Lecture 22: Bezier Subdivision

And he took unto him all these, and divided them in the midst, and laid each piece one against another: Genesis 15:10

1. Bezier Subdivision

The control points of a Bezier curve typically describe the curve with respect to the parameter interval $[0,1]$. Sometimes, however, we are interested only in the part of the curve in some subinterval $[a,b]$. For example, when rendering a Bezier curve we may need to clip the curve to a window. Since any segment of a Bezier curve is a polynomial curve, we should be able to represent such a segment as a complete Bezier curve with a new set of control points. Splitting a Bezier curve into smaller pieces is also useful as a divide and conquer strategy for intersection algorithms (see Section 3). The process of splitting a Bezier curve into two or more Bezier curves that represent the exact same curve is called subdivision (see Figure 1)

![Figure 1: Subdivision. The Bezier curve with control points $P_0,\ldots,P_n$ is split into two Bezier curves: the left Bezier segment has control points $Q_0,\ldots,Q_n$; the right Bezier segment has control points $R_0,\ldots,R_n$.](image)

2. The de Casteljau Subdivision Algorithm

The main problem in Bezier subdivision is: given a collection of control points $P_0,\ldots,P_n$ that describe a Bezier curve over the interval $[0,1]$, find the control points $Q_0,\ldots,Q_n$ and $R_0,\ldots,R_n$ that describe the segments of the same Bezier curve over the intervals $[0,r]$ and $[r,1]$. Remarkably the solution to this problem is provided by the same de Casteljau algorithm that we use to evaluate points on a Bezier curve.
De Casteljau’s Subdivision Algorithm

Let \( B(t) \) be a Bezier curve with control points \( P_0, \ldots, P_n \). To subdivide \( B(t) \) at \( t = r \), run the de Casteljau algorithm at \( t = r \). The points \( Q_0, \ldots, Q_n \) that emerge along the left lateral edge of the triangle are the Bezier control points of the segment of the curve from \( t = 0 \) to \( t = r \), and the points \( R_0, \ldots, R_n \) that emerge along the right lateral edge of the triangle are the Bezier control points of the segment of the curve from \( t = r \) to \( t = 1 \) (see Figure 2).

We shall defer the proof of the de Casteljau subdivision algorithm till Lecture 23. In this lecture we will study the properties and the consequences of this subdivision algorithm. Figure 3 provides a geometric interpretation for the de Casteljau subdivision algorithm.

Figure 2: The de Casteljau subdivision algorithm for a cubic Bezier curve with control points \( P_0, P_1, P_2, P_3 \). The points \( Q_0, Q_1, Q_2, Q_3 \) that emerge along the left edge of the triangle are the Bezier control points of the segment of the original curve from \( t = 0 \) to \( t = r \); the points \( R_0, R_1, R_2, R_3 \) that emerge along the right edge of the triangle are the Bezier control points of the segment of the original curve from \( t = r \) to \( t = 1 \).

Figure 3: Geometric interpretation of the de Casteljau subdivision algorithm for a cubic Bezier curve with control points \( P_0, P_1, P_2, P_3 \). The points \( Q_0, Q_1, Q_2, Q_3 \) are the Bezier control points of the segment of the original curve from \( t = 0 \) to \( t = r \), and the points \( R_0, R_1, R_2, R_3 \) are the Bezier control points of the segment of the original curve from \( t = r \) to \( t = 1 \).
If we continue recursively subdividing a Bezier curve, the control polygons get closer and closer to the original curve. Moreover, in the limit, these control polygons converge to a continuous curve (see Figure 4). We shall now show that this limit curve is, in fact, the original Bezier curve. For convenience, we restrict our attention to recursive subdivision at \( r = 1/2 \), though the results are much the same for any value of \( r \) between 0 and 1.

**Theorem 1:** The control polygons generated by recursive subdivision converge to the original Bezier curve.

Proof: Suppose that the maximum distance between any two adjacent control points is \( d \). By construction, the points on any level of the de Casteljau algorithm for \( t = 1/2 \) lie at the midpoints of the edges of the polygons generated by the previous level. Therefore it follows easily by induction that adjacent points on any level of the de Casteljau diagram for \( t = 1/2 \) are no further than \( d \) units apart (see Figure 5).

By the same midpoint argument, as we proceed up the diagram adjacent points along the left or right lateral edge of the triangle can be no further than \( d/2 \) units apart. Hence as we apply recursive subdivision, the distance between the control points of any single control polygon must converge to zero. Since the first and last control points of a Bezier control polygon always lie on the curve, these control polygons must converge to points along the original curve. ♦
We can also subdivide tensor product Bezier patches. Recall from Lecture 21 that a Bezier surface \( B(s,t) \) is defined by applying the de Casteljau algorithm in the parameter \( s \) to a collection of Bezier curves \( P_0(t), \ldots, P_m(t) \) in the parameter \( t \). To subdivide the Bezier patch along the parameter line \( t = t_0 \), simply subdivide each of the curves \( P_0(t), \ldots, P_m(t) \) at \( t = t_0 \). Similarly, to subdivide the Bezier patch along a parameter line \( s = s_0 \) recall that the same Bezier surface \( B(s,t) \) is defined by applying the de Casteljau algorithm in the parameter \( t \) to a collection of Bezier curves \( P_0^*(s), \ldots, P_n^*(s) \) in the parameter \( s \). Thus to subdivide the Bezier patch along the parameter line \( s = s_0 \), simply subdivide each of the curves \( P_0^*(s), \ldots, P_n^*(s) \) at \( s = s_0 \). Now we have the following result.

**Theorem 2:** The control polyhedra generated by recursive subdivision converge to the original tensor product Bezier surface provided that the subdivision is done in both the \( s \) and \( t \) directions.

The proof of Theorem 2 for Bezier surfaces is much the same as the proof of Theorem 1 for Bezier curves, so we shall not repeat the proof here. Notice, however, that we must be careful to subdivide in both the \( s \) and the \( t \) directions. If we subdivide only along one parameter direction, convergence is not assured because in the limit the control polyhedra will not shrink to a single point. Thus to assure convergence it is best to alternate subdividing in different parameter directions.

### 3. Rendering and Intersection Algorithms

Subdivision is the main mathematical tool for analyzing Bezier curves and surfaces. Here we shall apply recursive subdivision to develop algorithms for rendering and intersecting Bezier curves and surfaces.

#### 3.1 Rendering and Intersecting Bezier Curves

The convergence of recursive subdivision is quite fast. This rapid convergence leads to the following important recursive algorithms for rendering and intersecting Bezier curves.

**Rendering Algorithm -- Bezier Curves**

If the Bezier curve can be approximated to within tolerance by the straight line segment joining its first and last control points,

- then draw either this line segment or the control polygon.
- Otherwise *subdivide* the curve (at \( r = 1/2 \)) and render the segments recursively.
Intersection Algorithm -- Bezier Curves

If the convex hulls of the control points of two Bezier curves fail to intersect, then the curves themselves do not intersect. Otherwise if each Bezier curve can be approximated by the straight line segment joining its first and last control points, then intersect these line segments. Otherwise \textit{subdivide} the two curves and intersect the pieces recursively.

To determine whether a Bezier curve can be approximated to within tolerance by a straight line segment joining its first and last control points, it is sufficient, by the convex hull property, to test whether all the interior control points lie within tolerance of this line segment. By the Pythagorean theorem (see Figure 6), the distance between a point \( P \) and a line \( L \) determined by a point \( Q \) and a unit direction vector \( v \) is given by

\[
\text{dist}^2(P, L) = |P - Q|^2 - ((P - Q) \cdot v)^2.
\]

\[ \text{Figure 6:} \quad \text{A line } L \text{ determined by a point } Q \text{ and a unit direction vector } v, \text{ and a point } P \text{ not on } L. \]

By the Pythagorean theorem \( \text{dist}^2(P, L) = |P - Q|^2 - ((P - Q) \cdot v)^2 \).

Be careful. It may happen that a control point \( P_k \) is close to the line \( L \) determined by the first and last control points \( P_0, P_n \) even though the projection of \( P_k \) does not lie inside the line segment \( P_0P_n \) -- that is, \( P_k \) may be close to the line \( L \) even though it is not close to the line segment \( P_0P_n \). To be sure that this is not the case, one need only check that \( 0 \leq (P_k - P_0) \cdot (P_n - P_0) \leq |P_n - P_0|^2 \).

It is relatively easy to test whether or not a Bezier curve can be approximated to within some tolerance by a straight line segment. On the other hand, finding and intersecting the convex hulls of two Bezier curves can be quite difficult and time consuming. In practice, the convex hulls in the intersection algorithm are replaced by bounding boxes which are much easier to compute and intersect than the actual convex hulls: simply take the minimum and maximum \( x \) and \( y \) coordinates of the control points. Since the subdivision algorithm converges rapidly, not much time is lost by replacing convex hulls with bounding boxes.
To complete the intersection algorithm for Bezier curves, we need to be able to intersect two line segments:

\[ L_1(s) = (1 - s)P_0 + sP_1 = P_0 + s(P_1 - P_0) \quad 0 \leq s \leq 1 \]

\[ L_2(t) = (1 - t)Q_0 + tQ_1 = Q_0 + t(Q_1 - Q_0) \quad 0 \leq t \leq 1 \]

The two infinite lines intersect when \( L_1(s) = L_2(t) \) -- that is, when

\[ P_0 + s(P_1 - P_0) = Q_0 + t(Q_1 - Q_0) \]

or equivalently when

\[ s(P_1 - P_0) - t(Q_1 - Q_0) = (Q_0 - P_0) \]

Dotting both sides first with \( P_1 - P_0 \) and then with \( Q_1 - Q_0 \) generates two linear equations in two unknowns.

\[
\begin{align*}
\begin{aligned}
    s((P_1 - P_0) \cdot (P_1 - P_0)) - t((Q_1 - Q_0) \cdot (P_1 - P_0)) &= (Q_0 - P_0) \cdot (P_1 - P_0) \\
    s((P_1 - P_0) \cdot (Q_1 - Q_0)) - t((Q_1 - Q_0) \cdot (Q_1 - Q_0)) &= (Q_0 - P_0) \cdot (Q_1 - Q_0)
\end{aligned}
\end{align*}
\]  

(3.1)

which are easy to solve for the parameters \( s, t \). The intersection point lies on the two line segments if and only \( 0 \leq s, t \leq 1 \). In this case we can compute \( L_1(s) \) or \( L_2(t) \) to find the actual intersection point. By the way if the lines are skew, then the solutions to Equation (3.1) still give the parameter values of the two closest points on the two lines (see Exercise 4).

### 3.2 Rendering and Intersecting Bezier Surfaces.

The rendering and intersection algorithms for Bezier curves can be extended to rendering and intersection algorithms for Bezier surfaces. Line are replaced by planes, line segments are replaced by triangles, and recursive subdivision for Bezier curves is replaced by recursive subdivision for Bezier surfaces, but the basic structure of these algorithms for Bezier surfaces is much the same as the corresponding algorithms for Bezier curves.

**Rendering Algorithm -- Bezier Surfaces**

If the Bezier surface can be approximated to within tolerance by two triangles each joining three of its four corner points,

then draw the two triangles determined by the four corner points.

Otherwise *subdivide* the surface (at \( s = 1/2 \) or \( t = 1/2 \)) and render the segments recursively.

**Intersection Algorithm -- Bezier Surfaces**

If the convex hulls of the control points of two Bezier surfaces fail to intersect,

then the surfaces themselves do not intersect.

Otherwise if each Bezier surface can be approximated by two triangles each joining three of its four corner points,

then intersect the corresponding triangles.

Otherwise *subdivide* the two surfaces and intersect the segments recursively.
To determine whether a Bezier surface can be approximated to within tolerance by two triangles each joining three of its four corner points, it is sufficient, by the convex hull property, to test whether all the interior control points lie within tolerance of these two triangles. By straightforward trigonometry (see Figure 7), the distance between a point $P$ and a plane $S$ determined by a point $Q$ and a unit normal vector $N$ is given by

$$\text{dist}(P, S) = |P - Q| \cos \theta = |(P - Q) \cdot N|.$$ 

**Figure 7:** A plane $S$ determined by a point $Q$ and a unit normal vector $N$, and a point $P$ not on $S$. By the trigonometry $\text{dist}(P, S) = |P - Q| \cos \theta = |(P - Q) \cdot N|.$

Again be careful. It may happen that a control point $P_{ij}$ lies close to the plane $S$ of a triangle $\Delta$ even though the projection of the point $P_{ij}$ onto the plane $S$ does not lie inside the triangle $\Delta$ -- that is, $P_{ij}$ may be close to the plane $S$ even though $P_{ij}$ is not close to the triangle $\Delta$. To be sure that this is not the case, one need only check that the projection $P_{ij} - [(P_{ij} - Q) \cdot N] N$ lies inside $\Delta$ (see Exercise 5).

Finding and intersecting the convex hulls of two Bezier surfaces can be quite difficult and time consuming. In practice, as with curves, the convex hulls in the intersection algorithm are replaced by bounding boxes. Again since the subdivision algorithm converges rapidly, not much time is lost by replacing convex hulls with bounding boxes.

To complete the intersection algorithm for Bezier surfaces, we need to be able to intersect two non-coplanar triangles. Suppose that these triangles lie in the two planes

$$N_1 \cdot (P - Q_1) = 0$$
$$N_2 \cdot (P - Q_2) = 0.$$

These two infinite planes intersect in a straight line. We can specify this line $L$ by a point $Q$ and a direction vector $N$. Since the line $L$ lies in both planes, the direction vector $N$ must be perpendicular
to the normal to both planes. Therefore we can choose
\[ N = N_1 \times N_2. \]
To find a point \( P = (x, y, z) \) on the line \( L \), we need to solve two linear equations in three unknowns:
\[
N_1 \cdot (P - Q_1) = 0 \\
N_2 \cdot (P - Q_2) = 0.
\]
There are infinitely many solutions to these two equations, since there are infinitely many points \( P \) on the line \( L \). To find a unique solution and make the problem deterministic, we can add the equation of any plane that intersects the line \( L \). Since the vector \( N_1 \times N_2 \) is parallel to the direction of the line \( L \), any plane with normal vector \( N_1 \times N_2 \) will certainly intersect the line \( L \). Therefore we can choose any point \( Q_3 \), and find a unique point \( P \) on the line \( L \) by solving three linear equations in three unknowns
\[
N_1 \cdot (P - Q_1) = 0 \\
N_2 \cdot (P - Q_2) = 0 \\
(N_1 \times N_2) \cdot (P - Q_3) = 0
\]
for the coordinates of the point \( P \) (see also Lecture 18, Exercise 2).

4. The Variation Diminishing Property of Bezier Curves

We introduced Bezier curves because, unlike Lagrange interpolation, Bezier approximation does not introduce oscillations not already present in the data. This property of Bezier curves is called the variation diminishing property. We are now going to use Bezier subdivision to derive the variation diminishing property of Bezier curves.

To prove that Bezier curves never oscillate any more than their control polygons, we need a way to quantify oscillation. One way to measure oscillations is to compute intersections with straight lines. The number of intersections of a curve with a line tells us how much the curve oscillates with respect to the line (see Figure 8).

![Figure 8: The number of intersections of a curve with a straight line counts the number of oscillations of the curve with respect to the line.](image)
A curve scheme $B(P)(t)$ for a control polygon $P$ is said to be variation diminishing if for every line $L$

\[ \# \text{intersections of } B(P)(t) \text{ and } L \leq \# \text{intersections of } P \text{ and } L. \]

Figure 9 illustrates the difference between variation diminishing and non-variation diminishing curve schemes.

**Figure 9:** The curve scheme on the left is variation diminishing because the number of intersections of the curve $C$ and any line $L$ is never greater than the number of intersections of the control polygon and the line. The curve scheme on the right is not variation diminishing because the curve $D$ and the line $L$ intersect in three points, while the control polygon and the line $L$ intersect only twice.

One way to introduce variation diminishing schemes is by corner cutting. Start with a polygon $P$ and form another polygon $Q$ by cutting a corner off $P$ (see Figure 10). Then it is easy to see that every line $L$ intersects $P$ at least as often as $L$ intersects $Q$. Thus $Q$ is variation diminishing with respect to $P$.

**Figure 10:** Corner cutting. The polygon with vertices $Q = \{Q_i\}$ is generated from the polygon with vertices $P = \{P_j\}$ by cutting off the corner at $P_1$. Every line $L$ intersects $P$ at least as often as $L$ intersects $Q$ because if $L$ intersects the line segment $Q_1Q_2$, then $L$ must intersect either $P_0P_1$ or $P_1P_2$.  

9
Now look at the geometric interpretation of the de Casteljau subdivision algorithm in Figure 3. Evidently, the de Casteljau subdivision algorithm is a corner cutting procedure: first the corners at \( P_1, P_2 \) are cut off, then the corner at \( \Delta \) is cut off (see Figure 11).

![Figure 11: The de Casteljau subdivision algorithm as a sequence of corner cuts. In the first stage corners are cut off at \( P_1, P_2 \); in the second stage the corner is cut off at \( \Delta \). Compare to Figure 3.](image)

Although we have illustrated only the cubic case, it is easy to verify that the de Casteljau subdivision algorithm is a corner cutting procedure in all degrees. This observation leads to the following result.

**Theorem 3:** Bezier curves are variation diminishing.

Proof: Since recursive subdivision is a corner cutting procedure, the limit curve must be variation diminishing with respect to the original control polygon. But by Theorem 1, the Bezier curve is the limit curve generated by recursive subdivision, so Bezier curves are variation diminishing.

By the way, there is no known analogue of the variation diminishing property for Bezier surfaces, since for Bezier surfaces subdivision is not a corner cutting procedure.

5. Joining Bezier Curves Smoothly

In Lecture 21 we used our procedure for differentiating the de Casteljau algorithm for Bezier curves to derive constraints on the location of the control points to insure that two Bezier curves meet smoothly where they join. Given a Bezier curve with control points \( P_0, \ldots, P_n \), we derived the following constraints for the location of the control points \( Q_0, \ldots, Q_n \) of another Bezier curve that meets the first curve and matches its first \( k \) derivatives at its initial point:
\[ k = 0 \implies Q_0 = P_n \]
\[ k = 1 \implies Q_1 = P_n + (P_n - P_{n-1}) \]
\[ k = 2 \implies Q_2 = P_{n-2} + 4(P_n - P_{n-1}) \]

(5.1)

We observed that each derivative generates one new constraint on one additional control point. These constraints are easy to solve, but rather cumbersome to write down. Here we show how to use subdivision to determine the location of the points \( Q_0, \ldots, Q_k \) that guarantee \( k \)-fold continuity at the join.

We know that the location of the points \( Q_0, \ldots, Q_k \) is unique. If we could find a curve \( Q(t) \) that meets the original curve \( P(t) \) smoothly at the join, then the control points of \( Q(t) \) would necessarily give the location of the points \( Q_0, \ldots, Q_k \). But we certainly do know such a curve, for \( P(t) \) meets itself smoothly at the join! Suppose that \( P(t) \) is parameterized over the interval \([0,1]\). Let \( Q(t) \) be \( P(t) \) over the interval \([1,2]\). Then \( P(t) \) and \( Q(t) \) meet smoothly at \( t = 1 \). All we need now are the Bezier control points of \( Q(t) \).

To find the Bezier control points of \( P(t) \) over the intervals \([0,r]\) and \([r,1]\), we can subdivide the Bezier curve \( P(t) \) at \( t = r \). Nothing in our subdivision algorithm requires that \( 0 \leq r \leq 1 \). If we take \( r = 2 \), then the de Casteljau subdivision algorithm will generate the Bezier control points for the curve \( P(t) \) over the intervals \([0,2]\) and \([2,1]\). By the symmetry property of Bezier curves, the Bezier control points for the interval \([2,1]\) are the same as the Bezier control points for the interval \([1,2]\) but in reverse order. Thus we can read off the points \( Q_0, \ldots, Q_k \) from the right edge of the de Casteljau subdivision algorithm for \( r = 2 \). We illustrate this algorithm in Figure 12 for cubic Bezier curves.

**Figure 12:** The de Casteljau subdivision algorithm at \( r = 2 \) for cubic Bezier curves. The first \( k + 1 \) points \( Q_0, \ldots, Q_k \) that emerge on the right lateral edge of the diagram are the control points that guarantee \( k \)-fold continuity at the join. Compare to the points to the values in Equations (5.1).
6. Summary

Subdivision is the main technical tool we have for analyzing Bezier curves and surfaces. Subdivision can be applied as a divide and conquer technique for rendering and intersecting Bezier curves and surfaces.

De Casteljau’s evaluation algorithm for Bezier curves is also a subdivision procedure. Since the de Casteljau algorithm is a corner cutting procedure, we can use subdivision to derive the variation diminishing property of Bezier curves. We can also apply subdivision to find the location of the control points that guarantee $k$-fold smoothness of two Bezier curves at their join.

Exercises:

1. Bezier subdivision at $t = 1/2$ generates a binary tree whose nodes are control polygons. Denote the original control polygon by $P$, and let this polygon be the root of the tree. Let $P_0$ -- the left child of $P$ -- denote the control polygon for the left segment of the Bezier curve (from $t = 0$ to $t = 1/2$), and let $P_1$ -- the right child of $P$ -- denote the control polygon for the right portion of the curve (from $t = 1/2$ to $t = 1$). Continue to build this binary tree recursively in this fashion. Thus if $P_b$ is a node in the tree, then $P_b$ represents the control polygon for a portion of the curve, and $P_{b0}$ -- the left child of $P_b$ -- represents the control polygon for the left half of the Bezier segment represented by $P_b$, while $P_{b1}$ -- the right child of $P_b$ -- represents the control polygon for the right half of the Bezier segment represented by $P_b$ (see Figure 13).

![Figure 13: The binary tree of control polygons generated by recursive subdivision of a Bezier curve at $t = 1/2$.](image-url)
a. Prove that $P_{b_1 \ldots b_n}$ is the control polygon for the original Bezier curve from $t = b$ to $t = b + 2^{-n}$, where $b$ is the binary fraction represented by $0.b_1 \cdots b_n$.

b. Prove that the sequence of control polygons $P_{b_1}, P_{b_1b_2}, \ldots, P_{b_1 \ldots b_n}, \ldots$ converges to the point on the Bezier curve at parameter value $b = \lim_{n \to \infty} 0.b_1 \cdots b_n$.

2. Show that to find the control points of the segment of a Bezier curve between $t = r$ and $t = s$, we can subdivide the original Bezier curve at $t = s$ and then subdivide the first segment of the subdivided curve at $t = r/s$.

3. Prove that the arc length of a Bezier curve is less than or equal to the perimeter of its control polygon. Explain how to use this result to compute the arc length of a Bezier curve.

4. Recall that the projection of a point $P$ into a plane $S$ determined by a unit normal vector $N$ and a point $Q$ is given by

$$P_\perp = P - ((P - Q) \cdot N)N.$$  

Using this result, show that for two skew lines, the solutions to Equation (3.1) give the closest points on the two lines.

5. Suppose that the point $P$ lies in the plane of $\Delta Q_1Q_2Q_3$. Give two algorithms for determining if $P$ lies inside of $\Delta Q_1Q_2Q_3$.

6. Develop an algorithm to intersect a Bezier curve with a Bezier surface based on recursive subdivision.

7. Develop an algorithm to ray trace a Bezier surface based on recursive subdivision.