CONVERSION BETWEEN RATIONAL B-SPLINE AND PARAMETRIC SPLINE REPRESENTATIONS

January 7, 1985

Prepared by:______________________________
Richard D. Fuhr
Boeing Commercial Airplane Company
PO Box 3707 M/S 6E-30
Seattle, WA 98124

THE BOEING COMPANY (BOEING) MAKES NO REPRESENTATION, WARRANTY, OR GUARANTEE WHATSOEVER IN CONNECTION WITH THIS PROGRAM AND RELATED MATERIALS (COLLECTIVELY, THE SOFTWARE), AND THE SOFTWARE IS BEING MADE AVAILABLE TO ALL RECIPIENTS AND/OR USERS WITH THE UNDERSTANDING THAT SUCH USERS WAIVE, RELEASE AND RENOUNCE ALL WARRANTIES, OBLIGATIONS AND LIABILITIES OF BOEING AND ANY POSSIBLE RIGHTS, CLAIMS AND REMEDIES OF THE USER AGAINST BOEING EXPRESS OR IMPLIED, ARISING BY LAW OR OTHERWISE, WITH RESPECT TO ANY DEFECT IN OR USE OF THE SOFTWARE INCLUDING, BUT NOT LIMITED TO, (A) ANY IMPLIED WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE, (B) ANY IMPLIED WARRANTY ARISING FROM COURSE OF PERFORMANCE, COURSE OF DEALING OR USAGE OF TRADE, (C) ANY OBLIGATION, LIABILITY, RIGHT, CLAIM OR REMEDY ARISING IN TORT, WHETHER OR NOT ARISING FROM BOEING'S NEGLIGENCE, ACTUAL OR IMPUTED, AND (D) ANY OBLIGATION, LIABILITY, RIGHT, CLAIM OR REMEDY FOR LOSS OF USE, REVENUE OR PROFIT, OR FOR ANY OTHER DIRECT, INCIDENTAL OR CONSEQUENTIAL DAMAGES.

TABLE OF CONTENTS

1.0 INTRODUCTION

IGES (reference [4]) provides two different forms for representing spline curve and surface entities. One form is the “parametric spline,” with entity type number 112 for curves and 114 for surfaces. The other form is the “rational B-spline,” with entity type number 126 for curves and 128 for surfaces. The Pascal code we have provided enables us to convert between these two forms. In this paper, we give an overview of the underlying algorithms, together with some examples.

Although IGES uses the term “parametric spline,” this is somewhat of a misnomer. The rational B-spline entities are also parametric splines, so a better name for IGES entities 112 and 114 would be “power basis” curve and surface spline entities. In reference [2], deBoor uses the term “piecewise polynomial” for such entities. In this paper we shall use the three terms “parametric spline,” “piecewise polynomial” and “power basis spline” interchangeably.

The core of the code is, of course, the set of conversion algorithms. To support these, we have provided the necessary mathematical and vector procedures. In addition, we have supplied procedures to convert between rational forms and the corresponding homogeneous forms.

To supplement the code necessary for the actual conversion, we have also provided procedures to run test cases and determine their accuracy. These include input/output and comparison procedures as well as drivers for the whole process. A list of all the Pascal procedures organized by category is provided in section 7 below. More detailed documentation appears in the code itself.

2.0 SCOPE

The spline data forms that we use are modifications of the corresponding IGES entities in the following ways:

A. Degree. The IGES parametric spline entities have their degree fixed at 3, but the degree in these data structures is arbitrary (up to a limit of 12). The IGES rational B-spline entities are already of arbitrary degree.

B. Dimension. All the present IGES spline entities have coefficients with dimension fixed at 3, but the dimension in these data structures is arbitrary (up to a limit of 6).

C. Rational. The IGES parametric spline entities are not rational, but the IGES rational B-spline entities, of course, are. Therefore, we have extended the parametric spline entity data structures here to make them rational, too.

D. Breakpoints and Knots. The IGES parametric spline entities provide for arbitrarily many breakpoints, and the IGES rational B-spline entities provide for arbitrarily many knots. However, in our implementation of the conversion algorithms we have had to set on upper limit an the number of breakpoints and knots. The user can increase or decrease these limits as he sees fit. Alternatively, he can reformulate the data structures using dynamic variables.

3.0 EXAMPLES

Before going into details about the algorithms, we present several examples to establish the notation and to illustrate the Pascal data structures.

Example A. <u>Quarter Circle</u>

<u>Input</u> (in rational B-Spline form)

degree = 2
dim = 2
number_of_knots = 6
knot(1) = knot(2) = knot(3) = 0
knot(4) = knot(5) = knot(6) = 1

coeff(1) = (1,0)	weight(1) = 1
coeff(2) = (1,1)	weight(2) = 1
coeff(3) = (0,1)	weight(3) = 2

This means that the quarter circle is parameterized by

$$C(t) = \frac{1*(1,0)*b_1(t) + 1*(1,1)*b_2(t) + 2*(0,1)*b_3(t)}{1*b_1(t) + 1*b_2(t) + 2*b_3(t)}$$

where b_1, b_2 and b_3 are the quadratic B-spline basis functions defined on the knot sequence 0,0,0,1,1,1. Thus, the parameter range is $0 \leq t \leq 1$, and this is a one-span B-spline curve.

<u>Output</u> (in parametric spline form)
The equivalent parametric spline form representation to the above rational B-spline curve is

$$C(t) = \left\langle \frac{1-t^2}{1+t^2}, \frac{2t}{1+t^2} \right\rangle$$

The corresponding Pascal record is as follows:

degree = 2
dim = 2
num_segs = 1
break_point(1) = 0
break_point(2) = 1

rpp_segment(1) is a rational polynomial whose coefficients are:

top_coeff(0)	= (1, 0)	(the const terms for x and y)
top_coeff(1)	= (0, 2)	(the coeffs of t^1 for x and y)
top_coeff(2)	= (-1, 0)	(the coeffs of t^2 for x and y)

bot_coeff(0)	= 1	(the const term in denominator)
bot_coeff(1)	= 0	(the coeff of t^1 in denominator)
bot_coeff(2)	= 1	(the coeff of t^2 in denominator)

Example B. Six-Segment - Spline Curve

In this example, we have a six-segment non-rational parametric spline curve which converts to a B-spline curve in which all possible knot multiplicity values occur.

Input: (See Fig. 1)
degree = 3
dim = 2
break points = -2, -1, 0, 1, 2, 3, 4
global parameter is u, $-2 \le u \le 4$
local parameter is s, defined for each segment

Segment 1:	$-2 \le u < -1$,	$s = u-(-2)$
	$x(s) = -2 + s$,	$y(s) = 1$
Segment 2:	$-1 \le u < 0$,	$s = u-(-1)$
	$x(s) = -1 + s$,	$y(s) = 2$
Segment 3:	$0 \le u < 1$,	$s = u$
	$x(s) = s$,	$y(s) = 2 + s$
Segment 4:	$1 \le u < 2$,	$s = u - 1$
	$x(s) = 1 + s$,	$y(s) = 3 + s + s^2$
Segment 5:	$2 \le u < 3$,	$s = u - 2$
	$x(s) = 2 + s$,	$y(s) = 5 + 3s + s^2 + s^3$
Segment 6:	$3 \le u \le 4$	$s = u - 3$
	$x(s) = 3 + s$,	$y(s) = 10 + 8s + 4s^2 + s^3$

Output (see Fig. 2)

Degree = 3

Dimension = 2

Knot Sequence:

KNOT VALUE	MULTIPLICITY
-2	4
-1	4
0	3
1	2
2	1
3	0
4	4

B-Spline Coefficients (all weights = 1)

Coeff(1) = (-2, 1)

Coeff(2) = ($-1\frac{2}{3}$, 1)

Coeff(3) = ($-1\frac{1}{3}$, 1)

Coeff(4) = (-1, 1)

Coeff(5) = (-1, 2)

Coeff(6) = ($-\frac{2}{3}$, 2)

Coeff(7) = ($-\frac{1}{3}$, 2)

Coeff(8) = (0, 2)

Coeff(9) = ($\frac{1}{3}$, $2\frac{1}{3}$)

Coeff(10) = ($\frac{2}{3}$, $2\frac{2}{3}$)

Coeff(11) = ($1\frac{1}{3}$, $3\frac{1}{3}$)

Coeff(12) = ($2\frac{1}{3}$, $5\frac{1}{3}$)

Coeff(13) = ($3\frac{1}{3}$, $10\frac{1}{3}$)

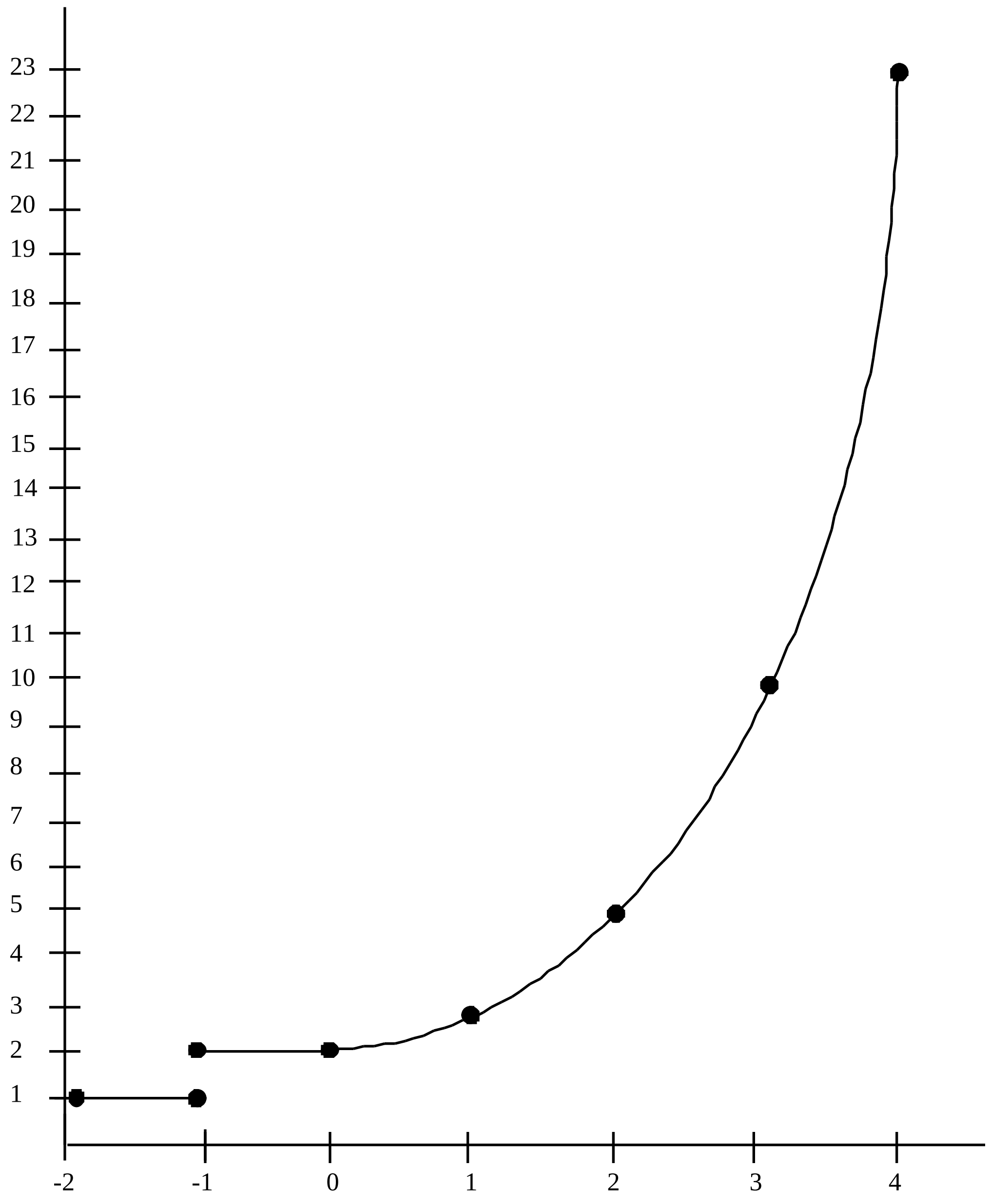
Coeff(14) = (4, 23)

Note the following characteristics of the knot sequence.

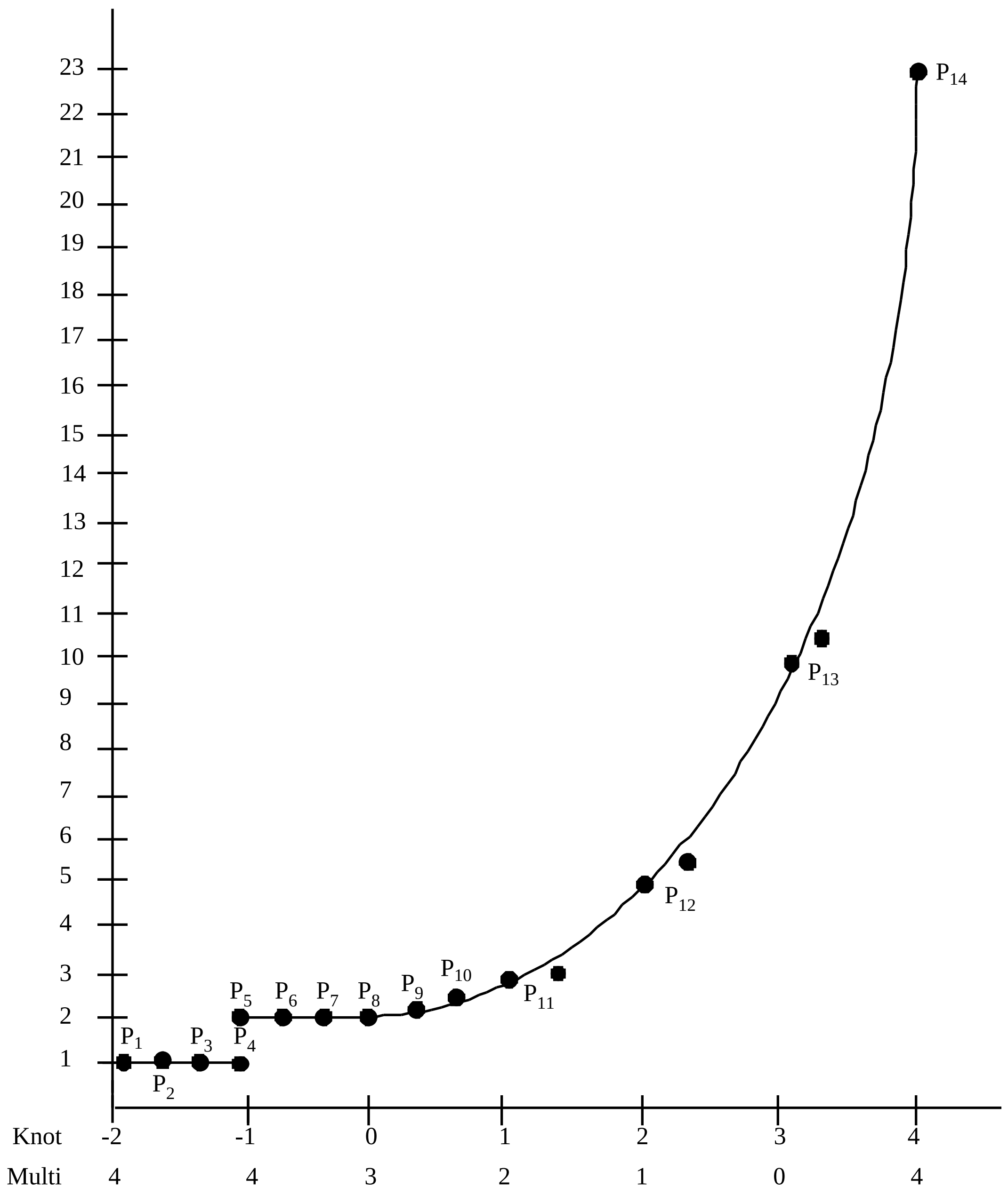
1) The start and end knots each have multiplicity = degree + 1.

2) For all interior knots, we have multiplicity = degree - k, where k is the level of derivative continuity. Thus, at knot value u = 1, the function is C^1 continuous, so the knot multiplicity is 3 - 1 = 2. At knot value u = 3, the function is C^3 continuous, so the knot multiplicity is zero, i.e., the knot isn't there at all, and on the interval [2, 4] the curve can be represented by a single polynomial piece.

[Figure 1. *Input: A Piecewise Polynomial Curve*]



[Figure 2. *Output: A B-Spline Curve, With B-Spline Coefficients and Knot Multiplicities Portrayed*]



Example C. <u>Four-Patch Surface</u>

In this example, we start out with a B-spline surface, convert it to a power basis (parametric spline) representation, and convert it back to a B-spline surface. As we shall see, it turns out that the second B-spline surface is simpler than the first, in that one of the original knot values was not really necessary.

Surface #1¾¾¾¾	→ Surface #2¾¾¾¾	→ Surface #3
B-Spline Form	Power Basis Form	B-Spline Form

<u>Surface #1 Defining Data</u>

Degree in 1st Parameter = 2
Degree in 2nd Parameter = 3
First Knot Sequence: 0, 0, 0, 1, 2, 2, 2
Second Knot Sequence: 0, 0, 0, 0, 1, 2, 2, 2, 2
B-Spline Coeffs $C_{i,\,j}$ defined as follows:

i j→	1	2	3	4	5
↓1	(1, 1, 0)	(1, 2, 1)	(1, 3, 2)	(1, 4, 1)	(1, 5, 0)
2	(2, 1, 1)	(2, 2, 2)	(2, 3, 3)	(2, 4, 2)	(2, 5, 1)
3	(3, 1, 1)	(3, 2, 2)	(3, 3, 3)	(3, 4, 2)	(3, 5, 1)
4	(4, 1, 0)	(4, 2, 1)	(4, 3, 2)	(4, 4, 1)	(4, 5, 0)

All weights are set equal to 1.

<u>Surface #2 Defining Data</u>

Four power basis polynomial patches each of the form

$F(s, t) = \sum_{i=0}^{2} \sum_{j=0}^{3} C_{i,j}\, s^i\, t^j$ where s and t are local parameters defined for each patch. The global parameters are $0 \le u \le 2$ and $0 \le v \le 2$ with suitable restrictions on each patch.

Patch # (1, 1) Global Parameters $0 \le u < 1$, $0 \le v < 1$
Local Parameters $s = u$, $t = v$

$C_{00} = (1, 1, 0)$	$C_{01} = (0, 3, 3)$	$C_{02} = (0, -1.5, -1.5)$	$C_{03} = (0, .5, 0)$
$C_{10} = (2, 0, 2)$	$C_{11} = (0, 0, 0)$	$C_{12} = (0, 0, 0)$	$C_{13} = (0, 0, 0)$
$C_{20} = (-.5, 0, -1)$	$C_{21} = (0, 0, 0)$	$C_{22} = (0, 0, 0)$	$C_{23} = (0, 0, 0)$

Patch # (1, 2) Global Parameters $0 \leq u < 1$, $1 \leq v \leq 2$
Local Parameters $s = u$, $t = v - 1$

C_{00} = (1, 3, 1.5)	C_{01} = (0, 1.5, 0)	C_{02} = (0, 0, -1.5)	C_{03} = (0, .5, 0)
C_{10} = (2, 0, 2)	C_{11} = (0, 0, 0)	C_{12} = (0, 0, 0)	C_{13} = (0, 0, 0)
C_{20} = (-.5, 0, -1)	C_{21} = (0, 0, 0)	C_{22} = (0, 0, 0)	C_{23} = (0, 0, 0)

Patch # (2, 1) Global Parameters $1 \leq u \leq 2$, $0 \leq v < 1$
Local Parameters $s = u - 1$, $t = v$

C_{00} = (2.5, 1, 1)	C_{01} = (0, 3, 3)	C_{02} = (0, -1.5, -1.5)	C_{03} = (0, .5, 0)
C_{10} = (1, 0, 0)	C_{11} = (0, 0, 0)	C_{12} = (0, 0, 0)	C_{13} = (0, 0, 0)
C_{20} = (.5, 0, -1)	C_{21} = (0, 0, 0)	C_{22} = (0, 0, 0)	C_{23} = (0, 0, 0)

Patch # (2, 2) Global Parameters $1 \leq u \leq 2$, $1 \leq v \leq 2$
Local Parameters $s = u - 1$, $t = v - 1$

C_{00} = (2.5, 3, 2.5)	C_{01} = (0, 1.5, 0)	C_{02} = (0, 0, -1.5)	C_{03} = (0, .5, 0)
C_{10} = (1, 0, 0)	C_{11} = (0, 0, 0)	C_{12} = (0, 0, 0)	C_{13} = (0, 0, 0)
C_{20} = (.5, 0, -1)	C_{21} = (0, 0, 0)	C_{22} = (0, 0, 0)	C_{23} = (0, 0, 0)

<u>Surface #3 Defining Data</u>

This surface is obtained from #2 by applying the appropriate conversion algorithm. Note that it differs from Surface #1 in that there is one less knot value in the second parameter and thus there are different control points.

Degree in 1st Parameter = 2
Degree in 2nd Parameter = 3
First Knot Sequence: 0, 0, 0, 1, 2, 2, 2
Second Knot Sequence: 0, 0, 0, 0, 2, 2, 2, 2
(Note that the knot value 1 has been removed because the function is C^3 continuous there.)

B-Spline Coeffs C_{ij} defined as follows:

i j→	1	2	3	4
↓1	(1, 1, 0)	(1, 3, 2)	(1, 3, 2)	(1, 5, 0)
2	(2, 1, 1)	(2, 3, 3)	(2, 3, 3)	(2, 5, 1)
3	(3, 1, 1)	(3, 3, 3)	(3, 3, 3)	(3, 5, 1)
4	(4, 1, 0)	(4, 3, 2)	(4, 3, 2)	(4, 5, 0)

4.0 GENERAL METHODOLOGY

In this section we present an overview of the mathematical conversion algorithms used. We discuss the simpler non-rational case first.

A. B-Spline to Parametric Spline

To obtain the power basis coefficients, we evaluate all derivatives from the right at the appropriate knot values and divide by the appropriate factorials. In the case of surfaces, we take the appropriate mixed partial derivatives.

B. Parametric Spline to B-Spline

The following steps outline the distinct methods we have used to convert curves and surfaces. The interested user may want to experiment with different approaches.

1. Curves

Step 1:

Left and right derivatives of all orders are taken at each break point to determine the level of continuity.

Step 2:

Using this continuity information, a knot sequence with appropriate multiplicities is constructed.

Step 3:

Using this knot sequence, the deBoor-Fix algorithm (see reference [2]) is applied to obtain the B-spline coefficients.

2. Surfaces

Step 1:

A knot sequence of maximum multiplicity (i.e., degree + 1) is constructed in each parameter.

Step 2:

The tensor product version of the deBoor-Fix algorithm is applied to obtain the B-spline coefficients for the surface. Because we imposed maximum multiplicity in the knot sequence, at this stage the surface is actually piecewise Bezier.

Step 3:

We attempt to remove as many knots as possible by reversing the process described in Boehm’s paper, [1].

C. Extensions to the Rational Case

The general approach is depicted by the following diagram:

$$\begin{tikzcd}
\text{B-spline} \arrow[r, leftrightarrow] \arrow[d, leftrightarrow] & \text{Parametric Spline} \arrow[d, leftrightarrow] \\
\text{Rational B-Spline} \arrow[r, leftrightarrow, dashed] & \text{Rational Parametric Spline}
\end{tikzcd}$$

The mappings between the (non-rational) B-spline and the parametric spline are performed as described in sections A and B above. This is where the bulk of the mathematical manipulations appear.

However, the initial input and final output are assumed to be rational curve or surface entities. Therefore, we have provided procedures to map a rational entity to and from the corresponding non-rational entity in homogeneous form. See the code for details.

D. Determination of Smoothness

The method we have used to convert between power basis spline and B-spline curves (but not surfaces) follows that described in deBoor, [2].

On page 121 of [2], deBoor points out that it is more difficult to convert from power basis to B-spline because the power basis representation contains no explicit information about the smoothness of the function at the breakpoints. As we noted above, this smoothness information is required to determine the (minimal) knot multiplicities for the B-spline representation.

Given an input power basis spline curve from an IGES file, parameter #2 in the parameter data section specifies the degree of derivative continuity with respect to arc length. However, we cannot use this parameter to determine minimal knot multiplicities for the following reason: A parametric spline curve can be k times continuously differentiable with respect to arc length (we denote this by G^k; G for geometric continuity) but may be less than k times continuously differentiable with respect to the parameter value.

Therefore, to determine the degree of derivative continuity, we check to see whether the left derivatives and right derivatives agree, to within a suitable tolerance. The tolerance we use has been determined in a rather ad hoc manner as a function of the machine epsilon, as well as the magnitude and the order of the derivatives in question. The interested user may wish to experiment with the method of calculating tolerances.

It is sometimes useful to represent a B-spline curve or surface using additional knot values, or having greater multiplicity at an existing knot value. Therefore, we have provided “add-knot” procedures following the algorithm given by Boehm in [1].

As we mentioned earlier, the method we have used to convert from power basis to B-spline surfaces involves first constructing the corresponding B-spline surface having maximal knot multiplicity and then trying to remove as many knots as possible in each parameter value. The interested user may wish to try the same method for the curve case.

When we try to remove knots from a B-spline curve or surface we once again run into a tolerancing problem. The problem arises at the point in the reverse Boehm algorithm when a certain B-spline coefficient can be expressed in two different ways. If these two values agree (to within tolerance) we go ahead and remove the knot. As with our approach to continuity determination, the choice of tolerance is made in a rather ad hoc manner, and the interested user may wish to experiment with other approaches.

5.0 TESTING

As we mentioned earlier, we have provided Pascal procedures to test the conversion algorithms. There are basically two kinds of tests we use.

A. Evaluation and Comparison:

The curve or surface is evaluated over a uniformly spaced array of parameter values. Evaluation is performed independently using both the rational parametric spline and rational B-spline representations. The number of evaluation points is specified by the user. The maximum difference, as well as the maximum and minimum norms for each representation are written out.

B. Repeated Conversions:

The input entity is converted back and forth between rational parametric spline and rational B-spline representations. The number of pairs of evaluations is specified by the user. The last set of coefficients generated is compared with the initial set, and the maximum differences (in the numerator and denominator) are written out.

C. Examples:

When we applied the tests outlined above to some curves and surfaces related to the examples presented in Section A, we obtained the following results using the double precision on the IBM 4341.

ENTITY	MAXIMUM ERROR AFTER	MAXIMUM
ERROR		
	100 EVALUATIONS	AFTER 100 PAIRS OF CONVERSIONS
Quarter Circle Centered at Origin	2.1×10^{-16}	0.0
Quarter Circle Translated Slightly	4.4×10^{-16}	2.2×10^{-14}
Conic Obtained by Perturbing Quarter Circle Coefficients	4.9×10^{-16}	4.8×10^{-14}
Six-Segment Spline Curve	1.1×10^{-14}	6.4×10^{-13}
Four-Patch Surface	1.3×10^{-15}	7.0×10^{-14}

6.0 IMPLEMENTATION CONSIDERATIONS

The conversion algorithms were implemented on an IBM 4341 using VM/CMS. The language chosen was Pascal/VS Release 2.1. We found Pascal to be useful because the language enables us to readily define complex data structures (such as rational B-spline surface) and because it provides a rich supply of control structures. Users may either invoke all or portions of the code directly, tailor it to their needs, or simply use it as a reference for the ideas it contains.

The user should be aware of the following features of this implementation.

A. No Pointers.

We have not used dynamic variables at all in this implementation, because we felt that having lots of pointers would make the code harder to read and make it difficult to convert to a language such as FORTRAN. On the other hand, not having dynamic variables meant that we had to hard code in specific upper limits such as maximum knot sequence size and maximum dimension of the coefficients.

B. No Indefinite Array Size.

Some versions of Pascal allow the user to specify an indefinite array size in the type of a variable used as a parameter in a procedure. For instance, one could say "PROCEDURE TEST(X:ARRAY[INTEGER] OF REAL);". However, Pascal/VS does not have this feature.

C. Double Precision.

Those variables that are of Pascal/VS type REAL would be called "double precision" or REAL * 8 in other implementations. That is, variables of type REAL occupy 8 bytes of memory, and double precision floating point arithmetic is used.

D. No Square Brackets for Arrays.

Most Pascal books and manuals we have seen use square brackets, '[' and ']' in the declaration of arrays. However, the I/O character set available to us on the 4341 did not include square brackets. Therefore, for array declarations we used '('and ')' wherever brackets are called for.

E. Control Structures.

We use the standard Pascal control structures and do not use any "go to" statements. However, in several places we use the non-standard "leave" statement to cause an immediate, unconditional exit from a loop. Such use of "leave" statements can be reformulated, if necessary, in terms of "go to" statements.

F. Input/Output.

In this implementation all input is from the terminal, using logical file name "INPUT". There are three output files with logical file names "OUTPUT", "DEBUG", and "REPORT". These refer, respectively, to terminal output, and the files named "PPBSPLN DEBUG" and "PPBSPLN REPORT". The functions "TERMIN", "TERMOUT" and "REWRITE" are used to prepare the above files for input or output. Standard output information is written to the "REPORT" file. If the global variable "IDEBUG" is set to 1,then additional information could be written to the "DEBUG" file by adding suitable statements to the code.

If the user wishes to implement this code on another system, he should probably be aware that file-handling conventions will probably differ.

G. Other Non-Standard Features.

The only other non-standard features in the code are the following:

1. %INCLUDE CMS - enables CMS (operating system) commands to be invoked from the program.
2. % PAGE - causes the following lines of the listing file generated by the compiler from source code to start on a new page.
3. CMS('CLRSCRN',RETURN-CODE) - clears the screen.

7.0 SUMMARY OF PASCAL PROCEDURES USED

We have grouped the Pascal procedures and functions into a number of categories. A brief description of each procedure follows the list of categories below. More detailed documentation appears in the code itself.

<u>Categories</u>

VECTOR OPERATIONS
MATHEMATICAL UTILITIES
CREATE FRACTIONAL PARTS
INPUT/OUTPUT
INTERVAL FINDING
EVALUATION
KNOT FINDING
KNOT MANIPULATION
CONVERSION BETWEEN LIKE ENTITIES
CONVERSION BETWEEN UNLIKE ENTITIES
TEST UTILITIES
TEST DRIVERS

<u>Procedures and Functions by Category</u>

CATEGORY: <u>VECTOR OPERATIONS</u>

Since the B-spline and power-basis spline coefficients and function values are arbitrary dimensional vectors, we have provided the following procedures and functions.

WRITEVEC	:	Writes the value of a vector to a file
SUM	:	Finds the sum of two vectors
DIFFERENCE	:	Finds the difference of two vectors
NORM	:	Finds the Euclidean norm of a vector
DIST	:	Finds the Euclidean distance between two vectors
PRODUCT	:	Finds the product of a scalar and a vector
QUOTIENT	:	Finds the quotient of a vector and a scalar
ZEROVEC	:	Creates the zero vector of a given dimension
PROJECT	:	Projects a vector onto one of one less dimension
APPEND	:	Appends a scalar to a vector to form a new vector

CATEGORY: <u>MATHEMATICAL UTILITIES</u>

The following utilities perform various mathematical tasks which were not available as system routines.

FINDEPS	:	Finds an approximation to the machine epsilon

SYMMETRIC:	Computes values of the symmetric functions
FACTORIAL :	Computes the factorial function
POWER :	Raises an integer to a non-negative integral power

CATEGORY: <u>CREATE FRACTIONAL PARTS</u>

For convenience in the parametric evaluation of rational B-spline and power basis curves and surfaces separate numerator and denominator fields are created by these procedures and added to the appropriate record.

RP_FRACTION :	Create numerator and denominator for a rational polynomial curve
RPSURF_FRACTION:	Create numerator and denominator for a rational polynomial surface
RBS_FRACTION :	Create numerator and denominator for a rational B-spline curve
RBSURF_FRACTION:	Create numerator and denominator for a rational B-spline surface

CATEGORY: <u>INPUT/OUTPUT</u>

These procedures enable the user to read in or write out the various entities of interest. The interested user may wish to make enhancements to them such as being able to save entities by name, read from a file, etc.

READ_KNOTS	:	Prompt the user for a knot sequence of the appropriate length and check that it is nondecreasing.
READ_RATIONAL_BSPLINE	:	Prompt the user and read in a rational B-spline curve.
WRITE_RBSURF	:	Write the defining data for a rational B-spline surface to a designated output file.
WRITE_RBS	:	Write the defining data for a rational B-spline curve to a designated output file.
READ_RATIONAL_PP	:	Prompt the user and read in a rational piecewise polynomial curve.
READ_RPPSURF	:	Prompt the user and read in a rational piecewise polynomial surface.
WRITE_RPSURF	:	Write the defining data for a rational piecewise polynomial surface to a designated output file.
WRITE_RPP	:	Write the defining data for a rational piecewise polynomial curve to a designated output file.

CATEGORY: <u>INTERVAL FINDING</u>

In order to evaluate piecewise polynomial or B-spline curves or surfaces, we need to find the indices of the intervals of the given parameter values.

BRACKET	:	Given a parameter and a set of indexed break points, find an index

		so that the corresponding break points bracket the parameter.
KNOT_INTERVAL	:	Given a parameter and a set of indexed knots, find an index so that the corresponding knots bracket the parameter.

CATEGORY: <u>EVALUATION</u>

Perform parametric evaluation of the value and, in some cases, derivatives of a piecewise polynomial or B-spline curve or surface. Separate routines are provided for the rational case.

LEEVAL	:	Evaluates the values and derivatives of a one-span vector-valued B-spline curve using an algorithm of E. Lee [3].
HORNER	:	Uses Horner's method to evaluate a polynomial with vector coefficients.
FORMAL_DERIVATIVE	:	Finds, but does not evaluate, the polynomial which is the derivative of a given polynomial.
EVAL_POLY	:	Evaluates a vector-valued polynomial curve and its derivatives at a given parameter value.
EVAL_PSURF	:	Evaluates a vector-valued polynomial surface and its derivatives at a given pair of parameter values.
EVAL_PPSURF	:	Evaluates a vector-valued piecewise polynomial surface and its derivatives at a given pair of parameter values.
EVAL_RPPSURF	:	Evaluates a vector-valued rational piecewise polynomial surface at a given pair of parameter values.
EVAL_PP	:	Evaluates a vector-valued piecewise polynomial curve and its derivatives at a given parameter value.
EVAL_RPP	:	Evaluates a vector-valued rational piecewise polynomial curve at a given parameter value.
KNOTEVAL	:	Finds the left and right derivatives of a piecewise polynomial curve at a breakpoint.
EVAL_BSPLINE	:	Evaluates a B-spline curve and its derivatives at a given parameter.
EVAL_RBS	:	Evaluates a rational B-spline curve at a given parameter.
EVAL_BSURF	:	Evaluates a B-spline surface and its derivatives at a given pair of parameters.
EVAL_RBSURF	:	Evaluates a rational B-spline surface at a given pair of parameters.

CATEGORY: <u>KNOT FINDING</u>

This set of procedures is used to determine the location and multiplicities of knot values for B-spline curves and surfaces that are converted from piecewise power basis representations. The underlying rule is that the knot multiplicity is equal to the degree minus the level of derivative continuity. Knots of full multiplicity (degree plus one) are placed at the start and end points.

DETERMINE_CONTINUITY	:	Find the level of derivative continuity at each breakpoint of a piecewise polynomial curve, and use this information to determine knot multiplicities.
FIND_KNOTS	:	Using the information from procedure DETERMINE_CONTINUITY, build the knot sequence for a B-spline curve.
FIND_KNOTS2	:	For a piecewise polynomial surface, build two knot sequences of maximal multiplicity.
FIND_DISTINCT_KNOTS	:	Given a knot sequence, find the distinct knots. These will become the breakpoints of the corresponding piecewise polynomial.

CATEGORY: <u>KNOT MANIPULATION</u>

Given a B-spline curve or surface, one can always add more knots. This will result in a curve or surface which is parametrically identical to the original, but which has a larger (and partially different) set of B-spline coefficients. Conversely, one can sometimes, but not always, remove a knot without changing the parameterization of the curve or surface.

ADD_KNOT	:	Add a knot to a B-spline curve.
ADD_SURF_KNOT	:	Add a knot to a B-spline surface.
REMOVE_KNOT	:	Try to remove a knot from a B-spline curve.
REMOVE_SURF_KNOT	:	Try to remove a knot from a B-spline surface.
REMOVE_ALL_SURF_KNOTS	:	Remove as many knots as possible from a B-spline surface.

CATEGORY: <u>CONVERSION BETWEEN LIKE ENTITIES</u>

The purpose here is to convert to and from rational entities and the corresponding non-rational entity of one dimension higher.

RPOLY_TO_POLY	:	Rational polynomial curve to polynomial curve.
PP_TO_RPP	:	Piecewise polynomial curve to rational piecewise polynomial curve.
RPP_TO_PP	:	Rational piecewise polynomial curve to piecewise polynomial curve.
BSPLINE_TO_RBSPLINE	:	B-spline curve to rational B-spline curve.

RBSPLINE_TO_BSPLINE	:	Rational B-spline curve to B-spline curve.
PSURF_TO_RPSURF	:	Polynomial surface to rational polynomial surface.
PPSURF_TO_RPPSURF	:	Piecewise polynomial surface to rational piecewise polynomial surface.
RPPSURF_TO_PPSURF	:	Rational piecewise polynomial surface to piecewise polynomial surface.
BSURF_TO_RBSURF	:	B-spline surface to rational B-spline surface.
RBSURF_TO_BSURF	:	Rational B-spline surface to B-spline surface.

CATEGORY: <u>CONVERSION BETWEEN UNLIKE ENTITIES</u>

This category contains the most important algorithms in the whole package, because they convert between power basis curves and surfaces and the corresponding B-spline curves and surfaces.

DEBOOR_FIX	:	Given a piecewise polynomial curve and a knot sequence, find the corresponding B-spline coefficients using a result of deBoor and Fix described in reference [2].
DEBOOR_FIX2	:	Given a piecewise polynomial surface and two knot sequences, find the corresponding B-spline coefficients. This is the tensor product version of DEBOOR_FIX.
PP_TO_BSPLINE	:	Convert a piecewise polynomial curve to a B-spline curve.
BSPLINE_TO_PP	:	Convert a B-spline curve to a piecewise polynomial curve.
RPP_TO_RBSPLINE	:	Convert a rational piecewise polynomial curve to a rational B-spline curve.
RBSPLINE_TO_RPP	:	Convert a rational B-spline curve to a rational piecewise polynomial curve.
PPSURF_TO_BSURF	:	Convert a piecewise polynomial surface to a B-spline surface.
RPPSURF_TO_RBSURF	:	Convert a rational piecewise polynomial surface to a rational B-spline surface.
BSURF_TO_PPSURF	:	Convert a B-spline surface to a piecewise polynomial surface.
RBSURF_TO_RPPSURF	:	Convert a rational B-spline surface to a rational piecewise polynomial surface.

CATEGORY: <u>TEST UTILITIES</u>

These utilities enable the user to test the results of converting between the power basis representation and the corresponding B-spline representation of a curve or surface.

C_EVAL_AND_COMPARE : Compare a given rational piecewise polynomial curve and a

rational B-spline curve by evaluating both of them at a

		designated number of points.
RPP_COMPARE	:	Find the maximum difference in coefficients between two rational piecewise polynomial curves.
RBS_COMPARE	:	Find the maximum difference in coefficients between two rational B-spline curves.
RPPSURF_COMPARE	:	Find the maximum difference in coefficients between two rational piecewise polynomial surfaces.
RBSURF_COMPARE	:	Find the maximum difference in coefficients between two rational B-spline surfaces.
S_EVAL_AND_COMPARE	:	Compare a given rational piecewise polynomial surface and a rational B-spline surface by evaluating both at a designated number of points.

CATEGORY: TEST DRIVERS

One of these procedures is invoked by the main program in response to the user's input regarding what kind of entity he wishes to start with. They obtain the appropriate input, see that it is converted to the other form, perform tests as requested by the user, and write out results as they occur.

START_WITH_RPP_CURVE	:	Invoked if the user indicated that he wished to start with a rational piecewise polynomial curve.
START_WITH_RBS_CURVE	:	Invoked if the user indicated that he wished to start with a rational B-spline curve.
START_WITH_RPP_SURFACE	:	Invoked if the user indicated that he wished to start with a rational piecewise polynomial surface.
START_WITH_RBS_SURFACE	:	Invoked if the user indicated that he wished to start with a rational B-spline surface.
MAIN PROGRAM	:	Opens all files, prompts the user for what type of curve or surface he wants to start with, and calls the appropriate test driver procedure.

8.0 REFERENCES

1. Boehm, W. (1980), Inserting new knots into B-spline curves, Computer-aided Design 12, 199-201.

2. deBoor, C. (1978), A Practical Guide to Splines, Springer, Berlin.

3. Lee, E. (1982), A simplified B-spline computation routine, Computing 29, 365-373.

4. The Initial Graphics Exchange Specification (IGES), Version 2.0, Bradford Smith, Kalman G. Brauner, Philip Kennicott, Michael Liewald, Joan Wellington, NBSIR 82-2631.