

Lecture 26: Blossoming

blossom abundantly, and rejoice even with joy and singing: Isaiah 35:2

1. Motivation

Linear functions are simple; polynomials are complicated. If $L(t) = at$, then clearly

$$L(\mu s + \lambda t) = \mu L(s) + \lambda L(t).$$

More generally, if $L(t) = at + b$, then it is easy to verify that

$$L((1 - \lambda)s + \lambda t) = (1 - \lambda)L(s) + \lambda L(t).$$

In the first case $L(t)$ preserves linear combinations; in the second case $L(t)$ preserves affine combinations. But if $P(t) = a_n t^n + \dots + a_0$ is a polynomial of degree $n > 1$, then

$$P(\mu s + \lambda t) \neq \mu P(s) + \lambda P(t)$$

$$P((1 - \lambda)s + \lambda t) \neq (1 - \lambda)P(s) + \lambda P(t).$$

Thus arbitrary polynomials preserve neither linear nor affine combinations. The key idea behind blossoming is to replace a complicated polynomial function $P(t)$ in one variable by a simple polynomial function $p(u_1, \dots, u_n)$ in many variables that is either linear or affine in each variable. The function $p(u_1, \dots, u_n)$ is called the *blossom* of $P(t)$, and converting from $P(t)$ to $p(u_1, \dots, u_n)$ is called *blossoming*.

Blossoming is intimately linked to Bezier curves. We shall see in Section 3 that the Bezier control points of a polynomial curve are given by the blossom of the curve evaluated at the end points of the parameter interval. Moreover, there is an algorithm for evaluating the blossom recursively that closely mimics the de Casteljau evaluation algorithm for Bezier curves. In this lecture we shall apply the blossom to derive two standard algorithms for Bezier curves: the de Casteljau subdivision algorithm and the procedure for differentiating the de Casteljau evaluation algorithm. We shall also derive algorithms for converting between the Bernstein and monomial representations of a polynomial. Other properties of Bezier curves such as degree elevation are derived using blossoming in the exercises at the end of this lecture.

2. The Blossom

The *blossom* of a degree n polynomial $P(t)$ is the unique symmetric multi-affine function $p(u_1, \dots, u_n)$ that reduces to $P(t)$ along the diagonal. That is, $p(u_1, \dots, u_n)$ is the unique multivariate polynomial satisfying the following three axioms:

Blossoming Axioms

i. Symmetry

$$p(u_1, \dots, u_n) = p(u_{\sigma(1)}, \dots, u_{\sigma(n)}) \text{ for every permutation } \sigma \text{ of } \{1, \dots, n\}$$

ii. Multiaffine

$$p(u_1, \dots, (1 - \alpha)u_k + \alpha v_k, \dots, u_n) = (1 - \alpha)p(u_1, \dots, u_k, \dots, u_n) + \alpha p(u_1, \dots, v_k, \dots, u_n)$$

iii. Diagonal

$$p(t, \dots, t) = P(t)$$

The symmetry property says that the order of the parameters u_1, \dots, u_n does not matter when we evaluate the blossom $p(u_1, \dots, u_n)$. The multiaffine property is just a fancy way to say that $p(u_1, \dots, u_n)$ is degree one in each variable (see Exercise 4). The diagonal property connects the blossom back to the original polynomial. At first, these three axioms may seem very abstract and complicated, but we shall see shortly that blossoming is both explicit and straightforward.

We are interested in blossoming because of the following key property, which we shall derive in Section 3, relating the blossom of a polynomial to its Bezier control points.

Dual Functional Property

Let $P(t)$ be a Bezier curve over the interval $[a, b]$ with control points P_0, \dots, P_n . Then

$$P_k = p(\underbrace{a, \dots, a}_{n-k}, \underbrace{b, \dots, b}_k) \quad k = 0, \dots, n.$$

We have yet to establish the existence and uniqueness of a function satisfying the three blossoming axioms. But before we proceed to prove both existence and uniqueness, let us get a better feel for the blossom by computing a few simple examples.

Example 1: Cubic Polynomials.

Consider the monomials $1, t, t^2, t^3$ as cubic polynomials. It is easy to blossom these monomials, since in each case it is easy to verify that the associated function $p(u_1, u_2, u_3)$ given below is symmetric, multiaffine, and reduces to the required monomial along the diagonal:

$$P(t) = 1 \Rightarrow p(u_1, u_2, u_3) = 1$$

$$P(t) = t \Rightarrow p(u_1, u_2, u_3) = \frac{u_1 + u_2 + u_3}{3}$$

$$P(t) = t^2 \Rightarrow p(u_1, u_2, u_3) = \frac{u_1 u_2 + u_2 u_3 + u_3 u_1}{3}$$

$$P(t) = t^3 \Rightarrow p(u_1, u_2, u_3) = u_1 u_2 u_3.$$

Notice that the functions on the right hand side are, up to a constant multiple, simply the elementary symmetric functions in three variables. Notice too that we must decide the degree of each monomial before we blossom, since the degree of the monomial tells us how many variables must appear in the blossom. Using these results, we can blossom any cubic polynomial, since

$$P(t) = a_3 t^3 + a_2 t^2 + a_1 t + a_0$$

$$p(u_1, u_2, u_3) = a_3 u_1 u_2 u_3 + a_2 \frac{u_1 u_2 + u_2 u_3 + u_3 u_1}{3} + a_1 \frac{u_1 + u_2 + u_3}{3} + a_0.$$

Similar techniques can be applied to blossom polynomials of any arbitrary degree n by first blossoming the monomials $M_k^n(t) = t^k$, $k = 0, \dots, n$, using elementary symmetric functions in n variables, and then applying linearity. This observation leads to the following results.


Proposition 1: (*Blossoming Monomials*)

The blossom of the monomial $M_k^n(t) = t^k$ (considered as a polynomial of degree n) is given by

$$m_k^n(u_1, \dots, u_n) = \frac{\sum u_{i_1} \cdots u_{i_k}}{\binom{n}{k}} \tag{2.1}$$

where the sum is taken over all subsets $\{i_1, \dots, i_k\}$ of $\{1, \dots, n\}$.

Proof: To show that $m_k^n(u_1, \dots, u_n)$ is the blossom of $M_k^n(t)$, we need to verify that the three blossoming axioms are satisfied. By construction $m_k^n(u_1, \dots, u_n)$ is symmetric, since the sum on the right hand side of Equation (2.1) is taken over all subsets $\{i_1, \dots, i_k\}$ of $\{1, \dots, n\}$. Also the function on the right hand side of Equation (2.1) is multiaffine, since each variable appears in each term to at most the first power. Finally, since the number of subset of $\{1, \dots, n\}$ with k elements is $\binom{n}{k}$, along the diagonal

$$m_k^n(t, \dots, t) = \frac{\binom{n}{k} t^k}{\binom{n}{k}} = M_k^n(t).$$


Theorem 2: (*Existence of the Blossom*)

For every degree n polynomial $P(t)$, there exists a symmetric multiaffine function $p(u_1, \dots, u_n)$ that reduces to $P(t)$ along the diagonal. That is, there exists a blossom $p(u_1, \dots, u_n)$ for every polynomial $P(t)$.

Proof: By Proposition 1, the blossom exists for the monomials t^k , $k = 0, \dots, n$. Therefore since every polynomial is the sum of monomials and since the blossom of the sum is the sum of the blossoms (see Exercise 1), a blossom $p(u_1, \dots, u_n)$ exists for every polynomial $P(t)$.

3. Blossoming and the de Casteljau Algorithm

A word about notation before we proceed. Throughout this lecture, especially in the diagrams, we shall adopt the multiplicative notation $u_1 \cdots u_n$ to represent the blossom value $p(u_1, \dots, u_n)$. Though an abuse of notation, this multiplicative representation is highly suggestive. For example, multiplication is commutative and the blossom is symmetric

$$u_1 \cdots u_n = u_{\sigma(1)} \cdots u_{\sigma(n)} \iff p(u_1, \dots, u_n) = p(u_{\sigma(1)}, \dots, u_{\sigma(n)}).$$

Moreover, multiplication distributes through addition and the blossom is multiaffine. Thus

$$u = \frac{b-u}{b-a}a + \frac{u-a}{b-a}b \implies u_1 \cdots u_n u = \frac{b-u}{b-a}u_1 \cdots u_n a + \frac{u-a}{b-a}u_1 \cdots u_n b,$$

and similarly

$$u = \frac{b-u}{b-a}a + \frac{u-a}{b-a}b \implies p(u_1, \dots, u_n, u) = \frac{b-u}{b-a}p(u_1, \dots, u_n, a) + \frac{u-a}{b-a}p(u_1, \dots, u_n, b).$$

Since symmetry and multiaffinity are the main properties featured in the diagrams, the same diagrams make sense both for multiplication and for blossoming. Thus this multiplicative notation for the blossom is both natural and evocative. In addition, these similarities between multiplication and blossoming suggest that corresponding to identities for multiplication we should expect analogous identities for the blossom. In fact, we shall see an important example of such an analogous identity in Section 4, where we introduce an analogue of the binomial theorem for the blossom.

Using this multiplicative notation, Figure 1 shows how to compute values of $p(t, \dots, t)$ from the blossom values $p(\underbrace{a, \dots, a}_{n-k}, \underbrace{b, \dots, b}_k)$, $k = 0, \dots, n$, recursively by applying the multiaffine and

symmetry properties at each node. For example, since

$$t = \frac{b-t}{b-a}a + \frac{t-a}{b-a}b$$

it follows by the multiaffine property that

$$p(a, b, t) = p\left(a, b, \frac{b-t}{b-a}a + \frac{t-a}{b-a}b\right) = \frac{b-t}{b-a}p(a, b, a) + \frac{t-a}{b-a}p(a, b, b).$$

Similar identities hold at all the other nodes in Figure 1.

Now compare the blossoming algorithm in Figure 1 to the de Casteljau algorithm in Figure 2. Clearly Figure 1 is the de Casteljau algorithm for $p(t, \dots, t) = P(t)$ with control points $P_k = p(\underbrace{a, \dots, a}_{n-k}, \underbrace{b, \dots, b}_k)$. This observation has several important consequences, which we summarize

in Theorems 3,4,5.

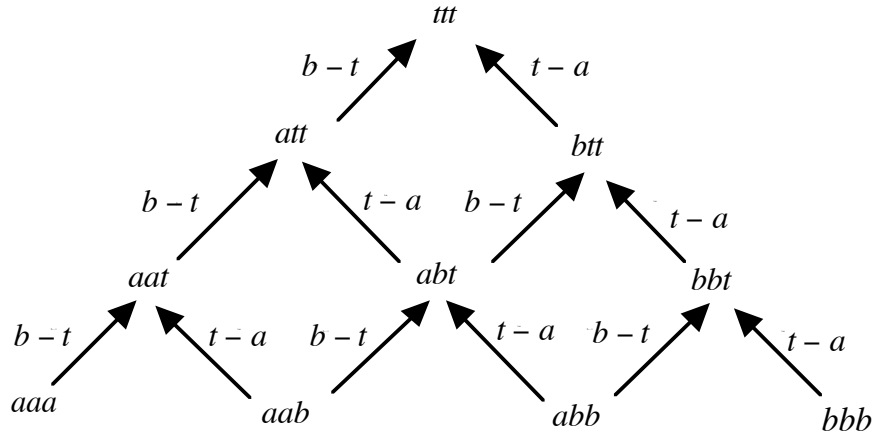


Figure 1: Computing $p(t, \dots, t)$ from the blossom values $p(\underbrace{a, \dots, a}_{n-k}, \underbrace{b, \dots, b}_k)$, $k = 0, \dots, n$. Here we illustrate the cubic case, and we adopt the multiplicative notation uvw for $p(u, v, w)$. As usual the label on each edge must be normalized by dividing by $b - a$, so that the labels along the two arrows entering each node sum to one.

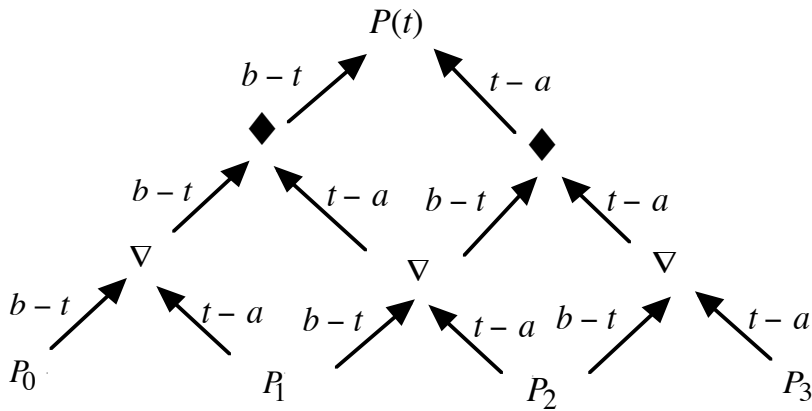


Figure 2: The de Casteljau algorithm for a cubic Bezier curve $P(t)$ on the interval $[a, b]$. Compare to Figure 1.

Theorem 3: (Every Polynomial Curve is a Bezier Curve)

Every polynomial can be expressed in Bezier form. That is, for every polynomial $P(t)$ and every interval $[a, b]$ there exist control points P_0, \dots, P_n such that over the interval $[a, b]$ the polynomial $P(t)$ is generated by the de Casteljau algorithm with control point P_0, \dots, P_n .

Proof: This result follows immediately from Figure 1, since Figure 1 is the de Casteljau algorithm for $p(t, \dots, t) = P(t)$ with control points $P_k = p(\underbrace{a, \dots, a}_{n-k}, \underbrace{b, \dots, b}_k)$, $k = 0, \dots, n$.

Theorem 4: (Dual Functional Property of the Blossom)

Let $P(t)$ be a Bezier curve defined over the interval $[a, b]$, and let $p(u_1, \dots, u_n)$ be the blossom of $P(t)$. Then

$$p(\underbrace{a, \dots, a}_{n-k}, \underbrace{b, \dots, b}_k) = P_k = k\text{th Bezier Control Point of } P(t). \quad (3.1)$$

Proof: Again, this result follows immediately from Figure 1.

Theorem 5: (Uniqueness of the Blossom)

Let $P(t)$ be a polynomial of degree n . Then the blossom $p(u_1, \dots, u_n)$ of $P(t)$ is unique. That is, for each polynomial $P(t)$ there is only one function $p(u_1, \dots, u_n)$ that is symmetric, multiaffine, and reduces to $P(t)$ along the diagonal.

Proof: For any polynomial $P(t)$, Figure 3 shows how to compute an arbitrary blossom value $p(u_1, \dots, u_n)$ from the blossom values $p(\underbrace{a, \dots, a}_{n-k}, \underbrace{b, \dots, b}_k)$, $k = 0, \dots, n$ by substituting u_k for t on the k th level of the de Casteljau algorithm.

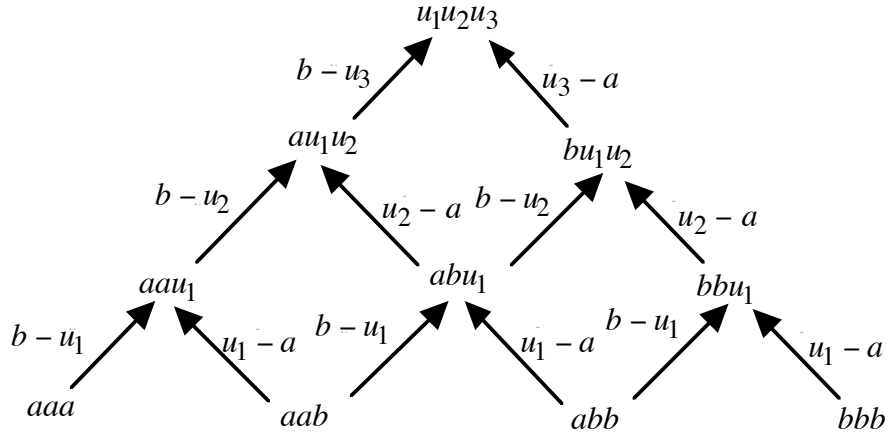


Figure 3: Computing an arbitrary blossom value $p(u_1, \dots, u_n)$ from the blossom values $p(\underbrace{a, \dots, a}_{n-k}, \underbrace{b, \dots, b}_k)$, $k = 0, \dots, n$. Here again we illustrate the cubic case, and we adopt the multiplicative notation uvw for $p(u, v, w)$. As usual the label on each edge must be normalized by dividing by $b - a$.

Thus every blossom value $p(u_1, \dots, u_n)$ is completely determined by the blossom values $p(\underbrace{a, \dots, a}_{n-k}, \underbrace{b, \dots, b}_k)$, $k = 0, \dots, n$. Now suppose that the polynomial $P(t)$ has two blossoms $p(u_1, \dots, u_n)$ and $q(u_1, \dots, u_n)$. Then by the Theorem 4

$$p(\underbrace{a, \dots, a}_{n-k}, \underbrace{b, \dots, b}_k) = q(\underbrace{a, \dots, a}_{n-k}, \underbrace{b, \dots, b}_k), \quad k = 0, \dots, n,$$

since both sides represent the Bezier control points of $P(t)$ and these control points are unique (see Lecture 24, Section 5.2). But we have seen in Figure 3 that any arbitrary blossom value is completely determined by these particular $n + 1$ blossom values. Therefore

$$p(u_1, \dots, u_n) = q(u_1, \dots, u_n),$$

so the blossom of $P(t)$ is unique.



3.1 Bezier Subdivision from Blossoming. The de Casteljau algorithm for subdividing Bezier curves is easy to derive using the dual functional property (Equation 3.1) of the blossom. Let $P(t)$ be a Bezier curve over the interval $[a, b]$ with control points P_0, \dots, P_n . By the dual functional property,

$$P_k = p(\underbrace{a, \dots, a}_{n-k}, \underbrace{b, \dots, b}_k) \quad k = 0, \dots, n.$$

To subdivide this Bezier curve at the parameter t , we must find the Bezier control points for $P(t)$ over the intervals $[a, t]$ and $[t, b]$. Again by the dual functional property, these control points Q_0, \dots, Q_n and R_0, \dots, R_n are

$$Q_k = p(\underbrace{a, \dots, a}_{n-k}, \underbrace{t, \dots, t}_k) \quad k = 0, \dots, n$$

$$R_k = p(\underbrace{t, \dots, t}_{n-k}, \underbrace{b, \dots, b}_k) \quad k = 0, \dots, n.$$

But look at Figure 1. If we interpret every triple uvw as the blossom value $p(u, v, w)$, then the control points Q_k and R_k emerge along the left and right lateral edges of the triangle. Generalized to arbitrary degree, this observation is precisely the de Casteljau subdivision algorithm of Lecture 25. Notice how blossoming simplifies the analysis of Bezier curves by providing natural labels for all the interior nodes in the de Casteljau algorithm. Blossoming is a beautiful, powerful, clever, almost magical idea.

4. Differentiation and the Homogeneous Blossom

Points along polynomial curves can be represented in terms of the blossom by invoking the diagonal property. But what about derivatives? Curves take on values that are points in affine space; so too does the blossom. Indeed, only affine combinations of blossom values are permitted. But derivatives represent tangent vectors not points. To represent derivatives, we need a variant of the blossom that takes on values in a vector space, rather than an affine space. Here we shall construct such a blossom by applying the technique of *homogenization*. Homogenization lifts the domain (and the range) from an affine space to a vector space, so we are no longer restricted to affine combinations, but can exploit instead arbitrary linear combinations.

4.1 Homogenization and the Homogeneous Blossom. Monomials are the simplest polynomials. If $P(t) = t^n$, then $P(ct) = c^n t^n = c^n P(t)$. Monomials are not linear -- $P(ct) \neq c P(t)$ -- but $P(ct) = c^n P(t)$ is almost as good. On the other hand, if $P(t) = a_n t^n + \dots + a_0$, $n > 1$, then $P(ct) \neq c^n P(t)$ because different terms have different degrees. The key idea behind homogenization is to introduce a new variable to homogenize the polynomial so that all the terms have the same degree.

To homogenize a degree n polynomial $P(t) = \sum_k a_k t^k$, we multiply each term t^k by w^{n-k} . Homogenization creates a new polynomial in two variables, $P(t,w) = \sum_k a_k t^k w^{n-k}$, which is homogeneous of degree n -- that is, each term has the same total degree n . Therefore $P(ct,cw) = c^n P(t,w)$. We can easily recover $P(t)$ from $P(t,w)$ because $P(t) = P(t,1)$. Constructing $P(t,w)$ from $P(t)$ is called *homogenization*; recovering $P(t)$ from $P(t,w)$ is called *dehomogenization*.

To homogenize the blossom $p(u_1, \dots, u_n)$, we homogenize with respect to each variable independently. Thus the homogeneous version of $p(u_1, \dots, u_n)$ is another polynomial $p((u_1, v_1), \dots, (u_n, v_n))$ that is homogeneous with respect to each pair of variables (u_k, v_k) . In every term of $p(u_1, \dots, u_n)$ each variable u_k appears to at most the first power, so every term of the homogeneous polynomial $p((u_1, v_1), \dots, (u_n, v_n))$ has as a factor either u_k or v_k but not both. Since $p((u_1, v_1), \dots, (u_n, v_n))$ is homogeneous of degree one in each pair of variables (u_k, v_k)

$$p((u_1, v_1), \dots, c(u_k, v_k), \dots, (u_n, v_n)) = c p((u_1, v_1), \dots, (u_k, v_k), \dots, (u_n, v_n)).$$

Again we can dehomogenize $p((u_1, v_1), \dots, (u_n, v_n))$ by setting $v_k = 1$, $k = 1, \dots, n$. Thus $p((u_1, 1), \dots, (u_n, 1)) = p(u_1, \dots, u_n)$.

We can also blossom the homogenization of $P(t)$. We define the blossom of a homogeneous polynomial $P(t,w)$ to be the unique symmetric, multilinear polynomial $p((u_1, v_1), \dots, (u_n, v_n))$ that reduces to $P(t,w)$ along the diagonal. Thus the homogeneous blossom satisfies the following axioms.

Homogeneous Blossoming Axioms

i. Symmetry

$$p((u_1, v_1), \dots, (u_k, v_k), \dots, (u_n, v_n)) = p((u_{\sigma(1)}, v_{\sigma(1)}), \dots, (u_{\sigma(n)}, v_{\sigma(n)})) \text{ for every permutation } \sigma \text{ of } \{1, \dots, n\}$$

ii. Multilinear

$$\begin{aligned} p((u_1, v_1), \dots, (u_k, v_k) + (r_k, s_k), \dots, (u_n, v_n)) \\ = p((u_1, v_1), \dots, (u_k, v_k), \dots, (u_n, v_n)) + p((u_1, v_1), \dots, (r_k, s_k), \dots, (u_n, v_n)) \\ p((u_1, v_1), \dots, c(u_k, v_k), \dots, (u_n, v_n)) = c p((u_1, v_1), \dots, (u_k, v_k), \dots, (u_n, v_n)) \end{aligned}$$

iii. Diagonal

$$p((t, w), \dots, (t, w)) = P(t, w)$$

Geometrically, the variables t, u_1, \dots, u_n represent parameters in a 1-dimensional affine space. The homogenizing parameters w, v_1, \dots, v_n play the role of mass, lifting the variable t, u_1, \dots, u_n from a 1-dimensional affine space to a 2-dimensional vector space, where we can perform linear algebra.

Thus in the homogeneous blossom, we have replaced the multiaffine property by the multilinear property:

$$\begin{aligned} p((u_1, v_1), \dots, (u_k, v_k) + (r_k, s_k), \dots, (u_n, v_n)) \\ = p((u_1, v_1), \dots, (u_k, v_k), \dots, (u_n, v_n)) + p((u_1, v_1), \dots, (r_k, s_k), \dots, (u_n, v_n)) \\ p((u_1, v_1), \dots, c(u_k, v_k), \dots, (u_n, v_n)) = c p((u_1, v_1), \dots, (u_k, v_k), \dots, (u_n, v_n)). \end{aligned}$$

This property is equivalent to the fact that for each parameter pair (u_i, v_i) , either u_i or v_i but not both appear in each term to the first power (see Exercise 5).

Now starting with any polynomial $P(t)$, we can blossom and then homogenize or we homogenize and then blossom. Figure 4 illustrates how this works in practice for the monomials $1, t, t^2, t^3$ considered as cubic polynomials. Notice that if we blossom and then homogenize we get the same result as when we homogenize and then blossom. We formalize this result in Theorem 6.

$$\begin{array}{ccc} P(t) = 1 & \xrightarrow{\text{blossom}} & p(u_1, u_2, u_3) = 1 \\ \downarrow \text{homogenize} & & \downarrow \text{homogenize} \\ P(t, w) = w^3 & \xrightarrow{\text{blossom}} & p((u_1, v_1), (u_2, v_2), (u_3, v_3)) = v_1 v_2 v_3 \end{array}$$

$$\begin{array}{ccc}
P(t) = t & \xrightarrow{\text{blossom}} & p(u_1, u_2, u_3) = \frac{u_1 + u_2 + u_3}{3} \\
\downarrow \text{homogenize} & & \downarrow \text{homogenize} \\
P(t, w) = tw^2 & \xrightarrow{\text{blossom}} & p((u_1, v_1), (u_2, v_2), (u_3, v_3)) = \frac{u_1 v_2 v_3 + u_2 v_3 v_1 + u_3 v_1 v_2}{3} \\
\\
P(t) = t^2 & \xrightarrow{\text{blossom}} & p(u_1, u_2, u_3) = \frac{u_1 u_2 + u_2 u_3 + u_3 u_1}{3} \\
\downarrow \text{homogenize} & & \downarrow \text{homogenize} \\
P(t, w) = t^2 w & \xrightarrow{\text{blossom}} & p((u_1, v_1), (u_2, v_2), (u_3, v_3)) = \frac{u_1 u_2 v_3 + u_2 u_3 v_1 + u_3 u_1 v_2}{3} \\
\\
P(t) = t^3 & \xrightarrow{\text{blossom}} & p(u_1, u_2, u_3) = u_1 u_2 u_3 \\
\downarrow \text{homogenize} & & \downarrow \text{homogenize} \\
P(t, w) = t^3 & \xrightarrow{\text{blossom}} & p((u_1, v_1), (u_2, v_2), (u_3, v_3)) = u_1 u_2 u_3
\end{array}$$

Figure 4: Blossoming and homogenizing the monomials $1, t, t^2, t^3$. Observe that blossoming and homogenization commute.

Theorem 6: *Blossoming and homogenization commute.*

Proof: This result follows immediately from Figure 5, since it is straightforward to verify that the functions in the right hand column are symmetric, multiaffine on top, multilinear on bottom, and reduce to the corresponding functions in the left hand column along the diagonal.

$$\begin{array}{ccc}
\sum_k c_k \binom{n}{k} t^k & \xrightarrow{\text{blossom}} & \sum_k c_k \sum u_{i_1} \cdots u_{i_k} \\
\downarrow \text{homogenize} & & \downarrow \text{homogenize} \\
\sum_k c_k \binom{n}{k} t^k w^{n-k} & \xrightarrow{\text{blossom}} & \sum_k c_k \sum u_{i_1} \cdots u_{i_k} v_{i_{k+1}} \cdots v_{i_n}
\end{array}$$

Figure 5: Blossoming and homogenization commute. The inner sum on the right hand side is taken over all subsets $\{i_1, \dots, i_k\}$ of $\{1, \dots, n\}$.

Notice that Figure 5 also establishes the existence of the homogeneous blossom, since the expression in the lower right hand corner of the diagram is symmetric, multilinear and reduces to the expression on the bottom left along the diagonal. Uniqueness follows from dehomogenization: if we dehomogenize the homogeneous blossom, then we get the multiaffine blossom. Since the multiaffine blossom is unique, rehomogenization shows that the homogeneous blossom is also unique.

We constructed the multiaffine blossom in Figure 3 by blossoming the de Casteljau algorithm, replacing t by a different parameter u_k on the k th level of the algorithm. We can do the same for the multilinear blossom. Begin by homogenizing the de Casteljau algorithm. This amounts to replacing $b - t \rightarrow bw - t$ and $t - a \rightarrow t - aw$ to insure that each term has the same total degree (see Figure 6). To blossom, we now replace the pair (t, w) by the pair (u_k, v_k) on k th level of the algorithm (see Figure 7). This process generates a symmetric, multilinear function that reduces to the homogeneous curve when we replace each pair (u_k, v_k) by (t, w) . This function is multilinear rather than multiaffine because it is linear in (u_k, v_k) on the k th level of the algorithm. Thus u_k or v_k , but not both, appears in every term to the first power. Notice, by the way, that if we blossom first and then homogenize, we get exactly the same diagram. Blossoming first gives us Figure 3; homogenizing this diagram generates Figure 7. We illustrate all four variants of the de Casteljau algorithm -- original, blossomed, homogenized, and blossomed and homogenized -- in Figure 8.

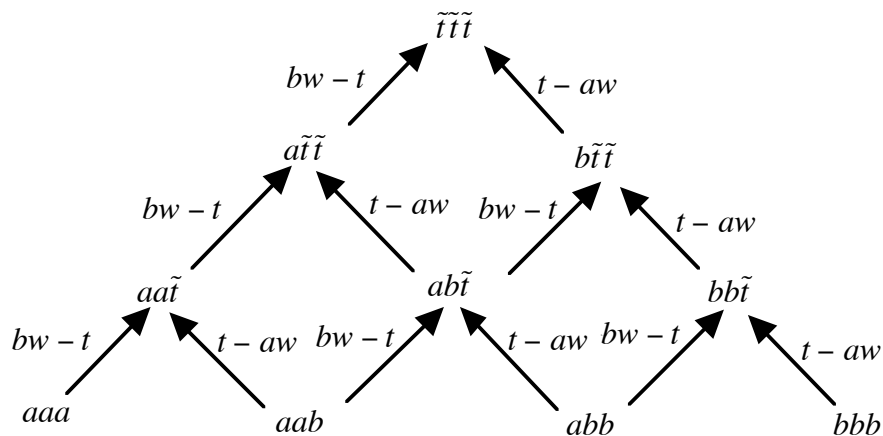


Figure 6: The homogeneous version of the de Casteljau algorithm for cubic Bezier curves. This diagram is generated from the de Casteljau algorithm (Figure 1) by replacing $b - t \rightarrow bw - t$ and $t - a \rightarrow t - aw$ along the edges of the triangle. Here we use \sim to denote homogeneous values, so $\tilde{t} = (t, w)$ while $a = (a, 1)$. Notice that

$$(t, w) = \frac{bw - t}{b - a}(a, 1) + \frac{t - aw}{b - a}(b, 1).$$

Therefore by the linearity of the homogenous blossom

$$p(a,b,\tilde{t}) = p\left(a,b,\frac{bw-t}{b-a}a + \frac{t-aw}{b-a}b\right) = \frac{bw-t}{b-a}p(a,b,a) + \frac{t-aw}{b-a}p(a,b,b),$$

and similar identities hold at all the other nodes in this diagram.

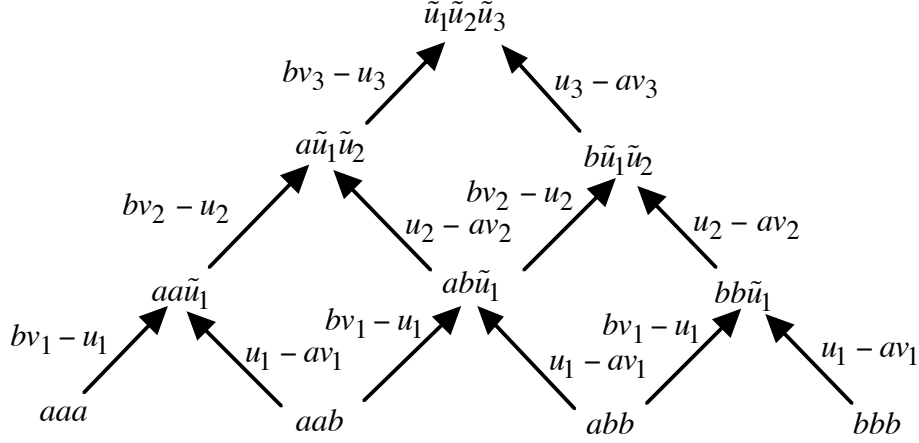


Figure 7: The homogeneous blossom of a cubic Bezier curve. This diagram is generated by blossoming the homogeneous version of the de Casteljau algorithm (Figure 6) -- that is, by replacing the pair (t,w) with the pair (u_k, v_k) on k th level of the algorithm. This diagram can also be generated from the multiaffine blossom (Figure 3) by homogenizing the functions along the edges. Thus once again we see that blossoming and then homogenizing is equivalent to homogenizing and then blossoming. As in Figure 6, we use $\tilde{}$ to denote homogeneous values, so $\tilde{u} = (u,v)$ while $a = (a,1)$. Notice that

$$(u,v) = \frac{bv-u}{b-a}(a,1) + \frac{u-av}{b-a}(b,1).$$

Therefore by the linearity of the homogenous blossom

$$p(a,b,\tilde{u}_1) = p\left(a,b,\frac{bv_1-u_1}{b-a}a + \frac{u_1-av_1}{b-a}b\right) = \frac{bv_1-u_1}{b-a}p(a,b,a) + \frac{u_1-av_1}{b-a}p(a,b,b),$$

and similar identities hold at all the other nodes in this diagram.

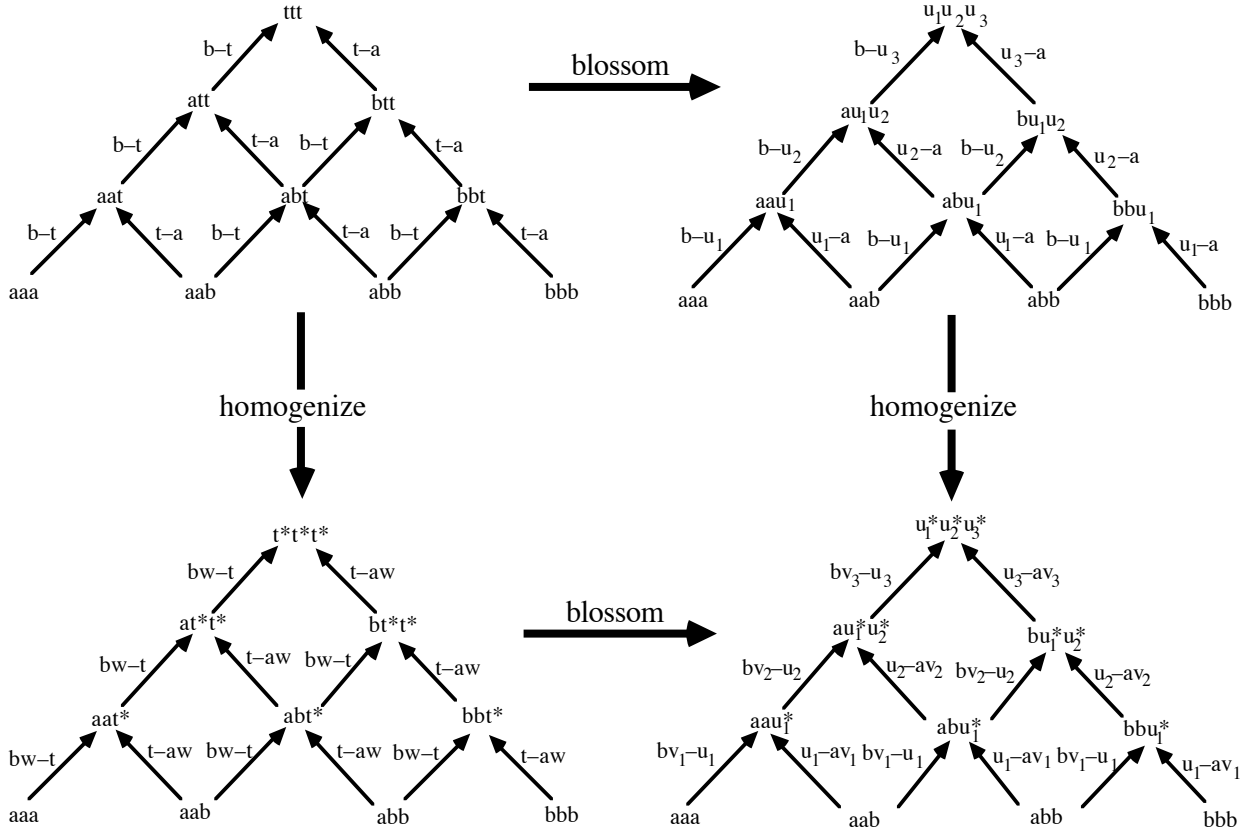


Figure 8: Four versions of the de Casteljau algorithm for cubic Bezier curves: the standard version (upper left), the homogeneous version (lower left), the blossomed version (upper right), and the homogeneous blossom (lower right). Blossoming and homogenization commute, since we get the same result by traversing right and then down or down and then right. Here we use $*$ to denote homogeneous values, so $t^* = (t, w)$ and $u^* = (u, v)$ whereas $a = (a, 1)$ and $b = (b, 1)$.

4.2 Differentiating the de Casteljau Algorithm. We built the homogeneous blossom to deal specifically with differentiation. We are now going to use a variant of Taylor's theorem along with the binomial expansion of the homogenous blossom to derive a formula for the derivatives of a polynomial in terms of the values of its homogeneous blossom.

Theorem 7: (*Taylor's Theorem*)

Let $P(t)$ be a polynomial of degree n . Then

$$P(t+h) = \sum_{k=0}^n \frac{P^{(k)}(t)}{k!} h^k. \quad (4.1)$$

Proof: Recall from calculus that the standard version of Taylor's theorem is:

$$P(x) = P(t) + P'(t)(x-t) + \frac{P''(t)}{2!}(x-t)^2 + \cdots + \frac{P^{(n)}(t)}{n!}(x-t)^n .$$

Setting $x = t + h$ yields

$$P(t+h) = P(t) + P'(t)h + \frac{P''(t)}{2!}h^2 + \cdots + \frac{P^{(n)}(t)}{n!}h^n .$$



We can also compute $P(t+h)$ using the diagonal property of the homogeneous blossom. Let $t = (t,1)$, and $\delta = (1,0)$. Then

$$t+h \equiv (t+h,1) = (t,1) + h(1,0) = t+h\delta .$$

Therefore by the diagonal property of the homogeneous blossom

$$P(t+h) = p(t+h\delta, \dots, t+h\delta) .$$

We can expand $p(t+h\delta, \dots, t+h\delta)$ by the multilinear property to get a formula for $p(t+h\delta, \dots, t+h\delta)$ similar in form to the binomial theorem.

Theorem 8: (*Binomial Expansion*)

Let $p(u_1, \dots, u_n)$ be a symmetric multiaffine function. Then

$$p(t+h\delta, \dots, t+h\delta) = \sum_{k=0}^n \binom{n}{k} p(\underbrace{t, \dots, t}_{n-k}, \underbrace{\delta, \dots, \delta}_k) h^k , \tag{4.2}$$

where $t = (t,1)$ and $\delta = (1,0)$.

Proof: The proof is an inductive argument, similar to the inductive proof of the binomial theorem. We illustrate the case $n = 2$ and leave the general case as an exercise (see Exercise 8). To prove the result for $n = 2$, just expand by linearity:

$$\begin{aligned} p(t+h\delta, t+h\delta) &= p(t, t+h\delta) + p(h\delta, t+h\delta) \\ &= p(t, t) + p(t, h\delta) + p(h\delta, t) + p(h\delta, h\delta) \\ &= p(t, t) + 2p(t, h\delta) + p(h\delta, h\delta) \\ &= p(t, t) + 2p(t, \delta)h + p(\delta, \delta)h^2 . \end{aligned}$$



Theorem 9: Let P be a polynomial of degree n , and let p be the homogenous blossom of P . Then

$$P^{(k)}(t) = \frac{n!}{(n-k)!} p(\underbrace{t, \dots, t}_{n-k}, \underbrace{\delta, \dots, \delta}_k) , \tag{4.3}$$


where $t = (t,1)$ and $\delta = (1,0)$.

Proof: By the diagonal property of the homogeneous blossom

$$P(t+h) = p(t+h\delta, \dots, t+h\delta) .$$

and this equality is valid for all values of h . Comparing the coefficients of h^k in Equation (4.1)

and Equation (4.2) for $P(t+h)$ and $p(t+h\delta, \dots, t+h\delta)$ yields

$$P^{(k)}(t) = \frac{n!}{(n-k)!} P(\underbrace{t, \dots, t}_{n-k}, \underbrace{\delta, \dots, \delta}_k).$$


Thus to find the k th derivative of a polynomial $P(t)$, we need only compute the blossom value $p(\underbrace{t, \dots, t}_{n-k}, \underbrace{\delta, \dots, \delta}_k)$ and multiply by a constant. But we have already shown that by blossoming and homogenizing the de Casteljau algorithm, we can compute the homogenous blossom $p((u_1, v_1), \dots, (u_n, v_n))$ for any values of the parameters $(u_1, v_1), \dots, (u_n, v_n)$ (see Figure 7). We illustrate the algorithms for computing $p(t, \dots, t, \delta)$ and $p(t, \dots, t, \delta, \delta)$ in Figures 9, 10.

Notice that if we homogenize the de Casteljau algorithm and then we replace $(t, w) \rightarrow (1, 0)$ on any level of the de Casteljau algorithm, the effect is to replace $bw - t \rightarrow -1$ and $t - aw \rightarrow +1$; that is, the effect is to differentiate one level of the de Casteljau algorithm. Now if we replace $(t, w) \rightarrow (t, 1)$ on the remaining levels of the algorithm -- that is, if we dehomogenize the remaining levels -- then, up to a constant multiple, we get the derivative of the original Bezier curve (see Figure 9). If on k levels of the homogeneous de Casteljau algorithm we replace $(t, w) \rightarrow (1, 0)$ and on the remaining $n - k$ levels we replace $(t, w) \rightarrow (t, 1)$, then, up to a constant multiple, we obtain the k th derivative of the original Bezier curve. This algorithm is precisely the derivative algorithm we introduced in Lecture 24. Blossoming provides an alternative proof as well as a more general algorithm.

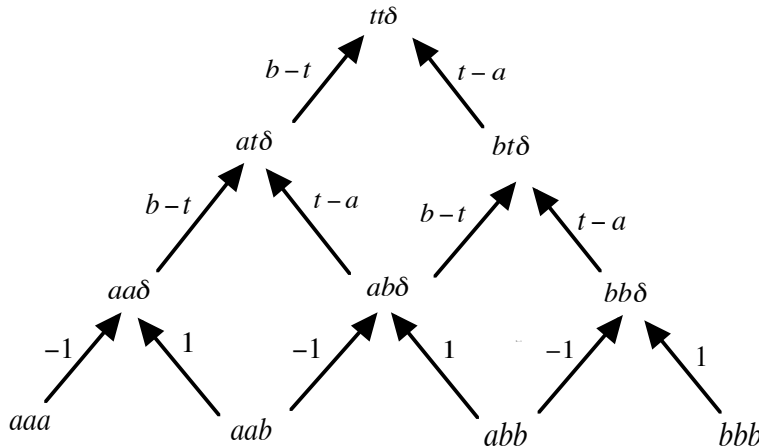


Figure 9: The first derivative of a cubic Bezier curve. Compare to Figure 7 with $(u_1, v_1) = \delta = (1, 0)$, and $(u_2, v_2) = (u_3, v_3) = (t, 1)$.

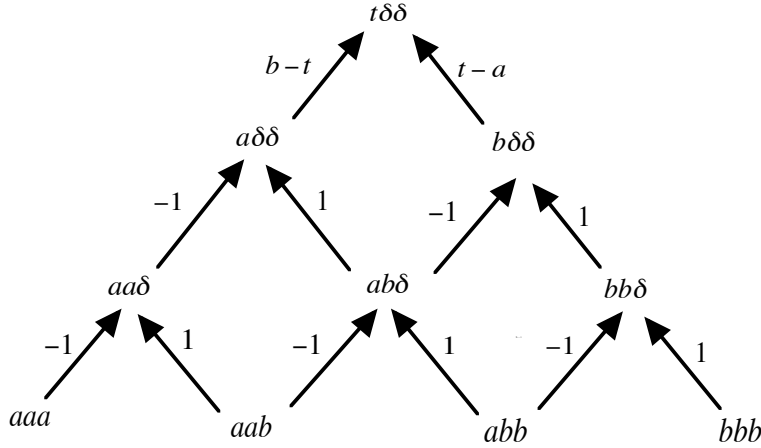


Figure 10: The second derivative of a cubic Bezier curve. Compare to Figure 7 with $(u_1, v_1) = (u_2, v_2) = \delta = (1, 0)$, and $(u_3, v_3) = (t, 1)$.

4.3 Conversion Algorithms Between Monomial and Bezier Form. We can also use blossoming to develop algorithms to convert between monomial and Bezier form. That is, given the monomial coefficients of a polynomial, we can find the Bezier control points, and conversely given the Bezier control points we can find the monomial coefficients. The key observation here is that the monomial coefficients are derivatives, at least up to constant multiples, and we have just seen that derivatives can be represented, at least up to constant multiples, by values of the homogeneous blossom.

To see the connection between monomial coefficients, derivatives, and blossom values, suppose

$$P(t) = a_0 + a_1 t + \cdots + a_k t^k + \cdots + a_n t^n.$$

Recall that by Taylor's theorem

$$P(t) = P(0) + P'(0)t + \cdots + \frac{P^{(k)}(0)}{k!} t^k + \cdots + \frac{P^{(n)}(0)}{n!} t^n.$$

Comparing the coefficients of t^k and invoking Theorem 9 yields

$$a_k = \frac{P^{(k)}(0)}{k!} = \binom{n}{k} p(\underbrace{0, \dots, 0}_{n-k}, \underbrace{\delta, \dots, \delta}_k), \quad k = 0, \dots, n, \quad (4.4)$$

or equivalently

$$p(\underbrace{0, \dots, 0}_{n-k}, \underbrace{\delta, \dots, \delta}_k) = \frac{a_k}{\binom{n}{k}},$$

where $\delta = (1, 0)$. Equation (4.4) is reminiscent of Equation (3.1) for the interval $[0, 1]$

$$P_k = p(\underbrace{0, \dots, 0}_{n-k}, \underbrace{1, \dots, 1}_k) \quad k = 0, \dots, n,$$

which gives the Bezier control points of $P(t)$ over the interval $[0,1]$. Thus to convert between monomial and Bezier form, we need only convert from one set of blossom values to another. We illustrate these conversion algorithms for cubic curves in Figure 11; these algorithms generalize to arbitrary degree in the obvious manner. Notice that to convert between monomial and Bezier form, the monomial coefficients must be normalized by multiplying or dividing by binomial coefficients.

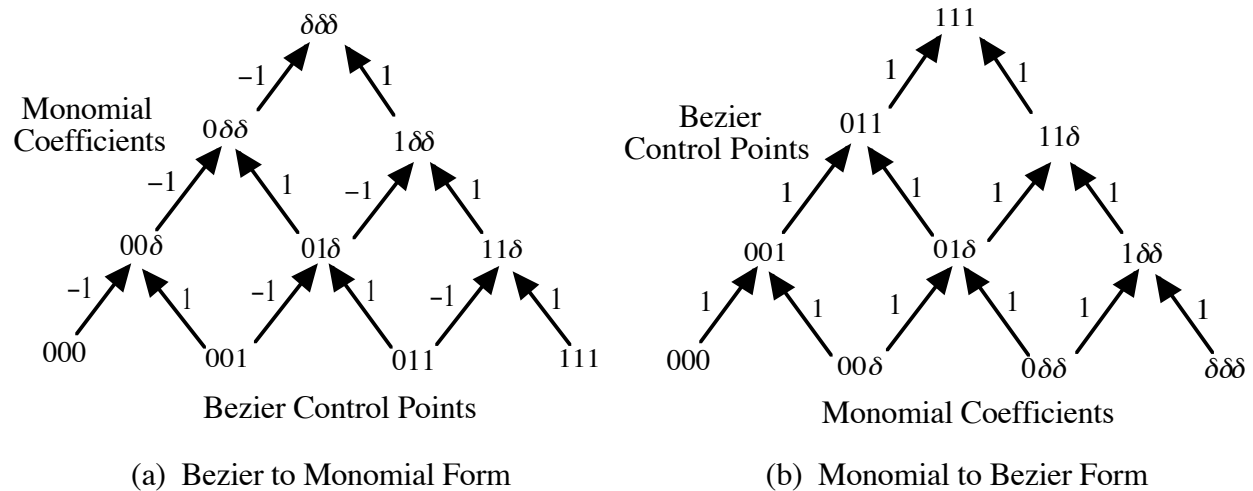


Figure 11: Conversion between cubic Bezier and cubic monomial form. In (a) we convert from Bezier to monomial form; in (b) we convert from monomial to Bezier form. Notice that the labels along the edges in these diagrams do not need to be normalized because $1 = (1,1) = (0,1) + (1,0) = 0 + \delta$. Thus we subtract at each node to get from Bezier control points to monomial coefficients, and we add at each node to get from monomial coefficients to Bezier control points. These algorithms work in arbitrary degree. Notice that the diagram for the algorithm to convert from Bezier to monomial form is the same as the diagram for the algorithm to compute the n th derivative of a Bezier curve over the interval $[0,1]$.

5. Summary

Blossoming is a potent tool for deriving properties of Bezier curves. In this lecture we applied blossoming to derive the de Casteljau subdivision algorithm and the de Casteljau differentiation algorithm for Bezier curves, as well as algorithms to convert between monomial and Bezier form. B-splines can also be conveniently investigated from the perspective of blossoming; we shall investigate B-splines in our next lecture.

Blossoming is related to Bezier curves because each node in the de Casteljau algorithm has an interpretation in terms of a blossom value. There are also several important variants of the de Casteljau algorithm that can be generated by straightforward substitutions.

Blossoming:

$t \rightarrow u_k$ on the k^{th} level

Homogenization

$t - a \rightarrow t - aw$

$b - t \rightarrow bw - t$ on every level

Differentiation (k^{th} derivative)

$t - a \rightarrow 1$
 $b - t \rightarrow -1$ on k levels and multiply the output by $\frac{n!}{(n-k)!}$

We illustrate the blossoming interpretation of the de Casteljau algorithm along with these three variants of the de Casteljau algorithm for cubic Bezier curves in Figure 12-15. As usual, we use the multiplicative notation uvw to represent the blossom value $p(u,v,w)$.

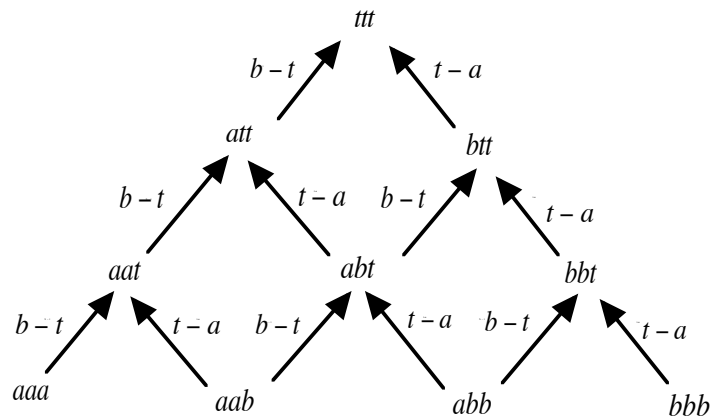


Figure 12: The de Casteljau evaluation algorithm for cubic Bezier curves with each node interpreted as a blossom value.

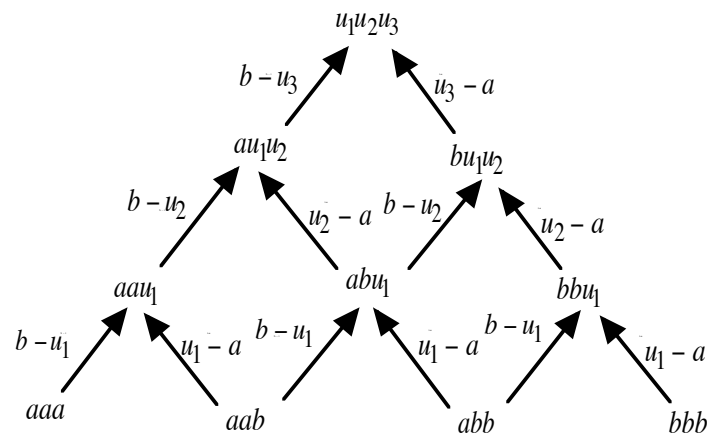


Figure 13: The blossomed version of the de Casteljau algorithm for cubic Bezier curves.

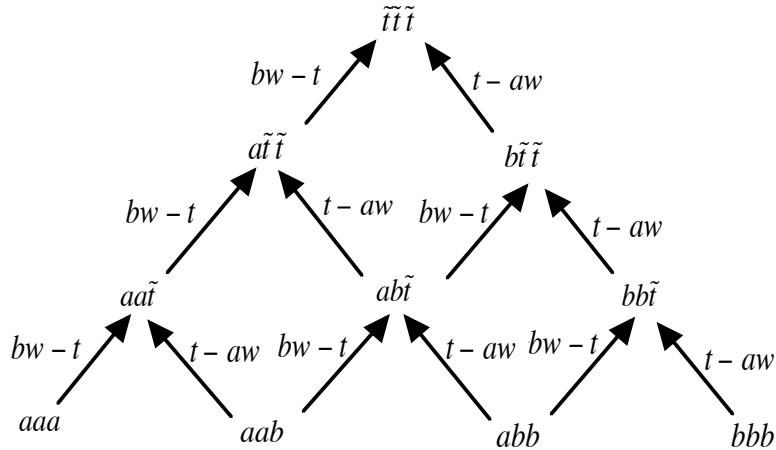


Figure 14: The homogeneous version of the de Casteljau algorithm for cubic Bezier curves: $\tilde{t} = (t, w)$

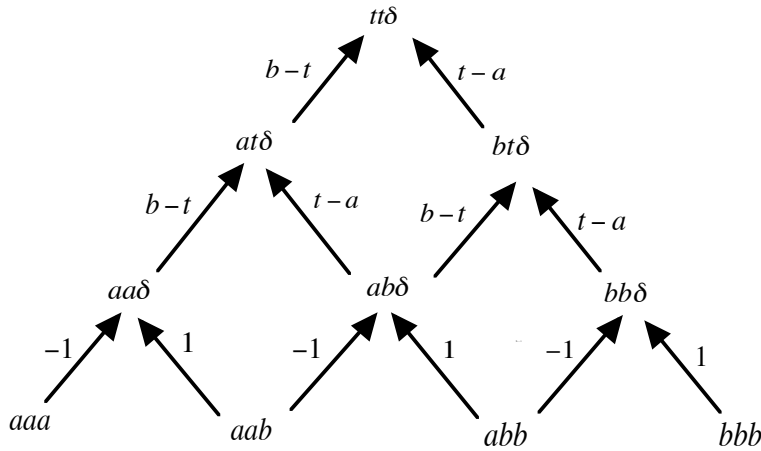


Figure 15: The differentiated version of the de Casteljau algorithm for cubic Bezier curves: $\delta = (1, 0)$.

We close with a summary of the primary properties of the blossom. The first three properties are the blossoming axioms -- the three properties that uniquely characterize the blossom. Property *iv* is the key connection between blossoming and Bezier curves, and Property *v* is the connection between blossoming and differentiation.

Primary Properties of the Blossom

i. *Symmetry*

$$p(u_1, \dots, u_n) = p(u_{\sigma(1)}, \dots, u_{\sigma(n)}) \text{ for every permutation } \sigma \text{ of } \{1, \dots, n\}.$$

ii. *Multiaffine*

$$p(u_1, \dots, (1 - \alpha)u_k + \alpha v_k, \dots, u_n) = (1 - \alpha)p(u_1, \dots, u_k, \dots, u_n) + \alpha p(u_1, \dots, v_k, \dots, u_n)$$

iii. *Diagonal*

$$P(t) = p(\underbrace{t, \dots, t}_n)$$

iv. *Dual Functional*

$$P_k = p(\underbrace{a, \dots, a}_{n-k}, \underbrace{b, \dots, b}_k) = \textit{kth Bezier Control Point}$$

v. *Differentiation*

$$P^{(k)}(t) = \frac{n!}{(n-k)!} p(\underbrace{t, \dots, t}_{n-k}, \underbrace{\delta, \dots, \delta}_k) \quad \delta = (1, 0)$$

Exercises:

1. Let $P(t)$ and $Q(t)$ be polynomials of degree n . Prove that:
 - a. If $R(t) = P(t) + Q(t)$, then $r(u_1, \dots, u_n) = p(u_1, \dots, u_n) + q(u_1, \dots, u_n)$.
 - b. If $S(t) = cP(t)$, then $s(u_1, \dots, u_n) = cp(u_1, \dots, u_n)$.
2. Let $P(t) = (t-a)^n$. Show that $p(u_1, \dots, u_n) = (u_1 - a) \cdots (u_n - a)$.
3. Let $P_k^n(t) = \frac{(t-t_k)^n}{\prod_{j \neq k} (t_j - t_k)}$, $k = 0, \dots, n$, and let $P(t) = \sum_{k=0}^n P_k^n(t) P_k$. Using the result of Exercise 2, show that $P_k = p(t_0, \dots, t_{k-1}, t_{k+1}, \dots, t_n)$.
4. Let $p(u_1, \dots, u_n)$ be a polynomial in which each variable appears to at most the first power. Show that $p(u_1, \dots, u_n)$ is multiaffine.
5. Let $p((u_1, v_1), \dots, (u_n, v_n))$ be a polynomial in which either u_k or v_k , but not both, appears in every term to the first power for $k = 1, \dots, n$. Show that $p((u_1, v_1), \dots, (u_n, v_n))$ is multilinear.
6. Let $u_1 \cdots u_n$ denote the polynomial $(u_1 - t) \cdots (u_n - t)$. Show that with this interpretation the algorithms represented by Figures 1 and 3 remain valid.
7. Let $P(t, w)$ and $Q(t, w)$ be two homogeneous polynomials of degree n . Show that
 - a. $\frac{P(ct, cw)}{Q(ct, cw)} = \frac{P(t, w)}{Q(t, w)}$
 - b. $P(t, w) = w^n P(t/w, 1)$.

8. In this exercise, you will complete the proof of Theorem 8.
- Use induction to prove the binomial theorem

$$(a + b)^n = \sum_{k=0}^n \binom{n}{k} a^k b^{n-k} .$$

- Mimic the inductive proof of the binomial theorem in part *a* to prove that

$$p(t + h\delta, \dots, t + h\delta) = \sum_{k=0}^n \binom{n}{k} p(\underbrace{t, \dots, t}_{n-k}, \underbrace{\delta, \dots, \delta}_k) h^k .$$

9. Let $P(t)$ be a polynomial of degree n over the interval $[a, b]$ with Bernstein coefficients P_0, \dots, P_n . Let $G(P)(t) = (t, P(t))$ denote the graph of the polynomial $P(t)$. Show that the Bezier control points of $G(P)(t)$ over the interval $[a, b]$ are given by

$$Q_k = \left(a + \frac{k(b-a)}{n}, P_k \right) \quad k = 0, \dots, n .$$

10. Let $M_k^n(t, a) = (t - a)^k$, $k = 0, \dots, n$ denote the monomial basis at $x = a$. Develop algorithms to convert between:

- the monomial basis at $x = a$ and the Bernstein basis over the interval $[a, b]$.
- the monomial basis at $x = 0$ and the monomial basis at $x = a$.
- the Bernstein basis over the interval $[a, b]$ and the monomial basis at $x = 0$.

11. Using the results of Exercises 9, 10c and the intersection algorithm for Bezier curves in Lecture 25. Section 3.1, develop an algorithm for finding the roots of a polynomial lying in the interval $[a, b]$.

12. Let $B_k^n(t)$ denote the Bernstein basis functions over the interval $[a, b]$ and let $L(t) = c t + d$ be a linear function.

- Using the dual functional property, show that

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

- $\sum_{k=0}^n \left(a + \frac{k}{n}(b-a) \right) B_k^n(t) \equiv t .$

- Using the result of part a, show that

$$\sum_{k=0}^n L \left(a + \frac{k}{n}(b-a) \right) B_k^n(t) \equiv L(t) .$$

13. Let $B_k^n(t)$ denote the Bernstein basis functions over the interval $[0,1]$. Using the dual functional property, prove the following identities:

a.
$$\sum_{k=0}^n \binom{k}{j} B_k^n(t) \equiv \binom{n}{j} t^j \quad j = 0, \dots, n$$

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

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

14. Let $P(t)$ be a Bezier curve of degree n over the interval $[a,b]$. Every polynomial of degree n is also a polynomial of degree $n+1$. Let $p_n(u_1, \dots, u_n)$ denote the blossom of $P(t)$ as a polynomial of degree n and let $p_{n+1}(u_1, \dots, u_{n+1})$ denote the blossom of $P(t)$ as a polynomial of degree $n+1$. In addition, let P_0, \dots, P_n be the Bezier control points for $P(t)$ as a polynomial of degree n and let Q_0, \dots, Q_{n+1} be the Bezier control points for $P(t)$ as a polynomial of degree $n+1$. Show that:

a.
$$p_{n+1}(u_1, \dots, u_{n+1}) = \sum_{k=1}^{n+1} \frac{p_n(u_1, \dots, u_{k-1}, u_{k+1}, \dots, u_{n+1})}{n+1}$$

b.
$$Q_k = \frac{k}{n+1} P_{k-1} + \frac{n+1-k}{n+1} P_k \quad k = 0, \dots, n+1.$$