These pages are intended for the audience of my lecture on the Mathematics of Computer–Aided Design, Univ. Göttingen 2010. The text is based on an earlier manuscript I wrote for 1999/2000 and 2002/2003, but it is extended at certain places. It requires some knowledge of standard material that is accessible in almost every book on Computer–Aided Design. I concentrate on some nonstandard stuff here. In particular, the way of introducing B–splines seems to be rather unusual, but efficient.

R. Schaback, TEXed 4. Mai 2010

1 Some Simple Splines

Splines are piecewise polynomial functions or curves, which are patched together in a suitable way to form smooth functions or curves. Each part can be called a “patch”, and the main problem is to handle such curves together with their smoothness conditions at the patch boundaries.

1.1 Polygonal curves

The simplest spline is the “connect–the–dots” polygonal line, which is C^0 and has no serious continuity problem.

1.2 Quadratic Splines

The next case will consist of pieces of quadratic polynomials, which we could write as Bernstein–Bezier patches with control nets $[b_{2i}, b_{2i+1}, b_{2i+2}]$ for $i = 0, \ldots, n$ to make them automatically continous at the patch boundary points b_{2i}. If we parametrize each patch over an interval of the same length, we get the C^1 conditions

$$\frac{b_{2i+1} + b_{2i-1}}{2} = b_{2i},\ 1 \le j \le n-1.$$

If we consider the b_{2i} as given, the remaining control points can be expressed via

$$b_{2i+1} = 2b_{2i} - b_{2i-1}$$

recursively, such that in general

$$b_{2i+1} = \sum_{j=0}^{i-1} (-1)^j b_{2i-2j} + (-1)^i b_1,\ 1 \le i \le n-1.$$

and finally

$$b_{2n-1} = \sum_{j=0}^{n-2} (-1)^j b_{2n-2-2j} + (-1)^{n-1} b_1.$$

This means that the control points b_1 and b_{2n-1} are related by

$$b_{2n-1} - (-1)^{n-1} b_1 = \sum_{j=0}^{n-2} (-1)^j b_{2n-2-2j}$$

where the right–hand side is given by the data. Conversely, any choice of control points b_1 and b_{2n-1} with this property will lead to a C^1 curve interpolating the points b_{2i}.

But one can proceed differently, assumeing points b_{2i+1} for $-1 \leq i \leq n$ to be given, with two auxiliary points b_1 and b_{2n+1} at the end. Then

$$b_{2i} := \frac{b_{2i+1} + b_{2i-1}}{2}, \; 0 \leq j \leq n$$

will lead to a C^1 spline curve without problems.

If we now look at subdivision at midpoints, we see that we have a case of "corner cutting". The partial control polygon $[b_{-1}, b_1, \ldots, b_{2n-1}, b_{2n+1}]$ gets its corners gut at 1/4 distance to the next point, and we know by the standard subdivision argument that this will generate the pecewise quadratic curve. This simple scheme is due to Chaikin, and it is the first case of a subdivision that can be implemented as a simple geometric rule on a control net.

1.3 Cubic Splines

Assume that we want to patch together a C^2 curve consisting of cubic patches with control nets $[b_{3i}, b_{3i+1}, b_{3i+2}, b_{3i+3}]$ for $i = 0, \ldots, n-1$ with a total of $3n+1$ control points in $I\!R^d$, since we already made sure by notation that the curve will be C^0 at the b_{3i} for $i = 1, \ldots, n-2$. If we assume each patch to be defined over a parameter interval of the same length, the smoothness conditions will be

$$\begin{array}{rcl} b_{3i+1} - b_{3i} &=& b_{3i} - b_{3i-1} \\ b_{3i+2} - 2b_{3i+1} + b_{3i} &=& b_{3i} - 2b_{3i-1} + b_{3i-2} \end{array}$$

for $i = 1, \ldots, n-2$. Like for univariate classical splines, we assume the $b_{3i} =: f_i$ to be given and use "second derivatives at the junctions" as unknowns. These are

$$\begin{array}{rcl} s_i &:=& b_{3i} - 2b_{3i-1} + b_{3i-2} \\ &=& b_{3i+2} - 2b_{3i+1} + b_{3i} \end{array}$$

where we have put the C^2 transition condition into the notation. We then have to show how the other control points can be derived from the s_i and the $b_{3i} = f_i$, and we have to write down the C^1 condition.

We start with

$$\begin{array}{rcl} s_{i+1} + 2s_i &=& b_{3i+3} - 2b_{3i+2} + b_{3i+1} + 2b_{3i+2} - 4b_{3i+1} + 2b_{3i} \\ &=& b_{3i+3} - 3b_{3i+1} + 2b_{3i} \\ s_{i+1} + 2s_i - f_{i+1} + f_i &=& -3(b_{3i+1} - b_{3i}) \\ s_{i-1} + 2s_i &=& b_{3i-1} - 2b_{3i-2} + b_{3i-3} + 2b_{3i} - 4b_{3i-1} + 2b_{3i-2} \\ &=& b_{3i-3} - 3b_{3i-1} + 2b_{3i} \\ s_{i-1} + 2s_i - f_{i-1} + f_i &=& -3(b_{3i-1} - b_{3i}) \end{array}$$

and note that we can use the third and sixth equation to get the missing control points from the unknowns and the given data via

$$\begin{array}{rcl} s_{i+1} + 2s_i - f_{i+1} - 2f_i &=& -3b_{3i+1}, \\ s_{i-1} + 2s_i - f_{i-1} - 2f_i &=& -3b_{3i-1}. \end{array} \qquad (1.1)$$

The C^1 condition then is

$$\begin{aligned} 0 &= -3(b_{3i-1}-b_{3i}) - 3(b_{3i+1}-b_{3i}) \\ &= s_{i+1} + 2s_i - f_{i+1} + f_i + s_{i-1} + 2s_i - f_{i-1} + f_i \\ &= s_{i+1} + 4s_i + s_{i-1} - f_{i+1} + 2f_i - f_{i-1} \end{aligned}$$

or, as a vector equation,

$$s_{i+1} + 4s_i + s_{i-1} = f_{i+1} - 2f_i + f_{i-1},\ 1 \le i \le n-1.$$

For $i = 0$ and $i = n$ we get the equations

$$\begin{array}{rcrcl} s_1 + 2s_0 & - & f_1 + f_0 & = & -3(b_1 - b_0) \\ s_{n-1} + 2s_n & - & f_{n-1} + f_n & = & -3(b_{3n-1} - b_{3n}) \end{array}$$

and assume b_1 and b_{3n-1} to be given. Altogether we get a block system of equations with a matrix

$$\begin{pmatrix} 2I & I & 0 & 0 & \dots & 0 \\ I & 4I & I & 0 & & 0 \\ 0 & I & 4I & I & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \vdots \\ 0 & & & I & 4I & I \\ 0 & \dots & \dots & 0 & I & 2I \end{pmatrix}$$

which has three bands of scaled $d \times d$ identity matrices I. We can deal with it blockwise as if we had the real–valued matrix

$$\begin{pmatrix} 2 & 1 & 0 & 0 & \dots & 0 \\ 1 & 4 & 1 & 0 & & 0 \\ 0 & 1 & 4 & 1 & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \vdots \\ 0 & & & 1 & 4 & 1 \\ 0 & \dots & \dots & 0 & 1 & 2 \end{pmatrix}$$

which is diagonally dominant and has eigenvalues between 1 and 6 due to Gerschgorin's theorem, independent of its size. Thus the system is solvable by $\mathcal{O}(n)$ operations. It can be checked that the solution in terms of the s_i and after use of (1.1) is indeed C^2.

This construction will have drawbacks if the data $f_i = b_{3i}$ are very unevenly distributed. In such a case, one can come closer to arclength parametrization if the patch between $f_i = b_{3i}$ and $f_{i+1} = b_{3i+3}$ is parametrized over an interval of length $\|f_i - f_{i+1}\|_2$. Now the transition conditions are somewhat more complicated.

We shall come back to the more general construction of cubic spline curves in Section 3. bla

2 Multiaffine Forms

2.1 Elementary Properties

Definition 2.1 *A mapping*

$$M\ :\ \left(I\!R^k\right)^n \to I\!R^d$$

is a **symmetric multiaffine form** *of degree n on $I\!R^k$ with values in $I\!R^d$, iff*

1. *M is affine in each argument,*
2. *the values of M are independent of the order of the arguments.*

The set of such M is denoted by $SMAF(n,k,d)$. Symmetric multiaffine forms are also called **polar forms** *or* **blossoms***.*

If an argument x of a symmetric multiaffine form M is repeated r times, we write $r\#x$ instead of $x,x,\ldots,x$ with r repetitions. We suggest to read $r\#x$ as "r times x". For an affine combination $c = \lambda a + (1-\lambda)b$ of a vector c in terms of a and b and a scalar λ we shall frequently use

$$M(c,\ldots) = \lambda M(a,\ldots) + (1-\lambda)M(b,\ldots)$$

either from left to right (i. e. replacing c by a and b) or right to left (i.e. coalescing a and b into c).

Note that the arguments of M are vectors in general, and M is vector–valued. The image space $\mathbb{R}^d$ will usually be of dimension $d = 2, 3, 4$, and k will be 1 for curves and 2 for surfaces.

Readers may consider multiaffine forms as a rather exotic subject. But if we write a univariate real–valued polynomial p with real roots $x_1,\ldots,x_n$ as

$$p(x) = \prod_{i=1}^{n}(x - x_i)$$

it is clear that we have a symmetric multiaffine form of $x_1,\ldots,x_n$ for each fixed x. Furthermore, all coefficients of p are multiaffine forms, and for $n = 2$ we get the three examples

$$\begin{array}{rcl} M(x_1,x_2) & = & 1 \\ M(x_1,x_2) & = & -(x_1 + x_2) \\ M(x_1,x_2) & = & x_1 \cdot x_2 \end{array}$$

from the coefficients of p. This can easily be generalized to

$$p(x) = \sum_{j=0}^{n} x^n (-1)^{n-j} S_{n-j}^n(x_1,\ldots,x_n) \tag{2.1}$$

with the classical *elementary symmetric functions*

$$S_j^n(x_1,\ldots,x_n) := \sum_{\substack{j_1 + j_2 + \ldots + j_n = j \\ j_k \in \{0,1\}}} x_1^{j_1} x_2^{j_2} \cdots x_n^{j_n} \tag{2.2}$$

having the property

$$S_j^n(n\#x) = \binom{n}{j} x^j. \tag{2.3}$$

In fact, evaluating the product form of p via the multinomial formula yields both (2.1) and (2.2), while (2.3) follows from the fact that there are $\binom{n}{j}$ indices in (2.2), because $\binom{n}{j}$ is the number of ways one can pick j out of n items.

Equation (2.3) is a fundamental observation when read from right to left. It then means that each monomial x^j can be written as

$$x^j = \frac{1}{\binom{n}{j}} S_j^n(n\#x)$$

via a symmetric multiaffine form S_j^n of some order $n \geq j$ with fully coalescing arguments. This fact will be proven below, but we start with an easier fact:

Lemma 2.1 *Let a polar form $M \in SMAF(n,k,d)$ be given. If an argument $t \in I\!R^k$ of M occurs precisely j times, and if all other arguments are kept fixed, the form M is a k–variate polynomial of degree at most j as a function of t.*

Proof: We proceed by induction over j, and the case $j = 0$ is trivial. Let the assertion hold for $j < n$, and look at $M(j+1\#t, x_{j+2}, \ldots, x_n)$. We write

$$t = \sum_m t_m e_m + \left(1 - \sum_m t_m\right) 0$$

and get

$$\begin{array}{rcl} M(j+1\#t, x_{j+2}, \ldots, x_n) & = & M(j\#t, t, x_{j+2}, \ldots, x_n) \\ & = & \sum\limits_m t_m M(j\#t, e_m, x_{j+2}, \ldots, x_n) \\ & & + \left(1 - \sum\limits_m t_m\right) M(j\#t, 0, x_{j+2}, \ldots, x_n) \end{array}$$

and get the assertion. □

Note how the trick of the above proof can be used repeatedly to evaluate M in terms of its values on unit vectors and zero.

Corollary 2.1 *For any polar form $M \in SMAF(n,k,d)$ the function $M(n\#x)$ is a k–variate polynomial of degree at most n with values in $I\!R^d$ as a function of x.*

Lemma 2.2 *Any symmetric multiaffine form $M \in SMAF(n,k,d)$ for $n > 1$ can be uniquely written as*

$$M(x_1, \ldots, x_n) = A(x_2, \ldots, x_n)x_1 + B(x_2, \ldots, x_n)$$

with $A \in SMAF(n-1, k, d \cdot k)$ and $B \in SMAF(n-1, k, d)$.

Proof: As M is an affine function of x_1, one can clearly write M in the above form, but it must be shown that A and B are multiaffine forms of $x_2, \ldots, x_n$ and unique. For B we get this due to

$$B(x_2, \ldots, x_n) = M(0, x_2, \ldots, x_n).$$

Furthermore, the action of A as a matrix on a vector z is

$$A(x_2, \ldots, x_n)z = M(z, x_2, \ldots, x_n) - M(0, x_2, \ldots, x_n)$$

and thus A is clearly symmetric and multiaffine in $x_2, \ldots, x_n$. □

Note that Lemma 2.2 also yields an easy inductive proof of Corollary 2.1.

We now go for the already announced converse result:

Theorem 2.1 *For each k–variate polynomial P of degree at most n with values in $I\!R^d$ there is precisely one polar form $M_P \in SMAF(n,k,d)$ such that $P(x) = M_P(n\#x)$.*

Proof: We first concentrate on the existence of M_P. For the univariate real–valued case we can use the elementary symmetric functions and (2.3) to get the result for the monomials, and by linear superposition also for arbitrary polynomials. The vector–valued case can be done componentwise from the scalar–valued case, and we are left with the scalar–valued multivariate case. But we can vary (2.2) for

$$S^n_\alpha(x_1,\ldots,x_n) = \sum_{\substack{\alpha^1+\ldots+\alpha^n=\alpha \\ \alpha^j \in \{0,1\}^k}} x_1^{\alpha^1}\cdots x_n^{\alpha^n} \text{ for all } \alpha \in Z\!\!Z^k_{\geq 0},\ x_1,\ldots,x_n \in I\!R^k$$

in standard multivariate notation

$$z^\beta = \prod_{j=1}^k z_j^{\beta_j} \text{ for all } z \in I\!R^k,\ \beta \in Z\!\!Z^k_{\geq 0}$$

and see that x^α and $S^n_\alpha(n\#x)$ coincide up to a positive factor.

We now have to prove uniqueness of M_P for given P. Assume that a polar form $M \in SMAF(n,k,d)$ with $M(n\#x) = 0$ for all $x \in I\!R^k$ is given, and we have to show that $M(x_1,\ldots,x_n)$ is zero for all $x_j \in I\!R^k$. We proceed by induction on n and state that the case $n=1$ is trivial. Writing

$$M(x_1,\ldots,x_n) = A(x_2,\ldots,x_n)x_1 + B(x_2,\ldots,x_n)$$

with $A \in SMAF(n-1,k,d\cdot k), B \in SMAF(n-1,k,d)$ due to Lemma 2.2, we get $\nabla_{x_1}M(x_1,\ldots,x_n) = A(x_2,\ldots,x_n)$. Now

$$\begin{aligned} 0 &= \nabla_x M(n\#x) \\ &= \sum_{j=1}^n \nabla_{x_j} M(x_1,\ldots,x_n)|_{n\#x} \\ &= n\nabla_{x_1} M(x_1,\ldots,x_n)|_{n\#x} \\ &= nA(n-1\#x) \end{aligned}$$

because M is symmetric. By induction, the matrix–valued polar form A must vanish, and we get $B(n-1\#x) = 0$. Again, induction yields $B = 0$. □

At this point, we can look more closely at the polar form of the gradient $\nabla_x P$ of a multivariate polynomial P corresponding to a multiaffine form M_P. Since the derivatives with respect to all arguments are equal, we have

$$\nabla_x P(x) = n \cdot \nabla M_P(n\#x) = n \cdot (\nabla_{x_1} M_P(x_1,\ldots,x_n))_{|n\#x} = n \cdot A(x_2,\ldots,x_n))_{|(n-1)\#x}$$

using A and B as above. This implies

$$M_{\nabla_x P}(x_2,\ldots,x_n) = n \cdot A(x_2,\ldots,x_n)$$

because the multiaffine form for $\nabla_x P$ is unique and $n\cdot A$ does the job. For any x and $x+h \in I\!R^k$ we can write

$$A(x_2,\dots,x_n)h = M_P(x+h,x_2,\dots,x_n) - M_P(x,x_2,\dots,x_n),$$

and thus we get

$$M_{\nabla_x P}(x_2,\dots,x_n)h = n(M_P(x+h,x_2,\dots,x_n) - M_P(x,x_2,\dots,x_n))$$

for all $x,h \in I\!R^k$. If all arguments are scalar (i.e. $k=1$), then

$$M_{P'}(x_2,\dots,x_n) = \frac{n}{h}(M_P(x+h,x_2,\dots,x_n) - M_P(x,x_2,\dots,x_n)) \tag{2.4}$$

for all $x,h \in I\!R$ with $h \neq 0$. We shall use (2.4) later.

Let us look at directional derivatives $\nabla_x^T r$ in the case of nonscalar arguments. We get

$$M_{(\nabla_x^T r)P}(x_2,\dots,x_n) = n\cdot A(x_2,\dots,x_n)r = n\cdot(M_P(y+r,x_2,\dots,x_n) - M_P(y,x_2,\dots,x_n)) \tag{2.5}$$

for arbitrary y by the same line of argumentation.

2.2 The Algorithm of de Casteljau Revisited

Assume that we have a polynomial curve of degree $n \in I\!R^d$ over $[\alpha,\beta] \subset I\!R$ in Bernstein–Bézier form

$$P(t) := \sum_{j=0}^{n} b_j \left(\frac{t-\alpha}{\beta-\alpha}\right)^j \left(\frac{\beta-t}{\beta-\alpha}\right)^{n-j} =: BB[b_0,\dots,b_n]_{[\alpha,\beta]}$$

with control points $b_j \in I\!R^d$. We can express the control points as values of the unique symmetric multiaffine form M_P associated with P:

Theorem 2.2 *For any polynomial curve P of degree n in $I\!R^d$ over $[\alpha,\beta] \subset I\!R$ in Bernstein–Bézier form, the control points $b_0, b_1, \dots, b_n$ are representable as*

$$b_j = M_P(n-j\#\alpha, j\#\beta),\ 0 \le j \le n.$$

Proof: We define c_j by the right–hand side of the above representation, and we define Q to be the Bernstein–Bézier polynomial curve with control points c_j. We shall first prove $P = Q$ and then get $b_j = c_j$ from the uniqueness of the Bernstein–Bézier representation.

If we define $c_j^n := c_j$ and proceed by the deCasteljau algorithm for some $t \in [\alpha,\beta]$ as

$$c_j^r = \left(\frac{t-\alpha}{\beta-\alpha}\right) c_{j+1}^{r+1} + \left(\frac{\beta-t}{\beta-\alpha}\right) c_j^{r+1},$$

we get

$$c_j^r = M_P(r-j\#\alpha, n-r\#t, j\#\beta),\ 0 \le j \le r \le n$$

by induction

$$\begin{array}{rl}
\left(\frac{t-\alpha}{\beta-\alpha}\right) c_{j+1}^{r+1} & + \\
\left(\frac{\beta-t}{\beta-\alpha}\right) c_{j}^{r+1} & = \\
\left(\frac{t-\alpha}{\beta-\alpha}\right) M_P(r-j\#\alpha, n-r-1\#t, j+1\#\beta) & + \\
\left(\frac{\beta-t}{\beta-\alpha}\right) M_P(r+1-j\#\alpha, n-r-1\#t, j\#\beta) & \\
 & = \\
M_P(r-j\#\alpha, n-r\#t, j\#\beta) &
\end{array}$$

because of the affine combination

$$t = \left(\frac{t-\alpha}{\beta-\alpha}\right)\beta + \left(\frac{\beta-t}{\beta-\alpha}\right)\alpha.$$

Now $c_0^0 = Q(t) = M_P(n\#t) = P(t)$ holds, and we are finished. $\square$

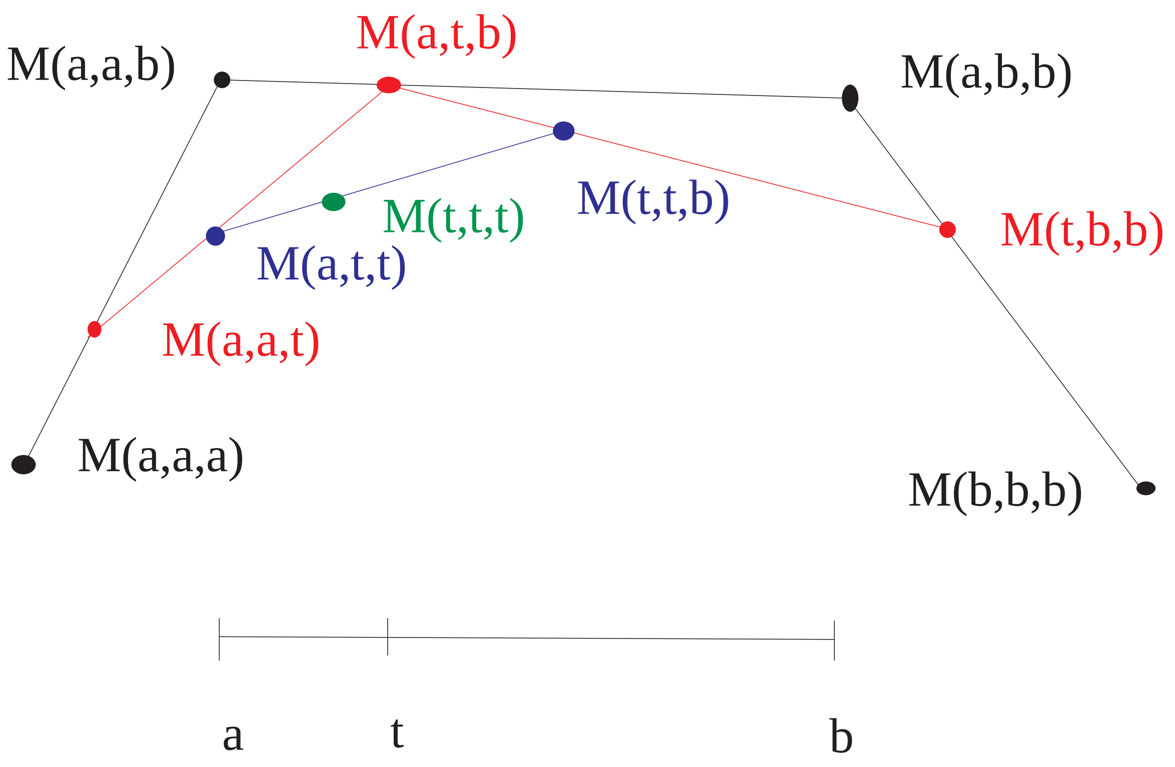
Corollary 2.2 *The intermediate points b_j^r of the deCasteljau algorithm*

$$b_j^r = \left(\frac{t-\alpha}{\beta-\alpha}\right) b_{j+1}^{r+1} + \left(\frac{\beta-t}{\beta-\alpha}\right) b_j^{r+1}$$

starting with $b_j =: b_j^n$ and corresponding to a polynomial P over $[\alpha,\beta]$ satisfy

$$b_j^r = M_P(r-j\#\alpha, n-r\#t, j\#\beta),\ 0 \le j \le r \le n.$$

$\square$



The above figure shows the values of the multiaffine form M corresponding to a Bernsterin–Bézier representation of a polynomial of degree 3 over $[a,b]$, if the de Casteljau algorithm is carried out for an intermediate point t.

2.3 Subdivision

We specialize Corollary 2.2 and get

$$\begin{aligned} b_0^r &= M_P(r\#\alpha, n-r\#t),\ 0 \le r \le n \\ b_r^r &= M_P(n-r\#t, r\#\beta),\ 0 \le r \le n. \end{aligned}$$

With Theorem 2.2 this proves

Theorem 2.3 *If the control points b_j^n, $0 \le j \le n$ represent a polynomial P in Bernstein–Bézier form over $[\alpha,\beta]$, then the deCasteljau control points $b_0^r = M_P(r\#\alpha, n-r\#t)$, $0 \le r \le n$ represent P over $[\alpha, t]$, while $b_r^r = M_P(n-r\#t, r\#\beta)$, $0 \le r \le n$ represent P over $[t,\beta]$.* □

Subdivision can be used recursively for generating approximations to the actual curve. We prove convergence of this process later. The recursion is stopped by a **flatness test** that we analyze now. The basic idea is that the curve is a straight line if the control points lie on a line (by the convex hull property). How far is a curve from a straight line if the control points are "nearly" on a line?

By degree elevation of the line segment $L(t)$ between the control points b_0 and b_n of a polynomial $P(t)$ in Bernstein–Bézier representation over $[\alpha,\beta]$, we get the representation of L by control points $c_j := b_0 + \frac{j}{n}(b_n - b_0), 0 \le j \le n$. Now

$$\begin{aligned} \|P(t) - L(t)\| &= \|\sum_{j=0}^{n}(b_j - c_j)\beta_j^n(t)\| \\ &\le \sum_{j=0}^{n}|\beta_j^n(t)|\|b_j - c_j\| \\ &\le \max_{0\le j\le n}\|b_j - c_j\|\sum_{j=0}^{n}\beta_j^n(t) \\ &= \max_{0\le j\le n}\|b_j - c_j\| \cdot 1 \\ &= \max_{0\le j\le n}\|b_j - b_0 - \frac{j}{n}(b_n - b_0)\| \end{aligned}$$

is a bound on the distance of a point of the curve $P(t)$ to a point on the line $L(t)$ at a parameter $t \in [\alpha,\beta]$. Note that we have used the partition of unity property of Bernstein polynomials $\beta_j^n(t)$ of degree n.

Recursive subdivision is stopped if the above bound is smaller than a prescribed constant. Then either the line segment between b_o and b_n or the control polygon through $b_0,\dots,b_n$ is used as an approximation of the curve segment. The analysis of the error wrt. the latter will be treated later, because it requires spline theory.

2.4 Convergence of Subdivision

We now want to show that the quantity

$$\max_{0\le j\le n}\|b_j - b_0 - \frac{j}{n}(b_n - b_0)\|$$

decreases quadratically during repeated subdivision. More precisely:

Theorem 2.4 *If a polynomial curve P is represented in Bernstein–Bézier form over $[\alpha, \beta]$, there is a constant C depending only on P, α, and β, such that for the representation of P over any subinterval $[\alpha', \beta'] \subseteq [\alpha, \beta]$ with control points $b_0, \dots, b_n$ the error bound*

$$\max_{0 \le j \le n} \|b_j - b_0 - \frac{j}{n}(b_n - b_0)\| \le C(\beta' - \alpha')^2$$

holds.

In other words, subdivision at the midpoint of the interval can be expected to reduce the error wrt. the line through b_0 and b_n by a factor of 0.25 in the limit. Or, each subdivision step usually gives two binary digits of relative accuracy.

Proof: Of course, we represent the control points as

$$b_j = M_P(n - j\#\alpha', j\#\beta'),\ 0 \le j \le n$$

and perform a Taylor expansion of the vector–valued univariate functions

$$g_j(t) := M_P(n - j\#\alpha', j\#\alpha' + t * (j\#1))),\ 0 \le j \le n$$

around zero. Here, we write $j\#\beta' = j\#\alpha' + (\beta' - \alpha') * (j\#1)$ and get, taking points $\xi_j \in [\alpha', \beta']$,

$$\begin{array}{rcl} g_j(\beta' - \alpha') & = & g_j(0) + (\beta' - \alpha')g_j'(0) + (\beta' - \alpha')^2 g_j''(\xi_j)/2 \\ b_j & = & b_0 + j(\beta' - \alpha')M_P'(n\#\alpha') + j^2(\beta' - \alpha')^2 M_P''(n - j\#\alpha', j\#\alpha' + \xi_j * (j\#1))/2 \\ b_n & = & b_0 + n(\beta' - \alpha')M_P'(n\#\alpha') + n^2(\beta' - \alpha')^2 M_P''(n\#\alpha' + \xi_n * (n\#1))/2 \end{array}$$

using symmetry of the j arguments that are used for differentiation. In fact, we have $g_j'(0) = jM_P'(n\#\alpha')$, where we write M_P' for any of the partial derivatives of M_P, which coincide by symmetry. For the same reason we can just write M_P' and M_P'', respectively. Now we evaluate $b_j - b_0 - \frac{j}{n}(b_n - b_0)$ and see that the constant terms drop out right away, while the linear terms vanish due to

$$j(\beta' - \alpha')M_P'(n\#\alpha') - \frac{j}{n}n(\beta' - \alpha')M_P'(n\#\alpha') = 0.$$

The rest is bounded above by

$$n^2(\beta' - \alpha')^2 \sup_{x_1, \dots, x_n \in [\alpha, \beta]} \|M_P''(x_1, \dots, x_n)\|,$$

proving the assertion. □

2.5 Continuity Conditions

We now look at conditions for a C^m transition from a Bernstein–Bézier representation

$$P = BB[b_0, \dots, b_n]_{[\alpha, \beta]}$$

of a polynomial curve P on $[\alpha, \beta]$ to a Bernstein–Bézier representation

$$Q = BB[c_0, \dots, c_n]_{[\beta, \gamma]}$$

of a curve Q on $[\beta, \gamma]$. We assume both degrees to be equal. If this is not satisfied, degree elevation should be applied.

We first look at the polynomials

$$P_k := BB[b_{n-k},\dots,b_n]_{[\alpha,\beta]}, Q_k := BB[c_0,\dots,c_k]_{[\beta,\gamma]}$$

of degree at most k, and defined by a Bernstein–Bézier representation of $k+1$ control points near the transition.

Theorem 2.5 *For $0 \le m \le n$, the polynomials P and Q of degree at most n have a C^m transition at β, if and only if the polynomials P_k and Q_k coincide for $0 \le k \le m$.*

Proof: Let's look at the derivatives:

$$\begin{aligned} Q^{(j)}(\beta) &= \frac{n!}{(n-j)!(\gamma-\beta)^j}\Delta^j c_0, 0 \le j \le n, \\ Q_k^{(j)}(\beta) &= \frac{k!}{(k-j)!(\gamma-\beta)^j}\Delta^j c_0, 0 \le j \le k. \\ P^{(j)}(\beta) &= \frac{n!}{(n-j)!(\beta-\alpha)^j}\Delta^j b_{n-j}, 0 \le j \le n, \\ P_k^{(j)}(\beta) &= \frac{k!}{(k-j)!(\beta-\alpha)^j}\Delta^j b_{n-j}, 0 \le j \le k, \end{aligned}$$

where Δ is the forward difference $\Delta d_j := d_{j+1} - d_j$. The conditions $P^{(j)}(\beta) = Q^{(j)}(\beta)$ are

$$\frac{1}{(\gamma-\beta)^j}\Delta^j c_0 = \frac{1}{(\beta-\alpha)^j}\Delta^j b_{n-j},$$

while $P_k^{(j)}(\beta) = Q_k^{(j)}(\beta)$ are

$$\frac{1}{(\gamma-\beta)^j}\Delta^j c_0 = \frac{1}{(\beta-\alpha)^j}\Delta^j b_{n-j}.$$

Thus a C^m transition from P to Q is equivalent to a C^k transition between P_k and Q_k for all $0 \le k \le m$. Since these polynomials have degree at most k, they must coincide with their own Taylor representation of degree k at β, which is the same for both of them, if and only if they have a C^k transition at β. □

We now look at the multiaffine form

$$M_k(x_1,\dots,x_k) := M_Q(x_1,\dots,x_k,(n-k)\#\beta)$$

of k arguments and compare it with M_{Q_k}. Clearly,

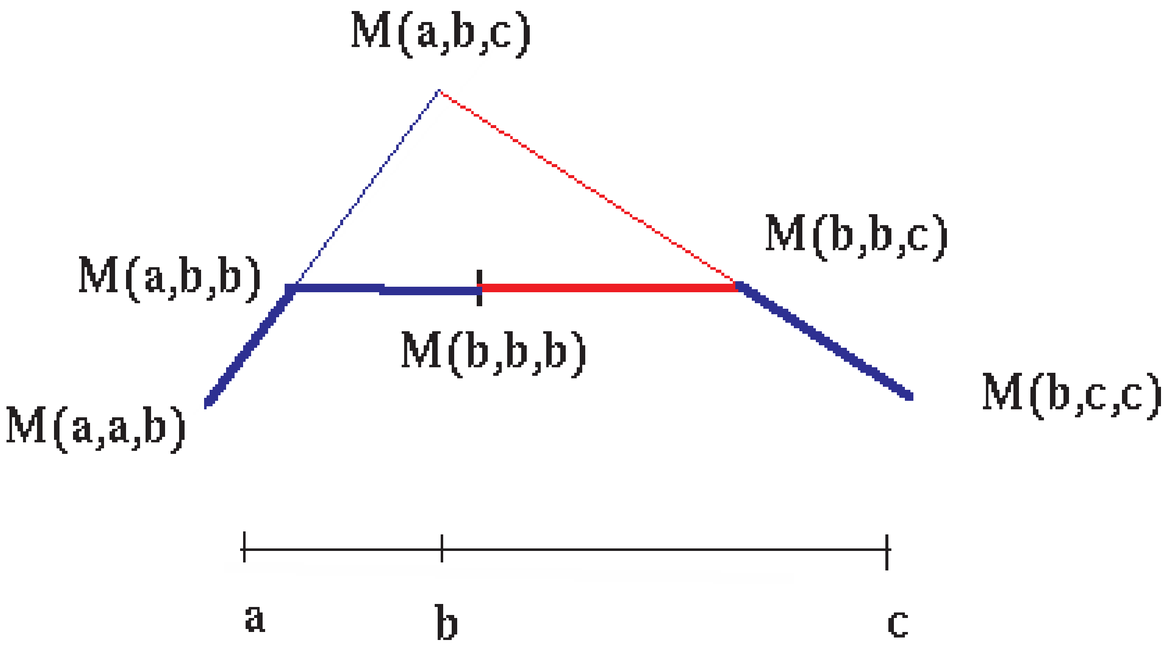
$$\begin{aligned} c_j &= M_Q(n-j\#\beta, j\#\gamma) \\ &= M_Q(n-k\#\beta, k-j\#\beta, j\#\gamma) \\ &= M_k(k-j\#\beta, j\#\gamma) \\ &= M_{Q_k}(k-j\#\beta, j\#\gamma), 0 \le j \le k. \end{aligned}$$

This means that the polynomial Q_k and the polynomial $M_k(k\#t)$ must coincide, because they generate the same Bernstein–Bézier representation. But then M_k and M_{Q_k} are the same. Using this argument on both sides of β, we get

Theorem 2.6 *For $0 \leq m \leq n$, the polynomials P and Q of degree at most n have a C^m transition at β, if and only if the multiaffine forms*

$$M_P(x_1, \dots, x_k, n-k\#\beta) = M_Q(x_1, \dots, x_k, n-k\#\beta)$$

coincide for $0 \leq k \leq m$. $\square$



The above picture shows how two cubic pieces P and Q have a C^2 transition. By the previous theorem, the multiaffine forms of P and Q coincide as long as one of the arguments is b. This is why we can write M instead of M_P or M_Q.

We now go back to the original notation and look at the polynomial $R := P_m = Q_m$ for a C^m transition between P and Q at β. We can represent everything over $[\alpha, \gamma]$ or the subintervals defined by β. We have the Bernstein–Bézier representations

$$\begin{aligned} R(t) &= BB[d_0, \dots, d_m]_{[\alpha,\gamma]} \\ P_m(t) &= BB[b_{n-m}, \dots, b_n]_{[\alpha,\beta]} \\ Q_m(t) &= BB[c_0, \dots, c_m]_{[\beta,\gamma]}, \end{aligned}$$

and by subdivision at β we get

Theorem 2.7 *For $0 \leq m \leq n$, the polynomials P and Q of degree at most n with Bernstein–Bézier representations*

$$\begin{aligned} P(t) &= BB[b_0, \dots, b_n]_{[\alpha,\beta]} \\ Q(t) &= BB[c_0, \dots, c_n]_{[\beta,\gamma]}, \end{aligned}$$

have a C^m transition at β, if and only if the control points $b_{n-m}, \dots, b_{n-1}, b_n = c_0, c_1, \dots, c_m$ are obtained by subdivision of a polynomial $R(t) = BB[d_0, \dots, d_m]_{[\alpha,\gamma]}$ of degree at most m at the intermediate point β. $\square$

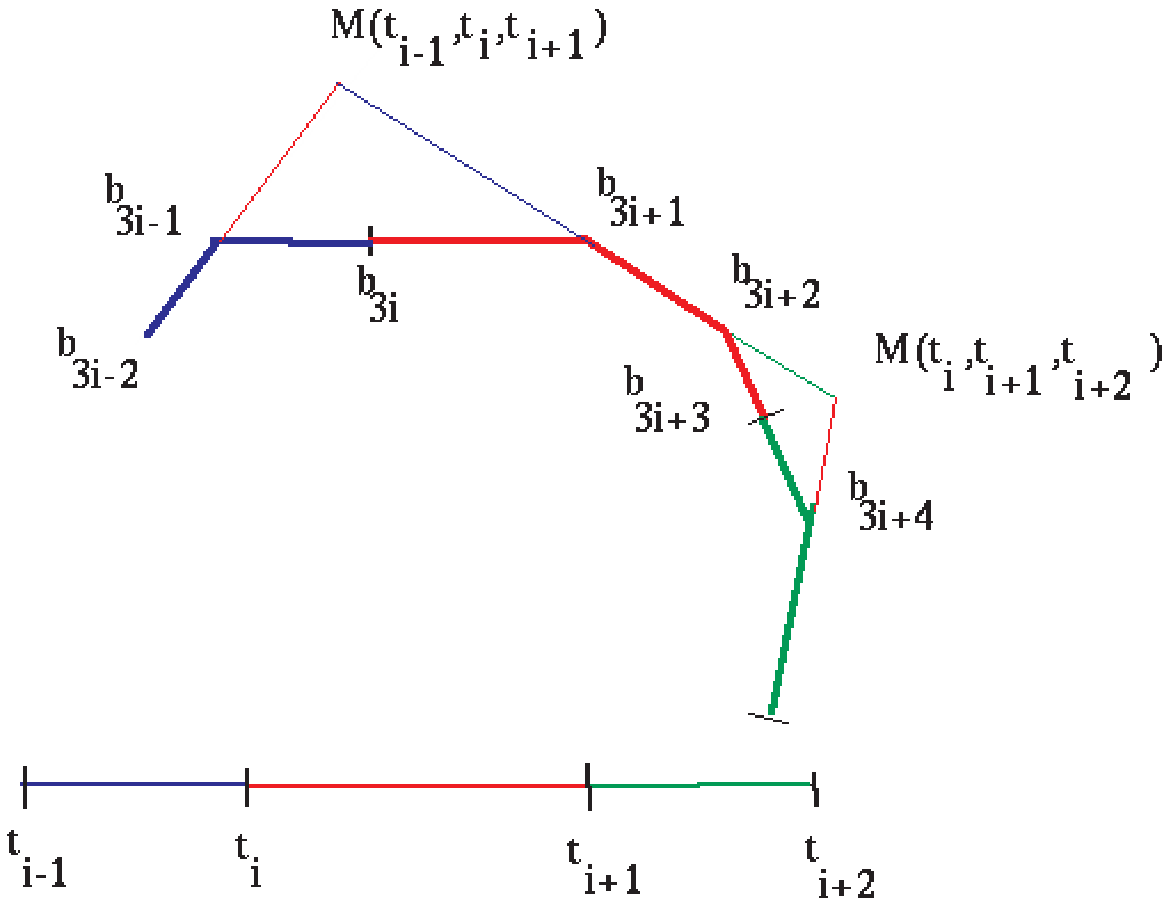
The previous picture illustrates this as well. Just consider the multiaffine form $M_R(x,y) := M(x,y,b)$ that defines R, and you can see the three–point control net of R over $[a,c]$ together with the two control nets generated by subdivision.

2.6 Cubic Spline Curves

We now patch many copies of cubic curves

$$P_i(t) := BB[b_{3i}, b_{3i+1}, b_{3i+2}, b_{3i+3}]_{[t_i,t_{i+1}]}$$

together in such a way that a C^2 curve results.



This works nicely as before, but we can also turn the strategy upside down, starting from the points

$$d_{i-2} := M(t_{i-1}, t_i, t_{i+1}).$$

If we split the line between d_{i-2} and d_{i-1} by two points into three pieces as in the split of $[t_{i-1}, t_{i+2}]$ by t_i and t_{i+1}, we get the control points b_{3i+1} and b_{3i+2}. If we do this for $i-1$ also, we get b_{3i-1} and b_{3i-2}. But then we find b_{3i} between b_{3i-1} and b_{3i+1} via the split of $[t_{i-1}, t_{i+1}]$ at t_i (see the colors in the figure, but note that the affine splits are not to scale). Note further that $M(t_{i-1}, t_i, t_{i+1})$ corresponds to the multiaffine forms of at least two polynomials, one over $[t_{i-1}, t_i]$ and one over $[t_i, t_{i+1}]$. A somewhat more precise analysis by Theorem 2.6 shows that the polynomial P_j over $[t_j, t_{j+1}]$ can be used for all multiaffine form values $d_\ell = M_{P_j}(t_{\ell+1}, t_{\ell+2}, t_{\ell+3})$ that have t_j or t_{j+1} in their argument list, i.e. for $j-3 \le \ell \le j$.

3 Spline Curves

3.1 Carl de Boor's Technique

We now generalize the above approach by brute force, starting from a (formally) biinfinite control net

$$\ldots, d_{\ell-1}, d_\ell, d_{\ell+1}, \ldots \quad \ell \in \mathbb{Z},\ d_\ell \in \mathbb{R}^d$$

and a weakly increasing sequence of **knots**

$$\ldots \le t_{\ell-1} \le t_\ell \le t_{\ell+1} \le \ldots \quad \ell \in \mathbb{Z},\ t_\ell \in \mathbb{R}$$

together with a fixed polynomial degree n. Governed by the previous example, we try to find a curve consisting of polynomial pieces P_j such that

$$d_\ell = M_{P_j}(t_{\ell+1}, \ldots, t_{\ell+n}) \tag{3.1}$$

holds for $j - n \le \ell \le j$, where P_j lives in $[t_j, t_{j+1}]$. Since we now allow coalescing points, we have to assume

$$t_j < t_{j+1} \tag{3.2}$$

for P_j to be well–defined. The basic idea of Carl de Boor's method now is to fix an intermediate point $t \in [t_j, t_{j+1}]$ and use affine combinations of the d_ℓ as if the hypothesis (3.1) were satisfied, ending up in $P_j(t) = M_{P_j}(n\#t)$.

More precisely, we start with

$$d_\ell^n := d_\ell,\ j - n \le \ell \le j,$$

and generate points $d_\ell^{n-r}(t)$ that hopefully satisfy

$$d_\ell^{n-r}(t) = M_{P_j}(r\#t, t_{\ell+1}, \ldots, t_{\ell+n-r}),\ j - (n-r) \le \ell \le j.$$

The rules for the transition $r \to r+1$ can be easily obtained from our heuristics, if we take

$$\begin{array}{rcll} d_\ell^{n-r}(t) &=& M_{P_j}(r\#t, t_{\ell+1}, \ldots, t_{\ell+n-r}), & j-(n-r) \le \ell \le j \\ d_{\ell-1}^{n-r}(t) &=& M_{P_j}(r\#t, t_\ell, \ldots, t_{\ell+n-r-1}), & j-(n-r) \le \ell - 1 \le j \end{array} \tag{3.3}$$

and compose t affinely by t_ℓ and $t_{\ell+n-r}$. For this composition as

$$t = \frac{t_{\ell+n-r} - t}{t_{\ell+n-r} - t_\ell} t_\ell + \frac{t - t_\ell}{t_{\ell+n-r} - t_\ell} t_{\ell+n-r}$$

we require

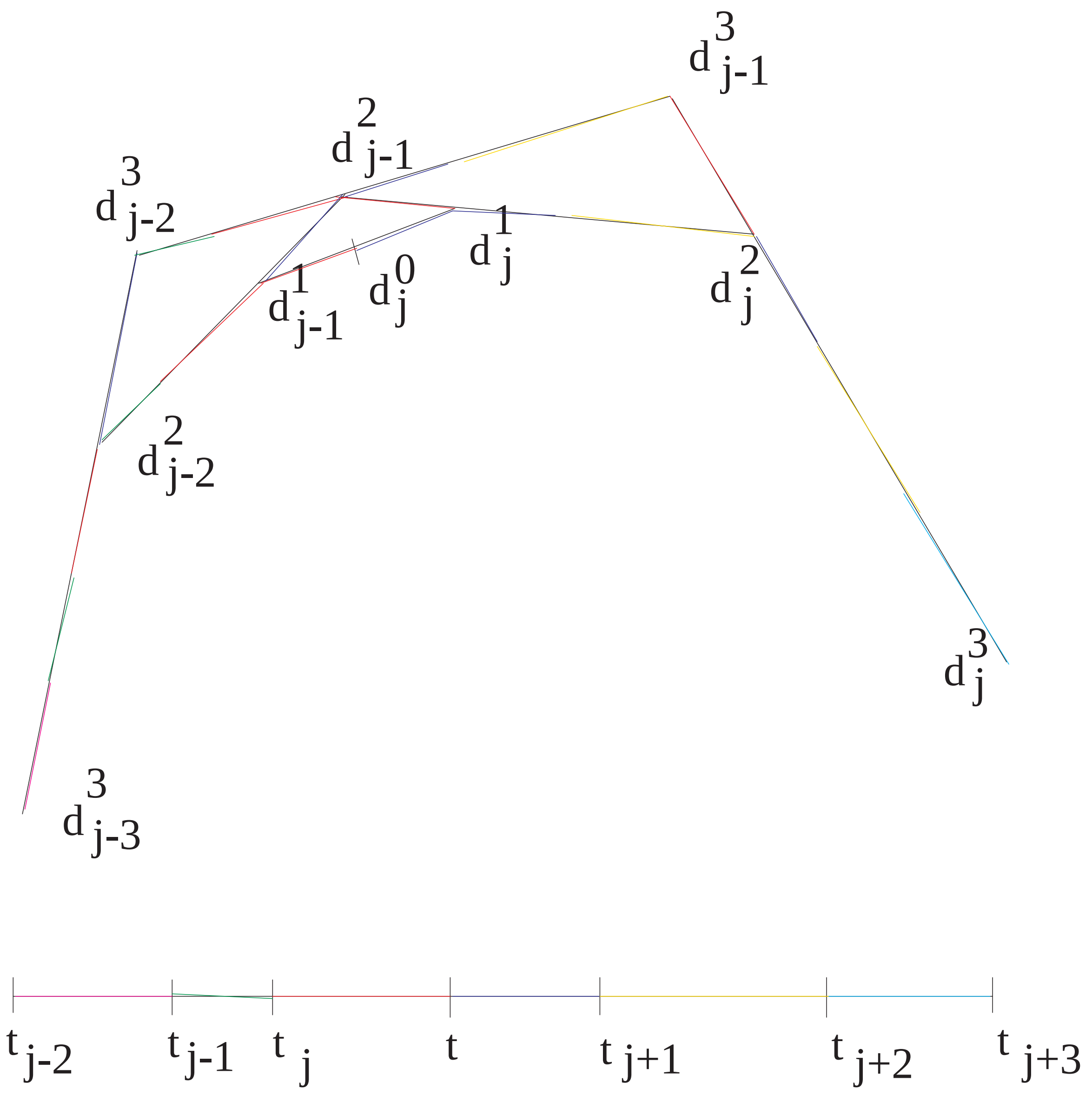
$$t_\ell \le t_j < t_{j+1} \le t_{\ell+n-r},$$

but this follows from $\ell \le j$ and $j - (n-r) \le \ell - 1$ in (3.3). Thus we define the recursion of **de Boor's algorithm** as

$$d_\ell^{n-r-1}(t) := \frac{t_{\ell+n-r} - t}{t_{\ell+n-r} - t_\ell} d_{\ell-1}^{n-r}(t) + \frac{t - t_\ell}{t_{\ell+n-r} - t_\ell} d_\ell^{n-r}(t),\ \ j-(n-r-1) \le \ell \le j,\ \ 0 \le r \le n. \tag{3.4}$$

This procedure is independent of our heuristics, and it clearly yields a polynomial P_j on $[t_j, t_{j+1}]$ via

$$P_j(t) := d_j^0(t).$$



The above picture shows how the de Boor algorithm works. The plot is not completely to scale, but note how the colours indicate the correct splitting of affine combinations of t from knots t_k and t_m. The first sweep calculates the three values of d^2 by picking three sets of four consecutive knots that overlap t. The second sweep takes two sets of three consecutive knots that overlap t, while the third sweep just picks t_j and t_{j+1}, i.e.one set of two knots that overlap t. The red part is always to the left of the new de Boor point, while the blue part is always to the right, the other colours following appropriately.

We now want to prove that our heuristics is correct. The method of de Boor defines a linear mapping

$$B \ : \ (d_{j-n}, \ldots, d_j) \mapsto P_j$$

from $I\!R^{d(n+1)}$ into the space of polynomials of degree up to n with values in $I\!R^d$, and this space of polynomials also is of dimension $d(n+1)$. The mapping A that associates to each polynomial P_j the $n+1$ vectors $d_\ell := M_{P_j}(t_{\ell+1}, \ldots, t_{\ell+n})$ maps the polynomials into a subspace C of $I\!R^{d(n+1)}$. On this subspace, the mapping B inverts A, because by our heuristics the method of de Boor recovers P_j when started from control points that actually come from P_j. But this argument proves that B is surjective, and by equality of dimensions of range and domain of B, we get that B must be bijective. This proves:

Theorem 3.1 *Under the hypothesis* (3.2)*, the algorithm of de Boor constructs a piecewise polynomial (***spline***) curve out of a control net* $\{d_\ell\}_{\ell\in\mathbb{Z}}$ *such that the polynomial piece* P_j *on* $[t_j, t_{j+1})$ *is of degree at most* n *and such that* (3.1) *holds for* $j-n \le \ell \le j$*. The construction uses* $d_{j-n}, \dots, d_j$ *only, and performs convex combinations of these points.* □

The dimension argument also proves

Corollary 3.1 *A multiaffine form* M *of order* n *with arguments in* $\mathbb{R}$ *is completely determined by its values* $d_\ell := M(t_{\ell+1}, \dots, t_{\ell+n})$, $j-n \le \ell \le j$ *for points satisfying*

$$t_{j-n+1} \le \dots \le t_j < t_{j+1} \le \dots \le t_{j+n}.$$

Proof: In fact, the de Boor algorithm based on these data will construct a unique polynomial P_j with $M = M_{P_j}$. □

Note that not necessarily all control points d_ℓ are used for the de Boor algorithm on nondegenerate intervals when the degree n is fixed. We can define the set

$$L_n := \{\ell \in \mathbb{Z} \ : \ j-n \le \ell \le j \text{ for some } j \text{ with } t_j < t_{j+1}\}. \tag{3.5}$$

of all indices that are actually used. If some index k is not in L_n, we have $t_k = \dots = t_{k+n+1}$, i.e. the knot t_k is at least $(n+2)$–fold. Or, by some additional elementary argumentation, we can rewrite L_n as

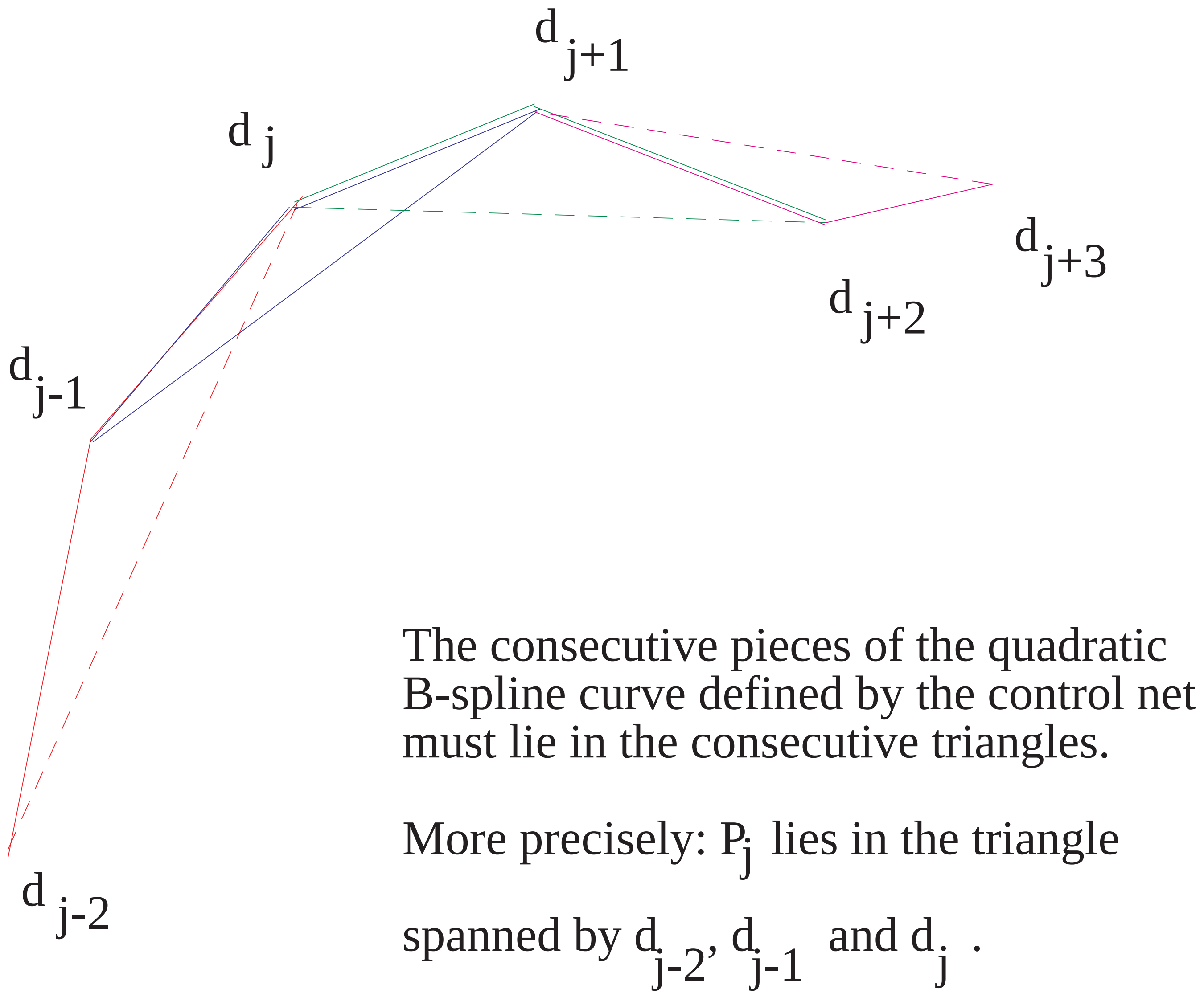
$$L_n := \{\ell \in \mathbb{Z} \ : \ t_\ell < t_{\ell+n+1}\}. \tag{3.6}$$

In fact, if $\ell \in L_n$ from (3.5), then we have

$$t_\ell \le t_j < t_{j+1} \le t_{\ell+n+1}. \tag{3.7}$$

proving that ℓ is in (3.6). The converse is simple: if ℓ comes from (3.6), then there must be some j with (3.7), and this j must satisfy $\ell \le j \le \ell + n$, ans required for (3.5).

Corollary 3.2 *The polynomial piece* P_j *of a spline curve on a nondegenerate interval* $[t_j, t_{j+1})$ *lies within the convex hull of the control points* $d_{j-n}, \dots, d_j$. □



The consecutive pieces of the quadratic B-spline curve defined by the control net must lie in the consecutive triangles.

More precisely: P_j lies in the triangle spanned by d_{j-2}, d_{j-1} and d_j .

Corollary 3.3 *If $t_j < t_{j+1}$ are two knots with multiplicity n, the $n+1$ control points $d_{j-n}, \ldots, d_j$ are control points for the Bernstein–Bézier representation of the polynomial P_j on $[t_j, t_{j+1})$, and the de Boor algorithm for evaluating $P_j(t)$ for $t \in [t_j, t_{j+1})$ coincides with the method of de Casteljau.*

Proof: Due to

$$t_{j-n+1} = \ldots = t_j < t_{j+1} = \ldots = t_{j+n},$$

we have

$$d_\ell = M_{P_j}(t_{\ell+1}, \ldots, t_{\ell+n}) = M_{P_j}((j-\ell)\#t_j, (n-(j-\ell))\#t_{j+1})$$

for $j - n \leq \ell \leq j$ and can use Theorem 2.2 to prove the first assertion. The second follows from inspection of the formulae. A short form of the proof just observes that the k–th sweep of the de Boor method for $k = 1, 2, \ldots, n$ takes $n+1-k$ affine combinations of $n+2-k$ points with the ratio defined by t within consecutive sets of $n+2-k$ consecutive knots. For n–fold knots

t_j and t_{j+1} this ratio is always the ratio of t in $[t_j, t_{j+1}]$, i.e. the de Casteljau ratio, and thus the geometric constructions must coincide. □

The practical consequence of Corollary 3.3 is that we can calculate a Bernstein–Bézier control net for each polynomial piece by insertion of additional knots up to multiplicity n. Knot insertion will be treated later.

Another useful application of Corollary 3.3 occurs at the "endpoints" of a control net. If we just want to work with a finite set $d_0, \ldots, d_{N-1}$ of control points, it is good practice to let the "ends" have the meaning of Bernstein–Bézier control points. This means that the control points

$$d_0 = M(t_1, \ldots, t_n),\ d_{N-1} = M(t_N, \ldots, t_{N+n-1})$$

should be based on coalescing points, i.e. the knot set should satisfy

$$t_1 = \ldots = t_n < t_{n+1} \leq \ldots \leq t_{N-1} < t_N = \ldots = t_{N+n-1}. \tag{3.8}$$

For adjacent spline curves one can use the techniques of section 2.5 to write down the conditions for a C^m transition at the "ends", provided that the knots t_{n+1} and t_{N-1} have been inserted up to multiplicity n.

3.2 Smoothness

We now have to work a little to prove something about the smoothness of the spline curve defined by de Boor's algorithm.

Theorem 3.2 *At a knot t_{j+1} with multiplicity p, $0 \leq p \leq n$, i.e.*

$$t_j < t_{j+1} = t_{j+2} = \ldots = t_{j+p} =: \tau < t_{j+p+1},$$

the spline curve is at least $n - p$ times differentiable.

Proof The adjacent polynomial pieces are P_j and P_{j+p}, respectively. We use Theorem 2.6 for P_j and P_{j+p} at the intermediate p–fold point τ, and we assert

$$M_{P_j}(p\#\tau, x_1, \ldots, x_{n-p}) = M_{P_{j+p}}(p\#\tau, x_1, \ldots, x_{n-p}) \tag{3.9}$$

for all arguments $x_1, \ldots, x_{n-p}$. Now we look at the d_ℓ that are admissible for **both** P_j and P_{j+p}. The corresponding indices ℓ are restricted by $j - n + p \leq \ell \leq j$, and thus we have

$$d_\ell = M_{P_j}(t_{\ell+1}, \ldots, t_{\ell+n}) = M_{P_{j+p}}(t_{\ell+1}, \ldots, t_{\ell+n})$$

for $j - n + p \leq \ell \leq j$. We can rewrite these inequalities in two ways:

$$\begin{array}{ccccc} j-n+p+1 & \leq & \ell+1 & \leq & j+1 \\ j+p & \leq & \ell+n & \leq & j+n \end{array}$$

to read off that there always is a p–fold instance of τ within the arguments $t_{\ell+1}, \ldots, t_{\ell+n}$ due to

$$t_{\ell+1} \leq t_{j+1} = \tau = \ldots = t_{j+p} = \tau \leq t_{\ell+n}. \tag{3.10}$$

The remaining $n-p$ arguments are different from τ. We now drop the p instances of τ from the knot sequence by defining

$$s_i := t_i,\ i \le j,\ s_{j+i} := t_{j+p+i},\ i > 0$$

and get

$$s_{j-n+1} \le \ldots \le s_j < s_{j+1} \le \ldots \le s_{j+n}. \tag{3.11}$$

We define

$$\begin{array}{rcl} N_j(x_1,\ldots,x_{n-p}) & := & M_{P_j}(p\#\tau, x_1,\ldots,x_{n-p}) \\ N_{j+p}(x_1,\ldots,x_{n-p}) & := & M_{P_{j+p}}(p\#\tau, x_1,\ldots,x_{n-p}) \end{array}$$

and get

$$\begin{array}{rclcl} d_\ell & = & M_{P_j}(t_{\ell+1},\ldots,t_{\ell+n}) & = & N_j(s_{\ell+1},\ldots,s_{\ell+n-p}) \\ & = & M_{P_{j+p}}(t_{\ell+1},\ldots,t_{\ell+n}) & = & N_{j+p}(s_{\ell+1},\ldots,s_{\ell+n-p}) \end{array}$$

for $j-n+p \le \ell \le j$ by careful enumeration of the arguments as in (3.10):

$$\begin{array}{ccccc} t_{\ell+1} & \le & t_{j+1} = \tau = \ldots = t_{j+p} = \tau & \le & t_{\ell+n} \\ s_{\ell+1} & \le & & \le & s_{\ell+n-p} \end{array}$$

The assertion now follows from Corollary 3.1, because N_j and N_{j+p} coincide on the points with (3.11). □

3.3 Derivatives

We now want to derive the control net representation for the derivative of a spline function S defined by a control net $\{d_\ell\}_\ell$ and a knot sequence $\{t_\ell\}_\ell$. On any nondegenerate interval $[t_j, t_{j+1})$ we can use (2.4) to write

$$\begin{array}{rcl} M_{P_j'}(t_{\ell+1},\ldots,t_{\ell+n-1}) & = & \frac{n}{t_{\ell+n}-t_\ell}(M_{P_j}(t_\ell + (t_{\ell+n}-t_\ell), t_{\ell+1},\ldots,t_{\ell+n-1}) \\ & & -M_{P_j}(t_\ell, t_{\ell+1},\ldots,t_{\ell+n-1})) \\ & = & \frac{n}{t_{\ell+n}-t_\ell}(d_\ell - d_{\ell-1}) \end{array}$$

for all $\ell,\ j-n+1 \le \ell \le j$. Thus the control net $\{\frac{n}{t_{\ell+n}-t_\ell}(d_\ell - d_{\ell-1})\}_{\ell\in L_{n-1}}$ generates S'. As in the Bernstein–Bézier case, we see that the control net of a derivative consists of scaled differences of the original control points. This makes it easy to evaluate derivatives by application of the de Boor algorithm.

3.4 Knot Insertion

We now insert a new knot τ into a nontrivial subinterval $[t_k, t_{k+1})$. If we use $'$ to denote the knots in the new numbering, we have

$$\begin{array}{ccccc} \ldots \le t_k & \le & \tau & < & t_{k+1} \le \ldots \\ \ldots \le t_k' & \le & t_{k+1}' & < & t_{k+2}' \le \ldots \end{array}$$

The polynomials P_i on $[t_i, t_{i+1})$ will turn into polynomials Q_m on $[t_m', t_{m+1}')$, but it is clear that the Q_m are the P_i in some new numbering. Of course, the given control net $\{d_\ell\}_{\ell\in\mathbb{Z}}$ now needs to be modified and extended by an additional vector, and we denote the new control net by

$\{d'_\ell\}_{\ell\in\mathbb{Z}}$. We derive the formulae for construction of $\{d'_\ell\}_{\ell\in\mathbb{Z}}$ from $\{d_\ell\}_{\ell\in\mathbb{Z}}$ by application of the necessary equations

$$\begin{array}{rcl} d_\ell &=& M_{P_j}(t_{\ell+1},\ldots,t_{\ell+n}),\ j-n\le\ell\le j\\ d'_{\ell'} &=& M_{Q_{j'}}(t'_{\ell'+1},\ldots,t'_{\ell'+n}),\ j'-n\le\ell'\le j', \end{array} \tag{3.12}$$

where P_j lives on $[t_j,t_{j+1})$ and $Q_{j'}$ lives on $[t'_{j'},t'_{j'+1})$, respectively. Let us first look at knots up to $t_k=t'_k$, where the renumbering is trivial. We can take all $\ell=\ell'$ in (3.12) with $\ell+n=\ell'+n\le k$, and this allows for intervals $[t_j,t_{j+1})=[t'_j,t'_{j+1})=[t'_{j'},t'_{j'+1})$ for all $j=j'\le k-n$. This means that we have

$$d'_\ell=d_\ell \text{ for all } \ell\le k-n \text{ and } P_\ell=Q_\ell \text{ for all } \ell\le k-n.$$

Now we look at knots from $t_{k+1}=t'_{k+2}$ on, where the renumbering is by adding 1 to the index of an old knot. We can take all ℓ in (3.12) with $\ell+1\ge k+1$, and all ℓ' with $\ell'+1\ge k+2$. Thus we should set $\ell'=\ell+1$ and get

$$d'_{\ell'}:=d'_{\ell+1}:=d_\ell \text{ and } Q_{\ell+1}=P_\ell \text{ for all } \ell\ge k.$$

So far, everything works as expected: we can keep the "lower" part of the control net, and just have to shift the numbering of the "upper" part to allow for one additional control point. The n control points $d'_{k-n+1},\ldots,d'_k$ in the "middle" need some more work. Let us look more closely at

$$\begin{array}{rcl} d'_{\ell'} &=& M_{Q_{j'}}(t'_{\ell'+1},\ldots,t'_k,t'_{k+1},t'_{k+2},\ldots,t'_{\ell'+n})\\ &=& M_{Q_{j'}}(t_{\ell'+1},\ldots,t_k,\tau,t_{k+1},\ldots,t_{\ell'+n-1}) \end{array}$$

for $k-n+1\le\ell'\le k$ and observe that this range implies that $\tau=t'_{k+1}$ is always one of the arguments. Furthermore, we can take $j'=k$ and $j'=k+1$ in the above identities for the full range of ℓ', and thus we have $Q_{j'}=P_k$ for these j'. We compare this to

$$\begin{array}{rcl} d_{\ell'-1} &=& M_{P_k}(t_{\ell'},\ldots,t_{\ell'+n-1})\\ d_{\ell'} &=& M_{P_k}(t_{\ell'+1},\ldots,t_{\ell'+n}), \end{array}$$

where the admissible range of ℓ' again is $k-n+1\le\ell'\le k$. With the affine combination

$$\tau=\frac{t_{\ell'+n}-\tau}{t_{\ell'+n}-t_{\ell'}}t_{\ell'}+\frac{\tau-t_{\ell'}}{t_{\ell'+n}-t_{\ell'}}t_{\ell'+n}$$

we get the affine combination

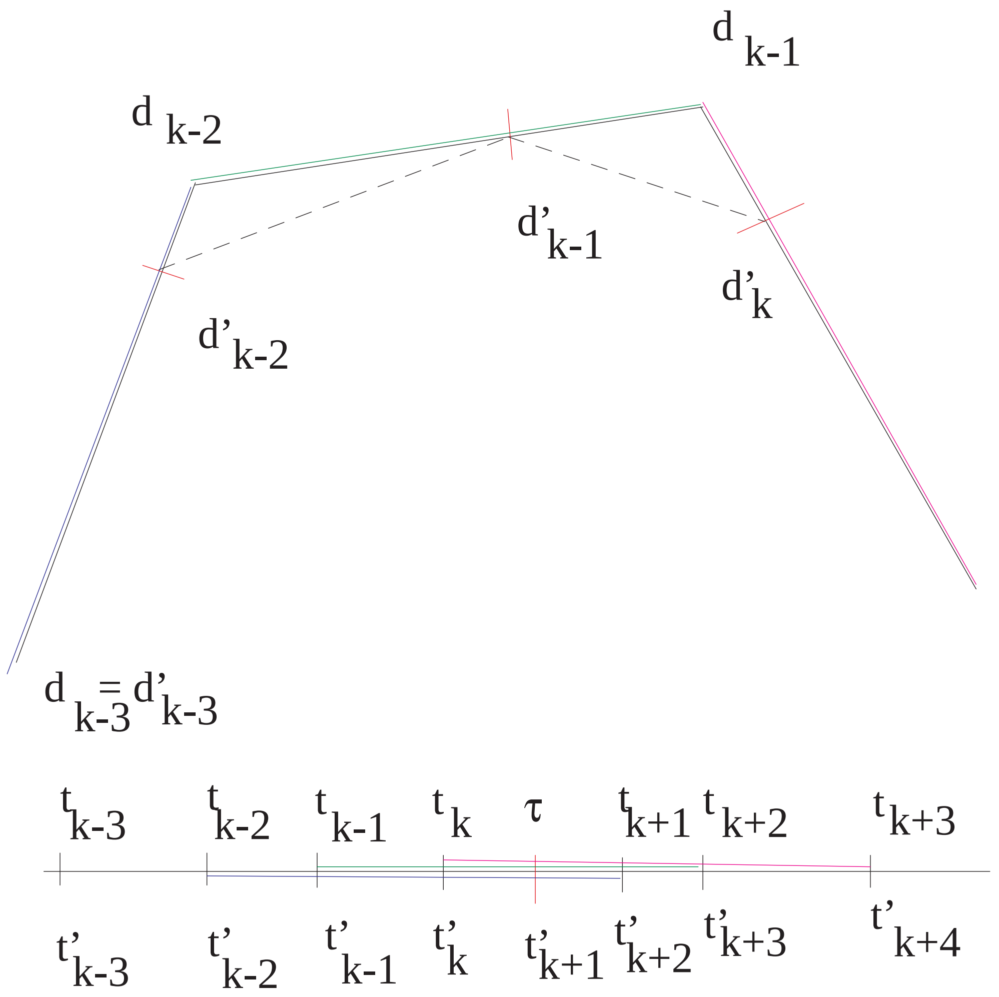
$$\begin{array}{rcl} \frac{t_{\ell'+n}-\tau}{t_{\ell'+n}-t_{\ell'}}d_{\ell'-1}+\frac{\tau-t_{\ell'}}{t_{\ell'+n}-t_{\ell'}}d_{\ell'} &=& M_{P_k}(\tau,t_{\ell'+1},\ldots,t_{\ell'+n-1})\\ &=& M_{Q_{j'}}(t_{\ell'+1},\ldots,t_k,\tau,t_{k+1},\ldots,t_{\ell'+n-1})\\ &=& d'_{\ell'}, \end{array} \tag{3.13}$$

and this is the recipe for knot insertion.

Theorem 3.3 *Let a control net $\{d_\ell\}_{\ell\in\mathbb{Z}}\subset\mathbb{R}^d$ and a knot sequence $\{t_\ell\}_{\ell\in\mathbb{Z}}\subset\mathbb{R}$ be given, and let S be the spline function of degree at most n defined via the algorithm of de Boor. If an additional knot $\tau\in[t_k,t_{k+1})$ is inserted, the new data*

$$\begin{array}{rcll} d'_\ell &:=& d_\ell, & \ell\le k-n\\ d'_\ell &:=& \frac{t_{\ell+n}-\tau}{t_{\ell+n}-t_\ell}d_{\ell-1}+\frac{\tau-t_\ell}{t_{\ell+n}-t_\ell}d_\ell & k-n+1\le\ell\le k\\ d'_\ell &:=& d_{\ell-1}, & k+1\le\ell\\ t'_\ell &:=& t_\ell, & \ell\le k\\ t'_\ell &:=& \tau, & k+1=\ell=k+1\\ t'_\ell &:=& t_{\ell-1}, & k+2\le\ell \end{array}$$

define the same spline S. □



Of course, the first sweep of the de Boor algorithm coincides with knot insertion. It is left to the reader that the full de Boor algorithm is the same as inserting a knot n times, provided that the new control points are stored and numbered appropriately.

3.5 Convergence of Knot Insertion

We now want to see whether the control net defined by insertion of sufficiently many knots must necessarily converge to the spline curve S it defines. Of course, we shall repeat the logic of the corresponding proof of the Bernstein–Bézier case and employ Taylor expansions of control points $d_\ell = M_{P_j}(t_{\ell+1},\ldots,t_{\ell+n})$ for $j-n \le \ell \le j$. The expansion point will be the **Greville abscissa**

$$\xi_\ell := (t_{\ell+1}+\ldots+t_{\ell+n})/n$$

satisfying

$$t_{\ell+1} \le \xi_\ell \le t_{\ell+n} \tag{3.14}$$

for all $\ell \in \mathbb{Z}$. For each ℓ we can find a j such that $\xi_\ell \in [t_j, t_{j+1})$, and this j must satisfy $j-n \le \ell \le j$ because of (3.14). In fact, $\xi_\ell \in [t_j,t_{j+1})$ implies $t_{\ell+n} \ge t_j$ and $t_{\ell+1} \le t_{j+1}$ because otherwise the Greville abscissa would not fall into the interval.

For such a pair (ℓ, j) we expand $d_\ell = M_{P_j}(t_{\ell+1},\ldots,t_{\ell+n})$ around $S(\xi_\ell) = P_j(\xi_\ell) = M_{P_j}(n\#\xi_\ell)$

and get, up to second–order terms in $t_{\ell+k} - \xi_\ell$, the error

$$d_\ell - S(\xi_\ell) \approx M'_{P_j}(n\#\xi_\ell)\sum_{k=1}^{n}(t_{\ell+k} - \xi_\ell) = 0.$$

Again, we used that the derivatives of a polar form with respect to each of the arguments must be the same, and the reader will see why the trick requires the Greville abscissae. We can add that the bound on the second–order terms depends on the maximum absolute value of the second derivative of M_{P_j} at arguments somewhere between t_ℓ and $t_{\ell+n}$, multiplied with a factor bounded by $n^2(t_{\ell+n} - t_\ell)^2$.

For a convergence argument, we start with a fixed set of polynomial pieces and do the analysis just for one piece and its two neighbors. Inserting lots of knots, with the quantity

$$h := \max_\ell (t_{\ell+n} - t_\ell) \tag{3.15}$$

tending to zero, we have just three different polynomials involved if h is small enough, and can thus bound the local error by a constant times h^2, where the constant depends on the second derivatives of the polar forms associated with the three local polynomial pieces.

If the spline curve is twice continuously differentiable, we can use standard arguments of linear interpolation to conclude that the control net converges to the spline at a local rate that is proportional to h^2. If we have just continuity of the spline, we have uniform continuity locally because the spline consists of polynomials, and then we still get local convergence of the control net to the spline, but no convergence rate.

So far, we looked at local convergence, disregarding "endpoints" of the control net. But in the standard situation (3.8) the constants are uniform, because we have just a finite number of polynomial pieces. Furthermore, the outermost Greville abscissae coincide with the endpoints of the finite spline curve piece we look at, and the convergence argument will work uniformly, because it works for all parts of the curve that correspond to arguments between Greville abscissae. This can be formulated more exactly, but we omit a proof:

Theorem 3.4 *For any finite and twice differentiable spline curve of degree at most n with* (3.8)*, the piecewise linear interpolant of the control net converges uniformly to the spline curve, and the error can be bounded by a constant times h^2, where h is defined in* (3.15)*. The error is measured by parametrizing the spline curve linearly between the Greville abscissae associated to the points of the control net.*

The reason for not giving the proof is as follows: it is common practice to evaluate a spline by inserting each knot sufficiently many times to make in n–fold. The one can use subdivision on the Bernstein–Bézier control net. And we have already proven quadratic convergence of this process.

3.6 B–Splines

Though we did not need to define and use them, B–splines always were behind the scene. They come out to form the partition of unity building the representation of a spline curve S of degree at most n defined by a control net $\{d_\ell\}_\ell$ and a knot sequence $\{t_\ell\}_\ell$. In particular, we want to write

$$S(t) = \sum_{\ell \in L_n} d_\ell N_\ell(t) \tag{3.16}$$

with certain piecewise polynomials $N_\ell(t)$ called B–splines that are a partition of unity in such a way that only a finite number of them is nonzero for each fixed t. Here, we only use the relevant control vectors with indices in the set L_n from (3.5). The others are irrelevant and should be dropped right from the start.

For any nontrivial interval $[t_j, t_{j+1})$ and the corresponding part $d_{j-n}, \dots, d_j$ of the control net, the algorithm of de Boor constructs a polynomial

$$P_j(t) = \sum_{\ell=j-n}^{j} p_{j,\ell}(t) d_\ell$$

with certain nonnegative scalar–valued polynomials $p_{j,\ell}(t)$ that depend only on the knot vector, but not on the control net. Thus we can write

$$\begin{aligned} S(t) &= \sum_j \chi_{[t_j,t_{j+1})}(t) S(t) \\ &= \sum_j \chi_{[t_j,t_{j+1})}(t) P_j(t) \\ &= \sum_j \chi_{[t_j,t_{j+1})}(t) \sum_{\ell=j-n}^{j} p_{j,\ell}(t) d_\ell \\ &= \sum_{\ell \in L_n} d_\ell \sum_{j=\ell}^{\ell+n} \chi_{[t_j,t_{j+1})}(t) p_{j,\ell}(t) \end{aligned}$$

with the characteristic function

$$\chi_{[t_j,t_{j+1})}(t) = 1 \text{ for all } t \in [t_j, t_{j+1}), \text{ zero elsewhere.}$$

Thus the B**–splines**

$$N_{\ell,n}(t) := \sum_{j=\ell}^{\ell+n} \chi_{[t_j,t_{j+1})}(t) p_{j,\ell}(t), \ \ell \in L_n \tag{3.17}$$

are what we asked for. They yield (3.16) for all spline functions defined by control nets associated with a fixed knot vector.

Theorem 3.5 *The B–splines corresponding to a knot sequence $\{t_\ell\}_\ell$ have the following properties:*

1. *$N_{\ell,n}$ is defined for the ℓ with $t_\ell < t_{\ell+n+1}$.*
2. *$N_{\ell,n}$ is consisting of polynomial pieces of degree at most n, with breakpoints at the knots of $[t_\ell, t_{\ell+n+1}]$ and vanishing outside this interval.*
3. *The spline curve defined by the algorithm of de Boor can be represented as* (3.16) *with* (3.17)*.*

4. *$N_{\ell,n}$ can be calculated via the algorithm of de Boor, started with the scalar–valued control net $\{\delta_{k,\ell}\}_k$. Therefore the B–splines are uniquely defined by a knot sequence and the de Boor algorithm on this special control net.*

5. *$N_{\ell,n}$ is nonnegative on $[t_\ell, t_{\ell+n+1})$ and zero elsewhere.*

6. *If a knot of $[t_\ell, t_{\ell+n+1}]$ has multiplicity p, the function $N_{\ell,n}$ has continuous derivatives up to order $n-p$ there.*

7. *The set of all B–splines $N_{\ell,n}$ for $\ell \in L_n$ and a fixed knot sequence forms a partition of unity.*

8. *If a spline curve of degree at most n vanishes outside some interval $[t_k, t_{k+n}]$, it vanishes everywhere. In particular, the B–splines $N_{\ell,n}$ for $\ell \in L_n$ are linearly independent.*

Proof: The first property follows from (3.6), while the second is contained in (3.17). Our discussion preceding the theorem proves the third statement, and the fourth is just a special scalar case. Now the fifth follows from the fourth due to the convex combinations performed by the de Boor algorithm. Theorem 3.2 yield the sixth property. The first property that needs a real proof is number 7. For the scalar–valued control net $\{1\}_k$, the de Boor algorithm takes affine combinations of 1 all the time, and it just ends up with

$$S(t) = 1 = \sum_{\ell \in L_n} 1 \cdot N_{\ell,n}(t).$$

For the proof of 8, take a vanishing linear combination of the form (3.16) outside of some interval $[t_k, t_{k+n}]$ and consider the polynomials $P_j = 0$ on nondegenerate intervals $[t_j, t_{j+1})$ for $j+1 \leq k$ and $j \geq k+n$, respectively. The multiaffine forms $d_\ell = M_{P_j}(t_\ell, \ldots, t_{\ell+n})$ must vanish for the ranges $j-n \leq \ell \leq j$ for these j. But this means that $d_\ell = 0$ for $\ell \leq j \leq k-1$ and $\ell \geq j-n \geq k$, respectively, and since all d_ℓ vanish, the complete spline is zero. If we take a scalar control net generating a vanishing spline function, we can use our previous proof to conclude that the control net must be zero, proving linear independence of the B–splines. Note that these proof steps do not account for terms with indices $\ell \notin L_n$. The corresponding control points d_ℓ are redundant in de Boor's algorithm, and they spoil any uniqueness proof. □

3.7 Recursion of B–Splines

If we look at the formula (3.4) for the de Boor algorithm, we can see that it has a recursive structure. In fact, if the transition from the control points $d_{j-n}, \ldots, d_j$ for some $t \in [t_j, t_{j+1})$ to $d^{n-1}_{j-n+1}(t), \ldots, d^{n-1}_j(t)$ is defined as a map $D_{j,n,t} : I\!R^{d(n+1)} \to I\!R^{dn}$, we can write (3.4) as an action of $D_{j,n-r,t}$ on $d^{n-r}_{j-n+r}(t), \ldots, d^{n-r}_j(t)$.

But this implies for $t \in [t_j, t_{j+1})$ the identity

$$\begin{aligned} S(t) &= D_{j,1,t} \circ \ldots \circ D_{j,n-1,t} \circ D_{j,n,t}(d_{j-n}, \ldots, d_j) \\ &= D_{j,1,t} \circ \ldots \circ D_{j,n-1,t}(d^{n-1}_{j-n+1}(t), \ldots, d^{n-1}_j(t)). \end{aligned}$$

Since the second line means that the de Boor algorithm for degree $n-1$, when applied to the control points $d^{n-1}_{j-n+1}(t), \ldots, d^{n-1}_j(t)$ gives the same result as the standard form, we can write the second line in terms of the B–splines $N_{\ell,n-1}$ of degree at most $n-1$, and recursively all the way down, i.e.

$$\begin{array}{rcl} S(t) & = & \sum\limits_{\ell=j-n}^{j} d_\ell N_{\ell,n}(t) \\ & = & \sum\limits_{\ell=j-n+1}^{j} d_\ell^{n-1}(t) N_{\ell,n-1}(t) \\ & = & \sum\limits_{\ell=j-n+r}^{j} d_\ell^{n-r}(t) N_{\ell,n-r}(t) \\ & = & d_j^0(t) N_{j,0}(t) \\ & = & d_j^0(t). \end{array}$$

This is another close connection to the de Casteljau algorithm, but there still is something left. If we write the second formula down in explicit form, we get

$$\begin{array}{rcl} S(t) & = & \sum\limits_{\ell=j-n+1}^{j} d_\ell^{n-1}(t) N_{\ell,n-1}(t) \\ & = & \sum\limits_{\ell=j-n+1}^{j} \left(\frac{t_{\ell+n}-t}{t_{\ell+n}-t_\ell} d_{\ell-1}^{n}(t) + \frac{t-t_\ell}{t_{\ell+n}-t_\ell} d_\ell \right) N_{\ell,n-1}(t) \\ & = & \sum\limits_{\ell=j-n}^{j} \left(\frac{t_{\ell+1+n}-t}{t_{\ell+1+n}-t_\ell} N_{\ell+1,n-1}(t) + \frac{t-t_\ell}{t_{\ell+n}-t_\ell} N_{\ell,n-1}(t) \right) d_\ell, \end{array}$$

and since the B–splines are uniquely determined by the control net representation, we get the recursion formula

$$N_{\ell,n}(t) = \frac{t_{\ell+1+n}-t}{t_{\ell+1+n}-t_\ell} N_{\ell+1,n-1}(t) + \frac{t-t_\ell}{t_{\ell+n}-t_\ell} N_{\ell,n-1}(t)$$

for $t \in [t_j, t_{j+1})$ and all ℓ, $j-n \leq \ell \leq j$. This recursion needs some explanation, because for $\ell \in L_n$ the indices ℓ and $\ell+1$ need not be both in L_{n-1}. But if, by additional definition, $N_{\ell,n-1}(t)$ vanishes outside of $[t_\ell, t_{\ell+n})$ for **all** ℓ, there is no problem with the recursion.

3.8 Special Cases

For $n=0$, we have $d_j = M_{P_j} = P_j(t)$ for all $t \in [t_j, t_{j+1})$, and the B–spline $N_{j,0}$ is the characteristic function on $[t_j, t_{j+1})$. Here, the set L_0 is the set of j with $t_j < t_{j+1}$.

For $n=1$, the control points for evaluation in $t \in [t_j, t_{j+1})$ are $d_j = M_{P_j}(t_{j+1}), d_{j-1} = M_{P_j}(t_j)$. These are precisely the degree 1 Bernstein–Bézier control points on the interval $[t_j, t_{j+1}]$, and thus we have the linear interpolant

$$P_j(t) = \frac{t_{j+1}-t}{t_{j+1}-t_j} d_{j-1} + \frac{t-t_j}{t_{j+1}-t_j} d_j$$

between d_{j-1} and d_j. The spline is a continuous polygonal line, if all knots have multiplicity one. The set L_1 is the set of all ℓ where $t_\ell < t_{\ell+2}$, and the B–spline N_ℓ is the piecewise linear interpolant of the data 0,1,0 on the points $t_\ell, t_{\ell+1}, t_{\ell+2}$ on nontrivial half–open subintervals. Clearly, these functions must sum up to one on each nontrivial interval.

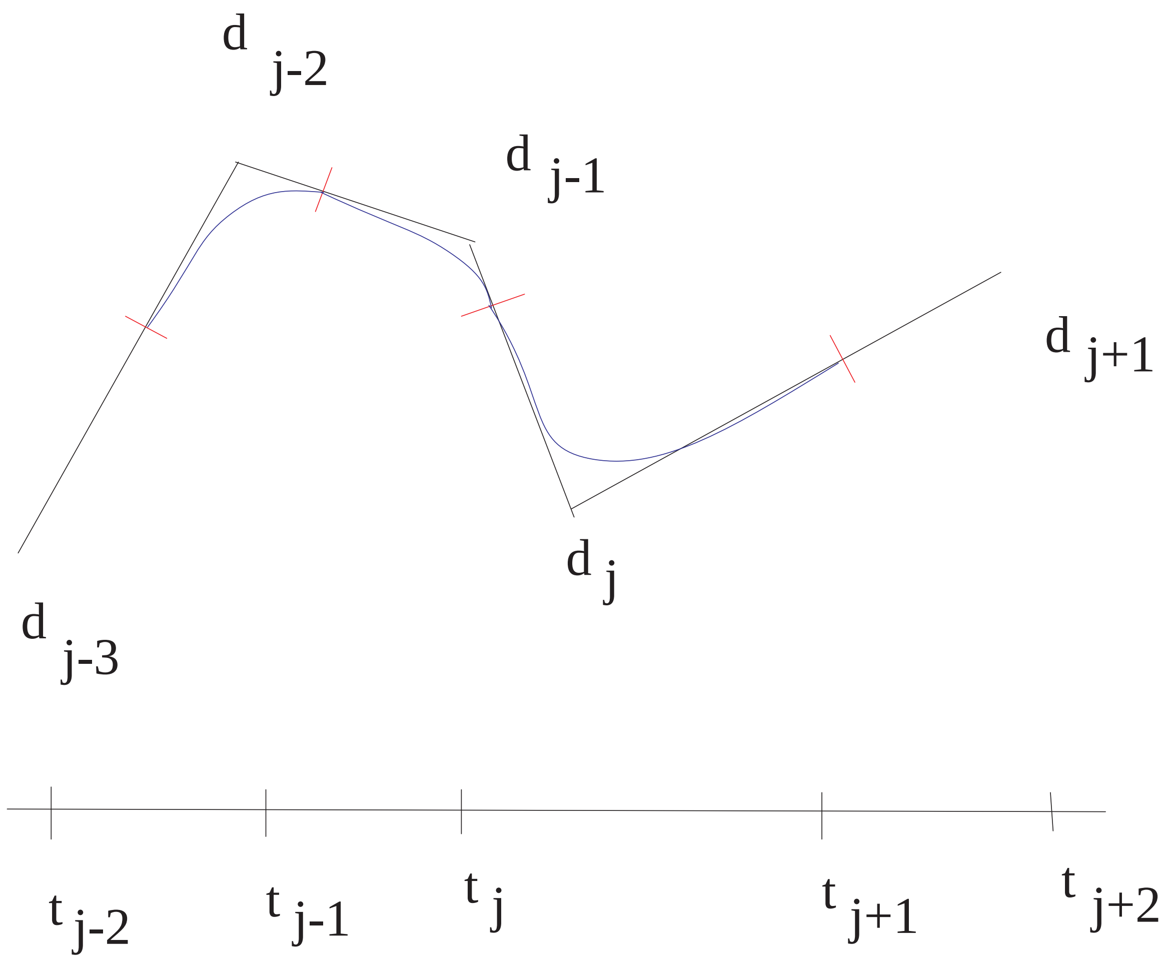
For $n = 2$, the control points for evaluation in $t \in [t_j, t_{j+1})$ are

$$\begin{array}{rcl} d_j & = & M_{P_j}(t_{j+1}, t_{j+2}) \\ d_{j-1} & = & M_{P_j}(t_j, t_{j+1}) \\ d_{j-2} & = & M_{P_j}(t_{j-1}, t_j). \end{array}$$

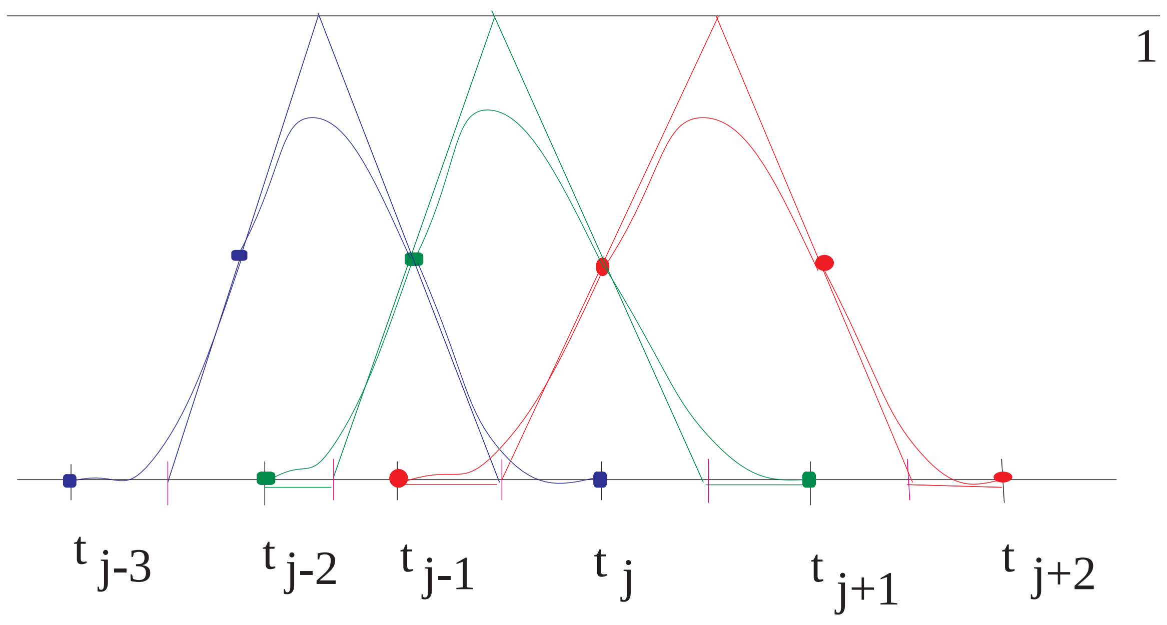
These are not Bernstein–Bézier control points for the interval $[t_j, t_{j+1}]$, but we can form these by the de Boor or knot insertion algorithm. In fact,

$$\begin{array}{rcl} M_{P_j}(t_j, t_j) & = & \dfrac{t_{j+1} - t_j}{t_{j+1} - t_{j-1}} d_{j-2} + \dfrac{t_j - t_{j-1}}{t_{j+1} - t_{j-1}} d_{j-1} \\ M_{P_j}(t_{j+1}, t_{j+1}) & = & \dfrac{t_{j+2} - t_{j+1}}{t_{j+2} - t_j} d_{j-1} + \dfrac{t_{j+1} - t_j}{t_{j+2} - t_j} d_j \end{array}$$

do the job together with $d_{j-1} = M_{P_j}(t_j, t_{j+1})$. The spline is a differentiable sequence of parabolic arcs, if all knots have multiplicity one, and the curve then touches each line segment $d_{j-1}d_j$ at the intermediate point defined by the ratio of t_{j+1} between t_j and t_{j+2}. These touchpoints are the images of the knots under the curve mapping.



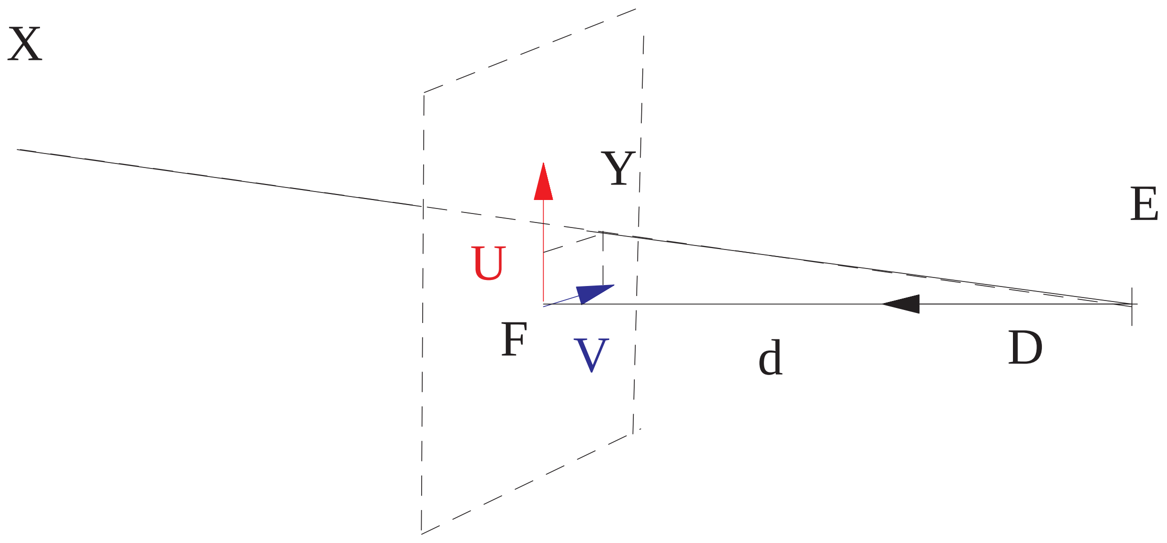
This construction also works for the B–spline $N_{j-1,2}$. The control net should be placed at the Greville abscissae, i.e. value 1 at $(t_j + t_{j+1})/2$ and zero at all other $(t_k + t_{k+1})/2$. The polygonal lines between these data are then marked at the abscissae t_k, and these points define the outer Bernstein–Bezier control points for the pieces of $N_{j-1,2}$.



4 Some Basics of Computer Graphics

4.1 The View Transformation

Here is a little bit of Mathematics behind Computer Graphics. Assume that we want to map a point $X \in I\!R^3$ to some plane that an observer sees if he is placed at an "eyepoint" E and looking into the direction D. The view direction vector D should have length one, and we assume another unit vector U to be given such that U and $V := U \times D$ span the viewing plane. The "view–up–vector" U should be perpendicular to D, and thus we have a full orthogonal coordinate system formed by D, U, and V. The origin F of the viewing plane is assumed to be at a distance d on D from E.



The point Y should be in the viewing plane and on the ray from E to X, when the observer looks from E at X. We thus write $Y = \lambda E + (1-\lambda)X$, and the local coordinates of the projected image Y of X on the viewing plane are

$$\begin{array}{rcllll} u & := & U^T(Y-F) & = & U^T(\lambda E+(1-\lambda)X-F) & = & U^T(X-F+\lambda(E-X)) \\ v & := & V^T(Y-F) & = & V^T(\lambda E+(1-\lambda)X-F) & = & V^T(X-F+\lambda(E-X)), \end{array}$$

while the "depth" coordinate of X with respect to the eyepoint E is $D^T(X-E)$. Since Y should be in the viewing plane, we know that

$$\begin{array}{rcl} D^T(Y-F) & = & 0 \\ & = & D^T(\lambda E+(1-\lambda)X-F) = D^T(X-F+\lambda(E-X)). \end{array}$$

We want to get rid of E in favour of F and allow parallel projection later, i.e. $d=\infty$. Thus we use $F=E+dD$ and get

$$0 \;=\; D^T(X-F+\lambda(E-X)) \;=\; D^T(X-F)+\lambda D^T(F-dD-X)$$

and λ comes out to be

$$\lambda=\frac{D^T(X-F)}{D^T(X-F)+d}\quad 1-\lambda=\frac{d}{D^T(X-F)+d}=\frac{1}{1+D^T(X-F)/d}.$$

We can introduce $\sigma := 1/d$ and write

$$1-\lambda=\frac{1}{1+\sigma D^T(X-F)}$$

in a form that makes sense for $d=\infty$ or $\sigma=0$. We now consider the depth coordinate

$$D^T(X-E)=D^T(X-F+dD)=D^T(X-F)+d=d(1+\sigma D^T(X-F)),$$

and for the application to z–buffering we simply rescale it by division by d to get

$$z:=D^T(X-E)/d=1+\sigma D^T(X-F),\ 1-\lambda=1/z.$$

Any monotone transformation would be feasible, because we just want to keep points on the rays from X to E ordered properly by the depth information. Then we go back to the cordinates

$$\begin{array}{rcllll} u & = & U^T(Y-F) & = & U^T(X-F+\lambda(F-X-dD)) & = & (1-\lambda)U^T(X-F) \\ v & = & V^T(Y-F) & = & V^T(X-F+\lambda(F-X-dD)) & = & (1-\lambda)V^T(X-F) \end{array}$$

for the final form

$$\begin{array}{rcl} z & = & 1+\sigma D^T(X-F) \\ u & = & U^T(X-F)/z \\ v & = & V^T(X-F)/z. \end{array}$$

Note the simplification for parallel projection, i.e. for $\sigma=0,\ z=1$.

4.2 Z–Buffering

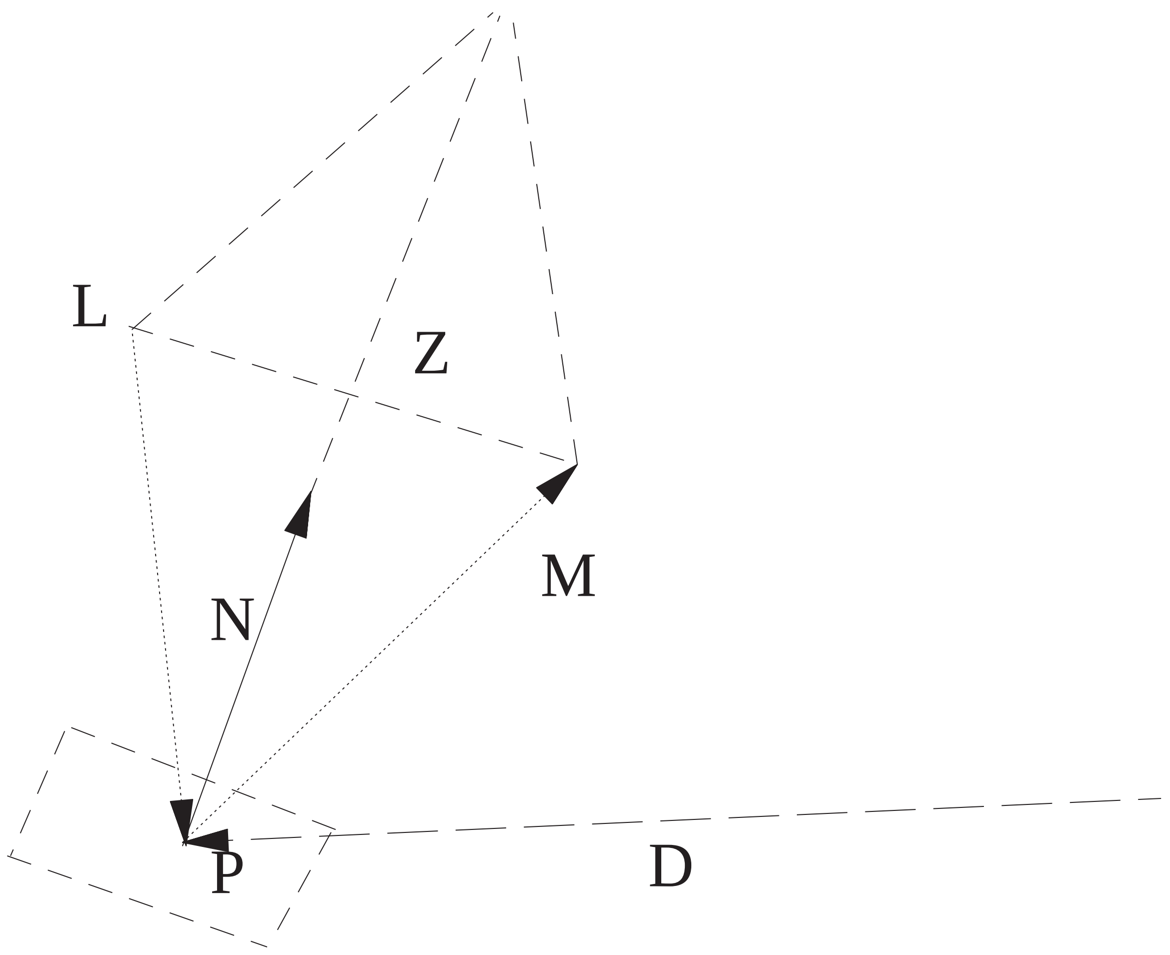
The standard way to render a scene with proper treatment of hidden parts is to calculate the (u,v,z) coordinates for each point of the scene, but to display for each (u,v) only the one with least value of z, because the others are hidden by it due to their larger distance from the eyepoint. This is done by discretizing the (u,v)–plane into pixels $P(u,v):=(i_u,i_v)\in \mathbb{Z}^2$ and to keep two arrays dimensioned by pixels, one, call it C, for the colour information and another, call it Z, for the depth information. The colour array is initialized by some default value, e.g. a background colour. Then one loops over all points (u,v,z) in the scene and performs the following:

- If the colour information for pixel $P(u, v)$ is not the background colour, and if z is larger than the value stored in $Z(P(u, v))$, then ignore the point, because it lies behind a point that was already generated. Otherwise calculate a colour c for the point (u, v, z) and update $C(P(u, v)) = c$ and $Z(P(u, v)) = z$.

Of course, this process is rather time–consuming, and therefore modern graphics hardware keeps an internal (discretized) Z–Buffer along with the array C with entries consisting of coded colour values. Furthermore, the calculations of the scene rendering are broken down by subdivision of surfaces, until here is a huge mass of small planar polygons, usually triangles or rectangles, that are later handed over to the hardware, which may be called a “polygon pipeline”. Note that planar polygons in world coordinates are mapped to polygons in the view plane **for every view**, may it be a parallel or a central projection (lines are always mapped to lines, planes are always mapped to planes...). The image of the view consists of planar polygons in the view plane, and these are finally displayed. The subdivision can often be made independent of the actual view on the scene, and then each new view just involves processing all the tiny polygons over again, assigning new colours and a new Z–Buffer for rendering. This is why the speed of modern graphics equipment is measured in polygons per second. Measuring frames per second, the latter being roughly equivalent to view calculations per second, depends on the complexity (or number of polygons) of the scene, while the polygon rate does not. We supply a simple self–made rendering program, implementing a Z–Buffer, for the exercises.

4.3 Colour, Shading, and Highlights

The standard way of assigning a colour to a point P of a surface $F(u, v)$ needs a normal vector N on that point, i.e. a unit vector that is perpendicular to the tangent plane, for instance $N = \frac{\partial F}{\partial u} \times \frac{\partial F}{\partial v}$ plus normalization. Note that the normal gives a local orientation of the surface, i.e. one can talk about the “front” or “back” side and assign different colours to these. The sign of the inner product of the normal N with the view direction D decides which side is seen, and the absolute value of the cosine of the angle between D and N can be used to define a “shading” towards black for the angle moving towards 90^0.



Another cheap way of improving the appearance of a rendered object is to add a highlight coming from a light source al L. From the ray $L - P$ of light falling at the point P, and using the standard reflection law, one gets a direction M for the outgoing reflected ray, i.e. the direction of the line between $L + 2(Z - L)$ and P, where $Z = P + N^T(L - P)$. Then the cosine of the angle between M and D describes how much reflection reaches the eyepoint, and by Phong's rule the colour of the surface point is changed towards white (or the colour of the light source) by an amount governed by a power of this cosine. For large powers, there will be a sharp and centered highlight, while small powers give diffuse and large highlights.

The polygon rendering technique, as described above, is the standard way of displaying 3D scenes quickly. More sophisticated techniques are

- ray tracing and
- radiosity methods,

but this lecture focuses on Mathematics, not on Computer Science.

5 Tensor Product Surfaces

5.1 General Facts

Surfaces are classically defined either

- implicitly by equations satisfied by the points on the surface (e.g. $(x-x_c)^2+(y-y_c)^2+(z-z_c)^2=r^2$ for all points $(x,y,z)\in I\!R^3$ on the surface of a ball in $I\!R^3$ with center (x_c,y_c,z_c) and radius r) or
- explicitly as the image of a two–dimensional domain $\Omega\subset I\!R^2$ under a map $F\;:\;\Omega\to I\!R^d,\ d>2$.

We concentrate on the second case with range in $I\!R^3$, though many facts are valid in arbitrary dimensions.

An important notion for surfaces with differentiable explicit parametric representations $F\;:\;\Omega\to I\!R^3$ is the **tangent space** spanned by $\frac{\partial F}{\partial u},\frac{\partial F}{\partial v}$ at a point $F(u,v),\ (u,v)\in\Omega\subset I\!R^2$. A vector perpendicular to the tangent space at $F(u,v)$ is called a **surface normal** at $F(u,v)$. The point $F(u,v)$ is called **nonsingular**, if the dimension of the tangent space is 2, i.e. if the above tangent vectors are linearly independent. In this case, the vector $\frac{\partial F}{\partial u}\times\frac{\partial F}{\partial v}$ is a surface normal, and in $I\!R^3$ the surface normals span a one–dimensional space. Assigning a surface normal to each point $F(u,v)$ gives a local orientation to the surface, and the practically interesting surfaces are those that have no singularities and allow assignment of a continuously varying surface normal, e.g. via the choice of $\frac{\partial F}{\partial u}\times\frac{\partial F}{\partial v}$ everywhere.

5.2 Tensor Products

A standard way to generate multivariate functions from univariate functions is by taking sums of products. In particular, if we start with two univariate partitions of unity

$$\begin{array}{lll} u_i & [\alpha,\beta]\to I\!R^3 & i\in I\\ v_j & [\gamma,\delta]\to I\!R^3 & j\in J\end{array}$$

we can define a bivariate partition of unity via

$$w_{ij}(u,v):=u_i(u)v_j(v),\ [\alpha,\beta]\times[\gamma,\delta]\to I\!R^3,\ (i,j)\in I\times J.$$

A **tensor product** surface on $[\alpha,\beta]\times[\gamma,\delta]$ will then be generated for any control net $\{b_{ij}\}_{(i,j)\in I\times J}$ via

$$F(u,v):=\sum_{i\in I}\sum_{j\in J}b_{ij}u_i(u)v_j(v),\ (u,v)\in[\alpha,\beta]\times[\gamma,\delta].$$

Note that we can use either Bernstein polynomials or B–splines for each of the univariate partitions of unity, and we thus can construct surfaces defined on rectangular domains. The sum can be reordered in two ways:

$$\begin{array}{rcl} F(u,v) & := & \sum_{i\in I}c_i(v)u_i(u)\\ c_i(v) & := & \sum_{j\in J}b_{ij}v_j(v),\ i\in I\\ F(u,v) & := & \sum_{j\in J}d_j(u)v_j(v)\\ d_j(u) & := & \sum_{i\in I}b_{ij}u_i(u),\ j\in J\end{array}$$

with two finite sets of curves c_i and d_j. These formulae can be used to boil the computation down to a finite number of univariate computations. In fact, to get $F(u,v)$ we can first fix v

and calculate the $c_i(v)$ for $i \in I$ by the second formula. Then we put these vectors into the first formula of the above display, using the $c_i(v)$ as a control net for the curve defined by the functions $u_i(u)$. But we can do better, as we shall see below, using the de Casteljau/de Boor algorithm and subdivision/knot insertion.

Keeping u or v fixed, the function $F(u, v)$ describes an **isoparametric curve** on the surface, and the above equations describe the univariate control nets for these curves.

5.3 Bilinear Patches

If we have two linear Bernstein–Bézier representations on $[0, 1]$, the resulting tensor product surface on $[0, 1] \times [0, 1]$ is **bilinear** and can be written as

$$F(u, v) = b_{00}(1-u)(1-v) + b_{01}(1-u)v + b_{10}u(1-v) + b_{11}uv$$

with four control points. Note that each isoparametric curve is linear, but the surface is not in general planar. It is a hyperboloid, as some further analysis reveals, and the hyperboloid is a member of the family of **ruled surfaces** that can be generated by a motion of a line through space.

Note that the bilinear surface interpolates the four control points, and patching many of these surface elements together will yield a continuous piecewise bilinear surface. This feature is used by many primitive display routines for general (nonparametric) bivariate functions. The function f is evaluated on a fine grid, the four values on each corner of a grid cell are interpolated by a local bilinear function, and then the resulting piecewise bilinear function s is used as an approximation to the original function. If this is done for a real–valued function $f(u, v)$, the standard way of plotting contour curves

$$C_t(f) := \{(u, v) \ : \ f(u, v) = t\}$$

is to plot the curves $C_t(s)$ instead, and this can be done locally in each grid cell, solving equations of the form

$$b_{00}(1-u)(1-v) + b_{01}(1-u)v + b_{10}u(1-v) + b_{11}uv = t$$

for four given scalar values b_{ij} by a curve $v(u)$ or $u(v)$. This leads to a quadratic equation, which may be solvable or not, depending whether the contour set hits the cell or not. A further simplification splits the quadrangle spanned by the four control points of a bilinear patch into two triangles by a diagonal line. Then the triangle vertices are interpolated by a linear function in each of the triangles. The resulting continuous piecewise planar and piecewise linear function is even simpler to handle, and pieces of contour lines will always be line segments. Note that the general solution of the equation $f(u, v) = t$ can be a mess, especially if there are many unconnected branches to pick up. Programs calculating contours of functions can sometimes have a very strange output, but if you understand the above logic behind such programs, you will stop wondering.

5.4 Multiaffine Forms

We now concentrate on the case of a polynomial tensor product of degree (m, n), i.e. we consider polynomials

$$P(u,v) = \sum_{i=0}^{m}\sum_{j=0}^{n} \alpha_{ij} u^i v^j \tag{5.1}$$

with vector coefficients $\alpha_{ij} \in I\!R^d$.

Theorem 5.1 *For any polynomial of the above form there is a unique multiaffine form*

$$M_P = M_P(\cdot;\cdot) \ : I\!R^m \times I\!R^n \to I\!R^d$$

such that $P(u,v) = M_P(m\#u; n\#v)$ holds for all u, v, and such that M_P is invariant under permutations of the first m and the last n arguments.

Proof: For existence, we just superimpose univariate multiaffine forms

$$M_P(u_1,\dots,u_m; v_1,\dots,v_n) := \sum_{i=0}^{m}\sum_{j=0}^{n} \alpha_{ij} M_{u^i}(u_1,\dots,u_m) M_{v^j}(v_1,\dots,v_n).$$

For uniqueness, we assume that besides the above M_P there is another M of the required form such that $P(u,v) = M_P(m\#u; n\#v) = M(m\#u; n\#v)$ holds for all u, v. Keeping u fixed, we get from the univariate case that the symmetric multiaffine forms $M_P(m\#u; v_1,\dots,v_n)$ and $M(m\#u; v_1,\dots,v_n)$ must coincide for all v_j and all u. Keeping $v_1,\dots,v_n$ fixed, we get that $M_P(u_1,\dots,u_m; v_1,\dots,v_n)$ and $M(u_1,\dots,u_m; v_1,\dots,v_n)$ must coincide for all u_i. □

We now want to relate the Bernstein–Bézier representation of a tensor product polynomial surface to its multiaffine form.

Theorem 5.2 *For any polynomial P of the form* (5.1)*, the control net of a Bernstein–Bézier representation*

$$P(u,v) = \sum_{i=0}^{m}\sum_{j=0}^{n} b_{ij} \beta^{(m)}_{i,[\alpha,\beta]}(u) \beta^{(n)}_{j,[\gamma,\delta]}(v)$$

over $[\alpha,\beta] \times [\gamma,\delta]$ can be uniquely obtained as

$$b_{ij} = M_P(m-i\#\alpha, i\#\beta; n-j\#\gamma, j\#\delta),\ 0 \le i \le m,\ 0 \le j \le n. \tag{5.2}$$

Proof: Let us keep v fixed for a moment, and consider the polynomial $Q_v(u) := P(u,v)$ with the multiaffine form $M_{Q_v}(u_1,\dots,u_m) = M_P(u_1,\dots,u_m; n\#v)$. If we form a Bernstein–Bézier representation of it on $[\alpha,\beta]$, we get

$$P(u,v) = Q_v(u) = \sum_{i=0}^{m} M_P(m-i\#\alpha, i\#\beta; n\#v) \beta^{(m)}_{i,[\alpha,\beta]}(u). \tag{5.3}$$

We now represent the polynomials $M_P(m-i\#\alpha, i\#\beta; n\#v)$ in Bernstein–Bézier style with respect to v over $[\gamma,\delta]$ and get

$$M_P(m-i\#\alpha, i\#\beta; n\#v) = \sum_{j=0}^{n} M_P(m-i\#\alpha, i\#\beta; n-j\#\gamma, j\#\delta) \beta^{(n)}_{j,[\gamma,\delta]}(v).$$

Plugging this into (5.3) gives the required representation, and the uniqueness follows from the linear independence of the Bernstein polynomials in each variable. □

5.5 Variations of de Casteljau and Subdivision

The process of the preceding proof can be used to define three different de Casteljau algorithms for evaluation. Let us go backwards and evaluate the $m+1$ polynomials $M_P(m-i\#\alpha, i\#\beta; n\#v)$ for $0 \leq i \leq m$ as functions of v at some $t \in [\gamma, \delta]$ by application of the univariate de Casteljau algorithm. We would fix i and start with $b_{ij}^n(t) := b_{ij},\ 0 \leq j \leq n$ to proceed via

$$b_{ij}^{n-r}(t) := \frac{\delta - t}{\delta - \gamma} b_{ij}^{n-r+1}(t) + \frac{t-\gamma}{\delta-\gamma} b_{i,j+1}^{n-r+1}(t),\ 0 \leq j \leq n-r,\ 1 \leq r \leq n.$$

By this affine combination, we have

$$b_{ij}^{n-r}(t) = M_P(m-i\#\alpha, i\#\beta; n-j-r\#\gamma, j\#\delta, r\#t)$$

and the procedure ends with $b_{i0}^0(t) = M_P(m-i\#\alpha, i\#\beta; n\#t)$. We now use these values for $0 \leq i \leq m$ and perform the univariate de Casteljau algorithm with respect to the variable u at some $s \in [\alpha, \beta]$ to arrive at $M_P(m\#s; n\#t) = P(s,t)$.

Let us check the computational cost of this. We first have $m+1$ instances of the deCasteljau algorithm for degree n in $I\!R^d$, needing $\mathcal{O}(dmn^2)$ operations. Then we have a single de Casteljau process for degree m, taking $\mathcal{O}(dm^2)$ operations. If $n > m$ holds, we should have worked the other way round, and this is a second way of writing a de Casteljau algrithm.

If we ignore the final single de Casteljau process, we already have the data needed for subdivision into polynomials over $[\alpha, \beta] \times [\gamma, t]$ and $[\alpha, \beta] \times [t, \delta]$. In fact, we have the control nets

$$\begin{aligned} b_{i0}^{n-r}(t) &= M_P(m-i\#\alpha, i\#\beta; n-r\#\gamma, r\#t) \\ b_{i,n-r}^{n-r}(t) &= M_P(m-i\#\alpha, i\#\beta; r\#t, n-r\#\delta) \end{aligned}$$

for $0 \leq r \leq n,\ 0 \leq i \leq m$ and this is all we need.

But there is a third variation that works by taking a **bilinear** formula, performing subdivision at a point $(s,t) \in [\alpha, \beta] \times [\gamma, \delta]$. For this, we assume $m = n$, possibly after degree elevation. The starting point now is the full array

$$b_{ij}^n(s,t) := b_{ij} = M_P(n-i\#\alpha, i\#\beta; n-j\#\gamma, j\#\delta),\ 0 \leq i,j \leq n.$$

We do two affine steps to introduce one instance of s and t into the multiaffine form, and this gives the bilinear formula

$$\begin{aligned} b_{ij}^{n-r}(s,t) &:= \frac{\delta-t}{\delta-\gamma}\left(\frac{\beta-s}{\beta-\alpha} b_{ij}^{n-r+1}(s,t) + \frac{s-\alpha}{\beta-\alpha} b_{i+1,j}^{n-r+1}(s,t)\right) \\ &\quad + \frac{t-\gamma}{\delta-\gamma}\left(\frac{\beta-s}{\beta-\alpha} b_{i,j+1}^{n-r+1}(s,t) + \frac{s-\alpha}{\beta-\alpha} b_{i+1,j+1}^{n-r+1}(s,t)\right) \end{aligned}$$

for $0 \leq i,j \leq n-r$. Clearly, these values have the form

$$b_{ij}^{n-r}(s,t) = M_P(n-i-r\#\alpha, i\#\beta, r\#s; n-j-r\#\gamma, j\#\delta, r\#t)$$

and the process ends up with

$$b_{00}^0(s,t) = M_P(n\#s; n\#t) = P(s,t).$$

But let us look at the cost. We have to work on a full array of $n^2, (n-1)^2, \ldots$ control vectors, and this yields a $\mathcal{O}(dn^3)$ computational cost.

In general, the calculated intermediate points of this method are not sufficient to form subdivision control nets. A full split of a control net into four subnets will take two applications of the first technique.

5.6 Flatness Test and Convergence of Subdivision

To look at subdivision and its convergence, we first need some bound on the distance between the original surface $P(u,v)$ of the form (5.1) and a simplified surface L. Our candidate is the bilinear surface L defined by the four control points $b_{00}, b_{m0}, b_{0n}, b_{mn}$ and with subsequent degree elevation. If we first lift the degree with respect to the variable u, we get the two control nets $\{b'_{i0}\}_i := \{b_{00}+(i/m)(b_{m0}-b_{00})\}_i$ and $\{b'_{in}\}_i := \{b_{0n}+(i/m)(b_{mn}-b_{0n})\}_i$ and the representation

$$L(u,v) = \sum_{i=0}^{m}(b'_{i0}(1-v)+b'_{in}v)\beta_i^{(n)}(u)$$

on $[0,1]^2$, and we can then lift the degree of the inner bracket to get the control net

$$\begin{array}{rcl} b''_{ij} & := & b'_{i0} + (j/n)(b'_{in} - b'_{i0}) \\ & = & b_{00} + (i/m)(b_{m0} - b_{00}) + (j/n)(b_{0n} + (i/m)(b_{mn} - b_{0n}) - b_{00} + (i/m)(b_{m0} - b_{00})) \\ & = & ((m-i)(n-j)b_{00} + (m-i)jb_{0n} + i(n-j)b_{m0} + ijb_{mn})\,/(mn) \end{array} \tag{5.4}$$

that we can use to compare with the control net of P. Due to the partition of unity property, we have

$$\begin{array}{rcl} \|P(u,v)-L(u,v)\|_\infty & = & \|\sum_{i=0}^{m}\sum_{j=0}^{n}(b_{ij}-b''_{ij})\beta_i^{(m)}(u)\beta_j^{(n)}(v)\|_\infty \\ & \le & \max\limits_{0\le i\le m, 0\le j\le n}\|b_{ij}-b''_{ij}\|_\infty \sum_{i=0}^{m}\sum_{j=0}^{n}\left|\beta_i^{(m)}(u)\beta_j^{(n)}(v)\right| \\ & = & \max\limits_{0\le i\le m, 0\le j\le n}\|b_{ij}-b''_{ij}\|_\infty \end{array} \tag{5.5}$$

as a possible flatness test.

We now want to show that the bound gets small if the domain gets small. For this, the differences $b_{ij}-b''_{ij}$ are treated as usual, i.e. by Taylor expansion of the multiaffine form, asserting that the constant and linear terms vanish, yielding a second–order bound with respect to the domain size. Since we work with a single polynomial piece, and since the first m and the last n arguments of the multiaffine form can be permuted, we just express the b_{ij} and b''_{ij} as multiaffine forms

$$\begin{array}{rcl} b_{ij} & = & M_P(u_1,\ldots,u_m; r_1,\ldots,r_n) \\ b''_{ij} & = & M_P(v_1,\ldots,v_m; s_1,\ldots,s_n) \end{array}$$

and prove

$$\sum_{i=0}^{m}(u_i-v_i) = 0 = \sum_{j=0}^{n}(r_j-s_j)$$

to get the asserted quadratic behaviour. This is due to the simple fact that for a C^2 map $F : I\!R^n \mapsto I\!R^d$ we can bound the difference $F(x) - F(y)$ via local expansion around some point z as

$$\begin{array}{rcl} F(x) - F(y) & = & F(z) + \nabla F(z)(x-z) - F(z) - \nabla F(z)(y-z) + \text{ second–order terms} \\ F(x) - F(y) & = & \nabla F(z)(x-y) + \text{ second–order terms,} \end{array}$$

where the second–order terms can be bounded by an upper bound on sums of absolute values of second derivatives of F near x, y, and z, multiplied by $\|x-y\|_\infty^2$. In our case, all first–order derivatives are the same, and thus we are left with the quadratic residuals if we can prove $\sum_i (x_i - y_i) = 0$. (I added this remark due to the problems the students had with Exercises 24 and 25).

Now back to our multiaffine forms. From (5.2) we can read off the multiaffine representation for b_{ij}, and we can sum up the arguments in two groups:

$$(m-i)\alpha + i\beta \text{ and } (n-j)\gamma + j\delta. \tag{5.6}$$

Now we still have to look at b''_{ij} in(5.4). The latter is a bi–affine combination of the four control points b_{00}, b_{m0}, b_{0n}, and b_{mn}, and therefore we can take bi–affine combinations of the respective arguments when plugged into (5.2). Then we have to sum the arguments up, to prove that the result equals (5.6). For

$$mnb''_{ij} = (m-i)(n-j)b_{00} + (m-i)jb_{0n} + i(n-j)b_{m0} + ijb_{mn}$$

this yields the argument sums

$$\begin{array}{rcl} (m-i)(n-j)m\alpha + (m-i)jm\alpha + i(n-j)m\beta + ijm\beta & = & mn((m-i)\alpha + i\beta) \\ (m-i)(n-j)n\gamma + (m-i)jn\delta + i(n-j)n\gamma + ijn\gamma & = & mn((n-j)\gamma + j\delta) \end{array}$$

as expected. All arguments are in the respective domains, and thus all argument differences can be bounded by $\max(\beta-\alpha, \delta-\gamma)$. Thus we finally have proven

Theorem 5.3 *Subdivision of rectangular polynomial Bernstein–Bézier surface patches converges quadratically. More precisely; the right–hand side of the final line in* (5.5) *can be used as a flatness test, and this quantity is bounded by the product of a constant (depending only on the starting domain and second derivatives of the surface polynomial) with* $(\max(\beta-\alpha, \delta-\gamma))^2$.

5.7 Smooth Transitions

We now want to glue two rectangular Bernstein–Bézier surface patches together. For this, we have to assume that the domain rectangles coincide on a full edge, and that along this edge the degree is the same on either side. Thus we assume a surface P of degree (m, n) to be defined on $[\alpha, \beta] \times [\gamma, \delta]$, while Q is defined on $[\alpha, \beta] \times [\delta, \epsilon]$ with degree (m, k). The domains meet on the line $v = \delta$, and we have to look at the cross–boundary derivatives $\partial^\ell / \partial v^\ell$ at $v = \delta$ as functions of u. For a C^r transition, these must be equal for $0 \le \ell \le r$ when calculated from either side.

If we use (5.1), we can write P in Bernstein–Bézier form

$$\begin{aligned}
P(u,v) &= \sum_{i=0}^{m}\sum_{j=0}^{n} b_{ij}\beta_i^{(m)}(u)\beta_j^{(n)}(v)\\
&= \sum_{i=0}^{m}\beta_i^{(m)}(u)\sum_{j=0}^{n} b_{ij}\beta_j^{(n)}(v)\\
\frac{\partial^\ell P}{\partial v^\ell}(u,v) &= \sum_{i=0}^{m}\beta_i^{(m)}(u)\frac{\partial^\ell P}{\partial v^\ell}\sum_{j=0}^{n} b_{ij}\beta_j^{(n)}(v)\\
&= \sum_{i=0}^{m}\beta_i^{(m)}(u)\frac{n!}{(n-\ell)!(\delta-\gamma)^\ell}\sum_{j=0}^{n-\ell}\Delta_j^\ell b_{i,\cdot}\beta_j^{(n-\ell)}(v)\\
\frac{\partial^\ell P}{\partial v^\ell}(u,\delta) &= \sum_{i=0}^{m}\beta_i^{(m)}(u)\frac{n!}{(n-\ell)!(\delta-\gamma)^\ell}\Delta_{n-\ell}^\ell b_{i,\cdot}
\end{aligned}$$

where we take forward differences in the respective control nets. Doing this on both sides will yield similar expressions, and since the corresponding Bernstein polynomials are linearly independent, we can compare the coefficients to check the smoothness at the transition. If we write

$$Q(u,v) = \sum_{i=0}^{m}\sum_{j=0}^{k} c_{ij}\beta_i^{(m)}(u)\beta_j^{(k)}(v)$$

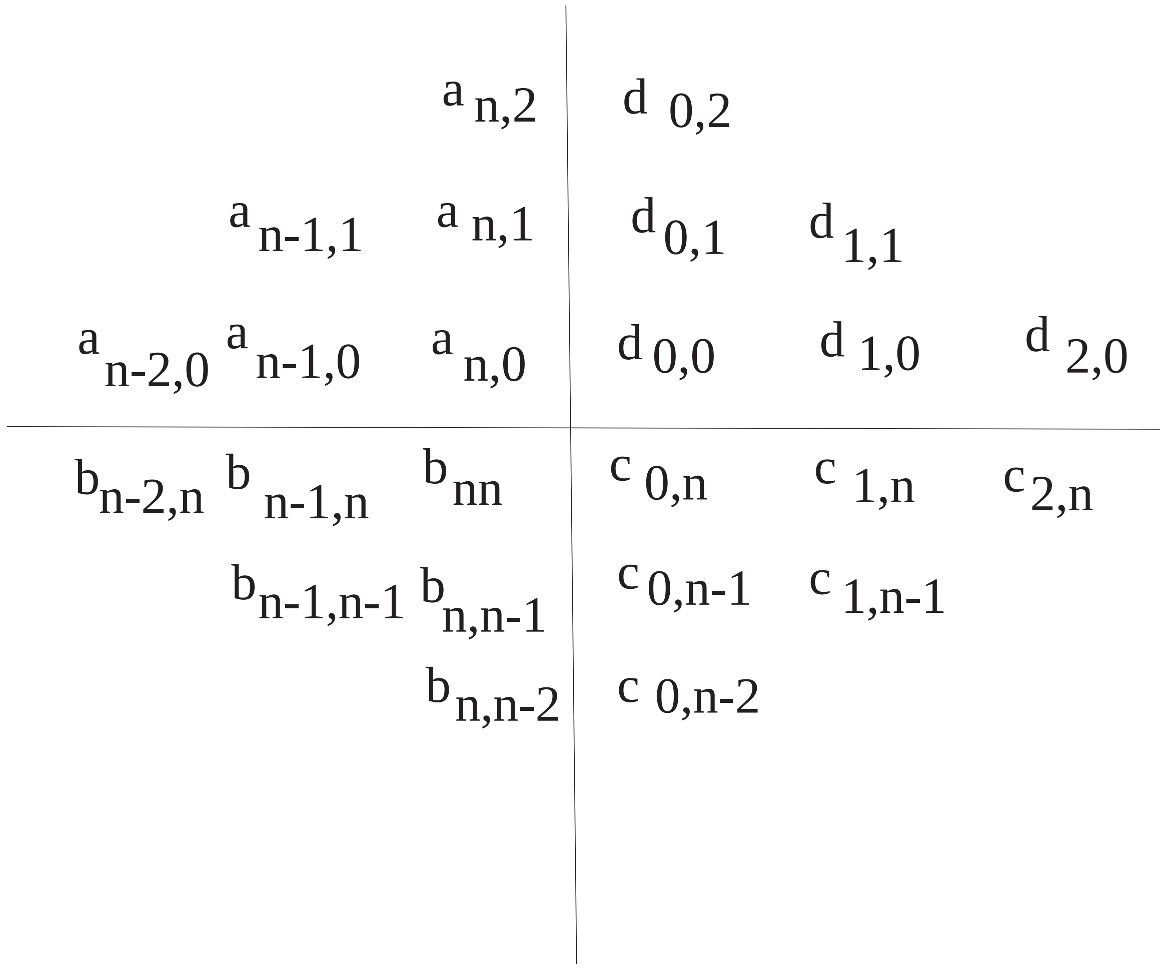
on the other side, this implies

$$\frac{n!}{(n-\ell)!(\delta-\gamma)^\ell}\Delta_{n-\ell}^\ell b_{i,\cdot} = \frac{k!}{(k-\ell)!(\epsilon-\delta)^\ell}\Delta_0^\ell c_{i,\cdot},\ 0\le\ell\le r,\ 0\le i\le m$$

as the full set of conditions for a C^r transition. The conditions can be rephrased in terms of de Casteljau steps or multiaffine forms, but we skip over these details.

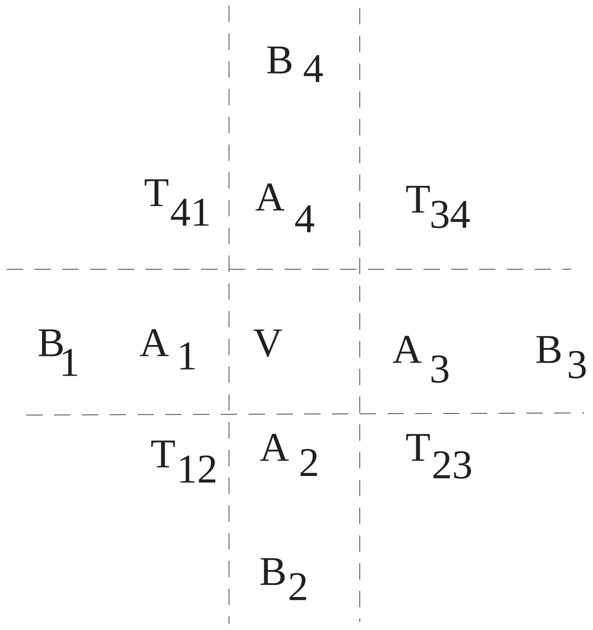
5.8 Patching Several Rectangular Surfaces

The previous paragraph patched just two rectangular polynomial surfaces in Bernstein–Bézier form together. If we want to repeat this in order to get a surface composed of lots of rectangular patches, we still have to restrict the transitions to full edges, and therefore the standard way of multiple patching involves four patches meeting at a "crossroads" vertex. For simplicity, we assume the degree to be equal to n everywhere, possibly after some degree elevation.



If we write down the continuity requirements for the four C^r transitions involved, the conditions will have a serious overlap, because the $r+1$ "strips" of control points near and parallel to the common edges are involved. For $r = 0$, we see that the control points at the joining vertex must be the same for all four patches. In the figure, where we ordered the control points around the vertex corresponding to their numbering (they may lie much more chaotically in $I\!R^3$), this means $a_{n0} = b_{nn} = c_{0n} = d_{00}$. Continuity along the border edges means $a_{i0} = b_{in}, b_{ni} = c_{0i}, c_{in} = d_{i0}, d_{0i} = a_{ni}$, each for $0 \leq i \leq n$, and these equations imply the preceding four.

If we go for $r = 1$, we need that all partial derivatives at the common vertex lie in the same (tangent) plane. This requires the five points $a_{n0} = b_{nn} = c_{0n} = d_{00}, a_{10} = b_{1n}, b_{n1} = c_{01}, c_{1n} = d_{10}, d_{01} = a_{n1}$ to be coplanar. Furthermore, continuity of the cross–derivatives along the joining edges needs $a_{j1} - a_{j0} = b_{jn} - b_{j,n-1}, b_{nj} - b_{n-1,j} = c_{1j} - c_{0j}, c_{jn} - c_{j,n-1} = d_{j1} - d_{j0}, a_{n-1,1} - a_{n-1,0} = d_{1j} - d_{0j}$ for $0 \leq j \leq n$. But these equations are not independent, because the **twist vectors** $a_{n-1,1}, b_{n-1,n-1}, c_{1,n-1}, d_{1,1}$ occur twice.



In this simplified plot we already have incorporated the C^0 conditions at the vertex V, and we rewrite the C^1 conditions:

$$\begin{aligned} 2A_1 &= T_{41} + T_{12} \\ 2A_2 &= T_{12} + T_{23} \\ 2A_3 &= T_{23} + T_{34} \\ 2A_4 &= T_{34} + T_{41} \end{aligned}$$

But, if we know the vectors A_1, A_2, A_3, A_4 from the C^0 transition on the joining edges, we are not able (in general) to solve the above system for the four unknown twist vectors! In fact, the homogeneous system has the nontrivial solution $(T_{41}, T_{12}, T_{23}, T_{34}) = (+1, -1, +1, -1)$. This makes it hard to construct piecewise rectangular surfaces just from precribed curves along the joining edges. A more close inspection yields that the equations $2A_1 + 2A_3 = 2A_2 + 2A_4 = 4V = T_{41} + T_{12} + T_{23} + T_{34}$ must hold for C^1 across the vertex V, and thus one of the equations is redundant anyway. If the A_i vectors are satisfying the above condition, the choice $T_{ij} = A_i + A_j - V$ does the job nicely, and without any further calculation. The reader should by now have realized that the coupling of the continuity requirements across a vertex may cause problems. The situation for surfaces on triangles will be even worse, because there may be any number of triangles meeting at a vertex.

6 Bernstein–Bézier Representations On Simplices

6.1 Generalized Bernstein Polynomials

We now want to generalize the case of polynomial Bernstein–Bézier curves to surfaces, 3D bodies, etc. The basic trick to generate a partition of unity is to write $1 = 1^n$ and to replace 1 in the right–hand side by an affine combination. Here, we want to work on d–dimensional simplices, and thus it is natural to use barycentric coordinates $\lambda_0(x), \lambda_1(x), \ldots, \lambda_d(x)$ of a point $x \in I\!R^d$ with respect to $d+1$ points $x_0, x_1, \ldots, x_d \in I\!R^d$ in general position, i.e. forming a simplex. Note that we have curve pieces on intervals for $d = 1$ and surfaces on triangles for $d = 2$.

We use the multinomial formula and multi–index notation to get

$$\begin{aligned}
1 &= 1^n \\
&= \left(\sum_{k=0}^{d} \lambda_j(x)\right)^n \\
&= \sum_{\substack{0 \le j_0, j_1, \dots, j_d \le n \\ j_0 + j_1 + \dots + j_d = n}} \frac{n!}{j_0! j_1! \cdots j_d!} \lambda_0^{j_0}(x) \lambda_1^{j_1}(x) \cdot \ldots \cdot \lambda_d^{j_d}(x) \\
&= \sum_{j \in I\!N_0^{d+1},\ |j|=n} \beta_j^{(n)}(x) \\
\beta_j^{(n)}(x) &= \frac{n!}{j_0! j_1! \cdot \ldots \cdot j_d!} \lambda_0^{j_0}(x) \lambda_1^{j_1}(x) \cdot \ldots \cdot \lambda_d^{j_d}(x) = \binom{n}{j} \lambda^j(x)
\end{aligned}$$

where the last line contains generalized Bernstein polynomials, forming a partition of unity on the base triangle.

The above notation looks somewhat strange at first, but we can go back to the univariate case and see what we have there, if we work on $[x_0, x_1] = [\alpha, \beta]$:

$$\begin{aligned}
1 &= \lambda_0(x) + \lambda_1(x) = \frac{x_1 - x}{x_1 - x_0} + \frac{x - x_0}{x_1 - x_0} \\
1 &= 1^n \\
&= (\lambda_0(x) + \lambda_1(x))^n \\
&= \sum_{\substack{0 \le j_0, j_1 \le n \\ j_0 + j_1 = n}} \frac{n!}{j_0! j_1!} \lambda_0^{j_0}(x) \lambda_1^{j_1}(x) \\
&= \sum_{j=0}^{n} \frac{n!}{j!(n-j)!} \lambda_0^{j}(x) \lambda_1^{n-j}(x) \\
&= \sum_{j=0}^{n} \frac{n!}{j!(n-j)!} \left(\frac{x_1 - x}{x_1 - x_0}\right)^j (x) \left(\frac{x - x_0}{x_1 - x_0}\right)^{n-j} (x)
\end{aligned}$$

We see that in the univariate case we should have given the j–th Bernstein polynomial the pair $(j, n-j)$ of indices, summing up to n, and running over $j = 0, \dots, n$.

We now have our partition of unity, and we can generate vector–valued polynomials by a control net representation using the same index set. In particular, we take a control net $\{b_j\}_{j \in I\!N_0^{d+1},\ |j|=n} \subset I\!R^D$, and then we define a Bernstein–Bézier representation over a simplex by

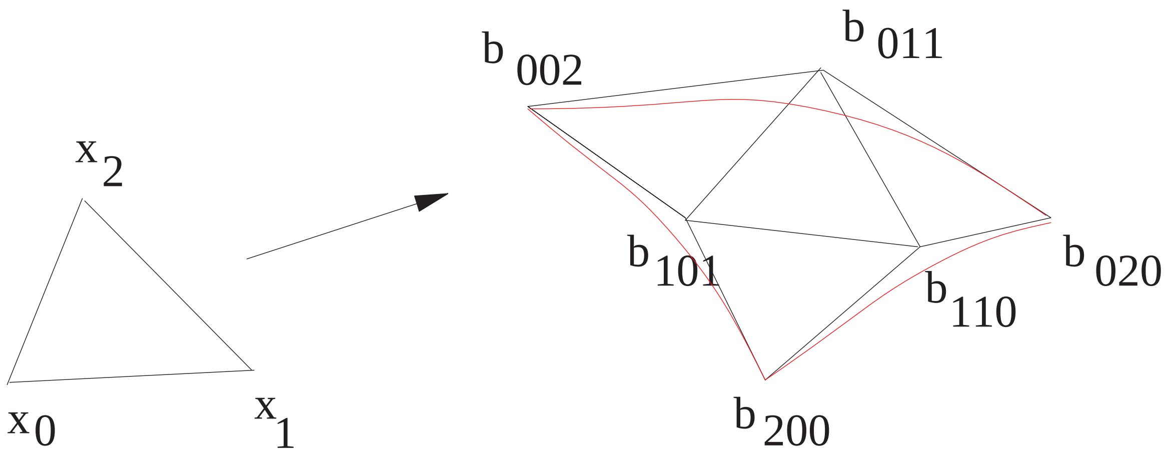
$$P(x) = \sum_{j \in I\!N_0^{d+1},\ |j|=n} b_j \beta_j^{(n)}(x). \tag{6.1}$$

6.2 Special Cases for Surfaces

Let us look first at the case $n = 1$. We construct a linear polynomial via three control points $b_{100}, b_{010}, b_{001} \in I\!R^3$ and a triangle spanned by non–collinear points $x_0, x_1, x_2 \in I\!R^2$. The surface is

$$P(x) = b_{100}\lambda_0(x) + b_{010}\lambda_1(x) + b_{001}\lambda_2(x)$$

forming a planar triangle in $I\!R^D$ with vertices $b_{100}, b_{010}, b_{001}$.

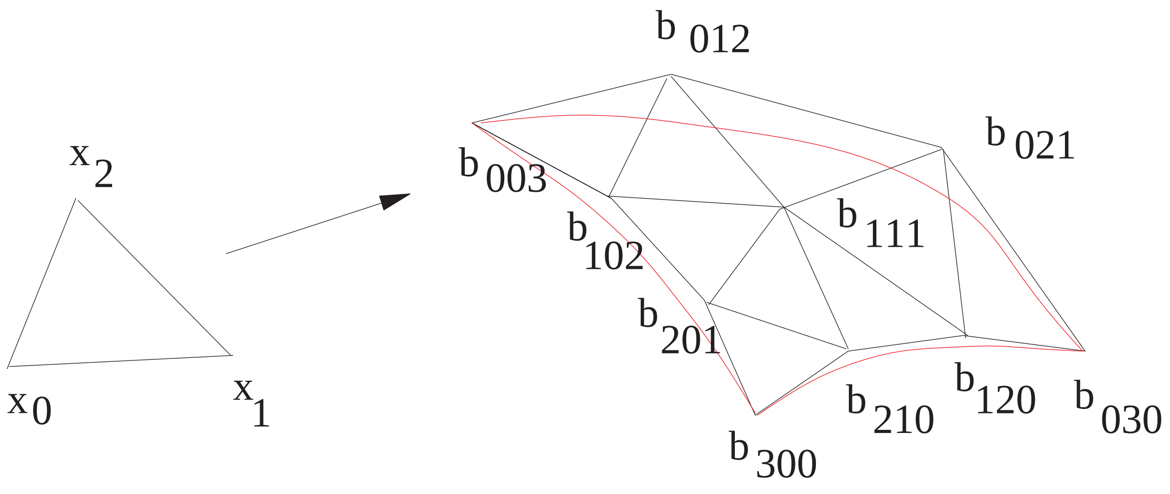


The quadratic case has six control points that are usually arranged in the form of a subdivided triangle

$$\begin{array}{ccccc} & & b_{002} & & \\ & b_{101} & & b_{011} & \\ b_{200} & & b_{110} & & b_{020} \end{array}$$

which can be viewed as the image of the base triangle

$$\begin{array}{ccc} & x_2 & \\ & & \\ x_0 & & x_1 \end{array}$$



Correspondingly, the cubic case looks like

$$\begin{array}{ccccccc} & & & b_{003} & & & \\ & & b_{102} & & b_{012} & & \\ & b_{201} & & b_{111} & & b_{021} & \\ b_{300} & & b_{210} & & b_{120} & & b_{030} \end{array}$$

6.3 The Algorithm of de Casteljau

We derive it the classical way and first generalize Pascal's triangle to

$$\frac{n!}{j_0!j_1!\cdot\ldots\cdot j_d!}=\sum_{k=0}^{d}\frac{(n-1)!j_k}{j_0!j_1!\cdot\ldots\cdot j_d!}=\sum_{k=0}^{d}\frac{(n-1)!}{j_0!j_1!\cdot\ldots\cdot j_{k-1}!(j_k-1)!j_{k+1}!\cdot\ldots\cdot j_d!}$$

for all $j, j\in I\!N_0^{d+1},\ |j|=n$, where terms with factorials of negative numbers are ignored.

If we denote the unit vectors by $e_0, e_1, \ldots, e_d$, the recursion for the factorials can be rewritten as

$$\binom{n}{j}=\sum_{k=0}^{d}\binom{n-1}{j-e_k}$$

and this immediately implies

$$\beta_j^{(n)}(x)=\binom{n}{j}\lambda^j(x)=\sum_{k=0}^{d}\binom{n-1}{j-e_k}\lambda^{j-e_k}(x)\lambda_k^1(x)=\sum_{k=0}^{d}\lambda_k^1(x)\beta_{j-e_k}^{(n-1)}(x)$$

where we omit a term as soon as a component of its index gets negative.

Now we proceed to the de Casteljau algorithm by starting with

$$b_j^{(n)}(x):=b_j,\ j\in I\!N_0^{d+1},\ |j|=n$$

and performing the recursion

$$\begin{array}{rcl}
P(x) & = & \displaystyle\sum_{j\in I\!N_0^{d+1},\ |j|=n} b_j^{(n)}(x)\beta_j^{(n)}(x)\\
& = & \displaystyle\sum_{j\in I\!N_0^{d+1},\ |j|=n} b_j^{(n)}(x)\sum_{k=0}^{d}\lambda_k^1(x)\beta_{j-e_k}^{(n-1)}(x)\\
(m & = & j-e_k)\\
& = & \displaystyle\sum_{;m\in I\!N_0^{d+1},\ |m|=n-1}\beta_m^{(n-1)}(x)\sum_{k=0}^{d}\lambda_k^1(x)b_{m+e_k}^{(n)}(x)\\
& = & \displaystyle\sum_{;m\in I\!N_0^{d+1},\ |m|=n-1}\beta_m^{(n-1)}(x)b_m^{(n-1)}(x)\\
& = & \displaystyle\sum_{m\in I\!N_0^{d+1},\ |m|=n-r}\beta_m^{(n-r)}(x)b_m^{(n-r)}(x)
\end{array}$$

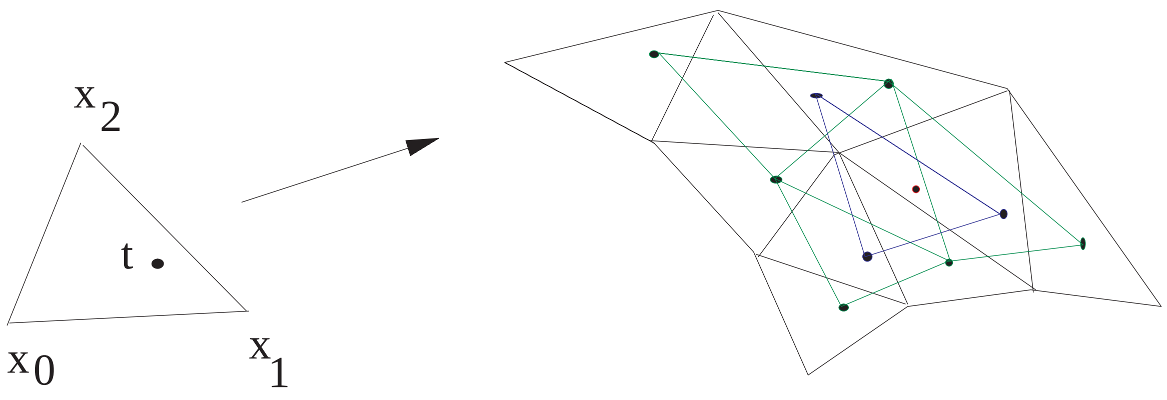
via

$$b_m^{(n-r)}(x)=\sum_{k=0}^{d}\lambda_k^1(x)b_{m+e_k}^{(n-r+1)}(x)$$

for all $m\in I\!N_0^{d+1},\ |m|=n-r,\ 0\le r\le n$. This ends up with

$$P(x)=\beta_0^{(0)}(x)b_0^{(0)}(x)=b_0^{(0)}(x)$$

as expected.



6.4 Degree Elevation

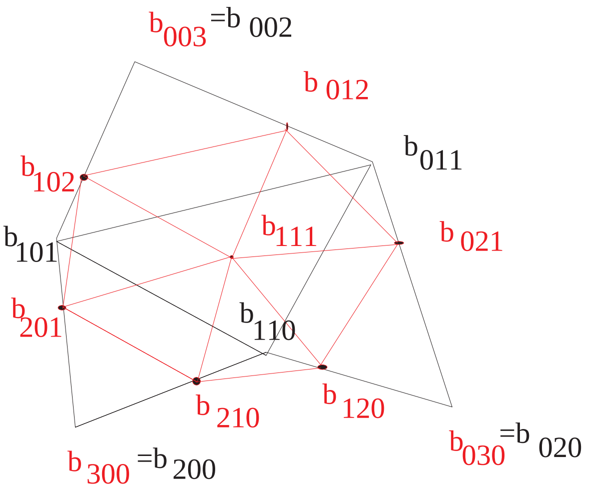
We want to raise the degree n of a Bernstein–Bézier representation (6.1) of a polynomial P.

$$\begin{aligned}
P(x) &= \sum_{j\in I\!N_0^{d+1},\ |j|=n} b_j\beta_j^{(n)}(x)\cdot 1\\
&= \sum_{j\in I\!N_0^{d+1},\ |j|=n} b_j \binom{n}{j} \lambda^j(x)\sum_{k=0}^{d}\lambda_k(x)\\
&= \sum_{j\in I\!N_0^{d+1},\ |j|=n} b_j \binom{n}{j} \sum_{k=0}^{d}\lambda^{j+e_k}(x)\frac{\binom{n+1}{j+e_k}}{\binom{n+1}{j+e_k}}\\
(m &= j+e_k)\\
&= \sum_{m\in I\!N_0^{d+1},\ |m|=n+1} \beta_m^{(n+1)}(x) \sum_{k=0,m_k>0}^{d} b_{m-e_k}\frac{\binom{n}{m-e_k}}{\binom{n+1}{m}}\\
&= \sum_{m\in I\!N_0^{d+1},\ |m|=n+1} \beta_m^{(n+1)}(x) \sum_{k=0,m_k>0}^{d} b_{m-e_k}\frac{m_k}{n+1}\\
&= \sum_{m\in I\!N_0^{d+1},\ |m|=n+1} \beta_m^{(n+1)}(x) \sum_{k=0}^{d} b_{m-e_k}\frac{m_k}{n+1}.
\end{aligned}$$

The final form makes sense if we ignore terms with illegal indices or zero scalar factors. The new control points for degree $n+1$ and indices $m \in I\!N_0^{d+1}$, $|m| = n+1$ are weighted means of the nearby control points:

$$\sum_{k=0}^{d}\frac{m_k}{n+1}b_{m-e_k},$$

where the weights neatly correspond to the index vectors.



Let us illustrate degree elevation from 2 to 3. Note that on the boundary we can work with the univariate rule, i.e. by splitting in thirds. The central point b_{111} is the mean of the points b_{011}, b_{101}, and b_{110}.

6.5 Multiaffine Forms

It is now easy to guess how the control points of a polynomial Bernstein–Bézier representation P over a simplex $X \subset I\!R^d$ spanned by $d+1$ points $x_0, x_1, \ldots, x_d \in I\!R^d$ in general position can be obtained from the multiaffine form M_P for P. We simply define

$$b_j := M_P(j\#x) := M_P(j_0\#x_0, j_1\#x_1, \ldots, j_d\#x_d),\ j \in I\!N_0^{d+1},\ |j| = n \tag{6.2}$$

and see how far we get with this. Let us forget about Bernstein polynomials and just evaluate P on $t \in X$ via the multiaffine forms. We get the de Casteljau algorithm again! In fact, if we represent t affinely by

$$t = \sum_{k=0}^{d} \lambda_k(t) x_k$$

we can introduce t into the multiaffine form, step by step. We define $b_j^{(n)}(t) := b_j$ to start, and we assert

$$b_m^{(n-r)}(t) = M_P(m\#x, r\#t),\ |m| = n-r,\ 0 \le r \le n,$$

which is true for $r = 0$. If we assume the above representation for some $r-1$ with $r > 0$, then we can write

$$\begin{aligned} M_P(m\#x, r\#t) &= M_P(m\#x, r-1\#t, \sum_{k=0}^{d} \lambda_k(t) x_k) \\ &= \sum_{k=0}^{d} \lambda_k(t) M_P(m\#x, r-1\#t, x_k) \\ &= \sum_{k=0}^{d} \lambda_k(t) M_P(m+e_k\#x, r-1\#t) \\ &= \sum_{k=0}^{d} \lambda_k(t) b_{m+e_k}^{(n-r+1)}(t) \\ &= b_m^{(n-r)}(t) \end{aligned}$$

provided that we take the de Casteljau recursion for the final step, and this proves our assertion. Now $b_0^0(t) = M_P(n\#t) = P(t)$ holds, and we have reconstructed P.

As in the spline case, we do not need to know the partition of unity, i.e. the multivariate Bernstein polynomials, to derive the de Casteljau formulae. But we want to prove that the Bernstein–Bézier coefficients have precisely the above form:

Theorem 6.1 *If a polynomial P of degree n is represented in Bernstein–Bézier form over a simplex X spanned by $d+1$ points $x_0, x_1, \dots, x_d \in \mathbb{R}^d$, its control points are necessarily of the form* (6.2)*.*

Proof: We do this the same way as in the spline case, invoking a dimension argument.

Let us first count the number of nonnegative integer index vectors $j \in \mathbb{N}_0^{d+1}$ with $|j| = n$. It is the number of integer index vectors $k \in \mathbb{N}_0^d$ witk $0 \leq |k| \leq n$ because we can uniquely extend each k to $(k, n-|k|)$. Thus this number is the same as the number of d–variate monomials x^k with $0 \leq |k| \leq n$. Therefore the space B of all d–variate polynomials of degree at most n with values in $\mathbb{R}^D$ has the same dimension as the space C for the full control nets $\{b_j \in \mathbb{R}^D\}_{j \in \mathbb{N}_0^{d+1}, |j|=n}$.

The linear mapping S from B to C defined by (6.2) and the linear mapping T from C to B defined by the de Casteljau algorithm have the property $T \circ S = Id$, due to our second presentation of the de Casteljau algorithm. Since B and C have the same finite dimension, S and T must be bijective. But then $T^{-1}P$ leads to the unique control net that generates P via the de Casteljau algorithm, and this is what we wanted to prove. □

By the same argument, we see that the multivariate Bernstein polynomials are linearly independent.

6.6 Subdivision

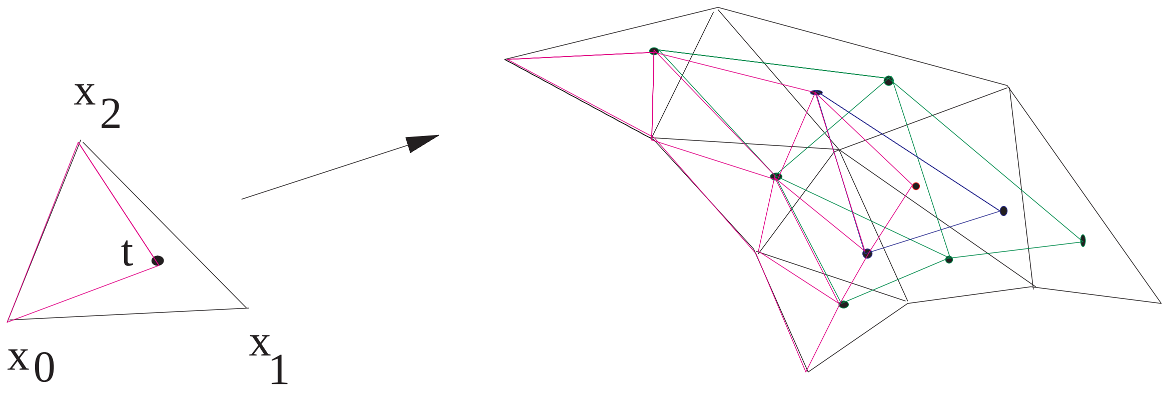
As in the case of surfaces on rectangles, we have several possibilities to do subdivision. The simple de Casteljau process for some t in a simplex spanned by $x_0, x_1, \dots, x_d \in \mathbb{R}^d$ generates intermediate points

$$b_\ell^{(n-r)}(t) = M_P(\ell\#x, r\#t), \ |\ell| = n-r, \ 0 \leq r \leq n,$$

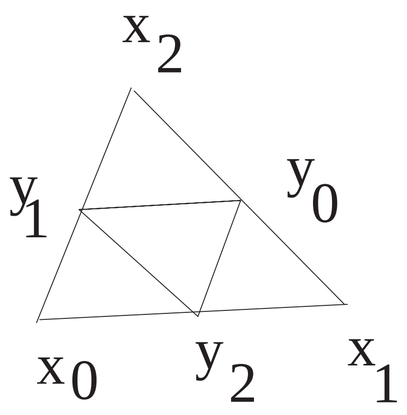
as we have seen. Among these are $d+1$ sets for the simplices spanned by letting t replace one of the simplex vertices. The simplex X_0 where t replaces x_0 needs the points

$$b_{\ell-\ell_0 e_0}^{(n-\ell_0)}(t) = M_P(0\#x_0, \ell_1\#x_1, \dots, \ell_d\#x_d, \ell_0\#t), \ |\ell| = n,$$

and the other simplices are treated similarly.



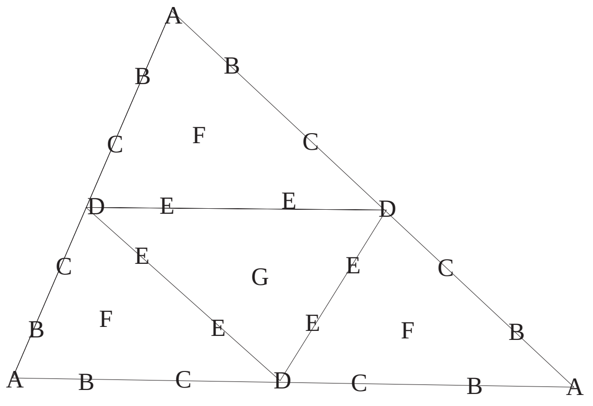
Splitting a triangle this way is not the best possible thing, because the angles get small and the triangles degenerate into short lines. A better way is to bisect the edges and to generate four similar triangles.



If y_i is the point in opposition to x_i, we have to generate the four sets

$$\begin{array}{c} M_P(\ell_0\#x_0, \ell_1\#y_1, \ell_2\#y_2) \\ M_P(\ell_0\#y_0, \ell_1\#x_1, \ell_2\#y_2) \\ M_P(\ell_0\#y_0, \ell_1\#y_1, \ell_2\#x_2) \\ M_P(\ell_0\#y_0, \ell_1\#y_1, \ell_2\#y_2) \end{array}$$

of control points, where $|\ell| = 3$. Of course one could write a recursive routine to do the evaluation, replacing points y_k by their neighbors x_{k-1} and x_{k+1}, taking indices mod 3.



But one can determine the correct linear combinations beforehand. Each control point of the four sets of 10 control points we look for must be an affine combination of the 10 control points we started with, and the weights should just be determined by the indices. The points we have to calculate can be split up in different classes A to G, depending on their relative position, as in the figure above. The univariate subdivision process can be used to determine the points of type A to D. Here are the examples for points of types B to D for the control net b'_j in the upper triangle:

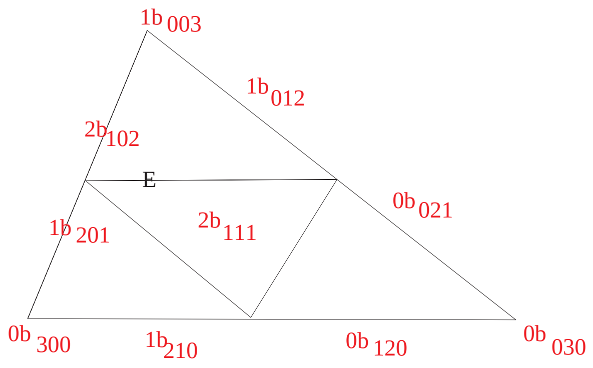
$$\begin{array}{rcll} M_P(y_0, x_2, x_2) &=& (M_P(x_1, x_2, x_2) + M_P(x_2, x_2, x_2))/2 & B \\ b'_{012} &=& (b_{012} + b_{003})/2 & B \\ M_P(y_0, y_0, x_2) &=& (M_P(x_1, x_1, x_2) + 2M_P(x_1, x_2, x_2) + M_P(x_2, x_2, x_2))/4 & C \\ b'_{021} &=& (b_{021} + 2b_{012} + b_{003})/4 & C \\ M_P(y_0, y_0, y_0) &=& (M_P(x_1, x_1, x_1) + 3M_P(x_1, x_1, x_2) + 3M_P(x_1, x_2, x_2) + M_P(x_2, x_2, x_2))/8 & D \\ b'_{030} &=& (b_{030} + 3b_{021} + 3b_{012} + b_{003})/8 & D \end{array}$$

For type F we pick the example

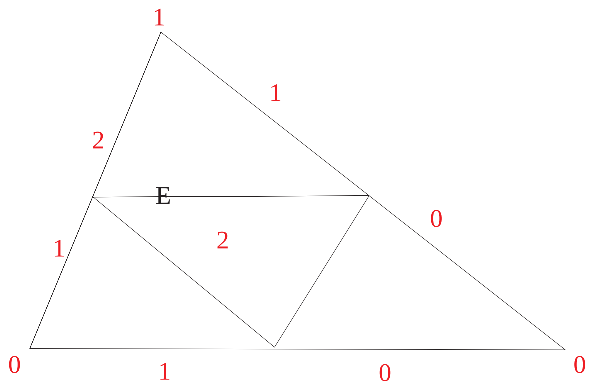
$$\begin{aligned} M_P(y_0, y_1, x_2) &= M_P((x_1 + x_2)/2, (x_0 + x_2)/2, x_2) \\ b'_{111} &= (M_P(x_1, x_0, x_2) + M_P(x_1, x_2, x_2) + M_P(x_2, x_0, x_2) + M_P(x_2, x_2, x_2))/4 \end{aligned}$$

and for type E we have

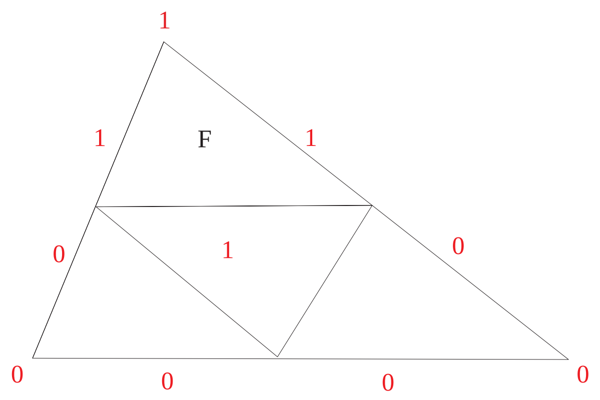
$$\begin{aligned} M_P(y_0, y_1, y_1) &= M_P((x_1 + x_2)/2, (x_0 + x_2)/2, (x_0 + x_2)/2) \\ b'_{210} &= (M_P(x_1, x_0, (x_0 + x_2)/2) + M_P(x_1, x_2, (x_0 + x_2)/2)+ \\ &\quad M_P(x_2, x_0, (x_0 + x_2)/2) + M_P(x_2, x_2, (x_0 + x_2)/2))/4 \\ &= (M_P(x_1, x_0, x_0) + M_P(x_1, x_0, x_2) + M_P(x_1, x_2, x_0) + M_P(x_1, x_2, x_2)+ \\ &\quad M_P(x_2, x_0, x_0) + M_P(x_2, x_0, x_2) + M_P(x_2, x_2, x_0) + M_P(x_2, x_2, x_2))/8 \\ &= (b_{210} + 2b_{111} + b_{012} + b_{201} + 2b_{102} + b_{003})/8 \end{aligned}$$



The above plot shows the weights (to be divided by 8) with respect to the original 10 control points. Or, look at the following simplified picture:



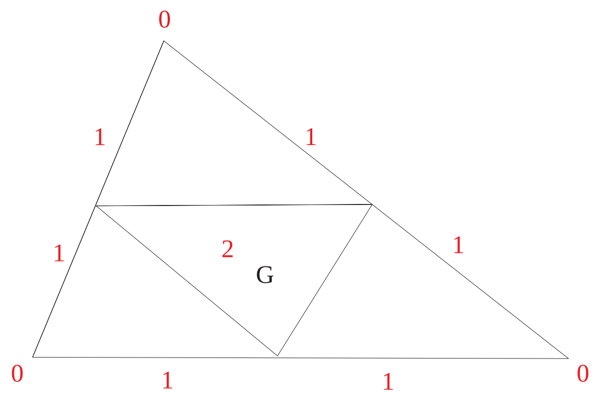
It should be clear by now how to get the other points of type E and F. The corresponding weights for F (to be divided by 4) are here:



The remaining case is G:

$$\begin{array}{rcl} M_P(y_0,y_1,y_2) & = & M_P((x_1+x_2)/2,(x_0+x_2)/2,(x_0+x_1)/2)\\ b^*_{111} & = & (M_P(x_1,x_0,(x_0+x_1)/2)+M_P(x_1,x_2,(x_0+x_1)/2)+\\ & & M_P(x_2,x_0,(x_0+x_1)/2)+M_P(x_2,x_2,(x_0+x_1)/2))/4\\ & = & (M_P(x_1,x_0,x_0)+M_P(x_1,x_0,x_1)+M_P(x_1,x_2,x_0)+M_P(x_1,x_2,x_1)+\\ & & M_P(x_2,x_0,x_0)+M_P(x_2,x_0,x_1)+M_P(x_2,x_2,x_0)+M_P(x_2,x_2,x_1))/8\\ & = & (b_{210}+b_{120}+2b_{111}+b_{021}+b_{201}+b_{102}+b_{012})/8 \end{array}$$

and here are the (relative) weights:



6.7 Derivatives

Let us recall from (2.5) that the derivative of an n–th degree polynomial P over a simplex in $I\!R^d$ in the direction $r\in I\!R^d$ has the symmetric multiaffine form

$$\begin{array}{rcl} M_{(\nabla_x^T r)P}(x_2,\ldots,x_n) & = & n\cdot(M_P(z+r,x_2,\ldots,x_n)-M_P(z,x_2,\ldots,x_n))\\ & = & n\Delta^u_{r|_z}M_P(u,x_2,\ldots,x_n) \end{array} \qquad (6.3)$$

for all $z\in I\!R^d$, and the directional derivative of P itself at y is

$$\begin{array}{rcl} (\nabla_x^T r)P_{|_y}=M_{(\nabla_x^T r)P}((n-1)\#y) & = & n\cdot(M_P(z+r,(n-1)\#y)-M_P(z,(n-1)\#y))\\ & = & n\Delta^u_{r|_z}M_P(u,(n-1)\#y) \end{array} \qquad (6.4)$$

for arbitrary $z\in I\!R^d$. Here $\Delta^z_{r|_y}$ is the first–order undivided difference evaluated at $y+r$ and y with respect to the variable z, i.e.

$$\Delta^z_{r|_y}g(z):=g(y+r)-g(y).$$

Again, a directional derivative of a polynomial is just a difference acting on the multiaffine form, and this generalizes easily to higher–order derivatives.

Lemma 6.1 *The k–th directional derivative at y in directions $r_1, \ldots, r_k \in I\!R^d$ of an n-th degree multivariate and vector–valued polynomial P is*

$$\nabla^z_{r_1|_y} \ldots \nabla^z_{r_k|_y} P(z) \;=\; \frac{n!}{(n-k)!} \Delta^{u_1}_{r_1|_{z_1}} \ldots \Delta^{u_k}_{r_k|_{z_k}} M_P(u_1, \ldots, u_k, (n-k)\#y)$$

and has the symmetric multiaffine form

$$M_{\nabla^z_{r_1|_y} \ldots \nabla^z_{r_k|_y} P(z)}(z_{k+1}, \ldots, z_n) = \frac{n!}{(n-k)!} \Delta^{u_1}_{r_1|_{z_1}} \ldots \Delta^{u_k}_{r_k|_{z_k}} M_P(u_1, \ldots, u_k, z_{k+1}, \ldots, z_n)$$

as a polynomial of degree $n-k$, where $z_1, \ldots, z_k \in I\!R^d$ can be arbitrary.

Proof: The first statement follows from the second. if all z_i are equal to y. But the second just is (6.3) extended by induction. □

Let us make all r_j equal to a fixed direction r and set the z_j equal to y. Then

$$\begin{aligned} M_{\nabla^{z\,k}_{r|_y} P(z)}(z_{k+1}, \ldots, z_n) &= \frac{n!}{(n-k)!} \Delta^{u_1}_{r\ |_y} \ldots \Delta^{u_k}_{r\ |_y} M_P(u_1, \ldots, u_k, z_{k+1}, \ldots, z_n) \\ &= \frac{n!}{(n-k)!} \sum_{j=0}^{k} \binom{k}{j} (-1)^{k-j} M_P(j\#(y+r), (k-j)\#y, z_{k+1}, \ldots, z_n) \end{aligned}$$

and

$$\left(\nabla^z_r\right)^k_{|_y} P(z) \;=\; \frac{n!}{(n-k)!} \sum_{j=0}^{k} \binom{k}{j} (-1)^{k-j} M_P(j\#(y+r), (n-j)\#y).$$

Assume now that we write P in Bernstein–Bézier representation on a simplex $S(x_0, \ldots, x_d)$ with control points $b_j = M_P(j\#x)$ for $|j| = n,\ j \in I\!N_0^{d+1}$. The directional derivatives are expressible via the control net if, for instance, we take $y = x_i$ and $r = x_m - x_i$ to describe derivatives from a vertex x_i towards another vertex x_m. The result is

$$\begin{aligned} \left(\nabla^z_{x_m - x_i}\right)^k_{|_{x_i}} P(z)_{|_{x_i}} &= \frac{n!}{(n-k)!} \sum_{j=0}^{k} \binom{k}{j} (-1)^{k-j} M_P(j\#x_m, (n-j)\#x_i) \\ &= \frac{n!}{(n-k)!} \sum_{j=0}^{k} \binom{k}{j} (-1)^{k-j} b_{je_m + (n-j)e_i} \end{aligned}$$

and we recover the formulae for derivatives of Bernstein–Bézier curves. More generally, the polynomial $\left(\nabla^z_{x_m - x_i}\right)^k_{|_{x_i}} P(z)$ of degree $n-k$ is represented via the control net

$$b_\ell(i, m, k) := \frac{n!}{(n-k)!} \sum_{j=0}^{k} \binom{k}{j} (-1)^{k-j} b_{\ell + je_m + (k-j)e_i},\ |\ell| = n-k,\ \ell \in I\!N_0^{d+1}.$$

over the same simplex as P.

6.8 Continuity Conditions over Simplicial Faces

We now want tor describe conditions in terms of control nets that describe a smooth C^m transition from one polynomial piece to another via an intersecting simplicial face. We take two polynomials P and Q over nondegenerate simplices $S := conv(x_0, x_1, \ldots, x_d)$ and $T := conv(y_0, x_1, \ldots, x_d)$ over the common face $conv(x_1, \ldots, x_d)$. Note that this numbering of vertices looks simple, but implies that the orientations of S and T differ. We shall assume that P and Q are degree–elevated to the same degree $n \geq m$.

Clearly, a C^m transition between P and Q on $S \cap T$ means that all directional derivatives in all directions r and of all orders $k \leq m$ taken at all points $y \in S \cap T$ must coincide. We now transform this statement step by step in order to find conditions in terms of control points.

Lemma 6.2 *A C^m transition between P and Q on $S \cap T$ is equivalent to*

$$\begin{aligned} & \Delta^{u_1}_{r_1|_{z_1}} \ldots \Delta^{u_k}_{r_k|_{z_k}} M_P(u_1, \ldots, u_k, (n-k)\#y) \\ = \; & \Delta^{u_1}_{r_1|_{z_1}} \ldots \Delta^{u_k}_{r_k|_{z_k}} M_Q(u_1, \ldots, u_k, (n-k)\#y) \end{aligned}$$

for all $r_1, \ldots, r_k, z_1, \ldots, z_k \in \mathbb{R}^d$, $0 \leq k \leq m$ and all $y \in S \cap T$.

Proof: We take an arbitrary $y \in S \cap T$ and apply Lemma 6.1. □.

Lemma 6.3 *A C^m transition between P and Q on $S \cap T$ is equivalent to*

$$M_P(z_1, \ldots, z_m, (n-m)\#y) = M_Q(z_1, \ldots, z_m, (n-m)\#y)$$

for all $z_1, \ldots, z_m \in \mathbb{R}^d$, $y \in S \cap T$.

Proof: From the preceding lemma we get equivalence to

$$M_P(z_1, \ldots, z_k, (n-k)\#y) = M_Q(z_1, \ldots, z_k, (n-k)\#y)$$

for all $z_1, \ldots, z_k \in \mathbb{R}^d$, $0 \leq k \leq m$ and all $y \in S \cap T$, because we can pick arbitrary points z_j and directions r_j anyway. The strongest case $k = m$ is the assertion of the lemma. □

Lemma 6.4 *A C^m transition between P and Q on $S \cap T$ is equivalent to*

$$M_P(z_1, \ldots, z_m, y_{m+1}, \ldots, y_n) = M_Q(z_1, \ldots, z_m, y_{m+1}, \ldots, y_n)$$

for all $z_1, \ldots, z_m \in \mathbb{R}^d$, $y_{m+1}, \ldots, y_n \in S \cap T$.

Proof: The assertion follows from the next lemma that we should have proven earlier. □

Lemma 6.5 *If two n–th degree polynomials P and Q coincide on a simplex C, then*

$$M_P(c_1, \ldots, c_n) = M_Q(c_1, \ldots, c_n) \textit{ for all } c_1, \ldots, c_n \in C.$$

Proof: Clearly, the polynomials P and Q have the same directional derivatives on C if these are evaluated at points in C and taken into directions leading into C again. Equation (6.4) then first proves

$$M_P(c_1, (n-1)\#c) = M_Q(c_1, (n-1)\#c) \text{ for all } c, c_1 \in C$$

and the rest can be done via induction. □

We consider control point representations of P and Q on s and T, respectively, by

$$\begin{aligned} b_j &= M_P(j_0\#x_0, j_1\#x_1, \ldots, j_d\#x_d),\ |j| = n,\ j \in I\!N_0^{d+1} \\ c_j &= M_Q(j_0\#y_0, j_1\#x_1, \ldots, j_d\#x_d),\ |j| = n,\ j \in I\!N_0^{d+1} \end{aligned}$$

and perform an "extrapolating" de Casteljau algorithm twice: we evaluate P at y_0 and Q at x_0, respectively. This yields control points

$$\begin{aligned} b_j^{(n-k)}(y_0) &= M_P(k\#y_0, j_0\#x_0, j_1\#x_1, \ldots, j_d\#x_d),\ |j| = n-k,\ j \in I\!N_0^{d+1},\ 0 \le k \le n \\ c_j^{(n-k)}(x_0) &= M_Q(k\#x_0, j_0\#y_0, j_1\#x_1, \ldots, j_d\#x_d),\ |j| = n-k,\ j \in I\!N_0^{d+1},\ 0 \le k \le n \end{aligned} \tag{6.5}$$

and due to Lemma 6.4 we see that

$$b_{\ell e_0+j}^{(n-k)}(y_0) = c_{ke_0+j}^{(n-\ell)}(x_0) \text{ for all } 0 \le \ell + k \le m,\ |j| = n-k-\ell,\ j_0 = 0 \tag{6.6}$$

is a necessary condition for a C^m transition.

To investigate sufficiency of (6.6), we note first that for fixed values of k and ℓ with $0 \le \ell+k \le m$ the polynomials $R_{k,\ell}$ with

$$M_{R_{k,\ell}}(y_{k+\ell+1}, \ldots, y_n) \ := \ M_P(k\#y_0, \ell\#x_0, y_{k+\ell+1}, \ldots, y_n)$$

are of degree $n-k-\ell$ and can be represented in Bernstein–Bézier form over $S \cap T$ by control points

$$\begin{aligned} & M_{R_{k,\ell}}(j_1\#x_1, \ldots, j_d\#x_d) && |j| = n-k-\ell,\ j \in I\!N_0^d \\ = \ & M_P(k\#y_0, \ell\#x_0, j_1\#x_1, \ldots, j_d\#x_d) && |j| = n-k-\ell,\ j \in I\!N_0^d \\ = \ & b_{\ell e_0+j}^{(n-k)}(y_0) && |j| = n-k-\ell,\ j \in I\!N_0^{d+1},\ j_0 = 0 \\ = \ & c_{ke_0+j}^{(n-\ell)}(x_0) && |j| = n-k-\ell,\ j \in I\!N_0^{d+1},\ j_0 = 0 \\ = \ & M_Q(\ell\#x_0, k\#y_0, j_1\#y_1, \ldots, j_d\#y_d) && |j| = n-k-\ell,\ j \in I\!N_0^d. \end{aligned}$$

This proves the identities

$$M_P(k\#y_0, \ell\#x_0, y_{k+\ell+1}, \ldots, y_n) = M_Q(k\#y_0, \ell\#x_0, y_{k+\ell+1}, \ldots, y_n)$$

whenever $y_{k+\ell+1}, \ldots, y_n$ are repetitions of the vertices $x_1, \ldots, x_d$ of $S \cap T$. By standard multi-affine combinations a la de Casteljau we find that the above equation holds for all $y_i \in S \cap T$. We then can invoke Lemma 6.5 on the convex hull C of $S \cup T$ to get

$$M_P(z_1, \ldots, z_{k+\ell}, y_{k+\ell+1}, \ldots, y_n) = M_Q(z_1, \ldots, z_{k+\ell}, y_{k+\ell+1}, \ldots, y_n)$$

for all $z_1, \ldots, z_{k+\ell} \in C,\ k+\ell \le m,\ y_{k+\ell+1}, \ldots, y_n \in S \cap T$. Since we assumed the simplices S and T to be nondegenerate, this means that the above equations also hold for all $z_1, \ldots, z_{k+\ell} \in I\!R^d,\ k+\ell \le m,\ y_{k+\ell+1}, \ldots, y_n \in S \cap T$. This is the assertion of Lemma 6.4, and this finishes the proof of

Theorem 6.2 *Equations* (6.6) *are necessary and sufficient for a C^m transition of polynomials of degree n between nondegenerate simplices.* □

If we restrict everything to the affine hull A of $S \cap T$, the same statement holds even for degenerate simplices. One must restrict everything to A instead of working in $I\!R^d$, in particular the directional derivatives.

One can look at equations (6.6) from various directions. Assume that we have the control net b_j and want to calculate as many as possible of the c_j via the C^m continuity conditions. We then take $\ell = 0$ in (6.6) and calculate all

$$c_{ke_0+j} = b^{(n-k)}_{0e_0+j}(y_0),\ 0 \leq k \leq m,\ |j| = n-k,\ j_0 = 0$$

by m steps of the de Casteljau algorithm, starting out from $S \cap T$. For $m = 0$ we just have the C^0 condition

$$c_{0*} = b_{0*}$$

that ensures $P = Q$ on $S \cap T$. For $m = 1$ we see that

$$c_{1j} = b^{(n-1)}_{0j}(y_0),\ |j| = n-1$$

calculates the next layer of control points adjacent to $S \cap T$ by simple de Casteljau extrapolation towards y_0.

Sorry, no pictures so far...

Definition 6.1 *In view of* (6.6) *we call control points b_j of a representation of a polynomial of degree n over a simplex S to be in layer k, if $\min\limits_{1 \leq i \leq d} j_i = k$.*

This means that layer 0 consists of control points corresponding to the boundary faces, and layers up to m are relevant for C^m continuity conditions.

6.9 Continuity Conditions over Vertices

For simplicity, let us restrict ourselves to triangles. In the parameter domain, any number $\ell \geq 3$ of triangles can meet at a common vertex,

Sorry, no pictures so far...

and the corresponding ℓ polynomial surfaces of degree n (by degree elevation) should be not only smooth over the triangles' egdes, but also be smooth around the vertex. For C^0 continuity this is easy: the control points corresponding to triangle faces and to the vertex must simply be well–defined and independent of the triangle in question.

A C^1 transition at a vertex is somewhat more complicated. Each C^1 transition over the faces will affect the layer 1 control points of all surfaces in such a way that the local geometry of the parameter triangles is mapped affinely into image space $R^D,\ D = 3, 4$. At the vertex this must hold for all involved triangles simultaneously, meaning that the local "star" of triangles in the parameter domain must be mapped affinely into the tangent plane to the full C^1 surface at the image of the vertex in such a way that the control points of layers 0 and 1 adjacent to the vertex are mapped to the tangent plane by this single affine map.

Sorry, no pictures so far...

If the degree n is 3 or more, this construction will only affect the control points near the vertex and not affect control points near the other vertices. This means that it is always possible to design a control net locally around a vertex in such a way that a C^1 surface results, if the degree is 3 or more. One prescribes a tangent plane, maps the local "star" of triangles affinely onto it and adjusts the rest of the layer 0 control points suitably.

In the general C^m case one can try to prescribe all m–th order directional derivatives at the vertex to generate all neighbouring control points of layers 0 to m. These do not interfere with control points being neighbours to other vertices, if control points like $b_{n-2m,m,m}$ and $b_{m,n-2m,m}$ do not interfere, and this works for $n \geq 2m+1$.

Sorry, no pictures so far...

Things get more complicated if we also consider C^m conditions along all edges. The C^1 condition along edges will affect the twist vectors of type $b_{1,1,n-2}, {}_{1,n-2,1}$, $c_{1,1,n-2}, c_{1,n-2,1}$ etc. in a cyclic fashion around a vertex. Even if the associated conditions are solvable, these conditions will interfere with those coming in from other edges unless n is 4 or more. The situation for 4 adjacent rectangular polynomial surface patches and their "twist vectors" was treated in section 5.8, and we do not want to pursue this subject any further. The theses of Armin Iske (Glattes Verheften von Bernstein-Bézier-Tensorproduktflächen. NAM, Göttingen, Diplomarbeit, 1991) and Marko Weinrich (Glattes Verheften von Bernstein-Beziér-Dreiecksflächen. NAM, Göttingen, Diplomarbeit, 1991) contain a fairly complete presentation of the patching problem.

7 Rational Curves

The stuff in *italics* is currently omitted, because it is more orc less standard and can be found anywhere in the literature

7.1 Elementary Properties

Short discussion of writing curves polynomially in homogeneous coordinates versus writing them as rational functions with certain weights

Some easy examples showing how to represent simple conics that way.

Motivates that conics should be representable as rational quadratic curves.

7.2 Weighted Bernstein–Bézier Form

Simple stuff: convex hull property, derivatives

7.3 Conics

Here, we follow a journal article of Lee (The rational Bézier representation for conics in Geometric Modeling, *Geometric Modeling: Algorithms and New Trends*, SIAM, 1987, p. 3–19) in order to give a more or less complete account of planar conics.

Conics are planar curves that arise when intersecting the surface of a 3–dimensional cone with a hyperplane. We omit a proof of the fact that conics in cartesian 2D coordinates x, y coincide with the set of nontrivial solutions of quadratic equations

$$a_0 + a_1x + a_2y + a_3x^2 + a_4xy + a_5y^2 = 0,$$

but we remark that this definition provides 5 degrees of freedom, because there is a free common nonzero factor. Everybody should know the standard cases of

- ellipses
$$\frac{(x-x_0)^2}{a^2} + \frac{(y-y_0)^2}{b^2} = 1$$
- hyperbola
$$\frac{(x-x_0)^2}{a^2} - \frac{(y-y_0)^2}{b^2} = 1$$
- parabola
$$\begin{array}{rcl} \dfrac{(x-x_0)^2}{a^2} - \dfrac{y-y_0}{b} & = & 1 \\ \dfrac{x-x_0}{a} - \dfrac{(y-y_0)^2}{b^2} & = & 1 \end{array}$$

that apparently have 4 varying parameters, but crucially depend on cartesian coordinates and should allow a rotation as a fifth parameter. Our goal is to represent the full range of all types of conics by 5 parameters in a simple and effective way.

We first look at the standard weighted rational Bernstein–Bézier representation

$$R(t) = \frac{P(t)}{p(t)} = \frac{w_0b_0(1-t)^2 + w_1b_12t(1-t) + w_2b_2t^2}{w_0(1-t)^2 + w_12t(1-t) + w_2t^2}$$

of degree 2 with three (finite Euclidean) control points b_0, b_1, b_2. We can assume these points to lie in a plane. e.g. in $I\!R^2$, and we can assume them to be non–collinear, because otherwise we would just get a strangely parameterized straight line. From the preceding section we know that this fixes two points b_0, b_2 on the conic together with the tangents there, defined by the directions $b_1 - b_0, b_2 - b_1$. This makes four pieces of information, and it defines the standard local second–order polynomial Bernstein–Bézier curve, if we simply ignore the denominator by putting all weights equal to one. We would thus get the parabolic case, but there are formally three weights to play with. It will turn out that just one additional parameter is relevant, making it 5 in total, which is what we are after.

We note in passing that any conic can be constructed from 5 "pieces" by a completely "projective" construction using only a rule, not even a compass. We provide a single case, consisting of the two points b_0, b_2 and two tangents we have so far, plus an additional point b inside the triangle spanned by b_0, b_1, b_2. This makes 5 "pieces" that uniquely define a conic, and we construct additional points on the conic as follows:

1. Pick any line L through b_0. We want to construct the new point on that line.
2. Intersect L with the line bb_1 to get a new point c_1.

3. Intersect b_2c_1 with the line b_0b to get a new point c_2.

4. Intersect b_1c_2 with the line L to get the desired point c.

Though we did not prove the above strategy to be correct, it teaches us that fixing a single additional point within the triangle should fix the complete conic and determine its type (elliptic, parabolic, hyperbolic). In fact, it will turn out that fixing a point above or below the parabolic polynomial curve will yield a hyperbolic or elliptic conic, respectively.

We now want to proceed somewhat more strictly and introduce barycentric coordinates $\lambda_0, \lambda_1, \lambda_2$ within the triangle $b_0b_1b_2$ in the standard way. Then we compare

$$\begin{array}{rcccccc} R(t) & = & \frac{w_0(1-t)^2}{p(t)}b_0 & + & \frac{2w_1t(1-t)}{p(t)}b_1 & + & \frac{w_2t^2}{p(t)}b_2 \\ & = & \lambda_0 b_0 & + & (1-\lambda_0-\lambda_2)b_1 & + & \lambda_2 b_2 \end{array}$$

and get

$$\begin{array}{rcl} \lambda_0 & = & \frac{w_0(1-t)^2}{p(t)} \\ 1-\lambda_0-\lambda_2 & = & \frac{2w_1t(1-t)}{p(t)} \\ \lambda_2 & = & \frac{w_2t^2}{p(t)}. \end{array}$$

This combines into the quadratic equation

$$\lambda_0 \cdot \lambda_2 = \frac{w_0(1-t)^2}{p(t)}\frac{w_2t^2}{p(t)} = \frac{w_0w_2}{4w_1^2}(1-\lambda_0-\lambda_2)^2 = k(1-\lambda_0-\lambda_2)^2 \tag{7.1}$$

for the barycentric coordinates, depending only on the single scalar quantity

$$k := \frac{w_0w_2}{4w_1^2}.$$

Curves with the same value of k will coincide, and the case $k = 1/4$ yields the standard second–degree polynomial case that we expect to turn out to produce a parabola. For simplicity, we can set $w_0 = w_2 = 1$ whenever we want, controlling everything via k or w_1.

The denominator polynomial has the form

$$p(t) = (1-t)^2 + w_1 2t(1-t) + t^2 = (2-2w_1)t^2 + 2t(w_1-1) + 1.$$

If it is of degree smaller than 2, this means $w_1 = 1$ and $k = 1/4$, i.e. we are in the standard polynomial case. In any other case, we have the possibly complex roots

$$\frac{w_1 - 1 \pm \sqrt{(w_1-1)^2 - (2-2w_1)}}{2-2w_1} = \frac{w_1 - 1 \pm w_1\sqrt{1-4k}}{2-2w_1}$$

and see that there are

- no real roots for $k > 1/4$
- two real roots for $k < 1/4$

- just one real root for $k = 1/4$, but then the polynomial deteriorates into $p = 1$ and has no finite roots at all. If w_1 tends to one from above, we are in the above case and see that the two real roots both tend to infinity and coalesce there in the limit.

Of course we guess that the three cases above are the elliptic, hyperbolic, and parabolic case, respectively. Clearly, in the first case the whole curve, when seen for all $t \in I\!R$, stays in a bounded region. In the second case there are two curve parameters t_1 and t_2 where the denominator vanishes. If the numerator does not vanish for those parameters, the curve will have a singularity there.

We now change the sign of w_1 by brute force and define

$$\tilde{R}(t) = \frac{\tilde{P}(t)}{\tilde{p}(t)} = \frac{w_0 b_0 (1-t)^2 - w_1 b_1 2t(1-t) + w_2 b_2 t^2}{w_0(1-t)^2 - w_1 2t(1-t) + w_2 t^2}$$

Again, we compare

$$\begin{array}{rcccccc} \tilde{R}(t) & = & \dfrac{w_0(1-t)^2}{\tilde{p}(t)} b_0 & - & \dfrac{2w_1 t(1-t)}{\tilde{p}(t)} b_1 & + & \dfrac{w_2 t^2}{\tilde{p}(t)} b_2 \\ & = & \tilde{\lambda}_0 b_0 & + & (1-\tilde{\lambda}_0-\tilde{\lambda}_2) b_1 & + & \tilde{\lambda}_2 b_2 \end{array}$$

and get

$$\begin{array}{rcl} \tilde{\lambda}_0 & = & \dfrac{w_0(1-t)^2}{\tilde{p}(t)} \\ 1-\tilde{\lambda}_0-\tilde{\lambda}_2 & = & \dfrac{2w_1 t(1-t)}{\tilde{p}(t)} \\ \tilde{\lambda}_2 & = & \dfrac{w_2 t^2}{\tilde{p}(t)}. \end{array}$$

This combines into the quadratic equation

$$\tilde{\lambda}_0 \cdot \tilde{\lambda}_2 = \frac{w_0(1-t)^2}{\tilde{p}(t)} \frac{w_2 t^2}{\tilde{p}(t)} = \frac{w_0 w_2}{4w_1^2}(1-\tilde{\lambda}_0-\tilde{\lambda}_2)^2 = k(1-\tilde{\lambda}_0-\tilde{\lambda}_2)^2$$

for the barycentric coordinates, and this is the same equation that we already had. Thus the point $\tilde{R}(t)$ must lie somewhere on our curve, but where? We call it *complementary* to $R(t)$ and look at

$$\begin{array}{rcl} \tilde{p}(t)(\tilde{R}(t)-b_1) & = & \tilde{P}(t) - b_1\tilde{p}(t) \\ & = & P(t) - 4w_1 b_1 t(1-t) - b_1\tilde{p}(t) \\ & = & P(t) - b_1 p(t) \\ & = & p(t)(R(t)-b_1) \end{array}$$

to see that the three points $R(t), b_1$, and $\tilde{R}(t)$ lie on a line. There should be a parameter $\tilde{t}$ with $\tilde{R}(t) = R(\tilde{t})$, and by some lengthy calculation one gets

$$\tilde{t} = \frac{t}{2t-1}.$$

We should verify this by

$$\begin{aligned}
P(\tilde{t}) &= P\left(\frac{t}{2t-1}\right) \\
&= w_0b_0\left(1-\frac{t}{2t-1}\right)^2 + 2w_1b_1\left(\frac{t}{2t-1}\right)\left(1-\frac{t}{2t-1}\right) + w_2b_2\left(\frac{t}{2t-1}\right)^2 \\
&= \frac{w_0b_0(t-1)^2 + 2w_1b_1t(t-1) + w_2b_2t^2}{(2t-1)^2} \\
&= \frac{\tilde{P}(t)}{(2t-1)^2}.
\end{aligned}$$

Similarly, replacing all b_* by 1 for a moment, we get that also

$$p(\tilde{t}) = \frac{\tilde{p}(t)}{(2t-1)^2}$$

holds and implies

$$R(\tilde{t}) = \tilde{R}(t)$$

because the denominator $(2t-1)^2$ cancels.

But note now that the function $\tilde{t}(t) = t/(2t-1)$ is a bijection on $\overline{I\!R}$ with fixed points $t = 1$ and $t = 0$, mapping $t = 1/2$ to ∞, $(1/2, 1]$ to $[1, \infty)$ and $[0, 1/2)$ to $(-\infty, 0]$. This means that $\tilde{R}(t) = R(\tilde{t})$ lies on the curve segment outside of the triangle $b_0b_1b_2$ when t varies in $(0, 1)$.

The point $\tilde{b}_1 := \tilde{R}(1/2) = R(\infty)$ is something special. For the parabolic case it lies at infinity, but clearly not for the other cases, where it comes out to be

$$\tilde{b}_1 = R(\infty) = \frac{w_0b_0 - 2w_1b_1 + w_2b_2}{w_0 - 2w_1 + w_2}.$$

Strangely enough, the point $\tilde{b}_1 := \tilde{R}(1/2) = R(\infty)$ always lies on the line connecting b_1 with $(b_0+b_2)/2$. To see this, consider the intersection point q of the line b_0b_2 with the line connecting b_1 and $\tilde{b}_1$. It has the form

$$\begin{aligned}
q &= \lambda b_0 + (1-\lambda)b_2 \\
&= b_1 + \tau\left(\frac{w_0b_0 - 2w_1b_1 + w_2b_2}{w_0 - 2w_1 + w_2} - b_1\right)
\end{aligned}$$

and we see that b_0 and b_2 carry the same weight, which must then be 1/2. The analogous point s with

$$s := R(1/2) = \tilde{R}(\infty) = \frac{w_0b_0 + 2w_1b_1 + w_2b_2}{w_0 + 2w_1 + w_2}$$

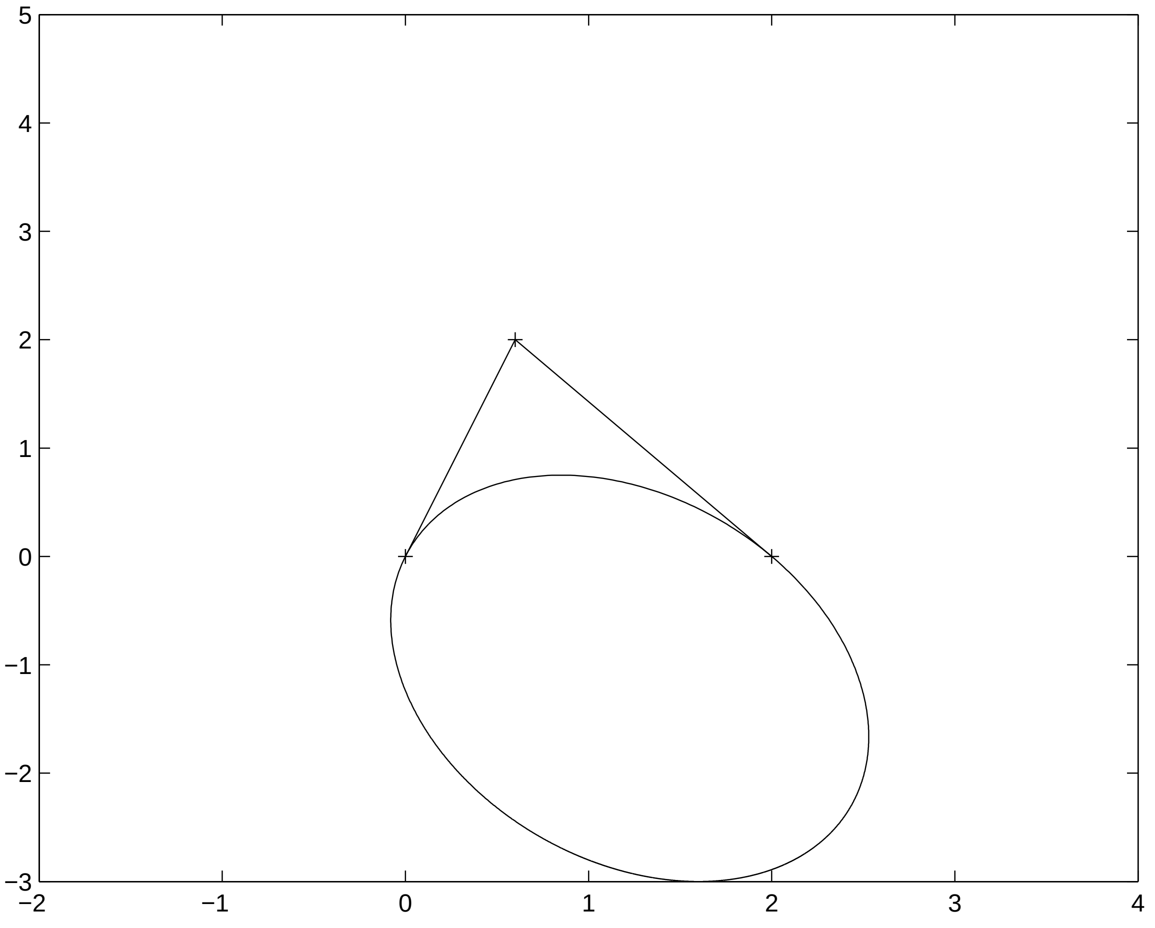
is called *shoulder point* and can come in useful as an additional point that controls the type of the conic.

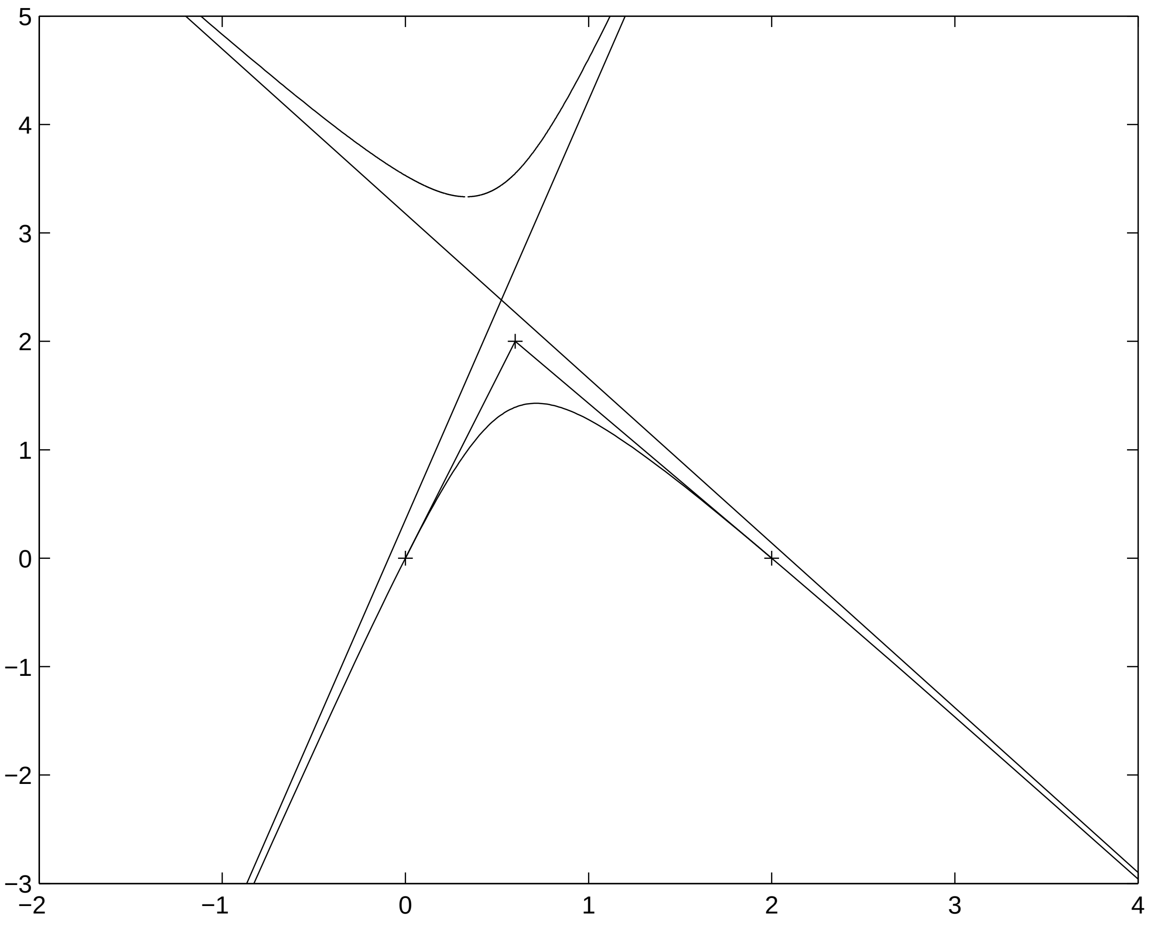
From this point on, one can calculate plenty of information about conics from the formulae we have up to now, i.e. axes, foci etc. We shall not do this now, but rather show that given an additional point S within the standard triangle we can identify a unique conic through this point. To this end, we have to find k and t such that $S = R(t)$ for the conic with parameter k.

But this is easy to do via the barycentric cordinates of S, We calculate k from (7.1). Then we set $w_0 = w_2 = 1$ to get w_1 from k and use the equations for the barycentric coordinates via

$$\sqrt{\frac{\lambda_2}{\lambda_0}} = \frac{t}{1-t}$$

to calculate t. The same argument shows that we can pick up any conic this way, once we know that conics are defined by two points with tangents and an additional point. We go to some place of the conic where it has no singularity, take two points and the tangents there, draw the standard triangle and select another point from those within the triangle. Then we perform what we did above, and we get the conic through these data.





These pictures were made by a simple MATLAB m-file:

```
% quadratic bezier curves
% points: [xi ; yi; wi]
p=[ 0 0.6 2 ; 0 2 0; 1 3 1]
% weighted points
pb=[p(1,:).*p(3,:) ;p(2,:).*p(3,:) ;p(3,:)]
% range for t from -100 to 100
rt=-100:0.01:100;
% Bernstein polynomial values over the full range
b0=(1-rt).^2;
b1=2*rt.*(1-rt);
b2=rt.^2;
% generate values over the range: [x; y; denominator]
pw=pb*[b0;b1;b2];
% this plots the upper two lines of the triangle
plot(p(1,:),p(2,:));
% this fixes the plot axes
axis([-2 4 -3 5]);
% hold axes scaling for subsequent plots
hold on
% plot the values of the function
```

```
plot(pw(1,:)./pw(3,:),pw(2,:)./pw(3,:));
hold on
% plot red crosses at the given control points
plot(p(1,:),p(2,:),'r+');
```

for varying values of the central weight, i.e. the penultimate value of the p array.

8 Transfinite Methods, Coons Patches

Standard material, in any book on CAGD, omitted

9 Constructing Surfaces from Point Clouds

The background for all of this is the problem of reconstructing geometric objects from given experimental data, e.g. from laser scans or satellite observations.

9.1 Constructing Curves from Point Data

Start: Standard material on interpolation, omitted

For interpolation of a sequence of data points $(t_j, y_j) \in I\!R^2,\ 0 \le j \le n$ one has both the order information of the points and the parametrization information. This is not available, if an unstructured set of n points in $I\!R^2$ or $I\!R^3$ is given. One needs to recover the order information first, which need not be implied by the numbering. and then find the "right" order of the points. This is a highly complicated problem, at least in theory, because it has apparent connections to the travelling salesman problem.

In what follows, we assume that we have an ordering $z_0, \ldots, z_n$ of the points. But usually there is no parametrization. A standard choice would be *chord–length* parametrization, assigning parameters

$$\begin{array}{rcll} t_0 & := & 0 & \text{to} \quad z_0 \\ t_j & := & t_{j-1} + \|z_j - z_{j-1}\|_2 & \text{to} \quad z_j,\ j \ge 1 \end{array}$$

that builds up the chord length of the piecewise linear interpolant. One then has vector–valued parametric data $(t_j, z_j),\ 0 \le j \le n$ and can use any standard interpolation scheme.

Another possibility is to estimate tangents at the data points, e.g. by differences of nearby points, and to use Hermite–type interpolants to construct an interpolation curve by a completely local process. This can be done along the lines of

`ftp://ftp.num.math.uni-goettingen.de/pub/preprints/schaback/OGHIoC.ps`

and details are give there. Also, one can do an interpolation by conics of varying type, see

`ftp://ftp.num.math.uni-goettingen.de/pub/preprints/schaback/PCIbPCoAT.ps.gz`

If planar interpolating curves are constructed by *implicit* representations, things are simpler. The basic idea is as follows. If the curve can be written as the set S_g of points $z \in I\!R^2$ satisfying an equation like $g(z) = 1$, one can try to approximate g by an interpolant s_g that attains the value 1 at all data points z_j and consider the set S_g of points z with $s_g(z) = 1$. Since $s_g(z_j) = 1 = g(z_j)$ holds, the set S_g contains all the data points z_j, and it is a curve, if the function s_g is nondegenerate. One way of constructing such a curve is to solve the linear system

$$\sum_{j=0}^{n} \alpha_j \exp(-\gamma \|z_j - z_k\|_2^2) = 1,\ 1 \leq k \leq n$$

for a fixed positive value of γ, and then setting

$$s_g(z) = \sum_{j=0}^{n} \alpha_j \exp(-\gamma \|z_j - z\|_2^2).$$

This technique does not need ordered data, but it yields an implicit representation instead of an explicit one. And, it needs proof that the symmetric $n \times n$ matrix with entries $\exp(-\gamma\|z_j - z_k\|_2^2)$ is nonsingular for any choice of n different points in $\mathbb{R}^2$. This is indeed true, and it holds even for all space dimensions. But there still is another drawback: it is not clear whether the set S_g splits into several disjoint connected components, failing to line the given points up in a single curve.

9.2 Constructing Surfaces from Point Data