In CAGD, Vol 12, No 1, February 1995; corrected lines marked with !!!

Short Communication

# Computing values and derivatives of Bézier and B-spline tensor products

Stephen Mann

*Computer Science Department, University of Waterloo, Waterloo, Ontario, N2L 3G1, Canada*

Tony DeRose

*Department of Computer Science and Engineering, University of Washington, Seattle, Washington, 98195, U.S.A.*

**Abstract:** We give an efficient algorithm for evaluating Bézier and B-spline tensor products for both positions and normals. The algorithm is an extension of a method for computing the position and tangent to a Bézier curve, and is asymptotically twice as fast as the standard bilinear algorithm.

**Keywords:** Tensor product surfaces, blossoms, evaluation, rendering

**Abbreviated title:** Tensor Product Evaluation

Many applications, such as rendering a surface using Phong shading, require evaluating both the value and derivatives of a surface. Repeated bilinear interpolation can be used to compute values and derivatives of $n \times n$ tensor product Bézier surfaces (see Figure 1a and Farin [Far93]). The final computed point is a point on the surface, and the points in the next to last step can be used to calculate the derivatives of the surface. However, this algorithm cannot be used for an $n \times m$ tensor product surface with $m \neq n$.

In this paper we develop an algorithm to handle the general case of arbitrary $m$ and $n$. Our approach is to run the univariate version of de Casteljau's algorithm successively in each parametric direction. Each time, we stop the evaluation "one short" of completion. The resulting multi-linear function is then evaluated to compute both the value and the derivatives.

The algorithm is illustrated for a tensor product Bézier surface by the following pseudo-Pascal code:

(* Given the control points $P_{i,j}$ for a $n \times m$ tensor product
Bezier patch $F$ and a pair of parameter values $u$ and $v$,
return the value and partial derivatives of $F$ at $u, v$ *)

EvalTensorProduct(IN: $P_{i,j}$, $n$, $m$, $u$, $v$; OUT: $F$, $\frac{\partial F}{\partial u}$, $\frac{\partial F}{\partial v}$)

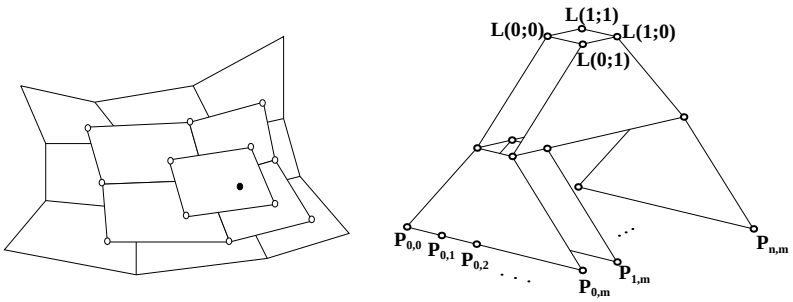


Figure 1: a) Bilinear interpolation. b) Data flow diagram of algorithm.

(* Run de Casteljau's curve algorithm on the rows of the net *)
for $i := 0$ to $n$ do (* for each row *)
  for $k := 1$ to $m - 1$ do
    for $j := 0$ to $m - k$ do
      $P_{i,j} := (1 - v) * P_{i,j} + v * P_{i,j+1}$;

(* Now run de Casteljau on the first two columns of the net *)
for $j := 0$ to 1 do (* for each column *)
  for $k := 1$ to $n - 1$ do
    for $i := 0$ to $n - k$ do
      $P_{i,j} = (1 - u) * P_{i,j} + u * P_{i+1,j}$;

(* now we have a bilinear patch in $P_{i,j}$ *)
$L_{u,0} := (1 - u) * P_{0,0} + u * P_{1,0}$; $L_{u,1} := (1 - u) * P_{0,1} + u * P_{1,1}$;
$L_{0,v} := (1 - v) * P_{0,0} + v * P_{0,1}$; $L_{1,v} := (1 - v) * P_{1,0} + v * P_{1,1}$;

(* Compute the normal by computing partial derivatives; ignore scale *)
$\frac{\partial F}{\partial u} := L_{1,v} - L_{0,v}$; $\frac{\partial F}{\partial v} := L_{u,1} - L_{u,0}$;

(* finally, evaluate the function...*)
$F := (1 - v)L_{u,0} + vL_{u,1}$; !!!

We now sketch the proof of our algorithm. A more complete proof can be found in [MDW93]. Let $F(u; v)$ be a degree $n \times m$ tensor product surface. Looking at the blossom $f$ of $F$ [Ram88], we see that, when evaluating $F(p; q)$ the algorithm first computes the bilinear function

$$L(u; v) = f(u, p, \ldots, p;\ v, q, \ldots, q).$$

The correctness of the algorithm follows immediately from the following relations:

$$F(p; q) \quad = \quad L(p; q)$$

$$\begin{aligned}\frac{\partial F}{\partial u}(p;q) &= n[L(1;q) - L(0;q)] \\ \frac{\partial F}{\partial v}(p;q) &= m[L(p;1) - L(p;0)].\end{aligned}$$

We now compare our algorithm to the bilinear algorithm for evaluating an $n \times n$ tensor product. In the bilinear algorithm, successive levels of control points are computed performing bilinear interpolation on sets of four control points on the previous level, typically using three linear interpolations, requiring six multiplications and three additions. For an $n \times n$ tensor product surface, there are $\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$ such combinations.

For our algorithm, each control point on a new level is a linear combination of two points from the previous level, with each new point being computed using two multiplications and one addition. When evaluating the first block of arguments, there are $(n+1)\sum_{i=1}^{n-1} i = \frac{(n+1)n(n-1)}{2}$ combinations. For the second block, there are $2\sum_{i=1}^{n-1} i = n(n-1)$ combinations. After these first two evaluations, we have reduced the control net to a bilinear patch and we need an additional six multiplications and three additions to compute the point on the surface.

Thus, the total number of multiplications used by the bilinear algorithm is

$$n(n+1)(2n+1),$$

while our algorithm uses

$$(n+1)(n+1)n + 2(n+1)n$$

multiplications. (Both algorithms require half the number of additions as multiplications.) While both algorithms have the same asymptotic complexity ($O(n^3)$), our algorithm is asymptotically twice as fast as the bilinear algorithm. In particular, our algorithm is computationally more efficient for surfaces of bi-degree $n \geq 2$.

Note that the number of multiplications at each step of the bilinear algorithm can be reduced from six to four, although three additions would still be required. With this variant of the bilinear algorithm, our algorithm is slightly slower when evaluating a bilinear patch, but is still more efficient for $n \geq 2$, and asymptotically our algorithm only requires two-thirds the number of multiplications and only half the additions.

A few notes are in order. First, although we emphasizes the bivariate case in this paper, the algorithm presented readily generalize to multivariate tensor products [MDW93]. Second, it is computationally advantageous to evaluate the tensor product first in the parametric direction of lowest degree. Third, we stopped at the next to last step of the de Casteljau algorithm since we were only interested in first derivatives. Higher order derivatives can be computed if the evaluation is stopped at an earlier stage. Finally, note that Sederberg has developed an $O(n^2)$ modified Horner's algorithm for evaluating the positions and tangents of rational tensor product Bézier surface patches [Sed].

**Acknowledgements:** This work was supported in part by the Xerox Corporation, Hewlett-Packard, the Digital Equipment Corporation, and the National Science Foundation under grant CCR-8957323.

# References

[Far93] Gerald Farin. *Curves and Surfaces for Computer Aided Geometric Design.* Academic Press, San Diego, third edition, 1993.

[MDW93] Stephen Mann, Tony DeRose, and Georges Winkenbach. Computing values and derivatives of bézier and b-spline tensor products. Research Report CS-93-31, Computer Science Department, University of Waterloo, Waterloo, Ontario, May 1993.

[Ram88] Lyle Ramshaw. Béziers and B-splines as multiaffine maps. In R. Earnshaw, editor, *Theoretical Foundations of Computer Graphics and CAD*, pages 757–776. Springer Verlag, 1988.

[Sed] Thomas W. Sederberg. Point and tangent computation of tensor product rational Bézier surfaces. *Computer Aided Geometric Design*, ?(?):??–??, ? ?

**Errata:**

- In the code, the line where we compute $F$ := should be an addition instead of a subtraction (this correction has been made in this copy of the paper; the line is marked with !!!).
- The formulation for the runtime of our algorithm is incorrect. A correct derivation is to note that we evaluate $n+3$ Bézier curves one short of completion. Each evaluation costs $\sum_{i=2}^{n} i$ which is $(\sum_{i=1}^{n} i) - 1 = (n+1)n/2 - 1$, giving a total of

$$(n+3)((n+1)n - 2) + 6$$

  multiplications (the extra six multiplications are the cost of evaluating the bilinear patch).

For purposes of further correspondence, please contact:

Professor Stephen Mann
Computer Science Department
University of Waterloo
Waterloo, Ontario, N2L 3G1
Canada