The de Casteljau Algorithm for Evaluating Bezier Curves

From Rockwood "Interactive Curves and Surfaces"
JDill deCasteljau.doc 29Oct00

Evaluating a Bézier curve at a given t gives *P(t)*. As *t* varies from 0 to 1, *P(t)* traces out the curve segment. One way to evaluate the Bezier equation

$$P(t) = \sum_{i=0}^{n} P_i B_{n,i}(t)$$

is simply substitute into the formula and do the calculation.

This is probably the *worst* method of evaluating a point on the curve! Numerical instability, caused by raising small values to high powers, generates errors.

A better way is the de Casteljau algorithm. It is fast and robust, gives insight into Bézier curve behavior and leads to important operations on the curves, such as:

- Computing derivatives (curve derivative gives the tangent vector at a point.)
- Subdividing the curve. It is sometimes necessary to take a single Bézier curve and produce two separate curve segments that together are identical to the original. To accomplish this, it is necessary to find two sets of control points for the two new curves.

The de Casteljau algorithm can be regarded as *repeated linear interpolation.*

Degree 1 "linear" Bezier,

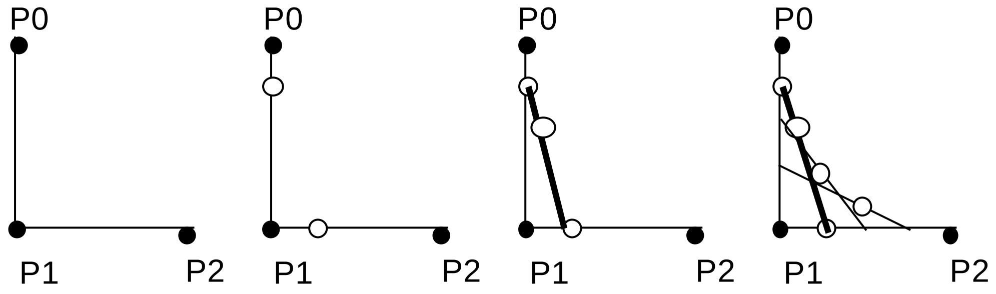
use standard linear interpolation:

$$P(t) = P_0(1-t) + P_1 t$$

Degree 2 (quadratic) Bezier

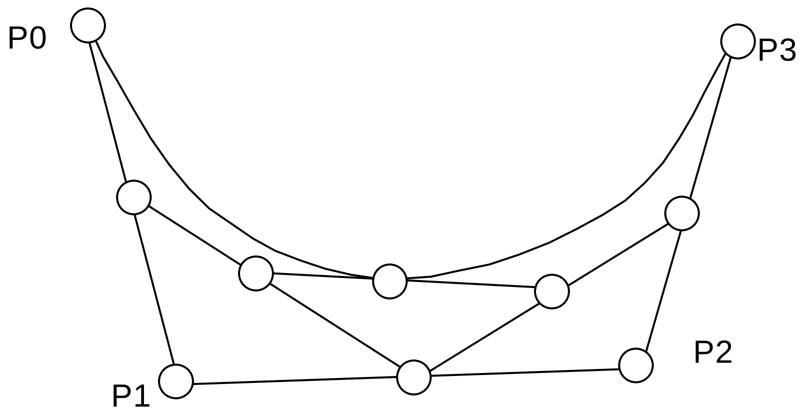
Consider a quadratic Bezier to be determined between P_0, P_1, and P_2,

- linearly interpolate adjacent points, and find the point at t (say t = 0.2)
- then linearly between those points and again find point at t
- repeat for other values of t to trace out the curve



Degree 3 (Cubic) Bezier

Applying the above procedure, with a 3rd linear interpolation gives (for t about 0.5):



de Casteljau's labeling scheme

This formalizes the method:

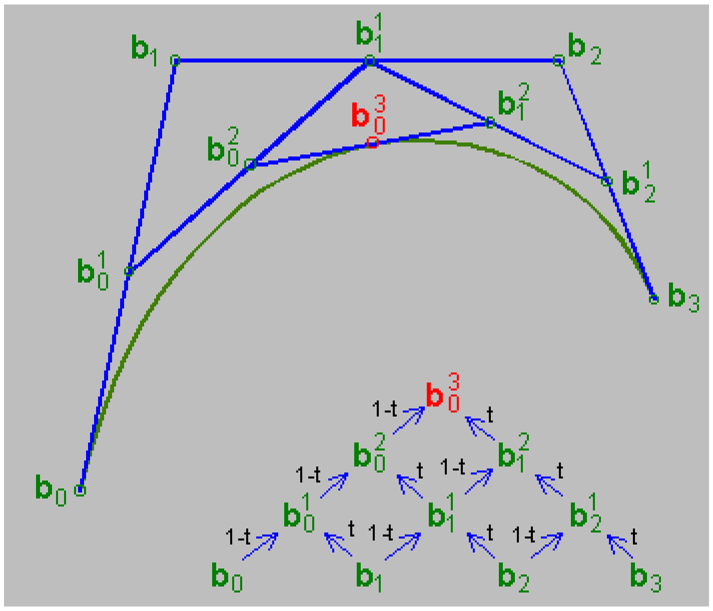
- Each level of recursion denoted by a superscript.
- control points are the zeroth level and do not need to be superscripted.
- Each successive level of recursion has one less point than the previous level.
- The final level $P_0^n(t) = P(t)$ is the point on the curve
- For any point, it can be shown that

$$P_i^j(t) = (1-t)P_i^{j-1} + tP_{i+1}^{j-1}(t) \text{ for } i = 0,..,n; \quad j = 0,1,..,i$$

It is useful to develop a more graphic representation of this equation. Note:

- P30 is point on curve
- Any pt in systolic array linearly interpolates two points in row below at parameter *t*; eg

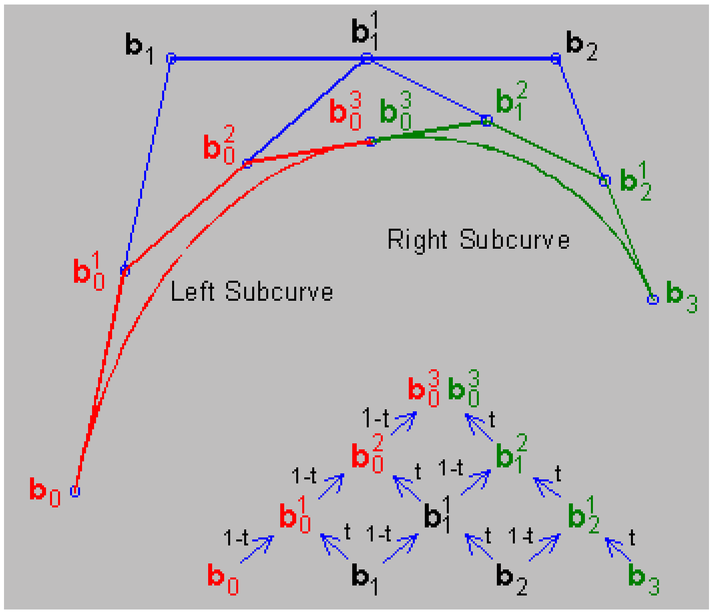
$$P_1^2 = P_1^1(1-t) + P_2^1 t$$



$$\begin{tikzcd}[column sep=small] & & & b_0^3 & & & \\ & & b_0^2 \arrow[ur, "1-t"] & & b_1^2 \arrow[ul, "t"'] & & \\ & b_0^1 \arrow[ur, "1-t"] & & b_1^1 \arrow[ul, "t"'] \arrow[ur, "1-t"] & & b_2^1 \arrow[ul, "t"'] & \\ b_0 \arrow[ur, "1-t"] & & b_1 \arrow[ul, "t"'] \arrow[ur, "1-t"] & & b_2 \arrow[ul, "t"'] \arrow[ur, "1-t"] & & b_3 \arrow[ul, "t"'] \end{tikzcd}$$

Subdivision of a Bézier Curve

An important operation is subdividing it. The de Casteljau algorithm gives a nice way of doing this



- The control points of the two new curves are the sides of the systolic array. The new curves match the original in position, although they differ in parameterization.

Why Subdivide?

Design Refinement

Existing designs can be refined and modified

- e.g. additional curves may be incorporated into an object by adding more control points for local control.

Clipping a Curve to a Boundary

- One method of intersecting a Bézier curve with a line is to recursively subdivide the curve, testing for intersections of the curve's control polygons with the line. Curve segments not intersecting the line are discarded. This process is continued until a sufficiently fine intersection is attained.