

B-Splines And Divided Differences

James D Emery

Version: 11/10/2009

Contents

1	Introduction	2
2	Divided Differences For Distinct Points	2
3	The Space of Piecewise Polynomials	10
4	B-spline Bases Functions	11
5	B-spline Bases Function Program	13
6	The Derivatives of a B-Spline Bases Function	14
7	Program For Bases Function Derivatives	15
8	B-Spline Curves	15
9	The Derivative of a B-Spline Curve	16
10	IGES Definition	16
11	PDES/STEP Definition	18
12	The B-Spline Dual Space	18
13	A B-Spline Representation of the WF-Spline	19
14	A B-Spline Representation of a Bezier Curve	20

15 B-Spline Interpolation	21
16 B-Spline Surface Patch	22

1 Introduction

B-spline curves are defined using the theory of divided differences. The theory of differences play an important role in computation and in Numerical Analysis. We shall define divided differences and derive some of their important properties. Then we shall define B-splines and methods of computing them. We shall show some of the interesting properties of B-splines including the fact that they form a basis for spaces of piecewise polynomials. We shall show that a Bezier curve is a special B-spline curve. Bezier curves are treated in the report titled **Bezier Curves** by Jim Emery. The calculations described here are carried out in a set of computer programs.

2 Divided Differences For Distinct Points

Divided differences are approximations to derivatives. They have important application in various areas of numerical analysis. They are used in various polynomial interpolation formulas. They are used to define a class of functions called B-splines.

The divided differences of a function f are calculated on a set of distinct points $\{x_0, x_1, \dots, x_n\}$ of its domain. They are called knots. These functions of the knot set are defined by recursion,

$$[x_0]f = f(x_0)$$

and

$$[x_0, x_1, \dots, x_n]f = \frac{[x_0, x_1, \dots, x_{n-1}]f - [x_1, x_2, \dots, x_n]f}{x_0 - x_n}.$$

An important application of divided differences is Newton's interpolation formula. This formula is

$$f(x) = [x_0]f + (x - x_0)[x_0, x_1]f \\ + \dots$$

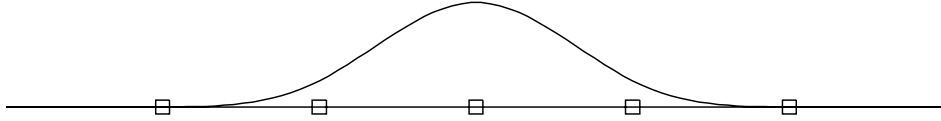


Figure 1: **Cubic B-Spline Function On Distinct Knots.** A *B-Spline function* defined on the knots $0, 1, 2, 3, 4$, so that it is a cubic spline with support on $(0, 4)$, with continuous first and second derivatives. The plot is not uniformly scaled.

$$\begin{aligned}
 &+(x - x_0)\dots(x - x_{n-1})[x_0, x_1, \dots, x_n]f \\
 &+[x, x_0, x_1, \dots, x_n]f(x - x_0)\dots(x - x_n).
 \end{aligned}$$

The last term is an error term. The other terms give a polynomial approximation to f . We will subsequently show that if f is a n th degree polynomial, then the error term vanishes. This holds because the $n + 1$ divided difference of an n th degree polynomial is zero. From this it follows that the approximating polynomial in Newton's formula is the unique polynomial that interpolates f at the knots.

Newton's interpolation formula may be proved by induction. We have

$$[x, x_0, x_1, \dots, x_{n+1}]f = \frac{[x, x_0, x_1, \dots, x_n]f - [x_0, x_1, \dots, x_{n+1}]f}{x - x_{n+1}}.$$

We may prove the formula by solving for $[x, \dots, x_n]f$ and substituting in the next lower order formula. We need to show that the divided differences are symmetric functions.

We shall use a new formula for the divided differences. This formula becomes evident on examining the relation between the Lagrange and the Newton forms of the interpolation polynomial, starting at the lowest order.

We define

$$\omega_n(x) = (x - x_0)(x - x_1)\dots(x - x_n).$$

We have

$$\omega'_n(x_i) = (x_i - x_0)(x_i - x_1)\dots(\widehat{x_i - x_i})\dots(x_i - x_n)$$

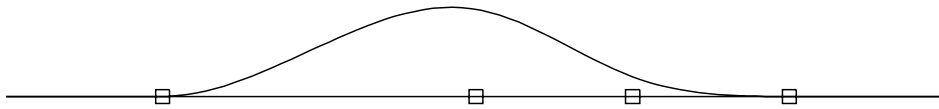


Figure 2: **Cubic B-Spline Function With Two Repeating Knots.** A B-Spline function defined on the knots $0,0,2,3,4$, so that there is a loss of continuity at 0 , namely the second derivative is not continuous there. The plot is not uniformly scaled.

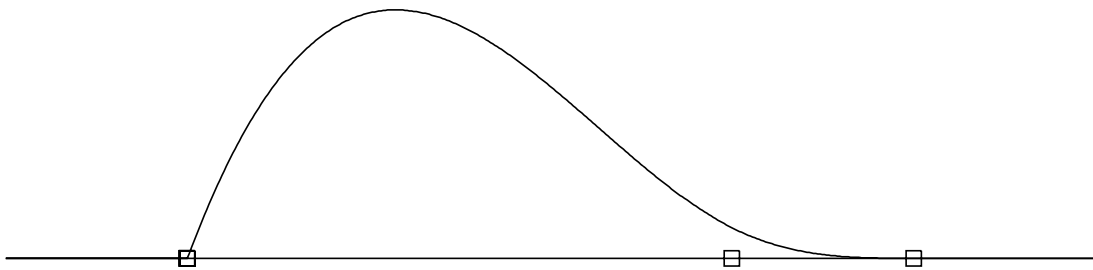


Figure 3: **Cubic B-Spline Function With Three Repeating Knots.** A B-Spline function defined on the knots $0,0,0,3,4$, so that there is a loss of continuity at 0 , namely the first derivative is not continuous there. The plot is not uniformly scaled.

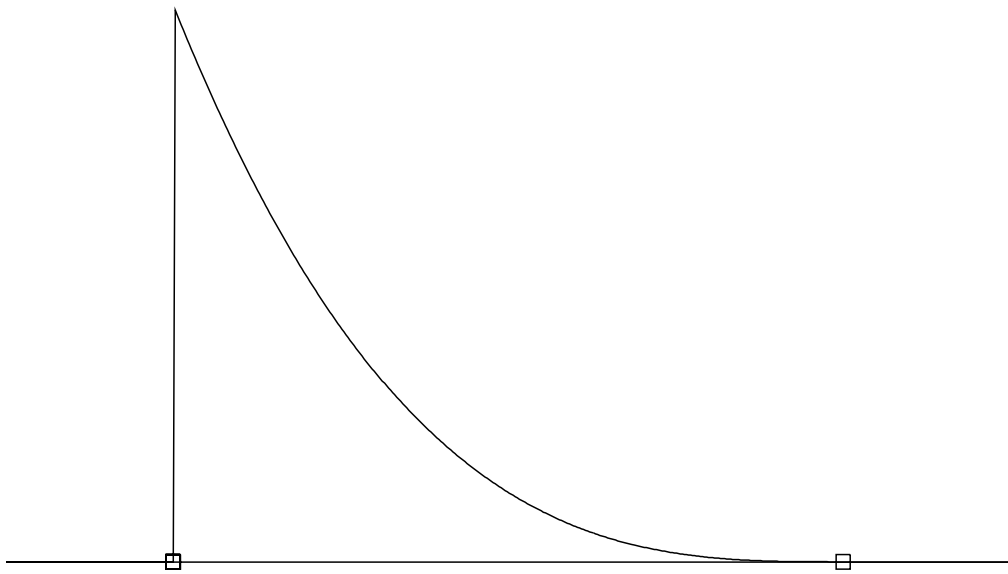


Figure 4: **Cubic B-Spline Function With Four Repeating Knots.** *A B-Spline function defined on the knots $0,0,0,0,4$, so that there is a loss of continuity at 0 , the function is not continuous at 0 . The plot is not uniformly scaled.*

where the hat means the symbol is omitted.

Proposition

$$[x_0, \dots, x_n]f = \sum_{i=0}^n \frac{f_i}{\omega'_n(x_i)}.$$

Proof. The proof is by induction.

$$\begin{aligned} [x_0, x_1, \dots, x_{n+1}]f &= \frac{[x_0, x_1, \dots, x_n]f - [x_1, x_2, \dots, x_{n+1}]f}{x_0 - x_{n+1}} \\ &= \frac{f_0}{\omega'_n(x_0)(x_0 - x_{n+1})} + \\ &\dots + \frac{f_i/[(x_i - x_0)\dots(x_i - x_n)] - f_i/[(x_i - x_1)\dots(x_i - x_{n+1})]}{x_0 - x_{n+1}} + \\ &\dots - \frac{f_{n+1}}{(x_{n+1} - x_1)(x_{n+1} - x_2)\dots(x_{n+1} - x_n)(x_0 - x_{n+1})} \end{aligned}$$

The i th term is $f_i/\omega'_{n+1}(x_i)$.

Proposition Divided differences are symmetric functions.

Proof. Consider the previous formula. Suppose points x_i and x_j are interchanged. Then the i th term and the j th term in the formula are interchanged. The k th term contains the factor $(x_k - x_i)(x_k - x_j)$ in the denominator and so is unchanged.

Proposition. The $n + 1$ divided difference of a polynomial of degree n vanishes.

Proof. It is sufficient to consider the polynomial $f(x) = x^n$. We have

$$[x_0, x_1]f = \frac{x_0^n - x_1^n}{x_0 - x_1} = x_0^{n-1} + x_0^{n-2}x_1 + \dots + x_1^{n-1}$$

This is a multivariate polynomial of degree $n - 1$. This result is true in general.

Lemma If $f(x) = x^n$ and $k \leq n$, then the difference $[x_0, \dots, x_k]f$ is a multivariate polynomial in the difference points of degree less than or equal to $n - k$.

Proof. The proof is by induction. We have

$$[x_0, x_1, \dots, x_{k+1}]f = \frac{[x_0, x_1, \dots, x_k]f - [x_1, x_2, \dots, x_{k+1}]f}{x_0 - x_{k+1}}.$$

The numerator is a polynomial by assumption. If $x_0 = x_{k+1}$ then the numerator vanishes. Hence it contains the denominator as a factor and so the expression is a polynomial. The degree is reduced by 1 so it is less than or equal to $n - (k + 1)$.

Remark. The condition for a factor may be established as follows. Given a polynomial $p(x_1, x_2, \dots, x_n)$ we may consider it to be a polynomial in x_1 , and write

$$p(x_1, x_2, \dots, x_n) = (x_1 - x_2)q(x_1, x_2, \dots, x_n) + r(x_2, \dots, x_n)$$

If p vanishes whenever $x_1 = x_2$ then it follows that r is the zero function. That the polynomial coefficients of r must be zero follows by applying partial derivative operators. Linear functionals may be constructed that are nonzero only on a particular term. For example

$$\frac{\partial^3}{\partial x^2 \partial y}$$

is nonzero at $x = 0, y = 0$, only on the term x^2y . Of course such operators on zero functions are zero. It follows that r has zero coefficients and is the zero polynomial. Thus $x_1 - x_2$ is a factor of p .

Returning to the proof of the proposition, by the lemma the n th divided difference of x^n is a constant and so the $n + 1$ divided difference is zero. This completes the proof of the proposition.

Corollary (Newton form of the Interpolation Polynomial) Let p be the unique polynomial that agrees with f at x_0, x_1, \dots, x_n then

$$\begin{aligned} p(x) &= [x_0]f + [x_0, x_1]f(x - x_0) \\ &+ \dots + \\ &[x_0, x_1, \dots, x_n]f(x - x_0) \dots (x - x_{n-1}). \end{aligned}$$

Corollary Suppose $f \in C^{(n)}$ then \exists a ζ so that

$$[x_0, \dots, x_n]f = \frac{f^{(n)}(\zeta)}{n!}$$

Definition. The n th generalized divided differences $G(f, p)$ of a function f on the points

$$p = (x_0, x_1, \dots, x_n),$$

(the points are not necessarily distinct) is defined recursively by

$$G^n(f, p) = \begin{cases} \frac{f^{(n)}(y_0)}{n!}, & y_0 = y_n \\ \frac{G^{n-1}(f, y_0, y_1, \dots, y_{n-1}) - G^{n-1}(f, y_1, y_1, \dots, y_n)}{y_0 - y_n}, & y_0 < y_n \end{cases}$$

where $y_0 \leq y_1 \leq \dots \leq y_n$ is an ordering of the points of p .

G^n is a function of both the function f and the point set p . When f is fixed we shall write $G_f^n(p)$ and when p is fixed we shall write $G_p^n(f)$.

Divided differences may be defined in other ways. Schumaker defines them as ratios of determinants. He uses a notation similar to the following.

$$M(t_1, \dots, t_m; u_1, \dots, u_m),$$

is the m by m matrix with element i, j equal to $u_i(t_j)$. The u_i are functions. The divided difference is (Schumaker p.45)

$$G^n(f, p) = \frac{D(x_0, \dots, x_n; 1, x, x^2, \dots, x^{n-1}, f)}{D(x_0, \dots, x_n; 1, x, x^2, \dots, x^n)}.$$

where D is the generalized determinant, i.e. if the knots coincide then function values in the matrix are derivatives. (Schumaker p21.) Note that the denominator is the Vandermode determinant.

Theorem. $G_f^n(p) : \mathfrak{R}^{n+1} \rightarrow \mathfrak{R}$ is continuous.

Proof. Obvious.

Proposition. (Generalized Newton formula). If $f \in C^{(n+1)}$ then

$$\begin{aligned} f(x) = & G_f^0(x_0) + G_f^1(x_0, x_1)(x - x_0) + \dots + G_f^n(x_0, \dots, x_n)(x - x_0)(x - x_1) \dots (x - x_{n-1}) \\ & + G_f^{n+1}(x, x_0, \dots, x_n)(x - x_0)(x - x_1) \dots (x - x_n) \end{aligned}$$

Proof. The result follows from the continuity of G_f^j .

We write

$$f(x) = p_n(x) + G_f^{n+1}(x, x_0, \dots, x_n)(x - x_0)(x - x_1) \dots (x - x_n).$$

Proposition. If at least $j + 1$ of the points are identical to x_k , then

$$p_n^{(j)}(x_k) = f^{(j)}(x_k)$$

for $i \leq j$.

Proof. The divided differences are independent of the ordering, so we may suppose

$$x_k = x_{k+1} = \dots = x_{k+j},$$

and we may write

$$p_f(x) = G_f^0(x_k) + G_f^1(x_k, x_{k+1})(x - x_k) + \dots$$

Then

$$f(x) = p_n(x) + G_f^{n+1}(x, x_k, \dots, x_n)(x - x_k)(x - x_{k+1}) \dots (x - x_{k+n}).$$

From this we see that the i th derivative of the last term vanishes at x_k if $i \leq j$, so

$$p_n^{(i)}(x_k) = f^{(i)}(x_k).$$

Also

$$p_n^{(j)}(x_k) = j!G_f^j(x_k, \dots, x_{k+j}).$$

Thus p_n is the osculating interpolation polynomial that "agrees" with f on the given knot set.

Proposition. p_n is unique.

Proof. Suppose q_n is a second interpolating polynomial. Let $r = p_n - q_n$. Suppose $x_k = x_{k+1} = \dots = x_{k+j}$, then the polynomial r has a root of multiplicity $j + 1$ at x_k because j derivatives vanish there. Thus r is a polynomial of degree at most n with $n + 1$ zeroes. So r is zero.

Theorem. There exists a vector d_ξ , depending only on the knot vector $\xi = \{x_i, \dots, x_{i+k}\}$, so that the divided difference operator on a function f is given as the inner product

$$G_\xi f = \langle d_\xi, f_\xi \rangle.$$

Proof. There is a mapping ϕ of f to its k degree interpolating polynomial (osculating). We have $G(\phi(f)) = Gf$, because G depends only on function and derivative values of f at knots. Without loss of generality we may assume that f is a k -degree polynomial. Let P be the vector space of such polynomials. Let V be the space of associated knot function vectors. There is an isomorphism from P to V . This follows from the existence and uniqueness of the interpolation polynomial. G is a linear functional on P and so on V .

Thus G is in the dual space of V and has a coordinate vector d_ξ with respect to the dual basis. Hence

$$G_\xi f = \langle d_\xi, f_\xi \rangle .$$

3 The Space of Piecewise Polynomials

Let

$$\xi = \{\xi_1, \xi_2, \dots, \xi_{l+1}\}$$

be a strictly increasing sequence. The space of piecewise polynomials of order k on the break points ξ , written $P_{k,\xi}$ is the set of functions defined on $[\xi_1, \dots, \xi_{l+1}]$ that are polynomials of order k on each subinterval $[\xi_i, \xi_{i+1})$ for $i = 1, \dots, l$. The dimension of the vector space $P_{k,\xi}$ is kl . Each element of the space may be represented by a matrix of right derivatives. For $x \in [\xi_j, \xi_{j+1})$

$$f(x) = \sum_{i=0}^{k-1} f^{(i)}(\xi_j) \frac{(x - \xi_j)^i}{i!}.$$

Define $C = \{c_{i,j}\}$ to be the k by l matrix with elements

$$c_{i,j} = f^{(i-1)}(\xi_j).$$

A general pp function may be written as $f_{\xi,C}$. The IGES definition of a cubic parametric "spline" is given as the break point vector ξ and four row vectors of length l ,

$$A_f(j) = \frac{c(1,j)}{0!},$$

$$B_f(j) = \frac{c(2,j)}{1!},$$

$$C_f(j) = \frac{c(3,j)}{2!},$$

$$D_f(j) = \frac{c(4,j)}{3!},$$

where f stands for one of the coordinate functions x , y , or z .

Let

$$\phi_{ij}(x) = \frac{(x - \xi_i)_+^j}{j!},$$

for $i = 1, 2, \dots, l$ and $j = 0, 1, \dots, k - 1$. Define the linear functionals $\lambda_{i,j}$ to be the jump in the j th derivative at ξ_i . Using these functionals it is clear that the set of ϕ_{ij} is a basis of $P_{k,\xi}$.

Now we will impose continuity conditions at the internal break points and generate subspaces. Let

$$\nu = \{\nu_1, \dots, \nu_l\}$$

specify the continuity at the break points. The subspace $P_{k,\xi,\nu}$ of $P_{k,\xi}$ consists of piecewise polynomials that are $c^{(\nu_i-1)}$ at ξ_i ($c^{(-1)}$ means no continuity requirement). Now suppose $f \in P_{k,\xi,\nu}$ is c^m at ξ_i . Then the coefficient of ϕ_{ij} is zero if $j \leq m$, which can be seen by applying the appropriate functional. Thus

$$\{\phi_{i,j} : 1 \leq i \leq l, \nu_i \leq j < k\}$$

is a basis of $P_{k,\xi,\nu}$. We see that the dimension of $P_{k,\xi,\nu}$ is

$$\sum_{i=1}^l (k - \nu_i).$$

4 B-spline Bases Functions

Let T_x^k be the truncated power function of order k centered at x .

$$T_x^k(t) = (x - t)_+^{k-1}.$$

The function represented by the notation $(y)_+^j$, $j \geq 0$ is defined as follows. If $y < 0$, then the function is 0. If, $y \geq 0$, then the function is y^j . If $j = 0$, and $y = 0$, we take 0^0 to be 1. Hence the truncated power functions are C^∞ from the right. The 0 degree truncated power function is not continuous at zero. It is discontinuous from the left, but is continuous from the right.

Let $\tau = \{t_i\}$ be a sequence of nondecreasing points on the real line. The value of the i th B-Spline of order k at x is

$$N_{i,k,\tau}(x) = (-1)^k (t_{i+k} - t_i) G_{\{t_i, \dots, t_{i+k}\}} T_x^k$$

when $t_{i+k} > t_i$. If $t_{i+k} = t_i$, then $N_{i,k,\tau}(x)$ is defined to be zero. We see that a B-spline is a linear combination of truncated power functions, and so is continuous from the right. G is the divided difference operator.

Here are some of the properties: (1) Each B-Spline has support in a finite interval, (2) The B-Splines form a partition of unity, (3) Each B-Spline is a piecewise polynomial of order k , (4) They and their derivatives can be computed recursively, (5) Given a space $P_{k,\xi,\nu}$ there exists a knot sequence τ so that the B-Splines are a basis of the space. (6) If the knot set consists of distinct points, then each B-Spline is a linear combination of shifted truncated power functions of the form $T_x^k(t_i)$, and thus is a spline in the traditional sense. That is, the k th order B-Spline, which is defined on distinct knots, has a continuous $k - 2$ order derivative. For example, if $k = 4$, then the B-spline is a cubic spline.

If $x < t_i$ then T_x is zero on the interval $[t_i, t_{i+k}]$. Hence $N_{i,k} = 0$. If $x > t_{i+k}$ then T_x is a k th order polynomial on the interval $[t_i, t_{i+k}]$. Hence its k th divided difference vanishes, and again $N_{i,k} = 0$. This shows that the support of $N_{i,k} = 0$ lies in $[t_i, t_{i+k}]$.

As an example, let us compute the B-Spline of order 1 on the knots t_0 and t_1 . We have

$$\begin{aligned} N_{0,1}(x) &= (-1)^1(t_1 - t_0)G_{t_0,t_1}T_x^1 \\ &= (t_0 - t_1)\frac{T_x^1(t_0) - T_x^1(t_1)}{t_0 - t_1} \\ &= T_x^1(t_0) - T_x^1(t_1), \end{aligned}$$

when the knots are distinct. Otherwise it is zero. $N_{0,1}(x)$ is a rectangular pulse. It equals 1 for $t_0 \leq x < t_1$, and is zero elsewhere. It is continuous from the right, but not from the left.

The generalized divided difference is defined by right derivatives when knots coincide. It follows that the B-Splines defined in this way are infinitely differentiable from the right. The B-Splines could have been defined so that they are infinitely differentiable from the left.

Let us now look at B-Splines of order 2. Let $\tau = \{0, 1, 2\}$. From the definition we find

$$\begin{aligned} N_{0,2}(x) &= T_x^2(2) - 2T_x^2(1) + T_x^2(0). \\ N_{0,2}(x) &= 0, x < 0 \\ N_{0,2}(x) &= x, 0 \leq x < 1 \\ N_{0,2}(x) &= 2 - x, 1 \leq x < 2 \\ N_{0,2}(x) &= 0, 2 \leq x. \end{aligned}$$

This is a triangular hat function with support in the interval $[0, 2]$. When knots coincide there is a loss of continuity. For example let $\tau = \{0, 0, 1\}$. Then

$$\begin{aligned} N_{0,2}(x) &= G_{\{0,0,1\}} T_x^2(0) \\ &= (T_x^2(1) - T_x^2(0)) - \frac{dT_x^2(0)}{dt} \\ &= T_x^2(1) - T_x^2(0) + 1. \end{aligned}$$

Thus

$$\begin{aligned} N_{0,2}(x) &= 0, x < 0 \\ N_{0,2}(x) &= 1 - x, 0 \leq x < 1 \\ N_{0,2}(x) &= 0, 1 \leq x. \end{aligned}$$

One order of continuity has been lost at 0.

Higher order B-splines can be calculated by recursion.

$$N_{i,k} = \frac{x - t_i}{t_{i+k-1} - t_i} N_{i,k-1} + \frac{t_{i+k} - x}{t_{i+k} - t_{i+1}} N_{i+1,k-1},$$

where a term is zero if its denominator is zero. This method of computation is known as the Cox-DeBoor algorithm.

5 B-spline Bases Function Program

This recursive C program returns a basis function value.

```

/*c+ bspln  b-spline function (recursive definition)*/
double bspln(k,x,t)
/*
  parameters:
    k-order of bspline
    x-point
    t-knot vector
  recursively calculated from lower order b-splines
  note: not all compilers support recursion
*/
double x,t[];
int k;
{
  extern double bspln();
  double zero,b,c;
  zero = 0.0;
  b = zero;

```

```

if(t[1+k-1] > t[0]) {
  if(k == 1) {
    if(x >= t[0] && x < t[1]) b = 1.0;
  }
  else {
    c = t[k-1]-t[0];
    if(c > zero) {
      b = (x-t[0])*bspln(k-1,x,&t[0])/c;
    }
    c = t[1+k-1]-t[1];
    if(c > zero) {
      b += ((t[1+k-1]-x)*bspln(k-1,x,&t[1])/c);
    }
  }
}
return b;
}

```

6 The Derivatives of a B-Spline Bases Function

We have

$$\frac{dN_{i,k}}{dx} = (-1)^k (t_{i+k} - t_i) G_{t_i, \dots, t_{i+k}} \frac{dT_x^k}{dx}.$$

And

$$\frac{dT_x^k}{dx} = (k-1)(x-t)_+^{k-1} = (k-1)T_x^{k-1}.$$

Then

$$\begin{aligned} \frac{dN_{i,k}}{dx} &= (k-1)(-1)^k (t_{i+k} - t_i) \frac{G_{t_i, \dots, t_{i+k-1}} T_x^{k-1} - G_{t_{i+1}, \dots, t_{i+k}} T_x^{k-1}}{t_i - t_{i+k}} \\ &= (k-1)(-1)^{k-1} G_{t_i, \dots, t_{i+k-1}} T_x^{k-1} - G_{t_{i+1}, \dots, t_{i+k}} T_x^{k-1} \\ &= (k-1) \left[\frac{N_i^{k-1}}{t_{i+k-1} - t_i} - \frac{N_{i+1}^{k-1}}{t_{i+k} - t_{i+k}} \right]. \end{aligned}$$

Note that if the denominator of a term is zero, then the corresponding difference operator G is the zero operator, so the term is zero. We may use this formula to compute higher derivatives. Thus

$$\frac{d^n N_{i,k}}{dx^n} = (k-1) \left[\frac{d^{n-1} N_i^{k-1} / dx^{n-1}}{t_{i+k-1} - t_i} - \frac{d^{n-1} N_{i+1}^{k-1} / dx^{n-1}}{t_{i+k} - t_{i+k}} \right].$$

7 Program For Bases Function Derivatives

This recursive C program returns a basis function derivative value.

```
/*c+ dbspln derivative of b-spline bases function*/
double dbspln(j,k,x,t)
/*
  j-jth derivative j > 0
  k-spline order (1 <= k <= 4)
  x-point
  t-knot vector
*/
double x,t[];
int k;
{
  extern double bspln();
  double zero,b,c;
  int i,km1;
  zero = 0.0;
  i = 1;
  b = 0.0;
  km1 = k-1;
  if(j == 1){
    if(t[i+k-1] > t[i-1]) {
      c = t[i+k-2]-t[i-1];
      if(c > zero) b = bspln(km1,x,&t[i-1])/c;
      c = t[i+k-1]-t[i];
      if(c > zero) b -= bspln(km1,x,&t[i])/c;
    }
  }
  if(j > 1){
    if(t[i+k-1] > t[i-1]) {
      c = t[i+k-2]-t[i-1];
      if(c > zero) b = dbspln(j-1,k-1,x,&t[i-1])/c;
      c = t[i+k-1]-t[i];
      if(c > zero) b -= dbspln(j-1,k-1,x,&t[i])/c;
    }
  }
  return(b*(k-1));
}
```

8 B-Spline Curves

A B-spline curve is defined by

$$f = \sum_{i=0}^{n-1} p_i N_{i,k}.$$

The n vectors p_0, \dots, p_{n-1} are called the control points. Given a knot set

$$t_0, t_1, \dots, t_{n+k-1},$$

the domain of the curve is $[t_{k-1}, t_n)$. The domain is an interval such that each point of the interval is in the support of k basis functions. Now the limit of the curve as t goes to t_n from the left is p_{n-1} , even when the last k knots are identical and the last basis function is discontinuous at t_n . Therefore we can extend the definition of the curve by defining

$$f(t_n) = p_{n-1},$$

in the case of k identical trailing knots. Then we may take the curve domain to be $[t_{k-1}, t_n]$. We have

$$f(t_0) = p_0,$$

$f'(t_0)$ has the direction of $p_1 - p_0$, and $f'(t_n)$ has the direction of $p_{n-1} - p_{n-2}$. Also because in this domain interval, the basis functions form a partition of unity, the curve lies in the convex hull of the control points.

9 The Derivative of a B-Spline Curve

Let f be a B-spline curve given by

$$f = \sum_{i=r}^s p_i N_{i,k},$$

then using the result of the previous section,

$$\frac{df}{dt} = \sum_{i=r}^{s+1} p'_i N_{i,k-1},$$

where

$$p'_i = (k-1) \frac{p_i - p_{i-1}}{t_{i+k-1} - t_i},$$

and where p_{r-1} and p_{s+1} are taken to be zero. Thus the derivative curve is computed by differencing the coefficients of the curve.

10 IGES Definition

The IGES k th order B-Spline is defined by the sum

$$f = \sum_{i=0}^{\rho} f_i N_{i-\mu}.$$

The B-Splines are defined on a knot sequence

$$\{t_{-\mu}, \dots, t_0, \dots, t_\eta, \dots, t_{\eta+\mu}\}.$$

The B-Spline coefficients are

$$\{f_0, f_2, \dots, f_\rho\}.$$

$$\mu = k - 1$$

is the degree of the B-Splines. The number of knots is

$$n = 2\mu + \eta + 1.$$

This sequence allows $n - k$ k th order B-Splines to be defined, so the number of control points is $n - k$. Then

$$\rho = n - k - 1 = (2\mu + \eta + 1) - (\mu + 1) - 1 = \mu + \eta - 1.$$

Each f is a homogeneous point with coordinates x, y, z . and w . f_0 is the coordinate of $N_{t_{-\mu}, k}$. The curve is parameterized on $[t_0, t_\eta]$. The partition of unity holds on this interval. The curve is defined as a rational function with the w function in the denominator.

$$c = \frac{\sum_{i=0}^{\rho} w_i P_i N_{i-\mu}}{\sum_{i=0}^{\rho} w_i N_{i-\mu}}$$

The coordinates of the control vectors are multiplied by the coordinates of the weight vector, so we define new vectors

$$x'_i = w_i x_i, y'_i = w_i y_i, z'_i = w_i z_i.$$

We may use the FORTRAN subroutine `bsplpp` to convert from the B-Spline representation to the IGES parametric representation. From the $\{x'_i\}$ we generate the A_x, B_x, C_x , and D_x .

We call subroutine `bstosp`: `call bstosp(x',n-k,tau,n,a,b,c,d,tknot,nknot)`

We compute repeat this for the other coordinates w, y' , and z' . If w is not equal to 1, then we make modifications to get a polynomial approximation to the rational function.

11 PDES/STEP Definition

(see Product Data Representation and Exchange ISO/DIS 10303-42, Integrated Generic Resources: Geometric and Topological Representation, U.S. Product Data Association c/o NCGA 2722 Merrilee Drive, Suite 200 Fairfax, VA 22031.

12 The B-Spline Dual Space

We will construct a linear functional λ_i so that

$$\lambda_i(N_J) = \delta_{ij}.$$

Lemma. Let

$$\phi_i(t) = (t - t_{i+1})(t - t_{i+2})\dots(t - t_{i+k-1})(\tau_i - t)_+^0(t_i - t_{i+k}),$$

where $t_i < \tau_i < t_{k+i}$. Then

$$G_{\{t_j, t_{j+1}, \dots, t_{j+k}\}} \phi_i = \delta_{ij}.$$

Proof

Case 1. $j > i$. We have $\phi_i(t_j) = \dots = \phi_i(t_{j+k}) = 0$. Thus the divided difference is zero.

Case 2. $i = j$. Let

$$f(t) = (t - t_{i+1})\dots(t - t_{i+k-1})(t - t_{i+k}).$$

Then $f(t) = \phi_i(t)$ for $t = t_i, t_{i+1}, \dots, t_{i+k}$. But f is a polynomial of degree k , so the $k + 1$ divided difference equals the high order coefficient, which is equal to one.

Case 3. $j < i$ Let $f(t) = (t - t_{i+1})(t - t_{i+2})\dots(t - t_{i+k-1})(t_i - t_{i+k})$ then $f(t) = \phi_i(t)$ for $t = t_j, t_{j+1}, \dots, t_{j+k}$. This is true because either a point is greater than τ_i , in which case both function values are zero at the point, or else the point is less than τ_i , where each function is equal to the same polynomial. But f is a degree $k - 1$ polynomial, so the divided difference vanishes. This completes the proof.

Now we will use a Taylor series expansion. Let

$$\psi_i(t) = (t - t_{i+1})(t - t_{i+2})\dots(t - t_{i+k-1})(t_i - t_{i+k}).$$

Writing it as a Taylor series, we have

$$\begin{aligned}\psi_i(t) &= \sum_{p=0}^{k-1} \frac{(t - \tau_i)^p}{p!} \frac{d^p \psi(\tau_i)}{dt^p} \\ &= \sum_{p=0}^{k-1} (-1)^p \frac{(\tau_i - t)^p}{p!} \frac{d^p \psi(\tau_i)}{dt^p}.\end{aligned}$$

One of the factors in the sum is a derivative of the truncated power function. We have

$$\frac{d^m}{dx^m} T_x(t)|_{\tau_i} = \frac{(\tau_i - t)^{k-m-1}}{(k-m-1)!} (\tau_i - t)_+^0.$$

Now

$$\begin{aligned}\phi_i(t) &= (t_i - t_{i+k}) \sum_{p=0}^{k-1} (-1)^p \frac{(\tau_i - t)^p}{p!} (\tau_i - t)_+^0 \frac{d^p \psi(\tau_i)}{dt^p} \\ &= (t_i - t_{i+k}) \sum_{p=0}^{k-1} (-1)^p \frac{d^{k-p-1}}{dx^{k-p-1}} T_x(t)|_{\tau_i} \frac{d^p \psi(\tau_i)}{dt^p}.\end{aligned}$$

Applying the j th finite difference operator we have

$$\begin{aligned}\delta_{ij} &= G_{\xi_j} \phi_i \\ &= (t_i - t_{i+k}) \sum_{p=0}^{k-1} (-1)^{p+k} \frac{d^{k-p-1}}{dx^{k-p-1}} (-1)^k G_{\xi_j} T_x(t)|_{\tau_i} \frac{d^p \psi(\tau_i)}{dt^p} \\ &= \lambda_i N_j,\end{aligned}$$

where the operator λ_i is defined by

$$\lambda_i f = \sum_{p=0}^{k-1} (-1)^{p+k} \frac{d^p \psi(\tau_i)}{dt^p} \frac{d^{k-p-1}}{dx^{k-p-1}} f(\tau_i).$$

13 A B-Spline Representation of the WF-Spline

The Wilson-Fowler spline is a parametric cubic interpolating a set of n points. It is parameterized by the arc length of the piecewise linear curve joining the interpolation points. The spline has continuous tangent direction and continuous curvature. The coordinate functions are only c^0 , so the B-Spline knot

multiplicity with this parameterization is 3. But it can easily be respiremetries so that they are c^1 . So it can be represented with double knots. This can be done by multiplying the parameter by a number, which is constant in each segment. Thus the lengths of the tangent vectors can be made continuous across break points. However we shall represent the spline as a general piecewise polynomial, and use knot multiplicity four. Let

$$\{s_1, s_2, \dots, s_n\}$$

be the parameter break points. Let each point be repeated four times. (last point?). Define B-Splines on this knot set. Each segment polynomial will equal a linear combination of four B-Splines based on the four repeated left segment points. Set up a four dimensional linear system whose solution is the coefficients of the four B-Splines. Do this using four interior points of the segment. The control polygon will thus have $4(n - 1)$ vertices.

14 A B-Spline Representation of a Bezier Curve

The i th B-spline bases function $N_{i,k,\tau}$, of order k (and degree $n = k - 1$), on knot set τ , is recursively defined by the Cox-DeBoor algorithm.

The algorithm takes the form

$$N_{i,k,\tau}(x) = \frac{x - t_i}{t_{i+k-1} - t_i} N_{i,k-1,\tau}(x) + \frac{t_{i+k} - x}{t_{i+k} - t_{i+1}} N_{i+1,k-1,\tau}(x),$$

where a term is zero if its denominator is zero. $N_{0,1}(x)$, is a square pulse. It takes a positive constant value for $t_0 \leq x < t_1$ and is zero elsewhere. It is continuous from the right, but not from the left.

Let τ_n be the special knot set $t_0 = t_1 = \dots = t_n = 0$, and $t_{n+1} = t_{n+2} = \dots = t_{2n+1} = 1$. Thus in the cubic case

$$\tau_3 = \{0, 0, 0, 0, 1, 1, 1, 1\}.$$

We claim that

$$N_{i,k,\tau_n} = B_i^n.$$

That is, the i th B-spline is the i th Bernstein polynomial.

To prove this we shall assume that the result is true for $k = k - 1$. We start the induction with the fact that $N_{0,1} = 1 = B_0^0$ with the knot set $\{0, 1\}$. Using the B-spline recursion formula we have

$$N_{i,k,\tau_n}(x) = xN_{i,k-1,\tau_n}(x) + (1 - x)N_{i+1,k-1,\tau_n}(x)$$

$$\begin{aligned}
&= xN_{i-1,k-1,\tau_{n-1}}(x) + (1-x)N_{i,k-1,\tau_{n-1}}(x) \\
&= xB_{i-1}^{n-1}(x) + (1-x)B_i^{n-1}(x) = B_i^n(x).
\end{aligned}$$

This completes the proof.

15 B-Spline Interpolation

Suppose we wish to approximate a function with a piecewise polynomial in B-spline form. First we must choose a knot sequence. Suppose we wish a degree 3 b-spline defined on the domain $[t_4, t_{n-3}]$. We take a knot set t_1, \dots, t_n . Let us approximate with a cubic spline and so use distinct internal knots. There will be b-spline bases functions based at $t_1, t_2, t_3, t_4, t_5, \dots, t_{n-4}$. Hence there are $n - 4$ b-spline basis functions. These require $n - 4$ interpolation points. There are $n - 6$ points in the domain: t_4, t_5, \dots, t_{n-3} . So we need two additional interpolation points. We may conveniently take these from the first interval, say

$$s_1 = t_4 + (t_5 - t_4)/3,$$

and

$$s_2 = t_4 + 2(t_5 - t_4)/3.$$

Then at each of the first four interpolation points t_4, s_1, s_2, t_5 only the B-splines N_1, N_2, N_3, N_4 have nonzero support. Thus if they are linearly independent on the four interpolation points, their coefficients may be evaluated. After they are determined then the coefficient of N_5 is determined by the interpolation point t_6 . This procedure may be continued until finally, the last B-spline function N_{n-4} is determined by the point t_{n-3} . There is problem if one sets

$$t_{n-3} = t_{n-2} = t_{n-1} = t_n.$$

For then all B-spline bases functions are zero at t_{n-3} and the set of B-spline bases functions are not linearly independent on the set of interpolation points. To solve this problem, we can move the last interpolation point to the mid-point of the last interval, or we can make the last three knot points distinct.

We can use this method to find a B-spline representation for a cubic spline computed by the traditional algorithm. Alternately this technique can be used to compute a cubic spline interpolate without specifying end conditions.

16 B-Spline Surface Patch

The B-Spline surface patch is defined by

$$f(u, v) = \sum \sum P_{i,j} N_{i,\sigma}(u) N_{j,\tau}(v).$$

The file structure expected by the program listed below is the following:

$$\left[\begin{array}{cccc} k_u & k_v & n_u & n_v \end{array} \right]$$

k_u is the u order, k_v is the v order, n_u is the number of control points in the u direction, and n_v is the number of control points in the v direction. This line is followed by the $n_u + k_u$ knots in the u direction

$$\left[\begin{array}{c} u_0 \\ u_1 \\ \dots \\ u_{n_u+k_u-1} \end{array} \right]$$

This is followed by the v knots

$$\left[\begin{array}{c} v_0 \\ v_1 \\ \dots \\ v_{n_v+k_v-1} \end{array} \right]$$

Then follows the grid of control points

$$\left[\begin{array}{c} p_{0,0} \\ p_{1,0} \\ p_{2,0} \\ \dots \\ p_{n_u-1,0} \\ p_{0,1} \\ \dots \\ p_{n_u-1,n_v-1} \end{array} \right]$$

The domain of the spline is

$$[u_{k_u-1}, v_{k_v-1}] \times [u_{n_u} - \epsilon, v_{n_v} - \epsilon]$$

The partial derivatives are

$$\frac{\partial f(u, v)}{\partial u} = \sum \sum P_{i,j} \frac{\partial N_{i,\sigma}(u)}{\partial u} N_{j,\tau}(v),$$

and

$$\frac{\partial f(u, v)}{\partial v} = \sum \sum P_{i,j} N_{i,\sigma}(u) \frac{\partial N_{j,\tau}(v)}{\partial v}.$$

They may be computed by a routine that calculates the derivative of the B-spline bases functions. The cross product of these two derivatives is a normal vector.

```

/* bssp.c 10/28/94 */
/* read a b-spline surface patch definition file */
/* produce a triangulation of the patch for program vg.c */
#include <stdio.h>
#include <math.h>
double px[2000],py[2000],pz[2000],pw[2000];
main(argc,argv)int argc;char *argv[];{
  extern int readr();
  extern void gettri();
  extern void bss3();
  extern void dvbss3();
  extern void dubss3();
  extern void rmsp();
  extern void mxmxy();
  extern void defpt();
  extern void mkunitv();
  extern void crsspr();
  double u[2000],v[2000];
  double ut[3],vt[3];
  double xmn=1.e30;
  double xmx=-1.e30;
  double ymn=1.e30;
  double ymx=-1.e30;
  double ain[6],t1,t2,te;
  double zero,ak,ai,s,x,y,z,w;
  double ua,ub,va,vb,uu,vv;
  double dx,dy,dz;
  double av[3],bv[3],cv[3];
  int nu,nv,nk,nr,k,np,j,i,n;
  int dim,l,ls,discont,outputp;
  int uord,vord,nucp,nvcp,nuk,nvk;
  char name[30],ans[1];
  FILE *in,*trif,*linef;
  if(argc != 4){
    printf("bssp bsplinesurfacefile trianglefile linefile\n");
    return(0);
  }
  in =fopen(argv[1],"r");
  if( in == NULL){
    printf(" can not find file %s \n",argv[1]);
    return(0);
  }
}

```

```

trif=fopen(argv[2],"w");
linef=fopen(argv[3],"w");
/* uord: spline order in u parameter*/
/* vord: spline order in v parameter*/
/* nucp: number of control points in u parameter*/
/* nvcp: number of control points in v parameter*/
/* write orders and number of control points*/
nr=readr(in,ain);
uord=ain[0];
vord=ain[1];
nucp=ain[2];
nvcp=ain[3];
nuk=nucp+uord;
nvk=nvcp+vord;
/* read u knots */
for(i=0;i < nuk;i++){
nr=readr(in,ain);
if(nr != 1){
printf(" error in reading u knots\n");
return(0);
}
u[i]=ain[0];
}
/* read v knots */
for(i=0;i < nvk;i++){
nr=readr(in,ain);
if(nr != 1){
printf(" error in reading v knots\n");
return(0);
}
v[i]=ain[0];
}
/* read control points */
for(j=0;j < nvcp;j++){
for(i=0;i < nucp;i++){
n=j*nucp+i;
nr=readr(in,ain);
if(nr != 3){
printf(" error in reading control points\n");
return(0);
}
px[n]=ain[0];
py[n]=ain[1];
pz[n]=ain[2];
}
}
printf(" read %d control points \n",n+1);
/* domain: */
ua=u[uord-1];
ub=u[nucp]-.00001;
va=v[vord-1];
vb=v[nvcp]-.00001;
uu=(ua + ub)/4.;
vv=(va + vb)/4.;
printf(" parameter domain: (%g,%g)X(%g,%g)\n",ua,ub,va,vb);
printf(" enter u and v \n");
while(readr(stdin,ain) == 2){

```

```

uu=ain[0];
vv=ain[1];
bss3(uu,vv,uord,vord,nucp,nvcp,u,v,px,py,pz,&x,&y,&z);
printf("point at u=%g, v=%g is %g %g %g \n",uu,vv,x,y,z);
dubss3(uu,vv,uord,vord,nucp,nvcp,u,v,px,py,pz,&dx,&dy,&dz);
printf("u derivative at u=%g, v=%g is %g %g %g \n",uu,vv,dx,dy,dz);
av[0]=dx;
av[1]=dy;
av[2]=dz;
dvbss3(uu,vv,uord,vord,nucp,nvcp,u,v,px,py,pz,&dx,&dy,&dz);
printf("v derivative at u=%g, v=%g is %g %g %g \n",uu,vv,dx,dy,dz);
bv[0]=dx;
bv[1]=dy;
bv[2]=dz;
crsspr(av,bv,cv);
mkunitv(cv);
printf("surface normal u=%g, v=%g is %g %g %g \n",uu,vv,cv[0],cv[1],cv[2]);
fprintf(linef,"%g %g %g\n",x,y,z);
fprintf(linef,"%g %g %g\n",x+cv[0],y+cv[1],z+cv[2]);
printf(" Enter u and v (<Return> to continue)\n");
}
/* triangulate patch */
nu=20;
nv=20;
for(k=1;k <= 2*nu*nv;k++){
  gettri(ua,ub,va,vb,nu,nv,k,ut,vt);
  /* printf("triangle %d\n",k);*/
  for(i=0;i<3;i++){
    bss3(ut[i],vt[i],uord,vord,nucp,nvcp,u,v,px,py,pz,&x,&y,&z);
    /* printf(" vertex: %g %g value %g %g %g\n",ut[i],vt[i],x,y,z);*/
    fprintf(trif,"%g %g %g\n",x,y,z);
  }
}
printf(" wrote file %s\n",argv[2]);
}
/*c+ bspln b-spline function (recursive definition)*/
double bspln(k,x,t)
/*
  parameters:
    k-order of bspline > 0
    x-point
    t-knot vector
  recursively calculated
  the support of the b-spline is in [t[0],t[k]]
  order= degree + 1
*/
double x,t[];int k;
{
  double zero,b,c;
  int t1,t2,i;
  zero = 0.0;
  b = zero;
  if(x < t[0])return b;
  if(x > t[k])return b;
  /* printf(" order= %d\n",k);*/
  /* printf("knots ");*/
  /* for(i=0;i <= k;i++)printf("%g ",t[i]);*/

```

```

/* printf("\n");*/
if(t[1+k-1] > t[0]) {
    if(k == 1) {
        if(x >= t[0] && x < t[1]) b = 1.0;
    }
    else{
        c = t[k-1]-t[0];
        if(c > zero){
            t1 = k-1;
            b = (x-t[0])*bspln(t1,x,&t[0])/c;
        }
        c = t[1+k-1]-t[1];
        if(c > zero) {
            t2 = k-1;
            b += ((t[1+k-1]-x)*bspln(t2,x,&t[1])/c);
        }
    }
}
return b;
}
/*
c+ readr read row of numbers, r*8
returned value is:
-1, end of file
0, no numbers
n, n is number of values read
fn-file pointer
a-array of numbers read
separate numbers by blanks.
from keyboard use: n=readr(stdin,a).
modified 9/3/91
*/
int readr(fn,a)
FILE *fn;
double *a;
{
    extern double atof();
    char b[200],c[25],d[2];
    int i,l,code,nr;
    strcpy(c,"");
    if(fgets(b,200,fn)==NULL){
        nr=-1;
    }
    else{
        nr=0;
        /* fgets puts newline character into string, subtract 1 */
        l=strlen(b)-1;
        d[0]=' ';
        for(i=0;i<l;i++){
            d[0]=b[i];
            if(d[0] != ' ')strncat(c,d,1);
            if ((d[0]==' ') || (i==l-1)){
                if(strlen(c) != 0){
                    nr=nr+1;
                    a[nr-1]=atof(c);
                    strcpy(c,"");
                }
            }
        }
    }
}

```

```

    }
  }
}
return(nr);
}
/*c+ xviewport define viewport*/
int xviewport(nf,x1,x2,y1,y2)
FILE *nf;
double x1,x2,y1,y2;
{
  char buf[50];
  double v[4];
  sprintf(buf,"v%7.5g %7.5g %7.5g %7.5g",x1,x2,y1,y2);
  rmsp(buf);
  fprintf(nf,"%s\n",buf);
  v[0]=x1;
  v[1]=x2;
  v[2]=y1;
  v[3]=y2;
  return 0;
}
/*
c+ xwindow define world window
*/
int xwindow(nf,x1,x2,y1,y2)
FILE *nf;
double x1,x2,y1,y2;
{
  char buf[50];
  double w[4];
  sprintf(buf,"w%7.5g %7.5g %7.5g %7.5g",x1,x2,y1,y2);
  rmsp(buf);
  fprintf(nf,"%s\n",buf);
  w[0]=x1;
  w[1]=x2;
  w[2]=y1;
  w[3]=y2;
  return 0;
}
/*
c+ xmove graphics move
*/
int xmove(nf,x,y)
FILE *nf;
double x,y;
{
  char buf[50];
  sprintf(buf,"m%7.5g %7.5g",x,y);
  rmsp(buf);
  fprintf(nf,"%s\n",buf);
  return 0;
}
/*
c+ xdraw graphics draw
*/
int xdraw(nf,x,y)
FILE *nf;

```

```

double x,y;
{
char buf[50];
sprintf(buf,"d%7.5g %7.5g",x,y);
rmsp(buf);
fprintf(nf,"%s\n",buf);
return 0;
}
/*c+ rmsp remove extra spaces in string*/
void rmsp(buf)char *buf;{
int p,q,n,i,k,sp;
n=strlen(buf);
sp=' ';
p=-1;
k=0;
for(i=0;i<n;i++){
q=buf[i];
if(q != sp){
if((p == sp) && (k != 0)){
buf[k]=sp;
k=k+1;
}
buf[k]=q;
k=k+1;
}
p=q;
}
buf[k]='\0';
}
void mxmny(x,y,xmn,xmx,ymn,ymx)
double x,y,*xmn,*xmx,*ymn,*ymx;{
if(x < *xmn)*xmn=x;
if(x > *xmx)*xmx=x;
if(y < *ymn)*ymn=y;
if(y > *ymx)*ymx=y;
}
void defpt(n,x,y,z)
int n;double x,y,z;{
px[n]=x;
py[n]=y;
pz[n]=z;
}
/*c+ gettri return kth triangle in a triangulated domain*/
void gettri(umn,umx,vmn,vmx,nu,nv,k,u,v)
/*
umn,umx,vmn,vmx rectangular domain definition
nu,nv domain divided into nu*nv rectangles
which are divided into two triangles
giving 2*nu*nv triangles.
k selected triangle, 1 <= k <= 2*nu*nv
u,v coordinates of 3 vertices of returned triangle.
*/
double umn,umx,vmn,vmx,u[],v[];
int nu,nv,k;
{
double du,dv;
int m,j,i;

```

```

int odd;
du = (umx-umn)/nu;
dv = (vmx-vmn)/nv;
odd = (k % 2) == 1;
if(odd) {
    m = k/2;
    j = m/nu;
    i = m-nu*j;
    u[0] = umn+i*du;
    v[0] = vmn+j*dv;
    u[1] = u[0]+du;
    v[1] = v[0];
    u[2] = u[1];
    v[2] = v[1]+dv;
}
else {
    m = k/2-1;
    j = m/nu;
    i = m-nu*j;
    u[0] = umn+i*du;
    v[0] = vmn+j*dv;
    u[1] = u[0]+du;
    v[1] = v[0]+dv;
    u[2] = u[0];
    v[2] = v[1];
}
}
}
/*c+ bss3 point on b-spline surface patch*/
void bss3(uu,vv,uord,vord,nucp,nvcv,u,v,px,py,pz,x,y,z)
double uu,vv,u[],v[],px[],py[],pz[],*x,*y,*z;
int uord,vord,nucp,nvcv;
/*input: */
/* (uu,vv) parameters of point on surface*/
/* uord b-spline order for u parameter*/
/* vord b-spline order for v parameter*/
/* nucp number of u control points*/
/* nvcv number of v control points*/
/* u nondecreasing sequence of nucp+uord u knots*/
/* (starting at u[0])*/
/* v nondecreasing sequence of nvcv+vord v knots*/
/* px x-coordinate of the nucp*nvcv control points*/
/* py y-coordinate of the nucp*nvcv control points*/
/* pz z-coordinate of the nucp*nvcv control points*/
/* the m=j*nucp+i control point corresponds to the*/
/* knot (u(i),v(j)).
/*output: */
/* x x-coordinate of point on surface*/
/* y y-coordinate of point on surface*/
/* z z-coordinate of point on surface*/
/* surface definition:*/
/* s(u,v)=sum p(j nucp+i) b(uord,u,i) b(vord,v,j)*/
/* where 0 <= i < nucp, 0 <= j < nvcv*/
/* the order of a b-spline is 1+degree.*/
/* usual domain is the product set of closed intervals:*/
/* [u_{uord-1},u_nucp] \times [v_{vord-1},v_nucp] */
{
extern double bspln();

```

```

double bu,bv;
int i,j;
*x = *y = *z = 0.0;
for(i=0;i < nucp;i++){
    bu = bspln(uord,uu,&u[i]);
/*    printf(" bu(%d) = %g\n",i,bu);*/
    for(j=0;j < nvcp;j++){
        bv = bspln(vord,vv,&v[j]);
/*    printf(" bv(%d) = %g\n",j,bv);*/
        *x += bu*bv*px[j*nucp+i];
        *y += bu*bv*py[j*nucp+i];
        *z += bu*bv*pz[j*nucp+i];
    }
}
}
}
/*c+ dubss3 u partial derivative of b-spline surface patch*/
void dubss3(uu,vv,uord,vord,nucp,nvcp,u,v,px,py,pz,dx,dy,dz)
double uu,vv,u[],v[],px[],py[],pz[],*dx,*dy,*dz;
int uord,vord,nucp,nvcp;
/*input: */
/* (uu,vv) parameters of point on surface*/
/* uord b-spline order for u parameter*/
/* vord b-spline order for v parameter*/
/* nucp number of u control points*/
/* nvcp number of v control points*/
/* u nondecreasing sequence of nucp+uord u knots*/
/* (starting at u[0])*/
/* v nondecreasing sequence of nvcp+vord v knots*/
/* px x-coordinate of the nucp*nvcp control points*/
/* py y-coordinate of the nucp*nvcp control points*/
/* pz z-coordinate of the nucp*nvcp control points*/
/* the m=j*nucp+i control point corresponds to the*/
/* knot (u(i),v(j)).
/*output: */
/* dx x-coordinate of vector tangent to surface*/
/* dy y-coordinate of vector tangent to surface*/
/* dz z-coordinate of vector tangent to surface*/
/* surface definition:*/
/* s(u,v)=sum p(j nucp+i) b(uord,u,i) b(vord,v,j)*/
/* where 0 <= i < nucp, 0 <= j < nvcp*/
/* the order of a b-spline is 1+degree.*/
/* usual domain is the product set of closed intervals:*/
/* [u_{uord-1},u_nucp] \times [v_{vord-1},v_nucp] */
{
extern double bspln();
extern double dbspln();
double bu,bv;
int i,j;
*dx = *dy = *dz = 0.0;
for(i=0;i < nucp;i++){
    bu = dbspln(1,uord,uu,&u[i]);
/*    printf(" bu(%d) = %g\n",i,bu);*/
    for(j=0;j < nvcp;j++){
        bv = bspln(vord,vv,&v[j]);
/*    printf(" bv(%d) = %g\n",j,bv);*/
        *dx += bu*bv*px[j*nucp+i];
        *dy += bu*bv*py[j*nucp+i];
    }
}
}

```

```

    *dz += bu*bv*pz[j*nucp+i];
  }
}
}
/*c+ dbspln derivative of b-spline bases function*/
double dbspln(j,k,x,t)
/*
j-jth derivative j > 0
k-spline order (1 <= k <= 4)
x-point
t-knot vector
*/
double x,t[];
int k;
{
  extern double bspln();
  double zero,b,c;
  int i,km1;
  zero = 0.0;
  i = 1;
  b = 0.0;
  km1 = k-1;
  if(j == 1){
    if(t[i+k-1] > t[i-1]) {
      c = t[i+k-2]-t[i-1];
      if(c > zero) b = bspln(km1,x,&t[i-1])/c;
      c = t[i+k-1]-t[i];
      if(c > zero) b -= bspln(km1,x,&t[i])/c;
    }
  }
  if(j > 1){
    if(t[i+k-1] > t[i-1]) {
      c = t[i+k-2]-t[i-1];
      if(c > zero) b = dbspln(j-1,k-1,x,&t[i-1])/c;
      c = t[i+k-1]-t[i];
      if(c > zero) b -= dbspln(j-1,k-1,x,&t[i])/c;
    }
  }
  return(b*(k-1));
}
/*c+ dvbss3 v partial derivative of b-spline surface patch*/
void dvbss3(uu,vv,uord,vord,nucp,nvcp,u,v,px,py,pz,dx,dy,dz)
double uu,vv,u[],v[],px[],py[],pz[],*dx,*dy,*dz;
int uord,vord,nucp,nvcp;
/*input: */
/* (uu,vv) parameters of point on surface*/
/* uord b-spline order for u parameter*/
/* vord b-spline order for v parameter*/
/* nucp number of u control points*/
/* nvcp number of v control points*/
/* u nondecreasing sequence of nucp+uord u knots*/
/* (starting at u[0])*/
/* v nondecreasing sequence of nvcp+vord v knots*/
/* px x-coordinate of the nucp*nvcp control points*/
/* py y-coordinate of the nucp*nvcp control points*/
/* pz z-coordinate of the nucp*nvcp control points*/
/* the m=j*nucp+i control point corresponds to the*/

```

```

/*      knot (u(i),v(j)).
/*output: */
/*      dx  x-coordinate of vector tangent to surface*/
/*      dy  y-coordinate of vector tangent to surface*/
/*      dz  z-coordinate of vector tangent to surface*/
/*      surface definition:*/
/*      s(u,v)=sum p(j nucp+i) b(uord,u,i) b(vord,v,j)*/
/*      where 0 <= i < nucp, 0 <= j < nvcv*/
/*      the order of a b-spline is 1+degree.*/
/*      usual domain is the product set of closed intervals:*/
/*      [u_{uord-1},u_nucv] \times [v_{vord-1},v_nucv] */
{
extern double bspln();
extern double dbspln();
double bu,bv;
int i,j;
*dx = *dy = *dz = 0.0;
for(i=0;i < nucv;i++){
    bu = bspln(uord,uu,&u[i]);
/*  printf(" bu(%d) = %g\n",i,bu);*/
    for(j=0;j < nvcv;j++){
        bv = dbspln(1,vord,vv,&v[j]);
/*  printf(" bv(%d) = %g\n",j,bv);*/
        *dx += bu*bv*px[j*nucv+i];
        *dy += bu*bv*py[j*nucv+i];
        *dz += bu*bv*pz[j*nucv+i];
    }
}
}
/*c+ mkunitv scale a vector to have unit length */
void mkunitv(cv)double cv[];
{
double a;
int i;
a=sqrt(cv[0]*cv[0]+cv[1]*cv[1]+cv[2]*cv[2]);
for(i=0;i< 3 ;i++){
    cv[i]= cv[i]/a;
}
}
/*c+ crsspr cross product of two vectors*/
void crsspr(a,b,c)
double a[], b[], c[];
{
/*
input:
a, b - 3d vectors
output:
c - 3d vector, c= a X b
*/
c[0] = a[1] * b[2] - a[2] * b[1];
c[1] = a[2] * b[0] - a[0] * b[2];
c[2] = a[0] * b[1] - a[1] * b[0];
}

```