Sparse Smooth Connection between Bézier/Bspline Curves

Chandrajit Bajaj and Guoliang Xu
Computer Sciences Department
Purdue University
West Lafayette, IN 47907

CSD-TR-94-045
June, 1994

Sparse Smooth Connection between Bézier/Bspline Curves

Chandrajit Bajaj
Department of Computer Science
Purdue University
West Lafayette, IN 47907-1398, USA
bajaj@cs.purdue.edu

Guoliang Xu
Department of Computer Science
Purdue University
West Lafayette, IN 47907-1398, USA
xuguo@cs.purdue.edu

June 20, 1994

Introduction

Often in interactive font design, free-form sketching and input path specification for graphics animation, one is faced with the problem of connecting two Bézier or Bspline polynomial curves with a smooth, piecewise transition polynomial curve achieving prescribed continuity at the two end points. Furthermore one desires the transition polynomial curve to have the fewest number of pieces. In this paper we address this issue by solving the following two problems:

Smooth Connection problem: Given two polynomials $P : [a, b] \to \mathbb{R}$ and $Q : [c, d] \to \mathbb{R}$ of degree n with $b < c$. Find a piecewise polynomial $R : [b, c] \to \mathbb{R}$ of degree n, such that (1°) R is $C^{n-\mu}$ continuous in (b, c) for a given integer μ with $1 \le \mu \le n$. (2°) P and R join at b with $C^{n-\mu_1}$ continuity for a given integer μ_1 with $1 \le \mu_1 \le n$. (3°) R and Q joint at c with $C^{n-\mu_2}$ continuity for a given integer μ_2 with $1 \le \mu_2 \le n$.

Sparse Smooth Connection problem: In addition to the above conditions (1°), (2°) and (3°), we require that (4°) R has the fewest number of segments.

As an example, the smooth composite function (P, R, Q) may be a single B-spline. It is obvious that there are possibly infinite ways to join any two polynomials with prescribed continuity. Our goal here is not only to achieve a smooth join but also to make the join as *simple* as possible. By *simple* we mean that the polynomial R is to be determined, as far as possible, from P and Q.

The solution to both the above problems are derived by the use of blossoming (see [2, 3]). For a given degree n polynomial $F : \mathbb{R} \to \mathbb{R}$, the blossom of F, denoted as $f = B(F)$, is an n-affine symmetric function satisfying $f(u, ..., u) = F(u)$. A function $f : \mathbb{R} \to \mathbb{R}$ is called *affine* if it preserves affine combinations, that is, if f satisfies $f(\sum_i a_i u_i) = \sum_i a_i f(u_i)$ for all real numbers $a_1, \cdots, a_k$, $u_1, \cdots, u_k \in \mathbb{R}$ with $\sum_i a_i = 1$. A function $f : \mathbb{R}^n \to \mathbb{R}$ is called *n-affine* if it is an affine function on each individual argument with the others held fixed. Finally, a function $f : \mathbb{R}^n \to \mathbb{R}$ is called *symmetric* if f keeps its value under any permutation of its arguments.

Solution of the Smooth Connection Problem

Lemma 1. *Let AS_n be the set of all n-affine symmetric functions, $t_1 \le \ldots \le t_n < t_{n+1} \le \ldots \le t_{2n}$. Then the map $M : f \in AS_n \to \{f(t_\ell, t_{\ell+1}, \ldots, t_{\ell+n-1})\}_{\ell=1}^{n+1} \in \mathbb{R}^{n+1}$ is a one to one map between AS_n and $\mathbb{R}^{n+1}$*

Proof: It is obvious that M is a linear map and by $M(f) = (0, \cdots, 0)$ we can prove that $f = 0$. In fact, by the progressive de Casteljau algorithm (see [2]), we have $f(x_1, \ldots, x_n) = 0$ i.e., $f = 0$. Now, the only thing left to be proved is that M is invertible, that is, given $(b_1, b_2, \ldots, b_{n+1}) \in \mathbb{R}^{n+1}$, there exist a $f \in AS_n$, such that

$M(f) = (b_1, \ldots, b_{n+1})$. This $f = f_1^n$ can be constructed by the following progressive de Casteljau algorithm:

$$
\begin{aligned}
f_i^0() &= b_i, \quad i = 1, 2, \ldots, n+1 \\
f_i^r(x_1, \ldots, x_r) &= \tfrac{t_{n+i}-x_r}{t_{n+i}-t_{i+r-1}} f_i^{r-1}(x_1, \ldots, x_{r-1}) \\
&+ \tfrac{x_r - t_{i+r-1}}{t_{n+i}-t_{i+r-1}} f_{i+1}^{r-1}(x_1, \ldots, x_{r-1}), \quad i = 1, 2, \ldots, n+1-r
\end{aligned}
$$

for $r = 1, \cdots, n$(see Theorem 7.1 of [2]). $\diamond$

Lemma 2. *Smooth connection problem always has solution.*

Proof: We prove this lemma via a constructive approach.

(i) If $n+1 \le \mu_1 + \mu_2$, then the piecewise polynomial R to be determined degenerates to a single segment, and R can be determined by using the Hermite interpolation conditions:

$$
\begin{aligned}
R^{(i)}(b) &= P^{(i)}(b), \quad i = 0, 1, \ldots, n - \mu_1 \\
R^{(i)}(c) &= Q^{(i)}(c), \quad i = 0, 1, \ldots, n - \mu_2
\end{aligned} \tag{1}
$$

If $n+1 = \mu_1 + \mu_2$, the solution is unique. If $n+1 < \mu_1 + \mu_2$, there is no uniqueness. If we let R to be degree $2n - \mu_1 - \mu_2 + 1 (< n)$. Then we have uniqueness.

(ii) If $n+1 > \mu_1 + \mu_2$, equation (1) has no solution in general. Here we construct a B-spline $F(x) : [a, d] \to \mathbb{R}$ such that

$$
F(x)|_{[a,b]} = P(x), \qquad F(x)|_{[c,d]} = Q(x) \tag{2}
$$

and $R(x) = F(x)|_{[b,c]}$ satisfies the conditions (1°), (2°) and (3°). Let

$$
\begin{aligned}
T = (t_0 = \ldots = t_n < t_{n+1} = \ldots = t_{n+\mu_1} < t_{n+\mu_1+1} \le \cdots \le \\
t_{2n-\mu_2+1} < t_{2n-\mu_2+2} = \ldots = t_{2n+1} < t_{2n+2} = \ldots = t_{3n+2})
\end{aligned}
$$

where $t_n = a, \quad t_{n+1} = b, \quad t_{2n+1} = c, \quad t_{2n+2} = d$ and $t_{n+\mu_1+1}, \ldots, t_{2n-\mu_2+1}$ are chosen such that each of them has multiplicity $\le \mu$ in T. Let $\{N_\ell^n(x)\}_{\ell=0}^{2n+1}$ be the normalized B-spline bases over T and let

$$
\begin{aligned}
d_\ell = f_1(t_{\ell+1}, \ldots t_{\ell+n}), \quad \ell = 0, 1, \ldots, n \\
d_\ell = f_2(t_{\ell+1}, \ldots t_{\ell+n}), \quad \ell = n+1, \ldots, 2n+1
\end{aligned}
$$

where $f_1 = B(P)$, $f_2 = B(Q)$ are the *blossoms* of P and Q, respectively. Then $F(x) = \sum_{\ell=0}^{2n+1} d_\ell N_\ell^n(x)$ is the required Bspline (see Theorem 3.4 of [3]). In fact, $F(x)$ is $C^{n-\mu_1}$ and $C^{n-\mu_2}$ continuous at b and c respectively, since b has multiplicity μ_1 and c has multiplicity μ_2. Furthermore, since $t_{n+\mu_1+1}, \ldots, t_{2n-\mu_2+1}$ have multiplicity $\le \mu$, $f(x)$ is $C^{n-\mu}$ continuous on (b, c). Now we only need to show that condition (2) is satisfied. From Theorem 3.4 of [3], we have

$$
\begin{aligned}
d_\ell &= B(F|_{[a,b]})(t_{\ell+1}, \ldots, t_{\ell+n}), \quad \ell = 0, 1, \ldots, n \\
d_\ell &= B(F|_{[c,d]})(t_{\ell+1}, \ldots, t_{\ell+n}), \quad \ell = n+1, \ldots, 2n+1
\end{aligned}
$$

Hence

$$
\begin{aligned}
f_1(t_{\ell+1}, \ldots, t_{\ell+n}) &= B(F|_{[a,b]})(t_{\ell+1}, \ldots, t_{\ell+n}), \quad \ell = 0, 1, \ldots, n \\
f_2(t_{\ell+1}, \ldots, t_{\ell+n}) &= B(F|_{[c,d]})(t_{\ell+1}, \ldots, t_{\ell+n}), \quad \ell = n+1, \ldots, 2n+1
\end{aligned}
$$

Since f_1, f_2, $B(F|_{[a,b]})$ and $B(F|_{[c,d]})$ are in AS_n, it follows from Lemma 1 that $f_1 = B(F|_{[a,b]})$, $f_2 = B(F|_{[c,d]})$, and then $P = F|_{[a,b]}$, $Q = F|_{[c,d]}$.

$\diamond$

In the above proof we insert $n + 1 - (\mu_1 + \mu_2)$ knots in (b, c). Hence R has at most $n + 2 - (\mu_1 + \mu_2)$ pieces. Since the $t_{n+\mu_1+1}, \ldots, t_{2n-\mu_2+1}$ knots can be arbitrarily chosen under the required conditions, R is not unique. We therefore have the following corollary

Corollary 3. *The Sparse Smooth Connection problem always has a solution.*

The Computation of the Sparse Smooth Connection Polynomial

The proof of Lemma 2 has already provides a way to compute the smooth transition polynomial R. Furthermore this uses only the information that comes from P and Q and some inserted knots. However the number of pieces of R may not be minimal. In order to get a sparse connection polynomial we intend to insert the least number of knots. As in the discussion above, there are two possible cases.

(i) If $n+1 \leq \mu_1 + \mu_2$, the problem is reduced to Hermite interpolation problem as before. One segment is enough to connect the two given polynomials. Then the number of segments is minimum. Now we give a B-spline representation of the composite function. Let

$$\begin{aligned} T &= (t_0 = \ldots = t_n < t_{n+1} = \ldots = t_{n+\mu_1} < t_{n+\mu_1+1} \\ &= \ldots = t_{n+\mu_1+\mu_2} < t_{n+\mu_1+\mu_2+1} = \ldots = t_{2n+\mu_1+\mu_2+1}) \end{aligned} \tag{3}$$

and $\{N_\ell^n(x)\}_{\ell=0}^{n+\mu_1+\mu_2}$ be the normalized B-spline bases over T. Then $F(x) = \sum_{\ell=0}^{n+\mu_1+\mu_2} d_\ell \; N_\ell^n(x)$ is the required function, where

$$\begin{array}{ll} d_\ell = f_1(t_{\ell+1}, \ldots t_{\ell+n}), & \ell = 0, 1, \ldots, n \\ d_\ell \text{ are free}, & \ell = n+1, \ldots, \mu_1 + \mu_2 - 1 \\ d_\ell = f_2(t_{\ell+1}, \ldots t_{\ell+n}), & \ell = \mu_1 + \mu_2, \ldots, n + \mu_1 + \mu_2 \end{array} \tag{4}$$

(ii) If $n+1 > \mu_1 + \mu_2$, then the computation of the inserted knots proceeds as following, with i increasing from 0 to $n+1-(\mu_1+\mu_2)$

(a) Let

$$\begin{aligned} T_i &= (t_0 = \ldots = t_n < t_{n+1} = \ldots = t_{n+\mu_1} < x_1 \leq \ldots \leq x_i < t_{n+\mu_1+i+1} \\ &= \ldots = t_{n+\mu_1+\mu_2+i} < t_{n+\mu_1+\mu_2+i+1} = \ldots = t_{2n+\mu_1+\mu_2+i+1}) \end{aligned} \tag{5}$$

where $t_n = a, \;\; t_{n+1} = b, \;\; t_{n+\mu_1+\mu_2+i} = c, \;\; t_{n+\mu_1+\mu_2+i+1} = d$ and $x_1, \ldots, x_i$ are the knots to be determined and satisfying the following conditions:

$$\begin{array}{l} b < x_j < c \\ x_j \text{ has multiplicity} \leq \mu \text{ in } T_i \end{array} \tag{6}$$

(b) For $\ell = i + \mu_1 + \mu_2, \ldots, n$, the de Boor points (see [3]) d_ℓ are determined satisfying conditions from both P and Q. These double conditions leads to the following equations for unknowns $x_1, \ldots, x_i$

$$\begin{array}{l} B(P)(t_{\ell+1}, \ldots, t_{n+\mu_1}, x_1, \ldots, x_i, t_{n+\mu_1+i+1}, \ldots, t_{\ell+n}) = \\ B(Q)(t_{\ell+1}, \ldots, t_{n+\mu_1}, x_1, \ldots, x_i, t_{n+\mu_1+i+1}, \ldots, t_{\ell+n}) \end{array}$$

for $\ell = i + \mu_1 + \mu_2, \ldots, n$, or

$$g_\ell(x_1, \ldots, x_i) := B(P-Q)(t_{\ell+1}, \ldots, t_{n+\mu_1}, x_1, \ldots, x_i, t_{n+\mu_1+i+1}, \ldots, t_{\ell+n}) = 0$$

for $\ell = i + \mu_1 + \mu_2, \ldots, n$. There are $n+1-(i+\mu_1+\mu_2)$ equations and i unknowns. The ideal cases (a unique solution is expected) are $i = n+1-(i+\mu_1+\mu_2)$ or $i = \frac{n+1-(\mu_1+\mu_2)}{2}$. Comparing with the proof of Lemma 2, in which we insert $n+1-(\mu_1+\mu_2)$ knots, this ideal case will reduce the number of the inserted knots to half. For example, if $n = 3$ (cubic), $\mu_1 = \mu_2 = 1$(C^2 continuity), then $i = 1$. If $\mu_1 = \mu_2 = 2$(C^1 continuity), then $i = 0$. If $n = 5$, $\mu_1 = \mu_2 = 2$(C^3 continuity), $i = 1$. If $\mu_1 = \mu_2 = 1$(C^4 continuity), $i = 2$.

Let $P(x) = \sum_{j=0}^n a_j x^j$, $Q(x) = \sum_{j=0}^n b_j x^j$. Then $B(P-Q)(u_1, \cdots, u_n) = \sum_{j=0}^n (a_j - b_j) / \binom{n}{j} \sigma_{jn}(u_1, \cdots, u$ where $\sigma_{jn}(u_1, \cdots, u_n)$ is the j-th n-variable elementary symmetric function[1]. Therefore $g_\ell(x_1, \ldots, x_i)$

can be written as $g_\ell = \sum_{j=0}^{i} a_j^{(\ell)} \sigma_{ji}(x_1, \ldots, x_i)$. Let $\sigma_j = \sigma_{ji}(x_1, \ldots, x_i)$ be the unknowns, $j = 1, 2, \ldots, i$, $\sigma_0 = 1$. We thus have the following system of linear equation

$$\begin{bmatrix} a_1^{(i+\mu_1+\mu_2)} & a_2^{(i+\mu_1+\mu_2)} & \cdots & a_i^{(i+\mu_1+\mu_2)} \\ a_1^{(i+\mu_1+\mu_2+1)} & a_2^{(i+\mu_1+\mu_2+1)} & \cdots & a_i^{(i+\mu_1+\mu_2+1)} \\ \vdots & \vdots & & \vdots \\ a_1^{(n)} & a_2^{(n)} & \cdots & a_i^{(n)} \end{bmatrix} \begin{bmatrix} \sigma_1 \\ \sigma_2 \\ \vdots \\ \sigma_i \end{bmatrix} = - \begin{bmatrix} a_0^{(i+\mu_1+\mu_2)} \\ a_0^{(i+\mu_1+\mu_2+1)} \\ \vdots \\ a_i^{(n)} \end{bmatrix} \tag{7}$$

(c) If Equation (7) has no solution, increase i by 1, until it has a solution (may have many solutions). Let $[\sigma_1, \ldots, \sigma_i]^T$ be a solution of (7). Form a polynomial equation

$$h(x) := \sum_{k=0}^{i} (-x)^{i-k} \sigma_k = 0 \tag{8}$$

If all the roots x_j of $h(x)$ are real, and they satisfy (6), then we get the required knots x_j. Otherwise, we increase i until the required knots are obtained. If (7) has many solutions, a closed form of the solution of (8) is helpful to get the required solution. If $i < 5$, the closed form of the root x_j are available.

The case $i = 0$ needs separate consideration, since the equation (7) and (8) are degenerate. In this case g_ℓ are constants. If they are all zero, then we do not need to insert knots in (b, c) and the de Boor points are computed by (4), but no degree of freedom is left. If not all g_ℓ are zero, we need to consider the next i.

Since we wish to find the solution x_j's that satisfy condition (6), we solve Equation (8) for σ_k that satisfies the following necessary condition

$$\binom{i}{k} b^k < \sigma_k < \binom{i}{k} c^k, \quad k = 1, 2, \cdots, i \tag{9}$$

(d) Let $t_{n+\mu_1+j} = x_j$ for $j = 1, \cdots, i$. Let

$$\begin{aligned} d_\ell &= f_1(t_{\ell+1}, \ldots t_{\ell+n}), \quad \ell = 0, 1, \ldots, n \\ d_\ell &= f_2(t_{\ell+1}, \ldots t_{\ell+n}), \quad \ell = n+1, \ldots, n+\mu_1+\mu_2+i \end{aligned} \tag{10}$$

Then similar to the proof of Lemma 2, we know that the Bspline function $F(x) = \sum_{\ell=0}^{n+\mu_1+\mu_2+i} d_\ell N_\ell^n(x)$ is what we require, where $\{N_\ell^n(x)\}_{\ell=0}^{n+\mu_1+\mu_2+i}$ is the normalized B-spline bases over T_i.

Psuedocode of the Algorithm

We present psuedocode for the above algorithm of computing Sparse Smooth Connection polynomials. Here we assume we have (by now standard) library procedures for solving a linear equation and for finding the real roots of a polynomial.

Sparse Smooth Connection Algorithm
P *is the input coefficients array of the polynomial P in power bases*
Q *is the input coefficients array of the polynomial Q in power bases*
A, B *are the input end points of interval [a,b]*
C, D *are the input end points of interval [c,d]*
N *is the degree of the given polynomials*
MU1 *is the input continuity at b*
MU2 *is the input continuity at c*

MU *is the input continuity in (b, c)*
D *is the output coefficients array of the de Boor points* d_l
Knots *is the output inserted knots in (b, c)*

```
I is the output number of inserted knots
    I = 0
    compute the blossoming of P, Q and P - Q
    for j = 0 to N step 1
        P(j) = P(j)/(N choose j)     Q(j) = Q(j)/(N choose j)     C(j) = P(j) - Q(j)
    next j
    form knots T, see (3)
    for j = 0 to 2N+MU1 + MU2 +I step 1
       if j ≤ N then     T(j) =A
       else  if j ≤ N + MU1 then     T(j) =B
       else  if j ≤ N + MU1 + MU2 then     T(j) =C
       else   T(j) =D
       end if
    next  j
    if N + 1 ≤ MU1 + MU2 then
       compute d_l, by formulas (4)
       for l = 0 to N+MU1 + MU2 step 1
          for j = 1 to N step 1
              Point(j) = T(l+j)
          next  j
          if l ≤ N then
             call EVALUATE(P, N, Point, N, Coeffout)
             D(l) = Coeffout(0)
          else if l ≥ MU1 + MU2 then
             call EVALUATE(Q, N, Point, N, Coeffout)
             D(l) = Coeffout(0)
          else
             D(l) are free , set to zero
          end if
       next  l
    else
       for i = 1 to N + 1 - (MU1 + MU2) step 1
          for l = i + MU1 + MU2 to N step 1
             for j = 1 to N - i step 1
                 Point(j) = T(l+j)
             next  j
             call EVALUATE(C, N, Point, N-i, Coeffout)
             for k = 1 to i step 1
                Matrix(l - i - MU1 - MU2, k - 1) = Coeffout(k)
             next k
             Lefthand(l - i - MU1 - MU2) = - Coeffout(0)
          next l
          call LINEARSOLVER(Matrix, Lefthand, Solution)
          call POLYSOLVER(Solution, i, Knots)
          if all Knots satisfy the condition (6) then goto L
       next i
       form knots T_i, see (5)
     L: I = i
       for j = 0 to 2N+MU1 + MU2 + i + 1 step 1
          if j ≤ N then     T(j) =A
          else  if j ≤ N + MU1 then     T(j) =B
          else  if j ≤ N + MU1 + i then     T(j) = Knots(j - N - MU1)
          else  if j ≤ N + MU1 + MU2 + i then  6  T(j) =C
          else   T(j) = D
          end if
       next  j
       compute the de Boor point d_l by (10))
       for l = 0 to N step 1
```

Procedure to evaluate an n-affine symmetric function

```
procedure EVALUATE(Coeffin, N, Point, M, Coeffout)
Coeffin is the input coefficients array
N − 1 is the number of coefficients
Point is the input evaluating points array
M is the number of evaluating points
Coeffout is the output coefficients array
    for j = 0 to N step 1
        Coeffout(j) = Coeffin(j)
    next j
    for k = 0 to M − 1 step 1
        for j = 1 to N − k step 1
            Coeffout(j−1) = Coeffout(j−1) + Point(k)*Coeffout(j)
        next j
    next k
    return
```

References

[1] C. Chrystal. *Algebra, Part I, 7th ed.* Chelsea Pub. Company, New York, 1964.

[2] L. Ramshaw. Blossoms are Polar Forms. *Computer-Aided Geometric Design*, 6:323–358, 1989.

[3] Seidel, H-P. A New Multiaffine Approach to B-splines. *Computer Aided Geometric Design*, 6:23–32, 1989.