 2g_{12}l_{12}}{|G|}\right)^2 \\
&= \frac{\left(g_{11}\hat{l}_{22} + \hat{l}_{11}g_{22} - 2g_{12}\hat{l}_{12}\right)^2}{|G|^2 \|\hat{n}\|^2}
\end{aligned} \tag{4.11}$$

and

$$\begin{aligned}
\phi &= (\kappa_n^1 - \kappa_n^2)^2 = \frac{b^2 - 4ac}{a^2} \\
&= \frac{(g_{11}l_{22} + l_{11}g_{22} - 2g_{12}l_{12})^2 - 4\,|G|\,|L|}{|G|^2} \\
&= \frac{(g_{11}\hat{l}_{22} + \hat{l}_{11}g_{22} - 2g_{12}\hat{l}_{12})^2 - 4\,|G|\,\left|\hat{L}\right|}{|G|^2\,\|\hat{n}\|^2}.
\end{aligned} \tag{4.12}$$

Both (4.11) and (4.12) can be represented without square roots and are therefore representable as NURBs using the model and tools defined in Chapter 2.

By using the property surface $\psi(u,v) = (\kappa_n^1(u,v) + \kappa_n^2(u,v))^2$ as a curvature estimate for the convex and concave regions, the computed curvature will be at most twice as large as the real normal curvature in the case where both $\kappa_n^1(u,v)$ and $\kappa_n^2(u,v)$ are equal. Similarly by using $\phi = (\kappa_n^1(u,v) - \kappa_n^2(u,v))^2$ as the curvature estimate for saddle regions one can obtain similar bounds.

$\psi(u,v)$ and $\phi(u,v)$ can be used as curvature estimates for the appropriate trimmed regions and can be contoured to isolate regions with curvature larger than some allowable threshold. Furthermore, one can use $\psi(u,v)$ and $\phi(u,v)$ as pseudo color values to render the input surface $F(u,v)$ according to its curvature and provide visual feedback on which regions are highly curved. In other words, make the color of $F(u,v)$ at the parameter value (u,v) depend on the value of $\psi(u,v)$ in convex and concave regions, and on the value of $\phi(u,v)$ in saddle regions. Using this technique, one can enhance the display of regions with high curvature, low curvature, or within certain bands of curvatures. Figures 4.10 through 4.12 demonstrate this. In Figure 4.10, the surface has been first subdivided into a saddle region (yellow) and a convex region (red). $\psi(u,v)$ has been used as the pseudo color in the convex region of the surface whereas $\phi(u,v)$ has been used for the same purpose in the saddle region, to render the image in Figure 4.12. Figure 4.11 shows $\psi(u,v)$ and $\phi(u,v)$. Not surprisingly, $\psi(u,v)$ is wider in the highly curved convex region because the two principal curvatures cancel each other in $\phi(u,v)$.

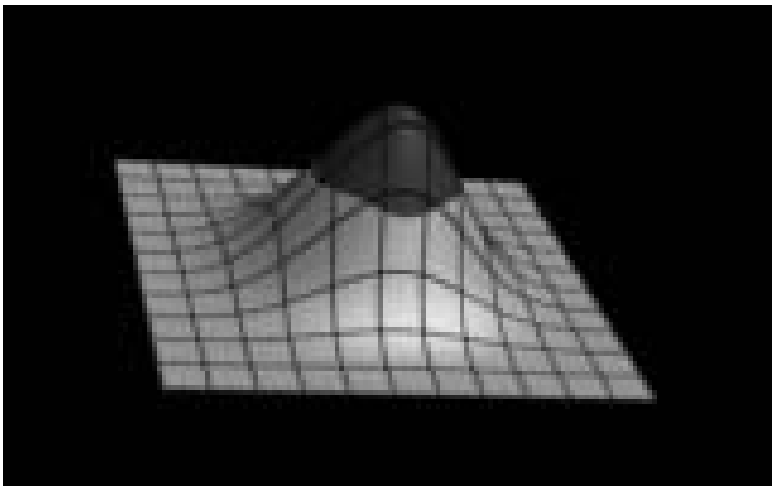


Figure 4.10. Surface dichotomy - saddle and convex regions.

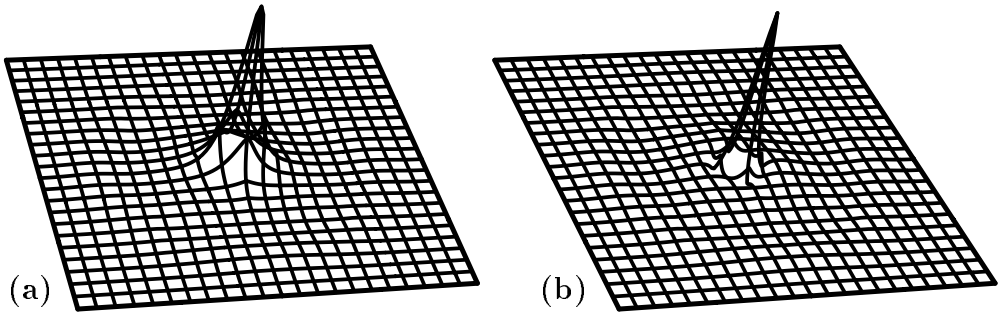


Figure 4.11. $\psi(u,v)$ (a), $\phi(u,v)$ (b), for the surface in Figure 4.10.

A different approach can be used to achieve a better bound. By expanding ϕ,

$$\begin{aligned} \phi &= (\kappa_n^1 - \kappa_n^2)^2 \\ &= (\kappa_n^1)^2 - 2\kappa_n^1\kappa_n^2 + (\kappa_n^2)^2. \end{aligned} \tag{4.13}$$

Or

$$\begin{aligned} \xi &= (\kappa_n^1)^2 + (\kappa_n^2)^2 \\ &= \phi + 2\kappa_n^1\kappa_n^2 \\ &= \phi + 2K \\ &= \phi + 2\frac{|L|}{|G|} \end{aligned}$$

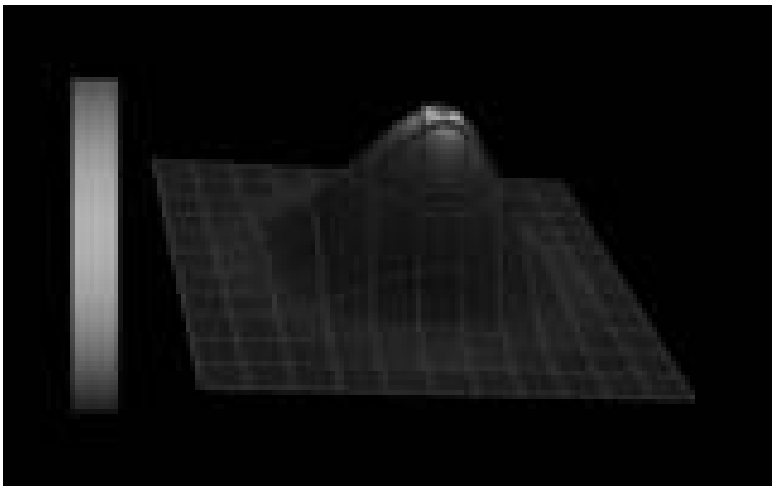


Figure 4.12. Curvature estimate using surface dichotomy, for surface in Figure 4.10.

$$
\begin{aligned}
&= \frac{(g_{11}\hat{l}_{22} + \hat{l}_{11}g_{22} - 2g_{12}\hat{l}_{12})^2 - 4\,|G|\,\left|\hat{L}\right|}{|G|^2\,\|\hat{n}\|^2} + 2\frac{|L|}{|G|} \\
&= \frac{(g_{11}\hat{l}_{22} + \hat{l}_{11}g_{22} - 2g_{12}\hat{l}_{12})^2 - 4\,|G|\,\left|\hat{L}\right|}{|G|^2\,\|\hat{n}\|^2} + 2\frac{\left|\hat{L}\right|}{|G|\,\|\hat{n}\|^2} \\
&= \frac{(g_{11}\hat{l}_{22} + \hat{l}_{11}g_{22} - 2g_{12}\hat{l}_{12})^2 - 2\,|G|\,\left|\hat{L}\right|}{|G|^2\,\|\hat{n}\|^2}. \qquad (4.14)
\end{aligned}
$$

$+\sqrt{\xi}$ is bounded to be at most $\sqrt{2}$ greater than the larger magnitude of the principal curvatures. This worst case occurs when the two principal directions have the same magnitudes. Furthermore, ξ can be represented using the tools described in Chapter 2. Figure 4.13 demonstrates this approach applied to the Utah teapot model. The use of ξ may help to isolate regions with low curvature, which can be milled using larger ball end tools in a more optimal way. Figure 4.14 shows such a surface subdivided in such regions. The curvature bound surface, $\xi(u, v)$, (Figure 4.15) of the surface in Figure 4.14 is being contoured and regions with different curvature bounds are formed. It is clear from Figure 4.14 that the blue regions can be milled using a very large ball end cutter, the green regions with a medium size cutter and only the yellow and red regions, which are less than 5% of the whole surface area, should be milled with a small size tool.

Figure 4.13. Utah teapot curvature estimation.

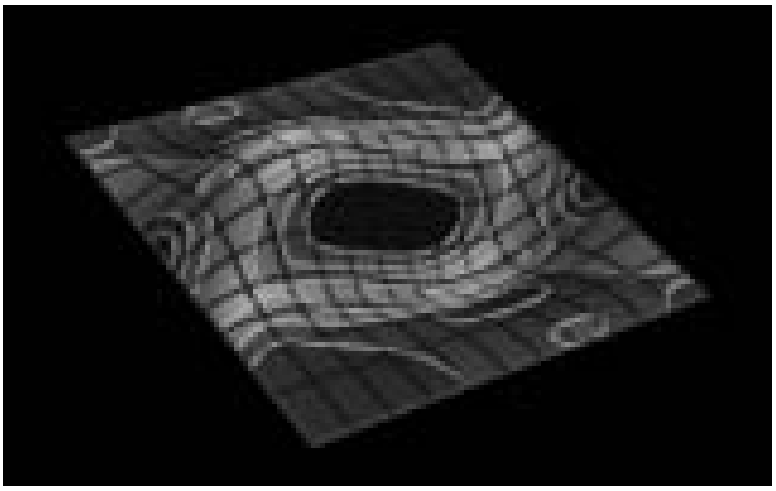


Figure 4.14. The surface is subdivided into regions with different curvature bounds.

4.3 Some Remarks

A method to partition a surface into three disjoint trimmed surfaces (convex, concave, and saddle) and to determine global bounds on surface curvatures, has been presented here which combines symbolic and numeric methods. The hybrid method was found to be robust and fast. The computation involved in the creation of a property surface that is exact to machine accuracy usually takes less than a second for a bicubic Bézier surface on an SGI 240/GTX (25MHz R3000). This symbolic computation has closed forms with complexity directly bounded by the

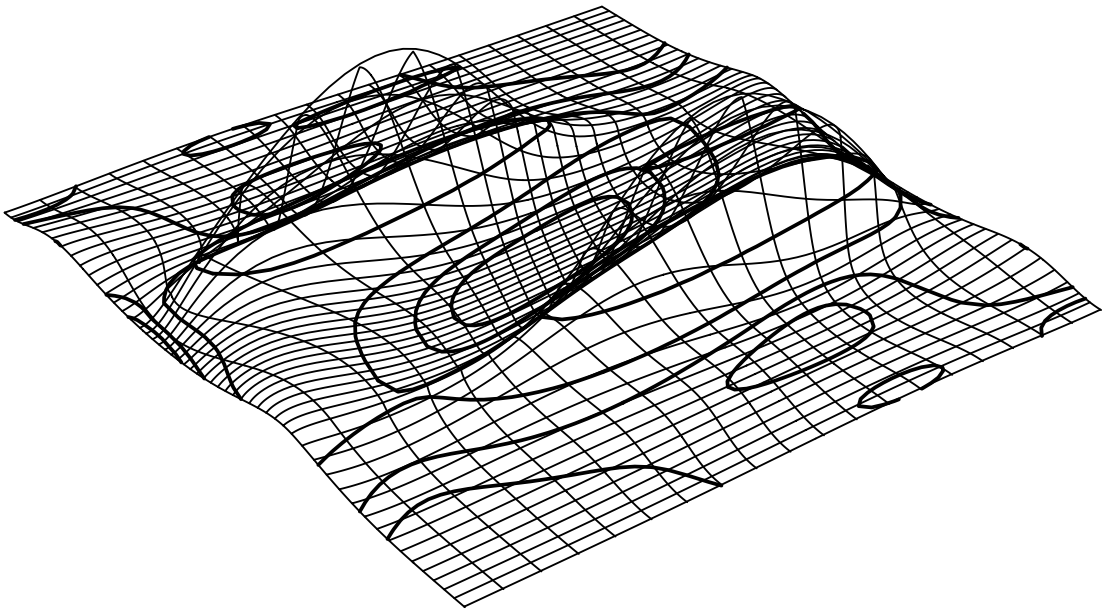


Figure 4.15. Curvature surface bound, ξ, of the surface in Figure 4.14.

surface orders and continuity (knot vectors). Contouring usually takes an order of magnitude longer than that. This numeric process involves high order property surfaces which make subdivision more expensive.

The orders of the resulting property surfaces are high. A second fundamental form determinant property surface for a bicubic B-spline surface has degree 14. The degree of the property surfaces $\psi(u,v)$, $\phi(u,v)$ and $\xi(u,v)$ is even higher, degree 30. However, because the evaluation of Bézier and B-spline representations is robust, the high order does not introduce any numerical problems [31] in evaluation.

Because milling is several magnitudes slower than even the contouring process, and the same toolpath may be used thousands of times, computation time is not a major factor in optimizing the milling process. The ability to isolate regions in a surface with specific curvature bounds makes it possible to mill the surface more optimally by using the largest tool possible for each region.

CHAPTER 5

MACHINING APPLICATIONS

Computing and numerical control (NC) has made great progress at that time, and it was certain that only numbers, transmitted from drawing office to tool drawing office, manufacture, patternshop, and inspection, could provide an answer; of course, drawings would remain necessary, but they would only be explanatory, their accuracy having no importance. Numbers would be the only and final definition.

P. Bézier (on NC capabilities in the 1960s)

5.1 Introduction

Generating optimal NC code to drive milling machines for models defined by freeform trimmed surfaces is a difficult problem. In practice, two main approaches are used to generate toolpaths for surfaces, neither of which is optimal, in general. The first exploits the parametric representation and generates isocurves that are uniformly distributed across the parametric domain. This approach is not optimal if the surface mapping into Euclidean space is not isometric. The second approach contours the models by intersecting the surfaces with planes equally spaced in Euclidean space, resulting in a piecewise linear toolpath approximation which is nonadaptive to the local surface geometry. Furthermore, the toolpath generated by contouring is suitable for 3 axis milling but is inappropriate for 5 axis milling. This Chapter addresses some of the relevant issues in this field of realizing computer models.

In section 5.2, an algorithm developed to adaptively extract isocurves for rendering [26] is adapted and enhanced to generate milling toolpaths for models consisting of trimmed surfaces, and can be used in both 3 and 5 axis milling. Section 5.2.1

develops and defines this new algorithm. Sections 5.2.2 and section 5.2.3 deal with some practical problems while section 5.2.4 provides some examples and results. Finally, in section 5.3, a whole new approach to realizing computer models is derived using piecewise ruled and developable surface approximations.

5.2 Adaptive Isocurves Toolpath

In order to evaluate the quality of toolpaths, two criteria are introduced. One deals with the validity of a set of toolpaths and the other with its optimality.

> **Definition 5.1** *A set of curves $\mathcal{C}$ in a given surface S is called a* valid coverage for S with respect to some constant δ *if for any point p on S there is a point q on one of the curves in $\mathcal{C}$, such that $\|p-q\|_2 < \delta$, where $\|\cdot\|_2$ denotes the Euclidean distance.*

Definition 5.1 provides a validation criterion on a given toolpath and a tolerance δ such that any point on the surface is at most δ from the nearest toolpath curve. Definition 5.1 takes into consideration only the distance between an arbitrary point p on the surface and the closest point on the toolpath. Other criteria, such as bounding the curvature, could be added to the definition of *validity* of a toolpath to provide a tighter bound on the resulting scallop height without affecting any of the rest of the algorithm.

We also would like to consider the *optimality* of a valid toolpath.

> **Definition 5.2** *A toolpath for a given surface is considered* optimal *if it is valid and if its path length is minimal.*

Definition 5.2 considers optimality based only on the cutting motion part of the toolpath. Tool retraction and traversals are not considered as optimality conditions in this Chapter. One might decide to traverse the iso-curves in incremental cross

iso-direction so that the portion of the surface of the machining tool that performs the actual milling is approximately the same throughout the milled surface. Any other type of traversal might, in some stage of the milling, require the tool to cut using its entire milling surface perimeter, an undesired tool machining motion. Unfortunately, the time to find an optimal traversal of the piecewise cutting motion toolpath is exponential in nature; more on this problem can be found in [12].

There are two main approaches used to generate tool paths for freeform surfaces. In one, isoparametric curves are extracted from the surface, usually in equally spaced parametric steps [10, 12, 32, 46]. These isocurves usually span the entire parametric domain of the surface (see Figures 5.1a and 5.2a) and will be referred to as *complete-isocurves*. Isocurves that span only a portion of the surface parametric domain (see Figures 5.1b and 5.2b) will be referred to as *subisocurves*. Although simple to determine, toolpaths created using complete isocurves equally spaced in parametric space, are clearly not optimal according to definition 5.2 and are redundant, as can be seen in the example of Figure 5.1a, where the toolpath is redundant in the middle region of the surface. In order to guarantee the validity of the toolpath, a certain parametric stepsize is selected for the complete isocurves (for example, derived by the top and bottom regions of the surface in Figure 5.1a) and which undoubtly leads to a much smaller distances between adjacent complete isocurves in other surface regions than required causing *redundancy* (in the middle of the surface in Figure 5.1a). Further, it might be difficult for the user to determine the parameter stepping tolerance that will create valid toolpaths to within a given δ, even if the top and bottom regions of the surface in Figure 5.1a are treated separately. The user is interested mainly in the shape of the represented geometry and the associated milling, so the parametric representation of the surface should not require his attention, but be internal.

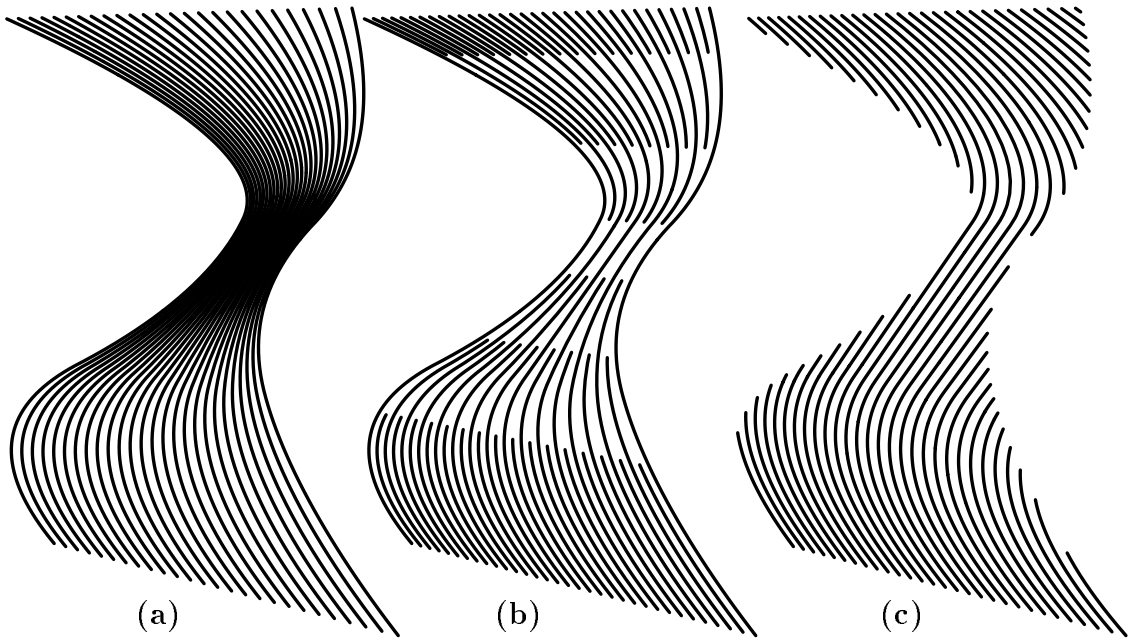


Figure 5.1. Isocurves are obviously not an optimal solution as a toolpath for this surface (a). Adaptive isocurves are, in general, more optimal, exact, and compact (b). Contouring with equally spaced parallel planes might be optimal but is piecewise linear (c).

An alternative method for generating toolpaths is based on contouring planes, in which the surface is intersected by (usually geometrically equally spaced) parallel planes. The intersection curves are used to drive the milling tools [8, 10]. The resulting toolpath is, in general, only a piecewise linear approximation to the real intersection, and the size of the piecewise linear approximations of the intersection curves is usually several magnitudes larger than isocurve data. For relatively flat surfaces the contouring algorithm seems to yield acceptable results (see Figure 5.1c). However, as is the case for the complete isocurves algorithm, some frequently occurring surfaces can be pathological to this contouring algorithm. If the surface has regions almost coplanar to the contouring plane, adjacent contours would be distant from each other, as can be seen from Figure 5.2c, invalidating the toolpath. How to set the parallel plane spacing and the parallel plane direction to create a valid toolpath is not obvious. Even if an algorithm could be created to adaptively

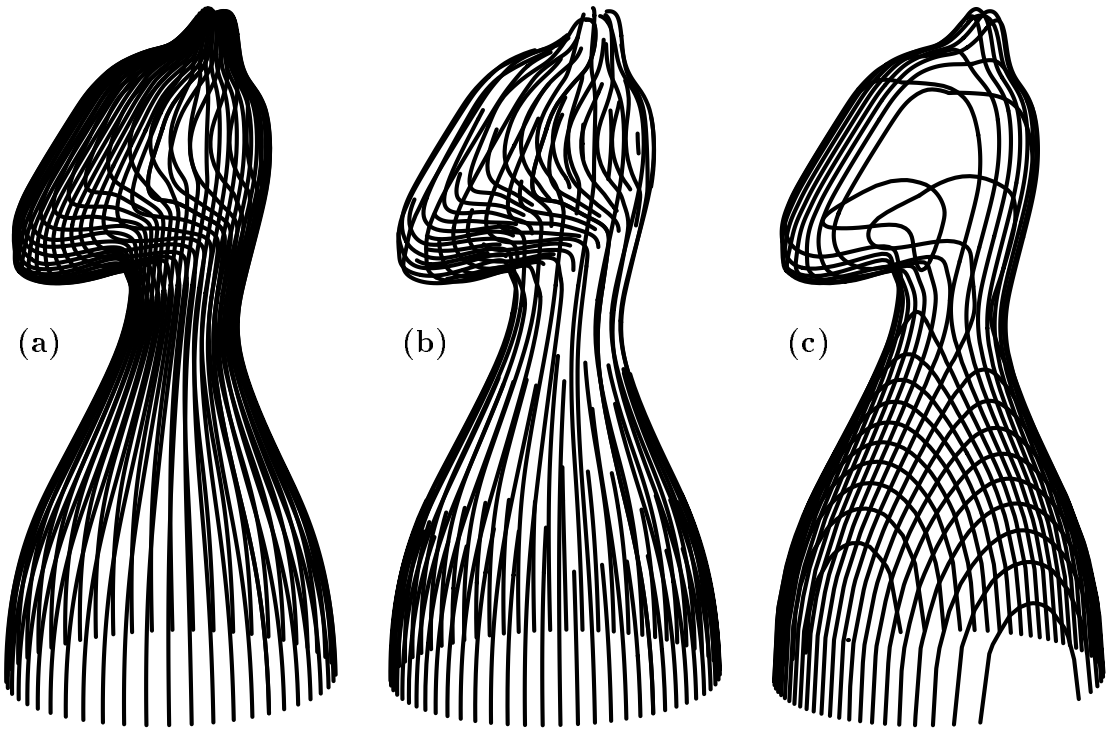


Figure 5.2. Toolpath using isocurves will be not optimal in this complex surface (a). Adaptive isocurves are more optimal, exact, and still correctly spans the entire surface (b). Contouring with equally spaced parallel planes is too sparse in coplanar regions (c).

space the contours based on the coplanarity of one surface region, this spacing would be fixed for the entire contoured model. Local coplanarity in one region of the surface would set the spacing for the entire model.

Attempts to improve those techniques have been geared mainly toward local adaptation of the algorithm to specific regions which require a different number of samples to gain the required tolerances [40, 46]. Others used adaptation of scanline fashion rendering [64, 67] to get a piecewise linear approximation for the toolpath.

An adaptive subisocurve extraction approach is introduced for rendering in [26]. That scheme provides a more optimal and valid coverage of the surface by adaptively introducing partial subisocurves in regions yet uncovered by already created subisocurves (definition 5.1). Furthermore, the algorithm frees the user from the need to determine both the surface parameter spacing or contouring plane spacing,

and the direction to use to insure adjacent isocurve distances to produce a valid coverage. Instead, a bound on the required distance between adjacent subisocurves can be directly specified, and guaranteed automatically.

It is clear that a valid coverage generated using complete isocurves can be very inefficient (see Figure 5.1a), which can increase machining time and affect part finish. If the redundant portion of each complete isocurve could be *a priori* detected and not be generated as part of the valid coverage, one would be able to generate a more optimal toolpath with the appeal of the isocurves. The adaptive isocurve extraction algorithm does exactly that for rendering (see Figures 5.1b and 5.2b). because the adaptive isocurve extraction algorithm is developed for rendering in [26] it will be briefly discussed here. The interested reader can also refer to [26].

It is appealing to use isocurves because their representations are compact, exact, and they are straightforward to use as milling toolpath. Isocurves can be approximated more compactly and accurately using piecewise arcs (and lines), if circular motion is supported by the milling machines than by using piecewise linear approximation alone. Furthermore, isocurves could be sent directly to a milling machine that supports NURBs or Bézier curve toolpaths. Isocurves are also invariant under affine transformations and therefore are view direction independent, unlike the results of the contouring technique. Scallops resulting from isocurve based toolpaths are usually more attractive than those resulting from contoured based toolpaths because they follow the model's basic streamlines. Finally, when computing toolpaths for models having trimmed surfaces, it is easier to trim isocurves to the appropriate domains than to trim contours whose parametric domain representation can be arbitrary.

Recent literature [8, 10, 40] has suggested that the contact point numerical improvement approach, such as used by APT [32], is unstable and slow. Computations of a toolpath for a single surface are usually measured in minutes [10, 40]. A

different known approach [40] was selected in this work. The model was offset by the tool ball end radius and toolpaths for the tool center were generated using the offset surface.

Section 5.2.1 briefly discusses the adaptive subisocurve algorithm. Section 5.2.2 describes the offset computation required for ball end tool milling, and section 5.2.3 deals with the method used for the rough cutting process. Finally, section 5.2.4 presents some results obtained from an implementation of the new algorithm for NURBs based models using the Alpha_1 solid modeler.

5.2.1 Adaptive Isocurves Algorithm

Using isocurves as the coverage for a surface, we define *adjacency* and *iso-distance* between isocurves.

Definition 5.3 *Two (sub)isocurves of surface* $S(u,v)$, $C_1(u) = S(u,v_1)$, $u \in [u_1^s, u_1^e]$ *and* $C_2(u) = S(u,v_2)$, $u \in [u_2^s, u_2^e]$, $v_1 \leq v_2$, *from a given set* $\mathcal{C}$ *of isocurves forming a valid coverage for* S *are considered* adjacent *if, along their common domain* $\mathcal{U} = [u_1^s, u_1^e] \cap [u_2^s, u_2^e]$, *there is no other isocurve from* $\mathcal{C}$ *between them. That is, there does not exist* $C_3(u) = S(u,v_3) \in \mathcal{C}$, $u \in [u_3^s, u_3^e]$ *such that* $v_1 \leq v_3 \leq v_2$ *and* $[u_3^s, u_3^e] \cap \mathcal{U} \neq \emptyset$.

Definition 5.4 *The* iso-distance *function* $\Delta_{12}(u)$ *between two adjacent (sub) isocurves along their common domain* $\mathcal{U}$ *is equal to*

$$\begin{aligned} \Delta_{12}(u) &= \|C_1(u) - C_2(u)\|_2 \\ &= \sqrt{(c_1^x(u) - c_2^x(u))^2 + (c_1^y(u) - c_2^y(u))^2 + (c_1^z(u) - c_2^z(u))^2}. \end{aligned} \tag{5.1}$$

Given two isocurves, $C_1(u)$ and $C_2(u)$, on a surface $S(u,v)$, one can compute and represent the square of the iso-distance, $\Delta_{12}^2(u)$, between them symbolically as a NURBs or as a Bézier curve. Computing the coefficients for the representation of $\Delta_{12}^2(u)$ requires the difference, sum, and product of curves, all computable and representable in the polynomial and piecewise polynomial domains. Furthermore, given some tolerance δ, it is possible to compute the parameter values where the iso-distance between $C_1(u)$ and $C_2(u)$ is exactly δ by computing the zero set of $(\Delta_{12}^2(u) - \delta^2)$ [44]. By subdividing the two curves at these parameters, new subisocurve pairs, $\{C_1^i(u), C_2^i(u)\}$, are formed with the characteristic that each pair is always iso-distance smaller or always larger than δ, in their open interval domains. If the two curves in the pair $\{C_1^i(u), C_2^i(u)\}$ are closer than δ in the iso-distance metric then the Euclidean distance tolerance condition is met for that pair. If, however, the two curves' iso-distance is larger than δ, a new subisocurve, $C_{12}^i(u)$, is introduced between $C_1^i(t)$ and $C_{12}^i(t)$ along their common domain $\mathcal{U}$ and the same iso-distance computation is recursively invoked for the two new pairs $\{C_1^i(u), C_{12}^i(u)\}$ and $\{C_{12}^i(u), C_2^i(u)\}$.

Starting with the two U boundaries or two V boundaries of the surface, the algorithm can invoke this iso-distance computation recursively and ensure two adjacent isocurves will always be closer than some specified distance δ by verifying that their iso-distance is not larger than δ. Because a middle isocurve is introduced iff the iso-distance is larger than δ and δ is small, resulting iso-distances between adjacent isocurves, as computed, are rarely less than $\frac{\delta}{2}$. Furthermore, because the resulting set of isocurves covers the entire surface S, it can serve as a valid toolpath for S with distance δ.

Algorithm 5.1, the adaptive isocurve extraction algorithm, generates a valid and more optimal coverage by minimizing the cutting speed motion required by minimizing redundancies while providing a bound on the scallop height via the

Algorithm 5.1

Input:
Surface $S(u,v)$.
Iso-distance tolerance δ.

Output:
Adaptive isocurve toolpath for $S(u,v)$.

Algorithm:
AdapIsoCurve($S(u,v)$, δ).
begin
$C_1(u)$, $C_2(u) \Leftarrow S(u,v)$ two u boundary curves.
return **AdapIsoCurveAux**($S(u,v)$, δ, $\{C_1(u), C_2(u)\}$).
end

bound on the distance between two adjacent isocurves.

It is important to realize that bounding the distance between adjacent isocurves is a necessary condition to bound the scallop height. The surface curvature bound (See [24]) could be added to the definition of validity to decide whether to introduce a middle isocurve in algorithm 5.1 and obtain a tighter bound on the scallop height.

5.2.2 The Offset Computation

Because the toolpath generated by the adaptive isocurve algorithm provides a valid coverage of the surface, it can serve as a toolpath for both 3 axis and 5 axis milling. In this discussion, we will concentrate on 3 axis milling using ball end tools. Such a method requires the computation of an offset surface to the model at a distance equal to the radius of the ball end tool. This simplifies the toolpath generation because keeping the center of the ball end tool on the offset surface, keeps the tool tangent to the original surface so it can not gauge.

Unfortunately, the exact offset of a freeform piecewise polynomial or rational surface is not representable, in general, as a piecewise polynomial or rational

Algorithm 5.1 continued

AdapIsoCurveAux($S(u,v)$, δ, $\{C_1(u), C_2(u)\}$ **)**
begin
  $\Delta_{12}^2(u) \Leftarrow \|C_1(u) - C_2(u)\|_2$, iso-distance between $C_1(u)$ and $C_2(u)$.
  if $(\Delta_{12}^2(u) < \delta^2, \forall u)$ then
    return $\emptyset$.
  else if $(\Delta_{12}^2(u) > \delta^2, \forall u)$ then
    begin
      $C_{12}(u) \Leftarrow$ Middle isocurve between $C_1(u)$ and $C_2(u)$.
      return **AdapIsoCurveAux(** $S(u,v)$, δ, $\{C_1(u), C_{12}(u)\}$ **)** $\bigcup$ **AdapIsoCurveAux(** $S(u,v)$, δ, $\{C_{12}(u), C_2(u)\}$ **)**.
    end
  else
    begin
      $\{C_1^i(u), C_2^i(u)\} \Leftarrow$ subdivided $\{C_1(u), C_2(u)\}$ at all u such that $\Delta_{12}^2(u) = \delta^2$.
      return $\bigcup_i$ **AdapIsoCurveAux(** $S(u,v)$, δ, $\{C_1^i(u), C_2^i(u)\}$ **)**.
    end
end

surface [25]. Quite a few methods have been developed in recent years to provide approximations to surface offsets [2, 13, 25, 29]. In [25], a technique to approximate offsets of freeform Bézier and NURBs surfaces by Bézier and NURBs surfaces was developed with the property that error in the approximation surface is *globally* bounded. That global bound can be used directly to determine a global bound on the accuracy of the milling and the amount of gouging that may occur.

Extending the generation of surface toolpaths to models defined using constructive solid geometry [34] and consisting of several, possibly trimmed, surfaces is not obvious. Let $O(\mathcal{A})$ denote the exact offset of $\mathcal{A}$. It is unfortunate but $O(\mathcal{A} \cap \mathcal{B})$ is not always the same as $O(\mathcal{A}) \cap O(\mathcal{B})$. For example, $\mathcal{A} \cap \mathcal{B}$ and hence $O(\mathcal{A} \cap \mathcal{B})$

could be empty but $O(\mathcal{A}) \cap O(\mathcal{B})$ might be a nonempty.

Several types of manufacturing offsets can be defined for piecewise C^1 models [54, 57] that are constructed by constructive solid geometry. In general, one should attempt to prevent gouging even at the expense of not being able to mill the entire model. A C^1 discontinuous concave corner created by a union of two surfaces cannot be milled using a ball end tool of any size. One could define the offset operator for a piecewise C^1 model so that at no time would the center of the ball end tool be closer than its radius to any of the surfaces of the model. Using such definition it can be guaranteed that during the entire milling process,

$$\|T_c - S_i(u^i, v^i)\|_2 \geq T_r, \qquad \forall i, \tag{5.2}$$

where T_c and T_r are the center and the radius of the ball end tool, respectively, where $S_i(u, v)$ is the ith surface in the model, and (u^i, v^i) is a parametric location in the untrimmed domain of surface S_i.

If $O(S_i)$ designates the exact offset surface to surface S_i at distance T_r, it is clear that the ball end tool could not gouge S_i if T_c were kept on $O(S_i)$. We define the manufacturing offset of a Boolean union operation of two surfaces, $S_i \bigcup S_j$, to be the union of the offset surfaces, that is $\widehat{O}(S_i \bigcup S_j) \equiv O(S_i) \bigcup O(S_j)$, even though the model might not be completely milled along the intersection curve of S_i and S_j in concave regions. Such a definition guarantees that the tool will gouge neither S_i nor S_j. Similarly, the manufacturing offset of a Boolean intersection operation of two surfaces, $S_i \bigcap S_j$ is defined as the intersection of the offset surfaces, that is $\widehat{O}(S_i \bigcap S_j) \equiv O(S_i) \bigcap O(S_j)$. Because the Boolean intersection operation only "removes material," it is not possible for it to form concave corners from an intersection of two C^1 continuous surfaces, so the $\widehat{O}$ definition of an offset of a Boolean intersection operation supports the milling of the entire region along the intersection curves.

Because an offset of a single surface is another single surface [25], Boolean operations can be performed on the offset surfaces in much the same way they were computed for the original models. Consider surfaces S_i and S_j that intersect. If the intersection occurs near the boundary of either surface, it can happen that $O(S_i)$ does not intersect $O(S_j)$. For open surfaces, one solution that forms correct intersection curves is to extend them in the cross boundary tangent directions.

5.2.3 Rough Cutting Stage

The toolpath derived in section 5.2.1 cannot, in general, be directly applied to the stock from which the model is to be machined. In some cases, the depth of milling required is simply too large. A rough cutting stage is usually applied in which the excessive material is removed crudely. Then, in the final stage, when the toolpath derived in section 5.2.1 is applied, it is necessary to remove only a limited amount of material.

One way to discard the excessive material, in 3 axis milling, is to slice the offset approximation of the model with several parallel planes and remove the material external to the part at each contour level. Two-dimensional pocketing operations [12] can be used to remove the excessive material at each contoured layer. Figure 5.3 shows those contours of a "house on the hill" model. The rough cutting stage can be automated, similarly to the adaptive isocurve extraction algorithm.

5.2.4 Results

Several results are presented in this section, as are some timing considerations. The adaptive isocurve toolpaths for the knight in Figure 5.2b have been used to mill the complete knight. Two fixtures, one for the right side and one for the left side of the knight have been used. Figure 5.4 shows a raytraced version of the model while Figure 5.5 shows the milled piece. A ball end tool was driven along an

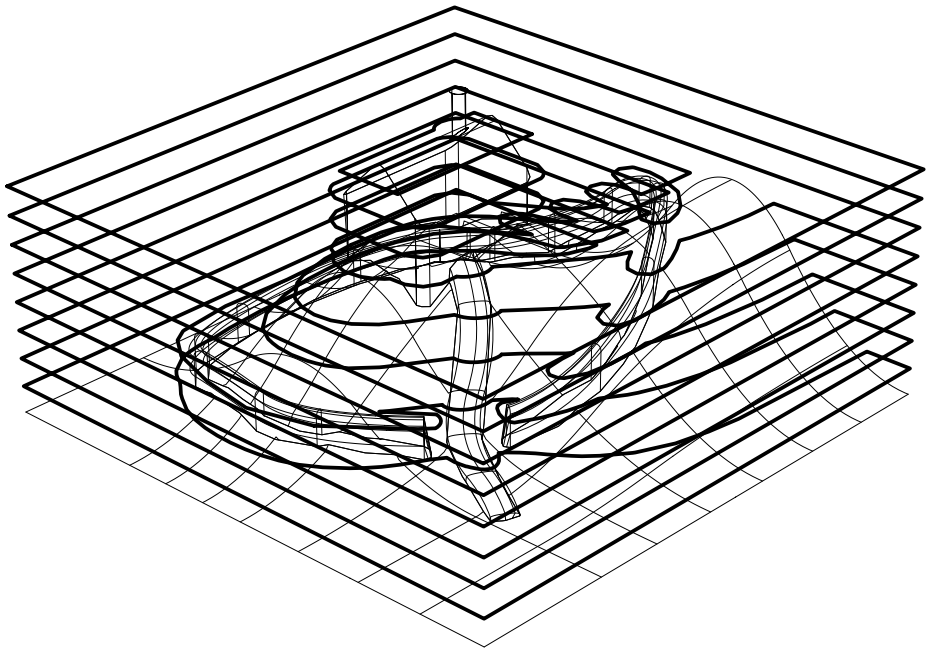


Figure 5.3. Parallel plane contouring is used to generate pockets for rough cutting.

offset [25] of the knight surface in 3 axis milling mode. The knight model consists of a single highly complex NURBs surface.

This algorithm produces only isoparametric curves that are simple to clip against the surface trimming curves defining a trimmed surface. A "house on a hill" model, consisting of several trimmed surfaces was used for this example. This model was milled using a ball end tool in 3 axis mode. Figure 5.6 shows a raytraced version of the model, while Figure 5.7 shows the adaptive isocurve toolpath used in the finish stage of the model in Figure 5.8. The offset of the model was automatically computed using the the $\widehat{O}$ offset method described in section 5.2.2. Furthermore, it was unnecessary to introduce any auxiliary check or driver surfaces [32] as part of this automated toolpath generation process.

To gain some insight regarding this algorithm, Table 5.1 provides some timing results for computing the adaptive isocurve toolpaths for the tests displayed. Tests were running on a SGI4D 240 GTX (R3000 25MHz Risc machine). The surface in Figure 5.1 is a B-spline ruled surface with 3 Bézier patches (patches of a NURBs

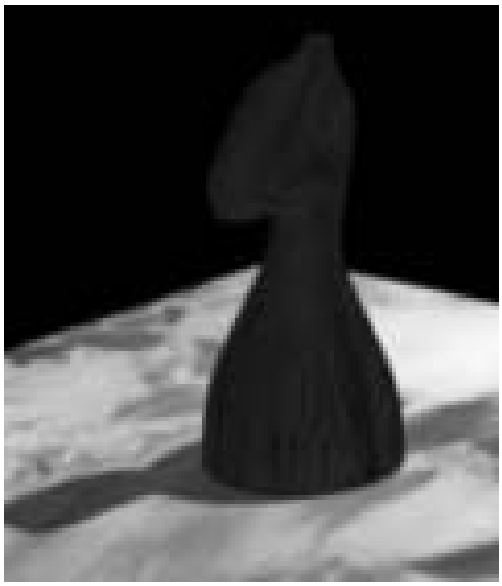


Figure 5.4. Raytraced image of the knight model.

Table 5.1. CPU times for adaptive iso-curves extraction.

model	cpu time	# isocurves.
Figure 5.1b - surface	5.6 sec.	61
Figure 5.5 - knight	54.3 sec.	122
Figure 5.7 - "house on a hill"	132.0 sec.	1013

surface are enumerated as the number of Bézier patches that would result from subdividing the NURBs surface at each original interior knot). The knight is a far more complex NURBs surface. Its 56 Bézier patches accounts for its long processing time. Although the "house on the hill" model has 7 NURBs surfaces in it, none of them is as complex as the single surface defining a knight.

5.3 Fabrication Using Layout Projection

It is common to find freeform surfaces manually approximated and assembled as sets of piecewise developable surfaces. "Developable surfaces are of considerable importance to sheet-metal- or plate-metal-based industries and to a less extent to fabric-based industries" [56]. Parts of aircrafts and ships are assembled from

Figure 5.5. Aluminum milled version of the knight model.

piecewise planar sheets unidirectionally bent into their model positions. Certain fabric and leather objects are made using patterns made from planar sheets.

Because developable surfaces can be unrolled onto a plane without distortion, they can be cut from planar sheets, bent back into their final position, and stitched together.

In [5], a flattening approximation is computed for freeform surfaces to eliminate the distortion in texture mapping. Surfaces are split into patches along feature (geodesic) lines and approximated as flats. However, we are mainly interested in isometric projections that preserves intrinsic distances and angles [21]. Physically, such maps only bend the surface with no stretching, tearing, or distortion. One of the most interesting properties of developable surfaces is their ability to be laid flat on a plane without distortion by simply unrolling them [21, 32]. Therefore, we would like to generate a surface approximation using piecewise developable surfaces [21, 32], for which an isometric map to a plane exists.

Currently, the process that determines how and where to decompose the model requires human ingenuity and does not provide a bound on the accuracy of the

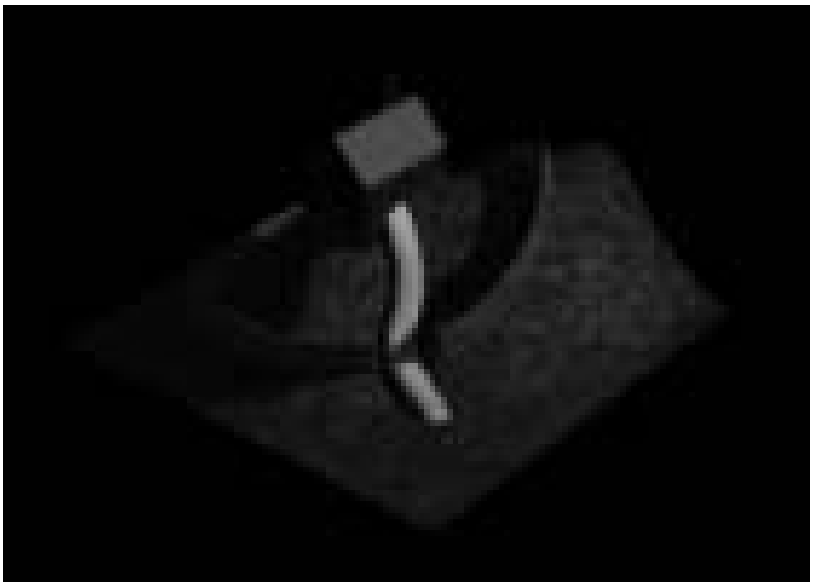


Figure 5.6. Raytraced image of the "house on the hill" model.

approximation. We explores a technique for automatically decomposing the sculptured model, using a C^0 approximation with error bound control, into sets of developable surfaces.

The Gaussian curvature of a developable surface $S(u,v)$, K, is zero everywhere [21, 32], i.e., $K(u,v) \equiv 0$. The class of developable surfaces is difficult to deal with, so we will first concentrate on a superset of it, namely the class of ruled surfaces. In order to be able to use ruled surfaces instead, we need to derive the conditions in which a ruled surface is also developable. Let $|G|$ and $|L|$ be the determinants of the first and second fundamental form [21], respectively.

Lemma 5.1 *Let R be a regular ruled surface, $R(u,v) = C_1(u)*v + C_2(u)*(1-v)$, $v \in (0,1)$. R is developable if and only if $\left\langle n_r, \frac{\partial^2 R}{\partial u \partial v} \right\rangle \equiv 0$,*

Proof: Given a regular surface S, its Gaussian curvature, K, is zero everywhere (therefore, it is developable) if $|L| \equiv 0$ because $K = \frac{|L|}{|G|}$, and $|G| \neq 0$ for regular surfaces.

$$|L| = \left\langle n, \frac{\partial^2 S}{\partial u^2} \right\rangle \left\langle n, \frac{\partial^2 S}{\partial v^2} \right\rangle - \left\langle n, \frac{\partial^2 S}{\partial u \partial v} \right\rangle^2 . \tag{5.3}$$

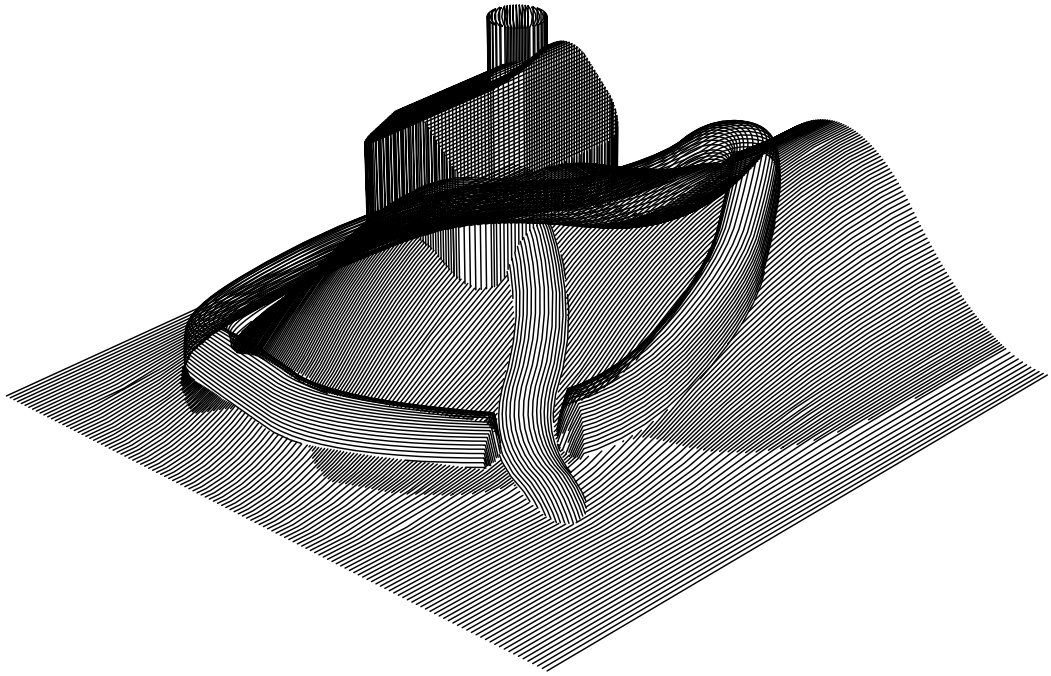


Figure 5.7. Adaptive isocurves toolpath for $\hat{O}$ offset of "house on the hill" model.

By differentiating R twice in v, it is clear that $\frac{\partial^2 R}{\partial v^2} \equiv 0$. We can immediately rewrite $|L|$ as

$$|L_R| = -\left\langle n_r, \frac{\partial^2 R}{\partial u \partial v} \right\rangle^2, \tag{5.4}$$

and the result follows. ■

Therefore, to determine if a ruled surface is developable, one can symbolically compute $\sigma(u,v) = \left\langle n_r, \frac{\partial^2 R}{\partial u \partial v} \right\rangle$ (That is, represent the scalar surface $\sigma(u,v)$ as a Bézier or NURBs scalar surface) and make sure it is zero everywhere within a prescribed tolerance. In other words, using the convex hull property of the Bézier and NURBs representations, all the coefficients of the scalar surface $\sigma(u,v)$ must be zero within a prescribed tolerance.

The mixed partials, also called the twist of the surface [3], are a measure of the "crosstalk" in the parameterization. Equation 5.4 measures this "crosstalk" projected in the direction of the surface normal.

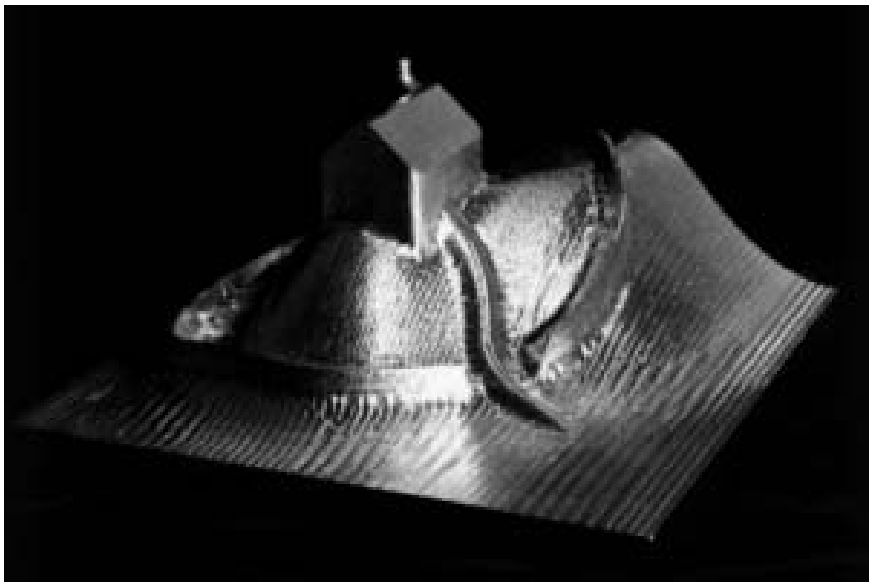


Figure 5.8. Aluminum milled version of the "house on the hill" model.

Assuming one can approximate a given surface by a set of disjoint (except along boundaries) piecewise ruled surfaces within a prescribed tolerance, lemma 5.1 can be used to test that each member of the set of ruled surfaces is also developable. Each developable surface can then be unfolded, laid flat and cut from a planar sheet such as paper or metal. By folding each back to its Euclidean orientation and stitching them all together, a C^0 approximation of the computer model is constructed.

Section 5.3.1 develops the background required for this method, and presents the basic algorithm. In section 5.3.2 we investigate several possible extensions including optimization, stub generation, and handling of trimmed surfaces. Section 5.3.3 lays out several examples including some models assembled from paper.

We will concentrate in our discussion on the NURBs representation although the developed technique may very well fit into any other piecewise polynomial or rational representation.

5.3.1 Algorithm

Let $S(u,v)$ be a nonuniform polynomial B-spline surface. Let the curves $C_1(u) = S(u, Vmin)$ and $C_2(u) = S(u, Vmax)$ be the $Vmin$ and $Vmax$ boundary curves of $S(u,v)$ respectively, $C_1(u) \neq C_2(u)$. Let $R(u,v)$ be the ruled surface constructed between $C_1(u)$ and $C_2(u)$. Let $\hat{R}(u,v)$ be the representation for $R(u,v)$ in the same B-spline basis as that of $S(u,v)$. $\hat{R}(u,v)$ can be obtained from $R(u,v)$ via appropriate degree raising [16, 17] and refinement [15] in the linear (ruled) direction, v. Then

$$\begin{aligned} \|S(u,v) - \hat{R}(u,v)\| &= \|\sum_{i=0}^{m}\sum_{j=0}^{n} P_{ij}B_{i,\tau}^{m}(u)B_{j,\xi}^{n}(v) - \sum_{i=0}^{m}\sum_{j=0}^{n} Q_{ij}B_{i,\tau}^{m}(u)B_{j,\xi}^{n}(v)\| \\ &= \|\sum_{i=0}^{m}\sum_{j=0}^{n} (P_{ij} - Q_{ij})B_{i,\tau}^{m}(u)B_{j,\xi}^{n}(v)\| \\ &\leq max(\|P_{ij} - Q_{ij}\|, \ \forall i,j), \end{aligned} \tag{5.5}$$

because the B-spline basis functions are nonnegative and sum to one.

The difference of two rational surfaces can be computed in a similar way although it is more complex and must deal with products of scalar surfaces when the two are brought to a common denominator.

From the way $\hat{R}$ is constructed it is clear that the first row of S control mesh is the same as the first row of $\hat{R}$ control mesh, that is $P_{0j} = Q_{0j}\ \forall j$. Similarly, the last row of S control mesh is the same as the last row of $\hat{R}$ control mesh, that is $P_{mj} = Q_{mj}\ \forall j$. The jth column of S control mesh will be referred to as $P_{\bullet j}$. Equation (5.5) provides a simple mechanism to bound the maximum distance between S and the ruled surface R.

Isocurves of R (and $\hat{R}$) in the ruled parametric direction have constant speed because R is linear in this parameter. The bound in equation 5.5 provides a good bound of the distance when the v isocurves of S also have constant speed, that is when $\|\frac{\partial S(u_0,v)}{\partial v}\| = c$, for all u_0.

Unfortunately, when $\frac{\partial S}{\partial v}$ is not constant, the number of ruled surfaces in the resulting approximation can be unnecessarily large in order to meet the required tolerance. A method to correct for this problem uses the fact that control points can be associated with spline node values to obtain a surface-mesh parametric relation [55]. By degree raising R into $\hat{R}$, equally spaced in Euclidean space rows are introduced into the mesh that preserves the constant speed in the ruled direction, v. Because S does not have, in general, constant speed v isocurves, one can consider unequal spacing of the introduced mesh rows. Such strategy can project a single column in the v direction of the control mesh of S, $P_{\bullet j}$, onto the linear segment connecting P_{0j} and P_{mj} which are also control points of $C_1(u)$ and $C_2(u)$ respectively (see Figure 5.9). The spacing of these projected points can then be used to place the interior control points of $\hat{R}$. Figure 5.9 demonstrates this process. Figure 5.9a has the original surface S. The control mesh of S is used in Figure 5.9b to define the mesh of $\hat{R}$, by projecting a single column of the mesh of S, $P_{\bullet j}$, onto the line connecting P_{0j} and P_{mj}. The new ruled surface, $\hat{R}$, constructed with this new spacing is shown in Figure 5.9c.

The added degree of freedom of a nonuniform v speed ruled surface approximation includes the uniform v speed ruled surface as a special case and so can always be as good approximation as the uniform speed approximation. Let $C_1(v)$ and $C_2(v)$ be two isocurves of S in the v direction. Because we consider only one column of S mesh, this strategy will be able to emulate S v speed well only if $\left|\frac{dC_1(v)}{dv}\right| \Big/ \left|\frac{dC_2(v)}{dv}\right|$ is almost constant for all v. This condition holds fairly well for large classes of surfaces, but will not necessarily hold for surfaces constructed via highly nonisometric operations such as warp [13]. However, it does eliminate the need for degree raising or refinement in the construction of $\hat{R}$, because the continuity (knot vector) of S in the v direction is inherited.

A distance bounded algorithm approximating an arbitrary tensor product sur-

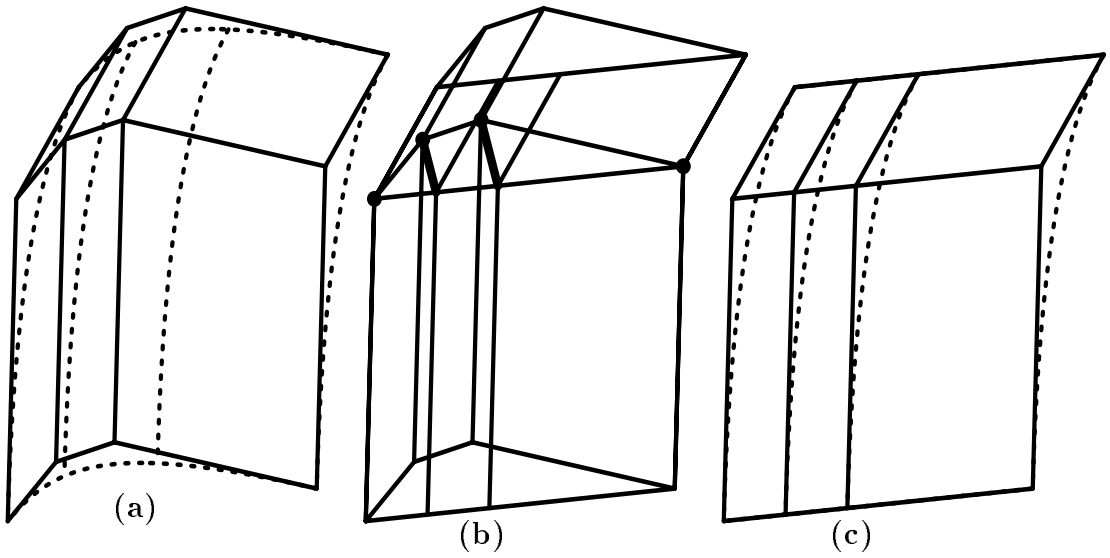


Figure 5.9. The speed of S's isocurve in the ruled direction is emulated by the ruled surface $\hat{R}$ approximating it. In (a), the jth column of S mesh, $P_{\bullet j}$, is projected in (b) onto the line connecting P_{0j} and P_{mj}. The spacing of the projected points is used to construct the mesh of $\hat{R}$'s in (c).

face as a set of ruled surfaces is derived in algorithm 5.2 based on this process.

Algorithm 5.2 returns a set of ruled surfaces that approximates the original surface S to within the required tolerance τ. Figure 5.10 shows an example of three consecutive stages of algorithm 5.2.

Assuming S satisfies a Lipschitz condition, which automatically holds for B-spline surfaces, let $(\frac{\Delta X}{\Delta v}, \frac{\Delta Y}{\Delta v}, \frac{\Delta Z}{\Delta v})$ be an upper bound on the first partial derivatives of S in the v direction. Given a finite range in the v parametric direction, $\mathcal{V}$, a bound on the Euclidean size is readily available as $(\mathcal{V}\frac{\Delta X}{\Delta v}, \mathcal{V}\frac{\Delta Y}{\Delta v}, \mathcal{V}\frac{\Delta Z}{\Delta v})$. Because algorithm 5.2 halves the parametric domain in each iteration, it halves the Euclidean bound in each iteration as well, so convergence in algorithm 5.2 is guaranteed. Note that we are concerned only with the v (ruled) direction because the representation is exact in the u direction.

Therefore, the less complex parametric direction, by some norm, may be a better candidate to select for the ruling direction approximation. Another measure for the selection of the subdivision direction may be the feasibility of the surface assembly.

Algorithm 5.2

Input:
$S(u,v)$, surface to be divided in the v parametric direction.
τ, tolerance of approximation to be used.

Output:
$\mathcal{S}$, Set of ruled surfaces, approximating $S(u,v)$ to within τ.

Algorithm:

```
RuledSrfApproximation( S, τ )
begin
  C1(t), C2(t) ⇐ Vmin and Vmax boundary of S.
  R ⇐ ruled surface between C1(t) and C2(t).
  R̂ ⇐ R refined and degree raised in v.
  If ( maxDistance( S, R̂ ) < τ )
    return { R }.
  else
  begin
    Subdivide S into two subsrfs S1, S2 along v.
    return
      RuledSrfApproximation( S1, τ ) ∪
      RuledSrfApproximation( S2, τ ).
  end
end
```

If S is an elongated tube, it may be easier to select and assemble the surfaces as sequence of rings than as a sequence of elongated strips. A third consideration may be whether the surface is closed in one direction or not. Such closed surfaces are very common, and it is very natural to approximate such surfaces as a set of rings (see Figures 5.10 and 5.11).

Once the set of ruled surfaces is determined, the surfaces must be laid flat on a plane, so they can be cut out. Lemma 5.1 can be used to verify whether the piecewise ruled surfaces are also developable. Because the isometry mapping is nonlinear, in general, an approximation must be used. We start the process by

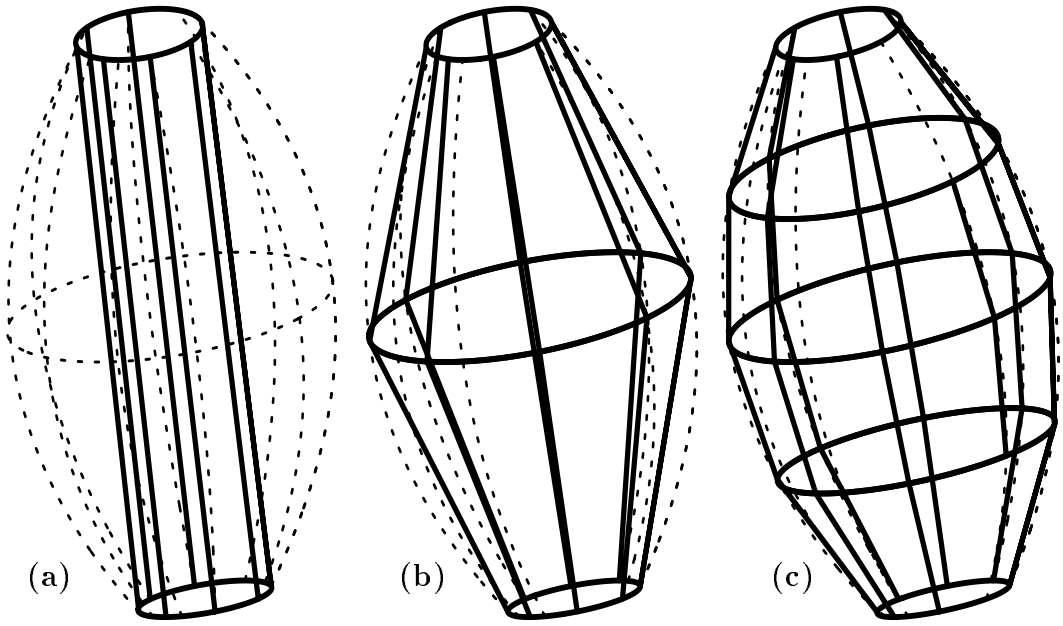


Figure 5.10. Three stages in approximating a surface with piecewise ruled surfaces.

approximating the two boundary curves of R that originated on S, $C_1(u)$ and $C_2(u)$, as piecewise linear curves $\hat{C}_1(u)$ and $\hat{C}_2(u)$, using refinement. An identical refinement should be computed and applied to both curves to insure they have the same number of linear segments, n. A one-to-one correspondence between the piecewise linear approximation of each curve is therefore established. Then, from each pair of corresponding linear segments, one from $\hat{C}_1(u)$ and one from $\hat{C}_2(u)$, a bilinear surface is created. Each bilinear is further approximated as two triangles along one of the bilinear diagonals. Finally, the $2n$ triangles are incrementally laid out and linearly transformed onto a plane (see Figure 5.12).

As stated above, the laying down of the surface is a nonlinear mapping and is only approximated. During the piecewise ruled surface approximation stage, it would be required to increase the number of ruled surfaces if a better approximation is necessary, complicating the assembly process. However, the penalty for a better layout approximation is reduced to only an enlargement of the data set.

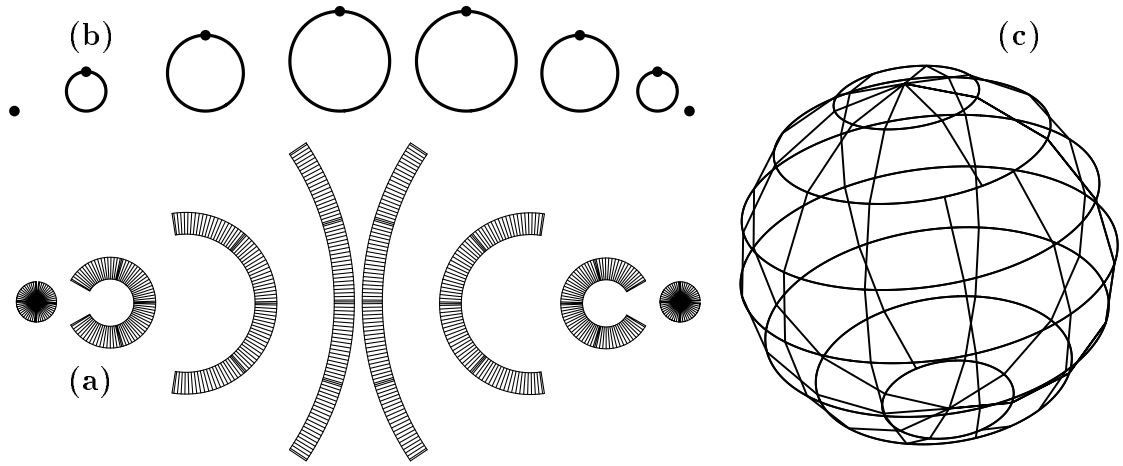


Figure 5.11. Piecewise ruled surface approximation layout of a sphere (a), its piecewise ruled surface cross sections (b), and assembled (c).

5.3.2 Extensions

It is a logical next step to improve the efficiency of algorithm 5.2 by subdividing S at v values that will minimize the number of ruled surfaces required to approximate S to within a given tolerance τ. Automatically determining candidate locations is difficult. However, a greedy approach can be adopted to determine a local minimum even though it does not guarantee global minimum in the number of ruled surfaces. The normal curvature in the v direction (the direction in which the approximating surfaces are ruled), $\kappa_n^v(u,v)$ can be computed symbolically. The maximum values of $\kappa_n^v(u,v)$ can then be used as subdivision locations. See Figure 5.13 for one such example. The normal curvature of the surface in tangent direction $\frac{\partial S}{\partial u}\frac{\partial u}{\partial t} + \frac{\partial S}{\partial v}\frac{\partial v}{\partial t}$ is

$$\kappa_n = \frac{II(a,b)}{I(a,b)} = \frac{II(\delta)}{I(\delta)} = \frac{\delta L \delta^T}{\delta G \delta^T}, \tag{5.6}$$

where $\delta = (\frac{\partial u}{\partial t}, \frac{\partial v}{\partial t}) = (a,b)$ and G and L are the matrices of the first and second fundamental forms [21, 32], respectively.

From equation (5.6), when $\delta = (0,d)$, that is, the tangent vector direction is $d\frac{\partial S}{\partial v}$,

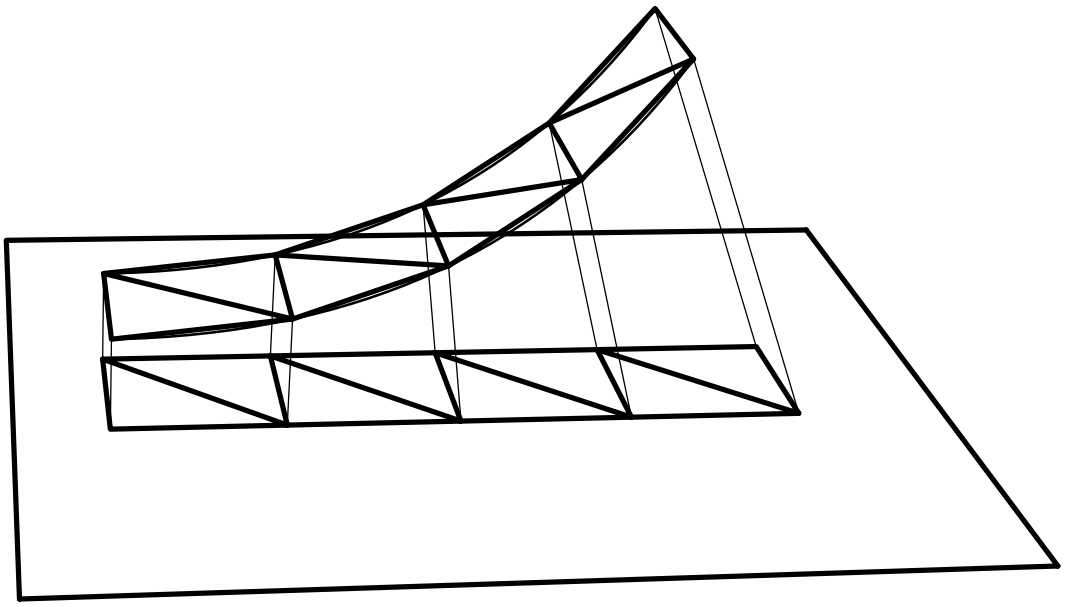


Figure 5.12. A ruled surface is approximated by triangles and unrolled onto a plane.

$$\begin{aligned}
\kappa_n^v &= \frac{(0,d)L(0,d)^T}{(0,d)G(0,d)^T} \\
&= \frac{d^2\left\langle n, \frac{\partial^2 F}{\partial v^2}\right\rangle}{d^2\left(\frac{\partial F}{\partial v}\right)^2} \\
&= \frac{\left\langle n, \frac{\partial^2 F}{\partial v^2}\right\rangle}{\left(\frac{\partial F}{\partial v}\right)^2}. \qquad (5.7)
\end{aligned}$$

Equation (5.7) is the normal curvature of the surface in the v direction. Equation (5.7) is also geometrically the curvature vector of the v iso-curve projected in the surface normal direction. For a nonarclength parameterized regular curve $C(t)$ (see [48]),

$$\kappa N = \kappa B \times T = \frac{\left(\frac{dC}{dt} \times \frac{d^2C}{dt^2}\right) \times \frac{dC}{dt}}{\left(\frac{ds}{dt}\right)^4},$$

where s is the arc length parameterization of C. Because $\langle u, (v \times w)\rangle = \langle (u \times v), w\rangle$

$$\kappa \langle N, n\rangle = \frac{\left\langle \left(\frac{dC}{dt} \times \frac{d^2C}{dt^2}\right) \times \frac{dC}{dt}, n\right\rangle}{\left(\frac{ds}{dt}\right)^4}$$

$$
\begin{aligned}
&= \frac{\left\langle \left(\frac{dC}{dt} \times \frac{d^2C}{dt^2}\right), \left(\frac{dC}{dt} \times n\right) \right\rangle}{\left(\frac{ds}{dt}\right)^4} \\
&= \frac{\left\langle \left(\frac{dC}{dt} \times n\right), \left(\frac{dC}{dt} \times \frac{d^2C}{dt^2}\right) \right\rangle}{\left(\frac{ds}{dt}\right)^4} \\
&= \frac{\left\langle \left(\frac{dC}{dt} \times n\right) \times \frac{dC}{dt}, \frac{d^2C}{dt^2} \right\rangle}{\left(\frac{ds}{dt}\right)^4} \\
&= \frac{\left\langle n, \frac{d^2C}{dt^2} \right\rangle}{\left(\frac{ds}{dt}\right)^2}, \\
&= \kappa_n^v \qquad (5.8)
\end{aligned}
$$

because $\|\frac{dC}{dt}\| = \frac{ds}{dt}$ and n is orthogonal to $\frac{dC}{dt}$.

κ_n^v can be symbolically represented as a scalar NURBs surface. Its isolated local maxima are the suggested preferred locations for the piecewise ruled surface approximation subdivision. For obvious reasons, a maximum occurring on the boundary is of no interest, but C^1 discontinuities in the v parametric direction are likely candidates for subdivision locations. Therefore, a surface should first be preprocessed and subdivided at all locations where it is not C^1 continuous. $\kappa_n^v(u,v)$ should then be computed for the resulting C^1 continuous subsurfaces. If the original surface is not C^2, κ_n^v will not even be C^0. Special care should be taken in evaluating κ_n^v along those discontinuous edges, because limits from both sides along the C^0 discontinuities would converge to different values.

An example is provided in Figure 5.13, which shows a surface with two very highly curved regions in the v direction (Figure 5.13a). Those regions are very noticeable in the $\kappa_n^v(u,v)$ (Figure 5.13b) computed for this surface. Therefore, $\kappa_n^v(u,v)$ can be used to automate the scheme to make more optimal ruled surface approximation.

In some cases, the laid out ruled surfaces can be insufficient to assemble the model, depending upon the assembly method. Some extra material might need to be

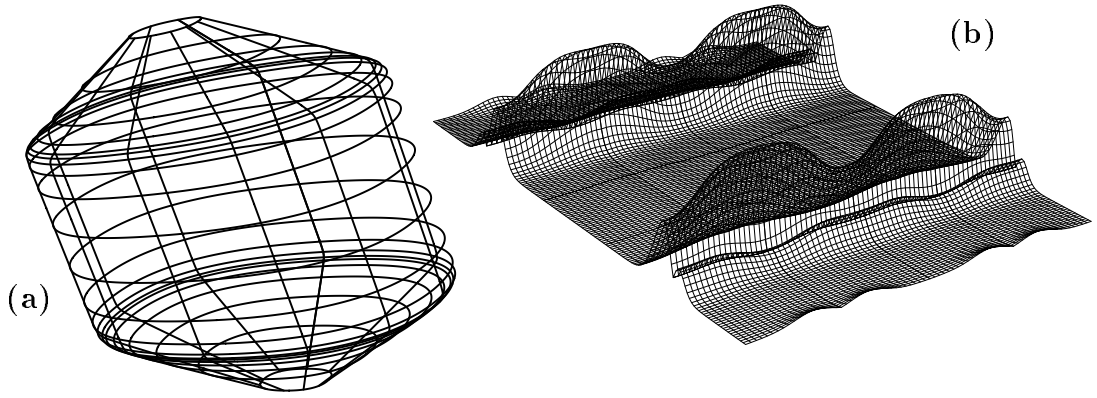


Figure 5.13. $\kappa_n^v(u,v)$ (b) is used to determine where to subdivide the surface (a).

included as stubs so the pieces may be stitched or welded together. Such stubs can be constructed by offsetting [25] the boundary curves of the planar representation of the approximating ruled surfaces. Figure 5.14 shows an examples of stubs generated using this approach that can be used for the layout of Figure 5.10 (c). However, such stubs can cause a C^0 seam between two folded developed surfaces resulting in a little stair with height equal to the material thickness. An alternative approach would be to connect two adjacent ruled surfaces using a separate stub made to span the two surfaces, from underneath, eliminating the stair.

When Boolean operators are applied to freeform models, trimmed surfaces result [47], and only part of each tensor product surface is used in the final model. In order to approximate freeform trimmed surfaces with piecewise ruled surfaces, it is necessary to position the trimming curves in the plane with the ruled surfaces. The problem is equivalent to finding the corresponding location of a specific surface Euclidean point in the planar representation of a ruled surface, given the (trimming curve) point in the surface parametric space. With the added constraint that the surface speed in the v direction must be constant to within a prespecified tolerance, locating the given (u,v) point in the planar ruled surface becomes a simplified

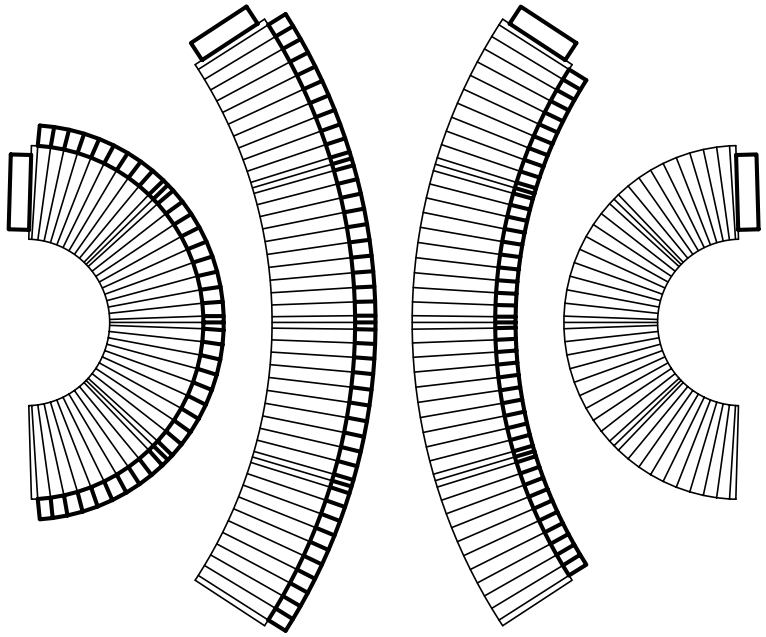


Figure 5.14. Stubs can be created by offsetting the planar boundary curves.

problem. Because the u direction is approximated as piecewise linear in the laying out stage, a binary search in u can efficiently reveal the bilinear segment containing the point. Within the bilinear surface the u direction is also assumed to be of constant speed and the exact location is then interpolated from the flat bilinear four corner points. Finally, because the ruled surface representation is only an approximation, it may be desired to re-execute the Boolean operations on the ruled surface approximations and create the appropriate trimming curves for the fabrication surfaces instead of the original surfaces because the intersections curves are not identical. Figure 5.15 shows a simple layout with trimming curves. Section 5.3.3 provides several examples of more complex models composed of trimmed surfaces as well.

5.3.3 Examples

The algorithm developed was used to generate layouts for several computer models, automatically. Figure 5.11 shows the sphere layout on a plane with its 3

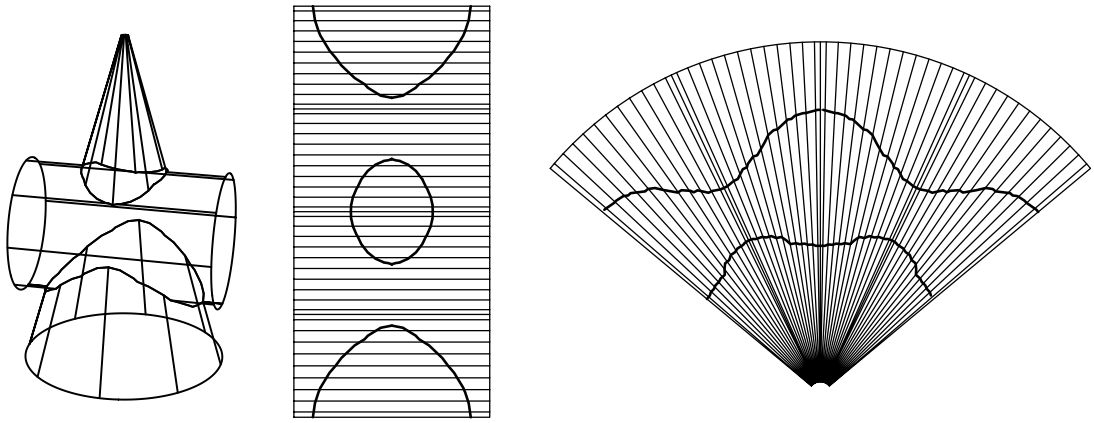


Figure 5.15. Trimming curves should be laid out with the ruled surfaces.

dimensional piecewise ruled surface approximation. Figure 5.14 shows the layout of the model in Figure 5.10 with an example of stubs. Figure 5.15 shows the layout of a cone and a cylinder intersecting each other with their trimming curves. Figure 5.16 shows a helicopter model [14], its layout projection with the ruled surface cross sections, and the assembled piece.

Figure 5.17 shows several models layed out using these techniques and then assembled from heavy paper. Each developable surface was cut from paper and folded into its 3-space shape. Paper connecting stubs were used to hold and keep the pieces together.

More complex models can be created using Boolean operations when the model is a union or intersection of several freeform surfaces. The layouts of the trimming curves of these surfaces are also computed, in a way similar to the ruled surface layouts. Figures 5.18 and 5.19 show more complex models having several trimmed surfaces.

Table 5.2 provides some timing results for the model decomposition and layout computation. Tests were run on a SGI4D 240 GTX (R3000 25MHz Risc machine). All tests are measured in seconds.

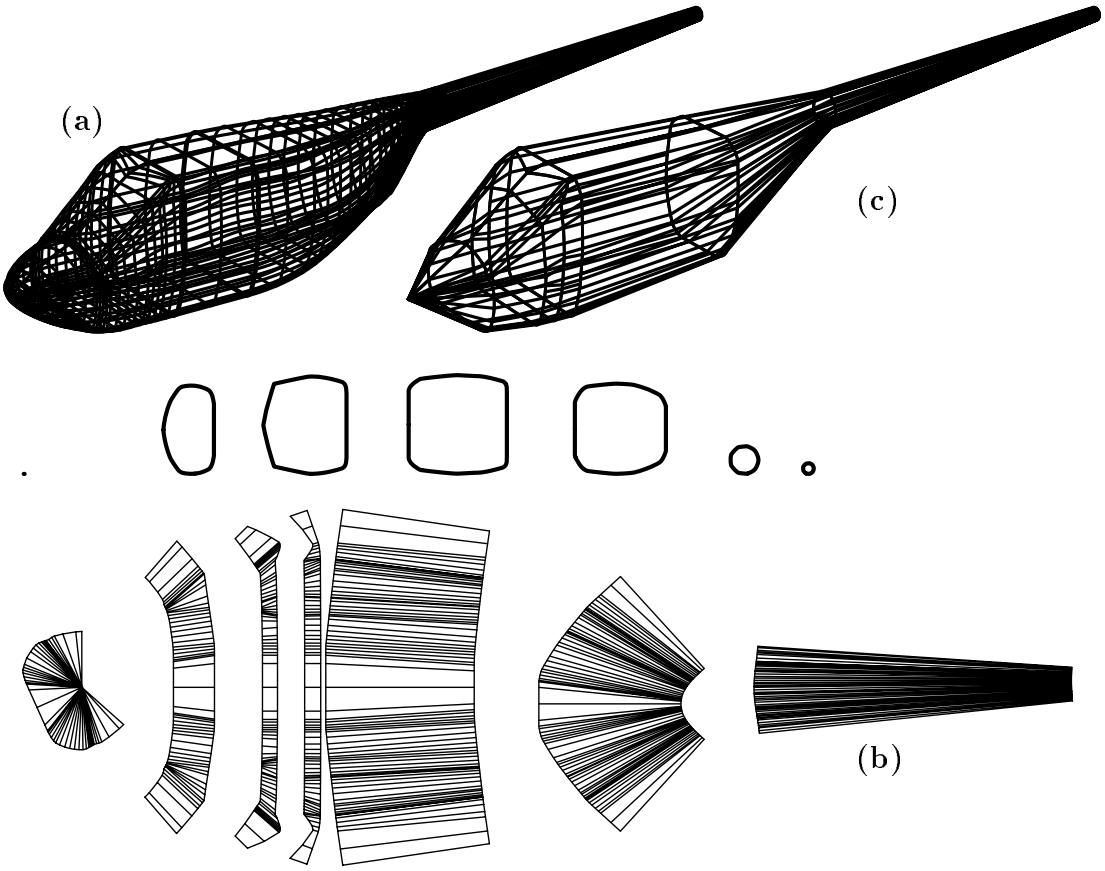


Figure 5.16. A helicopter model (a) laid out (b) and assembled (c).

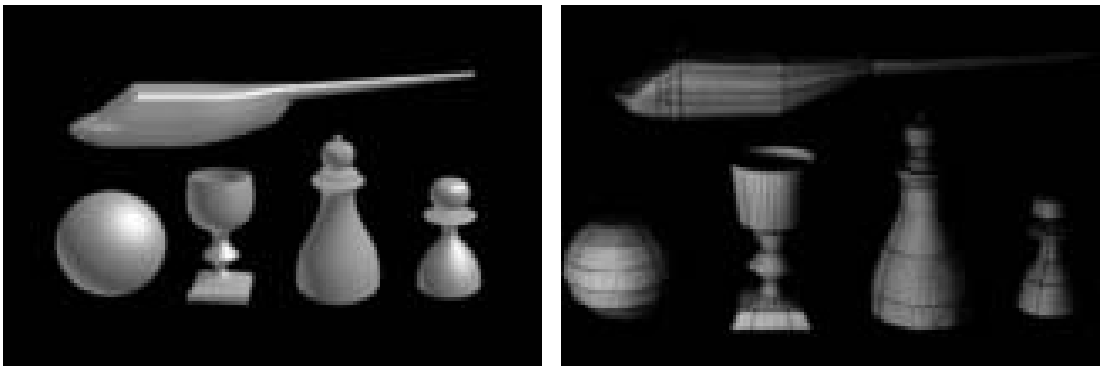
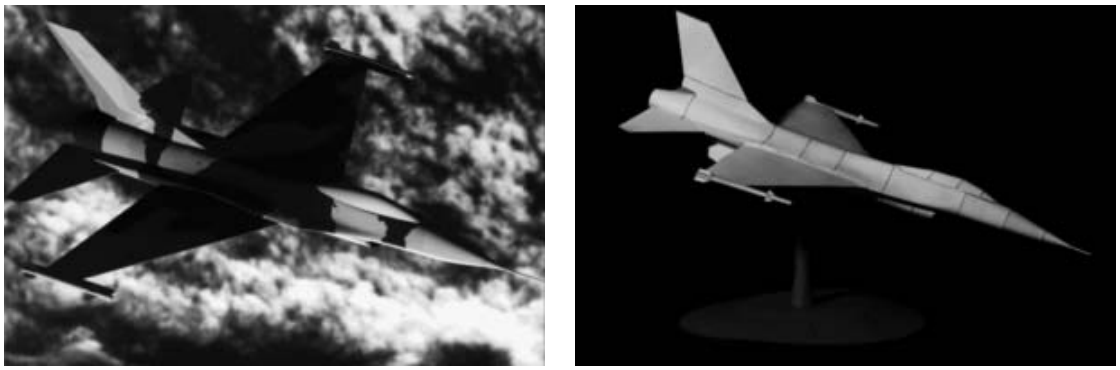


Figure 5.17. Computer models (a) and assembled out of heavy paper (b).

(a) (b)

Figure 5.18. Teapot computer model (a) and assembled out of heavy paper (b).



(a) (b)

Figure 5.19. Computer model of an f16 (a) and assembled out of heavy paper (b).

Table 5.2. Different models layout construction times.

Model	Time (Sec.)
Tube (Figure 5.14)	1.4
Helicopter (Figure 5.16)	7
Pawn (Figure 5.17)	3
Teapot (Figure 5.18)	5
f16 (Figure 5.19)	150

CHAPTER 6

OTHER APPLICATIONS

Nothing is particularly hard if you divide it into small jobs.

Henry Ford

This Chapter will present several other applications that can benefit from combined symbolic and numeric computation. Some of these problems have undergone extensive research and are provided here to reflect on the power of this combination. In section 6.1 we develop an adaptive technique to approximate higher order Bézier curves using cubic Bézier curves. In section 6.2 we develop the tools so the composition operation may be added to the set of operation defined in Chapter 2. The composition tool will open the way for solving a whole set of problems. In section 6.3, we develop techniques for visualizing surface slopes and steepnesses. The steepness of a model may be of interest when only a limited set of slopes is allowed. This is important for road design or even for slides. Section 6.4 discusses more surface properties. The speed of the surface has a direct affect on the way the surface is milled. It also provides a bound on the amount of Euclidean movement while moving a fixed distance in parametric space. The twist is another measure for a surface shape and is not as intuitive as one would like, as will be shown. Like curvature estimation methods, analysis techniques of twist have been previously based on a presampled grid from the surface parametric space [3]. As in Chapter 4, we will demonstrate in section 6.5 the use of property surfaces to globally bound the twist properties.

6.1 Bézier Curve Approximation

Cubic polynomial curves are frequently used in graphics and CAGD. The fact that piecewise cubic polynomial curves are the curves with the lowest order that can provide C^2 continuous interpolation or approximation is one of the main reasons.

Because any polynomial basis may be used, we select the Bernstein polynomial which is numerically stable [31].

One can find a growing number of hardware implementations for evaluating cubic polynomials in modern workstations and devices, for mainly display purposes. Taking advantage of these implementations speeds up algorithms. Converting other types of curves into this simple form is not always obvious. Higher order curves cannot, in general, be represented as cubic polynomials. This is also the case for rational curves. Even rational quadratic curves are not representable as cubic polynomials. In other words, approximation techniques must be used. The Postscript [53] language is an example in which only cubic polynomial are supported. Therefore NURBs curves or even rational Bézier curves must be approximated to display them on Postscript devices.

Currently, the most common technique is to refine the curves and approximate them as piecewise linear curves which are then displayed. Because linear segments are displayable by almost every device, portability is gained. However, this method suffers from two major drawbacks. First, the size of the data is huge - several magnitudes larger than the original curves. Furthermore, the data are not exact any more and are not even C^1 continuous. Approximating higher order or rational curves as cubic polynomials is probably a better approach. In [38, 39], a subdivision based approach is used to create such an approximation. A cubic polynomial is compared to the curve being approximated. If the cubic polynomial is not accurate enough, the curve is subdivided and the two new cubic approximations are compared to the two parts of the original curve.

In this section, we enhance this technique so that the approximating cubic piecewise polynomials join with C^1 continuity are everywhere within a prescribed tolerance to the original curve everywhere.

Because we choose to derive this approximation only for with higher order or rational Bézier curves, NURBs curves will be preprocessed and converted to an ordered set of rational Bézier curves of the same order.

In Chapter 3, we introduced a global method to compute the distance between a curve and its offset approximation. The same technique may be used here to compute the distance between a curve and its approximation (superscript denotes order, subscript a denotes approximation):

Algorithm 6.1 provides a piecewise cubic approximation to a given curve in line (1). A cubic polynomial curve has twelve degrees of freedom (four E^3 points). Six of them are used to interpolate the original curve end points. Because we preserve end points tangents (to easily preserve C^1 continuity), two degrees of freedoms are left - the speeds of the tangents. One can use the original curve speed to provide the information to determine the last two degrees of freedom, a necessary condition from the way $\delta(t)$ is computed (see below). This approach was used in Figures 6.1 and 6.2.

Raising the order of Bézier curves as done in line (2) in algorithm 6.1 is a fairly simple task (see Chapter 2, equation (2.9)).

The subdivision (line (3), algorithm 6.1) exploits the distance function, $\delta(t)$, and instead of subdividing in the middle of the parametric domain, a common technique, the curve is subdivided at the location of the maximum distance (error). Because the end points of the piecewise cubics interpolate the original curve, this error is automatically reduced to zero.

A different approach is to adjust the tangent speeds to minimize the distance between the original curve and it approximation, in a similar way to that of offsets in 3.2. This approach needs further investigation.

Algorithm 6.1

Input:
τ, required approximation curve tolerance.
$C^n(t)$, input curve of order n.

Output:
$\mathcal{C}$, set of piecewise cubic Bezier curves, $C_a^3(t)$ approx. $C^n(t)$.

Algorithm:
CubicBezierApprox$(C^n(t), \tau)$
begin
(1) $C_a^3(t) \Leftarrow$ cubic Bezier approximation to $C^n(t)$.
(2) Raise $C_a^3(t)$ order to order n: $C_a^n(t)$.
Compute distance $\delta(t)$ between $C_a^n(t)$ and $C^n(t)$.
if ($\delta(t)$ maximum distance $> \tau$) Do
begin
(3) Subdivide $C^n(t)$ at $\delta(t)$ maximum into $C_1^n(t)$, $C_2^n(t)$.
return **CubicBezierApprox**$(C_1^n(t), \tau) \cup$
CubicBezierApprox$(C_2^n(t), \tau)$.
end
else
return $C_a^3(t)$.
end

In some cases, when approximating shape of a higher order curve using a lower one, the speeds of the curve is irrelevant. One such case is display on Postscript devices, in which the only requirement is to preserve the curve shape. For each cubic Bézier and its corresponding higher order subcurve that it approximates, algorithm 6.1 guarantees

1. End points interpolate the original higher order subcurve,

2. End point tangents are in the same direction as the original higher order subcurve tangents (G^1), and possibly with the same length (C^1),

3. The distance between the two curves is within the specified tolerance, and

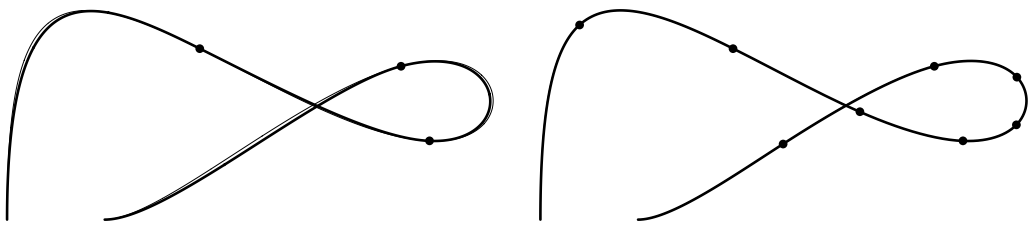


Figure 6.1. Cubic Bézier approximation to higher order curves (two tolerance).

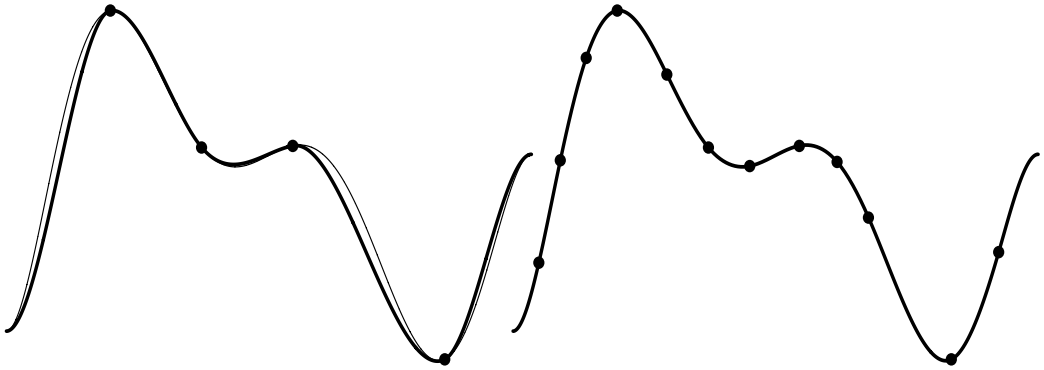


Figure 6.2. Cubic Bézier approximation to higher order curves (two tolerance).

4. The speed of both curves is the same to within the specified tolerance.

The first three properties are required in order to mimic the curve shape. However, property 4 is only a result of the way the distance, $\delta(t)$, between the two curves is computed. If one could efficiently answer whether the two curves are within the desired tolerance, this constraint could be omitted. Unfortunately, no such global and efficient algorithm exists and, as a result, the distances are computed with the respective parameter values and curve speed is preserved as well. Modifying the tangent lengths directly affects the speed of the curve so a different distance measure is needed.

6.2 Composition

Composition, $f \circ g$, is a powerful operation that has not found much use in graphics and CAGD yet. Some work can be found on the implicit use of composition

in deformations [19, 61]. The bivariate surface g is warped in a field defined as a trivariate volume f, resulting in a bivariate composed and deformed surface. This deformation, implicitly using composition, can serve as a modeling tool as well and can provide exact and well behaved results. In [11], a parametric curve $g = (u(t), v(t))$ is composed with the surface $f(u, v)$ to find the exact Euclidean representation of the curve $f(u(t), v(t))$, to be used as a fillet boundary curve in fillet construction. $f(u(t), v(t))$ can be used anywhere the exact representation of the Euclidean curve is needed, given the parametric curves. Mapping trimming curves from parametric space to the Euclidean space is another example.

In this section we will explore the composition $f(u(t), v(t))$. Extending this to a trivariate deformation volume $f(u(r, s), v(r, s), w(r, s))$ is simple.

Let $C(t) = (u(t), v(t))$ be a Bézier curve such that $u(t) \in (0, \ldots, 1), \forall t$ and $v(t) \in (0, \ldots, 1), \forall t$. Let S be a Bézier surface.

$$\begin{aligned} S(u(t), v(t)) &= \sum_{i=0}^{n} \sum_{j=0}^{m} P_{ij} \mathbf{B}_j^m(v(t)) \mathbf{B}_i^n(u(t)) \\ &= \sum_{i=0}^{n} \left(\sum_{j=0}^{m} P_{ij} \mathbf{B}_j^m(v(t)) \right) \mathbf{B}_i^n(u(t)). \end{aligned} \tag{6.1}$$

The curve-surface composition is now narrowed to the problem of computing the composition of $\mathbf{B}_i^n(c(t))$, where $c(t)$ is a scalar curve. Assuming one can compute and represent the composition $\mathbf{B}_i^n(c(t)), c(t) \in [u_{min}, u_{max}]$, as a curve, the curve $S(u(t), v(t))$ is also representable because it involves in scaling, addition and multiplication of $\mathbf{B}_i^n(c(t))$ terms only. These operations were explored in Chapter 2.

$$\mathbf{B}_i^n(c(t)) = \binom{n}{i} (1.0 - c(t))^{n-i} (c(t))^i. \tag{6.2}$$

Interestingly enough, equation (6.2) contains only tools developed in Chapter 2 namely, curve addition and curve multiplication (power).

What if either the curve or the surface is rational? For a rational surface, nothing changes. The P_{ij} in equation (6.1) should simply be treated as in projective space. If the curve is rational, equation (6.2) now becomes

$$\begin{aligned} \mathbf{B}_i^n(c(t)) &= \binom{n}{i} \left(1.0 - \frac{c(t)}{w(t)}\right)^{n-i} \left(\frac{c(t)}{w(t)}\right)^i \\ &= \binom{n}{i} \frac{(w(t) - c(t))^{n-i}(c(t))^i}{(w(t))^n}. \end{aligned} \tag{6.3}$$

Equation (6.3) should then be substituted into equation (6.1) in a similar way to equation (6.2). If surface $S(u, v)$ is rational as well the denominator term in equation (6.3), $(w(t))^n$, is canceled because it appears in both the surface numerator and denominator. If however, the surface was a polynomial, the resulting composed curve becomes rational.

Figures 6.3 and 6.4 show some examples for Bézier curves and surfaces. Figure 6.3 has a polynomial surface and several parametric curves mapped onto the surface. Figure 6.4 has a surface which is an extrusion of an arc and, as such, is rational. Both Figures have the parametric space on the left and the Euclidean mapping on the right.

Unfortunately, the order of the resulting composed curves is quite high. Let d be the curve degree while the surface degrees are m and n as can be seen from equation (6.1). It immediately follows from equations (6.1) and (6.2) that the degree of the composed curves is equal to $dn + dm$. Table 6.1 provides these orders for common cases. Note that even when either the surface or the curve is rational (or both), the order of the resulting curve does not change.

In some cases, it can be important to reparametrize a curve. The composition tool allows exactly that. By substituting s in $C(s)$ by $s = c(t)$, we reparametrize a curve to:

$$C(c(t)) = \sum_{i=0}^{n} P_i \mathbf{B}_i^n(c(t)). \tag{6.4}$$

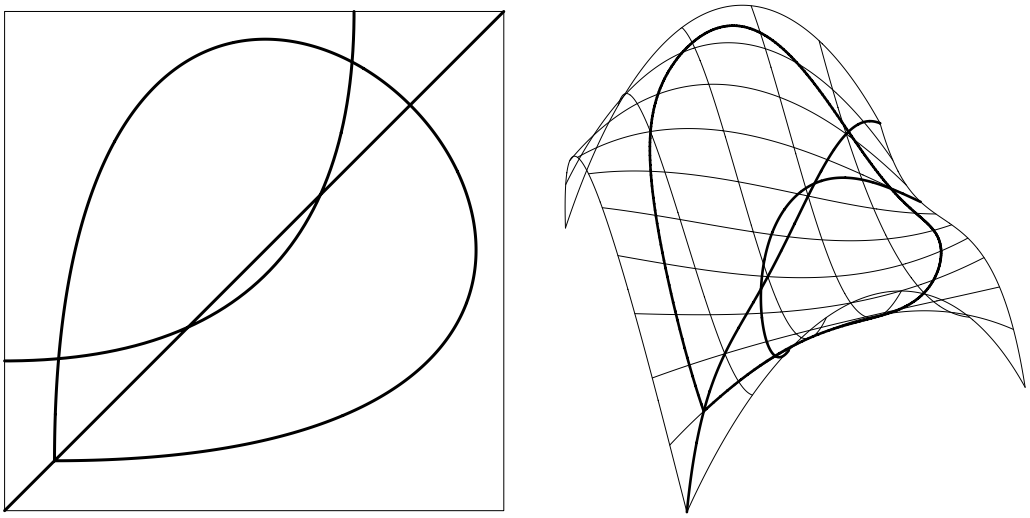


Figure 6.3. Bézier curve (polynomial) surface composition.

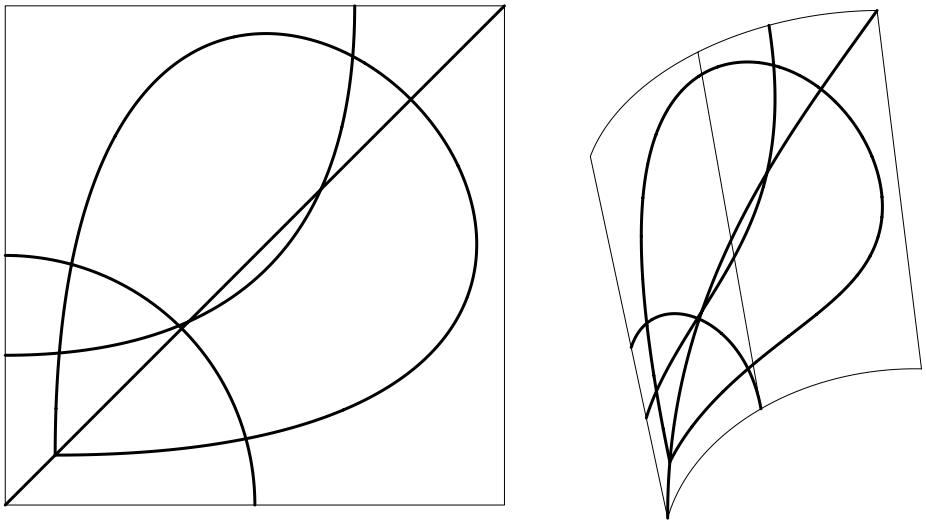


Figure 6.4. Bézier curve (rational) surface composition.

Computing equation (6.4) involves scaling (with P_i) and addition of curves resulting from the composition of $\mathbf{B}_i^n(c(t))$, which we dealt with in equations (6.2) and (6.3).

Figure 6.5 demonstrates the speed of three arcs after reparametrizing with $c(t) = t^2$ in the middle and reparametrizing with $c(t) = t^4$ on the right. The original arc is on the left.

Table 6.1. Curve on surface composition - orders.

Surface orders	Curve order	Composed curve order
3×3	2	5
3×3	3	9
3×3	4	13
4×4	2	7
4×4	3	13
4×4	4	19
3×4	2	6
3×4	3	11
3×4	4	16

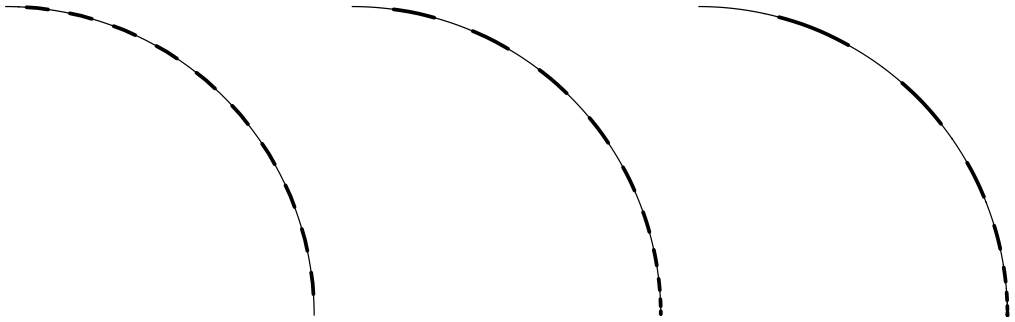


Figure 6.5. Rational Bézier curve reparametrizing using composition.

6.3 Surface Steepness

The slope of a planar curve at a given point is equal to the angle between the tangent to the curve and a reference line, usually the horizontal axis. In an analogous way we define the *surface slope* at a given point, p, as the angle between the plane tangent to the surface at p and a reference plane. Without loss of generality, in the discussion below we assume that the reference plane is the xy plane.

Because the angle between two planes is equal to the angle between their two normals, to compute surface slope, one need only compute the angle between the surface normal and the z axis. Let n be the surface unit normal and let n_z be its

z component. Then, the tangent of the slope angle $\mathcal{S}$ is equal to:

$$tan(\mathcal{S}) = \frac{\sqrt{1 - n_z^2}}{n_z}. \tag{6.5}$$

When $n_z = +1$ the surface orientation is horizontal. If $n_z = 0$ the surface is vertical, and finally if $n_z = -1$ that surface is horizontal again, but this time facing down.

Inspection of the surface unit normal equation shows that $n(u, v)$ cannot be computed directly using the symbolic tools of Chapter 2 because of the need to determine the square root. However, the z component of the unnormalized normal surface, $\hat{n}$, is equal to:

$$\hat{n}_z(u, v) = \frac{\partial x(u, v)}{\partial u}\frac{\partial y(u, v)}{\partial v} - \frac{\partial y(u, v)}{\partial u}\frac{\partial x(u, v)}{\partial v}, \tag{6.6}$$

where $x(u, v)$ and $y(u, v)$ are the x and y components of surface $S(u, v)$, and $n_z(u, v) = \hat{n}_z(u, v)/\|\hat{n}(u, v)\|$, where $\|\hat{n}(u, v)\|$ is the magnitude of $\hat{n}(u, v)$.

Even though $n_z(u, v)$ contains a square root factor, it can be squared and $n_z(u, v)^2$ can be represented as a rational function.

Given a slope $\mathcal{S}$ in degrees (or radians), binding $n_z^2(u, v)$ is straightforward using equation (6.5). Therefore, given a certain slope $\mathcal{S}$, one can compute n_z and n_z^2 using equation (6.5). Because n_z^2 is representable using (piecewise) rationals, one can contour this surface to find the specified n_z^2 levels. Figures 6.6 and 6.7 demonstrate this exact process.

Alternatively, one can use the symbolically computed property $n_z^2(u, v)$ as a scalar map designating the color of the surface at each location, much like a texture map. Figure 6.8 is an example for this approach, for the same surface as in Figure 6.7.

The technique presented here has also been used to compute silhouette curves of surfaces [22], and is equivalent to the zero set of equation (6.6). $\hat{n}_z(u, v)$ is symbolically computed and its intersection (contouring) with the plane $z = 0$ provides

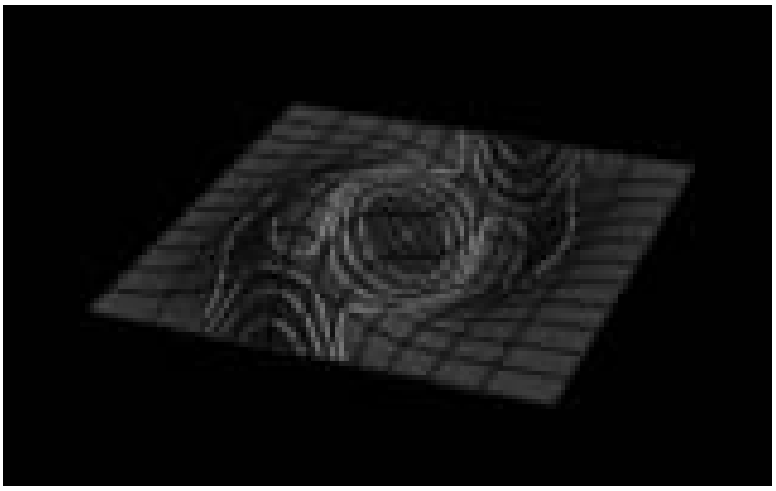


Figure 6.6. Different Steepness regions example

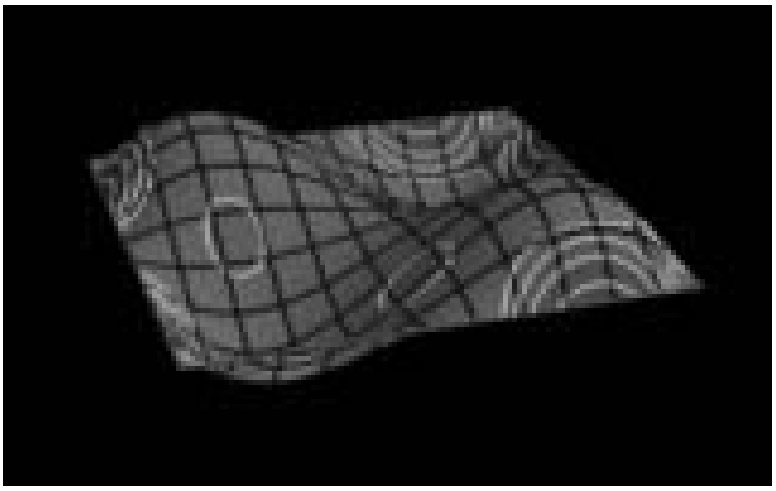


Figure 6.7. Different Slope or Steepness regions of the surface

the silhouette curves in parametric space for the specified speeds. Figure 6.9 shows one such example.

Unlike curvature, slope is not an intrinsic surface property. In fact, because it is orientation dependent, it provides the designer with a measure on the planarity of the surface in a specific orientation.

6.4 Surface Speed

The speed of a curve is defined as the distance moved in Euclidean space per unit of movement in parameter space. For a curve,

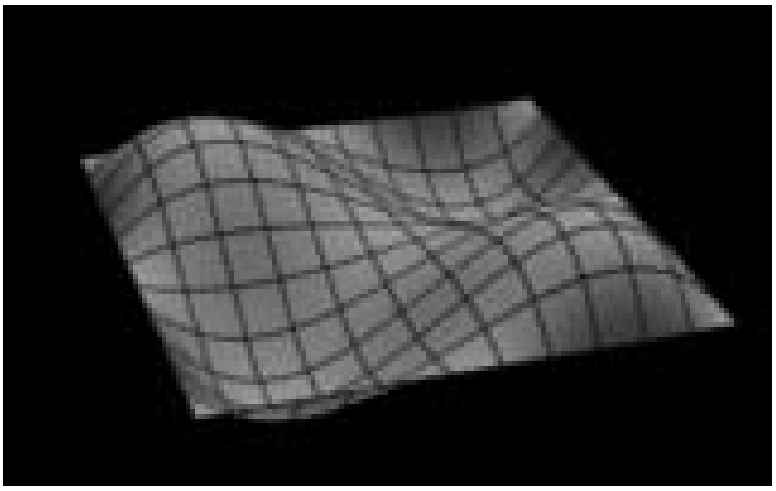


Figure 6.8. Continuous steepness of the surface in Figure 6.7

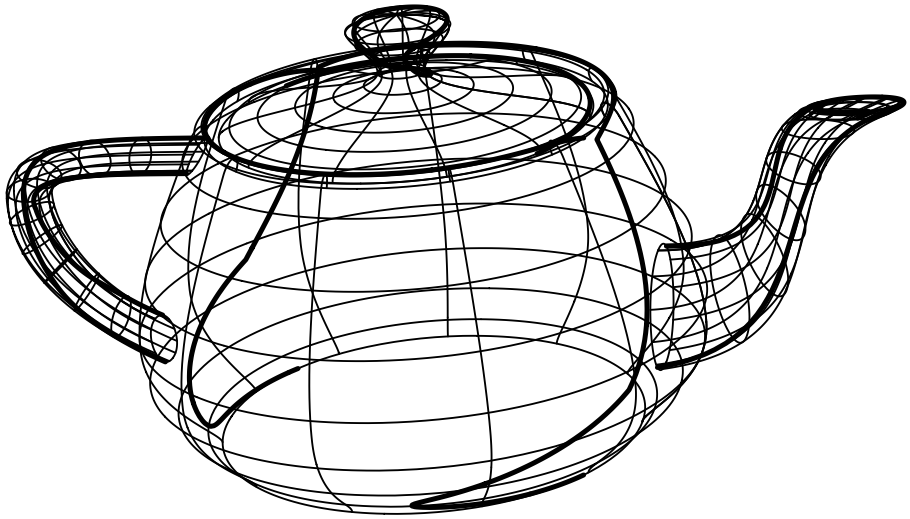


Figure 6.9. Silhouettes are equivalent to the zero set of equation (6.6) (rotated).

$$\begin{aligned} \mathcal{S}(t) &= \left\| \frac{dC(t)}{dt} \right\| \\ &= \sqrt{\left(\frac{dx}{dt}\right)^2 + \left(\frac{dy}{dt}\right)^2 + \left(\frac{dz}{dt}\right)^2}. \end{aligned} \tag{6.7}$$

We define the *speed bound* of surface $S(u, v)$ as the supremum of the speeds of all curves on the unit circle of the tangent plane using the first partials as a basis.

Let $\alpha(t)$ be a curve in the parametric domain of $S(u, v)$, i.e., $\alpha(t) = (u(t), v(t))$. By providing this speed bound of the surface parametrization, one can compute

certain properties on $\alpha(t)$ and use the speed bound to extrapolate and provide bounds on the properties on the composed curve $S \circ \alpha = S(u(t), v(t))$.

Let $\gamma(t)$ be an auxiliary arc length parametrized curve with its image in the parametric space of $S(u, v)$, i.e., $\gamma(t) = (u(t), v(t))$, with $\sqrt{\left(\frac{du}{dt}\right)^2 + \left(\frac{dv}{dt}\right)^2} = 1$, for all t. Then

$$\begin{aligned}
& \left\| \frac{dS(u(t), v(t))}{dt} \right\|^2 \\
= & \left\| \frac{\partial S}{\partial u}\frac{du}{dt} + \frac{\partial S}{\partial v}\frac{dv}{dt} \right\|^2 \\
= & \left(\frac{\partial x}{\partial u}\frac{du}{dt} + \frac{\partial x}{\partial v}\frac{dv}{dt} \right)^2 + \left(\frac{\partial y}{\partial u}\frac{du}{dt} + \frac{\partial y}{\partial v}\frac{dv}{dt} \right)^2 + \left(\frac{\partial z}{\partial u}\frac{du}{dt} + \frac{\partial z}{\partial v}\frac{dv}{dt} \right)^2 \\
\leq & \left(\frac{\partial x}{\partial u} \right)^2 + \left(\frac{\partial x}{\partial v} \right)^2 + \left(\frac{\partial y}{\partial u} \right)^2 + \left(\frac{\partial y}{\partial v} \right)^2 + \left(\frac{\partial z}{\partial u} \right)^2 + \left(\frac{\partial z}{\partial v} \right)^2, \qquad (6.8)
\end{aligned}$$

because

$$\begin{aligned}
\left(\frac{\partial x}{\partial u}\frac{du}{dt} + \frac{\partial x}{\partial v}\frac{dv}{dt} \right)^2 &= \left(\left(\frac{\partial x}{\partial u}, \quad \frac{\partial x}{\partial v} \right) \cdot \left(\frac{du}{dt}, \quad \frac{dv}{dt} \right) \right)^2 \\
&= \left\| \left(\frac{\partial x}{\partial u}, \quad \frac{\partial x}{\partial v} \right) \cdot \left(\frac{du}{dt}, \quad \frac{dv}{dt} \right) \right\|^2 \\
&\leq \left\| \left(\frac{\partial x}{\partial u}, \quad \frac{\partial x}{\partial v} \right) \right\|^2 \left\| \left(\frac{du}{dt}, \quad \frac{dv}{dt} \right) \right\|^2 \\
&= \left\| \left(\frac{\partial x}{\partial u}, \quad \frac{\partial x}{\partial v} \right) \right\|^2 \\
&= \left(\frac{\partial x}{\partial u} \right)^2 + \left(\frac{\partial x}{\partial v} \right)^2 \qquad (6.9)
\end{aligned}$$

If $\alpha\frac{\partial S}{\partial u} = \frac{\partial S}{\partial v}$ (see Figure 6.10 with collinear partials along the surface boundary, which implies the surface is not regular there) and $\alpha\frac{du}{dt} = \frac{dv}{dt}$, then

$$\begin{aligned}
\left\| \left(\frac{\partial x}{\partial u}, \quad \frac{\partial x}{\partial v} \right) \cdot \left(\frac{du}{dt}, \quad \frac{dv}{dt} \right) \right\|^2 &= \left\| \left(\frac{\partial x}{\partial u}, \quad \alpha\frac{\partial x}{\partial u} \right) \cdot \left(\frac{du}{dt}, \quad \frac{dv}{dt} \right) \right\|^2 \\
&= \left\| \frac{\partial x}{\partial u}(1, \quad \alpha) \left(\frac{du}{dt}, \quad \frac{dv}{dt} \right) \right\|^2 \\
&= \left| \frac{\partial x}{\partial u} \right|^2 (1 + \alpha^2)
\end{aligned}$$

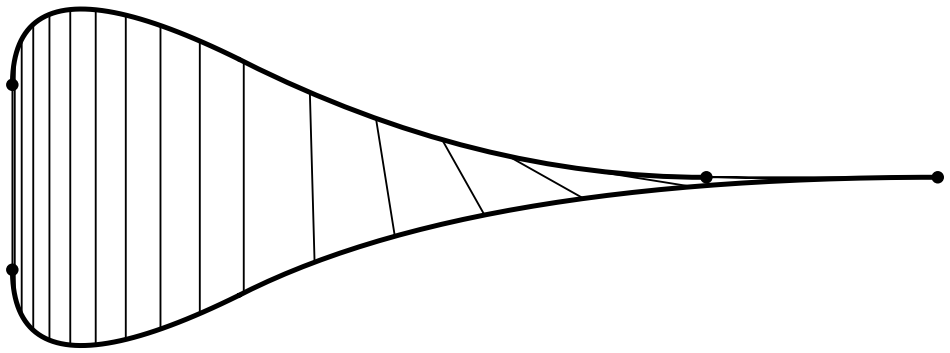


Figure 6.10. Degenerated boundary provides the two extremes on speed bound.

$$= \left(\frac{\partial x}{\partial u}\right)^2 + \left(\frac{\partial x}{\partial v}\right)^2, \tag{6.10}$$

and the upper bound established in equation (6.8) is reached. Therefore this bound is minimal.

Because it is not possible to represent the square root of equation (6.8) as a (piecewise) rational surface, in general, we compute instead

$$\hat{\mathcal{S}}(u,v) = \left(\left(\frac{\partial x}{\partial u}\right)^2 + \left(\frac{\partial y}{\partial u}\right)^2 + \left(\frac{\partial z}{\partial u}\right)^2 + \left(\frac{\partial x}{\partial v}\right)^2 + \left(\frac{\partial y}{\partial v}\right)^2 + \left(\frac{\partial z}{\partial v}\right)^2\right) \tag{6.11}$$

Figures 6.11 and 6.12 are two examples of using $\hat{\mathcal{S}}(u,v)$ to compute a speed bound on the surface.

The speed surface can be used to provide a measure on the quality of the parametrization. This can becomes especially important if the surface is to be evaluated (for any purpose, including rendering) at a predefined set of parameter values.

6.5 Variations on Surface Twist

Also interesting is the ability to visualize surface twist. Basically the twist is defined as the cross derivative component:

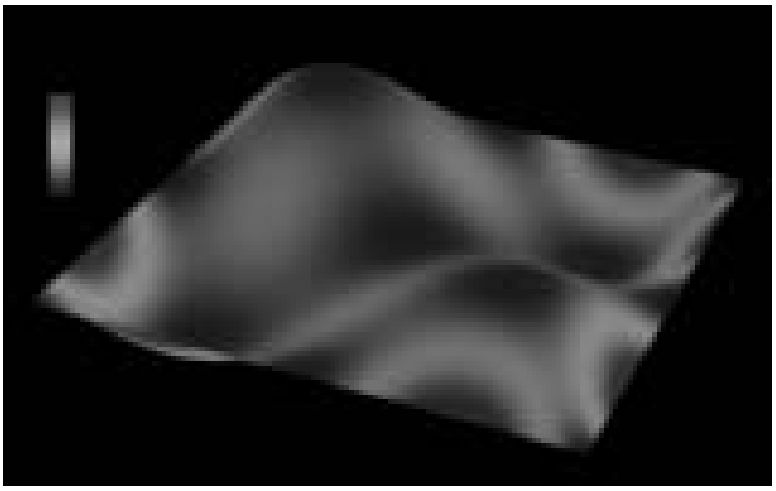


Figure 6.11. Parametrization speed estimate (same surface as Figure 6.7).

Figure 6.12. Parametrization speed estimate for the teapot model.

$$\mathcal{T}(u,v) = \frac{\partial^2 S(u,v)}{\partial u \partial v}. \tag{6.12}$$

This equation is representable and can always be computed symbolically for (piecewise) rationals. Figures 6.13, 6.14, and 6.15 shows this property as a texture mapped on the surfaces.

Using equation (6.12) as a twist measure has a major drawback as can be seen in Figure 6.14. Even though the surface is flat, the twist component is not zero because the speed of the parametrization is changing. In other words, the mapping

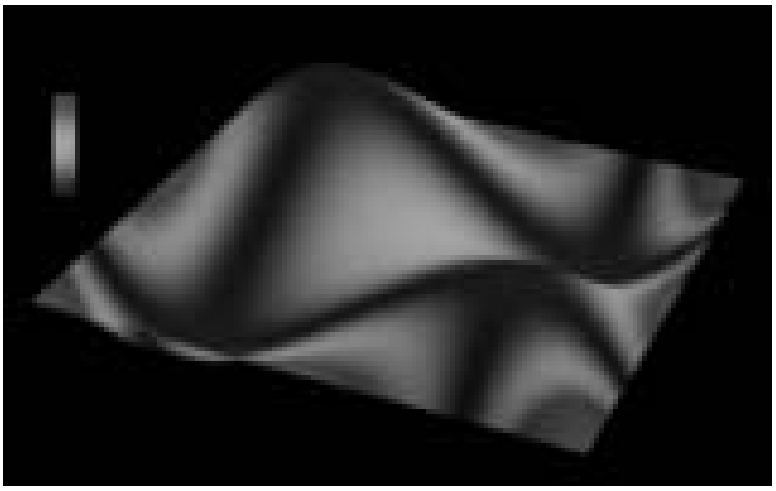


Figure 6.13. Twist component of a surface (same surface as Figure 6.7).

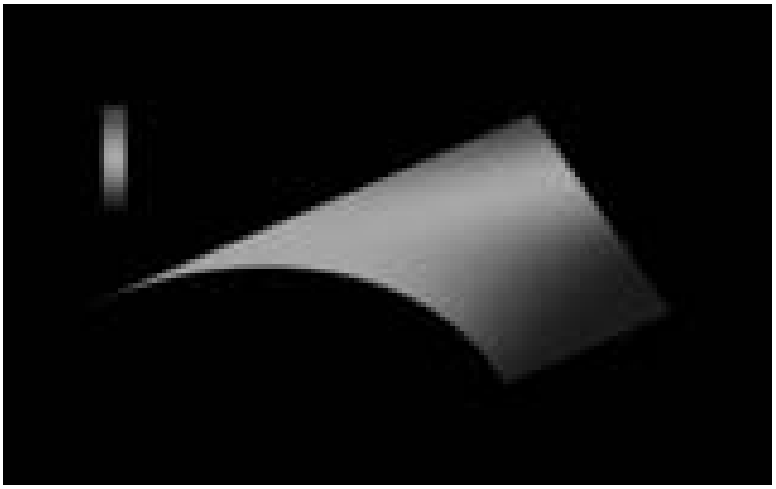


Figure 6.14. Twist component of a flat surface.

from the parametric space to the Euclidean space is not isometric. It would be more helpful to use the twist component in only the surface normal direction (see [3]) to eliminate the twist as a result of a nonisometric mapping.

$$l_{12} = l_{21} = \left(n, \frac{\partial^2 F}{\partial u \partial v} \right) \tag{6.13}$$

where l_{12}, and $= l_{21}$ are two of the components of second fundamental form, L (see Chapter 2).

Figure 6.15. Twist component of the teapot model.

Obviously, this time the l_{12} component in the flat surface in Figure 6.14 is zero showing no twist in the normal direction. Furthermore, the use of this property showed that the teapot has virtually no twist in the normal direction as well. All the twist in Figure 6.15 was a result of the nonisometric mapping. Figure 6.16 shows a nonplanar surface, similar to the one in Figure 6.14 using l_{12} as property surface mapping colors onto the surface, as texture.

Because now one can compute both the total twist (equation (6.12)), and the twist in the normal direction (equation (6.13)), one can consider computing the twist in the tangent plane to the surface as the difference of the two quantities. This difference would provide another measure as to the quality of the surface parametrization.

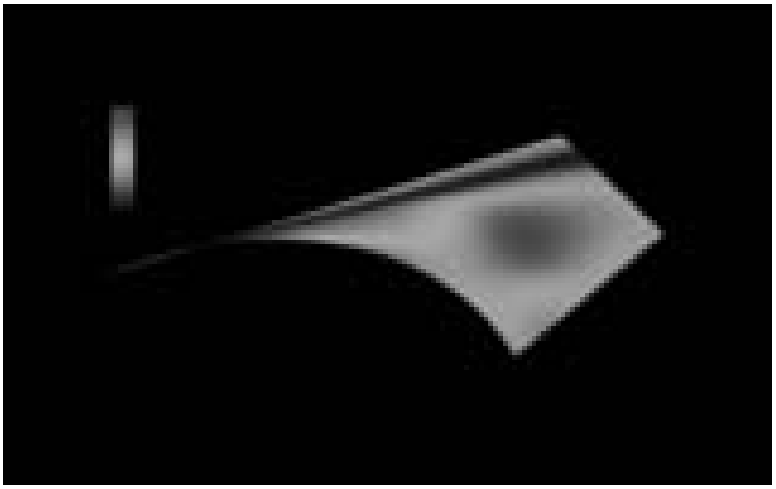


Figure 6.16. Twist component of a nonplanar twisted surface.

CHAPTER 7

CONCLUSIONS

Computers are useless; they can only give answers.

Picasso

It is our hope that symbolic computation will find its way as a useful tool in computer aided geometric design and in computer graphics. We hope that the symbolic approach developed throughout this work has demonstrated the capability and usefulness of this representation.

Several questions were left unanswered in this research. Detecting and isolating self-intersection in offsets of curves and surfaces is a difficult unsolved problem [51]. This thesis introduced a new robust method to isolate self intersections occurring in curve offsets. Extending the self-intersection isolation to surface offsets is difficult and is still a future research topic.

The work presented in Chapter 4 makes it practical to use second order surface analysis as a tool to support the development of robust, accurate, optimal algorithms for design and NC toolpath generation and to support alternative criteria for surface subdivision based on the second order properties of the shape. Consideration of Figures 4.3 and 4.5 shows another area of use. Users of NURBs are frequently unaware of the implications on the *shape* of the surface from using different orders. Manipulating the same control mesh can give different, unexpected, shapes depending on the order. The ability to accurately visualize second order properties in a reasonable time will enable better inspection and understanding of the effect of order, and potentially knot vector, changes. Furthermore, while NC

verifications frequently simulate the tool path moving over the surface geometry, they do not check that a tool path for a convex region is actually cutting a convex region. The work presented here can be used in implementing that larger *visual process validation*. The viewer can use the understanding gained from exhibiting second order properties to take effective action.

The adaptive isocurve generation algorithm developed in Chapter 5 was extensively and successfully used for 3 axis machining. Adopting it to 4 or 5 axis toolpath generation is straightforward. However, this multi axis toolpath generation raises difficult questions regarding accessibility that must be addressed first. In 3 axis milling the accessibility problem is equivalent to an orthographic projection the hidden surface problem. "What you see is what you can mill". Although not simple, the hidden surface problem is well understood by the computer graphics community. Unfortunately, this does not work in 5 axis any more. The tool axis ("view direction") is not constant and in fact may vary as the tool moves. This area is under current research.

During the presentation of the new layout fabrication method in Chapter 5, it was implicitly assumed that the material thickness is negligible. Unfortunately, this is not always the case and compensating for the distortion that can result should be further investigated. In addition, extending this methodology to support stretching and tearing, should be investigated as well. Not only will that enable dealing with arbitrary surfaces (which cannot be decomposed into piecewise developable surfaces) but this algorithm may then support the ability to handle fabric and other anisotropic materials.

Chapter 6 introduces several small applications that can benefit from symbolic computation. The high order curve approximation using lower order Bézier curves method should be qualitatively and quantitatively compared to currently known approaches. Furthermore, it should be investigated whether a combined approach

can yield an even better result.

The composition tool was also introduced in Chapter 6. In [42] the composition tools developed in this work were used to derive new and exact methods for fillet construction. The full potential of the composition operator combined with the symbolic tools derived in Chapter 2 should be further investigated.

Several shape measures, namely surface steepness, surface speed, and surface twist, where defined and shown to be computable and representable symbolically, in Chapter 6. The advantages these shape measures can provide the designer or the manufacturing engineer should be explored.

Undoubtedly, other applications in computer aided geometric design and in computer graphics can benefit from these tools. These fields matured enough to a level in which robustness is becoming an increasingly important issue. Symbolic computation is one such tool that can help alleviating the numerical problems we are facing today.

APPENDIX

CUSP EXISTENCE PROOF

This appendix shows that a cusp is formed in the offset curve $\mathcal{C}_d(t)$ any time the curve, $C(t)$, has curvature $\kappa(t)$ equal to $\frac{1}{d}$ where d is the offset distance and the mathematical curve normal $N(t)$, coincides with offset normal $N_o(t)$. Conditions for detecting curvature higher than $\frac{1}{d}$ are also derived.

Let $C(t)$ be a regular planar parametric curve that may not be arc length parameterized. Without loss of generality assume $C(t)$ is in the $x-y$ plane. Let $\mathcal{C}_d(t)$ be the offset curve of $C(t)$ by amount d. Let T, N and $\mathcal{T}$, $\mathcal{N}$ be their unit tangents and normals respectively. A nonunit length vector will be tagged with a hat, i.e., $\hat{T}$.

The tangent, T, of the planar curve, C, is equal to

$$\begin{aligned} T(t) &= \frac{\hat{T}(t)}{\|\hat{T}(t)\|} \\ &= \frac{(x'(t), y'(t))}{\sqrt{x'(t)^2 + y'(t)^2}}. \end{aligned} \tag{7.1}$$

From differential geometry theory [48, 63]:

$$\begin{aligned} \kappa(t)B(t) &= \frac{C'(t) \times C''(t)}{\|C'(t)\|^3} \\ &= \frac{(x'(t), y'(t), 0) \times (x''(t), y''(t), 0)}{\sqrt{x'(t)^2 + y'(t)^2}^3} \\ &= \frac{(0, 0, x'(t)y''(t) - y'(t)x''(t))}{\|\hat{T}\|^3} \end{aligned}$$

$$= \frac{(0,0,\Psi)}{\|\hat{T}\|^3}. \tag{7.2}$$

Because $B_o(t)$ has been selected in $+z$ direction (see equations (3.1) and (3.2)), $N_o(t)$ is equal to

$$\begin{aligned} N_o(t) &= B_o(t) \times T(t) \\ &= \frac{(-y'(t), x'(t))}{\sqrt{x'(t)^2 + y'(t)^2}} \\ &= \frac{(-y'(t), x'(t))}{\|\hat{T}\|}. \end{aligned} \tag{7.3}$$

The offset curve $\mathcal{C}_d(t)$ of the planar curve $C(t)$ by amount d is defined as (equation (3.2)):

$$\begin{aligned} \mathcal{C}_d(t) &= C(t) + N_o(t)d \\ &= (x(t), y(t)) + \frac{(-y'(t), x'(t))}{\|\hat{T}\|}d \\ &= \frac{(x(t)\|\hat{T}\| - y'(t)d, y(t)\|\hat{T}\| + x'(t)d)}{\|\hat{T}\|}. \end{aligned} \tag{7.4}$$

The first derivative $\hat{\mathcal{T}}(t)$ of the offset curve $\mathcal{C}_d(t)$ is:

$$\begin{aligned} & \hat{\mathcal{T}}(t) \\ = \; & \mathcal{C}_d'(t) \\ = \; & \left(\frac{(x'(t)\|\hat{T}\| + x(t)\|\hat{T}\|' - y''(t)d)\|\hat{T}\| - (x(t)\|\hat{T}\| - y'(t)d)\|\hat{T}\|'}{\|\hat{T}\|^2}, \right. \\ & \left. \frac{(y'(t)\|\hat{T}\| + y(t)\|\hat{T}\|' + x''(t)d)\|\hat{T}\| - (y(t)\|\hat{T}\| + x'(t)d)\|\hat{T}\|'}{\|\hat{T}\|^2} \right) \\ = \; & \left(\frac{x'(t)\|\hat{T}\|^2 - y''(t)\|\hat{T}\|d + y'(t)\|\hat{T}\|'d, y'(t)\|\hat{T}\|^2 + x''(t)\|\hat{T}\|d - x'(t)\|\hat{T}\|'d}{\|\hat{T}\|^2} \right). \end{aligned} \tag{7.5}$$

We are now ready to inspect the value of $\hat{\mathcal{T}}(t)$ in a case where d is equal to $\frac{1}{\kappa(t)}$. Using equation (7.2):

$$d = \frac{1}{\kappa(t)}$$

$$= \frac{\|\hat{T}\|^3}{\mid \Psi \mid}. \tag{7.6}$$

Substituting d in the x component of $\hat{\mathcal{T}}(t)$ we have:

$$\begin{aligned}
\hat{\mathcal{T}}_x(t) &= \frac{x'(t)\|\hat{T}\|^2 - y''(t)\|\hat{T}\|d + y'(t)\|\hat{T}\|'d}{\|\hat{T}\|^2} \\
&= x'(t) + \frac{-y''(t)\|\hat{T}\|^2 + y'(t)\|\hat{T}\|\|\hat{T}\|'}{\mid \Psi \mid} \\
&= x'(t) + \frac{-y''(t)x'(t)^2 - y''(t)y'(t)^2 + y'(t)x'(t)x''(t) + y'(t)^2y''(t)}{\mid \Psi \mid} \\
&= x'(t) + \frac{-y''(t)x'(t)^2 + y'(t)x'(t)x''(t)}{\mid \Psi \mid} \\
&= x'(t) + \frac{x'(t)(-y''(t)x'(t) + y'(t)x''(t))}{\mid \Psi \mid} \\
&= x'(t) - \frac{x'(t)\Psi}{\mid \Psi \mid} \\
&= \begin{cases} \equiv 0, & \Psi > 0 \\ = 2x'(t) & \Psi < 0 \end{cases}
\end{aligned} \tag{7.7}$$

because

$$\begin{aligned}
\|\hat{T}\|\|\hat{T}\|' &= \sqrt{x'(t)^2 + y'(t)^2}\frac{1}{2\sqrt{x'(t)^2 + y'(t)^2}}(2x'(t)x''(t) + 2y'(t)y''(t)) \\
&= x'(t)x''(t) + y'(t)y''(t)
\end{aligned}$$

and

$$\|\hat{T}\|^2 = x'(t)^2 + y'(t)^2.$$

From equation 7.6, d may be substituted into the y component of $\hat{\mathcal{T}}(t)$, $\hat{\mathcal{T}}_y(t)$, in a similar way for the same result. Therefore, $\hat{\mathcal{T}}(t) \equiv 0$ in this situation or $\mathcal{C}(t)$ has a cusp if $\Psi = x'(t)y''(t) - x''(t)y'(t) > 0$ or the binormal $B(t)$ is positive and coincides with the definition of $B_o(t)$.

Moreover, if $d > \frac{1}{\kappa(t)}$, then the tangent vector $\hat{\mathcal{T}}$ flips direction as can be shown by its dot product with $\hat{T}$. Rewriting equation (7.5) as:

$$\hat{\mathcal{T}}(t) = (x'(t), y'(t)) + \frac{(-y''(t), x''(t))d}{\|\hat{T}\|} + \frac{(y'(t), -x'(t))\|\hat{T}\|'d}{\|\hat{T}\|^2}$$

and substituting it into

$$\begin{aligned}
&\left\langle \hat{\mathcal{T}}(t), \hat{T}(t) \right\rangle \\
&= \left\langle \left((x'(t), y'(t)) + \frac{(-y''(t), x''(t))d}{\|\hat{T}\|} + \frac{(y'(t), -x'(t))\|\hat{T}\|'d}{\|\hat{T}\|^2} \right), (x'(t), y'(t)) \right\rangle \\
&= (x'(t)^2 + y'(t)^2) + \frac{(-y''(t)x'(t) + x''(t)y'(t))d}{\|\hat{T}\|} \\
&= (x'(t)^2 + y'(t)^2) - \frac{\Psi d}{\|\hat{T}\|},
\end{aligned}$$

because the last term of $\hat{\mathcal{T}}(t)$ is perpendicular to $\hat{T}(t)$. Using equation (7.2):

$$\begin{aligned}
&\left\langle \hat{\mathcal{T}}(t), \hat{T}(t) \right\rangle \\
&= (x'(t)^2 + y'(t)^2) - \frac{\Psi d}{\|\hat{T}\|} \\
&= \begin{cases} (x'(t)^2 + y'(t)^2) - \frac{\kappa(t)(x'(t)^2+y'(t)^2)^{\frac{3}{2}}d}{\sqrt{x'(t)^2+y'(t)^2}} = (x'(t)^2 + y'(t)^2)(1 - \kappa(t)d), & \Psi > 0 \\ (x'(t)^2 + y'(t)^2) + \frac{\kappa(t)(x'(t)^2+y'(t)^2)^{\frac{3}{2}}d}{\sqrt{x'(t)^2+y'(t)^2}} = (x'(t)^2 + y'(t)^2)(1 + \kappa(t)d), & \Psi < 0. \end{cases}
\end{aligned}$$

Because $C(t)$ is a regular curve, $T(t)$ is never zero and $(x'(t)^2 + y'(t)^2)$ is positive everywhere. Therefore, for cases where the mathematical normal, $N(t)$, coincides with the offset normal, $N_o(t)$, or $\Psi > 0$, we get:

$$\begin{aligned}
sign(\left\langle \hat{\mathcal{T}}(t), \hat{T}(t) \right\rangle) &= sign((x'(t)^2 + y'(t)^2)(1 - \kappa(t)d)) \\
&= sign(1 - \kappa(t)d). \qquad (7.8)
\end{aligned}$$

Now for small $\kappa(t)$ or a relatively straight curve, $(1 - \kappa(t)d)$ is positive. When $\kappa(t)$ reaches $\frac{1}{d}$, the expression becomes zero, or $\hat{\mathcal{T}}(t) = 0$ because $\hat{T}(t)$ is never zero. If $\kappa(t)$ is larger than $\frac{1}{d}$, the expression is negative, that is $\hat{\mathcal{T}}(t)$ has flipped its direction.

If $\Psi < 0$ the expression is never zero because both d and $\kappa(t)$ are positive. This is not a surprising result because such offset only *increases* the radius of the osculating circle and hence can never make it vanish.

REFERENCES

[1] A. V. Aho, J. E. Hopcroft and J. D. Ullman. Data Structures and Algorithms. Addison-Wesley, June 1983.

[2] S. Aomura and T. Uehara. Self-Intersection of an Offset Surface. Computer Aided Design, vol. 22, no. 7, pp 417-422, september 1990

[3] RE Barnhill, G. Farin, L. Fayard and H. Hagen. Twists, Curvatures and Surface Interrogation. Computer Aided Design, vol. 20, no. 6, pp 341-346, July/August 1988.

[4] J. M. Beck, R. T. Farouki, and J. K. Hinds. Surface Analysis Methods. IEEE Computer Graphics and Applications, vol. 6, no. 12, pp 18-36, December 1986.

[5] C. Bennis, J. M. Vezien, and G. Iglesias. Piecewise Surface Flattening for Non-Distorted Texture Mapping. SIGGRAPH 1991.

[6] J. Bertrand. La Theorie Des Courbes a Double Courbure. Journal de Mathematique Pures et Appliquees, 15 (1850), 332-350.

[7] W. Boehm. Inserting New Knots into B-spline Curves. Computer Aided Design, vol. 12, no. 4, pp 199-201, July 1980.

[8] J. E. Bobrow. NC Machine Toll Path Generation From CSG Part Representations. Computer Aided Design, vol. 17, no. 2, pp 69-76, March 1985.

[9] M. S. Casale. Free-Form Solid Modeling with Trimmed Surface Patches. IEEE Computer Graphics and Applications, vol. 7, no. 1, pp 33-43, January 1987.

[10] B. K. Choi and C. S. Jun. Ball-End Cutter Interference Avoidance in NC Machining of Sculptured Surfaces. Computer Aided Design, vol. 21, no. 6, pp 371-378, July/August 1989.

[11] B. K. Choi and S. Y. Ju. Constant-Radius Blending in Surface Modelling. Computer Aided Design, vol. 21, no. 4, pp 213-220, May 1989.

[12] J. J. Chou. Numerical Control Milling Machine Toolpath Generation for Regions Bounded by Free Form Curves and Surfaces. Ph.D. Thesis, University of Utah, Computer Science Department, June 1989.

[13] B. Cobb. Design of Sculptured Surfaces Using The B-spline Representation. Ph.D. Thesis, University of Utah, Computer Science Department, June 1984.

[14] E. Cohen. Some Mathematical Tools for a Modeler's Workbench IEEE Computer Graphics and Applications, vol. 3, pp 63-66, October 1983.

[15] E. Cohen, T. Lyche, and R. Riesenfeld. Discrete B-splines and Subdivision Techniques in Computer Aided Geometric Design and Computer Graphics. Computer Graphics and Image Processing, 14, 87-111 (1980).

[16] E. Cohen, T. Lyche, and L. Schumaker. Degree Raising for Splines. Journal of Approximation Theory, vol 46, Feb. 1986.

[17] E. Cohen, T. Lyche, and L. Schumaker. Algorithms for Degree Raising for Splines. ACM Transactions on Graphics, vol 4, No 3, pp 171-181, July 1986.

[18] S. Coquillart. Computing Offset of Bspline Curves. Computer Aided Design, vol. 19, no. 6, pp 305-309, July/August 1987.

[19] S. Coquillart. Animated Free-Form Deformation: An Interactive Animation Technique. SIGGRAPH 1991, pp 23-26.

[20] J. C. Dill. An Application of Color Graphics to the Display of Surface Curvature. SIGGRAPH 1981, pp 153-161.

[21] M. P. DoCarmo. Differential Geometry of Curves and Surfaces. Prentice-Hall 1976.

[22] G. Elber and E. Cohen. Hidden Curve Removal for Untrimmed and Trimmed NURB Surfaces. University of Utah, Computer Science Dept., Technical Report UUCS-89-019, May 1989.

[23] G. Elber and E. Cohen. Hidden Curve Removal for Free Form Surfaces. SIGGRAPH 1990, pp 95-104.

[24] G. Elber and E. Cohen. Second Order Surface Analysis Using Hybrid Symbolic and Numeric Operators. To appear in Transaction on Graphics.

[25] G. Elber and E. Cohen. Error Bounded Variable Distance Offset Operator for Free Form Curves and Surfaces. International Journal of Computational Geometry and Applications, vol. 1., no. 1, pp 67-78, March 1991.

[26] G. Elber and E. Cohen. Adaptive Isocurves Based Rendering for Freeform Surfaces. Submitted for Publication.

[27] G. Farin. Curves and Surfaces for Computer Aided Geometric Design Academic Press, Inc. Second Edition 1990.

[28] R. T. Farouki and C. A. Neff. Some Analytic and Algebraic Properties of Plane Offset Curves. Research Report 14364 (#64329) 1/25/89, IBM Research Division.

[29] R. T. Farouki. The Approximation of Non-Degenerate Offset Surfaces Computer Aided Geometric Design, vol. 3, no. 1, pp 15-43, May 1986.

[30] R. T. Farouki and V. T. Rajan. Algorithms For Polynomials In Bernstein Form. Computer Aided Geometric Design, vol. 5, pp 1-26, 1988.

[31] R. T. Farouki and V. T. Rajan. On The Numerical Condition of Polynomial in Bernstein Form. Computer Aided Geometric Design 4, pp 191-216, 1987.

[32] I. D. Faux and M. J. Pratt. Computational Geometry for Design and Manufacturing. John Wiley & Sons, 1979.

[33] A. R. Forrest. On the Rendering of Surfaces. SIGGRAPH 1979, pp 253-259.

[34] C. M. Hoffmann. Geometric & Solid Modeling, an Introduction. Morgan Kaufmann Publisher, Inc..

[35] J. Hoschek. Offset Curves in the Plane. Computer Aided Design, vol. 17, no. 2, pp 77-82, March 1985.

[36] J. Hoschek. Spline Approximation of Offset Curves. Computer Aided Geometric Design 5, pp 33-40, 1988.

[37] J. Hoschek and N. Wissel. Optimal Approximate Conversion of Spline Curves and Spline Approximation of Offset Curves. Computer Aided Design, vol. 20, no. 8, pp 475-483, October 1988.

[38] J. Hoschek. Approximate Conversion of Spline Curves. Computer Aided Geometric Design 4, pp 59-66, 1987.

[39] J. Hoschek, F. J. Schneider, and P. Wassum. Optmial Approximate Conversion of Spline Surfaces. Computer Aided Geometric Design 6, pp 293-306, 1989.

[40] R. B. Jerard, J. M. Angleton and R. L. Drysdale. Sculptured Surface Tool path Generation with Global Interference Checking. Design Productivity Conference, Feb. 6-8, 1991, Honolulu, Hawaii.

[41] J. T. Kajiya. Ray Tracing Parametric Patches. SIGGRAPH 1982, pp 245-256.

[42] K. Kim. Blending Parametric Surfaces. M.S. Thesis, University of Utah, Computer Science Department, August 1992.

[43] J. Lane and R. Riesenfeld. A Theoretical Development for the Computer Generation and Display of Piecewise Polynomial Surfaces. IEEE Transaction on pattern analysis and machine intelligence, vol. PAMI-2, no. 1, January 1980.

[44] J. Lane and R. Riesenfeld. Bounds on a Polynomial BIT 21, pp 112-117, 1981.

[45] R. T. Lee and D. A. Fredericks. Intersection of Parametric Surfaces and a Plane. IEEE Computer Graphics and Applications, vol. 4, no. 8, pp 48-51, August 1984.

[46] G. C. Loney and T. M. Ozsoy. NC Machining of Free Form Surfaces. Computer Aided Design, vol. 19, no. 2, pp 85-90, March 1987.

[47] T. McCollough. Support for Trimmed Surfaces. M.S. Thesis, University of Utah, Computer Science Department, 1988.

[48] Millman and Parker. Elements of Differential Geometry. Prentice Hill Inc., 1977.

[49] K. Morken. Some Identities for Products and Degree Raising of Splines. To appear in the Journal of Constructive Approximation.

[50] B. Pham. Offset Approximation of Uniform B-splines. Computer Aided Design, vol. 20, no. 8, pp 471-474, October 1988.

[51] B. Pham. Offset Curves and Surfaces: a Brief Survey. Computer Aided Design, vol. 24, no. 4, pp 223-229, April 1992.

[52] H Persson. NC Machining of Arbitrarily Shaped Pockets. Computer Aided Design, vol. 10, no. 3, pp 169-174, May 1978.

[53] PostScript Language Reference Manual. Adobe Systems Incorporated. Addison Wesley Publishing Company, Inc., 1987.

[54] A. A. G. Requicha. Toward a Theory of Geometric Tolerancing. International Journal of Robotics Research, vol. 2, no. 4, pp 45-49, Winter 1983.

[55] R. Riesenfeld. Applications of B-spline Approximation to Geometric Problems of Computer-Aided Design. Ph.D. Thesis. University of Utah, Technical Report UTEC-CSc-73-126, March 1973.

[56] D. F. Rogers and J. A. Adams. Mathematical Elements for Computer Graphics, second edition. McGraw-Hill Publishing Company, 1990.

[57] J. R. Rossignac and A. A. G. Requicha. Constant Radius blending In Solid Modeling. Computers in Mechanical Engineering, pp 65-73, July 1984.

[58] F. Salkowski. Zur Transformation von Raumkurven. Mathematische Annalen, 66 (1909), 517-557.

[59] S. G. Satterfield and D. F. Rogers. A Procedure for Generating Contour Lines From a Bspline Surface. IEEE Computer Graphics and Applications, vol. 5, no. 4, pp 71-75, April 1985.

[60] T. W. Sederberg and S. Parry. Comparison of Three Curve Intersection Algorithms. Computer Aided Design, vol. 18, no. 1, pp 58-63, January/February 1986.

[61] T. W. Sederberg and S. Parry. Free-Form Deformation of Solid Geometric Models. SIGGRAPH 1986, pp 151-160.

[62] T. W. Sederberg and A. K. Zundel. Scan Line Display of Algebraic Surfaces. SIGGRAPH 1989, pp 147-156.

[63] J. J. Stoker. Differential Geometry. Wiley-Interscience 1969.

[64] S. W. Thomas. Scanline Rendering for 3-Axis NC Toolpath Generation, Simulation and Verification. Dept. of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI 48109-2122, Technical Report CSE-TR-43-90, January 1990.

[65] A. Voss. Uber Kurvenpaare in Raume. Sitzungsberichte, Akadamie der Wissenschaften zu Munchen. 39 (1909), 106.

[66] D. J. Walton abd D. S. Meek. Curvature Bounds For Planar B-spline Curve Segments. Computer Aided Design, vol. 20, no. 3, pp 146-150, April 1988.

[67] D. Zhang and A. Bowyer. CSG Set-Theoretical Solid Modelling and NC Machining of Blend Surfaces. The Second Computation Geometry Conference, ACM 1986.