This is to certify that the

dissertation entitled

THE DESIGN OF RATIONAL B-SPLINE ALGORITHMS
FOR
INTERACTIVE COLOR SHADING OF SURFACES

presented by

Mark Norman Pickelmann

has been accepted towards fulfillment
of the requirements for

Doctor of Philosophy degree in Mechanical Engineering

[illegible]
Major professor

Eric D. Goodman

Date 4/18/85

THE DESIGN OF RATIONAL B-SPLINE ALGORITHMS FOR INTERACTIVE COLOR SHADING OF SURFACES

By

Mark Norman Pickelmann

A DISSERTATION

Submitted to

Michigan State University

in partial fulfillment of the requirements

for the degree of

DOCTOR OF PHILOSOPHY

Department of Mechanical Engineering

1985

XI 353 ... [illegible]

THE DESIGN OF RATIONAL B-SPLINE ALGORITHMS FOR INTERACTIVE COLOR SHADING OF SURFACES

Submitted by:

Mark N. Pickelmann　　　　4-18-85

Mark N. Pickelmann　　　　Date
Ph.D. Candidate

Approved by:

[illegible]　　　　4/18/85

Erik D. Goodman　　　　Date
Professor
Dissertation Advisor

[illegible]　　　　4-18-85

John R. Lloyd　　　　Date
Chairperson
Department of Mechanical Engineering

ABSTRACT

THE DESIGN OF RATIONAL B-SPLINE ALGORITHMS FOR INTERACTIVE COLOR SHADING OF SURFACES

By

Mark Norman Pickelmann

This dissertation presents several algorithms developed for use with CAD/CAM systems. The new algorithms allow for more efficient evaluations of the entire range of rational B-spline curves and surfaces. A class of rational B-spline called Enhanced Uniform is defined. The algorithms developed include an algorithm which is used as a preprocessor to transform the definition of a nonuniform rational B-spline surface to an equivalent surface based on the Enhanced Uniform Rational B-spline. The second algorithm developed is used for evaluation of enhanced uniform B-splines and their derivatives. Numerical considerations in the implementation of these algorithms are discussed. Examples of the use of these algorithms to generate color images in a surface assessment program are included.

To my loving wife Kristi.

For my sons Daniel and Kevin

and any others who come along.

AKNOWLEDGEMENTS

The author would like to thank my committee of Dr. Erik D. Goodman, Dr. James E. Bernard, Dr. Ronald C. Rosenberg, and Dr. Jake M. Plotkin for all their help getting me through.

A special thanks goes to Dr. James E. Bernard, for his help, friendship, and advice. Although he left during the course of this work he is responsible for starting me in the Ph.D. program and he was there to get me through qualifers when I needed him the most.

The author would like to thank the General Dynamics Corporation for their funding of the COLORSCOPE project.

I would like to thank my family which includes mom and dad, my brother, my mother and father in law, and my brothers and sisters in law for all their encouragement during the course of my studies.

Finally, I would like to thank my friends at MSU, for the classes we took together and for hanging out on friday afternoons, but most of all I would just like to thank them for being my friends.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER I

INTRODUCTION

1.0 INTRODUCTION

The goal of this dissertation is to present several algorithms developed for use with CAD/CAM systems. The new algorithms allow for more efficient evaluation of the entire range of rational B-spline curves and surfaces. These algorithms allow a considerable increase in the speed of response of interactive CAD/CAM programs which utilize them.

The use of mathematical models to represent sculptured surfaces has become not only a design tool but also a manufacturing tool. Before a model can be used for production or analysis it must be checked for errors. Common errors which occur in these types of models are misplaced corner points, missing patches, slope discontinuities between patches, and gaps between patches.

1.1 ERROR CHECKING

Checking for errors in models can be a very costly process. Two methods in common use are two-dimensional line drawings and proofing runs of trial parts on a numerically controlled machine. In the case of two-dimensional line drawings small errors such as slope discontinuities are very difficult to detect due to the complexity of the drawings.

To check the validity of tool paths for numerical control machining, parts are often milled out of a substitute material to check the model. This is costly because it ties up a mill and an operator, and it increases material costs.

Another method for checking the model is to use the computer to generate an accurate shaded image of the model.

This method is becoming popular because it saves much of the expense of checking the model [1].

Currently many color shading packages are available for a wide range of surface representations [2]. However, most of these packages sacrifice some of the surface information to produce smoothly shaded pictures. This is done by tiling the surface and utilizing some type of filtering to hide the effects of tiling. These altered surfaces do not represent the true surface definitions and hide many of the errors.

Researchers at the A. H. Case Center for Computer-Aided Design have developed a surface assessment package called MSU COLORSCOPE to produce accurate shaded images. The package allows shading based on various light models as well as surface curvature properties. The MSU COLORSCOPE approach is to calculate and shade the surface at the pixel level using specialized scan line techniques [3] [4] [5]. This creates a very accurate picture of the surface. These pictures can then be used for rapid detection of errors or flaws in the model.

1.2 SURFACE MODELS

Rapid growth in the computer-aided design / computer-aided manufacturing (CAD/CAM) field has resulted in the lack of a standard method for defining a three-dimensional surface. In recent years the most popular way has been the bi-cubic patch. But sophisticated applications demand more flexibility than the bi-cubic patch can offer. In many situations the rational B-spline surface is being used [6]. The rational B-spline surface allows a great deal of flexibility and the capability of representing many popular types of surface descriptions in an exactly equivalent form. While rational B-spline surfaces cannot precisely represent all mathematical surfaces, they are a very powerful form and are an emerging de facto standard in today's CAD/CAM systems.

MSU Colorscope is intended to be a general-purpose surface assessment package. However, prior to this work, it used bi-cubic patch definitions based on the Coons [7] blending functions. This has severely limited applications of the package in today's CAD/CAM community. This dissertation develops both a method for the efficient incorporation of the rational B-spline into a general purpose surface assessment package, and the mathematical forms and algorithms to allow conversion among a variety of surface types.

1.2.1 The Rational B-spline Surface

B-splines have only recently been widely used in engineering applications. Perhaps this has been due to the complexity of the basis functions and the computing burden associated with calculating them each time a surface point is to be evaluated. It is well known that the time to calculate the basis functions can be reduced if certain restrictions are placed on the class of B-spline [8]. We shall show that it is possible to achieve similar time reductions without any constraints placed on the class of B-splines used.

An important component of this work is to shade rational B-spline surfaces using scan line methods in an interactive mode. The definition of interactive varies from user to user and task to task; however, for the present purpose, it is taken to mean that one should speed up the shading process as much as possible without sacrificing any of the surface information. The techniques presented here will take advantage of a modern thirty-two-bit computing processor using virtual memory.

1.3 OVERVIEW OF THE DISSERTATION

Chapter Two presents a review of the rational B-spline surface. Chapter Three defines a class of B-spline called _enhanced_ _uniform_ and develops an algorithm, called the Converted to Uniform Rational B-Splines (CURBS) algorithm, for calculating with enhanced uniform rational B-splines. A comparison of the commonly used Cox-deBoor algorithm to the CURBS algorithm using the enhanced uniform B-spline is made in Chapter Four. Chapter Five develops an algorithm to transform any rational B-spline into an exactly equivalent enhanced uniform B-spline. Discussed in Chapter Six are some of the numerical problems encountered in the implementation of the B-spline algorithms. Conclusions are drawn and examples given in Chapter Seven.

CHAPTER II

B-SPLINE CURVES AND SURFACES

2.0 B-SPLINE REVIEW

This chapter presents a review of the rational B-spline surface patch. The review starts with a the B-spline space curve and extends to the rational B-spline space curve. Next the B-spline surface patch is defined and extended to the rational B-spline surface patch. Finally, various classifications of B-splines are discussed.

2.1 B-SPLINE SPACE CURVE

The B-spline approximation was first presented by I. J. Schoenberg [9] in 1946 for statistical data smoothing. More recent work has been done by deBoor [10], Cox [11], Riesenfeld [12], and Versprille [8]. Cox and deBoor each developed a numerically stable algorithm to evaluate the B-spline points and derivatives. Riesenfeld used the Cox-deBoor algorithm to apply the B-spline to geometric problems in computer-aided design. Versprille used the rational B-spline, but restricted the knot vector so as to produce only a small subset of the B-spline family.

2.1.1 Basis Functions

Let P be the Cartesian position along a curve as a function of the parametric variable t. A curve generated using the B-spline basis is given by :

$$P(t) = \sum_{i=0}^{n} CP_i \cdot N_{i,k}(t) \qquad 2.1.1$$

where

CP are the n+1 control points

k is the order of the blending functions

n is the number of control points minus one

N are the n+1 blending functions

The order of the B-spline curve is the degree plus one. If the number of control points exceeds the order of the curve the B-spline will have more than one segment. The segments which make up a B-spline will be called <u>subsegments</u>. The above blending functions are defined by the recursion formulas [13] :

$$\begin{aligned} N_{i,1}(t) &= 1 \quad \text{if } x_i \leq t < x_{i+1} \\ &= 0 \quad \text{otherwise} \end{aligned} \qquad 2.1.2$$

$$\begin{aligned} N_{i,k}(t) = {} & (t-x_i)\, N_{i,k-1}(t) \,/\, (x_{i+k-1} - x_i) \\ & + (x_{i+k}-t)\, N_{i+1,k-1}(t) \,/\, (x_{i+k} - x_{i+1}) \end{aligned} \qquad 2.1.3$$

where

x_i are the entries of the knot vector

The knot vector specifies how the parametric value is distributed along the B-spline. Subsegments start and end when the parametric value crosses the knot values.

For example if the knot vector is taken to be :

X_1	=	[	0	0	0	0	1	2	3	4	5	6	7	8	9	9	9	9]
i	=		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

The values of the blending functions for i=6 with k=1, k=2, k=3, and k=4 are shown in Figure 2.1.1 as a function of the parametric variable t.

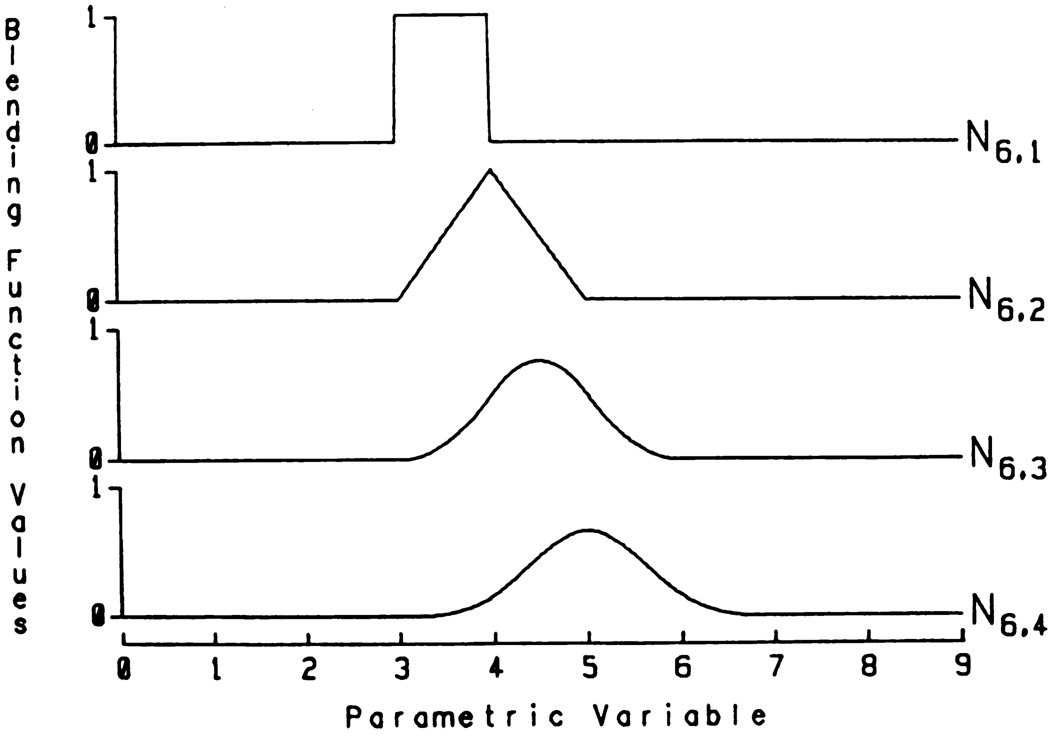

Figure 2.1.1 : Blending Functions of Order Up to Four

From Figure 2.1.1 we can see that the the blending function is always zero for a value of t from zero to three. As k increases, the blending functions are non-zero for a larger and larger value of t. But the maximum value of the blending functions is decreasing as k is increased.

In Figure 2.1.2 the values for all of the blending functions for order k=4 have been plotted. The value plotted for each $N_{i,4}$ would be used with with the corresponding CP_i to determine values along the B-spline curve. At any value of t there are at most four non-zero blending functions. As t crosses a knot, the blending function $N_{j,4}$ goes to zero and the blending function $N_{j+4,4}$ becomes non-zero. It is at this point that one subsegment ends and another starts.

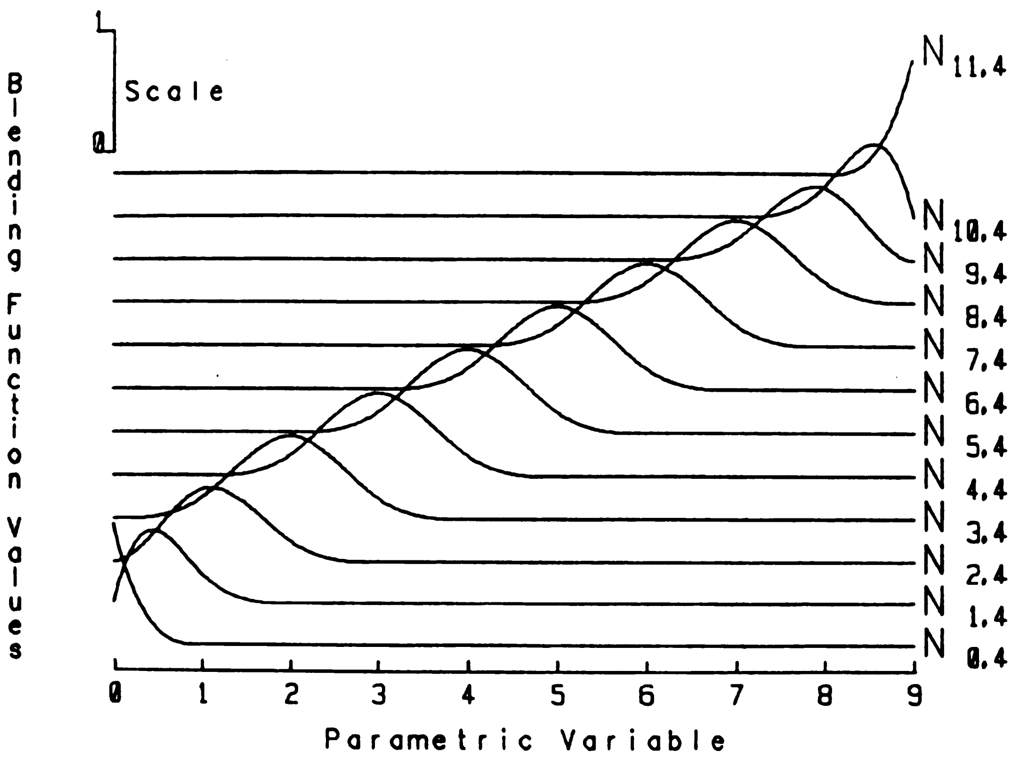

Figure 2.1.2 : Fourth Order Blending Functions

Figure 2.1.3 shows the non-zero blending functions for $i=6$ for orders $k=1$, $k=2$, $k=3$, and $k=4$ as a function of the parametric variable t. Here t varies from three to four. For order $k=4$, we see that $N_{3,4}$ is non-zero at $t=3$ and goes to zero at $t=4$, while $N_{6,4}$ starts at zero and ends up non-zero.

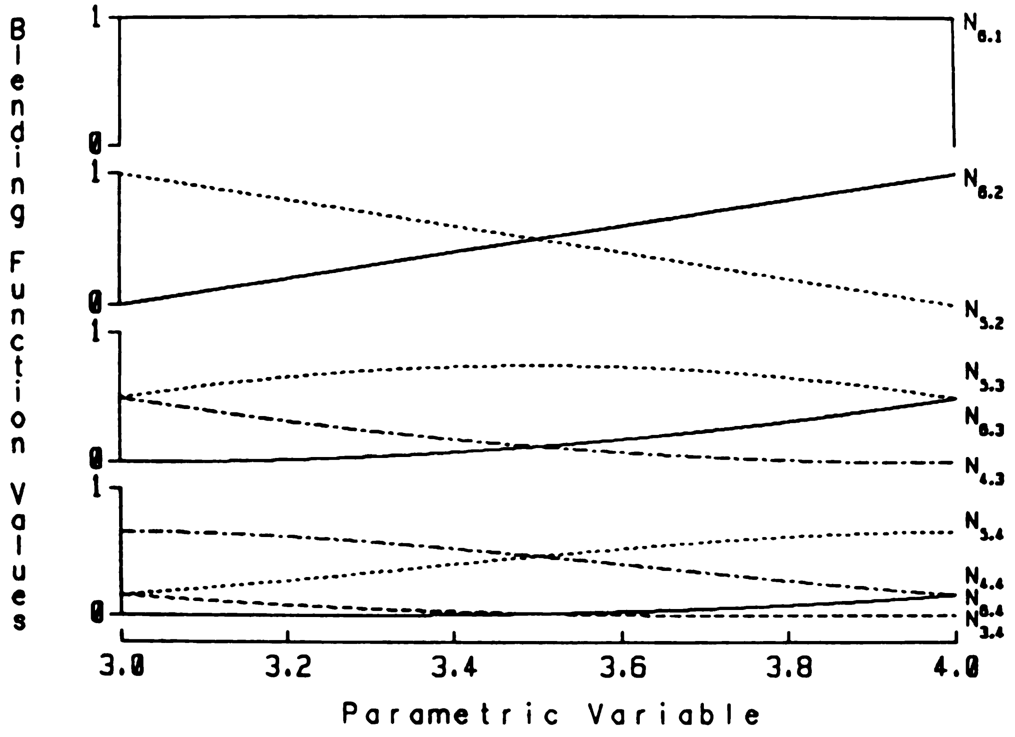

Figure 2.1.3 : Non-Zero Blending Functions for i=6

The effect on the blending functions of repeating interior knots is shown in Figures 2.1.4 and 2.1.5.

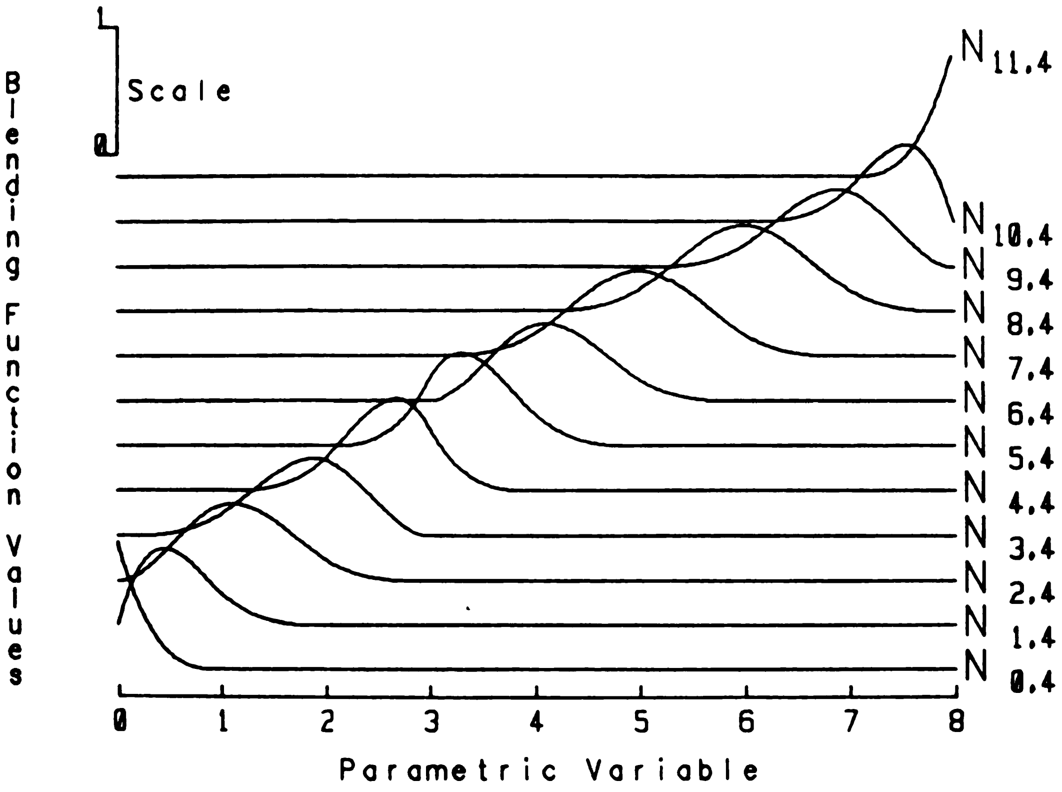

Figure 2.1.4 : Fourth Order Blending Functions Knot Seven Equal to Knot Six

The knot vector used for Figure 2.1.4 is :

$$X_2 = [\ 0\ 0\ 0\ 0\ 1\ 2\ 3\ 3\ 4\ 5\ 6\ 7\ 8\ 8\ 8\ 8\ ]$$
$$i = \ 0\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10\ 11\ 12\ 13\ 14\ 15$$

The knot vector used for Figure 2.1.5 is :

$$X_3 = [\ 0\ 0\ 0\ 0\ 1\ 2\ 3\ 3\ 3\ 4\ 5\ 6\ 7\ 7\ 7\ 7\ ]$$
$$i = \ 0\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10\ 11\ 12\ 13\ 14\ 15$$

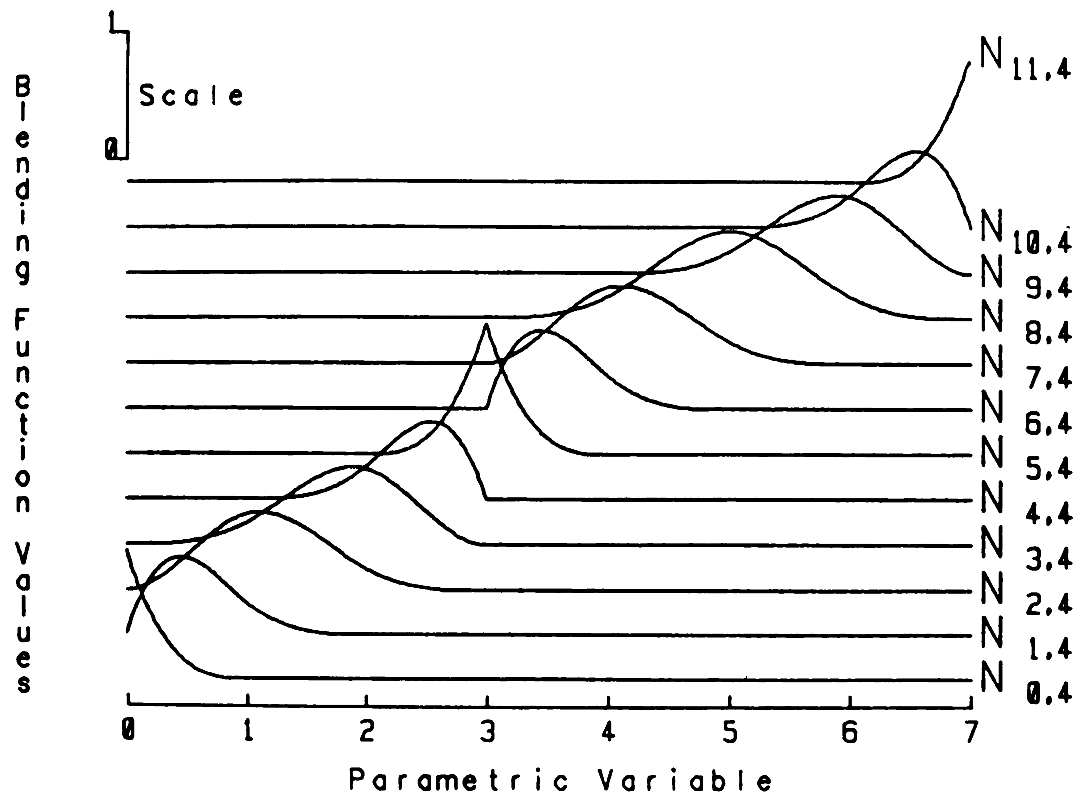

Figure 2.1.5 : Fourth Order Blending Functions
Knots Six, Seven, and Eight Equal

The non-zero blending functions of Figures 2.1.2, 2.1.4, and 2.1.5 for t between two and four have been plotted in Figure 2.1.6.

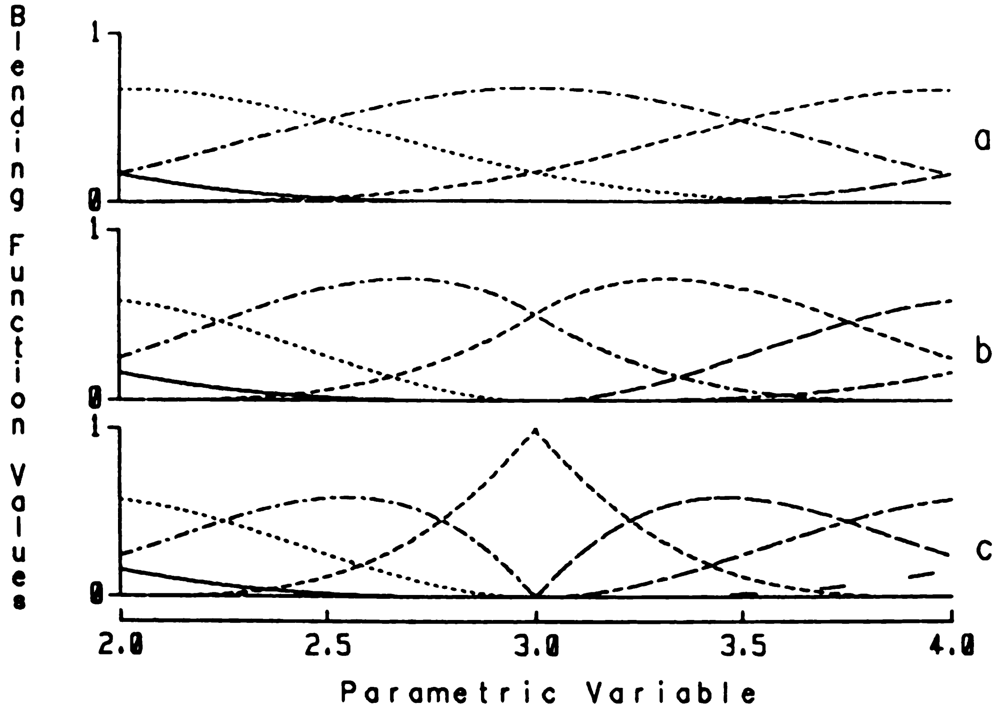

Figure 2.1.6 : Fourth Order Blending Functions
a) Knot Vector X_1
b) Knot Vector X_2
c) Knot Vector X_3

With no repeated knots there are five non-zero blending functions in this range. With one repeated interior knot there are six non-zero blending functions. At t=3 two of the blending functions go to zero as opposed to one with no repeated interior knots. With the interior knot repeated two times, three blending functions go to zero at t=3.

Each control point is associated with an N_i and each N_i is associated with a knot x_i in the knot vector. However, it is difficult to associate a control point with a knot in the knot vector. As shown later, only a number of the control points equal to the order of the curve affects a given subsegment. But the number of knots which affect the blending functions for that subsegment is greater than the number of control points. A way to think of this is that a given control point affects more than one subsegment. Because of continuity constraints between subsegments, the control point must be able to look ahead and look back to influence the other subsegments. The control point uses the knot vector to do the looking. Repeated knots in the knot vector can then be viewed as walls to limit the number of subsegments the control point can see forward or backward.

The only restriction placed on the elements of the knot vector is that $x_i \geq x_{i-1}$. For a given set of control points, an infinite number of curves could be generated, depending on the values assigned to each x_i. It is this fact which causes the need to evaluate equations 2.1.2 and 2.1.3 or some version thereof, each time a point is to be found. However, not all of the $N_{i,k}$ are non-zero for a given subsegment. In fact only a number equal to the order (k) of the curve have non-zero values. These $N_{i,k}$ are the only ones that need be evaluated for that subsegment. The k non-zero $N_{i,k}$ are the same for all t such that $x_j \leq t < x_{j+1}$; points on this part of the curve are said to form the j^{th} subsegment of the curve.

It is possible to derive a formula for the $N_{i,k}$ for a given subsegment without an explicit value of t, provided the knot vector is given. These formulas are valid for any t in the range $x_j \leq t < x_{j+1}$. However, there are an infinite number of possible knot vectors. These formulas would have to be derived each time a subsegment was to be evaluated for a new knot vector.

It has been shown in [10] and [11] that the use of equations 2.1.2 and 2.1.3 directly can cause numerical instabilities when the entries of the knot vector become irregularly spaced or the order of the curve becomes large. Thus Cox [11] and deBoor [10] independently developed the same algorithm which is numerically stable. But this algorithm returns values of the $N_{i,k}$ and/or their derivatives only for particular values of t, and must be recalculated entirely for each new t. We shall return to the stability of equations 2.1.2 and 2.1.3 later in the discussion of the restrictions that can be placed on the knot vector.

2.1.2 Rational B-spline Space Curves

The extension to rational B-spline curves is handled by assigning a positive non-zero weight to each of the control points. The weights are used to form a homogeneous coordinate system. Each control point $CP_i(x,y,z,1)$ is assigned a weight w_i. For calculations with this control point, it is multiplied by w_i to form a new control point $CP'_i(wx,wy,wz,w)$. The CP' points are used in equation 2.1.1 to calculate a homogeneous point $P'(wx,wy,wz,w)$. The three-space point $P(x,y,z,1)$ is calculated by dividing each coordinate of P' by the calculated value of w.

$$\begin{aligned} X(t) &= WX(t)/W(t) \\ Y(t) &= WY(t)/W(t) \\ Z(t) &= WZ(t)/W(t) \end{aligned} \qquad 2.1.4$$

It is hard to think of a curve in four-space, but one way to visualize this is to use a curve in two dimensions. The following three figures show an example of a two-dimensional rational B-spline curve.

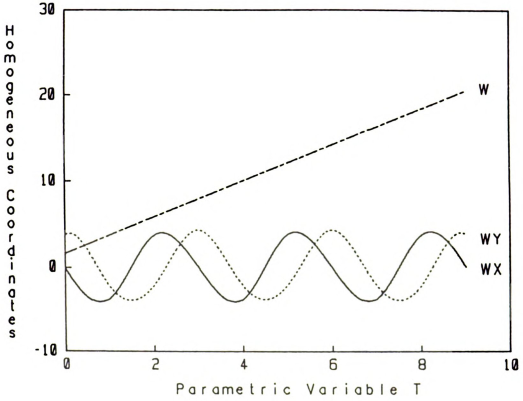

Figure 2.1.7 : Homogeneous Coordinates vs Parametric Variable

The data for Figures 2.1.7, 2.1.8 and 2.1.9 are given in Appendix A. Figure 2.1.7 shows the values of WX, WY and W as a function of the parametric value t. The three ordinates of Figure 2.1.7 are plotted in three-space in Figure 2.1.8. Figure 2.1.9 is a plot of the resulting curve in X-Y space; also plotted is the curve which would result if all the weights were equal.

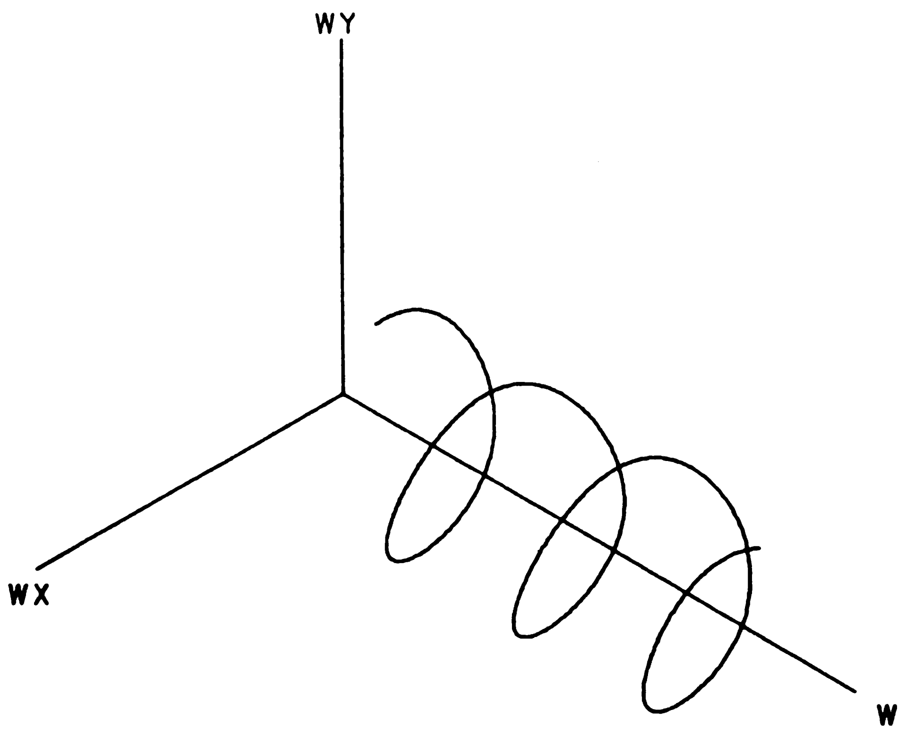

Figure 2.1.8 : Three-Space Plot of Homogeneous Coordinates

A B-spline is then also a rational B-spline with all the weights set equal to one. A <u>strictly rational B-spline</u> will have one or more weights not equal to one or $CP'_i \neq CP_i$.

Assigning unequal weights to the control points introduces an additional degree of freedom. This degree of freedom can be used to produce curves or shapes that are not attainable with non-rational curves, such as conics or circular arcs.

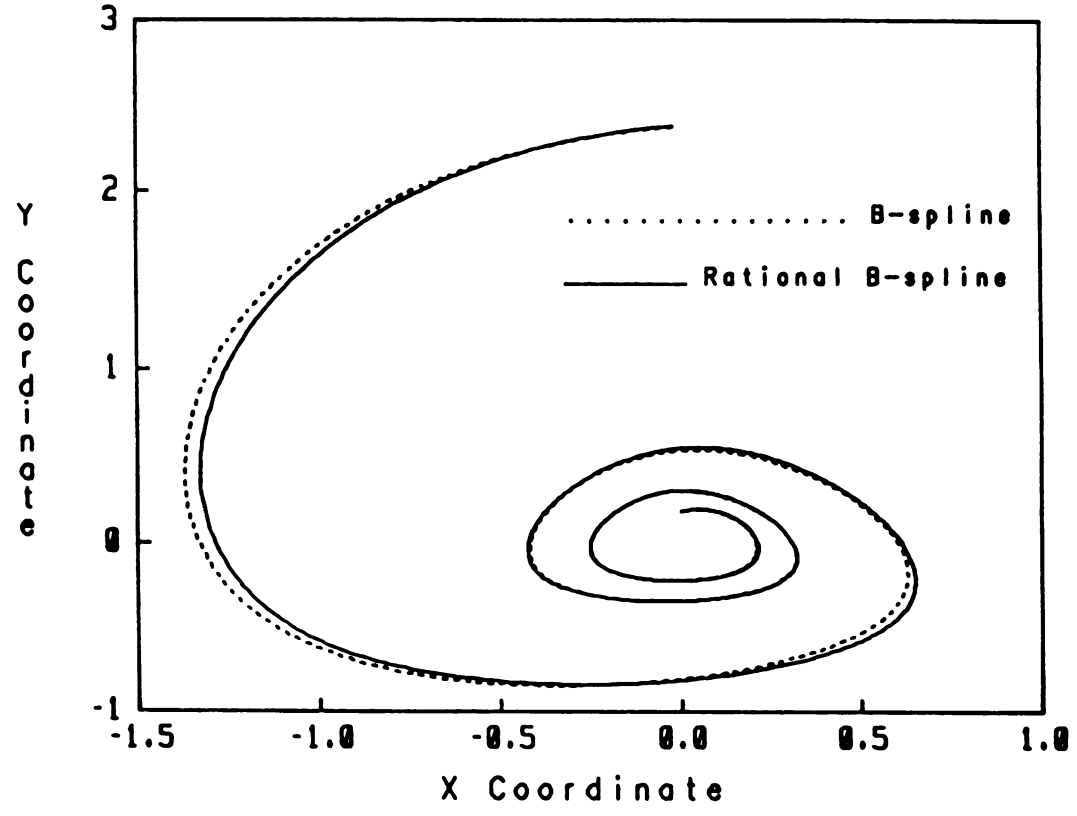

Figure 2.1.9 : Plot of X-Y Curve

2.2 B-SPLINE SURFACE

The extension from the space curve to the surface patch is done by adding a second parametric coordinate and associated knot vector. Points on the surface are given by :

$$P(s,t) = \sum_{i=0}^{n} \sum_{j=0}^{m} CP_{i,j} \cdot N_{i,k}(t) \cdot M_{j,h}(s) \qquad 2.2.1$$

Where

CP	are the $(n+1)\cdot(m+1)$ control points
k	is the order of the curve in the t direction
h	is the order of the curve in the s direction
n	is the number of control points minus one in the t direction
m	is the number of control points minus one in the s direction
N	are the n+1 blending functions of order k
M	are the m+1 blending functions of order h

The control points are now associated with a row and column. The number of rows does not have to equal the number of columns, nor does the order in the s direction have to equal the order in the t direction. A basis function is associated with each row and a different function is associated with each column. As with the B-spline curve, only a given number of these functions are non-zero. Analogous to the subsegment of the curve, the resulting submatrix of the control point matrix defines a <u>subpatch</u>. The entire B-spline patch will be referred to as a <u>mother patch</u>.

2.2.1 Rational B-spline Surfaces

More complex surfaces may be constructed if a weight is also assigned to each control point of the B-spline surface. To determine points on the surface, equation 2.2.1 is used in the homogeneous coordinate system as described above. The resulting surface point in three-space is then found by equation 2.1.4.

2.3 KNOT VECTOR RESTRICTIONS

The only restriction placed on the knot vector is that each entry must be a real number equal to or greater than the one immediately preceding it. This type of knot vector is referred to as a <u>nonuniform knot vector</u> and produces B-splines called Nonuniform Rational B-splines (NURBS). Knot vectors are often <u>extended</u> by adding knots to the beginning and end equal to the first and last entries of the vector respectively. Repeating the end knots causes the curve to pass through the end control points. In this case the end control points are called a <u>geometric knot</u>.

2.3.1 Uniform Knot Vectors

Placing a restriction on the knot vector that the entries are only allowed to take on successive integer values ($x_{i+1} = x_i + 1$), produces a knot vector known as a <u>uniform knot vector</u>. This knot vector produces what is called a <u>uniform basis</u> [12], which leads to Uniform Rational B-splines (URBS). This knot vector can also be extended to form an <u>extended uniform knot vector</u>.

The use of uniform or extended uniform knot vectors has some side effects. In particular, the parameterization of the curve is <u>not</u> uniform with respect to arc length along the curve, and local refinement is difficult. However, a uniform knot vector makes evaluation of the basis functions easier because equations 2.1.2 and 2.1.3 can be used directly. Thus an equation for each subsegment can be derived without a specific value of the parametric variable.

The nonuniform parameterization which often accompanies a uniform knot vector is undesirable in some applications such as numerically controlled machining, in which it is desirable to have a constant step size of the parametric variable produce a constant step size along the curve. However, nonuniform parameterization does not affect the evaluation of surface properties for the purposes of shading.

2.3.2 Enhanced Uniform Knot Vectors

To restrict the knot vector to be uniform or extended uniform for the purposes of a CAD/CAM modeler would over-restrict the class of B-splines that could be used. For example, this would not allow repeated interior knots, which can be used to introduce slope discontinuities between the subsegments.

While slope discontinuities can instead be introduced by repeating control points in the surface definition, this type of discontinuity has the undesirable property of a zero length normal. To allow for slope discontinuity and also retain the necessary surface properties, knots must be allowed to be repeated. The effect of repeating knots is to relax the condition on continuity of derivatives at the ends of subsegments -- one less derivative for each time a knot is repeated. However, the surface properties are defined. Repeating knots also makes some subsegments of the curve have zero length. A knot vector with repeated interior knots produces what is called a <u>subspline basis</u> [12].

In order to include a more general class of B-spline, the uniform condition $x_{i+1} = x_i + 1$, is relaxed to also allow $x_{i+1} = x_i$. If x_{i+1} is equal to x_i then the knot is said to be repeated. It can be repeated up to k-1 times, where k is the order of the curve. This type of knot vector will be called an <u>enhanced uniform knot vector</u> which will produce <u>Enhanced Uniform Rational B-splines</u> (EURBS).

After a review of rational B-spline curves and surfaces, restrictions that can be applied to the knot vector were discussed. The uniform condition of the extended uniform knot vector was relaxed to introduce the enhanced uniform knot vector. The next chapter will discuss the enhanced uniform knot vector and present the CURBS algorithm for calculating with it.

CHAPTER III

CURBS ALGORITHM

3.0 ENHANCED UNIFORM RATIONAL B-SPLINE

The last chapter defined some restrictions on the knot vector. A class of Rational B-spline, the enhanced uniform, was introduced. This chapter describes the advantages of using EURBS and an algorithm for calculating with them. Finally, it presents the limitations of using this class of B-spline, as opposed to the more general NURBS.

3.1 ADVANTAGES OF UNIFORM KNOT VECTOR

In most engineering applications, surfaces free of ripples are desired. The higher the order of the curve used, the more difficult it is to eliminate ripples. To produce a rippleless surface, the order of the curves used is usually restricted to six or less. Therefore, if a uniform or enhanced uniform knot spacing is used, then equations 2.1.2 and 2.1.3 can be used to calculate the basis functions with acceptable accuracy. This allows a simple equation of degree k-1, for each of the basis functions, to be formulated before a specific surface is to be evaluated. An example of this formulation is given in Appendix B. There are, however, an infinite number of possible knot vectors, resulting in an infinite number of blending functions. Next it will be shown how to translate the parametric value to reduce the number of possible blending functions to a finite tractable set, for any given order k of B-spline.

3.1.1 Knot Vector Translations

Making the EURBS restriction to the knot vector allows for the derivation of all possible blending functions of a given order. In order to do this derivation, the parametric variable is restricted to values between zero and one for each subsegment. This is done by translating the knot vector for each non-zero length subsegment so that the i^{th} knot is zero and the i+1 knot is one. Repeated knots form zero-length subsegments which are of no interest. Therefore the transformation is possible for any non-zero subsegment. For example, the knot vector

$$\begin{array}{llcccccccccccc} X & = & [0 & 0 & 0 & 0 & 1 & 2 & 2 & 3 & 4 & 4 & 4 & 4] \\ i & = & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \end{array}$$

would be translated to the knot vector

$$\begin{array}{llcccccccccccc} X & = & [-1 & -1 & -1 & -1 & 0 & 1 & 1 & 2 & 3 & 3 & 3 & 3] \\ i & = & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \end{array}$$

for the second subsegment and

$$\begin{array}{llcccccccccccc} X & = & [-2 & -2 & -2 & -2 & -1 & 0 & 0 & 1 & 2 & 2 & 2 & 2] \\ i & = & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \end{array}$$

for the third subsegment. Appendix C shows that this is a linear mapping of the parametric variable and has no effect on the values of the resulting blending functions for any given value of the parametric variable.

3.1.2 Enhanced Uniform Knot Vector Compression

For a given subsegment of a curve, only a finite portion of the knot vector is actually used in the calculation of the non-zero basis functions for that subsegment. If the recursion of equation 2.1.3 is followed, the portion of the knot vector which affects the blending functions can be determined. If k is the order of the curve, the <u>effective knot vector</u> for a subsegment reaches k-2 knots back from the starting knot and k-1 knots forward of the starting knot, for a total of $2\cdot(k-1)$ knots. In the above example, if k=4 the effective knot vector would have a length of $2\cdot(4-1)$ or six. For the first subsegment, it would be

$$X_e = [0\ 0\ 0\ 1\ 2\ 2].$$

For the second subsegment, it would be

$$X_e = [0\ 0\ 1\ 2\ 2\ 3]$$

which could be shifted to

$$X_e' = [-1\ -1\ 0\ 1\ 1\ 2].$$

For the third it would be

$$X_e = [1\ 2\ 2\ 3\ 4\ 4]$$

which could be shifted to

$$X_e' = [-1\ 0\ 0\ 1\ 2\ 2].$$

Using parameter translation and the finite length of the the effective knot vector, the set of possible effective knot vectors has been reduced to a finite set, given a bound on the order of the spline. The number of possible knot vectors in the set is determined by the order k of the curve and is $2^{(2\cdot k-4)}$. For k=4, there are $2^{(2\cdot 4-4)}$ or sixteen possible knot vectors for a subsegment. For k=5, there are sixty-four possible knot vectors.

Each of the $2^{(2\cdot k-4)}$ knot vectors yields a unique set of blending functions. In order to determine easily which set of blending functions should be used for a given subsegment, each subsegment is given a label or pointer to the correct set of blending functions. The label is determined by compressing the effective knot vector into a binary-encoded label.

3.1.3 Knot Vector Compression Algorithm

The effective knot vector is compressed into a unique binary-encoded label by the compression algorithm given in Appendix D. The algorithm compares successive entries of the effective knot vector to define a sequence of $2 \cdot k-4$ bits, which are reversed and stored as an integer label. This label will be referred to as a subsegment label. Since a unique set of effective knot vectors for a given order has been identified, a set of blending functions for each unique knot vector can be formulated, as in Appendix B.

Now that all the enhanced uniform blending functions have been identified and labeled we are ready to present the CURBS algorithm for calculating with rational B-splines.

3.2 THE EVALUATION ALGORITHM

This section presents an algorithm we call the Converted to Uniform Rational B-Splines (CURBS) algorithm for evaluation of enhanced uniform rational B-splines. It assumes that all the blending functions have been preformulated and stored in matrix fashion according to the labels determined from the effective knot vector. Note that the formulation of the blending functions is done only once and the results are stored for later use. The algorithm as stated below also assumes that the subsegment labels are calculated or read in with the control points that define the surface.

The algorithm requires the following information to be given :

1) Control points for the subsegment : CP' vector
2) Subsegment label : LAB
3) Order of the curve : K
4) A strictly rational indicator : RATNOT
5) Parametric value (between 0.0 and 1.0) : T

The following informal explanation preceding the algorithm specification may assist the reader in understanding the algorithm.

```
Step 1: Use LAB and K to retrieve the proper
        set of blending functions : $[MAT_K]$
Step 2: Construct the t vector :
        $\{t \text{ vector}\} = \{ t^{K-1}, t^{K-2}, \ldots, t, 1 \}$
Step 3: Evaluate blending functions :
```

$\{Bf\} = \{t \text{ vector}\} \cdot [MAT_K]$

Step 4: Calculate homogeneous point :

$\{WP\} = \{Bf\} \cdot \{CP'\}$

Step 5: Calculate three-dimensional point if strictly rational :

$\{P\} = \{WP\} / W$

For K=4 the CURBS algorithm can be coded as follows: (the coordinates of the control point CP$'$ for a subsegment are contained in the PX, PY, PZ, and PW vectors).

```
T2 = T*T
T3 = T*T2
BF(1) = T3*MAT4(1,1,LAB) + T2*MAT4(2,1,LAB)
      + T*MAT4(3,1,LAB) + MAT4(4,1,LAB)
BF(2) = T3*MAT4(1,2,LAB) + T2*MAT4(2,2,LAB)
      + T*MAT4(3,2,LAB) + MAT4(4,2,LAB)
BF(3) = T3*MAT4(1,3,LAB) + T2*MAT4(2,3,LAB)
      + T*MAT4(3,3,LAB) + MAT4(4,3,LAB)
BF(4) = T3*MAT4(1,4,LAB)
WX = BF(1)*PX(1) + BF(2)*PX(2)
   + BF(3)*PX(3) + BF(4)*PX(4)
WY = BF(1)*PY(1) + BF(2)*PY(2)
   + BF(3)*PY(3) + BF(4)*PY(4)
```

```
WZ = BF(1)*PZ(1) + BF(2)*PZ(2)
   + BF(3)*PZ(3) + BF(4)*PZ(4)
IF(RATNOT) THEN
  W = BF(1)*PW(1) + BF(2)*PW(2)
    + BF(3)*PW(3) + BF(4)*PW(4)
  X = WX/W
  Y = WY/W
  Z = WZ/W
ELSE
  X = WX
  Y = WY
  Z = WZ
END IF
```

If derivatives of the point with respect to the parametric variable are required, the algorithm is modified. The appropriate derivative of the t vector is taken and used in step 3 to evaluate an alternate set of blending functions. This set of blending function derivatives is used in step 4. If the surface is strictly rational, then the chain rule must be employed to replace step 5 to calculate the derivative. For example, to calculate dX/dt it would be necessary to use the following equation:

$$dX/dt = dWX/dt \div W - WX \cdot dW/dt \div (W^2)$$

These modifications are shown in the following piece of code.

```
T2 = T*T
THREET2 = 3*T2
TWOT = 2*T
DBF(1) = THREET2*MAT4(1,1,LAB) + TWOT*MAT4(2,1,LAB)
       + MAT4(3,1,LAB)
DBF(2) = THREET2*MAT4(1,2,LAB) + TWOT*MAT4(2,2,LAB)
       + MAT4(3,2,LAB)
DBF(3) = THREET2*MAT4(1,3,LAB) + TWOT*MAT4(2,3,LAB)
       + MAT4(3,3,LAB)
DBF(4) = THREET2*MAT4(1,4,LAB)
IF(RATNOT) THEN
  T3 = T*T2
  BF(1) = T3*MAT4(1,1,LAB) + T2*MAT4(2,1,LAB)
        + T*MAT4(3,1,LAB) + MAT4(4,1,LAB)
  BF(2) = T3*MAT4(1,2,LAB) + T2*MAT4(2,2,LAB)
        + T*MAT4(3,2,LAB) + MAT4(4,2,LAB)
  BF(3) = T3*MAT4(1,3,LAB) + T2*MAT4(2,3,LAB)
        + T*MAT4(3,3,LAB) + MAT4(4,3,LAB)
  BF(4) = T3*MAT4(1,4,LAB)
  WX = BF(1)*PX(1) + BF(2)*PX(2)
     + BF(3)*PX(3) + BF(4)*PX(4)
  WY = BF(1)*PY(1) + BF(2)*PY(2)
     + BF(3)*PY(3) + BF(4)*PY(4)
  WZ = BF(1)*PZ(1) + BF(2)*PZ(2)
     + BF(3)*PZ(3) + BF(4)*PZ(4)
  W  = BF(1)*PW(1) + BF(2)*PW(2)
     + BF(3)*PW(3) + BF(4)*PW(4)
END IF
DWXDT = DBF(1)*PX(1) + DBF(2)*PX(2)
      + DBF(3)*PX(3) + DBF(4)*PX(4)
DWYDT = DBF(1)*PY(1) + DBF(2)*PY(2)
      + DBF(3)*PY(3) + DBF(4)*PY(4)
DWZDT = DBF(1)*PZ(1) + DBF(2)*PZ(2)
      + DBF(3)*PZ(3) + DBF(4)*PZ(4)
IF(RATNOT) THEN
  DWDT = DBF(1)*PW(1) + DBF(2)*PW(2)
       + DBF(3)*PW(3) + DBF(4)*PW(4)
  W2 = W*W
  DXDT = DWXDT/W - WX*DWDT/(W2)
  DYDT = DWYDT/W - WY*DWDT/(W2)
  DZDT = DWZDT/W - WZ*DWDT/(W2)
ELSE
  DXDT = DWXDT
  DYDT = DWYDT
  DZDT = DWZDT
END IF
```

The above algorithms are given for a curve. To evaluate surface points the CP' vector is replaced by a CP' matrix which is post-multiplied by the second set of blending functions corresponding to the other parametric variable.

Unlike the Cox-deBoor algorithm, this algorithm does not rederive the blending functions as they are being evaluated for a surface point and/or derivative. However, the application of algorithm is restricted to rational B-splines defined with enhanced uniform knot vectors.

3.3 LIMITATIONS OF ENHANCED UNIFORM KNOT VECTOR

Having the formulas for all of the blending functions predefined in the program appears to be more efficient for evaluating B-splines and derivatives than building and evaluating the blending functions for each point. However, the use of EURBS appears to limit the B-splines which can be described to a subset of the B-splines with a more general nonuniform knot vector. But Chapter Five presents an algorithm to insert knots into a knot vector so that any knot vector can be transformed to the enhanced uniform format, removing the apparent limitation.

We have discussed the advantages of a uniform knot vector, and reduced the infinite set of enhanced uniform knot vectors to a finite set by knot vector translation. The effective knot vector was defined and then compressed into a subsegment label. An evaluation algorithm was then presented. In the next chapter we shall discuss the performance differences between the above CURBS algorithm and the Cox-deBoor algorithm.

CHAPTER IV

CURBS ALGORITHM ANALYSIS

4.0 ALGORITHM COMPARISON

The last chapter presented the CURBS algorithm for calculating with EURBS. This algorithm relies on the use of preformulated blending function matrices and the ability to determine which matrix should be used for a given subsegment of the curve. In this chapter the CURBS algorithm will be compared to the well-known Cox-deBoor algorithm for calculating with B-splines. The comparison is made by classifying each algorithm according to order of complexity and then according to the number of machine operations required.

4.1 COX-DEBOOR ALGORITHM

Cox [11] and deBoor [10] separately developed the same numerically stable algorithm for computing the basis functions for B-splines. This algorithm is the standard of today's B-spline algorithms and is in wide use in many CAD/CAM systems. For the purposes of analysis the algorithm as presented by deBoor [10] is reproduced below:

```
Set N(1,1) = 1
for s = 1 ,...., k-1, do:
 .
 .      set DP(s) = ti+s-t, Dm(s) = t-ti+1-s,
 .
 .      set N(1,s+1) = 0:
 .
 .      for r = 1 ,...., s, do:
 .        .
 .        .    set M = N(r,s) / (DP(r)+DM(s+1-r)),
 .        .
 .        .    set N(r,s+1) = N(r,s+1) + DP(r)*M,
 .        .
 .............set N(r+1,s+1) = DM(s+1-r)*M.
```

where

k is the order of the blending functions.

The columns of N are used to evaluate the B-spline and/or its derivatives as shown below.

The B-spline is then evaluated by :

$$P(t) = \sum_{i=0}^{k-1} CP_i \cdot N_{i,k}(t) \qquad 4.1.1$$

Where

$N_{i,k}$ is the kth column of N in the above algorithm

To evaluate derivatives, the appropriate column of N would be used. However, N is a triangular matrix, so a set of derivative control points must first be calculated to be used with a specific column of N. The general formula given by deBoor for this evaluation is :

$$F^{(j)}(t) = (k-1) \ldots (k-j) \sum_{i=0}^{k-j-1} A_i^{(j)} \cdot N_{i,k-j}(t) \qquad 4.1.2$$

Where

j is the desired derivative $0 \leq j < k$

$N_{i,k-j}$ is the $(k-j)^{th}$ column of N

$$A_i^{(0)} = CP_i$$
$$A_i^{(j)} = (A_i^{(j-1)} - A_{i-1}^{(j-1)}) \;/\; (t_{i+k-j} - t_i) \qquad 4.1.3$$

4.2 BIG-OH ANALYSIS

One way to classify an algorithm is the big-oh [14] method. The big-oh analysis determines the order of complexity of an algorithm. This analysis performed on the above Cox-deBoor algorithm results in a classification of k^2 or $O(k^2)$. The same analysis of the CURBS algorithm given in the previous chapter produces $O(k)$.

Big-oh analysis indicates that the computation time using the CURBS algorithm increases linearly with k. It also indicates that computation time using the Cox-deBoor algorithm increases as k^2. This savings of an order of magnitude becomes significant when k becomes large. For our application, the size of k is usually six or less, so big-oh analysis is not a very complete measure of the relative performance of the algorithms. It does provides an indication of the relative behavior for cases in which k is required to be large (>>6). While the size of k is usually small, the number of points and/or derivatives to be calculated per picture is usually very large. Therefore the potential for considerable time savings is good if a more detailed operation analysis indicates a significant difference in relative efficiency for the usual range of values for k.

4.2.1 Operation Analysis

When the big-oh analysis is not sufficient to capture the differences between two algorithms, a more detailed analysis is needed to determine performance. A second way to classify algorithms is to determine the number of machine operations required. The following tables give the number of operations required by each algorithm for k=3 and k=5, where an operation is defined as an addition, subtraction, multiplication, division, or assignment of value. The tables present the number of operations required to evaluate a point (n=0), or the j^{th} derivative, or evaluate the point and n derivatives.

	Operations for the jth derivative			Operations for point and n derivatives			
	j=3	j=2	j=1	n=0	n=1	n=2	n=3
k=3	1	44	76	76	117	160	161
k=5	146	181	222	212	295	409	529

Table 4.2.1 : Analysis of the Operations for the Cox-deBoor Algorithm

	Operations for the jth derivative			Operations for point and n derivatives			
	j=3	j=2	j=1	n=0	n=1	n=2	n=3
k=3	9	9	16	20	36	45	54
k=5	30	42	56	54	106	146	176

Table 4.2.2 : Analysis of the operations for the CURBS Algorithm

This analysis was performed for the evaluation of a single coordinate of a B-spline curve and/or the derivatives of a single coordinate. The implied integer multiplication and addition for matrix subscript indexing are not included. To evaluate other coordinates only requires that the evaluated blending functions be used on the remaining coordinates of the control points. Thus the other coordinates would be quicker to evaluate once the first one is done. Table 4.2.3 contains the operations required by each algorithm to evaluate three coordinates and three coordinates with derivatives.

	k=3			k=5		
	n=0	n=1	n=2	n=0	n=1	n=2
C-D	104	227	356	280	529	871
CURBS	32	60	81	74	146	206

Table 4.2.3 : Operations Analysis for Three Coordinates

This table indicates that the Cox-deBoor algorithm requires more effort to evaluate the additional derivatives. This is because the Cox-deBoor algorithm must calculate the $A_i^{(j)}$ points for each coordinate for each derivative to be used with the columns of the N matrix. The CURBS algorithm always uses the same set of control points for all evaluations.

For a surface, the algorithms would have to be used a second time on the second parametric variable and the resulting evaluated blending functions used on a matrix of control points.

The above tables show that the CURBS algorithm uses fewer machine operations than Cox-deBoor for B-spline evaluations. While the actual time difference for using each algorithm will vary from processor to processor, the reduction of operations required does imply an approximate reduction in computation time.

The CURBS algorithm does require additional storage for the precalculated matrices. For k=4 the additional storage required is sixteen 4-by-4 matrices. For k=6 the storage required is 256 6-by-6 matrices. The additional storage is a minor factor with the low cost of computer memory and the availability of virtual memory in computers.

The CURBS algorithm has been shown to produce a three-to-one reduction in machine operations over using the Cox-deBoor algorithm. The CURBS algorithm has no numerical stability problems. It does, however, require that the blending function matrices be constructed in a numerically stable manner. The numerical problems of using equations 2.1.2 and 2.1.3 for construction of the the blending functions presented in [10] and [11] are avoided because of the uniform nature of the knot vector being used and because the order of the curves involved is not large.

The Cox-deBoor algorithm has been reviewed and a big-oh analysis performed comparing it to the CURBS algorithm. The big-oh analysis indicated a major complexity difference between the two algorithms, but was inconclusive as to the importance of the difference in commonly encountered situations. A more detailed operation analysis was performed which indicated that the CURBS algorithm would give a three-to-one operation savings over Cox-deBoor. The only limiting factor to the CURBS algorithm is that the knot vector must be enhanced uniform, which limits its application to a subset of the NURBS. In the next chapter, we will overcome this limitation by developing a transformation to transform an arbitrary NURBS surface definition to an equivalent EURBS surface.

CHAPTER V

CONVERSION ALGORITHM

5.0 NURBS-TO-EURBS CONVERSION

In the preceding chapter the CURBS algorithm was shown to have a significant performance advantage over the Cox-deBoor algorithm. However, it was also noted that the surface must be defined with an EURBS knot vector which is a subset of the NURBS knot vector. This appears to be significant in view of the fact that a uniform knot vector does not necessarily produce a curve with uniform arc length parameterization, a desirable attribute for applications such as generation of tool paths for numerically controlled machines. In order to produce a curve with uniform arc length parameterization, the knot spacing must reflect the relative arc lengths of the subsegments, which implies a nonuniform knot vector.

The above limitations on the use of the algorithm can be eliminated by converting surface representations based on nonuniform knot vectors into exactly equivalent surfaces based on our enhanced uniform knot vector. This chapter presents the design of an algorithm to do the NURBS-to-EURBS conversion. Discussion of the initial attempts to rescale without expanding the knot vector are followed by the exact reformulation of EURBS based on NURBS. After presentation of the algorithm, the additional storage required to represent the surface in the EURBS form is discussed.

5.1 INITIAL EXPLORATION OF THE CONVERSION PROBLEM

Two approaches to converting NURBS to EURBS were tried before a suitable method was derived. These initial approaches were tried to determine whether a conversion was possible which would use the same number of control points as the original NURBS surface. The next section presents a discussion of one initial approach and includes examples showing the shortcomings.

5.1.1 Knot Vector Rescaling

The first approach explored was the use of an enhanced uniform knot vector with the same number of entries as the nonuniform one to be replaced. This implies that both definitions use the same number of control points (although relocated) in the surface definition. This method results in a rescaling of the parametric variable for each non-zero subsegment from $\Delta = x_{i+1} - x_i$ to $\Delta = 1$, thus changing to EURBS-based blending functions. Unfortunately, this method is limited in the type of knot vector to which it can be successfully applied. This is shown in the following proof.

Define a nonuniform knot vector

$$X = [x_0, x_1, \ldots, x_n]$$

corresponding to the parametric variable t. Let Y be an enhanced uniform knot vector

$$Y = [y_0, y_1, \ldots, y_n]$$

corresponding to the parametric variable

s. The following relationship holds between s and t for the j th subsegment:

$$s_j = (t - x_j)/(x_{j+1} - x_j) + (j - 1).$$

If continuity of the first derivative is assumed between the j th and (j+1) th subsegments, then:

$$s_j = (t - x_j)/(x_{j+1} - x_j) + (j - 1)$$

$$s_{j+1} = (t - x_{j+1})/(x_{j+2} - x_{j+1}) + j$$

Equating these relations at the junction of the two subsegments, $t = x_{j+1}$ yields the relation that must exist for the knots.

$$(x_{j+1} - x_j) = (x_{j+2} - x_{j+1})$$

The x_j entries can take on values such that $(x_{j+1} - x_j) \neq (x_{j+2} - x_{j+1})$. This means that $ds_j \neq ds_{j+1}$ at the junction, which is a contradiction. Thus this method would not be able to yield an equivalent enhanced uniform representation for all nonuniform knot vectors.

Another way to view this rescaling is to think of it as a linear mapping from subsegments of t-space to a new s-space. Each non-zero distance Δ between knots in the t-space is mapped to $\Delta = 1$ in the s-space. Because the Δ's in the t-space do not have to be constant, the mapping to s-space is, in general nonlinear.

Thus to use this form of rescaling, the knots in each effective knot vector must have a constant or zero spacing, which will insure the linear mapping. An effective knot vector with a constant or zero knot spacing will be said to have the <u>constant delta property</u>.

5.1.1.1 An Example Of Approximate Knot Vector Rescaling

The following figures will demonstrate the above rescaling. Figure 5.1.1 contrasts the use of a nonuniform knot vector and a uniform knot vector for the same set of control points. (Appendix E presents the data for these figures). Figures 5.1.2 and 5.1.3 show the results of solving for new control points to try to fit the NURBS curve.

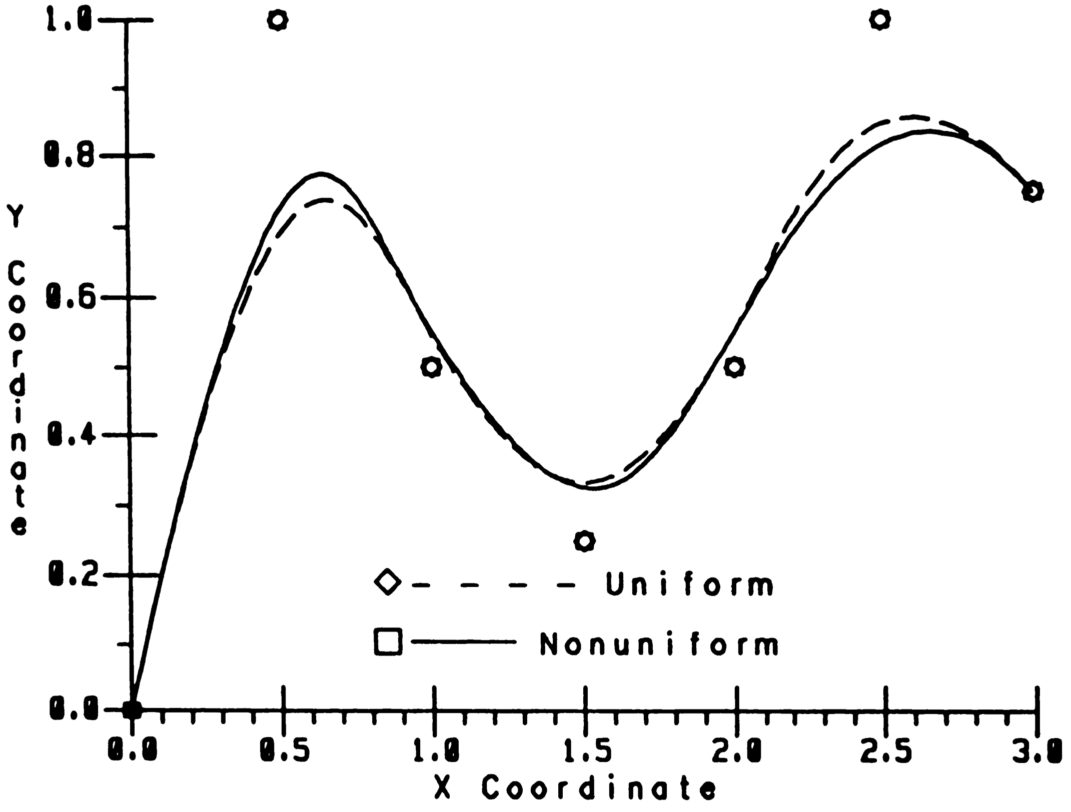

Figure 5.1.1 : Uniform vs Nonuniform Curves

The control points for Figure 5.1.2 were found by requiring the curve to pass through the ends of each segment and match the slopes on the ends.

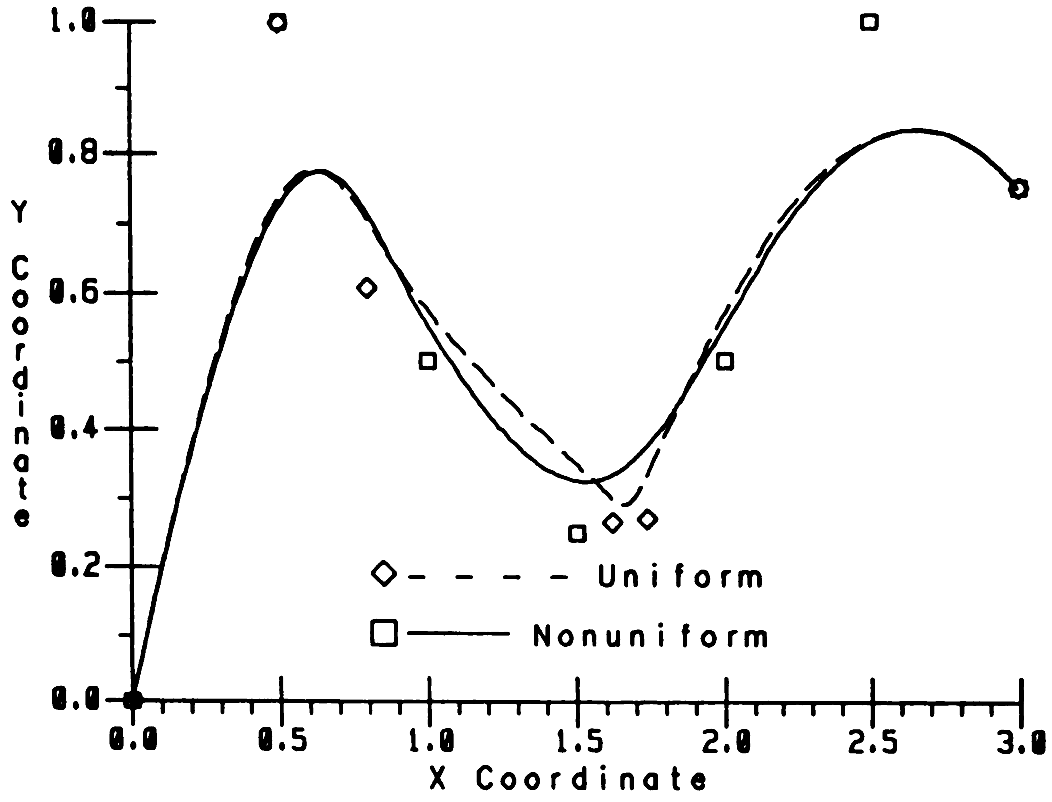

Figure 5.1.2 : Uniform Fit Using End Slopes

The control points for Figure 5.1.3 were found by fitting the first segment and then requiring the curve to pass through the end points of the remaining segments. By design there are no discontinuities, but the fitted curves do not represent the original definition.

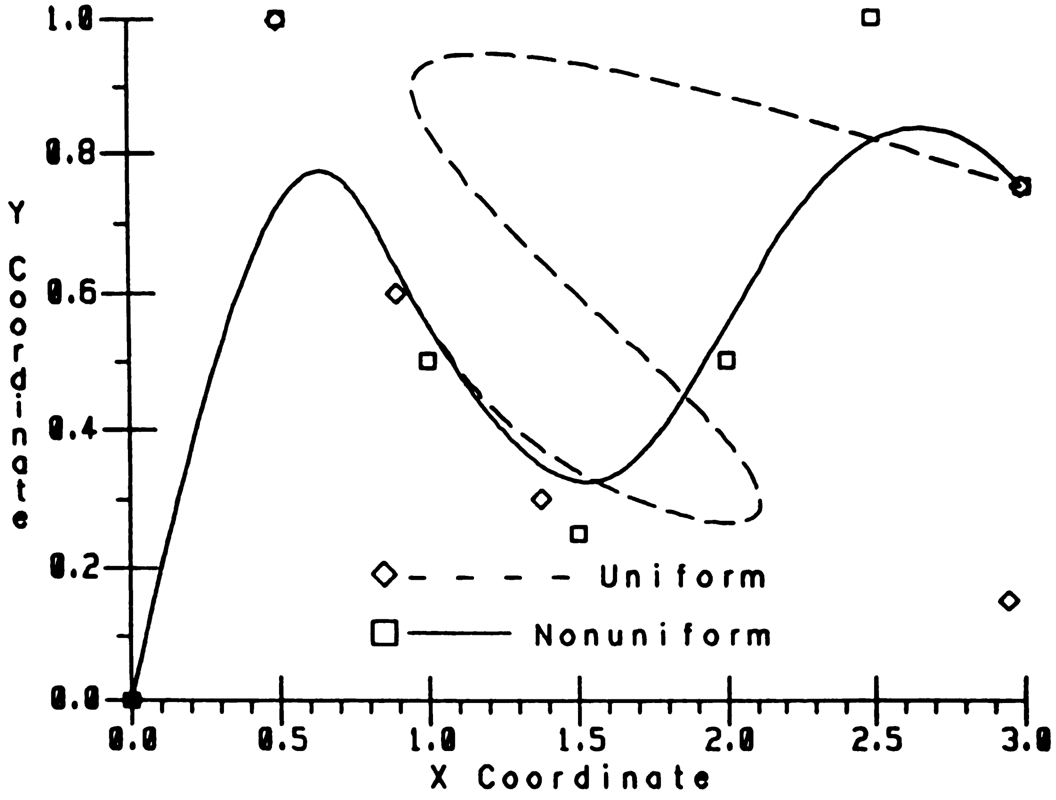

Figure 5.1.3 : Uniform Fit Using First Subsegment

In Figure 5.1.4 we have used ordinary least squares [16] to try to fit the nonuniform data to a model using the blending functions based on the enhanced uniform knot vector. The plot is the uniform curve that fits the original nonuniform curve with the least amount of error between input data points and corresponding points on the uniform curve.

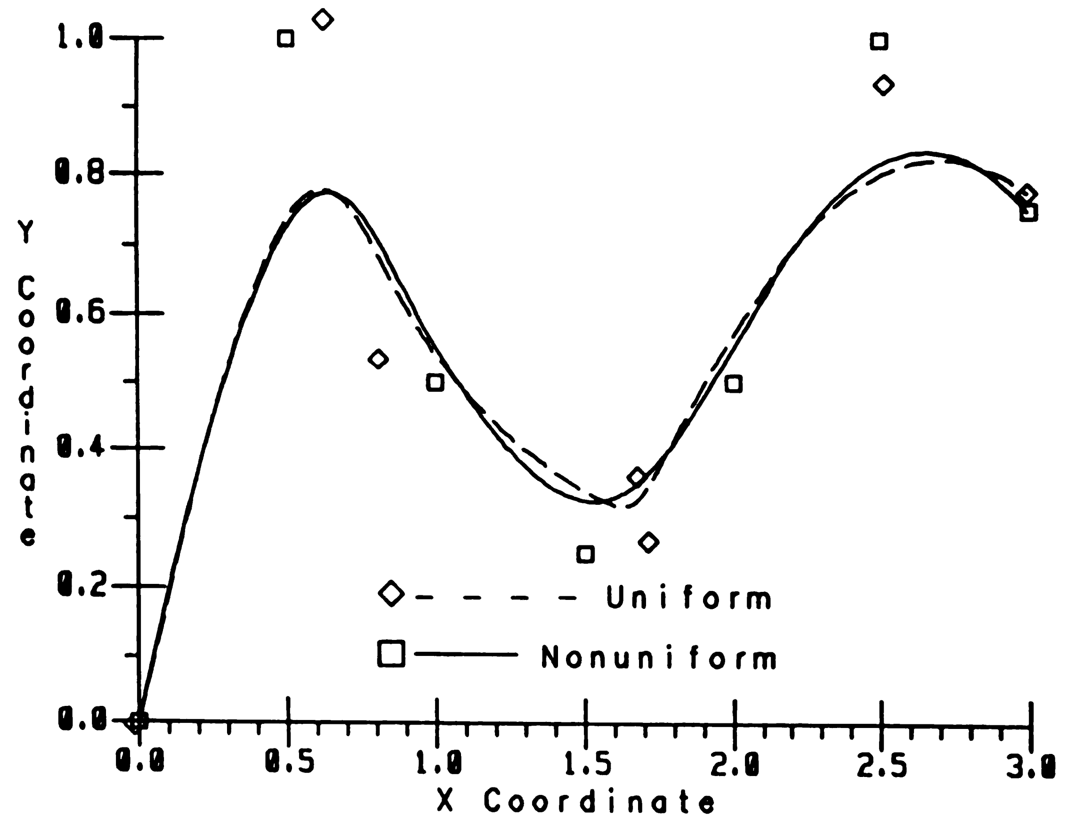

Figure 5.1.4 : Uniform Fit Using Least Squares

5.2 EXACT NURBS-TO-EURBS CONVERSION

The following conversion technique was derived after the initial explorations described above showed the necessity for additional knots and control points. While it may produce a surface with more control points than the original surface, it is an exact conversion and the resulting surface can take advantage of all the EURBS properties.

5.2.1 Expanding The Knot Vector

The method described in section 5.1.1 would work if each effective knot vector had the constant delta property. In order to obtain the constant delta property, new knots are inserted or repeated to expand the new EURBS knot vector. This implies that additional control points will also be inserted into the surface definition. Only zero-length subsegments are added, so the total number of non-zero subsegments remains the same.

The insertion is done such that the resulting effective knot vector for each subsegment has the constant delta property. This forces the mapping or rescaling of each subsegment to be linear. The additional enhanced uniform control points give the flexibility needed to match the NURBS curve exactly.

The result of this method is shown in Figure 5.2.1. The resulting EURBS curve is identical to the NURBS curve, and of course retains the same level of continuity between subsegments. That is, all continuous derivatives of the nonuniform spline remain continuous in the uniform representation. The algorithm for expanding the knot vector and finding the new set of control points is given below.

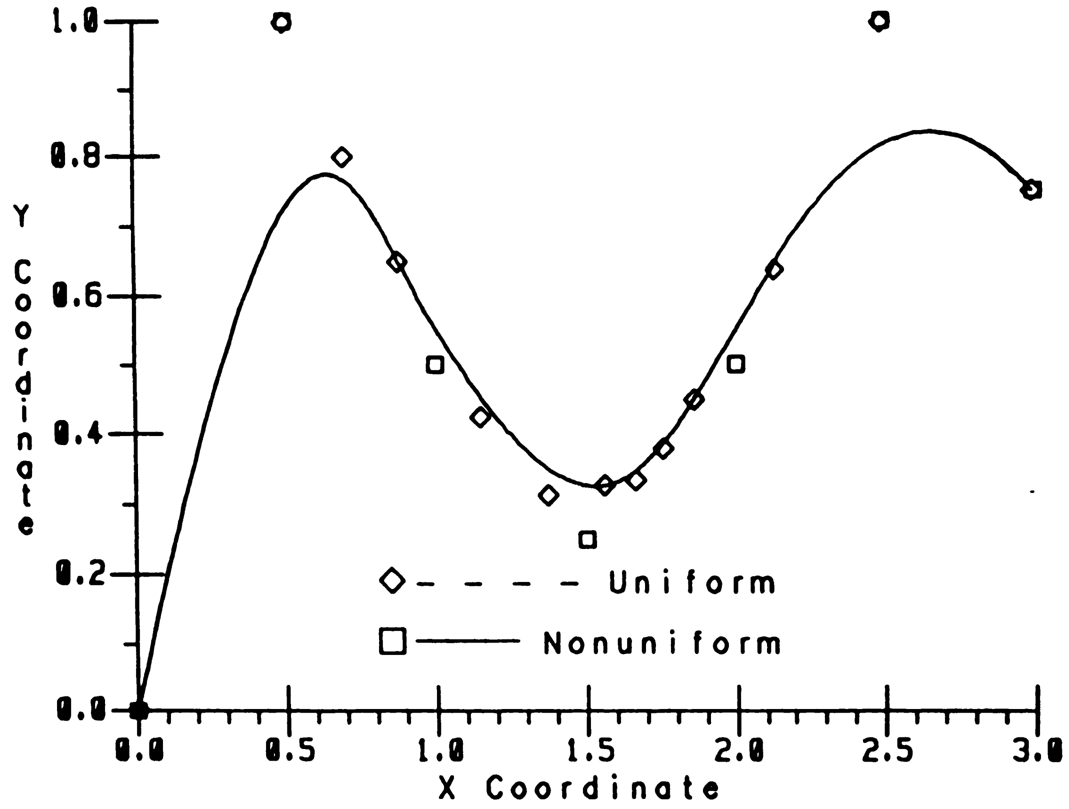

Figure 5.2.1 : Uniform Fit By Knot Vector Expansion

5.2.2 Algorithm for Conversion By Knot Vector Expansion (NURBS-To-EURBS Algorithm)

To fit a unique curve of a given order k, k pieces of information must be given. The information can take the form of k points along the curve or a point and k-1 derivatives or other such combinations. In this algorithm, the point at the end of the subsegment is used along with successive derivatives as necessary. The following outlines the steps involved in the conversion, and the code for this algorithm is given in Appendix F.

```
Step 1 : Copy the NURBS knot vector into
         the new EURBS knot vector.

Step 2 : For each successive effective knot
         vector in the EURBS knot vector Do:
           While not constant delta property Do:
             Duplicate the last knot with constant
             delta property.
           End While
           Label the subsegment per section 3.1.3.
         End For.
```

```
Step 3 : For each successive subsegment in
         the new EURBS knot vector Do:
           Use label to determine the number of
           undetermined control points.
           Evaluate the end point and necessary
           derivatives of the NURBS curve for
           this subsegment.
           Determine the unknown EURBS control
           points for this subsegment.
         End For.
```

This algorithm is given for a curve. To use it for a surface definition, the algorithm is first used on each column of the nonuniform control point matrix with the corresponding knot vector. The resulting expanded columns are used to form an intermediate matrix of control points and the algorithm is then used on the rows of this matrix with the other knot vector. The resulting expanded rows contain the uniform control points that define the mathematically equivalent surface.

5.2.3 Storage Considerations

To use the CURBS algorithm, a submatrix of the control point matrix must be retrieved. The retrieval time will be determined by the size of the submatrix and not the size of the mother matrix. The retrieval of the control points was not included in the operation analysis done because it was assumed to be the same for both algorithms. The additional control points needed for the EURBS definition will have no effect on the performance of the CURBS algorithm. However, there is a possibility of a slight increase in the paging time in a virtual memory environment because of the larger number of control points used *in toto*.

The NURBS-to-EURBS conversion will represent any surface originally based on a nonuniform knot vector with an equivalent surface based on an enhanced uniform knot vector. The price for doing this is that there will be more control points to store for the enhanced uniform representation. The amount of additional storage depends how many new control points are created. An additional k-1 control points will be inserted for each subsegment which does not initially possess the constant delta property. The number of non-zero length subsegments has not increased, and since the knot vector is stored as a set of labels, for non-zero length subsegments in EURBS form, there is no additional storage required for the longer knot vector. The extra virtual memory necessary to store the surface in this form is far outweighed by the time savings resulting from being able to calculate points and derivatives based on prestored blending functions.

Notice that the introduction of new zero-length subsegments does not increase the number of calculations required since no evaluations are done in these segments, and the cost of using the new control points is already included in the evaluation of the CURBS algorithm. Thus, the CURBS algorithm offers a fairly direct tradeoff of storage for computational time. The process of transforming a NURBS surface to an equivalent EURBS surface is done only once, after which the EURBS surface may be manipulated quickly and efficiently by the CURBS algorithm.

The CURBS and NURBS-to-EURBS algorithms which have been developed can be used to evaluate any rational B-spline curve or surface. The implementation of these algorithms in a general-purpose surface assessment package requires developing the routines for the necessary calculations. The next chapter discusses some numerical considerations of the implementation.

CHAPTER VI

IMPLEMENTATION CONSIDERATIONS

6.0 NUMERICAL CONSIDERATIONS

In the preceding chapters, algorithms have been developed for calculating with rational B-splines. If these algorithms are to be used on a computer, they must be implemented using floating point arithmetic. Floating point arithmetic with finite word lengths does not always yield accurate or even remotely usable results.

In this chapter some numerical considerations for implementing these algorithms on a computer will be discussed and some examples given.

When scan line techniques [3] [4] [5] are used, the patch coordinates are transformed to screen coordinates. Many calculations are done in screen space, which we define with X horizontal, Y vertical, and Z normal to the plane of the screen. Each pixel is one unit wide in X and one unit high in Y. This is illustrated in Figure 6.0.1. The coordinate transformation from world space to screen space can present some problems because of the size of the window assigned to the screen; we will discuss this in terms of the *zoom factor*. If the zoom factor is very small, the entire patch may be smaller than one pixel; if the zoom factor is very large, the smallest change that can be made to a parametric variable may represent a difference of more than one pixel.

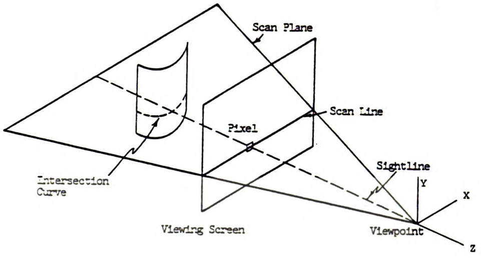

Figure 6.0.1 : Screen Space Coordinate System

6.1 BUILDING OF THE BLENDING FUNCTION MATRICES

The blending functions are used for the evaluation of the surface and its various properties and for the transformation from NURBS to EURBS as well. When building the matrices, care must be taken to avoid introducing numerical errors which change the surface definitions during the transformation. A surface could also be inadvertently altered by using blending functions different from those used by the original designer.

The uniformity of the knot vector yields a denominator which is common to each of the blending function matrices of a given order. When the denominator is factored out of the matrix, the entries take on integer values. The matrix can thus be stored as integer values along with the denominator. This will avoids the roundoff or truncation which occurs if the matrix is stored as real numbers.

6.2 EVALUATION OF THE COX-DEBOOR AND CURBS ALGORITHMS

Roundoff error is usually thought of as a problem when the number of operations becomes large, such as in inverting or reducing a large matrix. However, the examples given later show that roundoff error has a significant effect on the evaluated values of even low-order blending functions.

The algorithm for the transformation from NURBS to EURBS requires that the NURBS surface and its derivatives and the blending functions for the EURBS surface be evaluated at specific points. To ensure that the resulting

EURBS surface is identical to the NURBS surface within the full precision available, these evaluations should be done as accurately as possible.

6.3 SIMULTANEOUS EQUATION SOLVER

The transformation algorithm also requires the solution of a set of linear equations. The number of equations to be solved depends on the number of control points to be found. If only one control point needs to be found for a subsegment, then only one equation is generated. But if multiple control points are to be found, then a stable means must be used to solve the set of linear equations for the control points. Even for the relatively small number of equations to be solved, matrix inversion is not a good means to solve the system of equations because of the roundoff error which occurs and because of the problems which can arise if the matrix is ill-conditioned.

There are many good numerical analysis textbooks such as [17] and [18] which deal with the subject in depth. An acceptable procedure is to use row operations on the augmented matrix to reduce the matrix to triangular form so that back substitution can be used to find the control points.

If increased precision is desired, an iterative method or iterative improvement can be used.

6.4 EXAMPLES OF EVALUATION PROBLEMS

This section presents three examples of problems which occurred when implementing the CURBS algorithm. The central problem in each case was the evaluation of the blending functions. For each example, the order of the surface was 3 or 4 and the evaluation was done on a thirty-two-bit computer.

6.4.1 Evaluation of Normal Components

To use scan-line techniques [3] [4] [5], it is necessary to evaluate the silhouette edges as well as the physical edges of the patch. The silhouette edge of a patch is made up of points on the patch where the Z component of the normal is zero. The evaluations are done on these edges at points where the edge crosses a scan line. This is called walking the edge. Walking the physical edges from scan line to scan line is relatively easy to do.

To walk the silhouette edge, a bi-variate Newton-Raphson method is used to iterate to each scan line that is crossed by the edge.

The result of the Newton-Raphson calculation is a search direction and an approximation of the distance to go in that direction. Figure 6.4.1 is a plot of the Z component of the normal as a function of the distance along the search direction. The basis function evaluation was done in single precision (24-bit mantissa). Depending on the error bounds used for zero, the point could be anywhere along this search direction. Because of the noisy nature of the Z component, convergence is a problem. Figure 6.4.2 is a plot of the Z component along the same search direction with the evaluation of the basis function done in double precision. In this case, the method converges in three iterations.

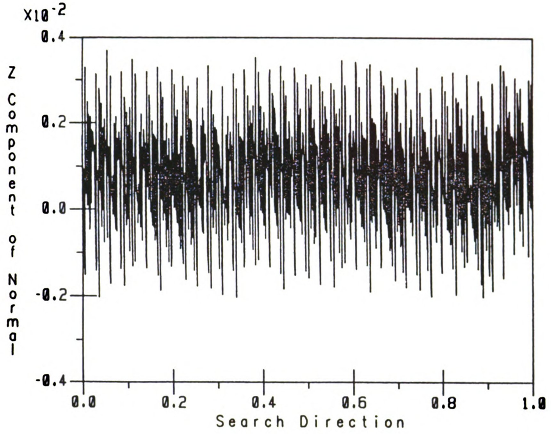

Figure 6.4.1 : Single Precision Evaluation

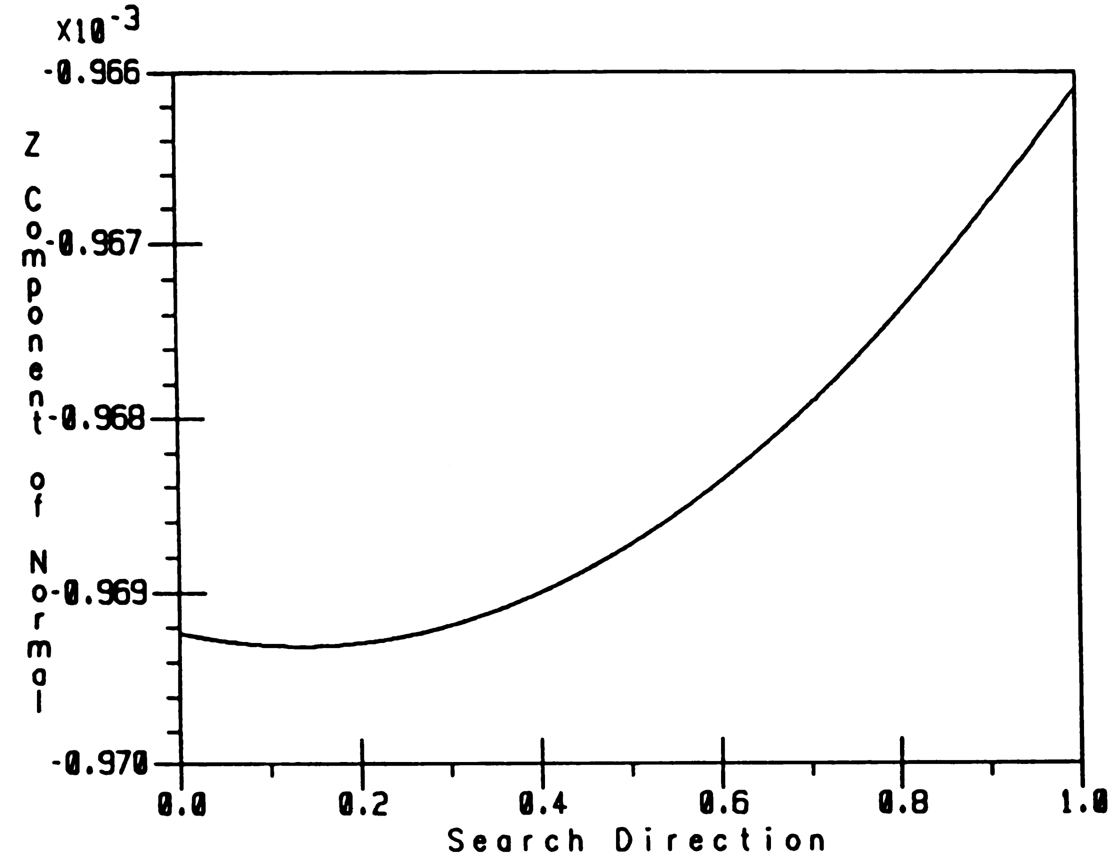

Figure 6.4.2 : Double Precision Evaluation

6.4.2 Normalization Of The Normal

Calculating the Z component of the normal alone requires less effort than computing all three components of the normal. However, this can cause problems with defining what is considered to be zero for the given component. In some cases the normal is changing so rapidly that it is difficult to find a small value of the unnormalized Z component. This is shown in Figure 6.4.3.

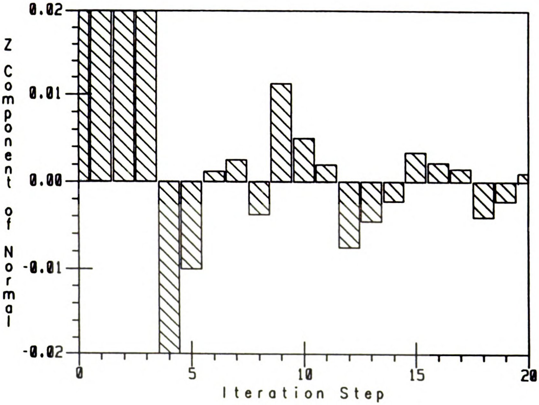

Figure 6.4.3 : Unnormalized Z Component Calculation

If all three components of the normal are calculated and the components normalized so the normal has length one, then what is considered to be zero is defined with respect to the value one. In this case, finding the zero point is not a problem, as shown in Figure 6.4.4. In this case the absolute value of the Z component is between zero and one.

Using this method, acceptable limits about zero can be defined.

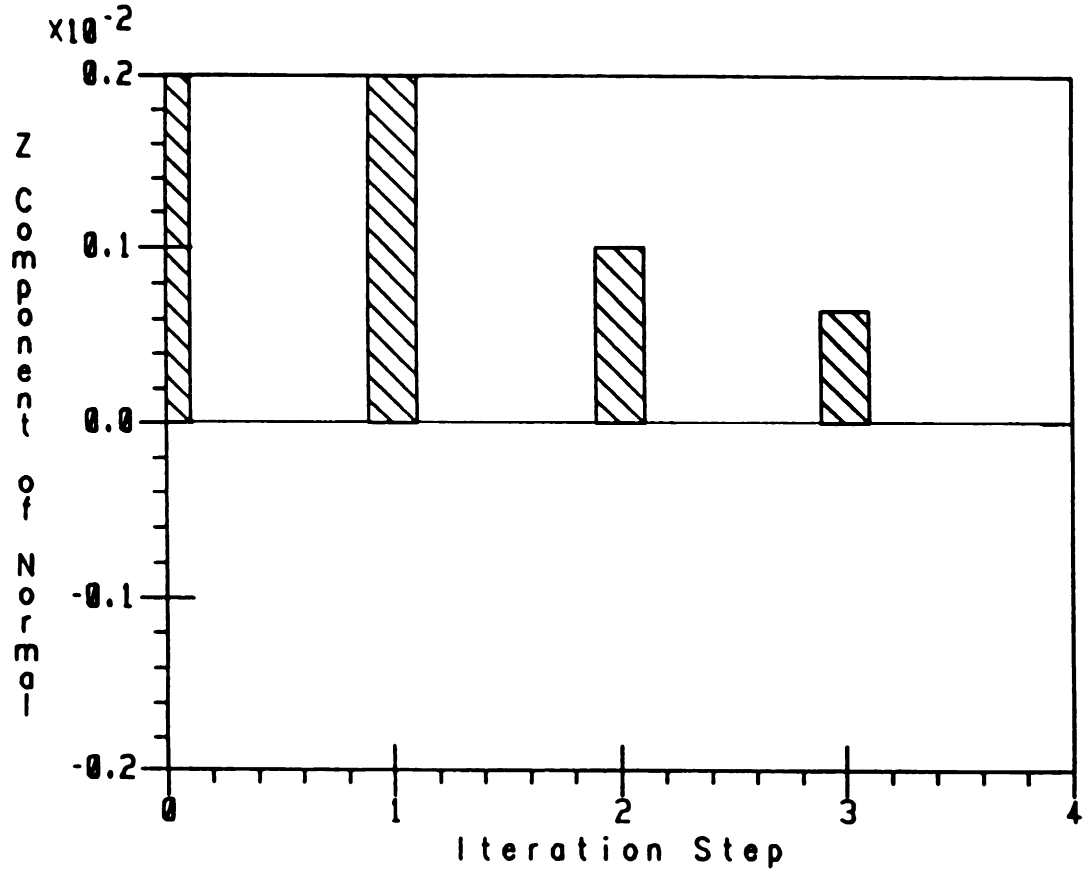

Figure 6.4.4 : Normalized Z Component Calculation

6.4.3 Face To Face

The Z component of the normal is also used to decide which face a point is on. If the Z component is positive, then the point is on the plus face of the surface. If it is negative, then the point is on the minus face. Silhouette edges are then the edges of a face. Even when the Z component is normalized, if the blending functions are not calculated in double precision, the errors which occur result in misleading results, as shown in the Table 6.4.1.

S	T	Single Precision	Double Precision
0.25	0.25	-0.5982137E-04	0.2415371E-02
0.00	0.25	0.0000000E+00	0.3129074E-03
0.75	0.50	-0.6958499E-05	0.7580551E-03

Table 6.4.1 : Example of Calculated Values of the Z Component of the Normal

The errors in the above table may not seem to be significant, and the results are more than adequate for such things as wire frame drawings, outlining patches or subpatches, or finding a pixel. The single precision values, however, are not acceptable for deciding which face a point is on.

Doing evaluations in double precision may require more computation time, as will computing all the components of the normal. But the additional effort is more than repaid when iterative methods converge quickly as opposed to taking many iterations or not converging at all.

In some modern processors, all instructions are done in double precision and rounded to single precision when the results are stored. In this case the only cost associated with using the double precision is the additional storage for the results in that form.

Numerical considerations of the implementation of evaluation algorithms have been discussed, and some examples of problems have been given. Sample output from the surface assesment package, MSU COLORSCOPE, which uses the algorithms developed in this dissertation, is given in the next chapter. In addition to the examples, conclusions are drawn about this work.

CHAPTER VII

EXAMPLES AND CONCLUSIONS

7.0 EXAMPLES AND CONCLUSIONS

The preceding chapters developed algorithms useful for the interactive shading of surfaces defined by rational B-splines. This chapter presents some examples of surfaces shaded using these algorithms. After the examples, some conclusions about this work are drawn.

7.1 Examples

The following seven figures were generated using the MSU COLORSCOPE surface assessment package. These surfaces represent geometric as well as freeform surfaces. These figures are not intended to demonstrate the primary error detection function of the COLORSCOPE package, but rather that the package can handle the rational B-spline surface. The figures also demonstrate some of the capability of the rational B-spline.

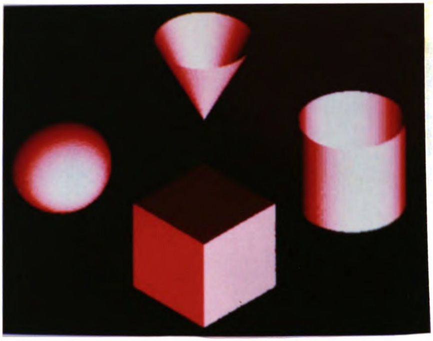
Figure 7.1.1 : Shaded Cube, Sphere, Cone, and Cylinder

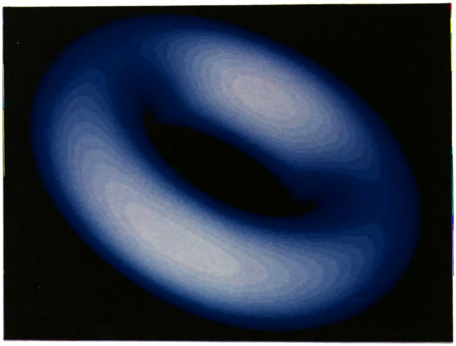
Figure 7.1.2 : Shaded Torus

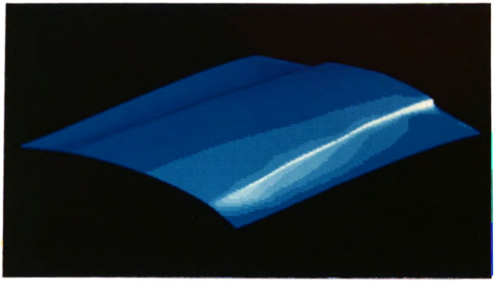

Figure 7.1.3 : Shaded Automobile Hood

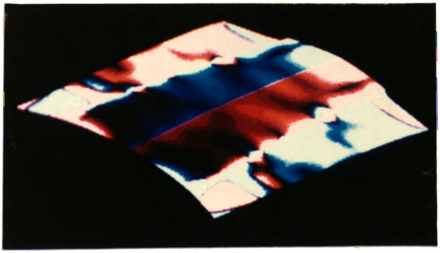

Figure 7.1.4 : Curvature Shading of Automobile Hood

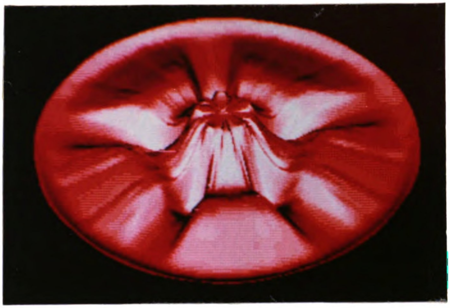

Figure 7.1.5 : Intermediate Stamping of a Styled Wheel

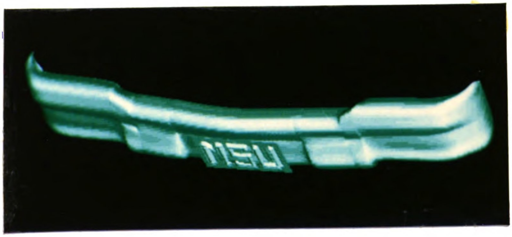

Figure 7.1.6 : Shaded Bumper

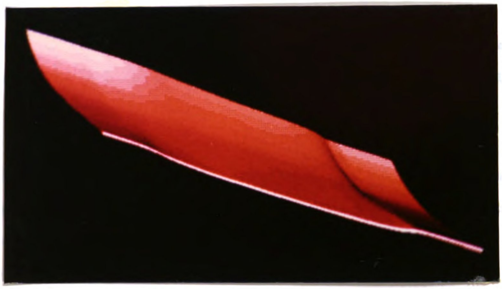

Figure 7.1.7 : Shaded Turbine Blade

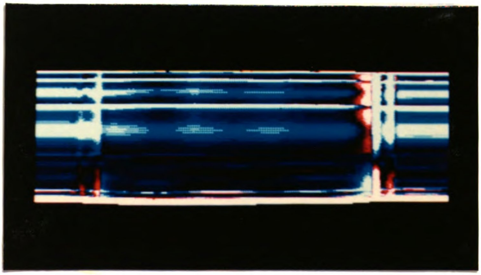

Figure 7.1.8 : Absolute Maximum Curvature Shading of Turbine Blade

7.2 Conclusions

This dissertation reviewed the rational B-spline. It defined a class of rational B-splines called Enhanced Uniform. The advantages of using this class of B-spline were discussed.

The CURBS algorithm for calculating with Enhanced Uniform Rational B-splines was developed and presented. The performance of this algorithm was compared to the commonly used Cox-deBoor algorithm for B-spline evaluation. This comparison indicated that the CURBS algorithm would typically produce a three-to-one savings over the use of the Cox-deBoor algorithm, even for low-order B-splines.

An apparent limitation of the CURBS algorithm, that the B-splines must be of the enhanced uniform rational B-spline type, was overcome by developing the nonuniform-rational-B-spline-to-enhanced-uniform-rational-B-spline (NURBS-to-EURBS) conversion algorithm to convert any surface based on the NURBS knot vector to a mathematically equivalent surface based on the EURBS knot vector. The converted surface was shown to have a storage penalty but no time penalty for surface evaluation. In fact, the CURBS algorithm can be used on the converted surface so that there is a three-to-one savings in machine operations. The CURBS and NURBS-to-EURBS algorithms combine to offer the tradeoff of lowered computation time for increased storage space.

Some numerical consideration of the implementation of the algorithms developed here were discussed. Finally, examples of shaded images produced by these algorithms were given.

The algorithms developed in this dissertation are a viable solution to the problem of rapidly rendering accurate shaded images of rational B-spline surfaces. They offer significant advantages over the prevalent algorithms now in use for this purpose, as well as for other tasks requiring calculations with rational B-splines.

APPENDIX A

CALCULATION EXAMPLE

A. RATIONAL B-SPLINE CALCULATION EXAMPLE

This appendix presents examples of calculating a B-spline and a rational B-spline curve. The results of these calculations were used to generate the figures in Chapter Two of this dissertation.

The control points used to define the B-spline curve are given in Table A.1.

Control point	X	Y
0	-0.1490382E-01	0.2372459E+01
1	-0.1279431E+01	0.2134099E+01
2	-0.1903875E+01	-0.1257925E+01
3	0.1243925E+01	-0.7112664E+00
4	0.1540538E-01	0.1074566E+01
5	-0.7496327E+00	-0.4256855E+00
6	0.6192721E+00	-0.3516818E+00
7	-0.8542724E-02	0.5969771E+00
8	-0.4414083E+00	-0.2523811E+00
9	0.3807828E+00	-0.2515893E+00
10	0.1471788E+00	0.2455054E+00
11	0.1153550E-02	0.1825014E+00

Table A.1 : Original Control Points

To define the rational B-spline curve, a positive non-zero weight is assigned to each control point CP_i. Each of the CP_i is then multiplied by its weight to form the set of homogeneous control points CP_i'. The CP' control points are given in Table A.2.

CP'	WX	WY	W
0	-0.2341119E-01	0.3726701E+01	0.1570818E+01
1	-0.2902891E+01	0.4842038E+01	0.2268892E+01
2	-0.6978094E+01	-0.4610554E+01	0.3665206E+01
3	0.7164465E+01	-0.4096585E+01	0.5759564E+01
4	0.1209919E+00	0.8439503E+01	0.7853872E+01
5	-0.7457782E+01	-0.4234966E+01	0.9948580E+01
6	0.7457602E+01	-0.4235138E+01	0.1204253E+02
7	-0.1207699E+00	0.8439560E+01	0.1413716E+02
8	-0.7164745E+01	-0.4096539E+01	0.1623156E+02
9	0.6978148E+01	-0.4610575E+01	0.1832580E+02
10	0.2902713E+01	0.4841945E+01	0.1972236E+02
11	0.2355580E-01	0.3726728E+01	0.2042028E+02

Table A.2 : Homogeneous Control Points

The curve has been chosen to be a cubic so the order is four (k=4). The following knot vector is used:

$$T = [\,0\ 0\ 0\ 0\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 9\ 9\ 9\,]$$

The parameter varies from zero to nine along the curve and the curve is made up of nine subsegments. Using equations 2.1.2 and 2.1.3 the following non-zero blending functions for each subsegment are found.

Subsegment one $0.0 \leq t < 1.0$

$$N_{00,4}(t) = -1.000 \cdot t^3 + 3.000 \cdot t^2 - 3.000 \cdot t + 1.000$$
$$N_{01,4}(t) = 1.750 \cdot t^3 - 4.500 \cdot t^2 + 3.000 \cdot t + 0.000$$
$$N_{02,4}(t) = -0.917 \cdot t^3 + 1.500 \cdot t^2 + 0.000 \cdot t + 0.000$$
$$N_{03,4}(t) = 0.167 \cdot t^3 + 0.000 \cdot t^2 + 0.000 \cdot t + 0.000$$

Subsegment two $1.0 \leq t < 2.0$

$$N_{01,4}(t) = -0.250 \cdot t^3 + 1.500 \cdot t^2 - 3.000 \cdot t + 2.000$$
$$N_{02,4}(t) = 0.583 \cdot t^3 - 3.000 \cdot t^2 + 4.500 \cdot t - 1.500$$
$$N_{03,4}(t) = -0.500 \cdot t^3 + 2.000 \cdot t^2 - 2.000 \cdot t + 0.667$$
$$N_{04,4}(t) = 0.167 \cdot t^3 - 0.500 \cdot t^2 + 0.500 \cdot t - 0.167$$

Subsegment three $2.0 \leq t < 3.0$

$$N_{02,4}(t) = -0.167 \cdot t^3 + 1.500 \cdot t^2 - 4.500 \cdot t + 4.500$$

$$N_{03,4}(t) = 0.500 \cdot t^3 - 4.000 \cdot t^2 + 10.000 \cdot t - 7.333$$

$$N_{04,4}(t) = -0.500 \cdot t^3 + 3.500 \cdot t^2 - 7.500 \cdot t + 5.167$$

$$N_{05,4}(t) = 0.167 \cdot t^3 - 1.000 \cdot t^2 + 2.000 \cdot t - 1.333$$

Subsegment four $3.0 \leq t < 4.0$

$$N_{03,4}(t) = -0.167 \cdot t^3 + 2.000 \cdot t^2 - 8.000 \cdot t + 10.667$$

$$N_{04,4}(t) = 0.500 \cdot t^3 - 5.500 \cdot t^2 + 19.500 \cdot t - 21.833$$

$$N_{05,4}(t) = -0.500 \cdot t^3 + 5.000 \cdot t^2 - 16.000 \cdot t + 16.667$$

$$N_{06,4}(t) = 0.167 \cdot t^3 - 1.500 \cdot t^2 + 4.500 \cdot t - 4.500$$

Subsegment five $4.0 \leq t < 5.0$

$$N_{04,4}(t) = -0.167 \cdot t^3 + 2.500 \cdot t^2 - 12.500 \cdot t + 20.833$$

$$N_{05,4}(t) = 0.500 \cdot t^3 - 7.000 \cdot t^2 + 32.000 \cdot t - 47.333$$

$$N_{06,4}(t) = -0.500 \cdot t^3 + 6.500 \cdot t^2 - 27.500 \cdot t + 38.167$$

$$N_{07,4}(t) = 0.167 \cdot t^3 - 2.000 \cdot t^2 + 8.000 \cdot t - 10.667$$

Subsegment six $5.0 \leq t < 6.0$

$$N_{05,4}(t) = -0.167 \cdot t^3 + 3.000 \cdot t^2 - 18.000 \cdot t + 36.000$$

$$N_{06,4}(t) = 0.500 \cdot t^3 - 8.500 \cdot t^2 + 47.500 \cdot t - 86.833$$

$$N_{07,4}(t) = -0.500 \cdot t^3 + 8.000 \cdot t^2 - 42.000 \cdot t + 72.667$$

$$N_{08,4}(t) = 0.167 \cdot t^3 - 2.500 \cdot t^2 + 12.500 \cdot t - 20.833$$

Subsegment seven $6.0 \leq t < 7.0$

$N_{06,4}(t) = -0.167 \cdot t^3 + 3.500 \cdot t^2 - 24.500 \cdot t + 57.167$

$N_{07,4}(t) = 0.500 \cdot t^3 - 10.000 \cdot t^2 + 66.000 \cdot t - 143.333$

$N_{08,4}(t) = -0.500 \cdot t^3 + 9.500 \cdot t^2 - 59.500 \cdot t + 123.167$

$N_{09,4}(t) = 0.167 \cdot t^3 - 3.000 \cdot t^2 + 18.000 \cdot t - 36.000$

Subsegment eight $7.0 \leq t < 8.0$

$N_{07,4}(t) = -0.167 \cdot t^3 + 4.000 \cdot t^2 - 32.000 \cdot t + 85.333$

$N_{08,4}(t) = 0.500 \cdot t^3 - 11.500 \cdot t^2 + 87.500 \cdot t - 219.833$

$N_{09,4}(t) = -0.583 \cdot t^3 + 12.750 \cdot t^2 - 92.250 \cdot t + 221.250$

$N_{10,4}(t) = 0.250 \cdot t^3 - 5.250 \cdot t^2 + 36.750 \cdot t - 85.750$

Subsegment nine $8.0 \leq t < 9.0$

$N_{08,4}(t) = -0.167 \cdot t^3 + 4.500 \cdot t^2 - 40.500 \cdot t + 121.500$

$N_{09,4}(t) = 0.917 \cdot t^3 - 23.250 \cdot t^2 + 195.750 \cdot t - 546.750$

$N_{10,4}(t) = -1.750 \cdot t^3 + 42.750 \cdot t^2 - 347.250 \cdot t + 938.250$

$N_{11,4}(t) = 1.000 \cdot t^3 - 24.000 \cdot t^2 + 192.000 \cdot t - 512.000$

Equation 2.1.1 is used to multiply the above blending functions by the appropriate homogeneous control point CP'_i. This yields the following equations for each subsegment:

Subsegment one $0.0 \leq t < 1.0$

$$WX(t) = 2.534 \cdot t^3 + 2.526 \cdot t^2 - 8.638 \cdot t - 0.023$$
$$WY(t) = 8.290 \cdot t^3 - 17.525 \cdot t^2 + 3.346 \cdot t + 3.727$$
$$W(t) = 0.000 \cdot t^3 + 0.000 \cdot t^2 + 2.094 \cdot t + 1.571$$

Subsegment two $1.0 \leq t < 2.0$

$$WX(t) = -6.907 \cdot t^3 + 30.848 \cdot t^2 - 36.961 \cdot t + 9.418$$
$$WY(t) = -0.445 \cdot t^3 + 8.682 \cdot t^2 - 22.861 \cdot t + 12.462$$
$$W(t) = 0.000 \cdot t^3 + 0.000 \cdot t^2 + 2.095 \cdot t + 1.571$$

Subsegment three $2.0 \leq t < 3.0$

$$WX(t) = 3.442 \cdot t^3 - 31.244 \cdot t^2 + 87.223 \cdot t - 73.372$$
$$WY(t) = -6.205 \cdot t^3 + 43.244 \cdot t^2 - 91.985 \cdot t + 58.545$$
$$W(t) = 0.000 \cdot t^3 + 0.000 \cdot t^2 + 2.095 \cdot t + 1.570$$

Subsegment four $3.0 \leq t < 4.0$

$$WX(t) = 3.838 \cdot t^3 - 34.812 \cdot t^2 + 97.927 \cdot t - 84.076$$
$$WY(t) = 6.314 \cdot t^3 - 69.433 \cdot t^2 + 246.044 \cdot t - 279.484$$
$$W(t) = 0.000 \cdot t^3 + 0.002 \cdot t^2 + 2.088 \cdot t + 1.577$$

Subsegment five $4.0 \leq t < 5.0$

$$WX(t) = -7.498 \cdot t^3 + 101.223 \cdot t^2 - 446.212 \cdot t + 641.442$$
$$WY(t) = 0.000 \cdot t^3 + 6.336 \cdot t^2 - 57.030 \cdot t + 124.615$$
$$W(t) = 0.000 \cdot t^3 - 0.003 \cdot t^2 + 2.109 \cdot t + 1.550$$

Subsegment six $5.0 \leq t < 6.0$

$$WX(t) = 3.838 \cdot t^3 - 68.817 \cdot t^2 + 403.989 \cdot t - 775.560$$

$$WY(t) = -6.314 \cdot t^3 + 101.052 \cdot t^2 - 530.608 \cdot t + 913.911$$

$$W(t) = 0.000 \cdot t^3 + 0.003 \cdot t^2 + 2.079 \cdot t + 1.598$$

Subsegment seven $6.0 \leq t < 7.0$

$$WX(t) = 3.442 \cdot t^3 - 61.690 \cdot t^2 + 361.227 \cdot t - 690.034$$

$$WY(t) = 6.205 \cdot t^3 - 124.304 \cdot t^2 + 821.525 \cdot t - 179.035$$

$$W(t) = 0.000 \cdot t^3 + 0.000 \cdot t^2 + 2.097 \cdot t + 1.563$$

Subsegment eight $7.0 \leq t < 8.0$

$$WX(t) = -6.907 \cdot t^3 - 155.644 \cdot t^2 - 1160.110 \cdot t + 2859.751$$

$$WY(t) = 0.445 \cdot t^3 - 3.337 \cdot t^2 - 25.246 \cdot t + 185.445$$

$$W(t) = 0.000 \cdot t^3 - 0.003 \cdot t^2 + 2.114 \cdot t + 1.523$$

Subsegment nine $8.0 \leq t < 9.0$

$$WX(t) = 2.535 \cdot t^3 - 70.958 \cdot t^2 + 652.700 \cdot t - 1974.409$$

$$WY(t) = -8.290 \cdot t^3 + 206.313 \cdot t^2 - 1702.444 \cdot t + 4657.971$$

$$W(t) = 0.000 \cdot t^3 + 0.011 \cdot t^2 + 2.000 \cdot t + 1.822$$

The above equations are evaluated at a given value of t. Equation 2.1.4 is then used to find the points on the rational B-spline curve.

To generate the nonrational B-spline curve, equation 2.1.1 is used with the nonhomogeneous control points CP_i and the above blending functions to yield the following equations for each subsegment:

Subsegment one $0.0 \leq t < 1.0$

$$X(t) = -0.272 \cdot t^3 + 2.857 \cdot t^2 - 3.794 \cdot t - 0.015$$

$$Y(t) = 2.397 \cdot t^3 - 4.373 \cdot t^2 - 0.715 \cdot t + 2.372$$

Subsegment two $1.0 \leq t < 2.0$

$$X(t) = -1.410 \cdot t^3 + 6.273 \cdot t^2 - 7.209 \cdot t + 1.124$$

$$Y(t) = -0.733 \cdot t^3 + 5.015 \cdot t^2 - 10.103 \cdot t + 5.502$$

Subsegment three $2.0 \leq t < 3.0$

$$X(t) = 0.807 \cdot t^3 - 7.028 \cdot t^2 + 19.392 \cdot t - 16.610$$

$$Y(t) = -0.754 \cdot t^3 + 5.145 \cdot t^2 - 10.363 \cdot t + 5.675$$

Subsegment four $3.0 \leq t < 4.0$

$$X(t) = 0.278 \cdot t^3 - 2.274 \cdot t^2 + 5.130 \cdot t - 2.348$$

$$Y(t) = 0.810 \cdot t^3 - 8.934 \cdot t^2 + 31.873 \cdot t - 36.560$$

Subsegment five $4.0 \leq t < 5.0$

$$X(t) = -0.688 \cdot t^3 + 9.328 \cdot t^2 - 41.279 \cdot t + 59.530$$

$$Y(t) = -0.117 \cdot t^3 + 2.186 \cdot t^2 - 12.607 \cdot t + 22.746$$

Subsegment six $5.0 \leq t < 6.0$

$$X(t) = 0.365 \cdot t^3 - 6.478 \cdot t^2 + 37.750 \cdot t - 72.185$$

$$Y(t) = -0.445 \cdot t^3 + 7.119 \cdot t^2 - 37.270 \cdot t + 63.851$$

Subsegment seven $6.0 \leq t < 7.0$

$$X(t) = 0.177 \cdot t^3 - 3.083 \cdot t^2 + 17.382 \cdot t - 31.449$$

$$Y(t) = 0.441 \cdot t^3 - 8.844 \cdot t^2 + 58.505 \cdot t - 127.700$$

Subsegment eight $7.0 \leq t < 8.0$

$$X(t) = -0.405 \cdot t^3 + 9.124 \cdot t^2 - 68.068 \cdot t + 167.935$$

$$Y(t) = -0.018 \cdot t^3 + 0.794 \cdot t^2 - 8.955 \cdot t + 29.708$$

Subsegment nine $8.0 \leq t < 9.0$

$$X(t) = 0.166 \cdot t^3 - 4.575 \cdot t^2 + 41.529 \cdot t - 124.324$$

$$Y(t) = -0.436 \cdot t^3 + 10.829 \cdot t^2 - 89.239 \cdot t + 243.797$$

The above equations are then used directly to generate the B-spline curve.

APPENDIX B

MATRIX FORMULATION

B. FORMULATION OF BLENDING FUNCTION MATRIX

The formulation of the blending function matrices for the subsegments of a given knot vector is discussed and an example given in this appendix. To determine which blending functions affect the ith subsegment, the recursion of equations 2.1.2 and 2.1.3 must be followed. Each non-zero blending function of a given order is used in two blending functions of the next order. The tree structure which results is shown in figure B.1. As an example, let the order of the curve equal four (k=4) and take the knot vector to be :

```
X = [ 0  0  0  0  1  2  3  4  5  6  6  6  6 ]
i =   0  1  2  3  4  5  6  7  8  9 10 11 12
```

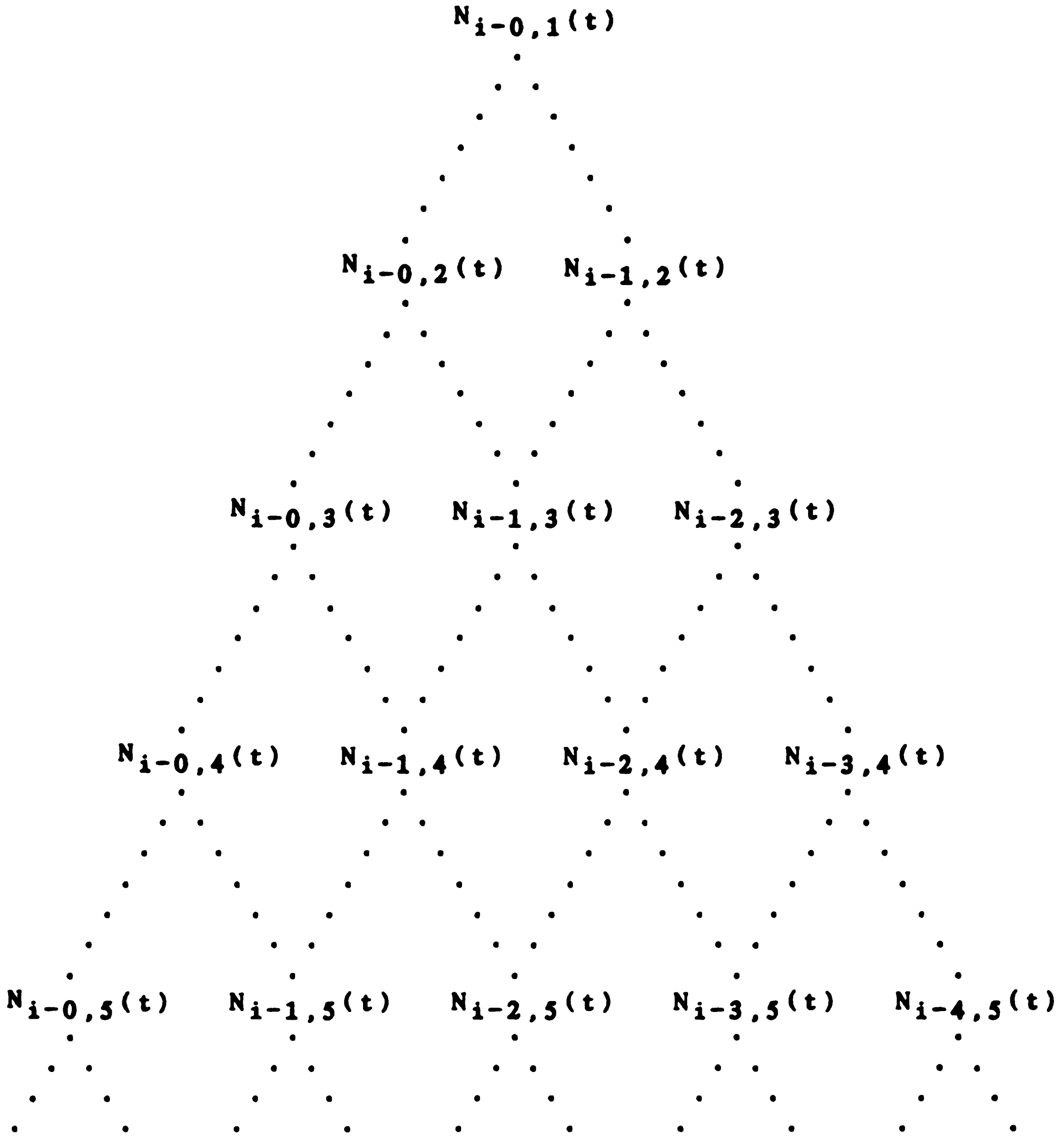
$N_{i-0,1}(t)$
$N_{i-0,2}(t)$ $N_{i-1,2}(t)$
$N_{i-0,3}(t)$ $N_{i-1,3}(t)$ $N_{i-2,3}(t)$
$N_{i-0,4}(t)$ $N_{i-1,4}(t)$ $N_{i-2,4}(t)$ $N_{i-3,4}(t)$
$N_{i-0,5}(t)$ $N_{i-1,5}(t)$ $N_{i-2,5}(t)$ $N_{i-3,5}(t)$ $N_{i-4,5}(t)$

Figure B.1 : Tree structure of non-zero blending functions for the ith subsegment

For this example the blending functions for the second subsegment will be found. Therefore i=4 and Figure B.1 becomes

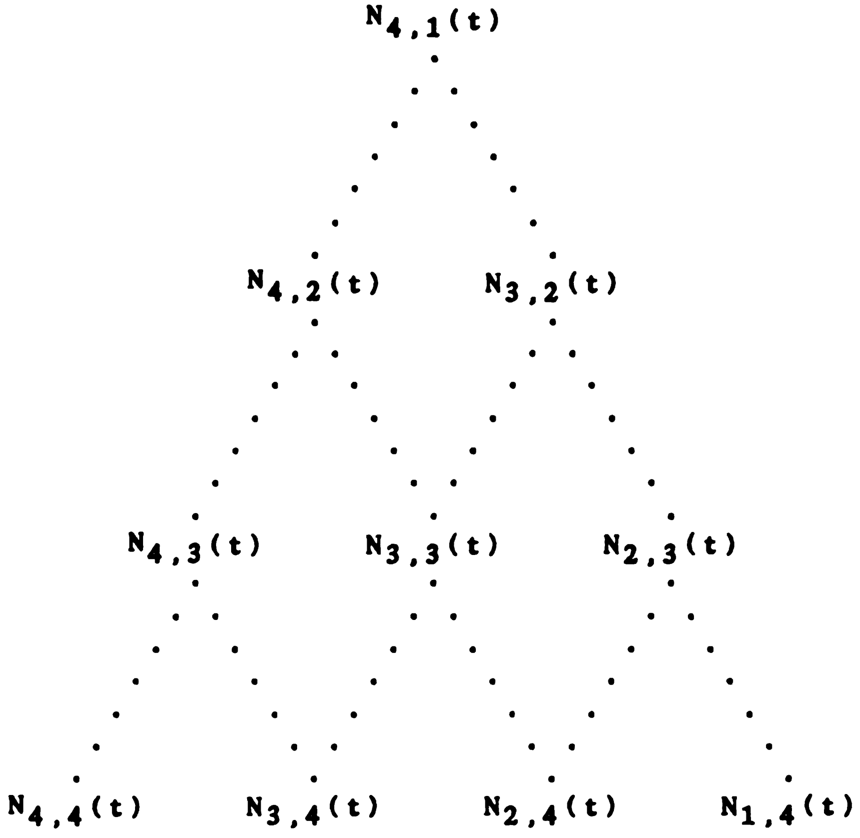

Figure B.2 : Tree structure of the non-zero blending functions for the 2nd subsegment k=4

Now that the non-zero blending function of each order have been identified, the effective portion of the knot vector can be determined. To find the starting knot of the effective knot vector, equation 2.1.3 is written for k=4 and i=1 :

$$N_{1,4}(t) = (t - X_1) \cdot N_{1,3}(t) \;/\; (X_4 - X_1) + (X_5 - t) \cdot N_{2,3}(t) \;/\; (X_5 - X_2) \qquad \text{B.1}$$

Equation B.1 requires knots X_1, X_2, X_4, and X_5. Knot one would appear to be the start of the effective knot vector. However, it is only used with $N_{1,3}$, which from Figure B.2 is always zero for this subsegment. Equation B.1 is rewritten as :

$$N_{1,4}(t) = (X_5 - t) \cdot N_{2,3}(t) \;/\; (X_5 - X_2) \qquad \text{B.2}$$

Equation B.2 requires knots X_2 and X_5; therefore, the effective knot vector starts with knot two.

To find the ending knot of the effective knot vector, Equation 2.1.2 is written for k=4 and i=4 :

$$N_{4,4}(t) = (t - X_4) \cdot N_{4,3}(t) \;/\; (X_7 - X_4) + (X_8 - t) \cdot N_{5,3}(t) \;/\; (X_8 - X_5) \qquad \text{B.3}$$

Equation B.3 requires knots X_4, X_5, X_7, and X_8. Knot eight is the last knot used in this subsegment; however, it is only used with $N_{5,3}$, which (from Figure B.2) is always zero for this subsegment. Equation B.3 is rewritten as :

$$N_{4,4}(t) = (t - X_4) \cdot N_{4,3}(t) \;/\; (X_7 - X_4) \qquad \text{B.4}$$

Equation B.4 requires knots X_4 and X_7; therefore, the effective knot vector ends at knot seven. The resulting effective knot vector for the second subsegment is then :

$$\begin{array}{cccccccc} X' = & [\;0 & 0 & 1 & 2 & 3 & 4\;] \\ i = & 2 & 3 & 4 & 5 & 6 & 7 \\ & & & \uparrow & & & \end{array}$$

Next the ten Ni,k of Figure B.2 must be found. Using equations 2.1.2 and 2.1.3 :

$$N_{4,1}(t) = 1 \qquad \text{B.5}$$

$$N_{3,2}(t) = (t-0)\cdot 0/(1-0) + (2-t)\cdot 1/(2-1) \qquad \text{B.6}$$

$$= (2-t)$$

$$N_{4,2}(t) = (t-1)\cdot 1/(2-1) + (3-t)\cdot 0/(3-2) \qquad \text{B.7}$$

$$= (t-1)$$

$$N_{2,3}(t) = (t-0)\cdot 0/(1-0) + (2-t)\cdot(2-t)/(2-0) \qquad \text{B.8}$$

$$= (t^2 - 4\cdot t + 4)/2$$

$$N_{3,3}(t) = (t-0)\cdot(2-t)/(2-0) + (3-t)\cdot(t-1)/(3-1) \qquad \text{B.9}$$

$$= (-2\cdot t^2 + 6\cdot t - 3)/2$$

$$N_{4,3}(t) = (t-1)\cdot(t-1)/(3-1) + (4-t)\cdot 0/(4-2) \qquad \text{B.10}$$

$$= (t^2 - 2\cdot t + 1)/2$$

$$N_{1,4}(t) = (t-0)\cdot 0/(1-0) + [(2-t)\cdot(t^2-4\cdot t+4)/2]/(2-0) \qquad \text{B.11}$$

$$= (-t^3 + 6\cdot t^2 - 12\cdot t + 8)/4$$

$$N_{2,4}(t) = [(t-0)\cdot(t^2-4\cdot t+4)/2]/(2-0)$$

$$+ [(3-t)\cdot(-2\cdot t^2+6\cdot t-3)/2]/(3-0) \qquad \text{B.12}$$

$$= (7\cdot t^3 - 36\cdot t^2 + 54\cdot t - 18)/12$$

$$N_{3,4}(t) = [(t-0)\cdot(-2\cdot t^2+6\cdot t-3)/2]/(3-0)$$

$$+ [(4-t)\cdot(t^2-2\cdot t+1)/2]/(4-1) \qquad \text{B.13}$$

$$= (-3\cdot t^3 + 12\cdot t^2 - 12\cdot t + 4)/6$$

$$N_{4,4}(t) = [(t-1)\cdot(t^2-2\cdot t+1)/2]/(4-1)$$

$$+ (5-t)\cdot 0/(5-2) \qquad \text{B.14}$$

$$= (t^3 - 3\cdot t^2 + 3\cdot t - 1)/6$$

Equations B.11, B.12, B.13, and B.14 are then written in matrix form as:

$$[N_{1,4}(t), N_{2,4}(t), N_{3,4}(t), N_{4,4}(t)] = 1/12 \cdot [t^3\ t^2\ t\ 1] \begin{bmatrix} -3 & 7 & -6 & 2 \\ 18 & -36 & 24 & -6 \\ -36 & 54 & -24 & 6 \\ 24 & -18 & 8 & -2 \end{bmatrix}$$

B.15

APPENDIX C

KNOT VECTOR TRANSLATION

C. EFFECTS OF KNOT VECTOR TRANSLATION ON BLENDING FUNCTIONS

Appendix B formulated the blending function matrix for the second subsegment of the fourth order knot vector :

$$\begin{array}{lllllllllllllll} X & = & [\ 0 & 0 & 0 & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 6 & 6 & 6\ ] \\ i & = & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \end{array}$$

This appendix formulates the blending function matrix for the second subsegment of the translated knot vector :

$$\begin{array}{lllllllllllllll} X' & = & [-1 & -1 & -1 & -1 & 0 & 1 & 2 & 3 & 4 & 5 & 5 & 5 & 5\ ] \\ i & = & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \end{array}$$

The effective knot vector for the second subsegment is :

$$\begin{array}{lcccccc} X'_e = & [-1 & -1 & 0 & 1 & 2 & 3\,] \\ i = & 2 & 3 & 4 & 5 & 6 & 7 \\ & & & \uparrow & & & \end{array}$$

The translation has not changed Figure B.2 and the same ten $N_{i,k}$ must be found. Using equations 2.1.2 and 2.1.3 with X' :

$N_{4,1}(t') = 1$ C.1

$N_{3,2}(t') = (t'+1)\cdot 0/(0+1) + (1-t')\cdot 1/(1-0)$ C.2

$= (1-t')$

$N_{4,2}(t') = (t'-0)\cdot 1/(1-0) + (2-t')\cdot 0/(2-1)$ C.3

$= (t'-0)$

$N_{2,3}(t') = (t'+1)\cdot 0/(0+1) + (1-t')\cdot(1-t')/(1+1)$ C.4

$= ((t')^2 - 2\cdot(t') + 1)/2$

$N_{3,3}(t') = (t'+1)\cdot(1-t')/(1+1) + (2-t')\cdot(t'-0)/(2-0)$ C.5

$= (-2\cdot(t')^2 + 2\cdot(t') + 1)/2$

$N_{4,3}(t') = (t'-0)\cdot(t'-0)/(2-0) + (3-t')\cdot 0/(3-1)$ C.6

$= (t')^2 /2$

$N_{1,4}(t') = (t'+1)\cdot 0/(0+1)$

$+ [(1-t')\cdot((t')^2-2\cdot(t')+1)/2]/(1+1)$ C.7

$$= (-(t')^3 + 3\cdot(t')^2 - 3\cdot(t') + 1)/4$$

$$N_{2,4}(t') = [(t'+1)\cdot((t')^2-2\cdot(t')+1)/2]/(1+1)$$
$$+ [(2-t')\cdot(-2\cdot(t')^2+2\cdot(t')-1)/2]/(2+1) \qquad C.8$$
$$= (7\cdot(t')^3 - 15\cdot(t')^2 + 3\cdot(t') - 7)/12$$

$$N_{3,4}(t') = [(t'+1)\cdot(-2\cdot(t')^2+2\cdot(t')-1)/2]/(2+1)$$
$$+ [(3-t')\cdot((t')^2)/2]/(3-0) \qquad C.9$$
$$= (-3\cdot(t')^3 + 3\cdot(t')^2 + 3\cdot(t') + 1)/6$$

$$N_{4,4}(t') = [(t'-0)\cdot((t')^2)/2]/(3-0)$$
$$+ (4-t')\cdot 0/(4-1) \qquad C.10$$
$$= ((t')^3)/6$$

Equations C.7, C.8, C.9, and C.10 are then written in matrix form as:

$$[N_{1,4}(t'),\ N_{2,4}(t'),\ N_{3,4}(t'),\ N_{4,4}(t')] =$$
$$1/12 \cdot [(t')^3\ (t')^2\ (t')\ 1] \cdot \begin{bmatrix} -3 & 7 & -6 & 2 \\ 9 & -15 & 6 & 0 \\ -9 & 3 & 6 & 0 \\ 3 & 7 & 2 & 0 \end{bmatrix}$$

C.11

The parametric variable t can be related to the parametric variable t' :

$$t = t' + 1 \qquad \text{C.12}$$

Squaring and cubing equation C.12 yields the following relations :

$$t^2 = (t')^2 + 2\cdot(t') + 1 \qquad \text{C.13}$$

$$t^3 = (t')^3 + 3\cdot(t')^2 + 3\cdot(t') + 1 \qquad \text{C.14}$$

Equation C.12, C.13, and C.14 are expressed in matrix form as

$$[t^3 \;\; t^2 \;\; t \;\; 1] = [(t')^3 \;\; (t')^2 \;\; (t') \;\; 1] \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 3 & 1 & 0 & 0 \\ 3 & 2 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

C.15

Replacing the t vector of equation B.15 with the right hand side of C.15 and equating the results with equation C.11 yields

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 3 & 1 & 0 & 0 \\ 3 & 2 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} -3 & 7 & -6 & 2 \\ 18 & -36 & 24 & -6 \\ -36 & 54 & -24 & 6 \\ 24 & -18 & 8 & -2 \end{bmatrix} = \begin{bmatrix} -3 & 7 & -6 & 2 \\ 9 & -15 & 6 & 0 \\ -9 & 3 & 6 & 0 \\ 3 & 7 & 2 & 0 \end{bmatrix}$$

C.16

The equality of equation C.16 holds; thus, translating the knot vector is a linear map of the resulting blending functions. The evaluated values of the $N^{i,k}$ are the same for equivalent values of t and t'.

APPENDIX D

KNOT VECTOR COMPRESSION ALGORITHM

D.0 THE COMPRESSION ALGORITHM

This appendix presents the algorithm used in the compression of the effective knot vectors to unique labels. The informal explanation preceding the algorithm specification may render the encoding transparent to the reader.

Step 1: The first and second values of the effective knot vector are compared :

if they are equal then the first bit is set to zero;

else the first bit is set to one.

Step 2: The second and third values are compared as as above to set the second bit.

Step 3: This is continued until the starting parametric value is reached.

Step 4: For a non-zero subsegment to exist, the starting parametric value and the next knot cannot be equal so there is no bit for that comparison.

Step 5: The remaining knots are compared in order and corresponding bits assigned zero or one as in step one.

Step 6: The bits are now reversed to form a binary number, which is stored as the integer label.

This algorithm could be coded as follows:

(Let KNOT(I) represent the effective knot vector, $I = 0, \ldots, 2 \cdot k-3$)

```
Set LABEL = 0
for I = 1, ..., K-2, do:
 .      IF( KNOT(I) .EQ. KNOT(I-1) ) THEN
 .      set LABEL = LABEL + 2^(I-1)
 .......END IF
for I = K, ..., 2*K-3, do:
 .      IF( KNOT(I) .EQ. KNOT(I-1) ) THEN
 .      set LABEL = LABEL + 2^(I-2)
 .......END IF
```

D.1 Example of Knot Vector Compression

Using the knot vector

$$X = [0\ 0\ 0\ 0\ 1\ 2\ 2\ 3\ 3\ 3\ 3]$$

with k=4, we have the following effective knot vectors and labels for each subsegment.

	effective		integer
<u>segment</u>	<u>knot vector</u>	<u>bits</u>	<u>label</u>
1	[0 0 0 1 2 2]	0 0 1 0	4
2	[0 0 1 2 2 3]	0 1 0 1	10
3	[1 2 2 3 3 3]	1 0 0 0	1

If the above portions of the knot vector are translated, the labels remain the same:

segment	effective knot vector	bits	integer label
1	[0 0 0 1 2 2]	0 0 1 0	4
2	[-1 -1 0 1 1 2]	0 1 0 1	10
3	[-1 0 0 1 1 1]	1 0 0 0	1

APPENDIX E

NURBS TO EURBS CONVERSION

E. EXAMPLE OF NURBS TO EURBS CONVERSION

This appendix presents the control points and knot vectors used to generate the figures in Chapter five of this dissertation. The order of each curve in these figures is four.

Figure 5.1.1 compares a nonuniform curve to a uniform curve. The two curves use the control points given in Table E.1.

Control Point	X	Y
0	0.0000000E+00	0.0000000E+00
1	0.5000000E+00	0.1000000E+01
2	0.1000000E+01	0.5000000E+00
3	0.1500000E+01	0.2500000E+00
4	0.2000000E+01	0.5000000E+00
5	0.2500000E+01	0.1000000E+01
6	0.3000000E+01	0.7500000E+00

Table E.1 : Original Control Points

The nonuniform knot vector for the nonuniform curve is :

$$X_n = [\ 0.0\ 0.0\ 0.0\ 0.0\ 0.3\ 0.75\ 1.0\ 1.66\ 1.66\ 1.66\ 1.66\ ]$$

The uniform knot vector is then found to be :

$$X_u = [\ 0\ 0\ 0\ 0\ 1\ 2\ 3\ 4\ 4\ 4\ 4\ ]$$

The uniform B-spline of Figure 5.1.2 was generated by using the above knot vector X_u and the control points in Table E.2

Control Point	X	Y
0	0.0000000E+00	0.0000000E+00
1	0.4999999E+00	0.9999998E+00
2	0.7979605E+00	0.6086380E+00
3	0.1737129E+01	0.2697527E+00
4	0.1623749E+01	0.2653303E+00
5	0.2500038E+01	0.1000007E+01
6	0.3000000E+01	0.7500000E+00

Table E.2 : Control Points for Figure 5.1.2

The control points in Table E.2 were found by requiring the uniform B-spline to match the slopes at t=0 and t=4, and also to pass through the nonuniform B-spline curve at the ends of each subsegment.

The uniform curve of Figure 5.1.3 was generated by using the above knot vector X_u and the control points in Table E.3

Control Point	X	Y
0	0.0000000E+00	0.0000000E+00
1	0.4999999E+00	0.9999998E+00
2	0.8999994E+00	0.5999985E+00
3	0.1379994E+01	0.2999918E+00
4	0.2950251E+01	0.1530137E+00
5	-0.3570400E+00	0.1241920E+01
6	0.2999855E+01	0.7499551E+00

Table E.3 : Control Points for Figure 5.1.3

The control points in Table E.3 were found by requiring the uniform B-spline to match the first subsegment of the nonuniform B-spline and also to pass through the ends of the subsegments.

The uniform curve of Figure 5.1.4 was generated by using the above knot vector X_u and the control points in Table E.4

Control Point	X	Y
0	-0.7910313E-02	-0.2146652E-01
1	0.5274833E+00	0.1050406E+01
2	0.8117139E+00	0.5352641E+00
3	0.1712133E+01	0.2676190E+00
4	0.1676200E+01	0.3606113E+00
5	0.2516168E+01	0.9357985E+00
6	0.2998456E+01	0.7769325E+00

Table E.4 : Control Points for Figure 5.1.4

To find the control points in Table E.4, one hundred discrete points were used from each subsegment of the nonuniform curve as the data to which the uniform curve was fit by ordinary least squares.

The NURBS to EURBS algorithm was used on the the knot vector X_n and the control points in Table E.1. The resulting knot vector for the uniform curve in Figure 5.2.1

was found to be :

$$X_u = [\ 0\ 0\ 0\ 0\ 1\ 1\ 1\ 2\ 2\ 2\ 3\ 3\ 3\ 4\ 4\ 4\ 4\ ]$$

The control points which result from using the above X_u knot vector to fit the curve defined by X_n and the control points of Table E.1 are given in Table E.5

Control Point	X	Y
0	0.0000000E+00	0.0000000E+00
1	0.4999999E+00	0.9999998E+00
2	0.6999992E+00	0.7999988E+00
3	0.8799984E+00	0.6499976E+00
4	0.8799983E+00	0.6499990E+00
5	0.1149997E+01	0.4249983E+00
6	0.1374994E+01	0.3124974E+00
7	0.1561703E+01	0.3254953E+00
8	0.1561675E+01	0.3254890E+00
9	0.1665399E+01	0.3327094E+00
10	0.1757320E+01	0.3786680E+00
11	0.1861713E+01	0.4497342E+00
12	0.1861710E+01	0.4497327E+00
13	0.2137309E+01	0.6373476E+00
14	0.2499873E+01	0.9999626E+00
15	0.2999841E+01	0.7499528E+00

Table E.5 : Control Points for Figure 5.1.5

The control points of Table E.5 and the above knot vector X_u were used to generate the uniform curve in Figure 5.1.5. This curve is identical to the nonuniform curve.

APPENDIX F

CONVERSION ALGORITHMS

F. CODE FOR THE NURBS TO EURBS CONVERSION

This appendix presents the code for three FORTRAN 77 subroutines used in the NURBS to EURBS conversion. These subroutines assume that the nonuniform surface definition has been read in. The first routine handles the switching of control points and is used to call the other two routines. The second routine, LABEL, is used to label the non-zero subsegments and determine where new control points are to be added. The last routine, FITSU, is used to determine the EURBS control points.

```
C NONTOU: NONuniform TO Uniform rational B-spline converter
CDCCCCC            Subroutine Description                CCCC
C
C Purpose and use: Convert NURBS to EURBS
C
C Error conditions:
C
C Implementation:
C
C Notes:
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
CCCCCCCCCCCCCCC          Statistics          CCCCCCCCCCCCCCCC
C
C Author: Mark N. Pickelmann
C
C Date written: April 30, 1984
C
C Modifications:
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
CSCCCCCCC            Entry _Storage Block               CCCCC
C
      SUBROUTINE NONTOU(NPNTS,NPNTT, ORDERS, ORDERT, RATNOT
     1                 ,SKNOTR, TKNOTR, CPW, CPX, CPY, CPZ)
C
C===Read-only arguments:
C --- Number of points and order of S and T curves
      INTEGER NPNTS, NPNTT, ORDERS, ORDERT
C --- Rational flag
      LOGICAL RATNOT
C --- Nonuniform knot vectors
      REAL SKNOTR(0:NPNTS+2*(ORDERS-1))
      REAL TKNOTR(0:NPNTT+2*(ORDERT-1))
C
C===Arguments returned:
C    none
C
C===Arguments modified:
C --- Maximum number of points
      INTEGER MAXPNT, MAXSUB
      PARAMETER (MAXPNT = 1000)
      PARAMETER (MAXSUB = 1000)
C
C --- Control points
      REAL CPW(0:MAXPNT)
      REAL CPX(0:MAXPNT)
      REAL CPY(0:MAXPNT)
      REAL CPZ(0:MAXPNT)
C
C
```

```
C===Common blocks:
      COMMON/CMN01/PW
      COMMON/CMN02/PX
      COMMON/CMN03/PY
      COMMON/CMN04/PZ
C
C===Local variables:
C --- Control points
      REAL PW(0:MAXPNT)
      REAL PX(0:MAXPNT)
      REAL PY(0:MAXPNT)
      REAL PZ(0:MAXPNT)
C --- Subsegment labels
      INTEGER LABELS(MAXSUB), LABELT(MAXSUB)
C --- Number of new points for each segment
      INTEGER EXTEND(MAXSUB)
C --- Number of new points S and T
      INTEGER NEWS, NEWT
C --- Non-zero segments S and T
      INTEGER SEGS, SEGT
C --- Counters
      INTEGER ROW, COL
C
C===Functions and subroutines:
      EXTERNAL LABEL, FITSU
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
CPCCCCCCCC           Process Block                 CCCCCC
C
C --- Determine the labels for the s direction
C
      CALL LABEL( NPNTS, ORDERS, SKNOTR, LABELS, EXTEND,
     1            NEWS, SEGS)
C
C --- Refit in the s direction
C
        CALL FITSU(CPW, CPX, CPY, CPZ, RATNOT,
     1             ORDERS, NPNTS, NPNTT, SKNOTR, LABELS,
     2             EXTEND, NEWS, SEGS, PW, PX, PY, PZ )
C
C --- Determine the labels for the t direction
C
      CALL LABEL( NPNTT, ORDERT, TKNOTR, LABELT, EXTEND,
     1            NEWT, SEGT )
C
C --- Refit in the t direction
C
C --- Transpose t and s
C
      DO 100 ROW = 0, NPNTT
        DO 100 COL = 0, NPNTS+NEWS
         CPW(ROW+COL*(NPNTT+1)) =
 PW(COL+ROW*(NPNTS+NEWS+1))
        CPX(ROW+COL*(NPNTT+1)) =
 PX(COL+ROW*(NPNTS+NEWS+1))
        CPY(ROW+COL*(NPNTT+1)) =
 PY(COL+ROW*(NPNTS+NEWS+1))
        CPZ(ROW+COL*(NPNTT+1)) =
 PZ(COL+ROW*(NPNTS+NEWS+1))
100   CONTINUE
C
C
      CALL FITSU(CPW, CPX, CPY, CPZ, RATNOT, ORDERT,
     1           NPNTT, NPNTS+NEWS, TKNOTR, LABELT, EXTEND,
     2           NEWT, SEGT, PW, PX, PY, PZ )
C
C --- Transpose back
C
      DO 200 ROW = 0, NPNTS+NEWS
        DO 200 COL = 0, NPNTT+NEWT
          CPW(ROW+COL*(NPNTS+NEWS+1)) =
     1    PW(COL+ROW*(NPNTT+NEWT+1))
          CPX(ROW+COL*(NPNTS+NEWS+1)) =
     1    PW(COL+ROW*(NPNTT+NEWT+1))
          CPY(ROW+COL*(NPNTS+NEWS+1)) =
     1    PY(COL+ROW*(NPNTT+NEWT+1))
          CPZ(ROW+COL*(NPNTS+NEWS+1)) =
     1    PZ(COL+ROW*(NPNTT+NEWT+1))
200   CONTINUE
C
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
      RETURN
      END
```

```
C LABEL: Labels subsegments for an enhanced knot vector
CDCCCCC            Subroutine Description              CCCC
C
C Purpose and use: Determine subsegment labels and where
C                  new control points must go.
C
C Error conditions:
C
C Implementation:
C
C Notes:
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
CCCCCCCCCCCCCCC          Statistics          CCCCCCCCCCCCCCCC
C
C Author: Mark N. Pickelmann
C
C Date written: April 30, 1984
C
C Modifications:
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
CSCCCCCCC           Entry _Storage Block             CCCCCC
C
      SUBROUTINE LABEL ( NPNT, ORDER, RKNOT, LAB, EXTEND,
                         NEW, SEG)
C
C===Read-only arguments:
C --- Number of points and order of the curve
      INTEGER NPNT, ORDER
C --- Nonuniform knot vector
      REAL RKNOT(0:NPNT+ORDER)
C
C===Arguments returned:
C --- Subsegment labels
      INTEGER LAB(NPNT-ORDER+2)
C --- If the knot vector was extended for this segment
      INTEGER EXTEND(NPNT-ORDER+2)
C --- Number of new knots
      INTEGER NEW
C --- Number of non-zero segments
      INTEGER SEG
C
C===Arguments modified:
C    none
C
C===Common blocks:
C    none
C
C===Local variables:
C --- Pointers and counters
```

```
      INTEGER PNTER, I
C --- Uniform knot vector
      REAL KNOT(0:40)
C --- A number like zero
      REAL ZERO
C --- Knot spacing
      REAL DELTA
C
C===Functions and subroutines:
      INTRINSIC ABS
C
C
      DATA ZERO/1.0E-06/
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
CPCCCCCCCC            Process Block                  CCCCCC
C
C --- Copy the knot vector
C
      DO 100 I=0,2*ORDER-3
        KNOT(I) = RKNOT(I)
100   CONTINUE
C
      NEW = 0
      SEG = 0
      PNTER = ORDER - 2
C
C --- Shift the knot vector
C
200   PNTER = PNTER + 1
        DO 250 I=0,2*ORDER-4
          KNOT(I) = KNOT(I+1)
250     CONTINUE
        KNOT(2*ORDER-3) = RKNOT(PNTER+ORDER-1)
C
C --- Check for repeat knot
C
      IF(PNTER.GT.NPNT) THEN
        GOTO 1000
        ELSE IF(KNOT(ORDER-1)-KNOT(ORDER-2).GE.ZERO) THEN
        SEG = SEG + 1
        DELTA = KNOT(ORDER-1)-KNOT(ORDER-2)
        ELSE
        GOTO 200
      END IF
C
C --- See if we need to extend the old knot vector
C
      EXTEND(SEG) = 0
      DO 300 I=1,2*ORDER-3
        IF(ABS(KNOT(I)-KNOT(I-1)-DELTA).GT.ZERO.AND.
     1     KNOT(I)-KNOT(I-1).GT.ZERO) THEN
```

```
            EXTEND(SEG) = EXTEND(SEG) + 1
            KNOT(I+1) = KNOT(I)
            KNOT(I) = KNOT(I-1)
          PNTER = PNTER - 1
        END IF
300     CONTINUE
C
C --- Label this segment
C
        LAB(SEG) = 0
        DO 400 I = 2*ORDER-4,0,-1
          IF(I.EQ.ORDER-2) GOTO 400
          IF(KNOT(I+1)-KNOT(I).GT.ZERO) THEN
            LAB(SEG) = 2*LAB(SEG) + 1
            ELSE
            LAB(SEG) = 2*LAB(SEG)
          END IF
400     CONTINUE
        NEW = NEW + EXTEND(SEG)
        GOTO 200
C
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
1000    RETURN
        END
```

```
C FITSU: Fit extended uniform rational B-splines in the S
C        direction
CDCCCCC             Subroutine Description               CCCC
C
C Purpose and use: Find the EURBS control points for the
C                  given NURBS control points
C
C Error conditions: Notifies if jump discontinuity is
 encountered
C
C Implementation:
C
C Notes:
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
CCCCCCCCCCCCCCC          Statistics          CCCCCCCCCCCCCCCC
C
C Author: Mark N. Pickelmann
C
C Date written: May 2, 1984
C
C Modifications:
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
CSCCCCCCC            Entry _Storage Block              CCCCCC
C
      SUBROUTINE FITSU(CPW, CPX, CPY, CPZ, RATNOT, ORDER
     1                NPNTS, NPNTT, RKNOT, LAB, EXTEND,
     2                NEW, NSEG, PW, PX, PY, PZ )
C===Read-only arguments:
C --- Rational or not
      LOGICAL RATNOT
C --- Order of the curve to be fit
      INTEGER ORDER
C --- Number of control points
      INTEGER NPNTS, NPNTT
C --- Number of new control points
      INTEGER NEW
C --- Knot vector
      REAL RKNOT(0:NPNTS+ORDER)
C --- Control points
      REAL CPW(0:NPNTS, 0:NPNTT)
      REAL CPX(0:NPNTS, 0:NPNTT)
      REAL CPY(0:NPNTS, 0:NPNTT)
      REAL CPZ(0:NPNTS, 0:NPNTT)
C --- Number of non-zero segments
      INTEGER NSEG
C --- Subsegment labels
      INTEGER LAB(NSEG)
C --- List of extentions and total number of extentions
      INTEGER EXTEND(NSEG)
```

```
C
C
C===Arguments returned:
C --- Uniform control points
      REAL PW(0:NPNTS+NEW,0:NPNTT)
      REAL PX(0:NPNTS+NEW,0:NPNTT)
      REAL PY(0:NPNTS+NEW,0:NPNTT)
      REAL PZ(0:NPNTS+NEW,0:NPNTT)
C
C===Arguments modified:
C    none
C
C===Common blocks:
C       none
C
C===Local variables:
C --- Counters and pointers
      INTEGER ROW, ROW2, COL, SEG, I, J1, J2, TERM, PIV,
     1        PIVOT
      REAL MAX, HOLD
C --- Uniform pointer and last pointer
      INTEGER UPNT, LPNT
C --- A number like zero
      REAL ZERO
C --- Number of points to determine for a given segment
      INTEGER DETR
C --- Equation matrix
      REAL A(20,20)
C --- Uniform blending function matrix
      REAL MAT(400)
C --- Nonuniform matrix
      REAL N(400)
C --- Scale factor
      REAL SF
C
C===Functions and subroutines:
      EXTERNAL CDALGO, CDANS, BASMAT
      INTRINSIC ABS
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
CPCCCCCCCC           Process Block                    CCCCCC
C
C --- Check to see if need to use the weights
C
      IF(RATNOT) THEN
        DO 50 COL = 0,NPNTT
          DO 50 ROW = 0, NPNTS
            CPX(ROW,COL) = CPX(ROW,COL) * CPW(ROW,COL)
            CPY(ROW,COL) = CPY(ROW,COL) * CPW(ROW,COL)
            CPZ(ROW,COL) = CPZ(ROW,COL) * CPW(ROW,COL)
50      CONTINUE
      END IF
```

```
C
      ZERO = 1.0E-06
      SEG = 0
      LPNT = -1
      I = ORDER - 2
      UPNT = I
C --- Find the next non-zero segment
100   IF(I.GT.NPNTS) THEN
        GOTO 1000
      ELSE IF(RKNOT(I+1)-RKNOT(I).LE.ZERO) THEN
        I = I + 1
        UPNT = UPNT + 1
        GOTO 100
      END IF
C
C --- Fit this segment
C
      SEG = SEG + 1
      IF(SEG.NE.1) THEN
        UPNT = UPNT + EXTEND(SEG-1)
      END IF
C --- Find the number of control points to determine
      DETR = UPNT - LPNT
      IF(DETR.GE.ORDER.AND.SEG.NE.1) THEN
        PRINT *,'FITSU : Jump discon'
        DETR = ORDER-1
        UPNT = LPNT + ORDER - 1
      END IF
      LPNT = UPNT
C
C --- Evaluate the nonuniform curve at the end
C
      CALL CDALGO(ORDER, RKNOT(I+2-ORDER),RKNOT(I+1),N)
      ROW = UPNT+1-DETR
      DO 200 COL = 0,NPNTT
        IF(RATNOT) THEN
          CALL CDANS(ORDER, RKNOT(I+2-ORDER), RKNOT(I+1),
     1      N, CPW((I+1-ORDER),COL), PW(ROW,COL), DETR-1)
          ELSE
          DO 225 J1 = ROW, UPNT
            PW(J1,COL) = 1
225       CONTINUE
        END IF
        CALL CDANS(ORDER, RKNOT(I+2-ORDER), RKNOT(I+1), N,
     1          CPX((I+1-ORDER),COL), PX(ROW,COL), DETR-1)
        CALL CDANS(ORDER, RKNOT(I+2-ORDER), RKNOT(I+1), N,
     1          CPY((I+1-ORDER),COL), PY(ROW,COL), DETR-1)
        CALL CDANS(ORDER, RKNOT(I+2-ORDER), RKNOT(I+1), N,
     1          CPZ((I+1-ORDER),COL), PZ(ROW,COL), DETR-1)
200   CONTINUE
C
C --- Scale results to extended knot vector
C
```

```
      SF = RKNOT(I+1) - RKNOT(I)
      DO 300 J1 = UPNT+2-DETR, UPNT
        DO 300 J2 = J1, UPNT
          DO 300 COL = 0,NPNTT
            IF(RATNOT) PW(J2,COL) = SF * PW(J2,COL)
            PX(J2,COL) = SF * PX(J2,COL)
            PY(J2,COL) = SF * PY(J2,COL)
            PZ(J2,COL) = SF * PZ(J2,COL)
300   CONTINUE
C
C --- Get the uniform blending function matrix
C
      CALL BASMAT(ORDER, LAB(SEG), MAT)
C
C --- Evaluate functions and derivatives
C
      DO 400 ROW = 1, DETR
        IF(ROW.NE.1) THEN
          DO 450 J1 = 1, ORDER
            DO 450 J2 = 1, ORDER-ROW
              MAT(J2+(J1-1)*ORDER) =
              MAT(J2+(J1-1)*ORDER)*(ORDER-ROW+2-J2)
450       CONTINUE
        END IF
C
C --- Make up the A matrix
C
        DO 475 COL = 1, ORDER
          A(ROW,COL) = 0
          DO 475 TERM = 1, ORDER+1-ROW
            A(ROW,COL) = A(ROW,COL)+MAT(TERM+(COL-1)*ORDER)
475     CONTINUE
400   CONTINUE
C
C --- Fill in the known information
C
      DO 500 J1 = 1, ORDER-DETR
        DO 500 J2 = UPNT+1-DETR, UPNT
        ROW = UPNT-ORDER+J1
          DO 500 COL = 0, NPNTT
            IF(RATNOT) PW(J2,COL) = PW(J2,COL) - A(J2,J1)*
     1                 PW(ROW,COL)
            PX(J2,COL) = PX(J2,COL) - A(J2,J1)*
     1                         PX(ROW,COL)
            PY(J2,COL) = PY(J2,COL) - A(J2,J1)*
     1                         PY(ROW,COL)
            PZ(J2,COL) = PZ(J2,COL) - A(J2,J1)*
     1                         PZ(ROW,COL)
500   CONTINUE
C
C --- Shift over
C
      IF(DETR.LT.ORDER) THEN
```

```
        DO 600 J1=1,DETR
          DO 600 J2=1,DETR
            A(J2,J1) = A(J2,J1+ORDER-DETR)
600     CONTINUE
      END IF
C
C --- Do some row reduction
C
      DO 700 PIV = 1, DETR-1
        MAX = 0
        PIVOT = PIV
        DO 750 J1 = PIV, DETR
          IF(ABS(A(J1,PIV)).GT.MAX) THEN
            MAX = ABS(A(J1,PIV))
            PIVOT = J1
          END IF
750     CONTINUE
C
        IF(PIVOT.NE.PIV) THEN
          DO 775 J1=PIV,DETR
            HOLD = A(PIV,J1)
            A(PIV,J1) = A(PIVOT,J1)
            A(PIVOT,J1) = HOLD
775       CONTINUE
        ROW = PIV+UPNT-DETR
        ROW2 = PIVOT+UPNT-DETR
          DO 780 COL = 0,NPNTT
            IF(RATNOT) THEN
              HOLD = PW(ROW,COL)
              PW(ROW,COL) = PW(ROW2,COL)
              PW(ROW2,COL) = HOLD
            END IF
            HOLD = PX(ROW,COL)
            PX(ROW,COL) = PX(ROW2,COL)
            PX(ROW2,COL) = HOLD
            HOLD = PY(ROW,COL)
            PY(ROW,COL) = PY(ROW2,COL)
            PY(ROW2,COL) = HOLD
            HOLD = PZ(ROW,COL)
            PZ(ROW,COL) = PZ(ROW2,COL)
            PZ(ROW2,COL) = HOLD
780       CONTINUE
        END IF
C
        DO 790 J1 = PIV+1, DETR
          DO 785 J2 = DETR, PIV, -1
            IF(J2.EQ.PIV) THEN
              A(J1,J2) = 0
              ELSE
              A(J1,J2) = A(J1,J2) -
     1        A(PIV,J2)*A(J1,PIV)/A(PIV,PIV)
            END IF
785       CONTINUE
```

```
          ROW = J1+UPNT-DETR
          ROW2 = PIV+UPNT-DETR
          DO 786 COL=0,NPNTT
            IF(RATNOT) PW(ROW,COL) = PW(ROW,COL) -
     1                A(J1,PIV)*PW(ROW2,COL)/A(PIV,PIV)
            PX(ROW,COL) = PX(ROW,COL) -
     1      A(J1,PIV)*PX(ROW2,COL)/A(PIV,PIV)
            PY(ROW,COL) = PY(ROW,COL) -
     1      A(J1,PIV)*PY(ROW2,COL)/A(PIV,PIV)
            PZ(ROW,COL) = PZ(ROW,COL) -
     1      A(J1,PIV)*PZ(ROW2,COL)/A(PIV,PIV)
786       CONTINUE
790     CONTINUE
700   CONTINUE
C
C --- Solve by back substitution
C
      DO 800 PIV = DETR, 1, -1
        ROW = UPNT+PIV-DETR
        DO 800 COL = 0, NPNTT
          DO 850, J1=DETR, PIV+1, -1
          IF(RATNOT) PW(ROW,COL) = PW(ROW,COL) -
     1      A(PIV,J1) * PW(UPNT+J1-DETR,COL)
        PX(ROW,COL) = PX(ROW,COL) - A(PIV,J1)*
     1                          PX(UPNT+J1-DETR,COL)
        PY(ROW,COL) = PY(ROW,COL) - A(PIV,J1)*
     1                          PY(UPNT+J1-DETR,COL)
        PZ(ROW,COL) = PZ(ROW,COL) - A(PIV,J1)*
     1                          PZ(UPNT+J1-DETR,COL)
850   CONTINUE
        IF(RATNOT) PW(ROW,COL) = PW(ROW,COL) /
     1              A(PIV,PIV)
         PX(ROW,COL) = PX(ROW,COL) / A(PIV,PIV)
         PY(ROW,COL) = PY(ROW,COL) / A(PIV,PIV)
         PZ(ROW,COL) = PZ(ROW,COL) / A(PIV,PIV)
800   CONTINUE
C
C
C --- Move to the next segment
      I = I + 1
      UPNT = UPNT + 1
      GOTO 100
C
C --- Take out the weights
C
1000  IF(RATNOT) THEN
        DO 1100 COL = 0,NPNTT
          DO 1100 ROW = 0, NPNTS
            PX(ROW,COL) = PX(ROW,COL) / PW(ROW,COL)
            PY(ROW,COL) = PY(ROW,COL) / PW(ROW,COL)
            PZ(ROW,COL) = PZ(ROW,COL) / PW(ROW,COL)
1100    CONTINUE
        DO 150 COL = 0,NPNTT
```

```
          DO 150 ROW = 0, NPNTS
            CPX(ROW,COL) = CPX(ROW,COL) / CPW(ROW,COL)
            CPY(ROW,COL) = CPY(ROW,COL) / CPW(ROW,COL)
            CPZ(ROW,COL) = CPZ(ROW,COL) / CPW(ROW,COL)
150       CONTINUE
      END IF
C
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
      RETURN
      END
```

LIST OF REFERENCES

[1] Coviak, R. A., "Color Graphics in Engineering Design", M.S. Thesis, Michigan State University, 1981.

[2] Blinn, J.F., "Computer Display of Curved Surfaces", Ph.D. Dissertation, University of Utah, 1978.

[3] Vanderploeg, M. J., "Surface Assessment Using Color Graphics", Ph.D. Dissertation, Michigan State University, 1982.

[4] Lane, J. M., Carpenter, L. C., Whitted, T., and Blinn, J. F., "Scan Line Methods For Displaying Parametrically Defined Surfaces", Communications of the A.C.M., Vol. 23, No. 1, pp23-34, Jan. 1980.

[5] Lane, J. M., Carpenter, L. C., "A Generalized Scan Line Algorithm for Computer Display of Parametrically Defined Surfaces", Computer Graphics and Image Processing, Vol. 11, pp290-297, 1979.

[6] Barnhill, R. E., "A Survey of Representation and Design of Surfaces", IEEE Computer Graphics and Applications, Vol. 3, Number 7, October 1983.

[7] Forrest, A. R., On Coons and Other Methods for the Representation of Curved Surfaces", Computer Graphics and Image Processing, 1972.

[8] Versprille, K. J., "Computer Aided Design Applications of the Rational B-Spline Approximation Form", Ph.D. Dissertation, Syracuse University, 1975

[9] Schoenberg, I. J., "Contribution to the Problem of Approximation of Equidistant Data by Analytic Functions", Quart. Appl. Math. 4 (1946), 45-99.

[10] deBoor, C., "On Calculation with B-Splines", J. Approx. Theory, Vol. 6, pp 50-62, 1972.

[11] Cox, M. G., "The Numerical Evaluation of B-splines", J. Inst. Maths. Applics, Vol. 10, pp. 134-147, 1972.

[12] Riesenfeld, R. F., "Applications of B-Spline Approximation to Geometric Problems of Computer Aided Design ", Ph.D. Dissertation, Syracuse University 1973

[13] Rogers, D. F. and Adams, J. A., Mathematical Elements for Computer Graphics, McGraw-Hill Book Company, New York, New York 1976.

[14] deBoor, C., A Practical Guide to Splines, Springer-Verlag New York Inc., New York, New York, 1978.

[15] Aho, A. V., Hopcroft, J. E., and Ullman, J. D., Data Structures and Algorithms, Addison-Wesley Publishing Co., Reading Massachusetts, 1983.

[16] Beck, J. V. and Arnold, K. J., Parameter Estimation in Engineering and Science, John Wiley and Sons, New York, New York, 1977.

[17] Stewart, G. W., Introduction to Matrix Computations, Academic Press, New York, 1973.

[18] Conte, S. D. and deBoor, C., Elementary Numerical Analysis an Algorithmic Approach, McGraw - Hill Book Company, New York, New York, Third Edition , 1980.