COMP3271 Computer Graphics

Curves & Surfaces (III)

2019-20

Objectives

The de Casteljau algorithm for evaluating Bézier curves

Other curves and surfaces:

- Conics & Quadrics
- Extrusion surfaces
- Surface of revolutions
- Sweep surfaces

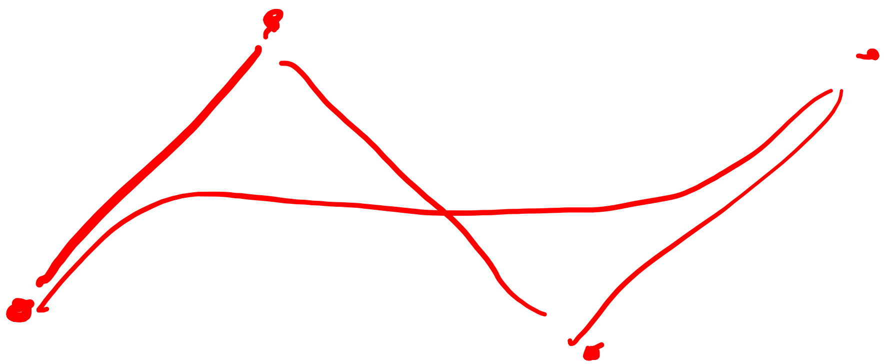
The de Casteljau Algorithm

We can use the convex hull property of Bézier curves to obtain an efficient recursive method that does not require any function evaluations

- Uses only the values at the control points

Based on the idea that "any polynomial and any part of a polynomial is a Bézier polynomial for properly chosen control data"

$$\sum_{i=0}^{3} B_{i,n}(t) P_i$$



The de Casteljau Algorithm

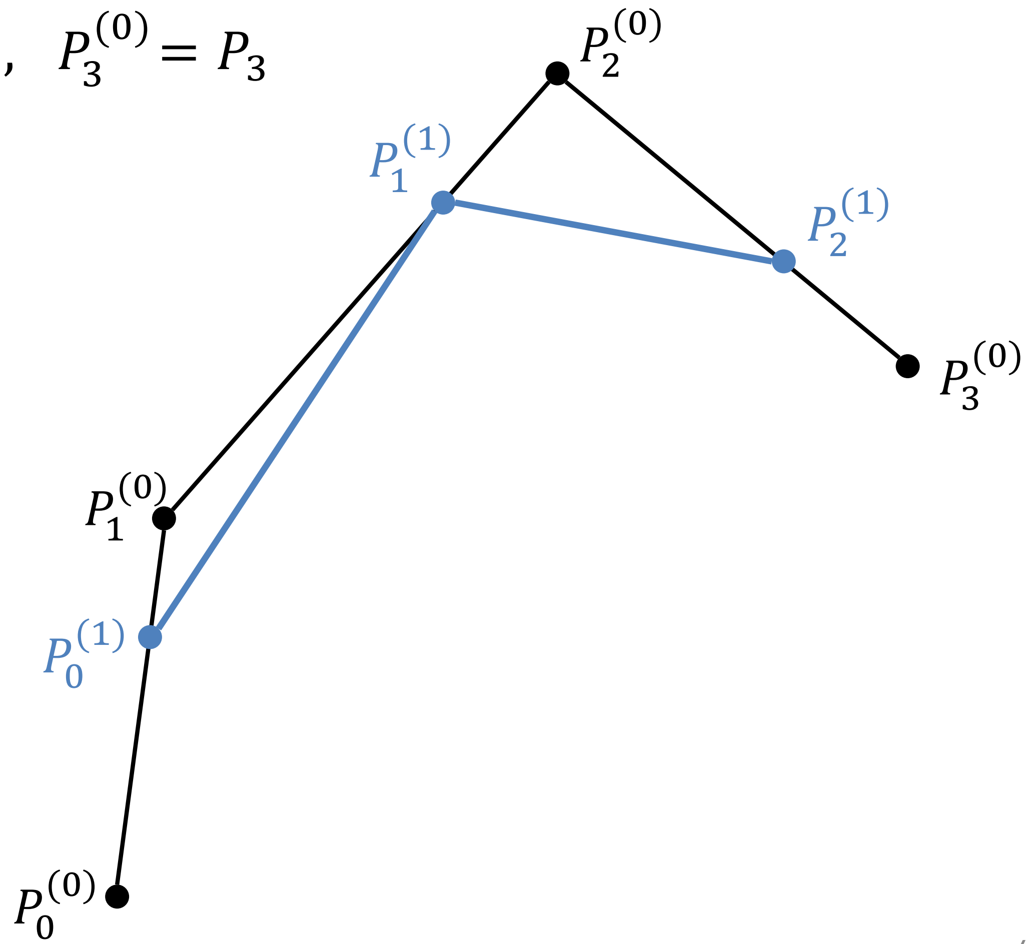
Given a cubic Bézier curve $P(t) = \sum_{i=0}^{3} P_i B_{i,3}(t)$. The following procedure produces a point $P(t)$ on the curve.

$$P_0^{(0)} = P_0,\ P_1^{(0)} = P_1,\ \ P_2^{(0)} = P_2,\ \ P_3^{(0)} = P_3$$

$$P_0^{(1)} = (1-t)P_0 + tP_1$$

$$P_1^{(1)} = (1-t)P_1 + tP_2$$

$$P_2^{(1)} = (1-t)P_2 + tP_3$$



The de Casteljau Algorithm

Given a cubic Bézier curve $P(t) = \sum_{i=0}^{3} P_i B_{i,3}(t)$. The following procedure produces a point $P(t)$ on the curve.

$$P_0^{(0)} = P_0,\ \ P_1^{(0)} = P_1,\ \ P_2^{(0)} = P_2,\ \ P_3^{(0)} = P_3$$

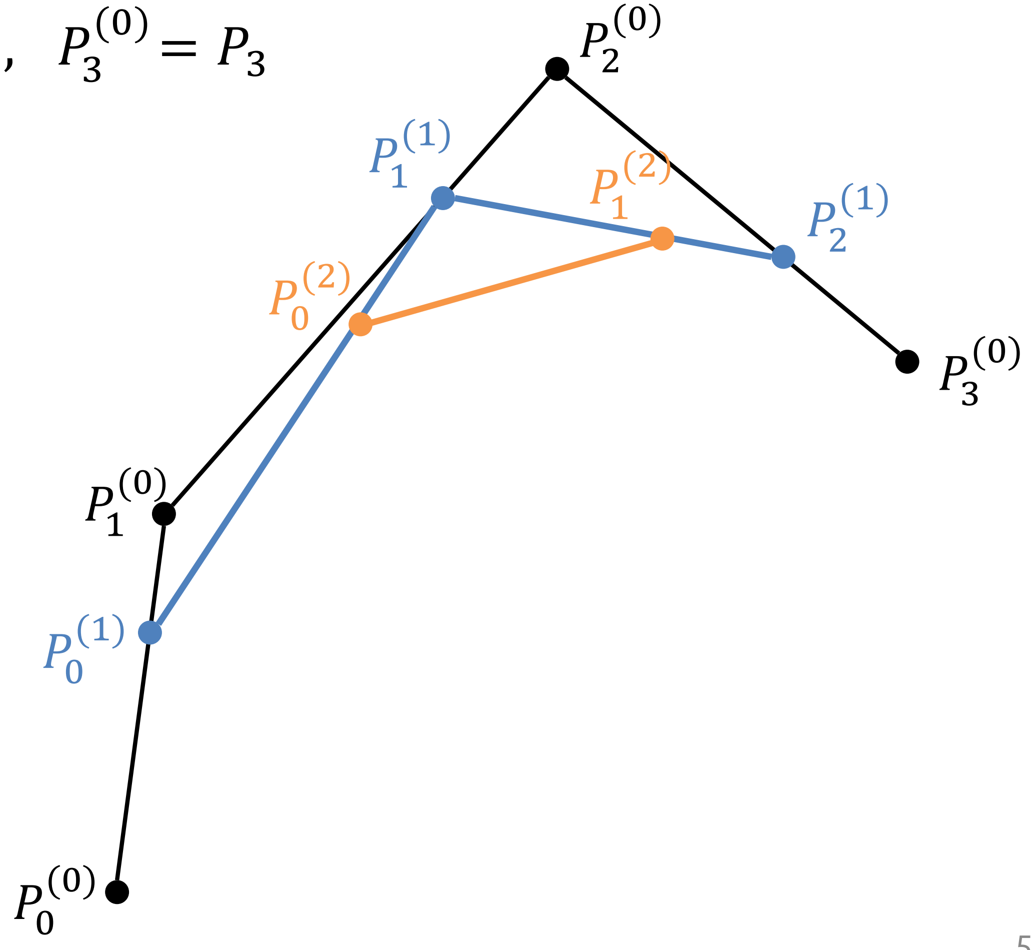
$$P_0^{(1)} = (1-t)P_0 + tP_1$$

$$P_1^{(1)} = (1-t)P_1 + tP_2$$

$$P_2^{(1)} = (1-t)P_2 + tP_3$$

$$P_0^{(2)} = (1-t)P_0^{(1)} + tP_1^{(1)}$$

$$P_1^{(2)} = (1-t)P_1^{(1)} + tP_2^{(1)}$$



The de Casteljau Algorithm

Given a cubic Bézier curve $P(t) = \sum_{i=0}^{3} P_i B_{i,3}(t)$. The following procedure produces a point $P(t)$ on the curve.

$$P_0^{(0)} = P_0,\ \ P_1^{(0)} = P_1,\ \ P_2^{(0)} = P_2,\ \ P_3^{(0)} = P_3$$

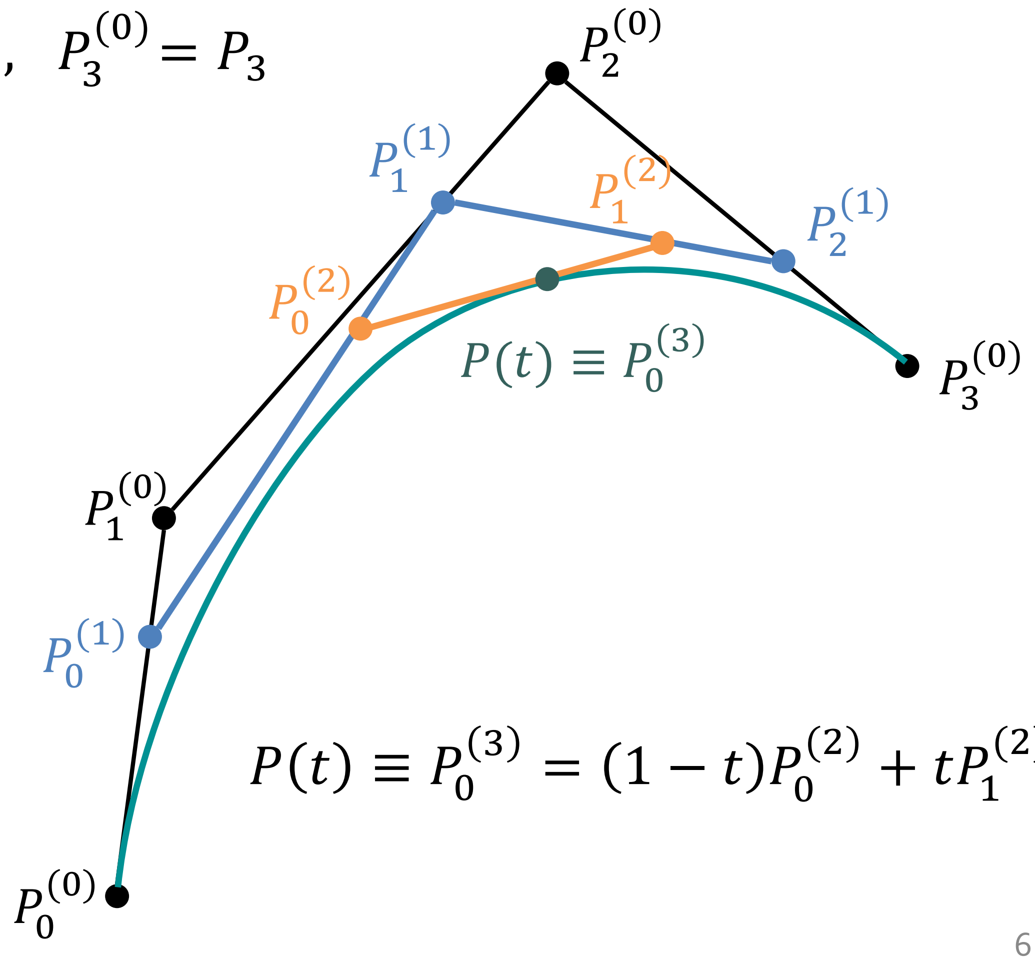
$$P_0^{(1)} = (1-t)P_0 + tP_1$$

$$P_1^{(1)} = (1-t)P_1 + tP_2$$

$$P_2^{(1)} = (1-t)P_2 + tP_3$$

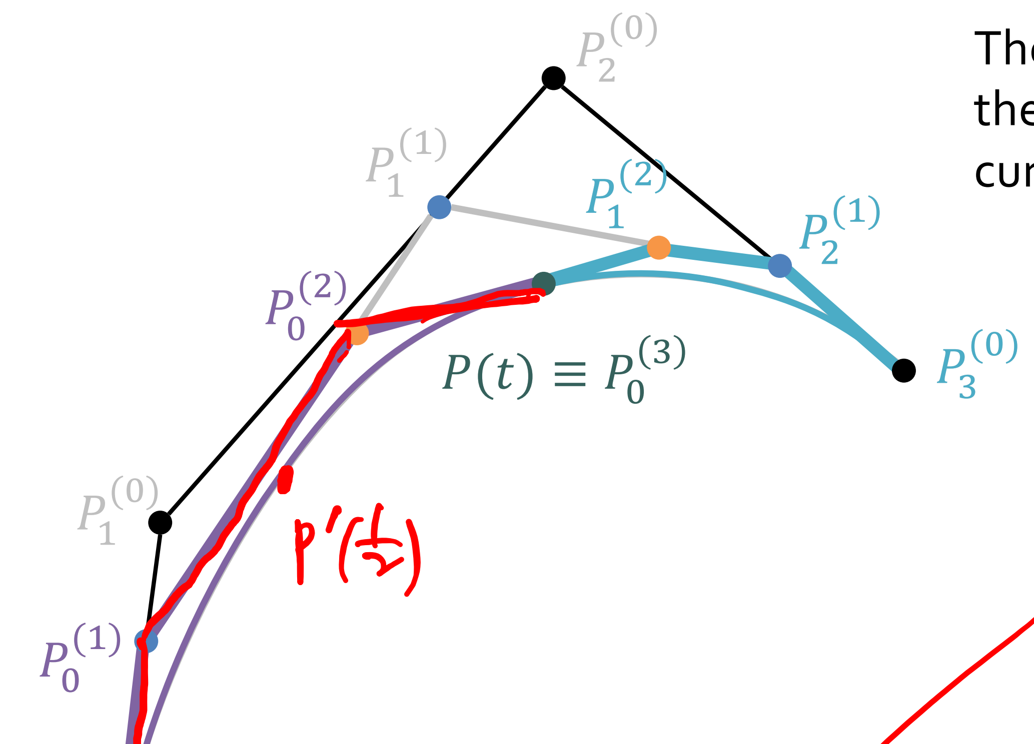
$$P_0^{(2)} = (1-t)P_0^{(1)} + tP_1^{(1)}$$

$$P_1^{(2)} = (1-t)P_1^{(1)} + tP_2^{(1)}$$



$$P(t) \equiv P_0^{(3)} = (1-t)P_0^{(2)} + tP_1^{(2)}$$

Splitting the Control Polygon



The points $P_0^{(3)}, P_1^{(2)}, P_2^{(1)}, P_2^{(0)}$ form the Bezier control polygon of the curve segment $P(t)P(1)$.

The points $P_0^{(0)}, P_0^{(1)}, P_0^{(2)}, P_0^{(3)}$ form the Bezier control polygon of the curve segment $P(0)P(t)$.

Middle-point Subdivision

When setting $t = \frac{1}{2}$, the arithmetic operations of the above algorithm are simplified to

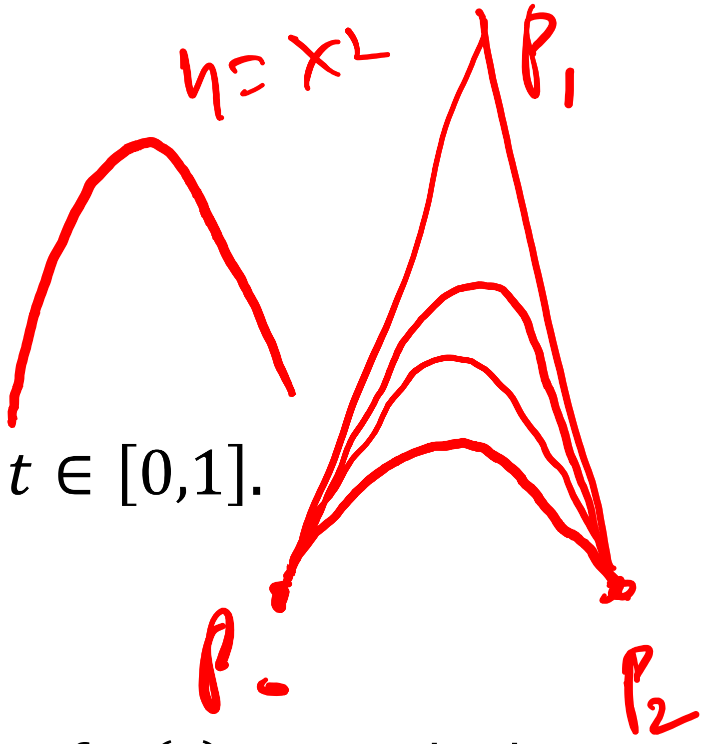
$$P_i^{(k+1)} = \frac{P_i^{(k)} + P_{i+1}^{(k)}}{2}.$$

So only addition and right shift (division by 2) are required.

The middle-point subdivision scheme works by computing the Bézier control polygons of the two sub-curves of $P(t), t \in [0,1]$, over subintervals $[0, \frac{1}{2}]$ and $[\frac{1}{2}, 0]$. Then each sub-curve is recursively subdivided.

Note that the depth of subdivision can be made adaptive to the local curvature or the error of approximation.

Rational Bézier Curves



A rational Bézier curve is represented by

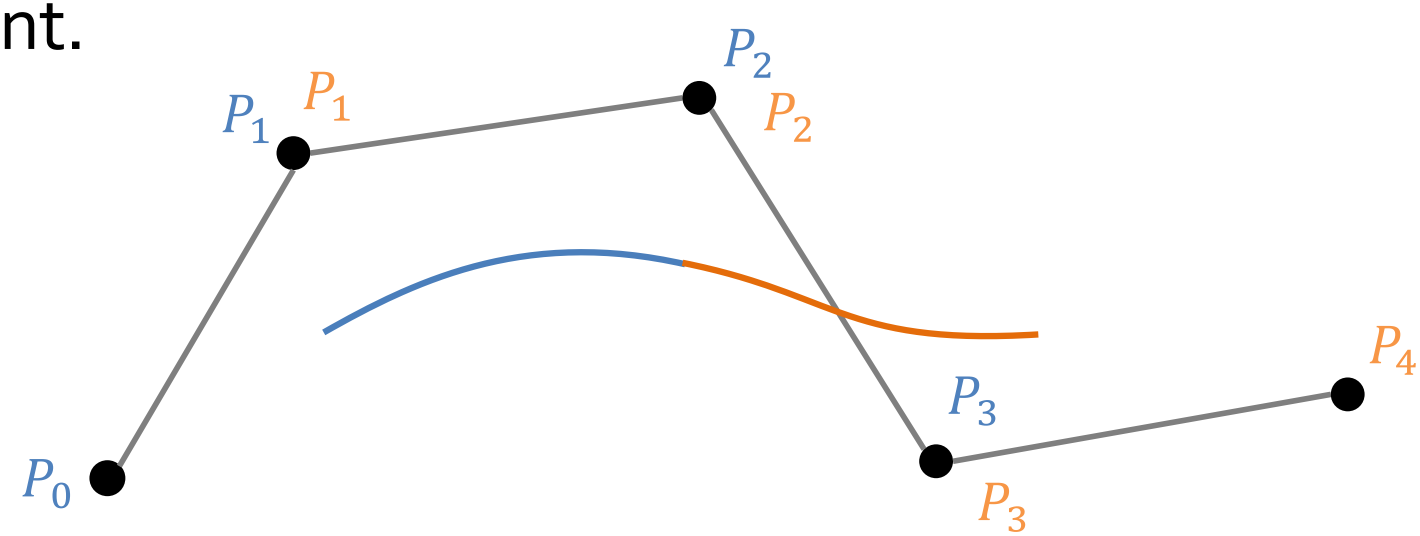
$$P(t) = \frac{\sum_{i=0}^{n} w_i B_{i,n}(t) P_i}{\sum_{i=0}^{n} w_i B_{i,n}(t)}, \qquad t \in [0,1].$$

- w_i can be thought of weights
- Increasing the weight w_i pulls a portion of $P(t)$ towards the control point P_i, and decreasing w_i pushes $P(t)$ away from P_i
- An advantage of the rational curve is that it encompasses all conic sections (with n = 2).

B-Splines

Basis splines: use the control points $p_{i-2}, p_{i-1}, p_i, p_{i+1}$ to define curve only between p_{i-1} and p_i (for cubic curves)

Allows us to apply more continuity conditions to each segment.



NURBS: Nonuniform Rational B-Spline curves and surfaces