Lecture 24: Blending Functions: An Alternative Approach to Freeform Curves and Surfaces

He made also bases 2 Chronicles 4:14

1. Blending Functions

Till now we have taken an algorithmic approach to freeform curves and surfaces: Neville's algorithm for Lagrange interpolation, de Casteljau's algorithm for Bezier approximation. In this lecture we are going to derive explicit formulas for these curve and surface schemes.

Both Neville's algorithm and de Casteljau's algorithm have the generic triangular structure depicted in Figure 1: the control points are placed at the base of the triangle, and points on the curve emerge at the apex of the triangle. A simple inductive argument shows that the curve that emerges at the apex is an affine combination of the control points at the base. Thus if $D(t)$ is the curve that emerges at the apex of the triangle with the control points $P_0,\ldots,P_n$, at the base, then there are functions $D_0(t),\ldots,D_n(t)$ such that

$$D(t) = \sum_k D_k(t)P_k . \qquad (1.1)$$

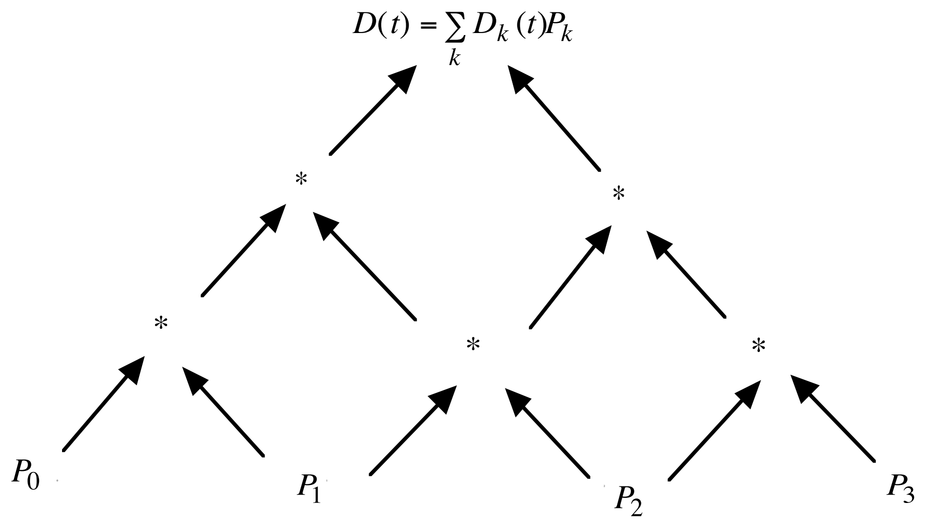

Figure 1: The generic up recurrence for evaluating points on a curve. The control points $P_0,\ldots,P_n$ are placed at the base of the diagram and points on the curve $D(t) = \sum_k D_k(t)P_k$ emerge at the apex of the triangle. The functions $D_0(t),\ldots,D_n(t)$ that multiply the control points are the blending functions for the curve and depend on the labels along the arrows. To keep the algorithm generic, we have not specified these labels along the arrows. Here we illustrate the cubic case.

The functions $D_0(t),\ldots,D_n(t)$ that multiply the control points are called *blending functions* because these functions blend together the control points to form a smooth curve. These blending functions depend only on the labels along the arrows and not on the particular control points. All the properties of a particular curve scheme can be derived by studying the properties of the associated blending functions. In this lecture we will derive the blending functions for the Lagrange and Bezier schemes, and we will show how the geometric properties of the corresponding curve and surface schemes depend on the algebraic properties of their associated blending functions.

There are several ways to compute the blending functions $D_0(t),\ldots,D_n(t)$. Since the function $D_k(t)$ is the coefficient of the control point P_k.

$D_k(t)$ = *the sum over all paths from* P_k *at the base to* $D(t)$ *at the apex of the triangle. (1.2)*

where a *path* means the product of the labels on all the arrows along the path. Though at first glance Equation (1.2) may appear daunting, in Section 2 we shall show how to apply this equation to compute explicit formulas for the blending functions of Lagrange and Bezier curves.

Perhaps more naturally we can also compute $D_k(t)$ directly from the evaluation algorithm for $D(t)$ by setting $P_k = 1$ and $P_j = 0$ for $j \neq k$, since in this case by Equation (1.1)

$$D(t) = \textstyle\sum_j D_j(t)P_j = D_k(t)\,.$$

We illustrate this algorithm in Figure 2.

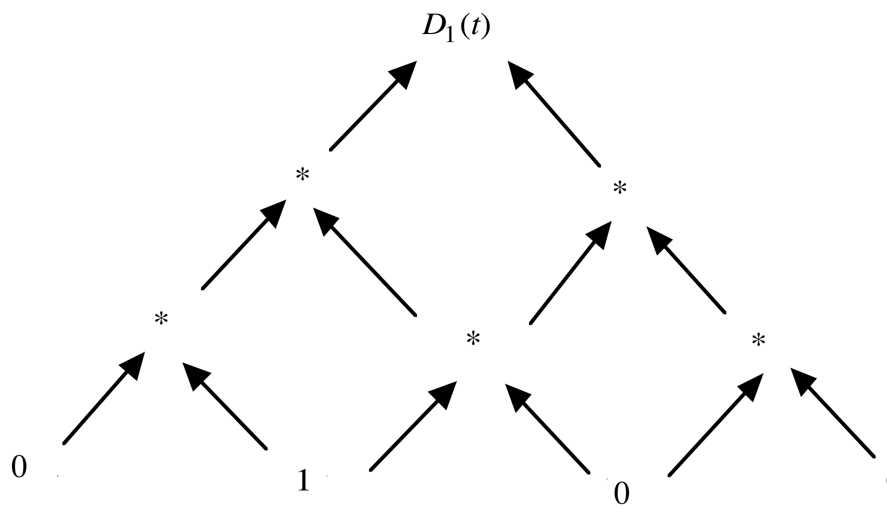

Figure 2: The up recurrence for computing one of the blending functions. A one is placed at the *kth* position and zeros are placed everywhere else at the base of the triangle. Since $D_k(t)$ is the sum over all paths from P_k at the base to $D(t)$ at the apex of the triangle, the function $D_k(t)$ emerges at the apex of the triangle. Here we illustrate the algorithm for computing $D_1(t)$ in the cubic case.

Alternatively, we can compute the blending functions by observing that the sum over all paths from the *kth* position at the base to the apex of the triangle is the same as the sum of all paths from the apex to the *kth* position at the base of the triangle. Thus

$$D_k(t) = \textit{the sum over all paths from the apex to the kth position at the base of the triangle.} \tag{1.3}$$

Therefore we can compute all the blending functions simultaneously by placing a 1 at the apex of the triangle and running the recurrence down instead of up by reversing all the arrows in the diagram. Since the *kth* blending function is the sum of all paths from the apex to the *kth* position at the base, all the blending functions emerge at the base of the triangle (see Figure 3).

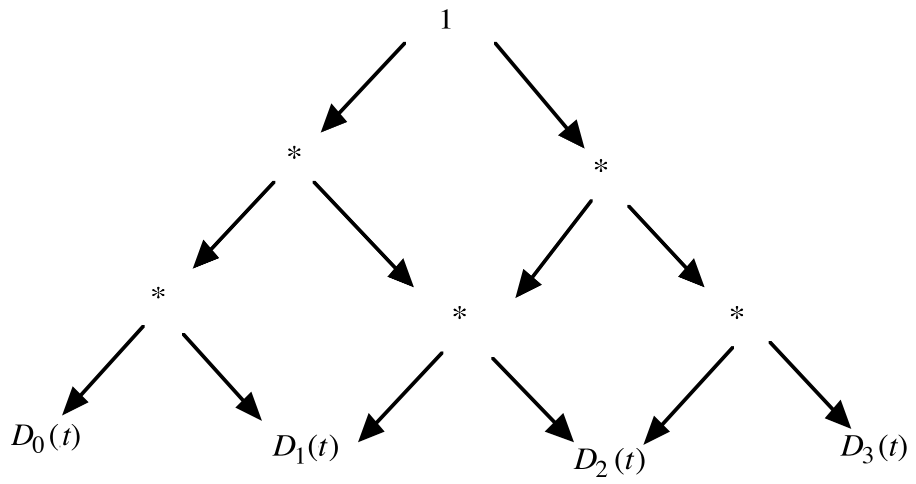

Figure 3: The down recurrence for the blending functions. Place 1 at the apex of the triangle and run the evaluation algorithm in reverse by reversing all the arrows. The blending functions $D_0(t),\ldots,D_n(t)$ now emerge at the base of the triangle. Again we illustrate the cubic case.

The preceding algorithms work generically for any recursive curve scheme. With these general ideas in hand, we are now ready to find explicit formulas for the blending functions of Lagrange and Bezier curves and surfaces.

2. The Lagrange and Bernstein Basis Functions

Since both Lagrange and Bezier curves and surfaces are polynomial curves and surfaces, the blending functions for these schemes are polynomial functions. For a fixed degree *n*, these polynomial blending functions turn out to be polynomial bases -- they are linearly independent and they span the space of all polynomials of degree *n*. The blending functions for Lagrange interpolation are called the *Lagrange basis functions,* and the blending functions for Bezier approximation are called the *Bernstein basis functions*. Here we shall derive explicit formulas for each of these sets of basis functions.

2.1 Lagrange Basis Functions. Before we can derive explicit formulas for the Lagrange basis functions, we need to fix our notation. Let $L_k^n(t \mid t_0,\ldots,t_n)$ denote the *kth* Lagrange basis function of degree n for the nodes $t_0,\ldots,t_n$. (Recall that the nodes $t_0,\ldots,t_n$ are the values of t where the interpolation occurs.) Since the nodes $t_0,\ldots,t_n$ are fixed, the Lagrange basis functions $L_k^n(t \mid t_0,\ldots,t_n)$, $k = 0,\ldots,n$, are polynomial functions in the parameter *t*; the nodes are simply there to remind us that if we change the nodes, the functions $L_k^n(t \mid t_0,\ldots,t_n)$ will also change.

We can use Equation (1.3) together with Neville's algorithm to derive an explicit formula for $L_k^n(t \mid t_0,\ldots,t_n)$. By Equation (1.3) $L_k^n(t \mid t_0,\ldots,t_n)$ is the sum over all paths from the apex to the *kth* position at the base of the Neville's algorithm. Now look at Neville's algorithm in Figure 4. By the parallel property, every path from the apex to the *kth* position at the base is the same! Actually the paths are only the same up to a constant multiple because the labels in the diagram are not normalized. To get from the apex to the base, we must make exactly *k* right turns and $n-k$ left turns. The first *k* right turns always have the same labels in the numerator as as the first *k* labels along the right lateral edge: $t-t_0,\ldots,t-t_{k-1}$. Similarly, the first $n-k$ left turns always have the same labels in the numerator as the first $n-k$ labels along the left lateral edge: $t_n,-t\ldots,t_{k+1}-t$. Thus up to a constant multiple, $L_k^n(t \mid t_0,\ldots,t_n)$ is just the product of these linear factors. That is, there is a constant c_k^n such that

$$L_k^n(t \mid t_0,\ldots,t_n) = c_k^n \times \prod_{j \neq k}(t-t_j).$$

Notice in particular that

$$L_i^n(t_j \mid t_0,\ldots,t_n) = 0 \qquad j \neq i.$$

To find the constant c_k^n, recall that Neville's algorithm interpolates the control point P_k at the node t_k. Therefore

$$P_k = P_{0\cdots n}(t_k) = \sum_{i=0}^{n} L_i^n(t_k \mid t_0,\ldots,t_n)P_i = \left\{c_k^n \times \prod_{j \neq k}(t_k-t_j)\right\}P_k$$

Comparing the coefficients of P_k, we conclude that

$$c_k^n = \frac{1}{\prod_{j \neq k}(t_k-t_j)},$$

so

$$L_k^n(t \mid t_0,\ldots,t_n) = \frac{\prod_{j \neq k}(t-t_j)}{\prod_{j \neq k}(t_k-t_j)} \qquad k = 0,\ldots,n. \tag{2.1}$$

Thus the Lagrange basis function $L_k^n(t \mid t_0,\ldots,t_n)$ is quite straightforward to construct: in the numerator omit the factor $t-t_k$, in the denominator omit the factor $t_k - t_k$ Below we provide an explicit example of the cubic Lagrange basis functions with the nodes at the integers.

$$\begin{tikzcd}
 & & & 1 \arrow[dl, "t_3 - t"'] \arrow[dr, "t - t_0"] & & & \\
 & & * \arrow[dl, "t_2 - t"'] \arrow[dr, "t - t_0"] & & * \arrow[dl, "t_3 - t"'] \arrow[dr, "t - t_1"] & & \\
 & * \arrow[dl, "t_1 - t"'] \arrow[dr, "t - t_0"] & & * \arrow[dl, "t_2 - t"'] \arrow[dr, "t - t_1"] & & * \arrow[dl, "t_3 - t"'] \arrow[dr, "t - t_2"] & \\
L_0^3(t) & & L_1^3(t) & & L_2^3(t) & & L_3^3(t)
\end{tikzcd}$$

Figure 4: The down recurrence for Neville's algorithm, the cubic case. Notice that up to constant multiples, all paths from the apex to the *kth* position at the base are identical.

Example 1: *Cubic Lagrange Basis Functions with Nodes at the Integers* $0,1,2,3$.

$$L_0^3(t) = -\frac{(t-1)(t-2)(t-3)}{6} \qquad L_1^3(t) = \frac{t(t-2)(t-3)}{2}$$

$$L_2^3(t) = -\frac{t(t-1)(t-3)}{2} \qquad L_3^3(t) = \frac{t(t-1)(t-2)}{6}$$

These four functions are easy to construct. In the numerator of $L_k^3(t)$, we omit the factor $t-k$, and the denominator is chosen so that $L_k^3(t_k) = 1$.

By Equation (2.1) the Lagrange basis functions satisfy

$$\begin{aligned} L_k^n(t_j \mid t_0,\ldots,t_n) &= 0 \qquad j \neq k \\ &= 1 \qquad j = k. \end{aligned} \tag{2.2}$$

These conditions are called the *cardinal conditions.* Any functions that satisfy the cardinal conditions can be used in interpolation, since

$$P_{0\cdots n}(t_k) = \sum_{i=0}^{n} L_i^n(t_k \mid t_0,\ldots,t_n) P_i = P_k .$$

The Lagrange basis functions are the unique polynomials of degree n that satisfy the cardinal conditions because if two polynomials of degree n agree at $n+1$ values, they must be identical.

Notice then that the Lagrange basis functions satisfy an interpolation problem: the interpolation problem specified by the cardinal conditions. Thus we could have constructed the Lagrange basis functions directly from the cardinal conditions. The conditions

$$L_k^n(t_j \mid t_0,\ldots,t_n) = 0 \qquad j \neq k$$

tell us that t_j, $j \neq k$, are roots of $L_k^n(t \mid t_0,\ldots,t_n)$, so $t - t_j$, $j \neq k$ are linear factors. These linear factors provide the numerator of $L_k^n(t \mid t_0,\ldots,t_n)$. To insure that $L_k^n(t_k \mid t_0,\ldots,t_n) = 1$, the denominator is just the numerator evaluated at $t = t_k$.

Neville's algorithm is an $O(n^2)$ evaluation algorithm for the polynomial interpolant $P_{0\cdots n}(t)$. There is, however, an $O(n)$ evaluation algorithm due to Joe Warren that takes advantage of the special structure of the Lagrange basis functions. Whereas Neville's algorithm lies on a triangle, Warren's algorithm lies on a ladder, so we shall call it the *ladder algorithm* (see Figure 5).

Label the arrows on the left side of the ladder with the functions $t - t_0,\ldots,t - t_{n-1}$, and on the right side with the functions $t - t_1,\ldots,t - t_n$. These are the linear functions that appear as factors in the Lagrange basis functions. Label the rungs with the constant values

$$Q_k = \frac{P_k}{\prod_{j\neq k}(t_k - t_j)} \qquad k = 0,\ldots,n ,$$

where P_k is the *kth* control points and $\prod_{j\neq k}(t_k - t_j)$ is the denominator of the *kth* Lagrange basis functions. Place a one at the top of the ladder and run the algorithm by following the arrows. Values at any node in the ladder are computed in the usual way by multiplying the labels on the arrows that enter the node by the values at the nodes from which the arrows emerge and adding the results. Tracing through the paths in the ladder, it is easy to see that

$$P_{0\cdots n}(t) = \sum_{k=0}^{n} L_k^n(t \mid t_0,\ldots,t_n) P_k$$

indeed emerges at the bottom of the ladder. This ladder algorithm uses only $3n+1$ multiplications for each coordinate function -- n multiplications along each side of the ladder and $n+1$ along the rungs.

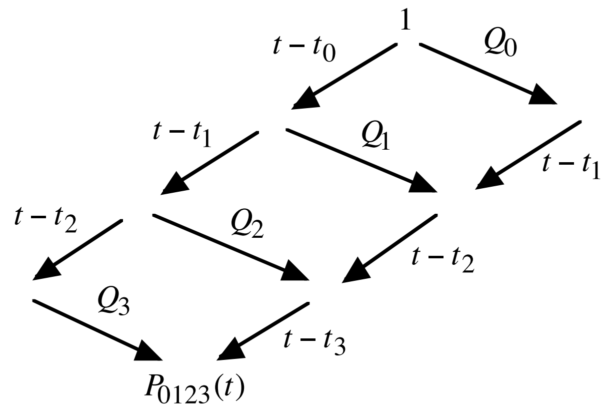

Figure 5: The ladder algorithm for the cubic Lagrange interpolant $P_{0123}(t)$.

2.2 Bernstein Basis Functions. To derive the blending functions for Bezier curves, we can proceed in a manner similar to our derivation of the Lagrange basis functions. However, the derivation will be simpler because there are no nodes to worry about in the case of Bezier curves. We shall use the notation $B_k^n(t)$ to denote the *kth* Bezier blending function of degree *n*.

For Bezier curves we can apply Equation (1.3) together with de Casteljau's algorithm to derive an explicit formula for $B_k^n(t)$. By Equation (1.3) $B_k^n(t)$ is the sum over all paths from the apex to the *kth* position at the base of the de Casteljau's algorithm (Figure 6). But in de Casteljau's algorithm, every path from the apex to the *kth* position at the base is exactly the same! To get from the apex to the base, we must make exactly k right turns and $n-k$ left turns. Every right has the label t; every left turn has the label $1-t$. Thus

$$B_k^n(t) = P(n,k) \times t^k (1-t)^{n-k},$$

where

$$P(n,k) = \#\text{ paths from the apex to the } kth \text{ position on the } nth \text{ level.}$$

To count the number of paths $P(n,k)$, observe that there are only two ways to reach the *kth* position on the *nth* level: either pass through the $(k-1)^{st}$ position on the $(n-1)^{st}$ level or pass through the *kth* position on the $(n-1)^{st}$ level (see Figure 7). Therefore

. $$P(n,k) = P(n-1,k-1) + P(n-1,k) \qquad P(0,0) = 1.$$

But this recurrence is identical to the recurrence satisfied by the binomial coefficients:

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k} \qquad \binom{0}{0} = 1.$$

Hence we conclude that

$$P(n,k) = \binom{n}{k}, \qquad (2.3)$$

so

$$B_k^n(t) = \binom{n}{k} t^k (1-t)^{n-k} \quad k = 0,\ldots,n.$$

Notice from the down recurrence that these functions satisfy the recurrence

$$B_k^{n+1}(t) = (1-t)\, B_{k-1}^n(t) + t\, B_k^n(t) \tag{2.4}$$

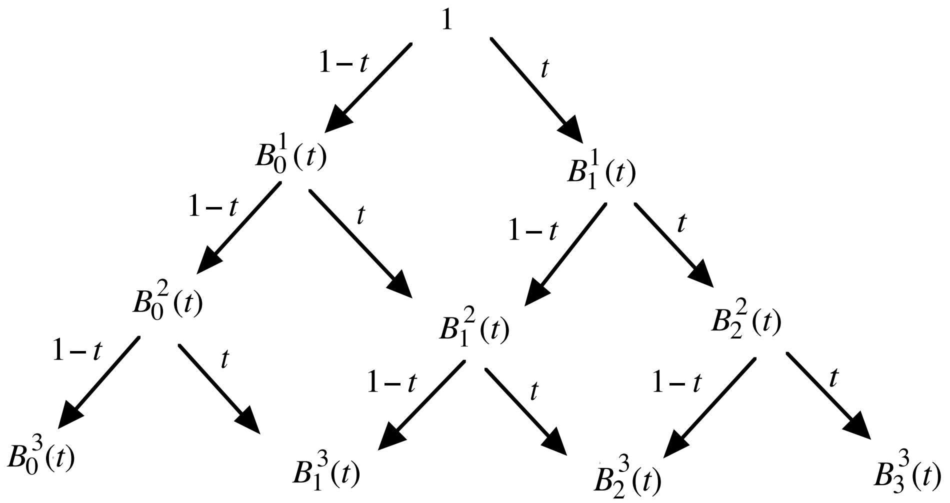

Figure 6: The the down recurrence for de Casteljau's algorithm, the cubic case. Notice that all paths from the apex to the *kth* position at the base are identical.

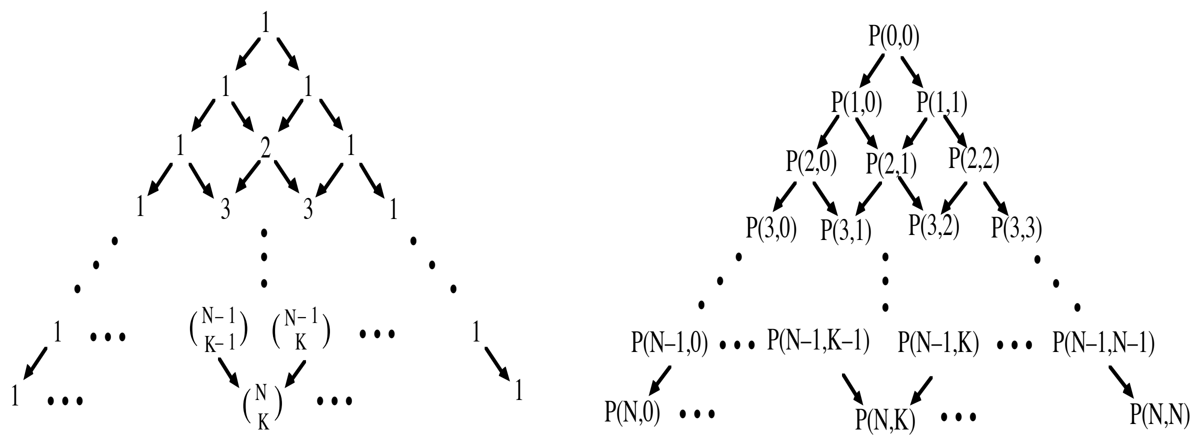
1
1 1
1 2 1
1 3 3 1
1 ••• $\binom{N-1}{K-1}$ $\binom{N-1}{K}$ ••• 1
1 ••• $\binom{N}{K}$ ••• 1
P(0,0)
P(1,0) P(1,1)
P(2,0) P(2,1) P(2,2)
P(3,0) P(3,1) P(3,2) P(3,3)
P(N−1,0) ••• P(N−1,K−1) P(N−1,K) ••• P(N−1,N−1)
P(N,0) ••• P(N,K) ••• P(N,N)

Figure 7: The number of paths in de Casteljau's triangle is given by Pascal's triangle because the number of paths and the binomial coefficients satisfy the same simple recurrence.

The functions $B_k^n(t)$ are called the *Bernstein basis functions* because they were used by Bernstein in his study of approximation theory. These Bernstein basis functions should look familiar because they represent the binomial distribution. We shall have more to say about the connection between probability theory and blending functions in Section 3.

The de Casteljau algorithm is an $O(n^2)$ evaluation algorithm for Bezier curves. We can now take advantage of the special structure of the Bernstein basis functions to generate an $O(n)$ evaluation algorithm similar to the ladder algorithm we constructed in Section 2.1 for Lagrange interpolation (see Figure 5).

Label the arrows on the left side of the ladder with the function t, and on the right side with the function $1-t$. These are the linear functions that appear as factors in the Bernstein basis functions. Label the rungs with the constant values

$$Q_k = \binom{n}{k} P_k \qquad k = 0,\ldots,n,$$

where P_k is the *kth* Bezier control point. Place a one at the top of the ladder and run the ladder algorithm in the usual manner (see Figure 8). Tracing through the paths in the ladder, it is easy to see that the Bezier curve

$$P(t) = \sum_{k=0}^{n} B_k^n(t) P_k$$

indeed emerges at the bottom of the ladder. This ladder algorithm again uses only $3n+1$ multiplications for each coordinate function -- n multiplications along each side of the ladder and $n+1$ along the rungs. A slight more efficient $O(n)$ evaluation algorithm for Bezier curves based on Horner's method is provided in the exercises (see Exercise 5).

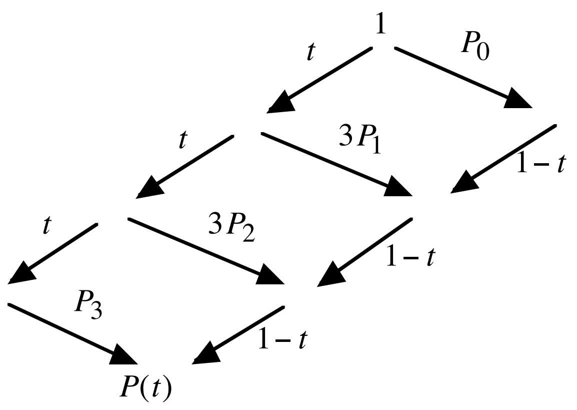

Figure 8: The ladder algorithm for the cubic Bezier curve $P(t)$ with control points P_k.

2.3 Basis Functions. For both the Lagrange and the Bezier blending functions, we have been using the term *basis functions*. A basis for a vector space is a collection of vectors that are linearly independent and span the space. The dimension of a vector space is the number of vectors in a basis. All bases have the same number of vectors.

The polynomials of degree n in t form a vector space of dimension $n+1$ because the monomials $1, t, \ldots, t^n$ are clearly a basis for the polynomials of degree *n*. We shall now show that the Lagrange basis functions $L_0^n(t \mid t_0, \ldots, t_n), \ldots, L_n^n(t \mid t_0, \ldots, t_n)$ and the Bernstein basis functions $B_0^n(t), \ldots, B_n^n(t)$ also form polynomial bases, so the term basis functions is appropriate in both settings.

There are two ways to prove that these particular sets of functions form polynomial basis. First we know that these two sets of functions both span the space of all polynomials of degree *n* because we showed that:

i. Lagrange interpolation reproduces polynomials (Lecture 20).
ii. Every polynomial can be represented in Bezier form (Lecture 23).

By the general theory of vector spaces, any collection of $n+1$ vectors that span an $(n+1)$-dimensional vector space are necessarily linearly independent and therefore form a basis. Alternatively, we can use the special properties of these particular basis functions to prove directly that they are linearly independent.

Proposition 1: *The Lagrange basis functions are linearly independent.*
Proof; Suppose that

$$\textstyle\sum_i c_i L_i^n(t \mid t_0, \ldots, t_n) \equiv 0. \tag{2.5}$$

We need to show that $c_k = 0$, $k = 0, \ldots, n$. Substituting $t = t_k$ into Equation (2.5) and applying the cardinal conditions (Equation 2.2), we find that

$$c_k = \textstyle\sum_i c_i L_i^n(t_k \mid t_0, \ldots, t_n) \equiv 0 \qquad k = 0, \ldots, n.$$

Proposition 2: *The Bernstein basis functions are linearly independent.*
Proof; Suppose that

$$\textstyle\sum_i c_i B_i^n(t) \equiv 0. \tag{2.6}$$

Again we need to show that $c_k = 0$, $k = 0, \ldots, n$. Dividing Equation (2.6) by $(1-t)^n$ and substituting $u = t/(1-t)$ yields

$$0 \equiv \frac{\sum_i c_i B_i^n(t)}{(1-t)^n} = \frac{\sum_i c_i \binom{n}{k} t^k (1-t)^{n-k}}{(1-t)^n} = \frac{\sum_i c_i \binom{n}{k} t^k}{(1-t)^k} = \textstyle\sum_i c_i \binom{n}{k} u^k.$$

But the functions $1, u, \ldots, u^n$ are certainly linearly independent. Therefore $c_k = 0$, $k = 0, \ldots, n$.

From linear independence it follows easily that the Bezier or Lagrange control points of a polynomial curve are unique. Below we give the proof for Bezier curves; the proof for Lagrange polynomials is virtually identical.

Corollary 3: *The Bezier control points of a polynomial are unique. That is, if two sets of control points* $Q_0,\ldots Q_n$ *and* $R_0,\ldots R_n$ *represent the same Bezier curve, then* $P_k = Q_k$, $k = 0,\ldots,n$.

Proof; If two sets of control points $Q_0,\ldots Q_n$ and $R_0,\ldots R_n$ represent the same Bezier curve, then

$$\sum_{k=0}^{n} B_k^n(t) P_k = \sum_{k=0}^{n} B_k^n(t) Q_k$$

Therefore

$$\sum_{k=0}^{n} B_k^n(t)(P_k - Q_k) = 0 .$$

Hence by the linear independence of the Bernstein basis functions $P_k = Q_k$, $k = 0,\ldots,n$.

3. Corresponding Properties of Curve Schemes and Blending Functions

Since curve schemes are completely determined by their blending functions, we can understand the properties of curve schemes by investigating the properties of their blending functions. In Table 1, we list some general correspondences between a curve scheme $\sum_k P_k D_k^n(t)$ and its blending functions $\{D_k^n(t)\}$.

Curve Scheme	*Blending Functions*
Polynomial	Polynomial
Up Recurrence	Down Recurrence
Translation Invariance	$\sum_k D_k^n(t) \equiv 1$
Lie in Convex Hull	$\sum_k D_k^n(t) \equiv 1$
of their Control Points	$D_k^n(t) \geq 0, \quad k = 0,\ldots,n$
Interpolates P_k at t_k	$D_j^n(t_k) = 0 \quad j \neq k$ $= 1 \quad j = k$
Non-Degenerate	Linearly Independent

Table 1: Correspondences between the geometric properties of a curve scheme and the algebraic properties of the blending functions.

Most of these correspondences are easy to understand. Clearly a curve scheme is a polynomial scheme if and only if its blending functions are polynomials. Also we observed in Section 2 that when a curve scheme has an up recurrence, the blending functions have a down recurrence. By a paths argument, the converse must also be true -- that is, if the blending functions have a down recurrence, then the curve scheme has an up recurrence (see Exercise 3).

We shall now look more carefully at the other properties in Table 1. Additional properties relating Bezier curves and Bernstein basis functions are provided in Exercises 4-9.

Recall that a curve scheme is said to be translation invariant if translating the control points by a vector v translates every point on the curve by the vector v -- that is, if for every vector v

$$D(t) + v = \sum_{k=0}^{n} D_k^n(t)(P_k + v) = D(t) + \sum_{k=0}^{n} D_k^n(t)v .$$

Thus a curve scheme is translation invariant if and only if the blending functions sum to one.

Both the Lagrange and Bezier schemes are translation invariant. We can see directly in each case that the blending functions sum to one. For the Lagrange basis functions

$$\sum_{k=0}^{n} L_k^n(t \mid t_0,\ldots,t_n) \equiv 1$$

by the uniqueness of polynomial interpolation. For the Bernstein basis functions, we can apply the binomial theorem:

$$\sum_{k=0}^{n} B_k^n(t) = \sum_{k=0}^{n} \binom{n}{k} t^k (1-t)^{n-k} = \big((1-t)+t\big)^n = 1^n = 1 .$$

The convex hull of two points is the line segment joining the two points. Thus

$$Convex\,Hull(P_0,P_1) = \{c_0P_0 + c_1P_1 \mid c_0 + c_1 = 1 \;\; and \;\; c_0,c_1 \ge 0\} .$$

More generally, it follows by induction (see Exercise 1) that for a collection of $n+1$ points, the convex hull is given by

$$Convex\,Hull(P_0,\ldots,P_n) = \left\{ \sum_{k=0}^{n} c_kP_k \mid \sum_{k=0}^{n} c_k = 1 \;\; and \;\; c_k \ge 0 \right\} .$$

Thus a curve $D(t) = \sum_k P_k D_k^n(t)$ will always lie in the convex hull of its control points if and only if

$$\begin{aligned} &\textstyle\sum_k D_k^n(t) \equiv 1 \\ &D_k^n(t) \ge 0, \quad k = 0,\ldots,n. \end{aligned} \tag{3.1}$$

Notice that the two properties in Equation (3.1) are the defining characteristics of discrete probability distributions, so if we want to construct curve schemes that lie in the convex hull of their control points, we must choose probability distributions for their blending functions. The Bernstein basis functions clearly satisfy Equation (3.1) over the interval $[0,1]$, so Bezier curves lie in the convex hull of their control points. In fact, as we observed in Section 2, the Bernstein basis functions represent the binomial distribution, so we see now that this seeming coincidence is no accident. On the other hand the Lagrange basis functions are not always non negative in their domain, so Lagrange polynomials are not confined to the convex hull of their control points.

Interpolation occurs at a node $t = t_j$ if for every choice of control points

$$D(t_j) = \sum_k P_k D_k^n(t_j) \equiv P_j .$$

Thus interpolation will always occur at $t = t_j$ if and only if

$$\begin{aligned} D_k^n(t_j) &= 0 \qquad k \neq j \\ &= 1 \qquad k = j . \end{aligned}$$

The Lagrange basis functions satisfy this constraint at the nodes, since these conditions are the cardinal conditions. The Bernstein basis functions satisfy these constraints only at $t - 0,1$, where it is easy to verify that

$$\begin{aligned} B_j^n(0) &= 0 \quad j \neq 0 & \qquad\qquad B_j^n(1) &= 0 \quad j \neq n \\ &= 1 \quad j = 0 & &= 1 \quad j = n \end{aligned}$$

Since zero and one are the only roots of the Bernstein basis functions, Bezier curves are guaranteed to interpolate only their first and last control points.

Finally a curve scheme is said to be *non-degenerate* if it collapses to a single point only if all the control points are located at that point. If a curve scheme is translation invariant, then

$$\sum_k D_k^n(t) \equiv 1 \Rightarrow P = \sum_k P D_k^n(t) .$$

Now the blending functions are linearly independent if and only if

$$\sum_k P_k D_k^n(t) = \sum_k Q_k D_k^n(t) \Leftrightarrow P_k = Q_k, \quad k = 0, \ldots, n .$$

Thus a translation invariant curve scheme is non-degenerate if and only if the blending functions are linearly independent. It follows by the discussion in Section 2.3, the both Bezier curves and Lagrange polynomials are non-degenerate.

4. Tensor Product Surfaces

For every curve scheme defined by an up recurrence, there is a corresponding tensor product surface scheme defined by an up recurrence. In fact, with the up recurrence for the curve scheme in Figure 1, there are actually associated two up recurrences -- Figure 9 and Figure 10 -- for the corresponding tensor product surface scheme. We have already encountered the tensor product Lagrange surfaces in Lecture 20 and the tensor product Bezier patches in Lecture 21.

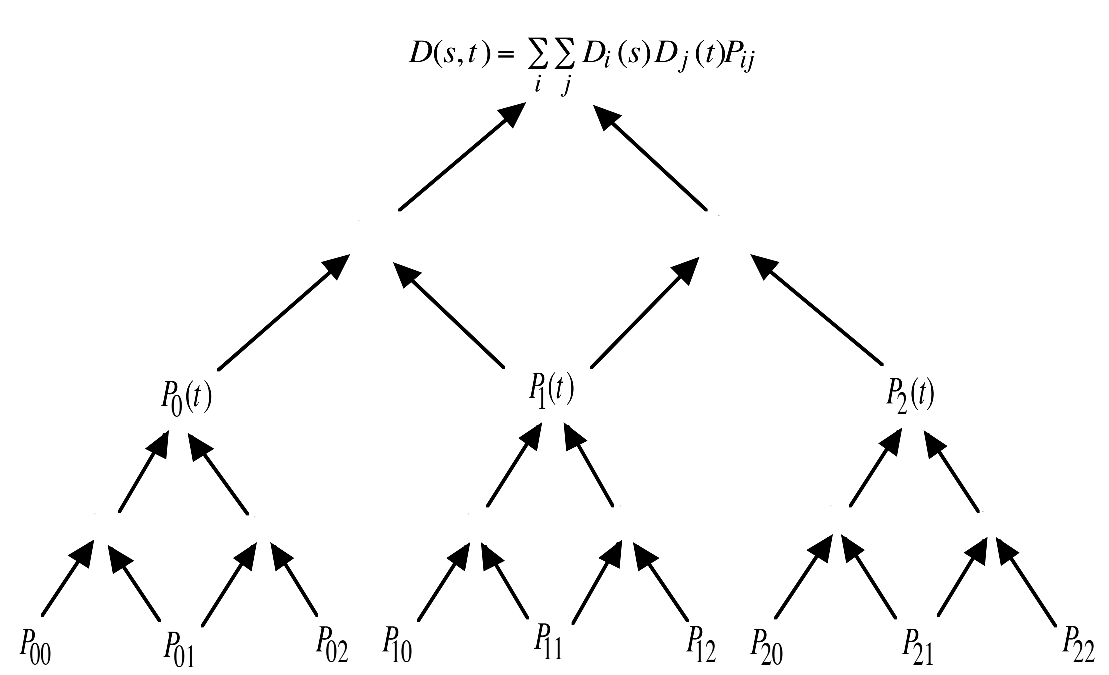

Figure 9: An up recurrence for a tensor product surface. By construction, the coefficient of P_{ij} in $P_i(t)$ is $D_j^n(t)$ and is the coefficient of $P_i(t)$ in $D(s,t)$ is $D_i^m(s)$. Hence $D_i^m(s)D_j^n(t)$ is the coefficient of P_{ij} in $D(s,t)$. Compare to Figure 10.

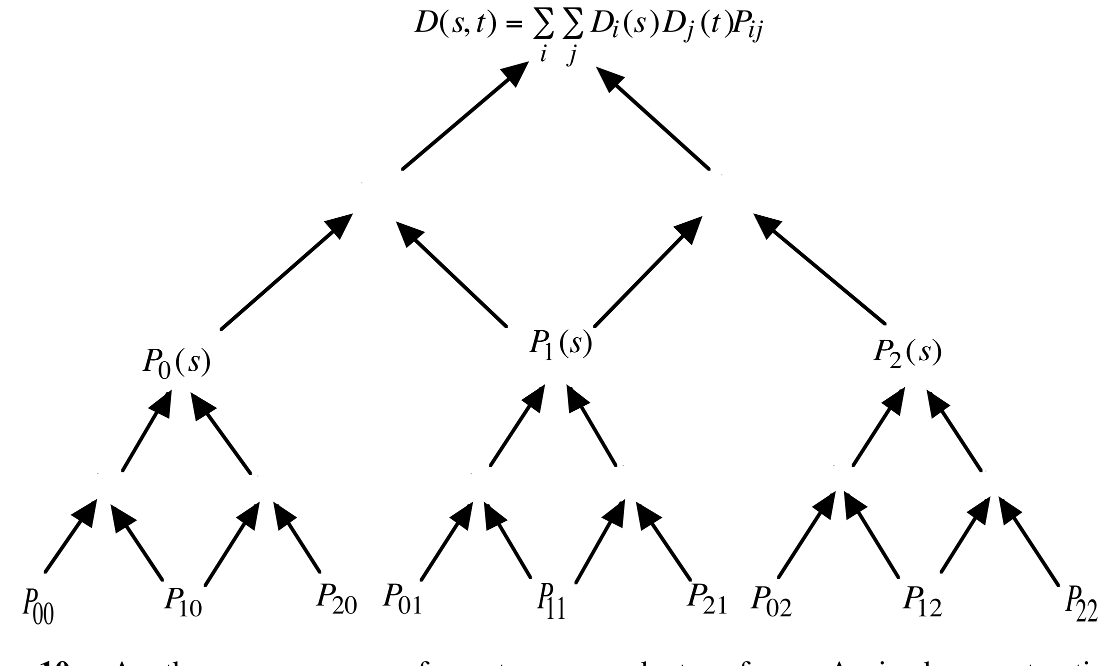

Figure 10: Another up recurrence for a tensor product surface. Again, by construction, the coefficient of P_{ij} in $P_j^*(s)$ is $D_i^m(s)$ and the coefficient of $P_j^*(s)$ in $D(s,t)$ is $D_j^n(t)$. Hence $D_i^m(s)D_j^n(t)$ is the coefficient of P_{ij} in $D(s,t)$. Compare to Figure 9.

The generic up recurrences for tensor product surfaces are defined in the following manner. Starting with a rectangular array of control points P_{ij}, form the curves $P_i(t)$, $i = 0,\ldots,m$, defined by the control points $P_{i,0},\ldots,P_{i,n}$ using the up recurrence for curves. Then for each value of *t*, form the curve $D(s,t)$ defined by the control points $P_0(t),\ldots,P_m(t)$ again using the up recurrence for curves (see Figure 9). As s,t vary over a rectangular domain, this construction sweeps out a rectangular surface patch. Alternatively, we could first form the curves $P_j^*(s)$, $i = 0,\ldots,n$, from the control points $P_{0,j},\ldots,P_{m,j}$ using the up recurrence for curves, and then for each value of *s*, form the curve $D(s,t)$ from the control points $P_0^*(s),\ldots,P_n^*(s)$ again using the up recurrence for curves (see Figure 10).

These two constructions generate the exact same surface. To see why, suppose that the blending functions for the curve scheme of degree *d* are $D_0^d(u),\ldots,D_d^d(u)$. Then the first construction yields the surface

$$D(s,t) = \sum_{i=0}^{m} D_i^m(s)P_i(t) \quad \text{where} \quad P_i(t) = \left\{ \sum_{j=0}^{n} D_j^n(t)P_{ij} \right\},$$

so

$$D(s,t) = \sum_{i=0}^{m} D_i^m(s)\left\{ \sum_{j=0}^{n} D_j^n(t)P_{ij} \right\} = \sum_{i=0}^{m}\sum_{j=0}^{n} D_i^m(s)D_j^n(t)P_{ij} \ .$$

Similarly, the second construction yields the surface

$$D(s,t) = \sum_{j=0}^{n} D_j^n(t)P_j^*(s) \quad \text{where} \quad P_j^*(s) = \left\{ \sum_{i=0}^{m} D_i^m(s)P_{ij} \right\},$$

so once again

$$D(s,t) = \sum_{j=0}^{n} D_j^n(t)\left\{ \sum_{i=0}^{m} D_i^m(s)P_{ij} \right\} = \sum_{i=0}^{m}\sum_{j=0}^{n} D_i^m(s)D_j^n(t)P_{ij} \ .$$

Thus in both cases

$$D(s,t) = \sum_{i=0}^{m}\sum_{j=0}^{n} D_i^m(s)D_j^n(t)P_{ij} \ . \tag{4.1}$$

The products $D_i^m(s)D_j^n(t)$ are called *the tensor product blending functions* and the surface $D(s,t)$ is called the *tensor product surface*.

By the way we can see directly from Figure 9 and Figure 10 that $D_i^m(s)D_j^n(t)$ must be the coefficient of the control point P_{ij}. In Figure 9, by construction, $D_j^n(t)$ is the coefficient of P_{ij} in $P_i(t)$ and $D_i^m(s)$ is the coefficient of $P_i(t)$ in $D(s,t)$. Hence $D_i^m(s)D_j^n(t)$ is the coefficient of P_{ij}

in $D(s,t)$. Similarly, in Figure 10, $D_i^m(s)$ is the coefficient of P_{ij} in $P_j^*(s)$ and $D_j^n(t)$ is the coefficient of $P_j^*(t)$ in $D(s,t)$, so once again $D_i^m(s)D_j^n(t)$ is the coefficient of P_{ij} in $D(s,t)$.

Since tensor product surfaces are completely characterized by their blending functions, we can understand the geometric properties of tensor product surfaces by investigating the algebraic properties of their associated blending functions. These properties are much the same as the properties summarized in Table 1 for curve schemes, so we shall not repeat the properties or their derivations here.

5. Summary

Blending functions provide an alternative approach to freeform curves and surfaces. Given any up recurrence for curves, the associated blending functions can be computed from the corresponding down recurrence. The blending functions for tensor product surfaces are just the products of the blending functions for curves.

For the Lagrange and Bezier schemes we can apply the blending functions to replace the $O(n^2)$ evaluation algorithms of Neville and de Casteljau by $O(n)$ evaluation algorithms based on the blending functions. In addition to supplying explicit formulas, the blending functions provide new insights because the geometric properties of curve and surface schemes correspond to the algebraic properties of the associated blending functions. A list of useful identities for the Bernstein basis functions are provided in the Appendix to this lecture.

Exercises:

1. Show that

$$Convex\,Hull(P_0,\dots,P_n) = \left\{ \sum_{k=0}^{n} c_k P_k \mid \sum_{k=0}^{n} c_k = 1 \;\; and \;\; c_k \geq 0 \right\}.$$

2. Suppose that in the up recurrence the labels on the arrows that enter each node sum to one.
   a. Show that in the down recurrence the functions on any level of the diagram sum to one.
   b. Conclude from part *a* that the blending functions generated by the down recurrence sum to one.

3. Show that a curve scheme has an up recurrence if and only if the associated blending functions have a down recurrence.

4. Let $B_0^n(t),\dots,B_n^n(t)$ denote the Bernstein basis functions of degree n.

   a. Show that: $B_k^n(1-t) = B_{n-k}^n(t)$.

   b. Use the result of part a to prove the symmetry property of Bezier curves.

5. Let $B(t) = \sum_k B_k^n(t) P_k$ be a Bezier curve.

   a. Show that

      i. $\dfrac{B_k^n(t)}{(1-t)^n} = \binom{n}{k} u^k \qquad \text{where } u = \dfrac{t}{1-t}$

      ii. $\dfrac{B_{n-k}^n(t)}{t^n} = \binom{n}{k} v^k \qquad \text{where } v = \dfrac{1-t}{t}$

   b. Let $Q_k = \binom{n}{k} P_k$. Using part a, show that

      i. $\dfrac{B(t)}{(1-t)^n} = \sum_{k=0}^{n} Q_k u^k, \qquad 0 \le t \le 1/2$

      ii. $\dfrac{B(t)}{t^n} = \sum_{k=0}^{n} Q_{n-k} v^k, \qquad 1/2 \le t \le 1$.

   c. A degree n polynomial $Q(u) = \sum_k Q_k u^k$ expressed in monomial form can be evaluated using only n multiplications via Horner's method (see Figure 11). Use part b and Horner's method to construct an $O(n)$ evaluation algorithm for Bezier curves.

   d. Which method is more efficient for evaluating points on a Bezier curve: the ladder algorithm in Section 2.2 or Horner's method?

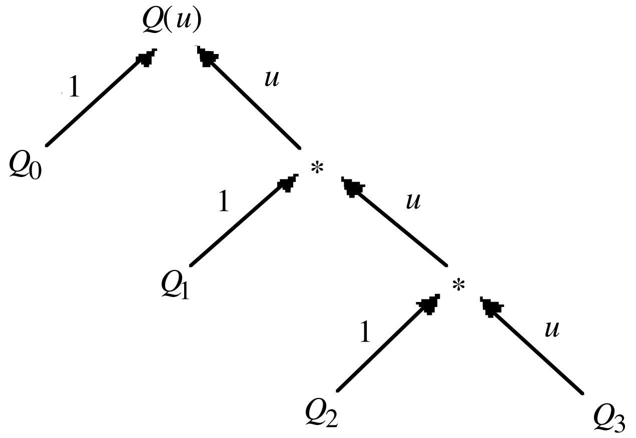

Figure 11: Horner's method. Values at any node are computed by one multiplication and one addition.

6. Let $B_0^n(t),\ldots,B_n^n(t)$ denote the Bernstein basis functions of degree n and let $f(t)$ be a degree n polynomial in t.

   a. Show that $\sum_k (k/n) B_k^n(t) = t$.

   b. The graph of $f(t)$ is the pair $(t, f(t))$. Using part *a*, show that if c_k. are the Bernstein coefficients of $f(t)$, then the Bezier control points for the graph of $f(t)$ are given by $P_k = (k/n, c_k)$, $k = 0,\ldots,n$.

7. Let $B(t) = \sum_k P_k B_k^n(t)$ be a Bezier curve. Show that

   a. $\dfrac{dB_k^n(t)}{dt} = n\left(B_{k-1}^{n-1}(t) - B_k^{n-1}(t)\right)$

   b. $B'(t) = n \sum\limits_{k=0}^{n-1} B_k^{n-1}(t)\left(P_{k+1} - P_k\right)$

   c. Using the result of part *b*, derive the procedure in Lecture 21 for differentiating the de Casteljau algorithm.

8. Let $B(t) = \sum_k P_k B_k^n(t)$ be a Bezier curve and let $Q_k(r) = \sum_k B_j^k(r) P_j$, $k = 0,\ldots,n$. Show that

   a. $B_j^n(rt) = \sum\limits_{k=j}^{n} B_j^k(r) B_k^n(t)$

   b. $B(rt) = \sum\limits_{k=0}^{n} B_k^n(t) Q_k(r)$

   c. Conclude from part b, that the points $Q_k(r)$, $k = 0,\ldots,n$, subdivide the Bezier curve $B(t)$ from $t = 0$ to $t = r$.

   d. Use the result of part *c* to derive the de Casteljau subdivision algorithm for Bezier curves.

9. Descartes' Law of Signs says that

$$Zeroes_{[0,\infty)}\left(\sum_{k=0}^{n} c_k u^k\right) \le Sign\ Alternations\left(c_0,\ldots,c_n\right).$$

   Using Descartes' Law of Sign, show that

$$Zeroes_{[0,1]}\left(\sum_{k=0}^{n} c_k B_k^n(t)\right) \le Sign\ Alternations\left(c_0,\ldots,c_n\right).$$

Appendix: Identities for the Bernstein Basis Functions

1. *Definitions*

$$B_k^n(t) = \binom{n}{k} t^k (1-t)^{n-k} \qquad 0 \le k \le n$$

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

2. *Non-Negativity*

$$B_k^n(t) \ge 0 \qquad 0 \le t \le 1$$

3. *Symmetry*

$$B_k^n(t) = B_{n-k}^n(1-t)$$

4. *Corner Values*

a. $B_k^n(0) = 0 \qquad k \ne 0$

$\qquad = 1 \qquad k = 0$

b. $B_k^n(1) = 0 \qquad k \ne n$

$\qquad = 1 \qquad k = n$

5. *Maximum Values*

$Max\{B_k^n(t)\}$ *occurs at* $t = k/n$

6. *Partition of Unity*

$$\sum_{k=0}^{n} B_k^n(t) = 1$$

7. *Alternating Sum*

$$\sum_{k=0}^{n} (-1)^k B_k^n(t) = (1-2t)^n$$

8. *Representation of Monomials*

$$\binom{n}{j} t^j = \sum_{k=j}^{n} \binom{k}{j} B_k^n(t) \qquad 0 \le j \le n$$

9. *Representation in Terms of Monomials*

$$B_k^n(t) = \sum_{j=k}^{n} (-1)^{j-k} \binom{j}{k}\binom{n}{j} t^j$$

10. *Conversion to Monomial Form*

a. $\dfrac{B_k^n(t)}{(1-t)^n} = \binom{n}{k} u^k \qquad u = \dfrac{t}{1-t}$

b. $\dfrac{B_k^n(t)}{t^n} = \binom{n}{k} v^{n-k} \qquad v = \dfrac{1-t}{t}$

11. *Linear Independence*

$$\sum_{k=0}^{n} c_k B_k^n(t) = 0 \quad \Leftrightarrow \quad c_k = 0 \ \ for \ \ all \ \ k$$

12. *Descartes' Law of Signs*

$$Zeros \ \ in \ \ (0,1) \ \left\{ \sum_{k=0}^{n} c_k B_k^n(t) \right\} \le Sign \ \ Alternations \ \{c_0,\ldots,c_n\}$$

13. *Generating Function*

$$\sum_{k=0}^{n} B_k^n(t) x^k = \{(1-t) + tx)\}^n$$

14. *Recursion*

$$B_k^{n+1}(t) = (1-t) B_k^n(t) + t B_{k-1}^n(t)$$

15. *Subdivision*

a. $B_i^n(rt) = \sum_{k=i}^{n} B_i^k(r) B_k^n(t)$

b. $B_i^n((1-t)r + t) = \sum_{k=0}^{i} B_{i-k}^{n-k}(r) B_k^n(t)$

c. $B_i^n((1-t)r + ts) = \sum_{k=0}^{n} \left\{ \sum_{p+q=i} B_p^{n-k}(r) B_q^k(s) \right\} B_k^n(t)$

16. *Derivatives*

a. $\dfrac{dB_k^n}{dt} = n \left\{ B_{k-1}^{n-1}(t) - B_k^{n-1}(t) \right\}$

b. $\dfrac{d^p B_k^n}{dt^p} = \dfrac{n!}{(n-p)!} \sum_{j=0}^{p} (-1)^{p-j} \binom{p}{j} B_{k-j}^{n-p}(t)$

17. *Integrals*

a. $\int_0^t B_k^n(\tau)d\tau = \frac{1}{n+1}\sum_{j=k+1}^{n+1} B_j^{n+1}(t)$

b. $\int_t^1 B_k^n(\tau)d\tau = \frac{1}{n+1}\sum_{j=0}^{k} B_j^{n+1}(t)$

c. $\int_0^1 B_k^n(\tau)d\tau = \frac{1}{n+1}$

18. *Degree Elevation*

a. $(1-t)B_k^n(t) = \frac{n+1-k}{n+1}B_k^{n+1}(t)$

b. $tB_k^n(t) = \frac{k+1}{n+1}B_{k+1}^{n+1}(t)$

c. $B_k^n(t) = \frac{n+1-k}{n+1}B_k^{n+1}(t) + \frac{k+1}{n+1}B_{k+1}^{n+1}(t)$

19. *Products and Higher Order Degree Elevation*

a. $B_j^m(t)B_k^n(t) = \frac{\binom{m}{j}\binom{n}{k}}{\binom{m+n}{j+k}}B_{j+k}^{m+n}(t)$

b. $B_k^n(t) = \sum_{j=0}^{m} \frac{\binom{m}{j}\binom{n}{k}}{\binom{m+n}{j+k}}B_{j+k}^{m+n}(t)$

20. *Marsden Identity (Binomial Theorem)*

a. $(x-t)^n = \sum_{k=0}^{n} \frac{(-1)^k}{\binom{n}{k}}B_{n-k}^n(x)B_k^n(t)$