Evaluation and approximate evaluation of the multivariate *
Bernstein-Bézier form on a regularly partitioned simplex

by
Jörg Peters

Key words: Bernstein-Bézier form, power form, evaluation, multivariate, subdivision

Running title: Evaluation of the Bernstein-Bézier form

Date printed: January 30, 1994 (version Apr 15 93)

Computing Reviews Categories: I.3.5, G.1.2

General Terms: Algorithms, (Performance, Theory)

Abstract

Polynomials of total degree d in m variables have a geometrically intuitive representation in the Bernstein-Bézier form defined over an m-dimensional simplex. The two algorithms given in this paper evaluate the Bernstein-Bézier form on a large number of points corresponding to a regular partition of the simplicial domain. The first algorithm is an adaptation of isoparametric evaluation. The second is a subdivision algorithm. In contrast to de Casteljau's algorithm, both algorithms have a cost of evaluation per point that is linear in the degree regardless of the number of variables. To demonstrate practicality, implementations of both algorithms on a triangular domain are compared with generic implementations of six algorithms in the literature.

* This research was supported by NSF DMS-8701275

Author's current address: Department of Computer Sciences, Purdue University, IN 47907-1398

1. Introduction

The Bernstein-Bézier form is an important tool for representing piecewise polynomials. Many applications in computer aided geometric design benefit from the intuitive geometric significance of its coefficients and the fact that, like the power or Taylor form, the Bernstein-Bézier form is capable of representing polynomials of total degree in many variables. When it comes to evaluation, however, the Bernstein-Bézier form is considered inefficient, because its natural evaluation algorithm, de Casteljau's algorithm, has a higher complexity than evaluation of the power form by nested multiplication, and a much higher complexity than forward differencing when generating a large number of points. It may therefore reassure users of the Bernstein-Bézier form that a conversion is not necessary since there exist two simple algorithms that evaluate it efficiently on a regular lattice. The cost per point generated by either algorithm is linear in the degree regardless of the number of variables. In fact, the constant associated with the linear term of the theoretical time complexity of the subdivision algorithm decreases with the number of variables and is lower than the constant associated with forward differencing. Numerical experiments, given at the end of this paper, confirm the theoretical complexity.

Polynomial pieces in m variables come both in tensor-product form, defined on an m-dimensional cube, and as total-degree polynomials, defined over an m-dimensional simplex. This paper is concerned with the stable and efficient evaluation of total degree polynomials. The goal is to generate a large number of points corresponding to parameters on a regular partition of the domain simplex. For example, if the domain simplex is the right-angled unit triangle, then the parameters are $(i/n, j/n)$, where $0 \leq i + j \leq n$. Approximate evaluation on such a lattice is of interest, e.g. when rendering a piecewise polynomial surface. While efficient lattice-oriented algorithms are known for polynomials in tensor-product form [LCR80][LR80], there are with the exception of repeated extrapolation [D87, 3.3.7 p43] no examples of algorithms that take advantage of the regular partition of a domain simplex.

The two algorithms presented in this paper are based on a specialized version of de Casteljau's algorithm that allows nesting operations both with respect to the degree and the parameter dimension. The first algorithm recursively reduces the parameter dimension to obtain a sequence of univariate polynomials corresponding to lines parallel to one edge of the simplex. The univariate polynomials can then be evaluated by standard methods, such as univariate forward differencing. This requires little space and offers flexibility with respect to the location and number of evaluation points. The second is a subdivision algorithm that generates points on or close to the polynomial. The key idea for efficiency is to reduce the multivariate subdivision to a sequence of univariate subdivisions. Each univariate subdivision is efficiently and stably encoded as an averaging of adjacent coefficients into the average of their storage locations.

The structure of this paper is a follows. Section 2 reviews the multiindex notation used throughout the paper and defines the Bernstein-Bézier form, de Casteljau's algorithm and subdivision in this notation. Section 3 describes an important subroutine shared by the two new algorithms given in Section 4 and 5. Section 6 compares implementations of the two new algorithms for two variables with generic versions of algorithms from the literature reviewed in the Appendix.

2. Notation, de Casteljau's algorithm and subdivision

The number of variables, the total degree of the polynomial, and the number of points minus one per edge of the domain simplex are denoted by

$$m, \quad d, \quad n$$

respectively. We use the following multiindex notation for $\alpha \in \mathbb{Z}_+^{m+1}$:

$$\alpha := (\alpha_0, \ldots, \alpha_m),\ \xi^\alpha := \xi_0^{\alpha_0} \cdots \xi_m^{\alpha_m},\ |\alpha| := \alpha_0 + \ldots + \alpha_m, \quad \text{and} \binom{|\alpha|}{\alpha} := \frac{|\alpha|!}{\prod_{i=0}^m \alpha_i!}.$$

Thus evaluation at the barycentric coordinates $\alpha/|\alpha|$, where $|\alpha| = n$ generates $\binom{n+m}{m}$ points. Let $V := [v_0, \ldots, v_m]$ be the domain m-simplex formed by the vertices $v_i \in \mathbb{R}^m$ and $\xi := [\xi_0, \ldots, \xi_m]$ a vector of barycentric coordinate functions; that is, $\sum \xi_i = 1$, $\sum \xi_i(x) v_i = x$ and $\xi_j(x)$ is the barycentric coordinate of x with respect to v_j. The Bernstein-Bézier form of total degree d in m variables and with coefficients $c(\alpha)$ is defined as (cf. [B87] [F88]),

$$b_{m,d}[V] := \sum_{|\alpha|=d} \binom{d}{\alpha} c(\alpha) \xi^\alpha.$$

For example, denoting the j^{th} unit vector by e_j,

$$b_{2,3}[0, e_1, e_2] = \sum_{\alpha_0+\alpha_1+\alpha_2=3} \frac{3}{\alpha_0!\alpha_1!\alpha_2!} c(\alpha_0, \alpha_1, \alpha_2)(1-x-y)^{\alpha_0} x^{\alpha_1} y^{\alpha_2}$$

is a bivariate cubic on the standard 2-simplex. For comparison, the power form of a polynomial $p_{m,d}$ of degree d with the multiindex $\beta := (\beta_1, \ldots, \beta_m)$ is $p_{m,d}(x) = \sum_{|\beta| \le d} c(\beta) x^\beta$.

In the pseudo code used to specify the algorithms for evaluating the Bernstein-Bézier form, the lists of input and output parameters are separated by ';'. Latin letters are used for superscripts, e.g.

$$V^i := [v_0^i, \ldots, v_m^i]$$

is the ith simplex, while greek letters denote exponents. In Sections 4 and 5, we follow the convention that $[v_0, \ldots, v_j, v_{j+1}, \ldots, v_m] = [v_{j+1}, \ldots, v_m]$ if $j < 0$. If $b_{m,d}[V]$ is an argument to a pseudocode function, then m, d, V and the coefficients $c(\alpha)$ are passed. To avoid specializations of subroutines, a dummy return argument $*$ is used that fits any returned object and indicates that the object is not needed for further computation. The scope of a for-loop is indicated by indentation. The statement **for** $|\alpha| < d$ can be implemented either as a sequence of nested loops, one for each of α_0 to α_m, if m is fixed or recursively if m is variable.

Following [SH82 p.331], the theoretical stability of an evaluation process can be measured in terms of the basic operations and the level of indirection. Averaging of coefficients of a polynomial using positive weights is considered more stable than extrapolation or differencing since loss of significant digits is less likely. The level of indirection, ℓ, indicates how many intermediate operations separate the output from the original coefficients of the polynomial. For example, if each f_i is a basic operation, c represents the original coefficients and a point is generated by $f_k \circ \ldots \circ f_1(c)$, then $\ell = k$. A large ℓ indicates potential loss of stability due to round-off.

Applied one point at a time, de Casteljau's algorithm [C59] serves as a standard for accuracy, since the Bernstein-Bézier form is naturally defined in terms of the algorithm. The reader familiar with the evaluation triangle in the univariate case (see e.g. [BFK84, p.8]), will recognize that the variable l below counts the levels of an $(m+1)$-dimensional simplex, filled layer by layer first with the original $\binom{m+d}{m}$ coefficients, then with barycentric combinations of coefficients in the lower layer. For arbitrary m and d, the following algorithm computes $b_{m,d}[V]$ at x.

DeCasteljau $(b_{m,d}[V], x; c(0), b_{m,d}[V^0], \ldots, b_{m,d}[V^m])$

$\beta_i = \xi_i(x) \quad i = 0..m$ [determine the barycentric coordinates]
for $l = 1..d$ [levels of the $(m+1)$-dimensional subdivision simplex]
  for $|\alpha| = d - l$
    $c(\alpha) = \sum_{i=0}^{m} \beta_i c(\alpha + e_i)$

De Casteljau's algorithm not only generates the point $c(0) = b_{m,d}[V](x)$, but in the process also computes the coefficients of the $m+1$ subpolynomials

$$b_{m,d}[V^i] := \sum_{|\alpha^i|=d} (\eta^i)^{\alpha^i} \binom{d}{\alpha^i} c_{V^i}(\alpha^i), \qquad \text{where } V^i := [v_0, \ldots, v_{i-1}, x, v_{i+1}, \ldots, v_m],$$

η^i is the vector of barycentric functions corresponding to V^i,

$\alpha^i := (\alpha_0, \ldots, \alpha_{i-1}, l, \alpha_{i+1}, \ldots, \alpha_m), l = d - |\alpha|$

and $c_{V^i}(\alpha^i) := c(\alpha_0, \ldots, \alpha_{i-1}, 0, \alpha_{i+1}, \ldots, \alpha_m)$

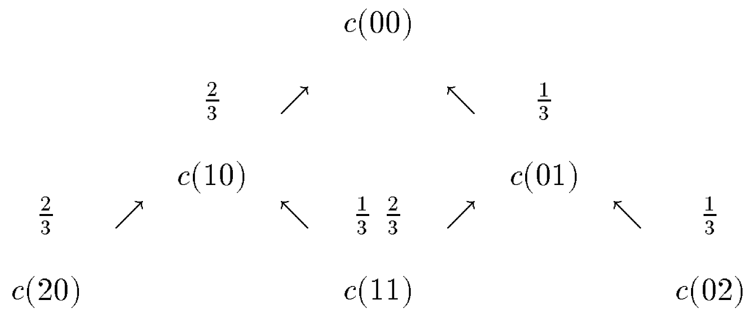
That is, if $x \in V$, each $b_{m,d}[V^i]$ represents $b_{m,d}[V]$ over a subsimplex of the original domain simplex and takes its coefficients from a facet of the $m+1$ dimensional subdivison simplex (cf. [Go83], [B87]).

(2.1) Example. We consider the case $d = 2$ and $m = 1$,

$$b_{1,2}[0,1] = 18(1-x)x + 18x^2.$$

With $\xi_0 = (1-x)$ and $\xi_1 = x$, $b_{1,2}[0,1] = 0 \cdot \xi_0^2 + 9 \cdot 2\xi_0\xi_1 + 18 \cdot \xi_1^2$ and the coefficients are $c(20) = 0$, $c(11) = 9$, and $c(02) = 18$. A call to DeCasteljau($b_{1,2}[0,1], \frac{1}{3}; \ldots$) generates

$\beta_0 = 2/3$, $\beta_1 = 1/3$, and the $(1+1)$ dimensional subdivision simplex

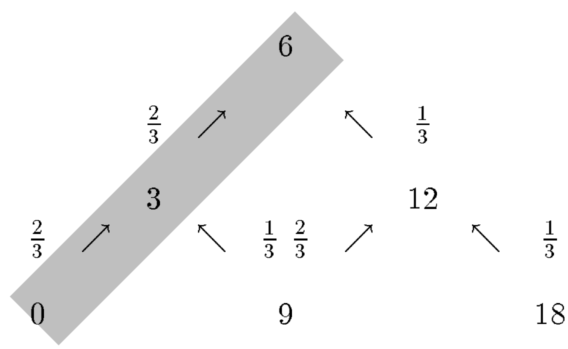


where for $l = 1$, $c(10) = \frac{2}{3}c(20) + \frac{1}{3}c(11) = 3$, $c(01) = \frac{2}{3}c(11) + \frac{1}{3}c(02) = 12$, and for $l = 0$, $b_{1,2}[0,1](\frac{1}{3}) = c(00) = 6$. The subpolynomials are

$$b_{1,2}[V^0] = b_{1,2}[\frac{1}{3}, 1] \text{ with coefficients } c_{[\frac{1}{3},1]}(20) = 6, c_{[\frac{1}{3},1]}(11) = 12, c_{[\frac{1}{3},1]}(02) = 18,$$

$$b_{1,2}[V^1] = b_{1,2}[0, \frac{1}{3}] \text{ with coefficients } c_{[0,\frac{1}{3}]}(20) = 0, c_{[0,\frac{1}{3}]}(11) = 3, c_{[0,\frac{1}{3}]}(02) = 6.$$

The coefficients $c_{[0,\frac{1}{3}]}(20) = c(20) = 0, c_{[0,\frac{1}{3}]}(11) = c(10) = 3, c_{[0,\frac{1}{3}]}(02) = c(00) = 6$ are displayed in the grey box below.



■

The summation in the inner loop of DeCasteljau uses m additions and $m+1$ multiplications. Since $\binom{d-1+m+1}{m+1}$ such sums are computed, the theoretical time complexity for the evaluation per point by DeCasteljau (ignoring index calculations) is

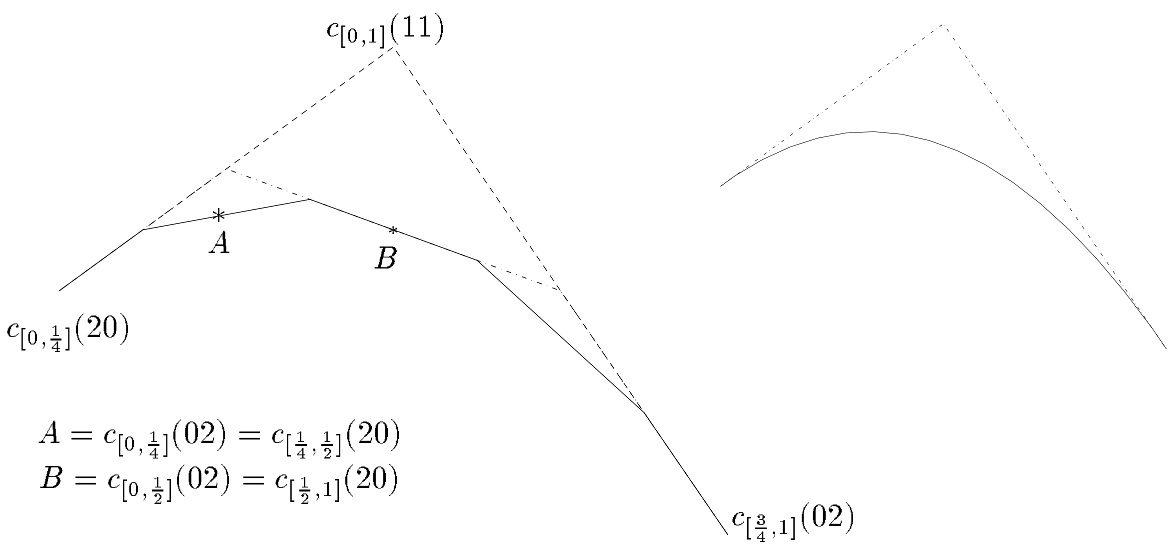
$$(2m+1)\binom{d+m}{m+1}.$$

This is inefficient even when evaluating at a single point since the number of operations is larger by a factor of d than the number of coefficients. However, the level of indirection, $\ell = d$, is minimal and the basic operation, averaging of the coefficients with positive weights, is stable for densely sampled polynomials.

Approximate evaluation by subdivision is motivated by the following theorem.

(2.2) Theorem. *[D86, Thm 4.1] A particular piecewise linear interpolant to the coefficients of the polynomial, the so-called Bernstein-Bézier net, converges quadratically in the size of the domain simplex and linearly in the size of the second derivative to the polynomial.*

The theorem is a direct consequence of the facts that any Bernstein-Bézier net as defined in [D86] reproduces linear functions and that the Bernstein-Bézier form is a stable basis. If, for example, the diameter of the domain simplex is halved at each subdivision step, then 10 subdivision steps reduce the distance between the polynomial and Bernstein-Bézier net to $(1/2^{10})^2 \approx 10^{-6}$ times the initial distance. The convergence is speeded up as pieces of maximal curvature are confined to subsimplices and thus the maximal curvature for the other polynomial pieces decreases. Consequently, after a number of subdivisions depending on the desired accuracy, all coefficients generated by the subdivision process can be accepted as good approximations to points on the surface.



(2.3) Figure: (Left) The solid polygon is the result of two steps of subdivision applied to $b_{1,2}[01]$ with coefficients c $= \begin{bmatrix} 0 & 9 & 18 \\ 9 & 18 & 0 \end{bmatrix}$ at $t = 1/2$.
(Right) An approximation generated by 4 subdivision steps with $t = 1/2$.

3. Evaluation on an edge of the domain simplex

When a single point is to be generated, nested multiplication as in SV-NestMult in the Appendix are, up to a constant, optimal since the cost equals the number of polynomial coefficients. However, for a large number of evaluations intermediate calculations can be reused. The key to efficiency is a specialization of de Casteljau's algorithm. If one evaluates at a point on the edge of the domain simplex, only two barycentric coordinates, say ξ_i and ξ_j, are non-zero and the summation in DeCasteljau simplifies to

for $l = 1..d$
  for $|\alpha| = d - l$
    $c(\alpha) = (1-x)c(\alpha + e_i) + xc(\alpha + e_j)$.

The inner for-loop can be split

for $l = 1..d$
  for $|\alpha_{\setminus ij}| < d$ where $\alpha_{\setminus ij} := (\alpha_0, \dots, \alpha_{i-1}, \alpha_{i+1}, \dots, \alpha_{j-1}, \alpha_{j+1}, \dots, \alpha_m)$
    $d' = d - |\alpha_{\setminus ij}|$
    for $\alpha_i + \alpha_j = d' - l$
      $c(\alpha) = (1-x)c(\alpha + e_i) + xc(\alpha + e_j)$

and rearranged in the form

for $|\alpha_{\setminus ij}| < d$
  $d' = d - |\alpha_{\setminus ij}|$
  for $l = 1..d'$
    for $\alpha_i + \alpha_j = d' - l$
      $c(\alpha) = (1-x)c(\alpha + e_i) + xc(\alpha + e_j)$

to reveal a univariate DeCasteljau algorithm in the inner two loops. Noting that indices α with $\alpha_{\setminus ij}$ fixed and $\alpha_i + \alpha_j = d'$ lie on a line in the simplex partition parallel to the boundary edge v_i, v_j, we see that the two inner loops average adjacent coefficients on such a line exactly as if they were the coefficients of a univariate polynomial $b_{1,d'}[0,1](t) = \sum_{\alpha_i+\alpha_j=d'} (1-t)^{\alpha_i} t^{\alpha_j} \frac{d'!}{\alpha_i!\alpha_j!} c(\alpha)$. Thus, given an input polynomial $b_{m,d}[V]$, $x \in [0,1] \subset \mathbb{R}$, and a point $w := (1-x)v_i + xv_j$ on an edge of the domain simplex V, a routine EdgeDeCasteljau based entirely on the *univariate* DeCasteljau algorithm can be devised that returns subpolynomials representing the input polynomial over the two simplices $V^i := [v_1, \dots, v_{i-1}, w, v_{i+1}, \dots, v_{m+1}]$ and $V^j := [v_1, \dots, v_{j-1}, w, v_{j+1}, \dots, v_{m+1}]$.

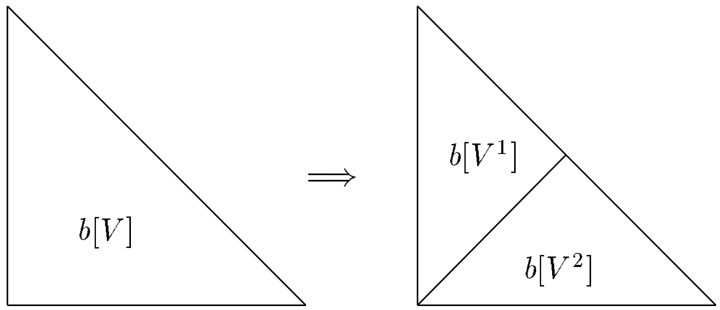
EdgeDeCasteljau $(b_{m,d}[V], i, j, x; b_{m,d}[V^i], b_{m,d}[V^j])$

for $|\alpha_{\setminus ij}| < d$ where $\alpha_{\setminus ij} := (\alpha_0, \dots, \alpha_{i-1}, \alpha_{i+1}, \dots, \alpha_{j-1}, \alpha_{j+1}, \dots, \alpha_m)$
  $d' = d - |\alpha_{\setminus ij}|$
  $b_{1,d'}[0,1](t) = \sum_{\alpha_i+\alpha_j=d'} (1-t)^{\alpha_i} t^{\alpha_j} \frac{d'!}{\alpha_i!\alpha_j!} c(\alpha)$
  DeCasteljau$(b_{1,d'}[0,1], x; *, b_{1,d'}[x,1], b_{1,d'}[0,x])$

For a fixed $\alpha_{\setminus ij}$, $\alpha_i = l$ and $\alpha_i + \alpha_j = d'$,

$$c_{V^i}(\alpha) = c_{[0,x]}(l, d'-l) \quad c_{V^j}(\alpha) = c_{[x,1]}(d'-l, l).$$

This is illustrated by the following example.



$V = [0, 2e1, 2e2]$

$V^1 = [0, e1 + e2, e2]$

$V^2 = [0, e1, e1 + e2]$

(3.1) Example. A call to EdgeDeCasteljau($b_{2,3}[0, 2e_1, 2e_2], 1, 2, \frac{1}{2}; \ldots$) results in $b[V^1]$ and $b[V^2]$ generated by

for $\alpha_0 = 0..2$ $\quad [\alpha_{\setminus 12} = \alpha_0$ since $m = 2.]$
$\quad d' = d - \alpha_0$
$\quad b_{1,d'}[0,1] = \sum\limits_{\alpha_1+\alpha_2=d'} (1-s)^{\alpha_1} s^{\alpha_2} \binom{d'}{\alpha_1} c(\alpha_0, \alpha_1, \alpha_2)$
$\quad$ DeCasteljau($b_{1,d'}[0,1], \frac{1}{2}; *, b_{1,d'}[\frac{1}{2}, 1], b_{1,d'}[0, \frac{1}{2}]$)

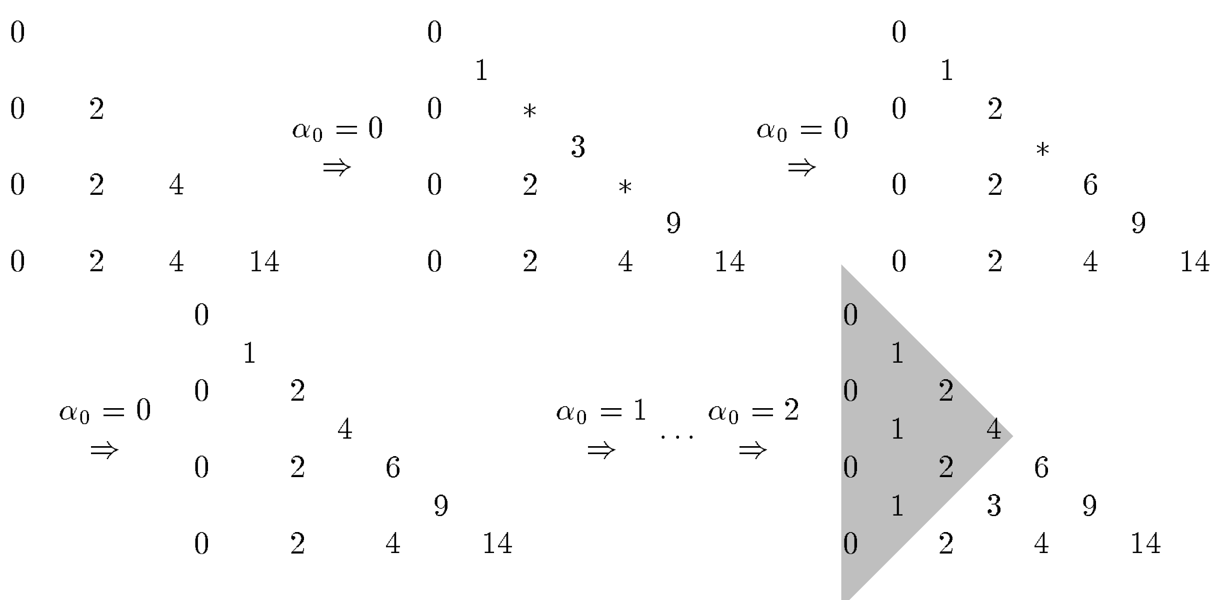
The assignment to $b_{1,d'}[0,1]$ interprets the right hand side as a univariate polynomial of degree $d - \alpha_0$ and with coefficients $c'(\alpha_1, \alpha_2) := c(\alpha_0, \alpha_1, \alpha_2)$ over $[0,1]$, α_0 fixed. If $b_{2,3}$ has the coefficients

$$c(300) = c(201) = c(102) = c(003) = 0, \quad c(210) = c(111) = c(012) = 2,$$
$$c(120) = c(021) = 4, \quad c(030) = 14,$$

then for $\alpha_0 = 0$, DeCasteljau($b_{1,3}[0,1], \frac{1}{2}; \ldots$) is executed where $b_{1,3}[0,1]$ has the coefficients

$$c'(30) = 0, \quad c'(21) = 2, \quad c'(12) = 4, \quad c'(03) = 14.$$

Next, for $\alpha_0 = 1$, the algorithm works on the coefficients $c'(20) = 0, c'(11) = 2, c'(02) = 4$, and finally the linear polynomial with coefficients $c'(10) = 0, c'(01) = 2$ is evaluated. The restriction $|\alpha_{\setminus ij}| < d$ avoids an evaluation of the constant polynomial. Associating the coefficients in a canonical way with the domain, the algorithm generates the following sequence of coefficients.



One reads off the coefficients of the subpolynomial $b_{2,3}[V^1]$ (grey)

$$c(300) = c(201) = c(102) = c(003) = 0, \quad c(210) = c(111) = c(012) = 1,$$
$$c(120) = c(021) = 2, \quad c(030) = 4,$$

and of the subpolynomial $b_{2,3}[V^2]$

$$c(300) = 0, c(201) = 1, c(102) = 2, c(003) = 4, c(210) = 2, c(111) = 3, c(012) = 6,$$
$$c(120) = 4, c(021) = 9, c(030) = 14.$$

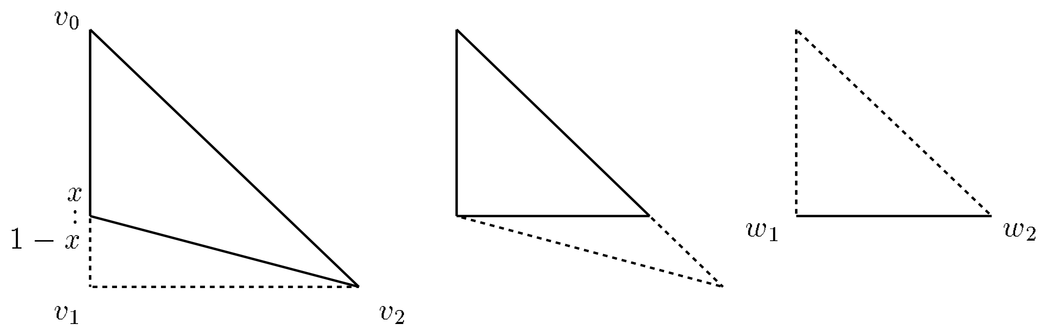
■

The innermost summation, $c(\alpha) = (1-x)c(\alpha+e_i) + xc(\alpha+e_j)$, consists of two multiplications and one addition, or one add-and-shift operation, if $x = \frac{1}{2}$. This implies the following lemma.

(3.2) Lemma. *The complexity of EdgeDeCasteljau is* $3\binom{d+m}{m+1}$ *operations, or* $\binom{d+m}{m+1}$ *add-and-shifts if* $x = \frac{1}{2}$. *The level of indirection is* $\ell = d$.

4. Isoparametric Evaluation

The following algorithm extracts a univariate polynomial for each choice of $m-1$ fixed parameters. Fixing $m-1$ barycentric coordinates leaves the task of aggregating the terms as coefficients of the univariate polynomial. The routine IsoParamEval is a systematic and stable way of performing this aggregation for polynomials in the Bernstein-Bézier form. Using EdgeDeCasteljau, the routine Slice extracts the $(m-1)$-variate polynomial $b_{m-1,d}[W]$ from $b_{m,d}[V]$. The domain vertices w_j correspond to the intersection of the hyperplane $\xi_0(u) = (1-x)$ with V.



(4.1) Figure: Slice($b_{2,d}[v_0, v_1, v_2], x; b_{1,d}[w_1, w_2]$).

Slice($b_{m,d}[V^0], x; b_{m-1,d}[W]$)

for $j = 1..m$
  EdgeDeCasteljau($b_{m,d}[V^{j-1}], 0, j, x; *, b_{m,d}[V^j]$)
    where $V^j = [v_0^{j-1}, \dots, v_{j-1}^{j-1}, w_j, v_{j+1}^{j-1}, \dots, v_m^{j-1}]$ and $w_j = (1-x)v_0 + xv_j$
$b_{m-1,d}[W] = b_{m-1,d}[v_1^m, \dots, v_m^m]$

The last assignment in Slice is based on the well-known fact that the restriction of the Bernstein-Bézier form to a facet is entirely defined by the coefficients associated with that facet. By dropping v_0, only the polynomial of $(m-1)$ variables associated with W is retained. Slice can be simplified if $x = 0$ or $x = 1$.

(4.2) Example. A call to Slice($b_{2,3}[2e_2, 0, 2e_1], \frac{1}{2}; \dots$) results in
for $j = 1..2$
  EdgeDeCasteljau($b_{2,3}[V^{j-1}], 0, j, \frac{1}{2}; *, b_{2,3}[V^j]$)
$b_{1,3}[W] = b[e_2, e_1 + e_2]$
Here $V^1 = [2e_2, e_2, 2e_1]$ and $V^2 = [2e_2, e_2, e_1 + e_2]$. With the coefficients associated with the domain triangle $[2e_2, 0, 2e_1]$ in the canonical way as in Example 3.1, the coefficient matrix transforms as follows.

$$
\begin{array}{cccc}
0 & & & \\
 & & & \\
0 & 2 & & \\
 & & & \\
0 & 2 & 4 & \\
 & & & \\
0 & 2 & 4 & 14
\end{array}
\;\Rightarrow .. \Rightarrow\;
\begin{array}{cccc}
0 & & & \\
0 & & & \\
0 & 2 & & \\
0 & 2 & & \\
0 & 2 & 4 & \\
0 & 2 & 4 & \\
0 & 2 & 4 & 14
\end{array}
\;\Rightarrow .. \Rightarrow\;
\begin{array}{ccccccc}
0 & & & & & & \\
0 & 1 & & & & & \\
0 & 1 & 2 & & & & \\
0 & 1 & 2 & 4 & & & \\
 & & 2 & 3 & 6 & & \\
 & & & & 4 & 9 & \\
 & & & & & & 14
\end{array}
$$

From this one reads off the coefficients of $b_{1,3}[e_2, e_1 + e_2]$ as

$$c'(30) = 0, \quad c'(21) = 1, \quad c'(12) = 2, \quad c'(03) = 4.$$

That is the boundary coefficients of $b_{2,3}[2e_2, e_2, e_1 + e_2]$ determine the polynomial over the boundary $[e_2, e_1 + e_2]$ of the domain of $b_{2,3}$. ■

The routine Slice is the basic building block of the isoparametric evaluation algorithm.

IsoParamEval $(b_{m,d}[V^0], n)$

output $c(d, 0, .., 0)$ of $b_{m,d}[V^0]$
for $i_1 = 1..n$
  Slice($b_{m,d}[V^0], \frac{i_1}{n}; b_{m-1,d}[V^1]$)
  output $c(d, 0, .., 0)$ of $b_{m-1,d}[V^1]$
    $\ddots$
    for $i_l = 1..i_{l-1}$
      Slice($b_{m+1-l,d}[V^{l-1}], \frac{i_l}{n}; b_{m-l,d}[V^l]$)
      output $c(d, 0, .., 0)$ of $b_{m-l,d}[V^l]$
        $\ddots$
        for $i_{m-1} = 1..i_{m-2}$
          Slice($b_{2,d}[V^{m-2}], \frac{i_{m-1}}{n}; b_{1,d}[V^{m-1}]$)
          output $c(d, 0)$ of $b_{1,d}[V^{m-1}]$
          BBeval($b_{1,d}[v_0^{m-1}, v_1^{m-1}], i_{m-1}$)

Here $x_l = i_l/n$, $V^l = [(1-x_l)v_0^{l-1} + x_l v_1^{l-1}, \ldots, (1-x_l)v_0^{l-1} + x_l v_{m+1-l}^{l-1}]$ and BBeval($b_{1,d}, k$) is any routine that outputs the value of the univariate polynomial $b_{1,d}$ in Bernstein-Bézier form at j/k, $j = 1..k$. If the number of variables, m is to vary, then the nested loops are implemented by recursion. To apply IsoParamEval to tensor-product polynomials only the upper bound of the for-loops has to be changed from i_j to n.

(4.3) Example. If $m = 3$ and $V^0 = [4e_3, 4(e_2 - e_3), -4e_3, 4(e_1 - e_3)]$, then $i_1 = n/2$ implies $x_1 = 1/2$ and the bivariate subpolynomial returned by Slice has coefficients with respect to the simplex $V^1 = [2e_2, 0, 2e_1]$ of the previous example. ■

Stability. The level of indirection in IsoParamEval is $\ell = d\frac{(m+1)m}{2}$ since the calls to the univariate DeCasteljau routine in EdgeDeCasteljau are independent and $m+(m-1)+\ldots+2$ applications of EdgeDeCasteljau are necessary. The basic operation in EdgeDeCasteljau is the forming of convex combinations of the form $C = (1-x)A + xB$ for $x \in [0..1]$ and adjacent Bernstein-Bézier coefficients A and B. The only other numerical task in IsoParamEval is the univariate evaluation. In the case of SV-NestMult the basic operations are of the form $C = xA + B$.

Theoretical Time Complexity. By Lemma 3.2, the cost of slicing $b_{m,d}$ to obtain $b_{m-1,d}$ for a fixed mth parameter is $w(m) = 3m\binom{m+d}{d-1}$. Thus a total slicing cost of $n\sum_{i=0}^{m-2} n^i w(m-i)$ is distributed over $\binom{m+n}{n}$ points. Since n is much larger than either d or m, the cost per point is essentially independent of the slicing cost and equal to the cost of the univariate evaluation routine.

Storage Complexity. EdgeDeCasteljau needs one array of size $\binom{m+d}{m}$. This array can be reused by Slice. For time efficiency it is reasonable to allocate $\binom{j+d}{j}$ space for the coefficients of each $b[V^j]$. If the univariate evaluation is not by subdivision, then the final points can be output directly and need not be stored.

Remarks. Evaluation along isoparametric lines does not readily yield multidirectional information, like normal fields or curvature fields. Two possible fixes are evaluation with different choices of fixed parameters or storing and connecting the surface points to analyze the divided differences of the piecewise linear approximant.

5. Binary subdivision

Binary subdivision is motivated by Theorem 2.2 which states that a piecewise linear interpolant to the coefficients, the Bernstein-Bézier net, converges quadratically in the diameter of the domain simplex to the polynomial. If a polynomial surface is to be displayed as a suitably connected mesh of points, the connecting lines between the points do in general not lie on the surface and hence little is gained by placing the mesh points exactly on the polynomial surface. Convergence of the mesh to the surface is guaranteed by using the subroutine BinarySub below which halves the domain diameter.

BinarySub$(b_{m,d}[V]; b[V^0], \ldots, b[V^{2^m-1}])$

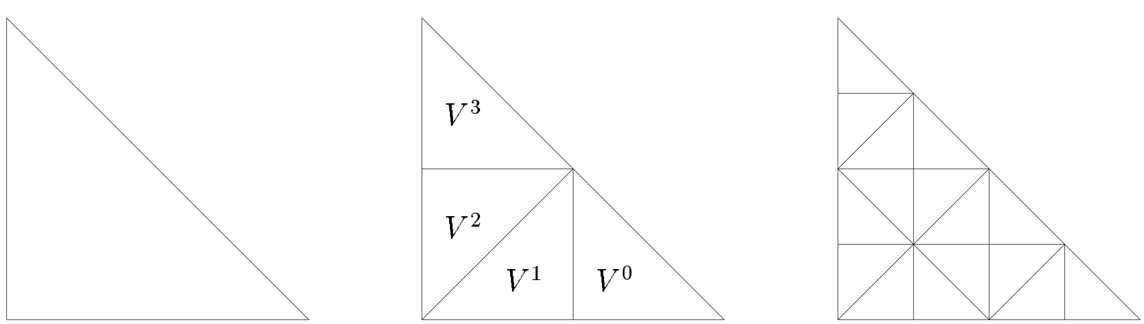
$V^0 = V$
for $i = 0..(m-1)$ [a subdivision sub-step]
  for $j = 2^i - 1..0$ [for each subsimplex]
    EdgeDeCasteljau$(b[V^j], i, m, \frac{1}{2}; b[V^{2j}], b[V^{2j+1}])$
      where $w_i = (v_i^j + v_m^j)/2$ [$v_m^j \neq v_m$ in general]
      $V^{2j} = [v_0^j, \ldots, v_{i-1}^j, w_i, v_{i+1}^j, \ldots, v_m^j]$
      $V^{2j+1} = [v_0^j, \ldots, v_{i-1}^j, w_i, v_i^j, \ldots, v_{m-1}^j]$

(5.1) Example. The subdivision of the domain simplex by BinarySub$(b_{2,3}[V]; \ldots)$ is illustrated in Figure 5.2 for $V := [2e_2, 0, 2e_1] = \begin{bmatrix} 0 & 0 & 2 \\ 2 & 0 & 0 \end{bmatrix}$. The domain simplices are

$$i=0, j=0: \quad w_0 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, V^0 = \begin{bmatrix} 1 & 0 & 2 \\ 1 & 0 & 0 \end{bmatrix}, V^1 = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 2 & 0 \end{bmatrix}$$

$$i=1, j=1: \quad w_1 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, V^2 = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \end{bmatrix}, V^3 = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 2 \end{bmatrix}$$

$$i=1, j=0: \quad w_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, V^0 = \begin{bmatrix} 1 & 1 & 2 \\ 1 & 0 & 0 \end{bmatrix}, V^1 = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}.$$



(5.2) Figure: Subtriangles generated by 2 steps of SimplexSub $(m = 2)$.

Starting with the coefficients of Example 3.1, the sequence of coefficient matrices is

$$\begin{array}{cccc} 0 &&& \\ &&& \\ 0 & 2 && \\ &&& \\ 0 & 2 & 4 & \\ &&& \\ 0 & 2 & 4 & 14 \end{array} \;\Rightarrow .. \Rightarrow\; \begin{array}{ccccccc} 0 &&&&&& \\ & 1 &&&&& \\ 0 && 2 &&&& \\ & 1 && 4 &&& \\ 0 && 2 && 6 && \\ & 1 && 3 && 9 & \\ 0 && 2 && 4 && 14 \end{array} \;\Rightarrow .. \Rightarrow\; \begin{array}{ccccccc} 0 &&&&&& \\ 0 & 1 &&&&& \\ 0 & 1 & 2 &&&& \\ 0 & 1 & 2 & 4 &&& \\ 0 & 1 & 2 & 4 & 6 && \\ 0 & 1 & 2 & 4 & 6 & 9 & \\ 0 & 1 & 2 & 4 & 6 & 9 & 14 \end{array}$$

From this we read off the coefficients of $b[V^2] = b\begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \end{bmatrix}$ on the grey triangle:

$$c(030) = c(021) = c(012) = c(003) = 0, \quad c(120) = c(111) = c(102) = 1,$$
$$c(201) = c(210) = 2, \quad c(300) = 4.$$

■

To complete the algorithm, BinarySub has to be called for each subsimplex at each step of the subdivision. Below σ is the number of subdivisions necessary to generate $n = d2^\sigma$ plus one points per edge of the parameter simplex.

SimplexSub$(b_{m,d}[V], \sigma; b[V^0], \ldots, b[V^{2^{m\sigma}}])$

for $s = 1..\sigma$ [steps of the subdivision]
  $l = 2^{m(s-1)} - 1$
  for $i = l..0$ [for each subsimplex]
    $j = i2^m$
    BinarySub$(b[V^i]; b[V^{j+2^m-1}], \ldots, b[V^j])$

Stability. The only operation is of the form $A = (B+C)/2$, i.e. add-and-shift. The level of indirection is at most $\ell = \sigma m d$.

Theoretical Time Complexity. Let σ be the number of subdivisions necessary to generate n coefficients per edge of the domain simplex, i.e. $\sigma = \log_2(n/d)$. Each traversal of the inner loop of SimplexSub generates $(2^m)^s$ simplices from $(2^m)^{s-1}$ since each call to BinarySub replaces one simplex by 2^m simplices. The ith step of BinarySub requires 2^i calls to EdgeDeCasteljau and each call to EdgeDeCasteljau consists of $\binom{d+m}{m+1}$ add-and-shifts according to Lemma 3.2. This yields the total work count

prev. subd's.	simplices per subd.	eval.'s per simplex	add-shift per eval.
$\sum_{s=0}^{\sigma-1}$	$(2^m)^s$	$\left(\sum_{i=0}^{m-1} 2^i\right)$	$\binom{d+m}{m+1}$

$$= (2^{m\sigma} - 1)\binom{d+m}{m+1}$$

Since the number of generated points is

$$\binom{n+m}{m} \quad \text{where } n := 2^\sigma d,$$

we obtain the following bound on the number of operations per coefficient.

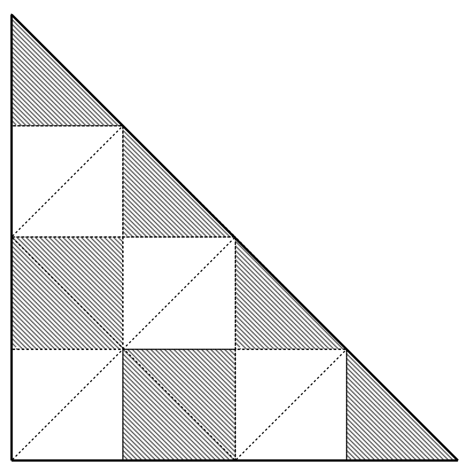
(5.3) Theorem. *The number of add-and-shifts per point generated by SimplexSub is*

$$w(m,d) := \frac{d}{m+1} \frac{(2^{m\sigma}-1)(d+m)\cdots(d+1)}{(2^\sigma d+m)\cdots(2^\sigma d+1)} < \frac{d}{m+1} \frac{(d+m)\cdots(d+1)}{d^m}.$$

In particular,

$$w(2,d) < \frac{1}{3}d + 1 + \frac{2}{3d},$$
$$w(3,d) < \frac{1}{4}d + \frac{3}{2} + \frac{11}{4d} + \frac{3}{2d^2},$$
$$w(4,d) < \frac{1}{5}d + 2 + \frac{7}{d} + \frac{10}{d^2} + \frac{24}{5d^2}$$

The asymptotic time complexity decreases with m for fixed d. The cost can be further decreased by storing the coefficients of all subpolynomials in a common m-dimensional simplicial array as in Example 5.1. That is, the coefficients of the subpolynomials are not returned but rather each polynomial piece is represented by a sector of the array. This storage allocation avoids redundant evaluation at facets shared by two or more subsimplices and simplifies the index calculation.



(5.4) Figure: Pairs of subtriangles that are subdivided in one sweep.

(5.5) Example: Figure 5.2(middle) illustrates Example 5.1 after the first subdivision step. In the second step, the edge from [1, 1] to [0, 0] is independently subdivided at the midpoint both for

$$b\begin{bmatrix}1 & 0 & 0\\ 1 & 1 & 0\end{bmatrix} \quad \text{and} \quad b\begin{bmatrix}1 & 1 & 0\\ 1 & 0 & 0\end{bmatrix}.$$

This can be avoided by placing the subpolynomials into a common 2-dimensional array and making the algorithm work on pairs of triangles. That is, edge-adjacent triangles of the same color in Figure 5.4 are subdivided in one sweep (cf. [P90] for the details of the index calculation). ■

For additional efficiency, the coefficients of adjacent C^0-connected triangles can be put into the same array to make use of a standard m-dimensional array and share the computational effort on the common face. Surface constructions based on splitting, e.g. [F83], [P90a], naturally pair up polynomials in the bivariate case.

Storage Complexity. As explained above, all computations can be performed in the simplicial array of size $\binom{n+m}{m}$ that stores the result. If more than one triangular patch is to be evaluated, storage efficiency can be improved by storing two adjacent polynomial pieces in one generic, rectangular array.

Remarks. (1) Multidirectional information such as directional derivatives and therefore normal and curvature fields can be obtained at all subdivision points by differencing of adjacent coefficients (cf. [B87]). (2) SimplexSub, as stated above, can be modified to allow for a biased distribution of parameter values, namely by choosing the evaluation parameter in EdgeDeCasteljau to be $\neq \frac{1}{2}$. (3) If the *in situ* construction is not used, subdivision need not proceed through all edges. For example, the edge with the longest associated Bernstein-Bézier polygon or highest curvature can be subdivided at each stage. With this strategy the same edge may be subdivided repeatedly in order to get more uniform coverage of the image of a parametric map. By selecting the subdivision edge globally, this adaptively subdivides adjacent polynomials in the same time step and creates a subdivision surface without gaps.

6. A comparison of evaluation and approximate evaluation methods for the bivariate Bernstein-Bézier form

This section compares implementations of algorithms for bivariate polynomials in Bernstein-Bézier form defined over a triangle. Figure 6.1 shows time per evaluation for a range of $d = 2..14$. The time axis is linear and in the *msec* range. Table 6.2 below lists the time complexity, the code length and miscellaneous observations on stability and storage complexity of the algorithms. Time complexity counts additions plus multiplications per point. For the run time comparison, $\binom{2^4 d+2}{2}$ points were generated to take account of the binary distribution of the parameter values for the subdivision algorithms. The observations on stability and storage requirements are abbreviated as follows.

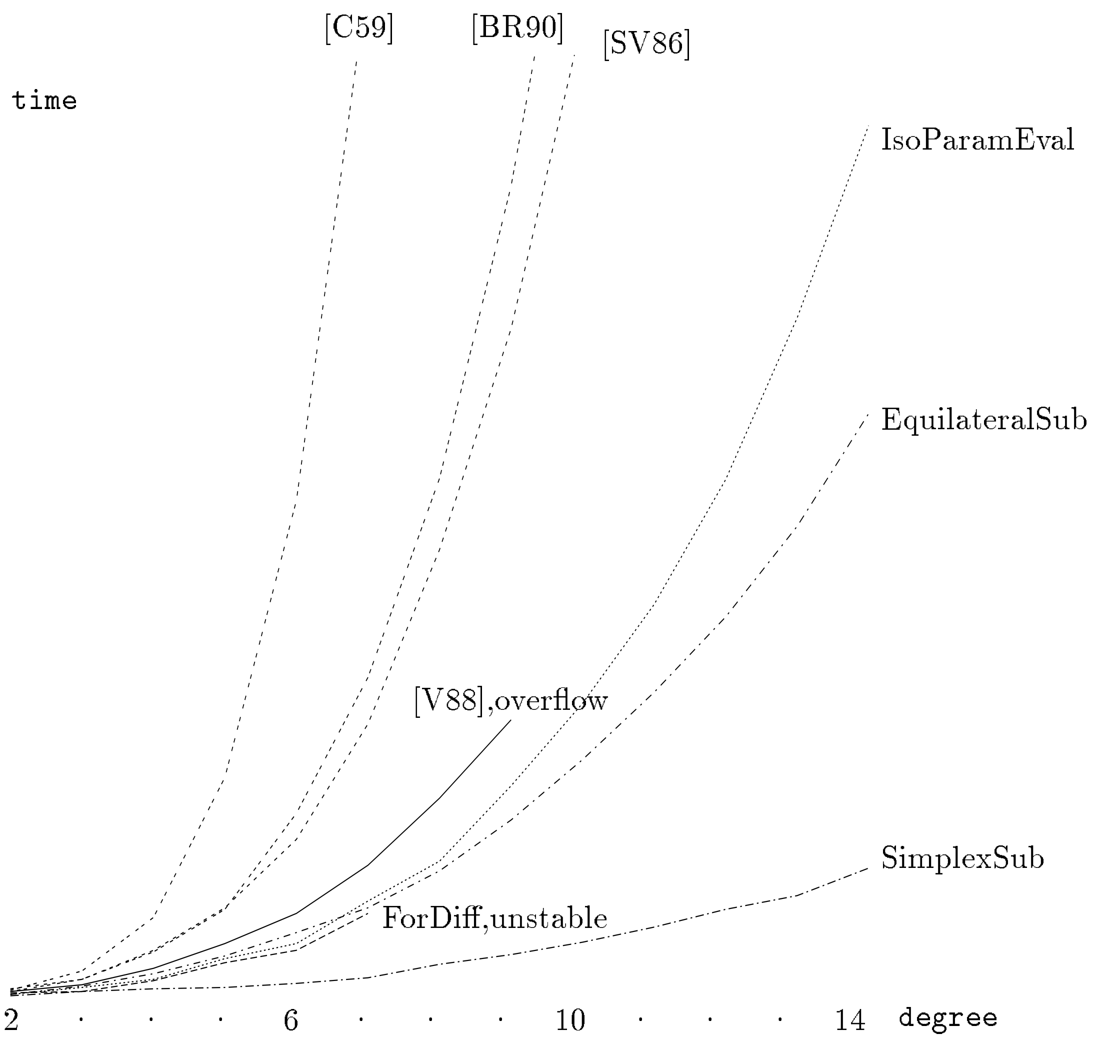
- A A conversion from the Bernstein-Bézier form to power form is necessary; the additional lines of C-language code for the conversion are listed in parentheses.
- B Multidirectional information such as normal and curvature fields are not easily generated together with the points.
- C The quantities γ used in the refinement grow rapidly with d and the reduction step. In the implementation, following [V88] and [V90], this leads to overflow.
- D(x) Instability due to round-off for $d \geq x$. Both D.I.M. and ForDiff were started with an accurate difference table.
- E All intermediate subdivision values have to be stored.
- G The number of coefficients generated is not arbitrary, but a power of 2. Hence the spacing over a parameter interval of length l is $l/(d2^\sigma)$.

algorithm	time complexity	stability storage	≈ lines of code	reference
DeCasteljau_2	$d(d+1)(d+2)$		30	[C59]
NestMult	$2d(d+1)$	A	20 (+20)	[BR90]
SV-NestMult	$(d^2+5d+4)/2$		25	[SV86]
D.I.M.	$1.5d+e+f$	B,C	200	[V88]
IsoParamEval	$2d+e$	B	60	§4
ForDiff	$d+e+f$	B,D(5),	50	eg[B78, p15]
EquilateralSub	$1.4d+7/3$	E,G	180	[Go83],§8
SimplexSub	$\frac{d}{3}+1$	E,G	50	§5

(6.2) Table: Evaluation of bivariate (triangular) patches in Bernstein-Bézier form.

Stability of the algorithms was checked against double precision DeCasteljau. Given the finite number of evaluations, the time complexity must account for the linear overhead when generating a quadratic number of points. This overhead is measured by e and f.

- $e := \frac{12}{n-1}\binom{d+2}{3}$ is the distributed cost of n calls to EdgeDeCasteljau to extract a univariate polynomial, distributed over $n(n-1)/2$ points.
- $f := \frac{2}{n-1}\binom{d+1}{2}$ is n times the cost of building the finite difference table distributed over $n(n-1)/2$ points.



(6.1) Figure: Time per point for $\binom{2^4 d+2}{2}$ points corresponding to the bivariate Bernstein-Bézier form. The time scale is linear.

SimplexSub has a distinct time advantage over its competitors already for polynomials of low degree. The table below shows the time per point for the evaluation of low degree polynomials.

degree	SimplexSub	EquilateralSub	ForDiff	IsoParamEval	[V88]
2	0.03	0.03	0.04	0.03	0.06
3	0.05	0.09	0.08	0.10	0.10
4	0.05	0.16	0.14	0.14	0.22
5	0.10	0.28	0.22	0.32	0.36

If low storage is required, then ForDiff is advantageous for polynomials of low degree, while IsoParamEval coupled with the univariate version of SV-NestMult is preferable for polynomials of high degree.

7. Conclusion

The two algorithms presented in this paper have a linear time complexity and low actual run time for generating a mesh of points corresponding to an m-variable total-degree polynomial in Bernstein-Bézier form. Both algorithms are based on a low cost specialization of de Casteljau's algorithm that amounts to repeated univariate averaging, and on nesting computations both with respect to the domain and the degree. While the isoparametric evaluation of Section 4 requires little storage and offers flexibility in the number and location of evaluation points, the subdivision method of Section 5 has the lowest run time and yields derivatives as a simple extension of the evaluation. Implementation in the bivariate case and a general complexity analysis show that binary subdivision and isoparametric evaluation compete favourably with standard algorithms in the literature.

Acknowledgement: I thank C. de Boor, D. Ng and W. Volk for their comments on earlier drafts of the paper.

References

[Boe83] W. Boehm (1983), Subdividing multivariate splines, *Computer Aided Design* **15** no 6 (nov):345-352.

[B78] C. de Boor (1978), *A practical guide to splines*, Applied Mathematical Sciences, Vol **27**, Springer, New York.

[B87] C. de Boor (1987), B-form basics, *Geometric Modeling: Applications and New Trends*, G. Farin ed., SIAM, Philadelphia.

[BR90] C. de Boor, A. Ron (1990), Computational aspects of polynomial interpolation in several variables, Comp. Sci. Tech. Rep. **924**, U. of Wisconsin, Madison.

[BFK84] W. Boehm, G. Farin, J. Kahmann (1984), A survey of curve and surface methods in CAGD, *CAGD* **1**, pp. 1–60.

[C59] F. de Casteljau, (1959), Courbes et surfaces á pôles, *André Citröen Automobiles*, Paris 1963.

[CB80] S.D. Conte, C. de Boor (1980) *Elementary Numerical Analysis*, Third Ed., McGraw-Hill, New York.

[D86] W. Dahmen (1986), Subdivision algorithms converge quadratically, *J. of Comput. Appl. Math.***16**, 145-158.

[D86a] W. Dahmen (1986), Bernstein-Bézier representation of polynomial surfaces, *SIGGRAPH '86* lecture notes.

[D87] W. Dahmen (1987), Subdivision algorithms – recent results, some extensions and further developments, in *Algorithms for approximation*, Mason, J.C. and Cox, M.G. eds., 1st Shrivenham Conference 1985, p 21-49.

[F83] G. Farin (1983), Smooth Interpolation to Scattered 3D-Data, *Surfaces in CAGD, R.E. Barnhill, W. Boehm, eds.*

[F88] G. Farin (1988), *Curves and surfaces for Computer Aided Geometric Design*, Academic Press, Boston.

[FR88] R.T. Farouki, V.T. Rajan (1988) Algorithms for polynomials in Bernstein form, *Computer Aided Geometric Design* **5** No 1, p. 1-26.

[F91] R.T. Farouki (1991) On the stability of transformations between power and Bernstein polynomial forms, *Computer Aided Geometric Design* **8** No 1, p. 29-36.

[Go83] R.N. Goldman (1983), Subdivision algorithms for Bézier triangles, *Computer Aided Design* **15** no 3 (may):159-166.

[GvL89] G. H. Golub, C. F. van Loan (1989), *Matrix Computations*, 2nd edition, Johns Hopkins University Press, London.

[Goo87] T.N.T. Goodman (1987), Variation diminishing properties of Bernstein polynomials on triangles, J. Approx. Theory **50**, 111–126.

[LR80] J. Lane, R.F. Riesenfeld (1980), A theoretical development for the computer generation and display of piecewise polynomial surfaces, *IEEE Transaction of Pattern Analysis and Machine Intelligence* **2**(1): 35–46.

[LSP87] S.-L. Lien, M. Shantz, V. Pratt (1987), Adaptive forward differencing for rendering curves and surfaces, *Computer Graphics* **21** (4), 111–118.

[LCR80] T. Lyche, E. Cohen, R. Riesenfeld (1980), Discrete B-splines and subdivision techniques in computer aided geometric design and computer graphics. *Comp. Graphics and Image Process.* **14** No.2, 87–111.

[P90] J. Peters (1990) Evaluation of multivariate Bernstein polynomials, CMS Tech. Report No. 91-1, U of Wisconsin.

[P90a] J. Peters (1990) Smooth mesh interpolation with cubic patches, *Computer Aided Design* **22** No 2, 109-120.

[SV86] L.L. Schumaker, W. Volk (1986), Efficient evaluation of multivariate polynomials, *Computer Aided Geometric Design* **3**: 149-154.

[Schö59] I.J. Schönberg (1959), On variation diminishing approximation methods, in *On numerical approximation*, R.E. Langer ed., U. of Wisconsin Press, 1959, 249–274.

[SH82] F. Stummel, K. Hainer (1982), *Praktische Mathematik*, Teubner, Stuttgart, 1982.

[V88] Volk W. (1988), An efficient raster evaluation method for univariate polynomials, *Computing* **40**: 163-173.

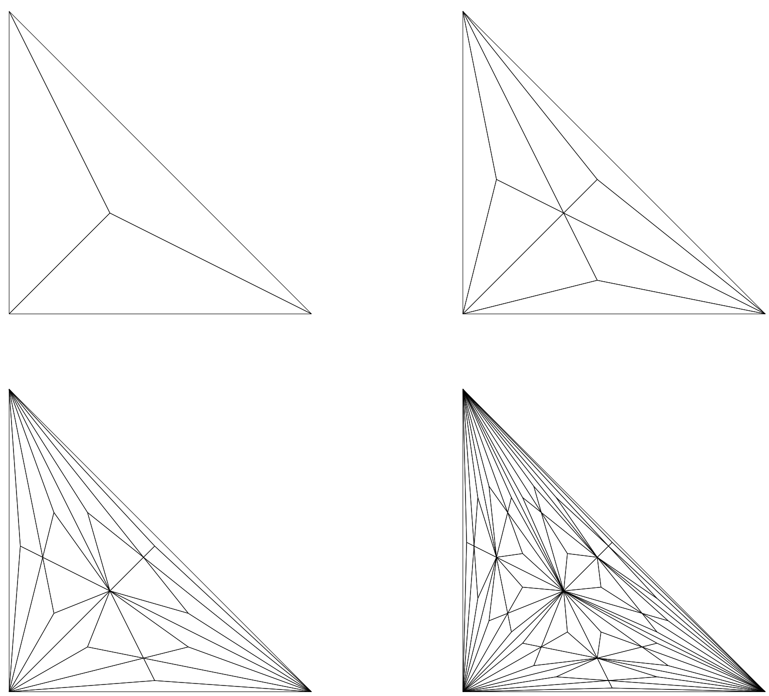
[V88a] Volk W. (1988), Die Auswertung von Polynomen mehrerer Veränderlicher auf Punktrastern, *Austrographics '88* A. Clauer, W. Purgathofer, eds., Springer Verlag, Berlin.

[V90] Volk W. (1990), Making the difference interpolation method for splines more stable, *J. of Comput. and Appl. Math.* **33**: 53-59.

8. Appendix: some alternative evaluation methods for polynomials in the Bernstein-Bézier form

This section reviews recursive use of de Casteljau's algorithm [Go83],[Boe83], two forms of nested multiplication [BR90],[SV86], generic forward differencing and the related difference interpolation method [V88].

Figure 8.1 shows that recursive subdivision at the centroid is not a good generalization of the univariate subdivision at the midpoint. In fact, Theorem 2.2 does not guarantee convergence of the corresponding piecewise linear interpolant to the polynomial.



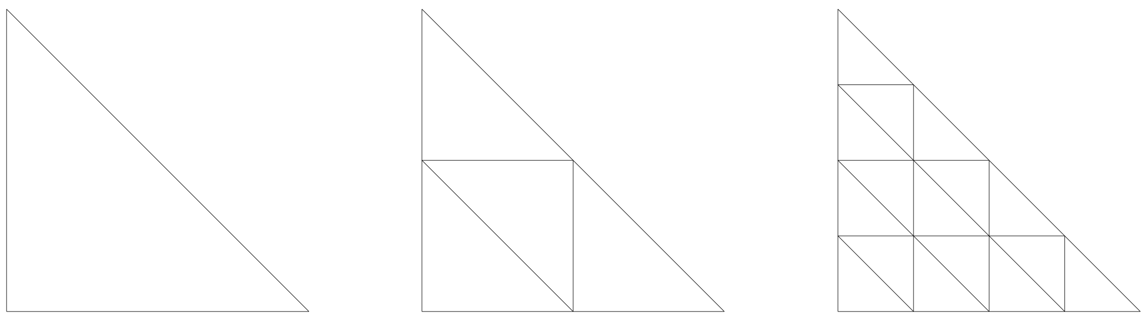
(8.1) Figure: Subtriangles generated by splitting at the centroid ($m = 2$).

A more uniform covering of the domain can be achieved in two dimensions by splitting an equilateral triangle into four as illustrated in Figure 8.2 (cf. [Go83, Fig. 9], [Goo87]). This subdivision can be achieved as shown in [Boe83, Fig11]. The algorithm coded for this paper uses however a slightly different sequencing of the intermediate subdivision steps to reduce ℓ to $\sigma 3(d-1)$ and use fewer and more symmetric extrapolations. C-language code is given in [P90].

EquilateralSub $(b[V], \sigma; b[V^0], \ldots, b[V^{4^\sigma}])$ $\quad$ [$m = 2$ only!]

for $s = 1..\sigma$ $\quad$ [each subdivision]
$k = 4^{s-1} - 1$
  for $i = k..0$ $\quad$ [for each subtriangle; overwrites]

for $j = 0, 1, 2$ [each vertex in V^i]
$b[V^{4i+j}] = b[V^i]$
for $l = 0, 1, 2,\ l \neq j$ [each edge emanating from the vertex]
EdgeDeCasteljau($b[V^{4i+j}], j, l, \frac{1}{2}; b[V^{4i+j}], *$)
Extrapolate($b[V^{4i}], b[V^{4i+1}], b[V^{4i+2}]; b[V^{4i+3}]$)
[extrapolate the interior triangle from the 3 surrounding ones]



(8.2) Figure: Subtriangles generated if EquilateralSub.

The next two methods generate and apply to polynomials in power form $p_{m,d}(x) = \sum_{|\beta|\le d} c(\beta)x^\beta$, where $\beta \in \mathbb{N}^m$.

SV-NestMult [SV86] converts the Bernstein-Bézier form to a *modified power form* which is then evaluated by nested multiplication (see e.g. NestMult below). The modified power form for $m = 1$ is obtained (see [Schö59, p.252]) by observing that

$$\sum_{\alpha_0+\alpha_1=d} (1-x)^{\alpha_0} x^{\alpha_1} \frac{d!}{\alpha_0!\alpha_1!} c(\alpha_0, \alpha_1) = x^d \sum_{\alpha_0=0}^{d} (\frac{1-x}{x})^{\alpha_0} \frac{d!}{\alpha_0!(d-\alpha_0)!} c(\alpha_0, d-\alpha_0)$$
$$= x^d \sum_{\beta=0}^{d} \eta^\beta c'(\beta),$$

where $\beta \in \mathbb{Z}$, $\eta := (1-x)/x$ and $c'(\beta) := \binom{d}{\beta} c(\beta, d-\beta) = \binom{d}{\beta} c(\alpha)$. In general,

$$\sum_{|\alpha|=d} \xi^\alpha \binom{d}{\alpha} c(\alpha) = \xi_i^d \sum_{|\beta|\le d} \eta^\beta c'(\beta)$$

for $\beta := (\alpha_1, \ldots, \alpha_{i-1}, \alpha_{i+1}, \ldots, \alpha_{m+1})$ and $\eta := (\frac{\xi_1}{\xi_i}, \ldots, \frac{\xi_{i-1}}{\xi_i}, \frac{\xi_{i+1}}{\xi_i}, \ldots, \frac{\xi_{m+1}}{\xi_i})$. That is, ξ_i is pulled outside the summation and the coefficients of the modified power form are $\binom{d}{\beta} c(\alpha)$. By choosing i to be the index corresponding to the largest barycentric coordinate, the transformation is stable and the stability of the evaluation is the same as for nested multiplication; [SV86] explains the bivariate case. In view of Section 4, we note that extraction of an isoparametric line from the transformed polynomial by keeping all variables fixed except for η_j amounts to tracing the patch along rays that emanate from the j^{th} vertex. This is different from tracing along isoparametric lines and does not cover the domain uniformly.

The classical univariate version of nested multiplication or Horner's Scheme (see e.g. [CB80 p.33]) computes the value $p(x)$ of the polynomial p with coefficients $c(0), \ldots, c(d)$ by overwriting $c(0)$ with $p(x)$:

$$c(k) = c(k+1) * x + c(k) \quad \text{for } k = (d-1)..0.$$

Observe that $c(k) = \frac{D^k p_d}{k!}(0)$. With $D^\beta := D_1^{\beta_1} \cdots D_m^{\beta_m}$ and D_i the derivative with respect to the ith variable, [BR90] generalizes the algorithm to the multivariate power form in the following natural way similar to De Casteljau's algorithm. (Passing the argument $p_{m,d}$, means passing m, d and the coefficients $c(\beta)$.)

NestMult $(p_{m,d}, x; p_{m,d}(x))$

$c(\beta) = D^\beta p_d(0)/|\beta|! \quad$ for $|\beta| = d$.
for $|\beta| = (d-1)..0$
  $c(\beta) = D^\beta p_d(0)/|\beta|! + \sum_{i=1}^m x_i c(\beta + e_i)$
$p_{m,d}(x) = c(0)$.

(8.3) Example. If $p_{2,3} = y^3 + 4x^2 + 2xy + 3x + 1$, then NestMult$(p_{2,3}, 2; 39)$. First the algorithm computes $c(\beta) = \begin{cases} 3!/3! & \text{if } \beta = (03) \\ 0 & \text{else} \end{cases}$, then

$$c(20) = 2c(30) + 2c(21) + \frac{1}{2}D_1^2 p(0) = 0 + 0 + 4$$
$$c(11) = 2c(21) + 2c(12) + \frac{1}{2}D_1 D_2 p(0) = 0 + 0 + 1$$
$$c(02) = 2c(12) + 2c(03) + \frac{1}{2}D_2^2 p(0) = 0 + 2 + 0$$

and finally

$$\begin{array}{lll} c(20) = 4 & & \\ & c(10) = 8 + 2 + 3 & \\ c(11) = 1 & & c(00) = 26 + 12 + 1 = 39 \\ & c(01) = 4 + 2 + 0 & \\ c(02) = 2 & & \end{array}$$

■

The algorithm is easy to implement and generates one point in $2m\binom{d-1+m}{m}$ operations. Since the algorithm overwrites the input polynomial, only $\binom{d+m}{m}$ space is necessary. In fact, by initializing $c(\alpha)$ only with $D^\alpha p_d(0)/|\alpha|!$ for $|\alpha| = d$ and computing $D^\alpha p_d(0)/|\alpha|!$ for $|\alpha| < d$ while iterating, the space requirement for $c(\alpha)$ can be reduced to $\binom{d+m-1}{m-1}$. The level of indirection is $l = d$.

The value of the dth divided difference of a polynomial of degree d is independent of the parameter value. Thus it is possible and efficient to obtain points at equal parameter intervals by extrapolating the difference table as follows.

```
ForDiff (P(0), ..., P(d))        [m = 1]
for l = 1..d        [build the difference table]
  for j = 0..d − l
    P(j) = P(j + 1) − P(j);
for i = d + 1..n        [extrapolate the table]
  for l = 1..d
    P(l) = P(l) + P(l − 1)
  output P(d)
```

The method does not depend on a particular representation of the polynomial; it can start with just $d+1$ equally spaced points. There are numerous improvements of the above generic forward differencing scheme and extensions to multivariate polynomials (see e.g. [V88a], [LSP87]). ForDiff requires little coding and only const$\binom{d+m}{m}$ space. While for a large number of evaluations in the univariate case, the cost of building the difference table is negligible, this cost has to be considered when the domain is a simplex since there are fewer and fewer evaluations per univariate isoparametric slice as the evaluation moves towards the apex. The main problem with ForDiff as stated above is loss of stability with increasing degree due to round-off. The start requires subtracting terms of similar magnitude and subsequent evaluations do not work with the original divided differences but with the newly computed differences. Thus $\ell = n^m$, where $n = d2^\sigma$.

The difference interpolation method, introduced by Volk in [V88], evaluates at many equidistant points along the real line. The idea is to improve stability of the forward difference calculation by reducing the level of indirection to $\ell = (n/\lambda)^m$, where $\lambda \geq 2$. Thus forward differences Δ_h with a short step length h are computed from differences $\Delta_{\lambda h}$ by (back-)solving a $d \times d$ upper triangular system of the form ([V90, (1.5)].

$$G\Delta_h = \Delta_{\lambda h} + C.$$

The entries in G and C can be computed stably according to the reference [V90]. However, the entries in G increase rapidly with d: at least one entry is of size $\binom{d(d-2)}{d(d-2)/2}$ (cf. [V90 (2.3)]). Like ForDiff, D.I.M. does not depend on a particular representation of the polynomial, but needs only $d+1$ points to start. D.I.M. can also be extended to multivariate polynomials. Terms of similar magnitude are subtracted to start the algorithm and this may cause instability. Extensive pseudocode of D.I.M. is given in [V88].