

Computer Graphics 2019

10. Spline and Surfaces

Hongxin Zhang

State Key Lab of CAD&CG, Zhejiang University

2019-11-27



Outline

- Introduction
- Bézier curve and surface
- NURBS curve and surface
- subdivision curve and surface

classification of curves

$$y = x^2 + 5x + 3$$



$$y = f(x)$$

(explicit curve)

$$(x - x_c)^2 + (y - y_c)^2 - r^2 = 0$$



$$g(x, y) = 0$$

(implicit curve)

$$x = x_c + r \cdot \cos\theta$$

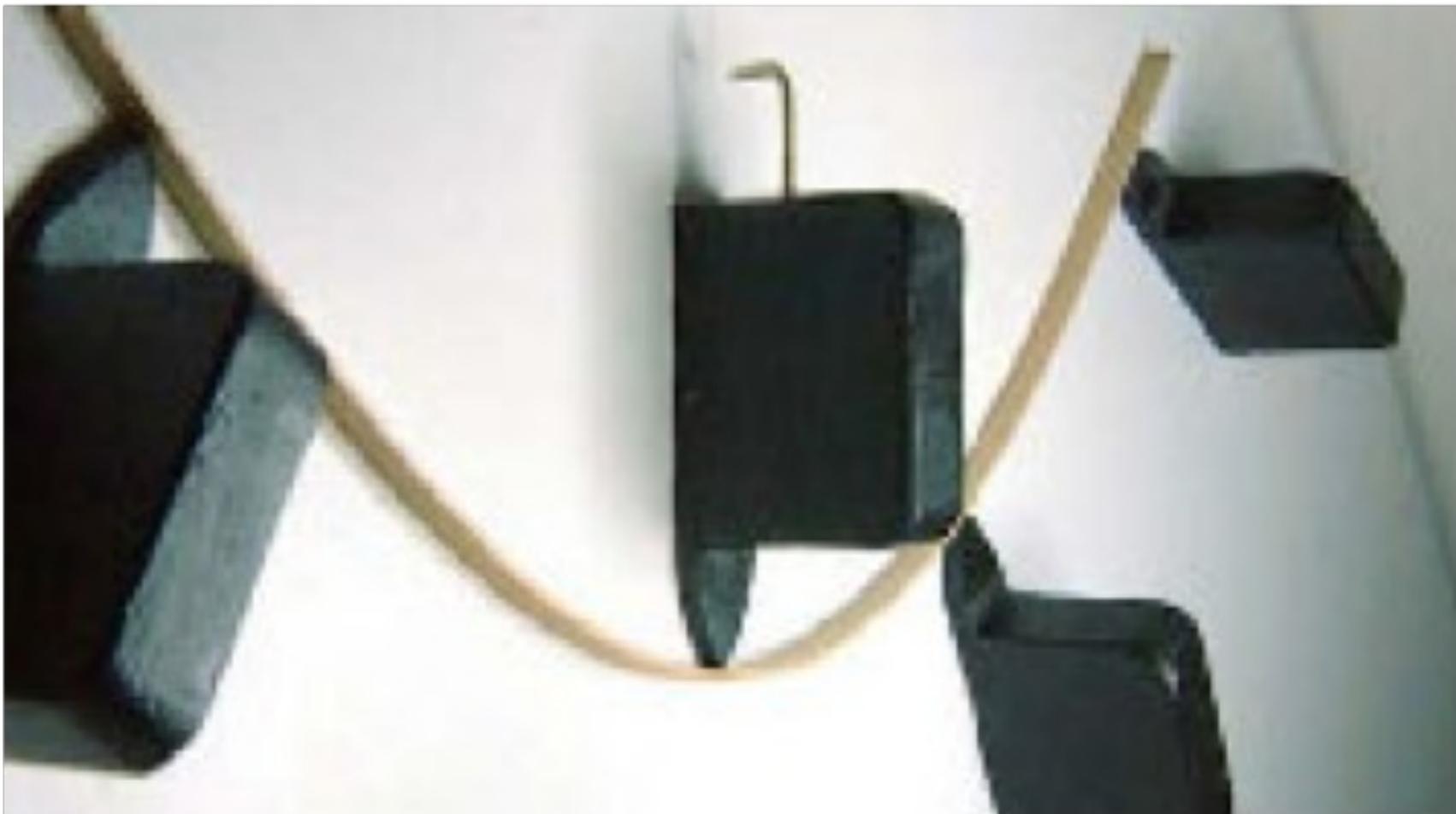
$$y = y_c + r \cdot \sin\theta$$



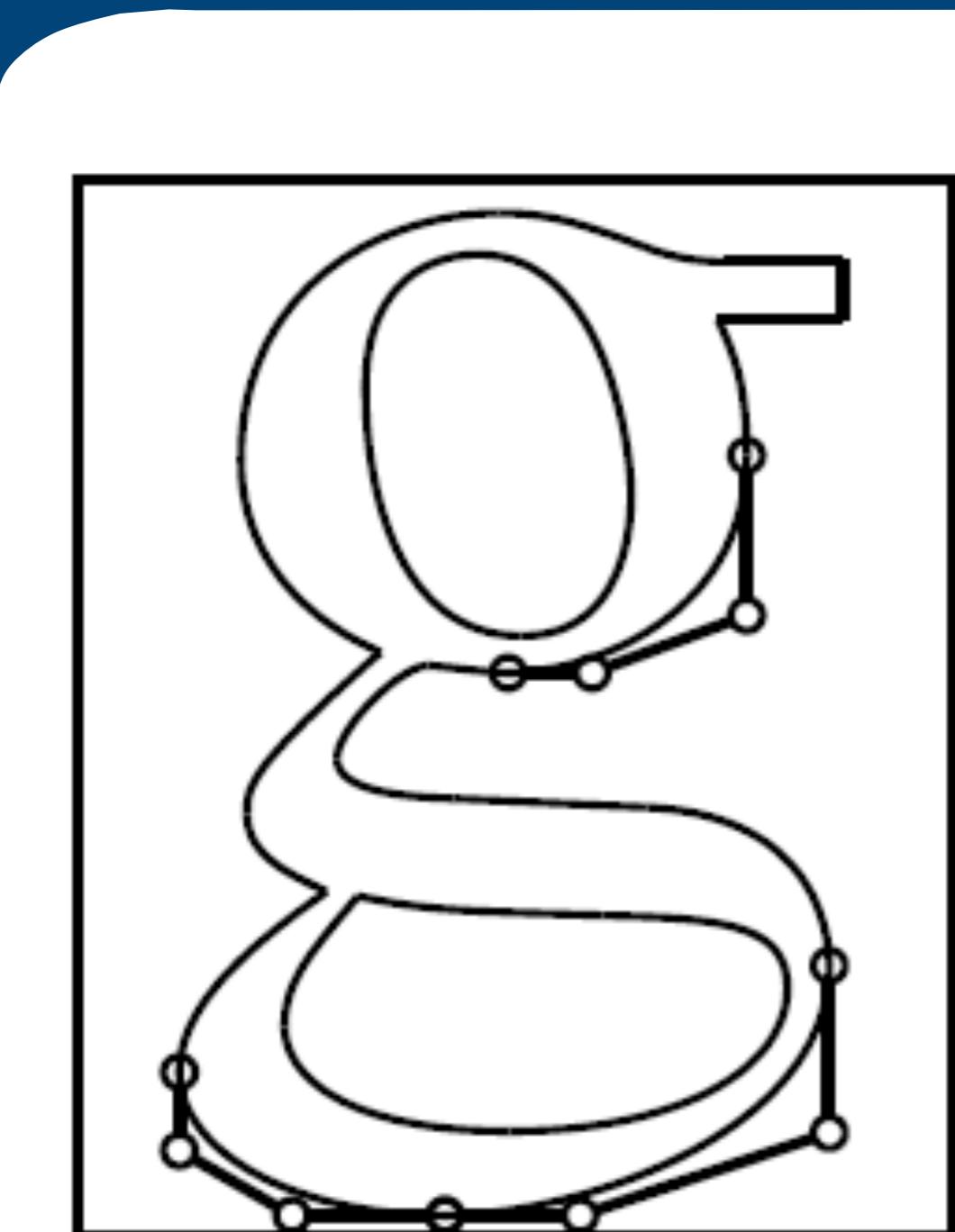
$$\begin{cases} x = x(t) \\ y = y(t) \end{cases}$$

**(parametric
curve)**

Splines



Bézier curve



Pierre Étienne Bézier
an engineer at Renault



Bézier curve

Bézier curve

$$\mathbf{C}(t) = \sum_{i=0}^n P_i B_{i,n}(t), \quad t \in [0, 1]$$

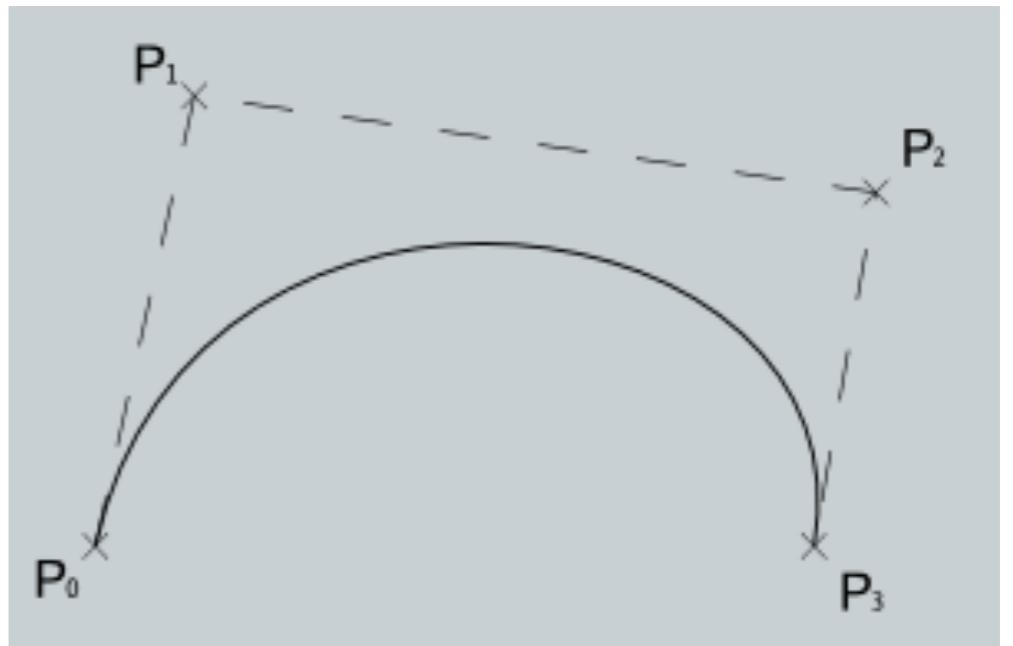
where, P_i ($i=0,1,\dots,n$) are control points.

$$B_{i,n}(t) = C_n^i t^i (1-t)^{n-i}, \quad t \in [0, 1]$$

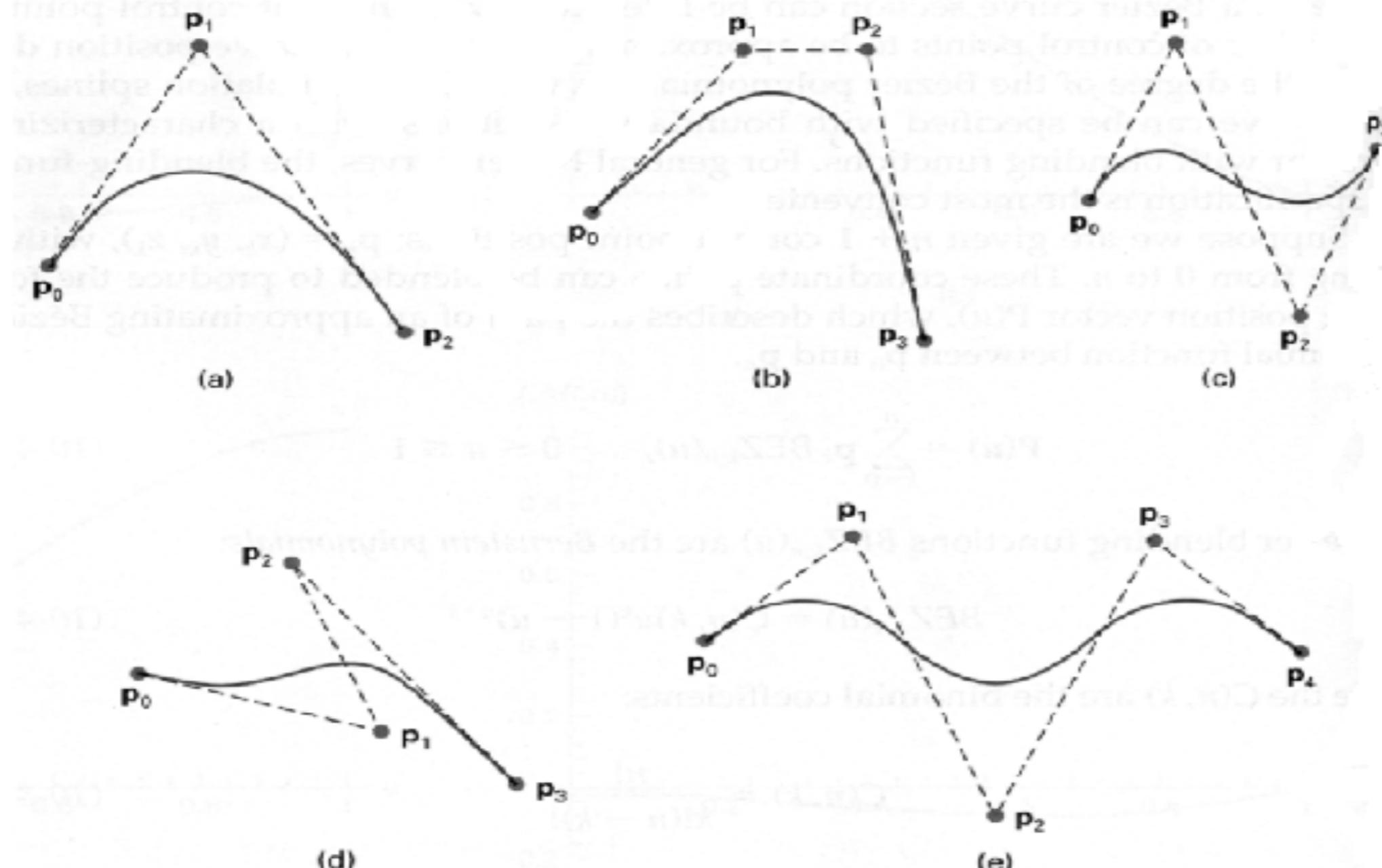
Bernstein basis

$$\begin{cases} X(t) = \sum_{i=0}^n x_i B_{i,t}(t) \\ Y(t) = \sum_{i=0}^n y_i B_{i,t}(t) \end{cases}$$

$$C(t) = \begin{pmatrix} X(t) \\ Y(t) \end{pmatrix}, \quad P_i = \begin{pmatrix} x_i \\ y_i \end{pmatrix}$$



Bézier curve



General spline curves

parametric curve



$$\mathbf{P}(t) = \sum_i \mathbf{P}_i B_i(t)$$

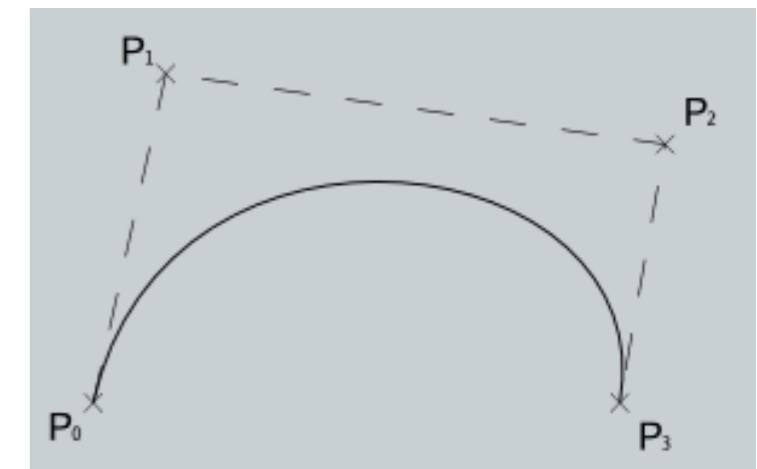
$t \in [t_0, t_1]$

basis functions



i

control pints



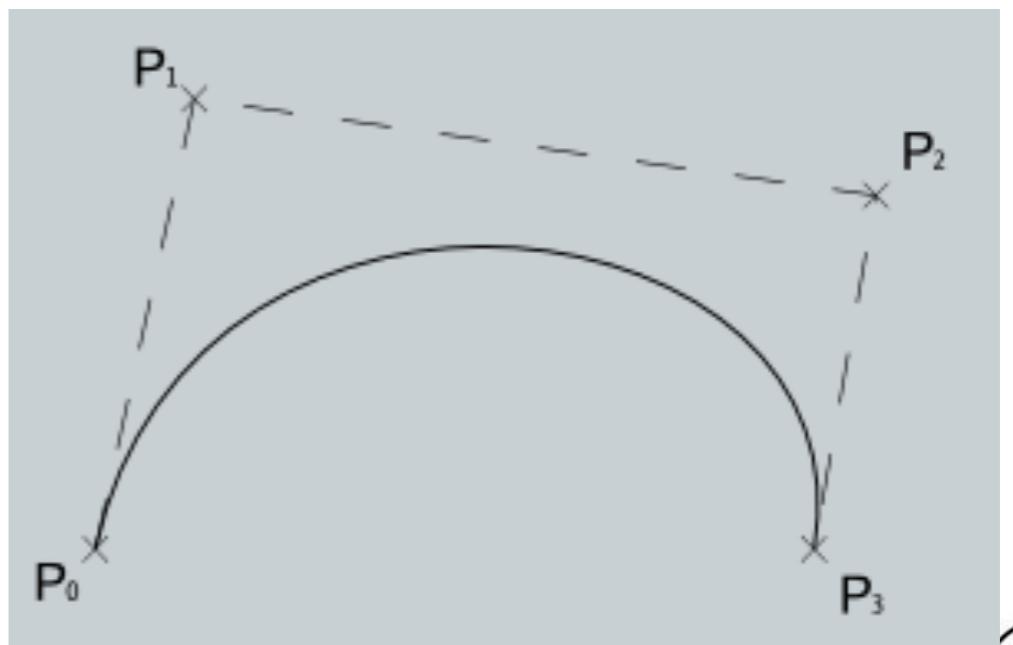
Bézier curve

$$\begin{cases} X(t) = \sum_{i=0}^n x_i B_{i,t}(t) \\ Y(t) = \sum_{i=0}^n y_i B_{i,t}(t) \end{cases}$$

$$\begin{cases} X(t) = \sum_{i=0}^n a_i t^i \\ Y(t) = \sum_{i=0}^n b_i t^i \end{cases}$$

$$B_{i,n}(t) = C_n^i t^i (1-t)^{n-i}, t \in [0,1]$$

$$C(t) = \begin{pmatrix} X(t) \\ Y(t) \end{pmatrix}, P_i = \begin{pmatrix} x_i \\ y_i \end{pmatrix}$$



Bézier curve

Properties of Bernstein **basis** $B_{i,n}(t) = C_n^i t^i (1-t)^{n-i}$, $t \in [0,1]$

1. $B_{i,n}(t) \geq 0$, $i = 0, 1, \dots, n$, $t \in [0,1]$.

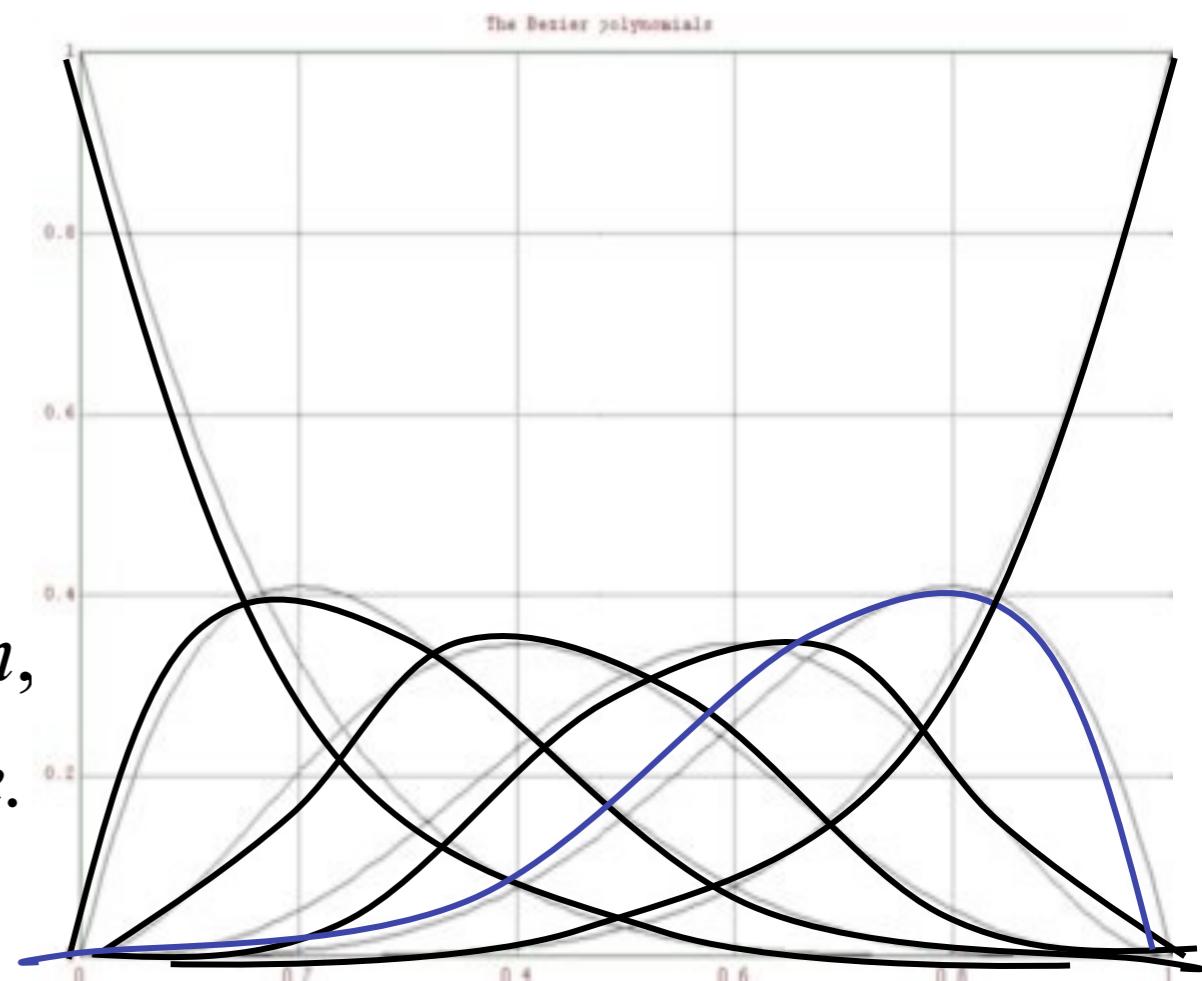
2. $\sum_{i=0}^n B_{i,n}(t) = 1$, $t \in [0,1]$.

3. $B_{i,n}(t) = B_{n-i,n}(1-t)$,

$i = 0, 1, \dots, n$, $t \in [0,1]$.

4.

$$B_{i,n}(0) = \begin{cases} 1, & i = 0, \\ 0, & \text{else;} \end{cases} \quad B_{i,n}(1) = \begin{cases} 1, & i = n, \\ 0, & \text{else.} \end{cases}$$



Bézier curve

Properties of Bernstein **basis**

5. $B_{i,n}(t) = (1-t)B_{i,n-1}(t) + tB_{i-1,n-1}(t), \quad i = 0, 1, \dots, n.$

6. $B'_{i,n}(t) = n[B_{i-1,n-1}(t) - B_{i,n-1}(t)], \quad i = 0, 1, \dots, n.$

7.

$$(1-t)B_{i,n}(t) = \left(1 - \frac{i}{n+1}\right)B_{i,n+1}(t);$$

$$tB_{i,n}(t) = \frac{i+1}{n+1}B_{i+1,n+1}(t);$$

$$B_{i,n}(t) = \left(1 - \frac{i}{n+1}\right)B_{i,n+1}(t) + \frac{i+1}{n+1}B_{i+1,n+1}(t).$$

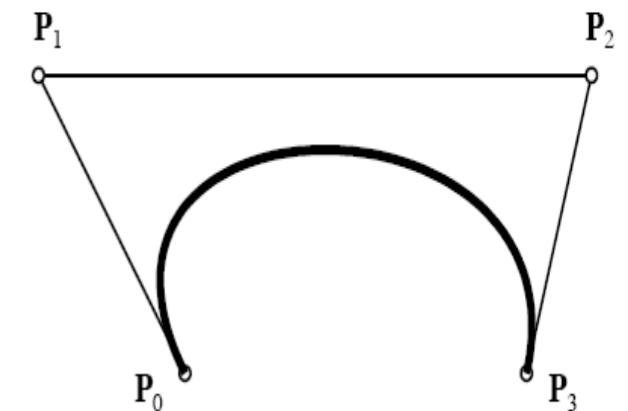
Bézier curve

properties of Bézier curves

$$C(t) = \sum_{i=0}^n P_i B_{i,n}(t), \quad t \in [0,1]$$

1. **Endpoint Interpolation:** interpolating two end points

$$C(0) = P_0, \quad C(1) = P_n.$$



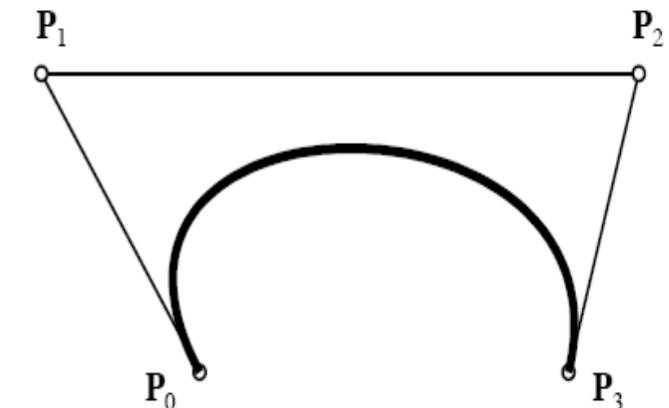
2. **tangent direction** of P_0 : P_0P_1 , tangent direction of P_n : $P_{n-1}P_n$.

$$C'(t) = n \sum_{i=0}^{n-1} (P_{i+1} - P_i) B_{i,n-1}(t), \quad t \in [0,1]; \quad C'(0) = n(P_1 - P_0), \quad C'(1) = n(P_n - P_{n-1}).$$

3. **Symmetry:** Let two Bezier curves be generated by ordered Bezier (control) points labelled by $\{p0, p1, \dots, pn\}$ and $\{pn, pn-1, \dots, p0\}$ respectively, then the curves corresponding to the two different orderings of control points look the same; they differ only in the direction in which they are traversed.

properties of Bézier curves

$$C(t) = \sum_{i=0}^n P_i B_{i,n}(t), \quad t \in [0,1]$$



4. Affine Invariance –

the following two procedures yield the same result:

- (1) first, from starting control points $\{p_0, p_1, \dots, p_n\}$ compute the curve and then apply an affine map to it;
- (2) first apply an affine map to the control points $\{p_0, p_1, \dots, p_n\}$ to obtain new control points $\{F(p_0), \dots, F(p_n)\}$ and then find the curve with these new control points.

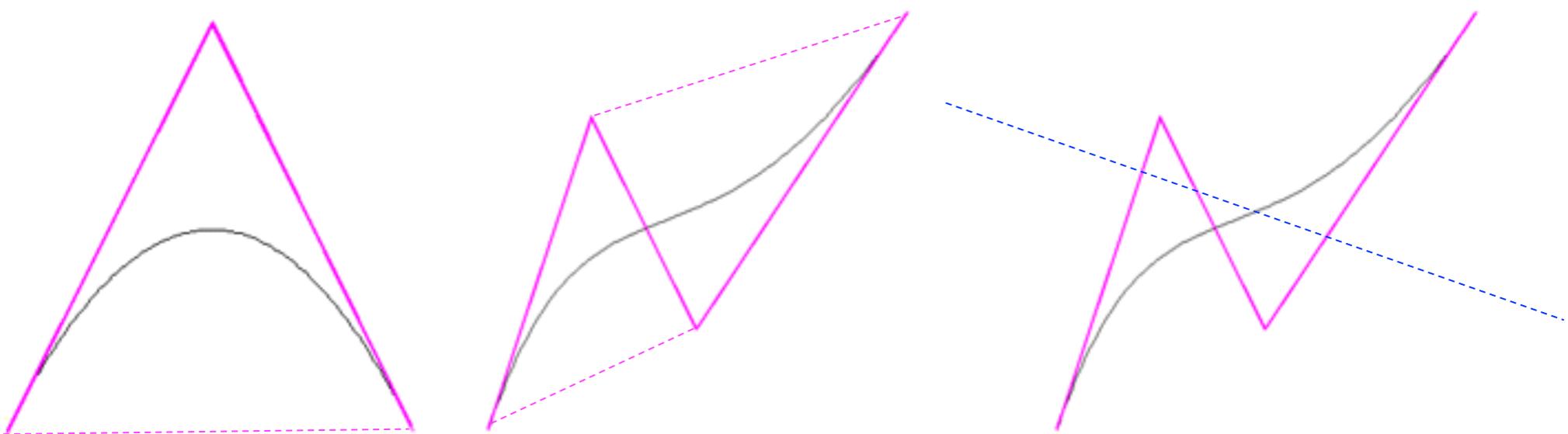


Bézier curve

properties of Bézier curves

5. **Convex hull property :** Bézier curve $\mathbf{C}(t)$ lies in the convex hull of the control points $\mathbf{P}_0, \mathbf{P}_1, \dots, \mathbf{P}_n$;

6. **Variation diminishing property.** Informally this means that the Bezier curve will not "wiggle" any more than the control polygon does..

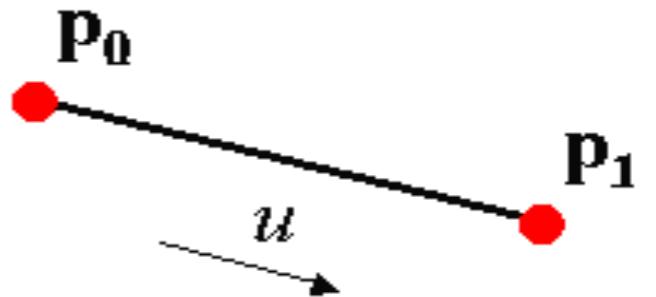


Bézier curve

Bézier curves

1. linear: $C(t) = (1-t)\mathbf{P}_0 + t\mathbf{P}_1, t \in [0,1]$,

$$C(t) = [t, 1] \begin{bmatrix} -1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{P}_0 \\ \mathbf{P}_1 \end{bmatrix}$$



2. quadratic

$$C(t) = (1-t)^2 \mathbf{P}_0 + 2t(1-t)\mathbf{P}_1 + t^2 \mathbf{P}_2$$



$$C(t) = [t^2 \quad t \quad 1] \begin{bmatrix} 1 & -2 & 1 \\ -2 & 2 & 0 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{P}_0 \\ \mathbf{P}_1 \\ \mathbf{P}_2 \end{bmatrix}$$

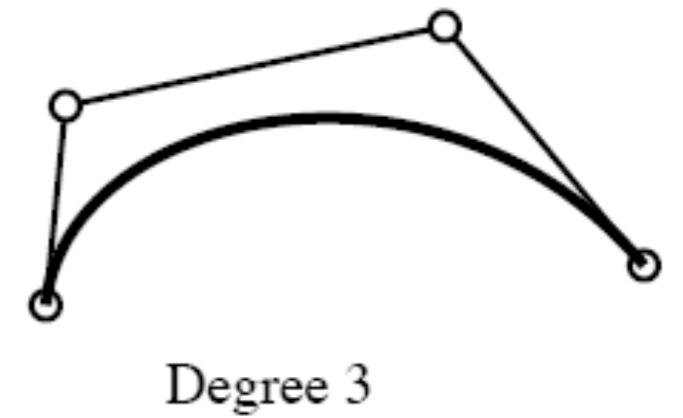


Bézier curve

3. cubic:

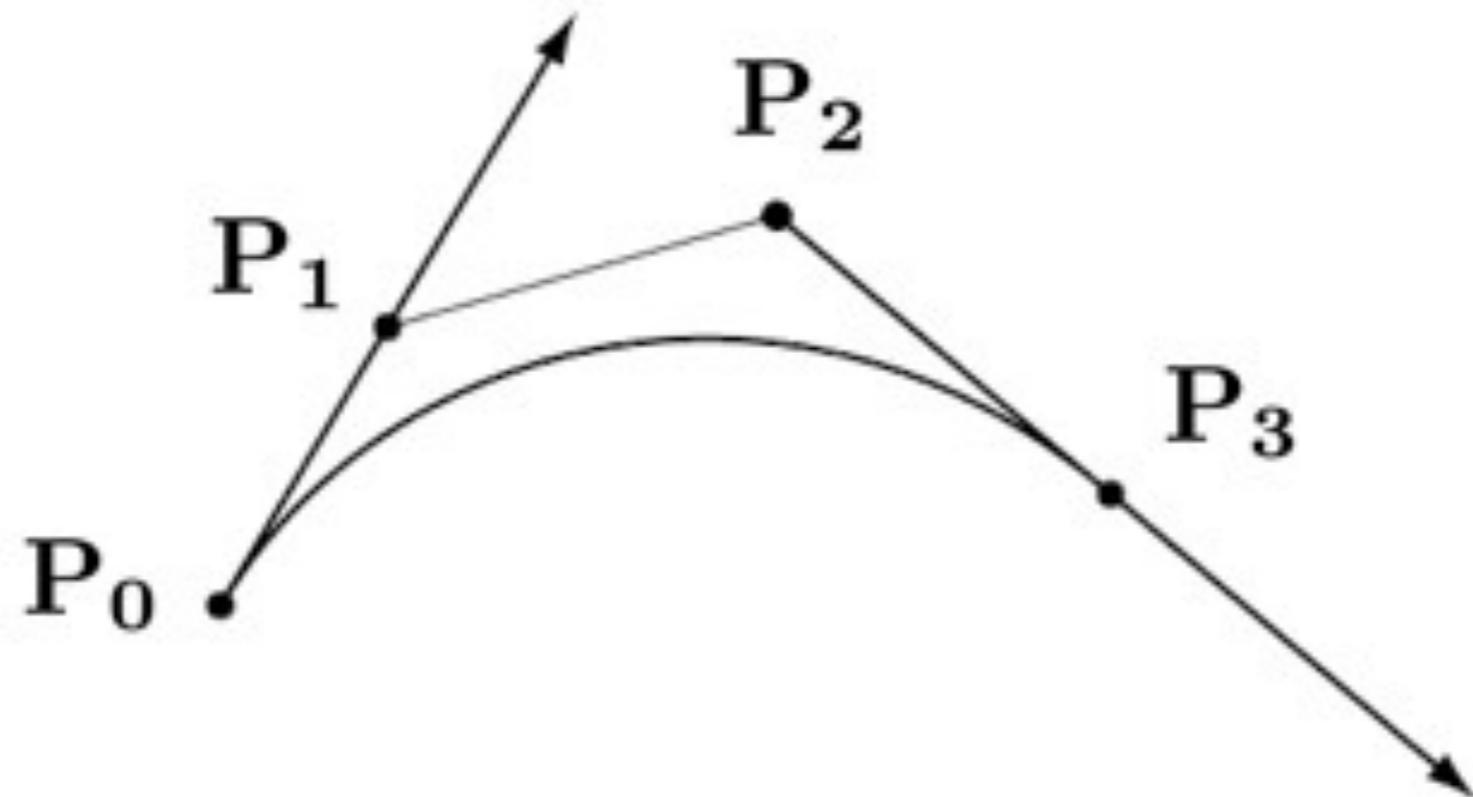
$$C(t) = (1-t)^3 \mathbf{P}_0 + 3t(1-t)^2 \mathbf{P}_1 + 3t^2(1-t)\mathbf{P}_2 + t^3 \mathbf{P}_3$$

$$\mathbf{C}(t) = [t^3 \quad t^2 \quad t \quad 1] \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{P}_0 \\ \mathbf{P}_1 \\ \mathbf{P}_2 \\ \mathbf{P}_3 \end{bmatrix}$$



Bezier Curve

$$T_0 = 3(P_1 - P_0)$$



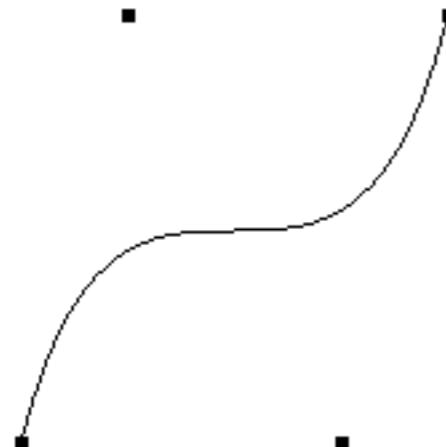
$$T_1 = 3(P_3 - P_2)$$

Bezier Curve in OpenGL

- `glMapI*(GL_MAPI_VERTEX_3,
 uMin, uMax, stride, nPts, *ctrlPts);`
- `glEnable/glDisable(GL_MAPI_VERTEX_3);`
- `glBegin(GL_LINE_STRIP);
for (...) {
 glEvalCoordI*(uValue);
}
glEnd();`

Bezier Curve in OpenGL

```
GLfloat ctrlpoints[4][3] = {  
    { -4.0, -4.0, 0.0}, { -2.0, 4.0, 0.0},  
    {2.0, -4.0, 0.0}, {4.0, 4.0, 0.0}};  
  
void init(void)  
{  
    glClearColor(0.0, 0.0, 0.0, 0.0);  
    glShadeModel(GL_FLAT);  
    glMapIf(GL_MAPI_VERTEX_3,  
    0.0, 1.0, 3, 4, &ctrlpoints[0][0]);  
    glEnable(GL_MAPI_VERTEX_3);  
}
```



```
void display(void)  
{  
    int i;  
  
    glClear(GL_COLOR_BUFFER_BIT);  
    glColor3f(1.0, 1.0, 1.0);  
    glBegin(GL_LINE_STRIP);  
        for (i = 0; i <= 30; i++)  
            glEvalCoordIf((GLfloat) i/30.0);  
    glEnd();  
    /* The following code displays the control points as dots. */  
    glPointSize(5.0);  
    glColor3f(1.0, 1.0, 0.0);  
    glBegin(GL_POINTS);  
        for (i = 0; i < 4; i++)  
            glVertex3fv(&ctrlpoints[i][0]);  
    glEnd();  
    glFlush();  
}
```

Bézier curve

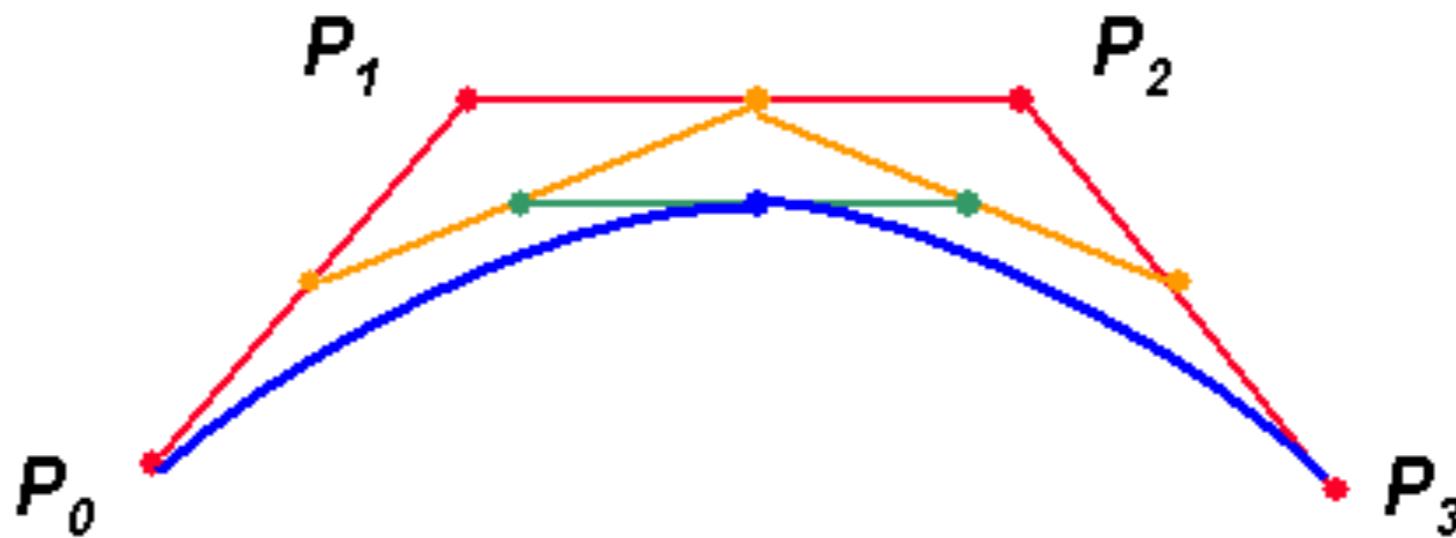


de Casteljau algorithm

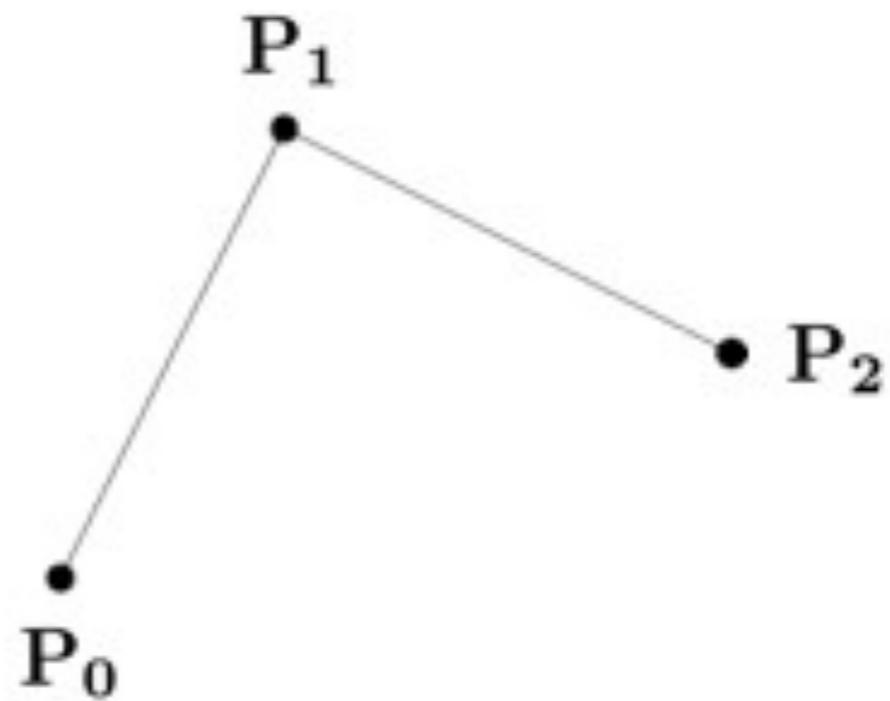
given the control points $\mathbf{P}_0, \mathbf{P}_1, \dots, \mathbf{P}_n$, and t of Bézier curve, let:

$$\mathbf{P}_i^r(t) = (1-t)\mathbf{P}_i^{r-1}(t) + t\mathbf{P}_{i+1}^{r-1}(t), \quad \begin{cases} r = 1, \dots, n; i = 0, \dots, n-r \\ P_i^0(u) = P_i \end{cases}$$

then $\mathbf{P}_0^n(t) = C(t)$.

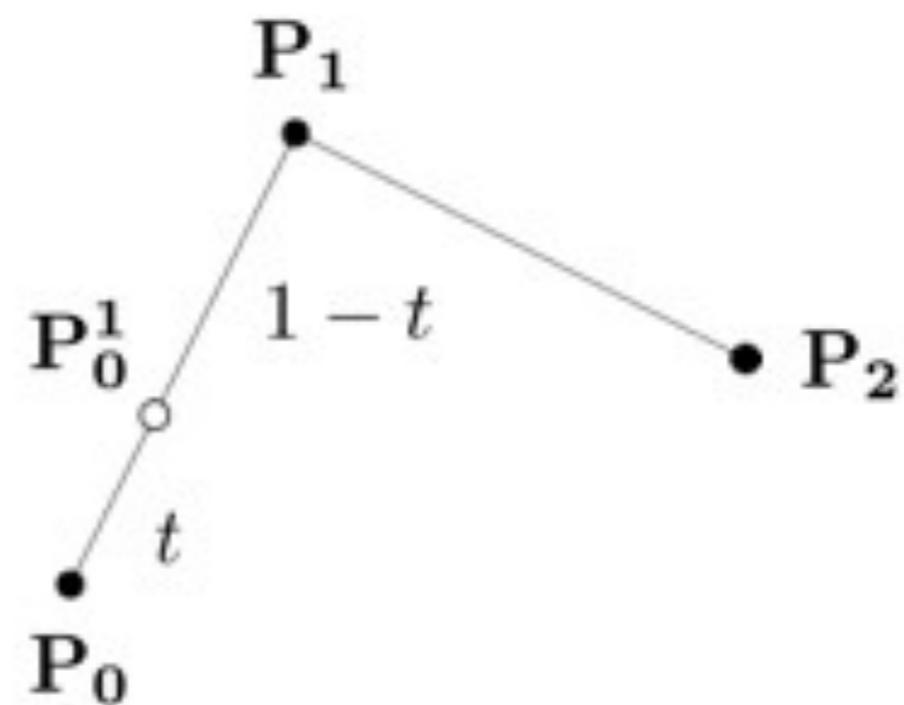


Consider Three Points



Insert Point Using Linear Interpolation

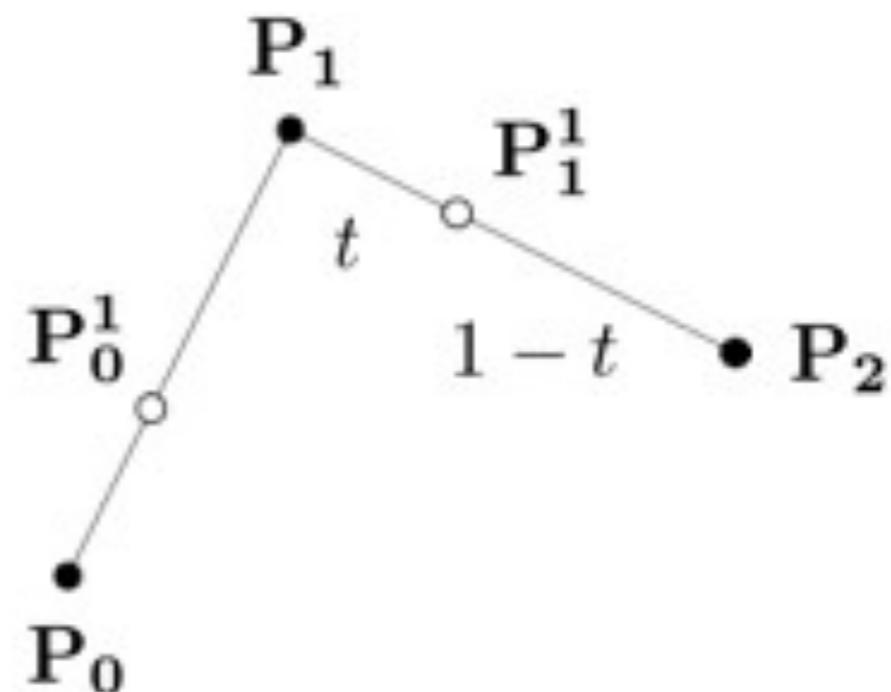
$$\mathbf{P}_0^1 = (1 - t)\mathbf{P}_0 + t\mathbf{P}_1$$



Insert Points on Both Edges

$$\mathbf{P}_0^1 = (1 - t)\mathbf{P}_0 + t\mathbf{P}_1$$

$$\mathbf{P}_1^1 = (1 - t)\mathbf{P}_1 + t\mathbf{P}_2$$

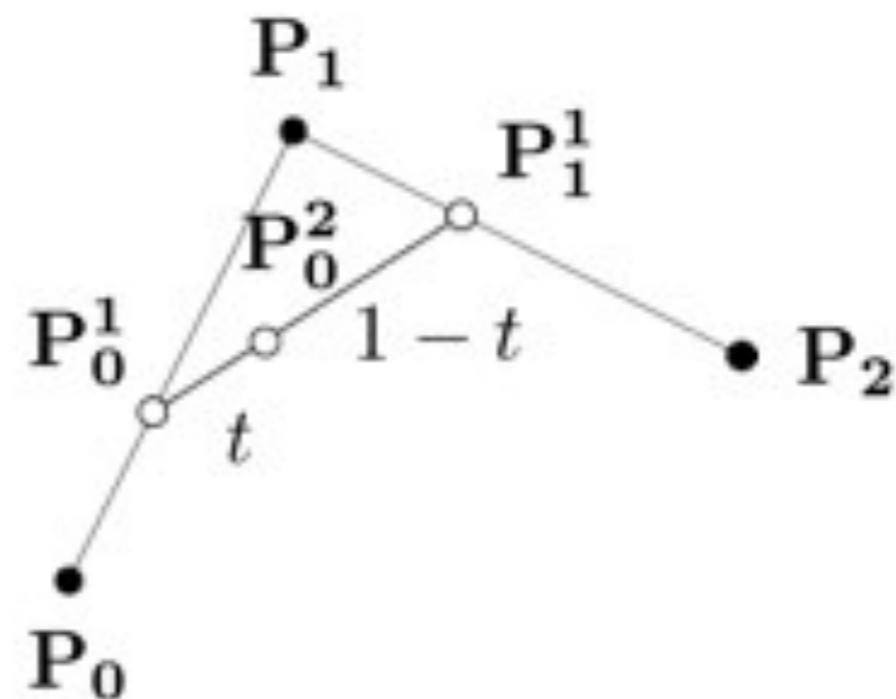


Repeat Recursively

$$\mathbf{P}_0^1 = (1 - t)\mathbf{P}_0 + t\mathbf{P}_1$$

$$\mathbf{P}_1^1 = (1 - t)\mathbf{P}_1 + t\mathbf{P}_2$$

$$\mathbf{P}_0^2 = (1 - t)\mathbf{P}_0^1 + t\mathbf{P}_1^1$$

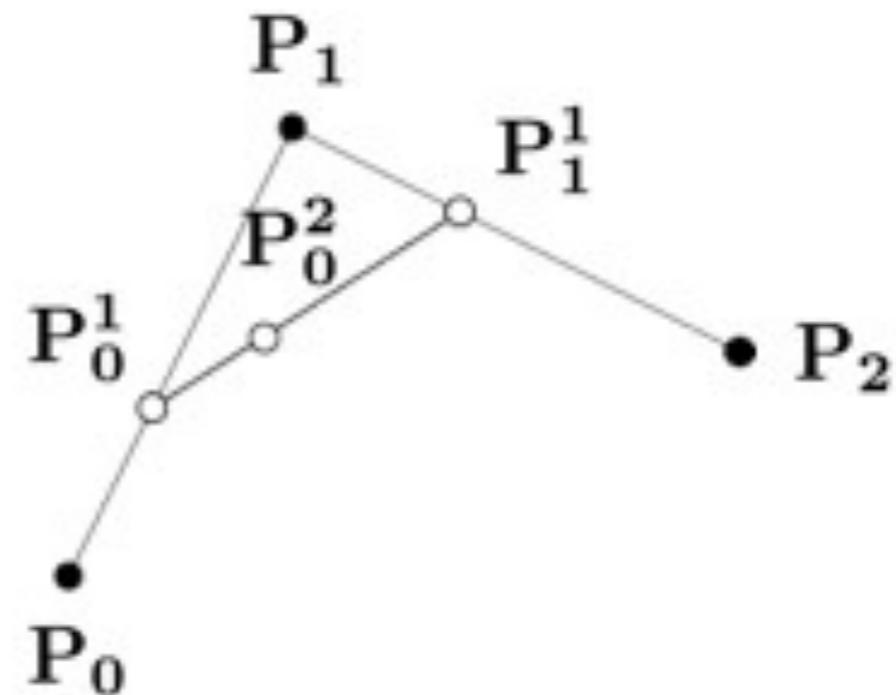


Algorithm Defines Curve

$$\mathbf{P}_0^1 = (1 - t)\mathbf{P}_0 + t\mathbf{P}_1$$

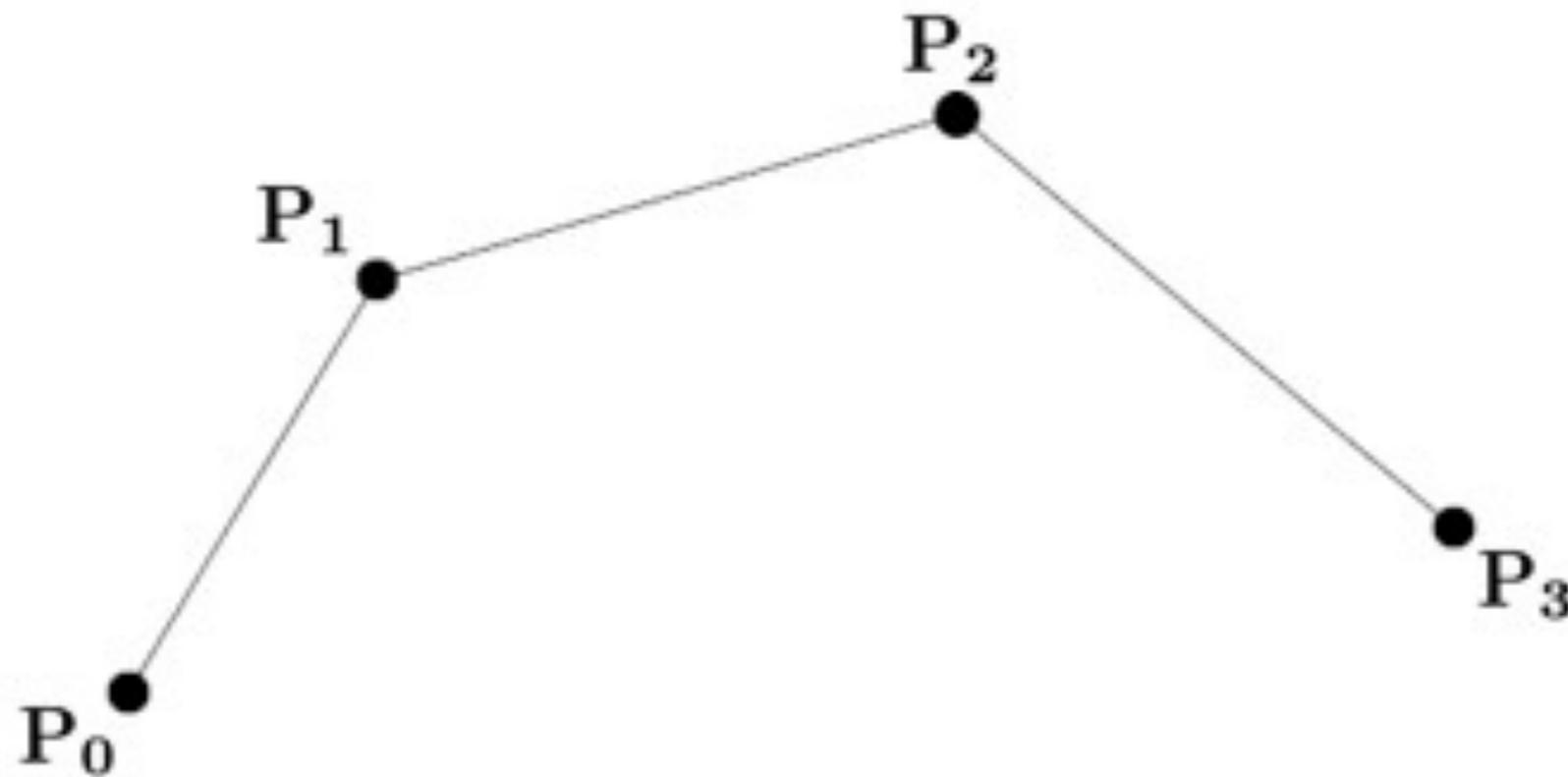
$$\mathbf{P}_1^1 = (1 - t)\mathbf{P}_1 + t\mathbf{P}_2$$

$$\mathbf{P}_0^2 = (1 - t)\mathbf{P}_0^1 + t\mathbf{P}_1^1$$

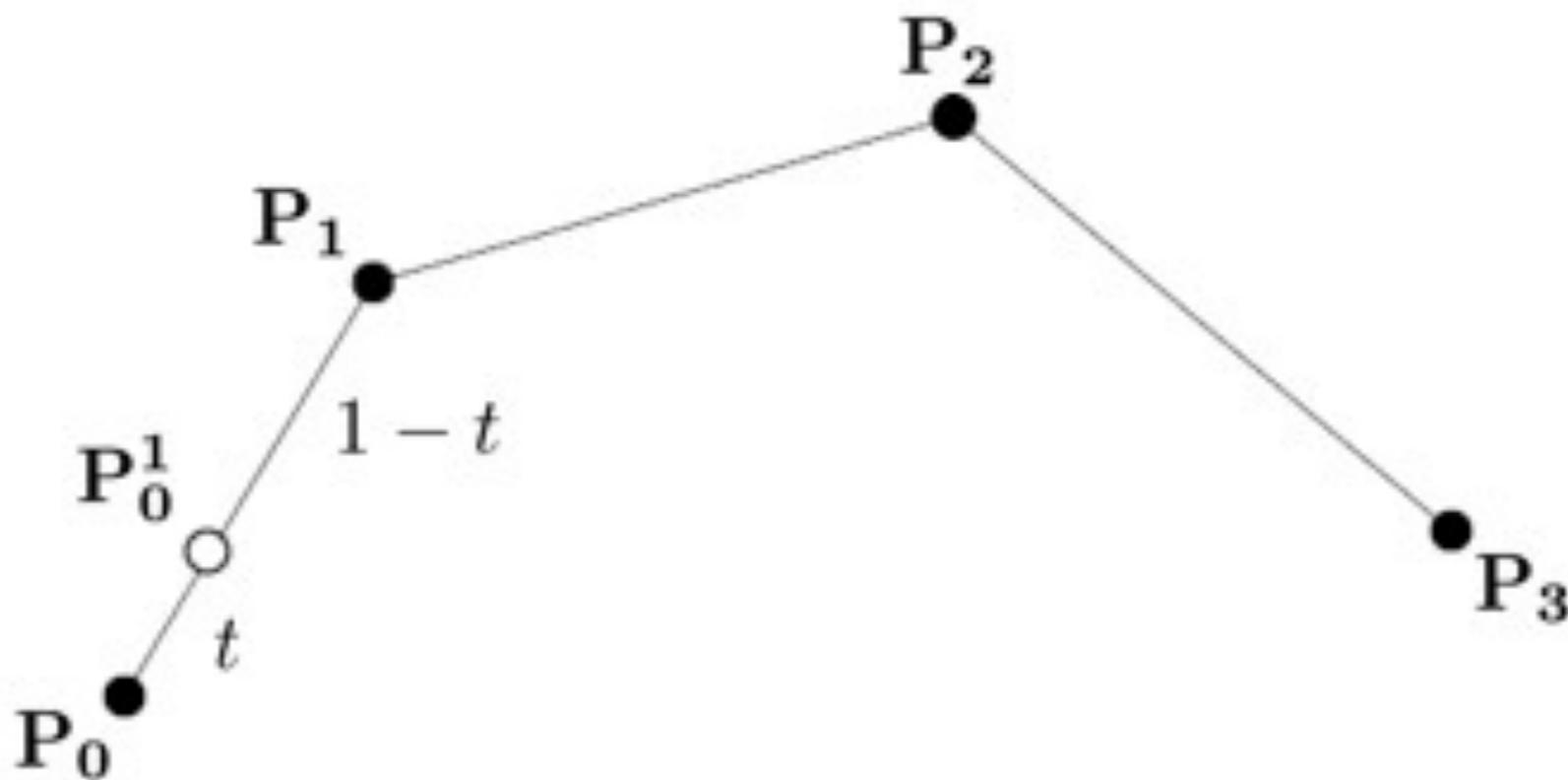


Resulting point $\mathbf{P}(t) = \mathbf{P}_0^2$

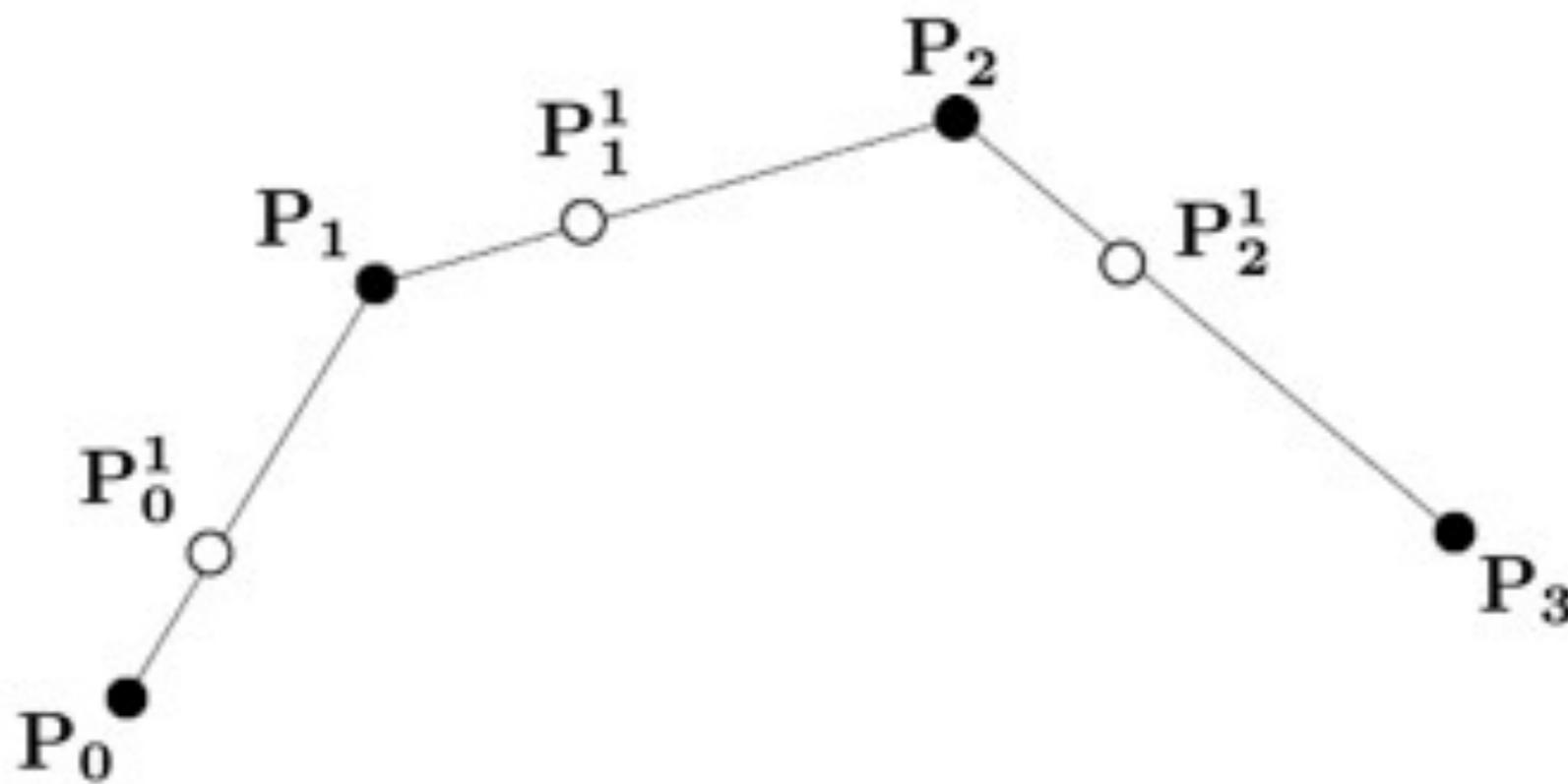
Consider Four Points



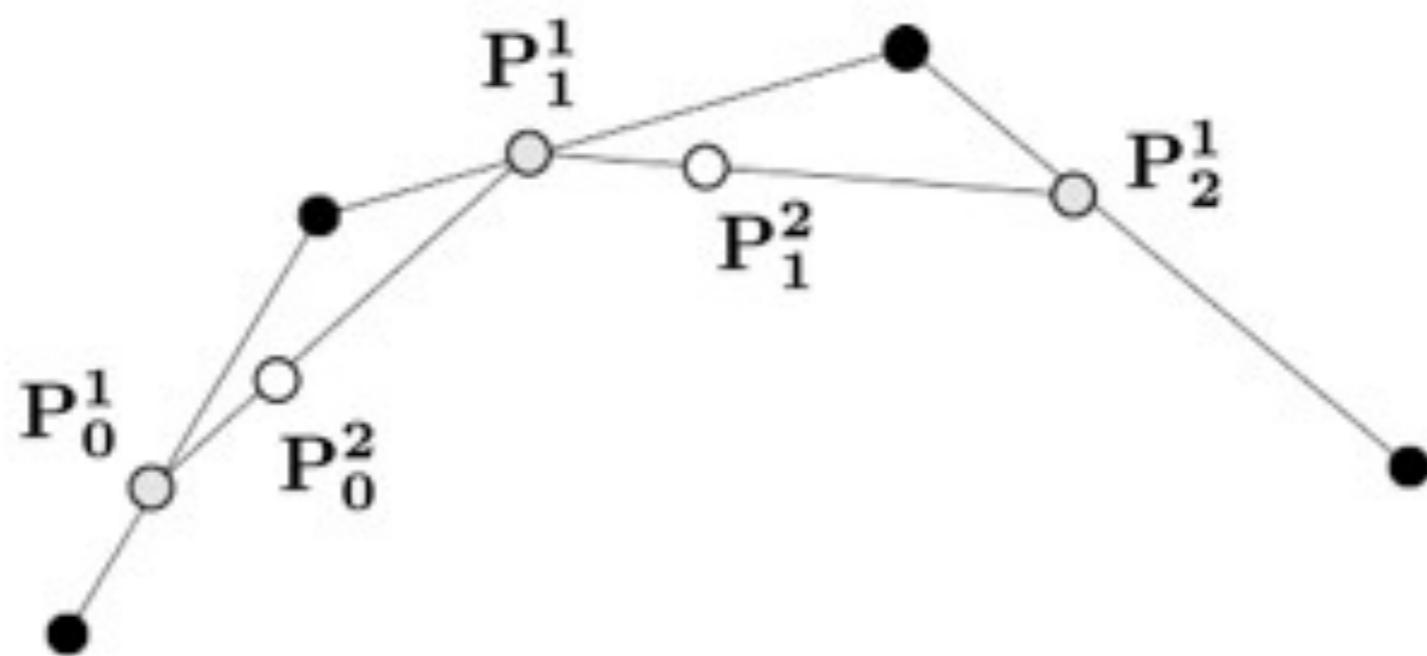
Linear Interpolation



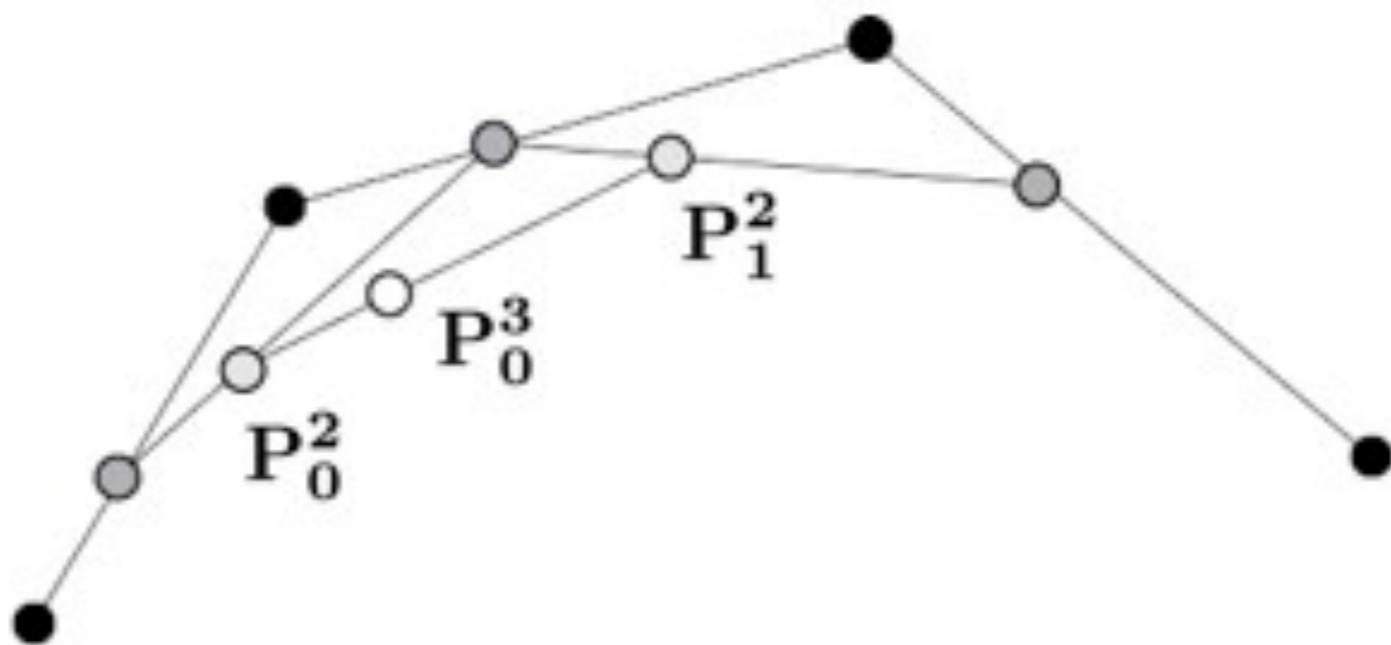
On All Edge Segments



Repeat Recursively

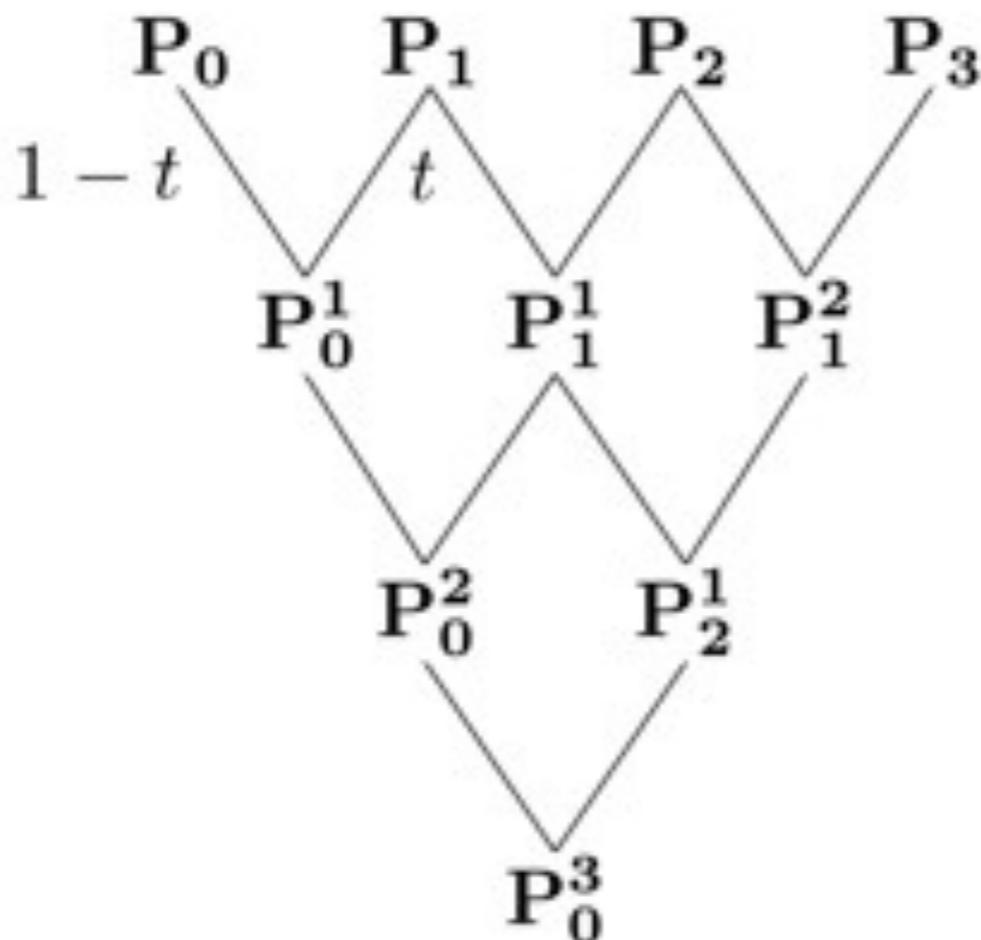


Algorithm Defines Curve



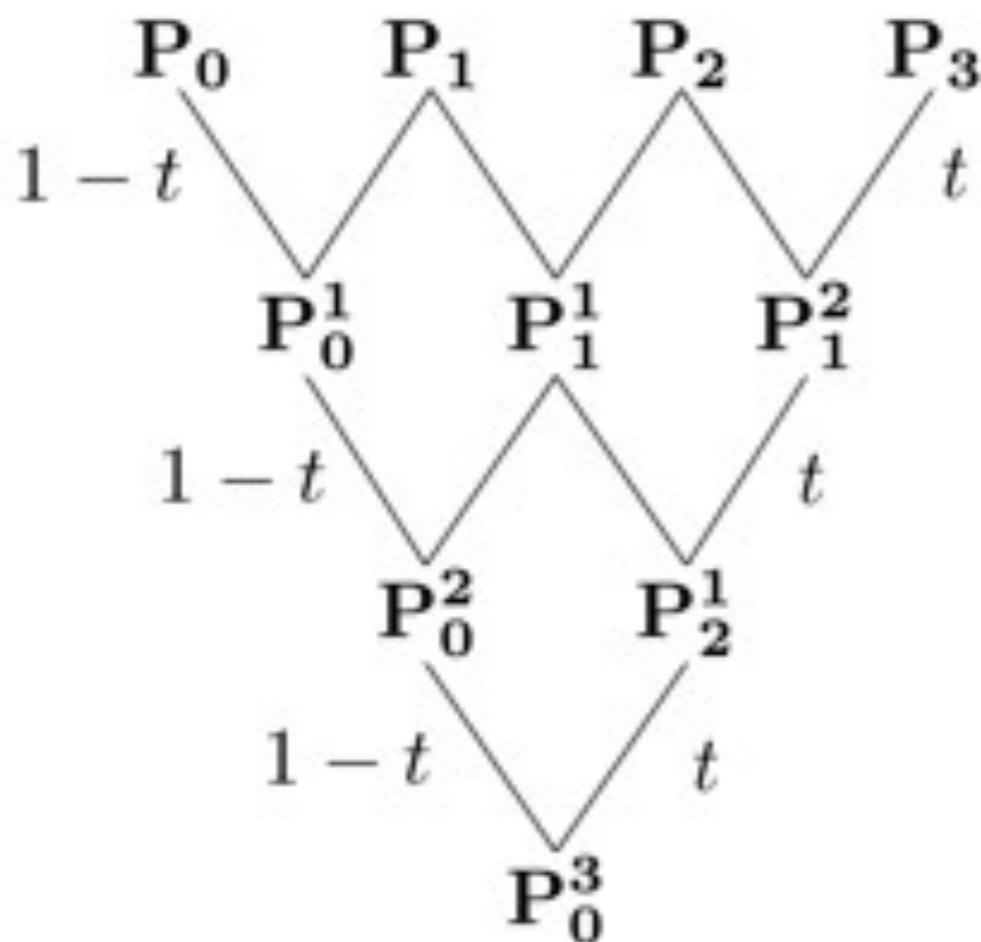
$$\mathbf{P}(t) = \mathbf{P}_0^3$$

Pyramid Algorithm



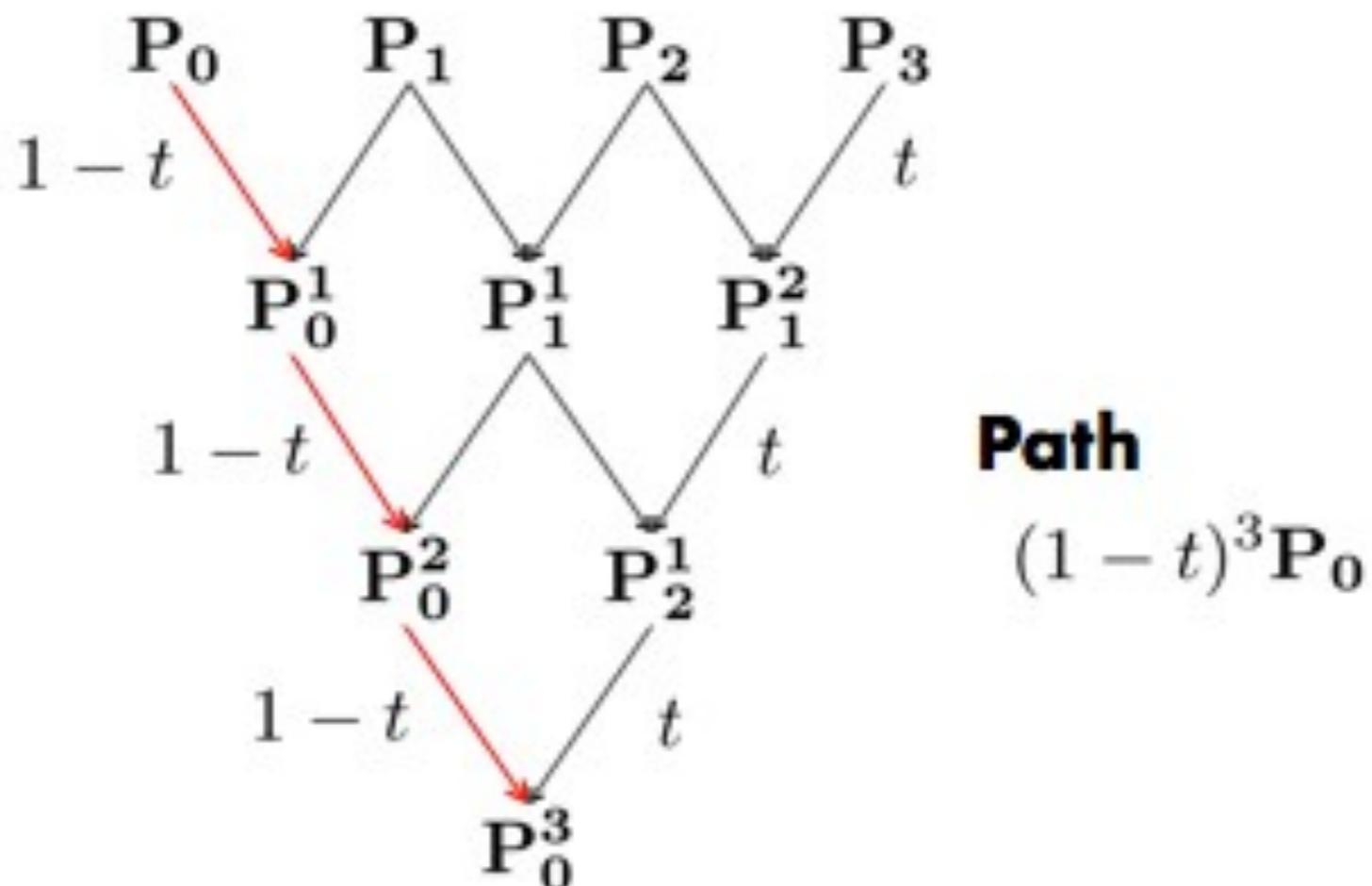
$$\mathbf{P}(t) = \sum_{i=0}^3 \mathbf{P}_i B_i(t)$$

Pyramid Algorithm

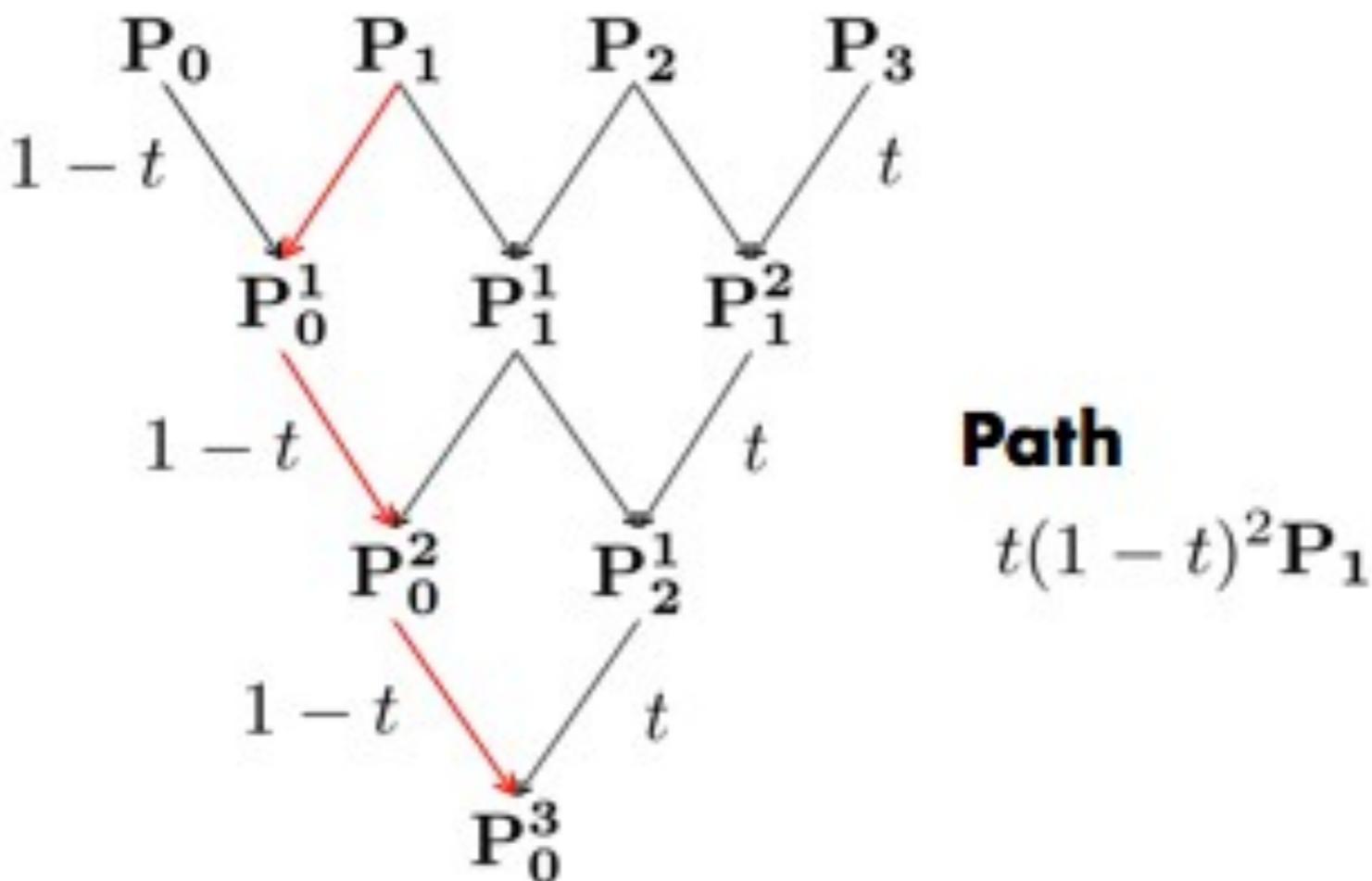


$$\mathbf{P}(t) = \sum_{i=0}^3 \mathbf{P}_i B_i(t)$$

Pyramid Algorithm



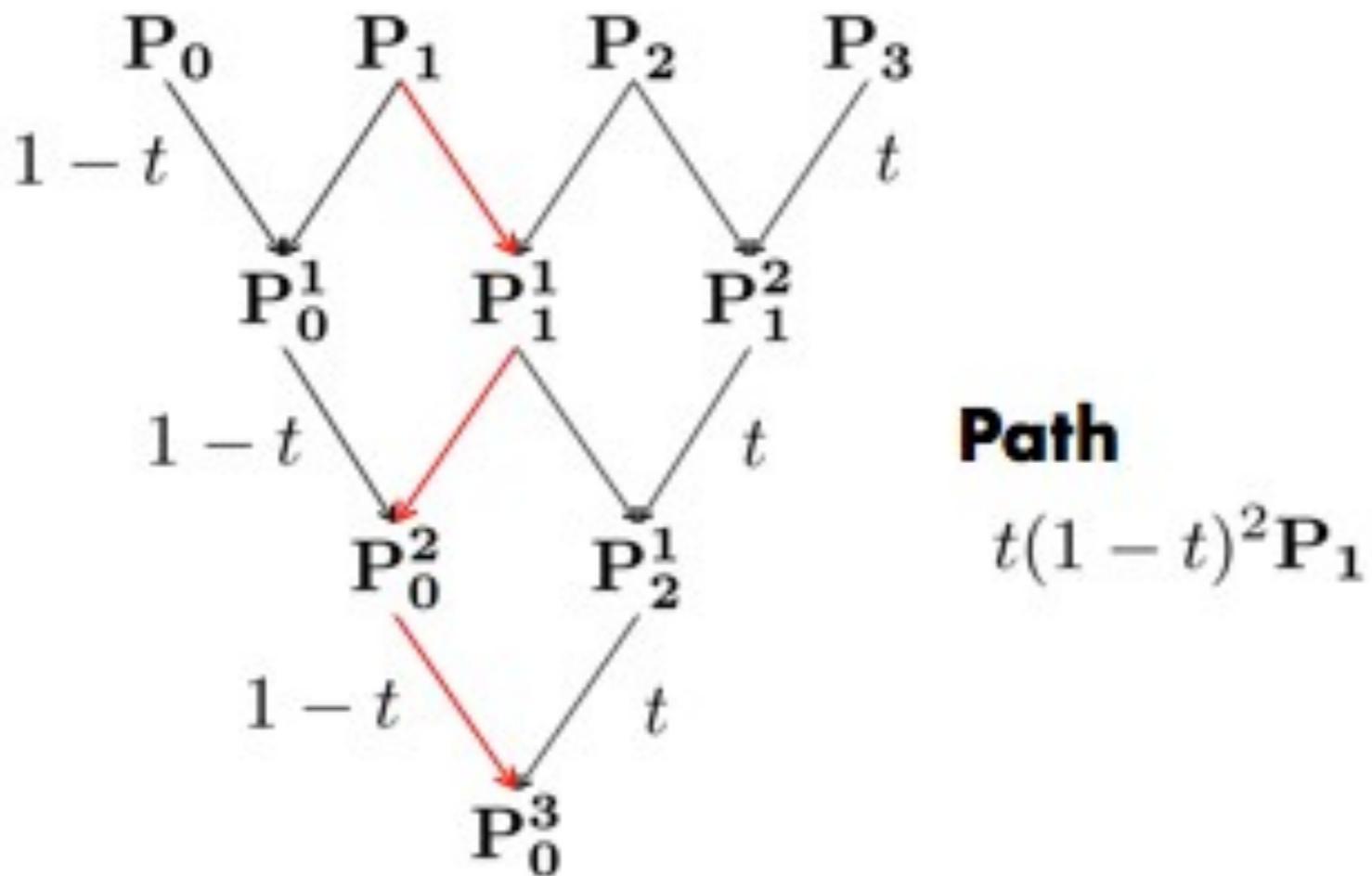
Pyramid Algorithm



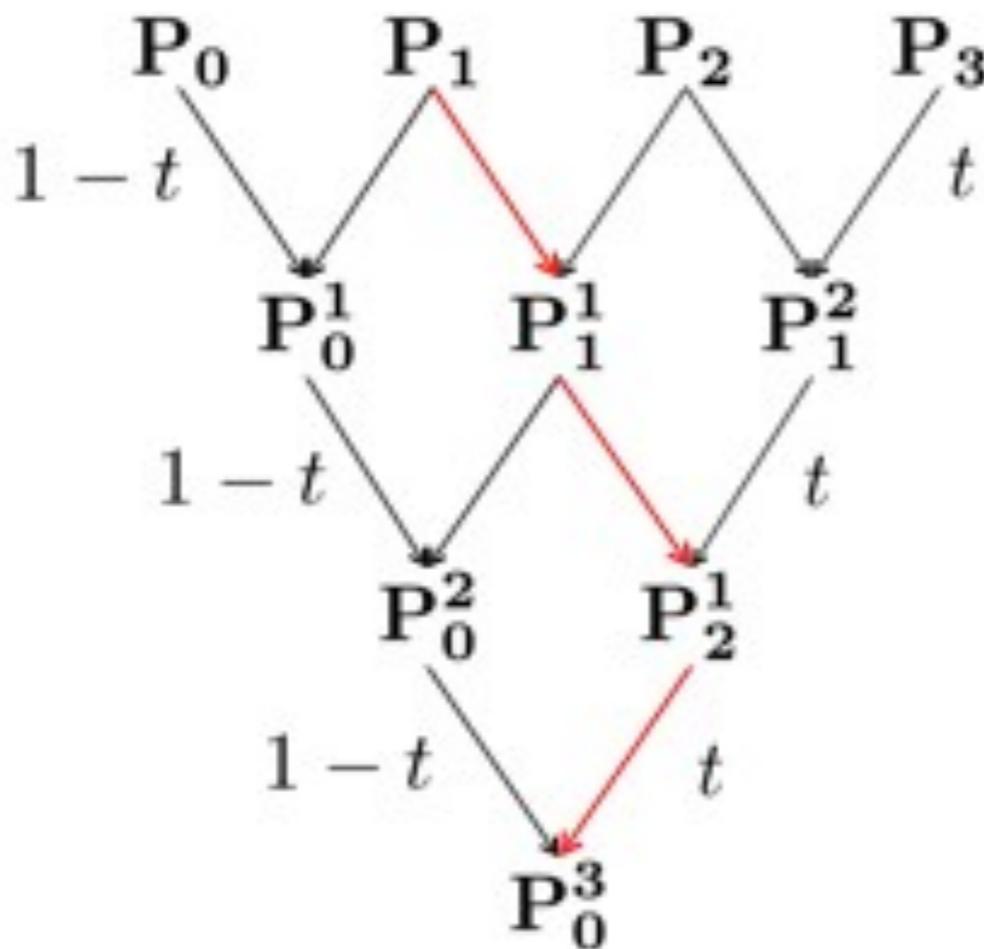
Path

$$t(1-t)^2 \mathbf{P}_1$$

Pyramid Algorithm



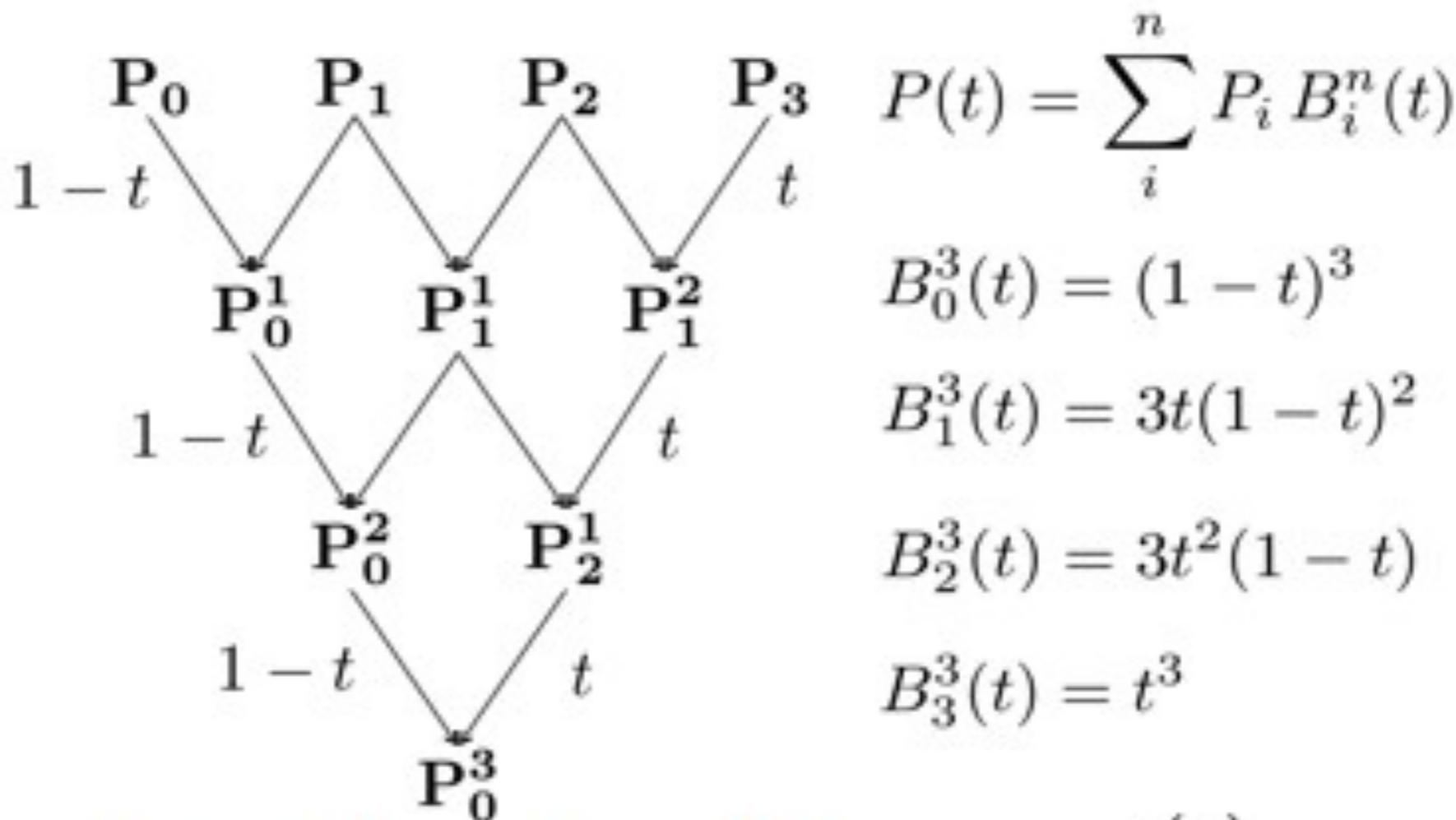
Pyramid Algorithm



Three paths total

$$3t(1-t)^2 P_0^1$$

Leads to a Cubic Polynomial Curve



Bernstein polynomials $B_i^n(t) = \binom{n}{i} t^i (1-t)^{n-i}$

Rational Bézier Curve

$$R(t) = \frac{\sum_{i=0}^n B_{i,n}(t) \omega_i P_i}{\sum_{i=0}^n B_{i,n}(t) \omega_i} = \sum_{i=0}^n R_{i,n}(t) P_i$$

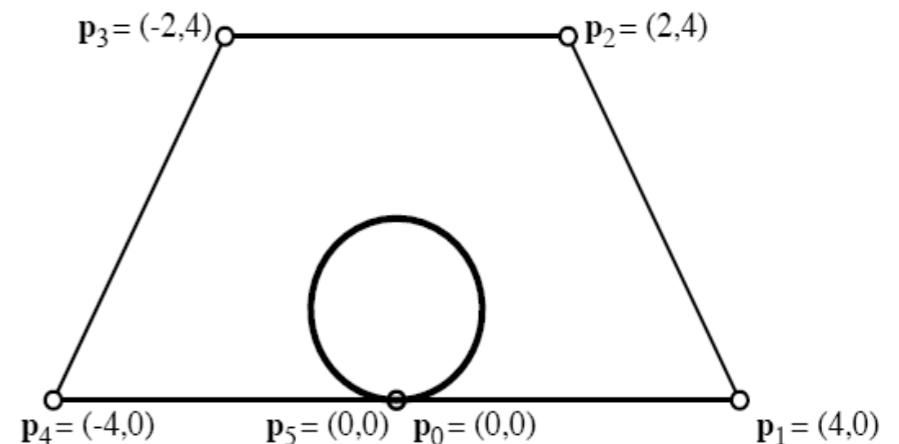


Figure 2.19: Circle as Degree 5 Rational Bézier Curve.

where $B_{i,n}(t)$ is Bernstein basis, ω_i is the weight at p_i .

It's a generalization of Bézier curve, which can express more curves, such as circle.

Bézier curve

Properties of rational Bézier curve:

1. endpoints: $\mathbf{R}(0) = \mathbf{P}_0$; $\mathbf{R}(1) = \mathbf{P}_n$

2. tangent of
endpoints:

$$\mathbf{R}'(0) = n \frac{\omega_1}{\omega_0} (\mathbf{P}_1 - \mathbf{P}_0); \quad \mathbf{R}'(1) = n \frac{\omega_{n-1}}{\omega_n} (\mathbf{P}_n - \mathbf{P}_{n-1})$$

3. Convex Hull Property

.....

5.

6. Influence of the weights

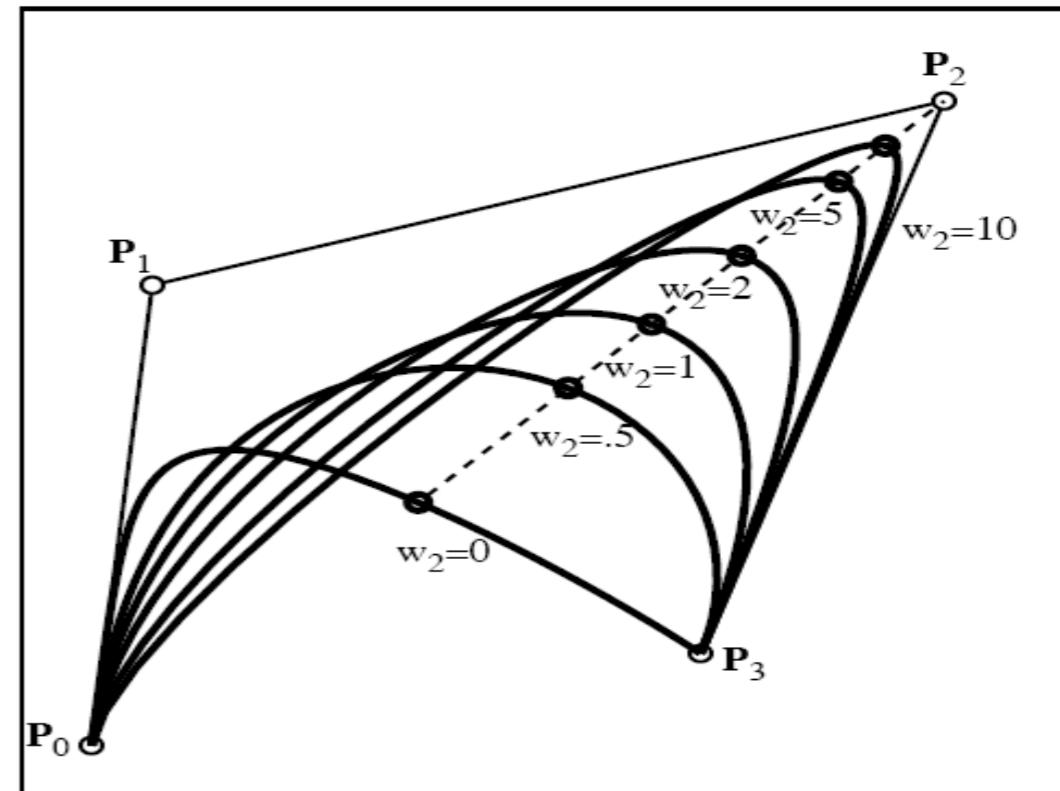


Figure 2.16: Rational Bézier curve.



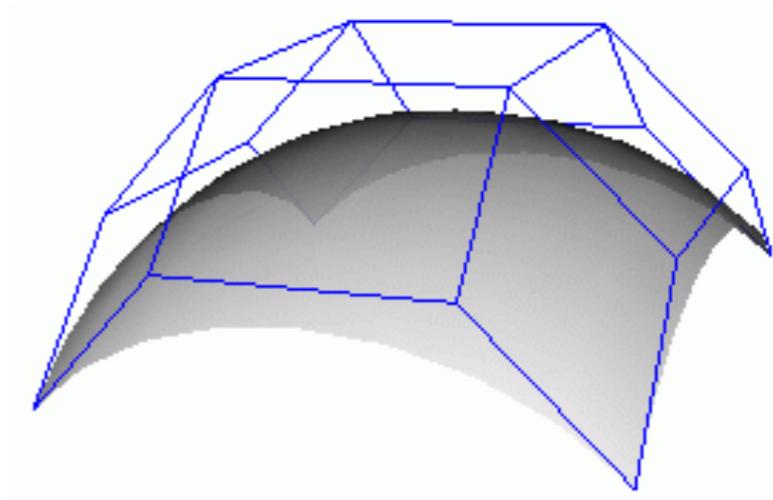
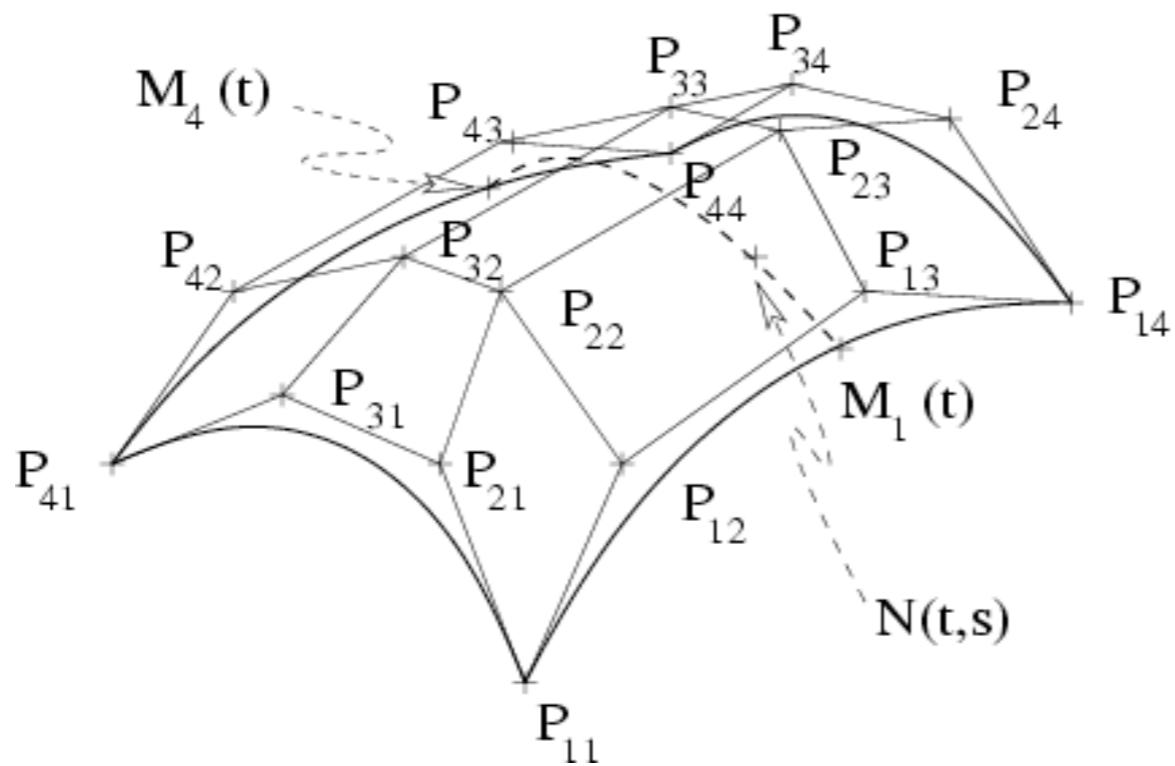
Bézier surface

Bézier surface

Bézier surface:

$$S(u, v) = \sum_{i=0}^n \sum_{j=0}^m P_{ij} B_{i,n}(u) B_{j,m}(v), \quad 0 \leq u, v \leq 1$$

where $B_{i,n}(u)$ and $B_{j,m}(v)$ Bernstein basis with n degree and m degree, respectively,
 $(n+1) \times (m+1)$ P_{ij} ($i=0, 1, \dots, n; j=0, 1, \dots, m$) construct the control meshes.



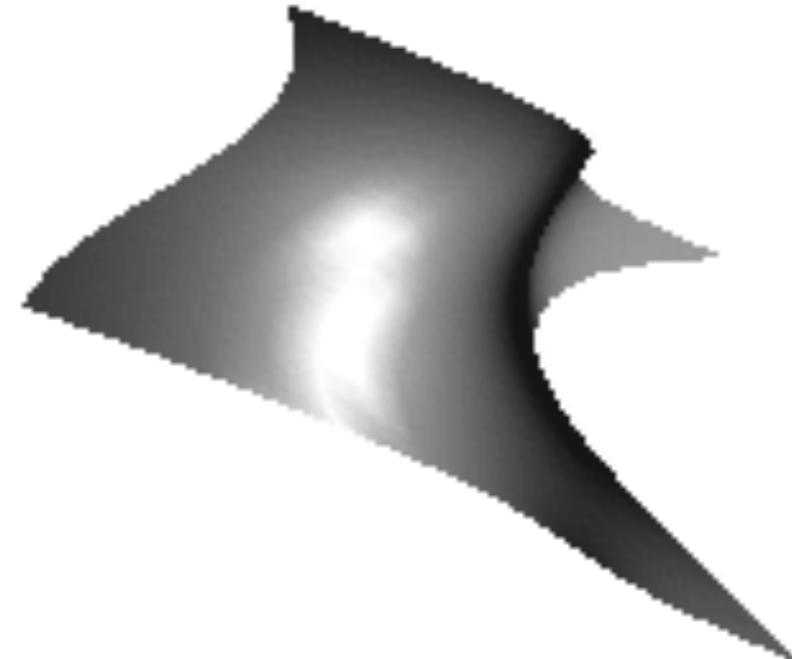
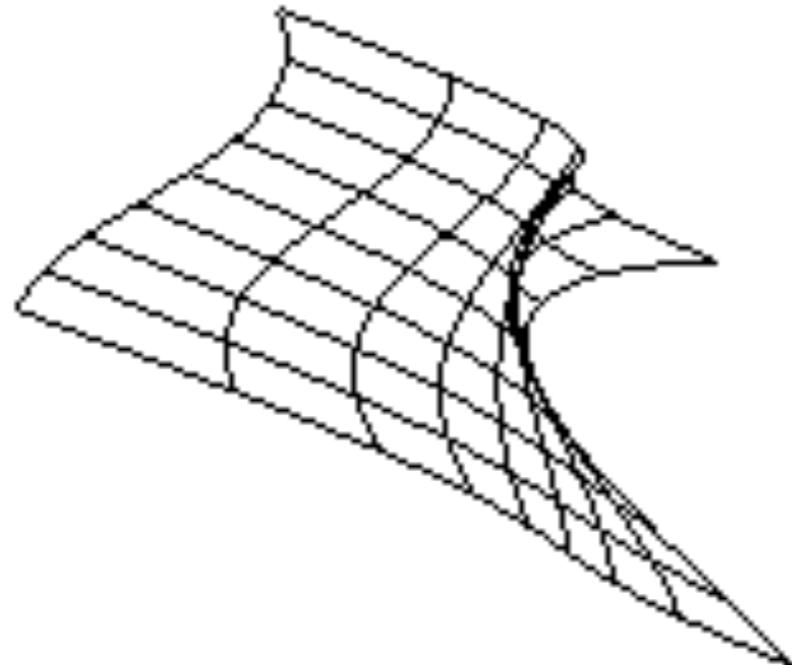
Bezier Surface in OpenGL

- `glMap2*(GL_MAP2_VERTEX_3,
 uMin, uMax, uStride, nuPts,
 vMin, vMax, vStride, nvPts, *ctrlPts);`
- `glEnable/glDisable(GL_MAP2_VERTEX_3);`
- `glBegin(GL_LINE_STRIP); / GL_QUAD_STRIP
for (...) {
 glEvalCoord2*(uValue, vValue);
}
glEnd();`

Bezier Surface in OpenGL

- `glBegin(GL_LINE_STRIP); // GL_QUAD_STRIP
for (...) {
 glEvalCoord2*(uValue, vValue);
}
glEnd();`

```
GLfloat ctrlpoints[4][4][3] = {  
    {{-1.5, -1.5, 4.0}, {-0.5, -1.5, 2.0},  
     {0.5, -1.5, -1.0}, {1.5, -1.5, 2.0}},  
    {{-1.5, -0.5, 1.0}, {-0.5, -0.5, 3.0},  
     {0.5, -0.5, 0.0}, {1.5, -0.5, -1.0}},  
    {{-1.5, 0.5, 4.0}, {-0.5, 0.5, 0.0},  
     {0.5, 0.5, 3.0}, {1.5, 0.5, 4.0}},  
    {{-1.5, 1.5, -2.0}, {-0.5, 1.5, -2.0},  
     {0.5, 1.5, 0.0}, {1.5, 1.5, -1.0}}};
```



Bézier surface

normal vector of Bézier surface

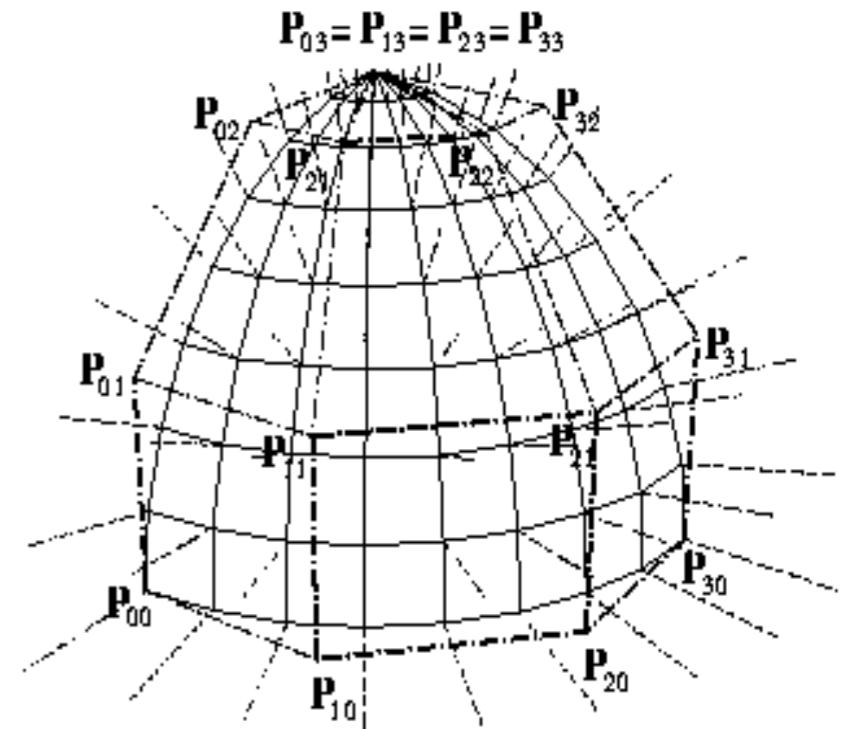
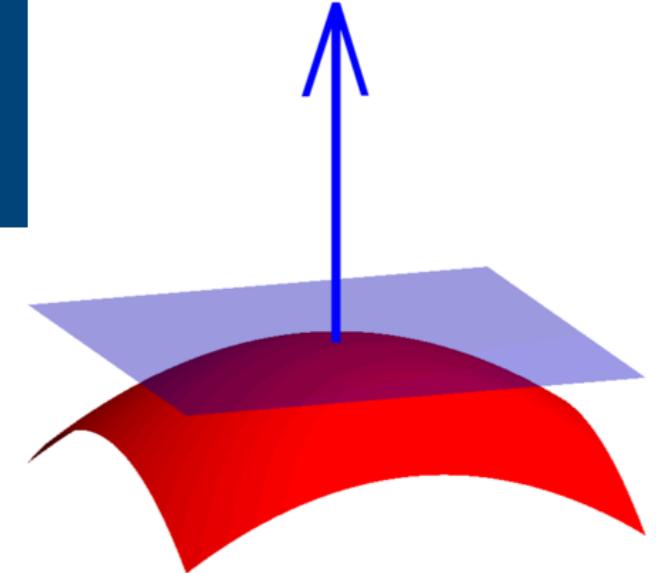
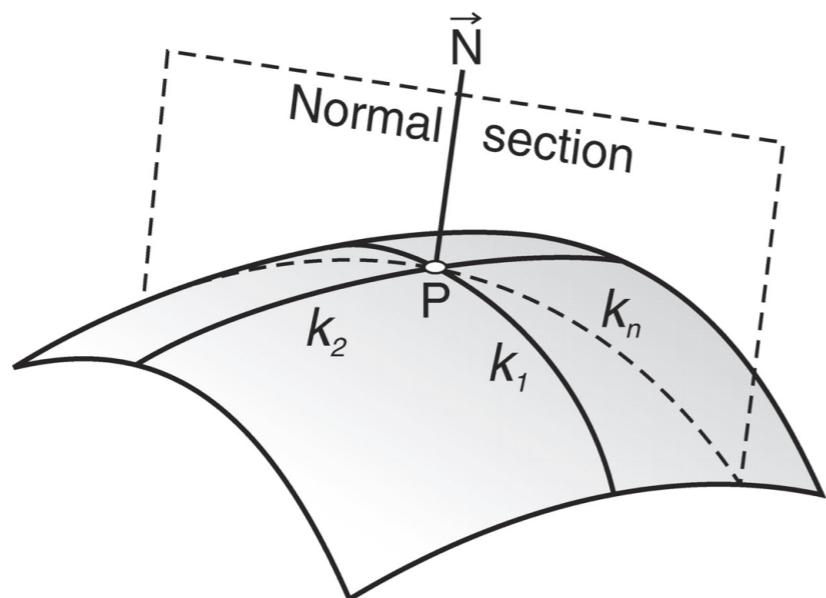
partial derivation of Bézier surface $\mathbf{S}(u,v)$:

$$\frac{\partial}{\partial u} \mathbf{S}(u,v) = \frac{\partial}{\partial u} \sum_{i=0}^n \sum_{j=0}^m \mathbf{P}_{ij} B_{i,n}(u) B_{j,m}(v) = n \sum_{i=0}^{n-1} \sum_{j=0}^m (\mathbf{P}_{i+1,j} - \mathbf{P}_{ij}) B_{i,n-1}(u) B_{j,m}(v)$$

$$\frac{\partial}{\partial v} \mathbf{S}(u,v) = \frac{\partial}{\partial v} \sum_{i=0}^n \sum_{j=0}^m \mathbf{P}_{ij} B_{i,n}(u) B_{j,m}(v) = m \sum_{i=0}^n \sum_{j=0}^{m-1} (\mathbf{P}_{i,j+1} - \mathbf{P}_{ij}) B_{i,n}(u) B_{j,m-1}(v)$$

normal $\mathbf{N}(u,v)$:

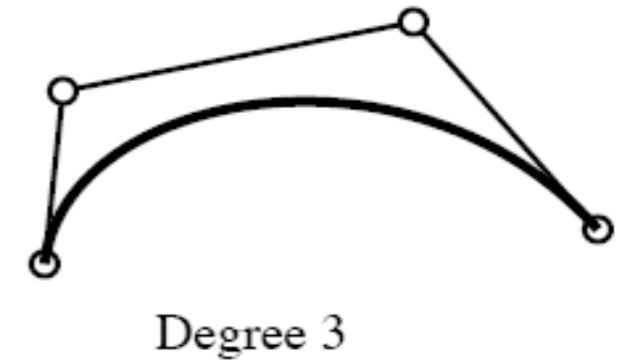
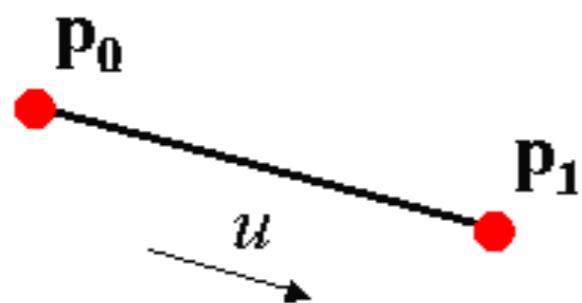
$$\mathbf{N}(u,v) = \frac{\partial \mathbf{S}(u,v)}{\partial u} \times \frac{\partial \mathbf{S}(u,v)}{\partial v}$$



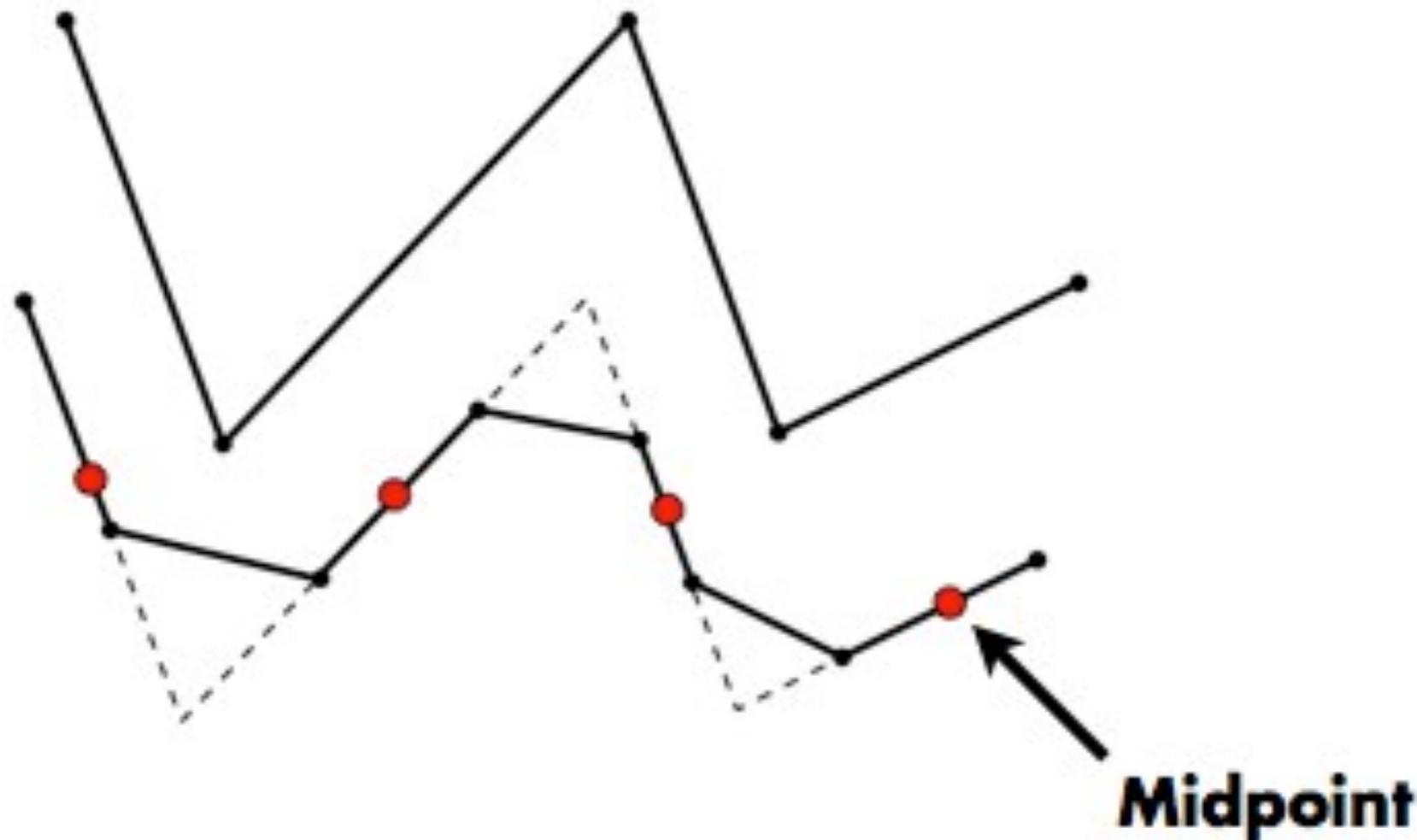
NURBS curve

Disadvantages of Bézier curve:

1. control points determine the degree of the curve; many control points means high degree.
2. It's **global**. A control point influences the whole curve.

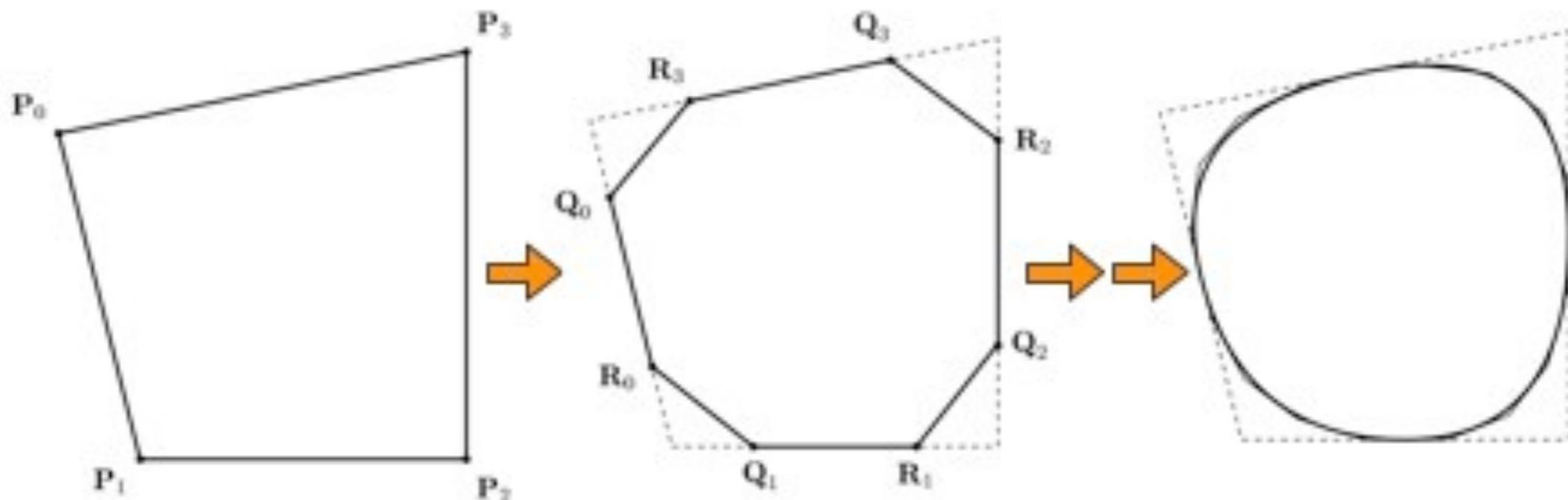


Corner Cutting Algorithm



Chaiken (1974)

Procedural Curve



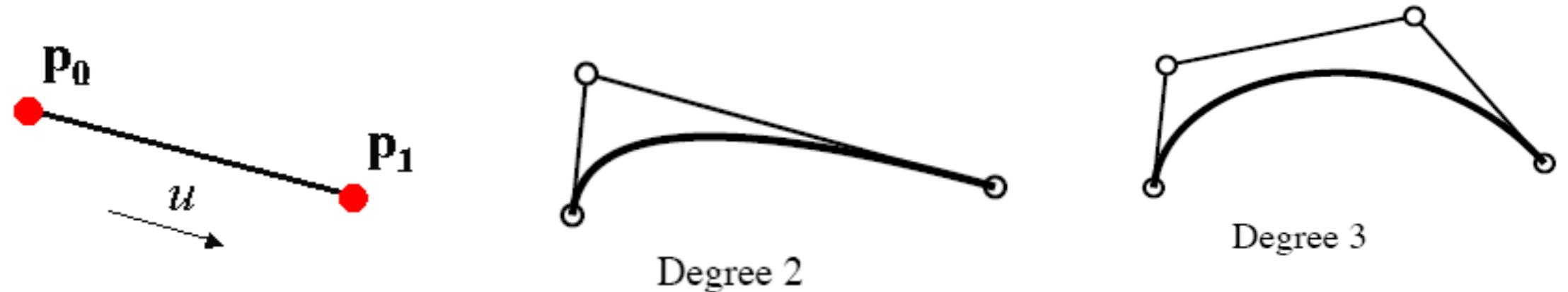
Repeatedly cutting corners generates a limit curve

1. Interpolates midpoints
2. Tangent preserved at midpoints

B-spline curve

- disadvantages of Bézier curve:

1. control points determine the degree of the curve. many control points means high degree.
2. It's global. A control point influences the whole curve.



de Boor et al. replaced Bernstein basis with B-spline basis to generate B-spline curve.



NURBS curve

B-spline curve:

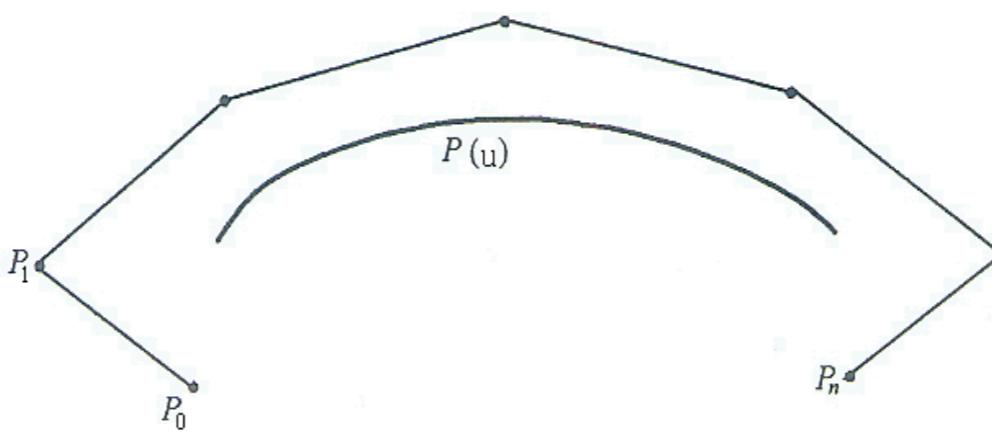
$$C(u) = \sum_{i=0}^n P_i N_{i,p}(u) \quad a \leq u \leq b$$

Where P_0, P_1, \dots, P_n are control points, $\mathbf{u} = [u_0=a, u_1, \dots, u_i, \dots, u_{n+k+1}=b]$.

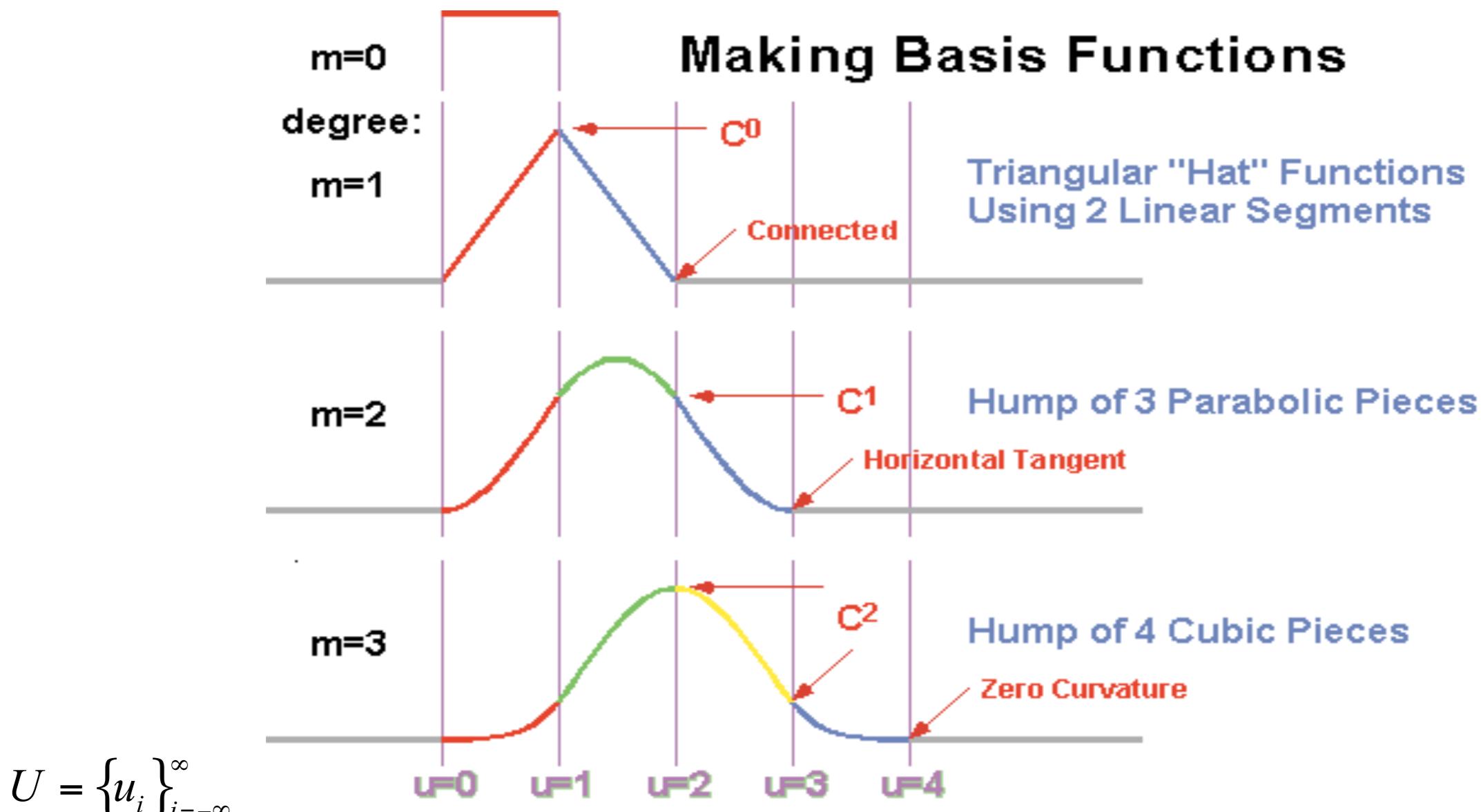
$$N_{i,0}(u) = \begin{cases} 1 & u_i \leq u < u_{i+1} \\ 0 & otherwise \end{cases}$$

$$N_{i,p}(u) = \frac{u - u_i}{u_{i+p} - u_i} N_{i,p-1}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}(u),$$

$$\frac{0}{0} = 0$$



B-spline basis



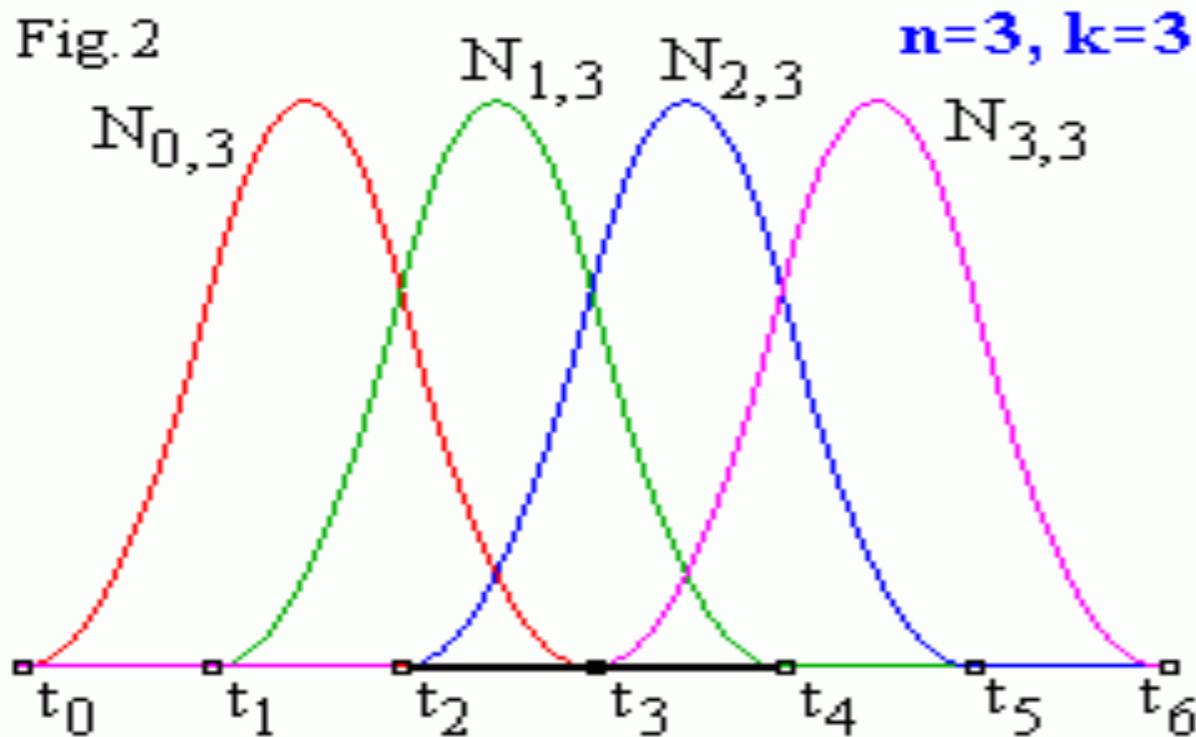
$$N_{i,0}(u) = \begin{cases} 1 & u_i \leq u < u_{i+1} \\ 0 & \text{else} \end{cases}$$

$$N_{i,p}(u) = \frac{u - u_i}{u_{i+p} - u_i} N_{i,p-1}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}(u),$$

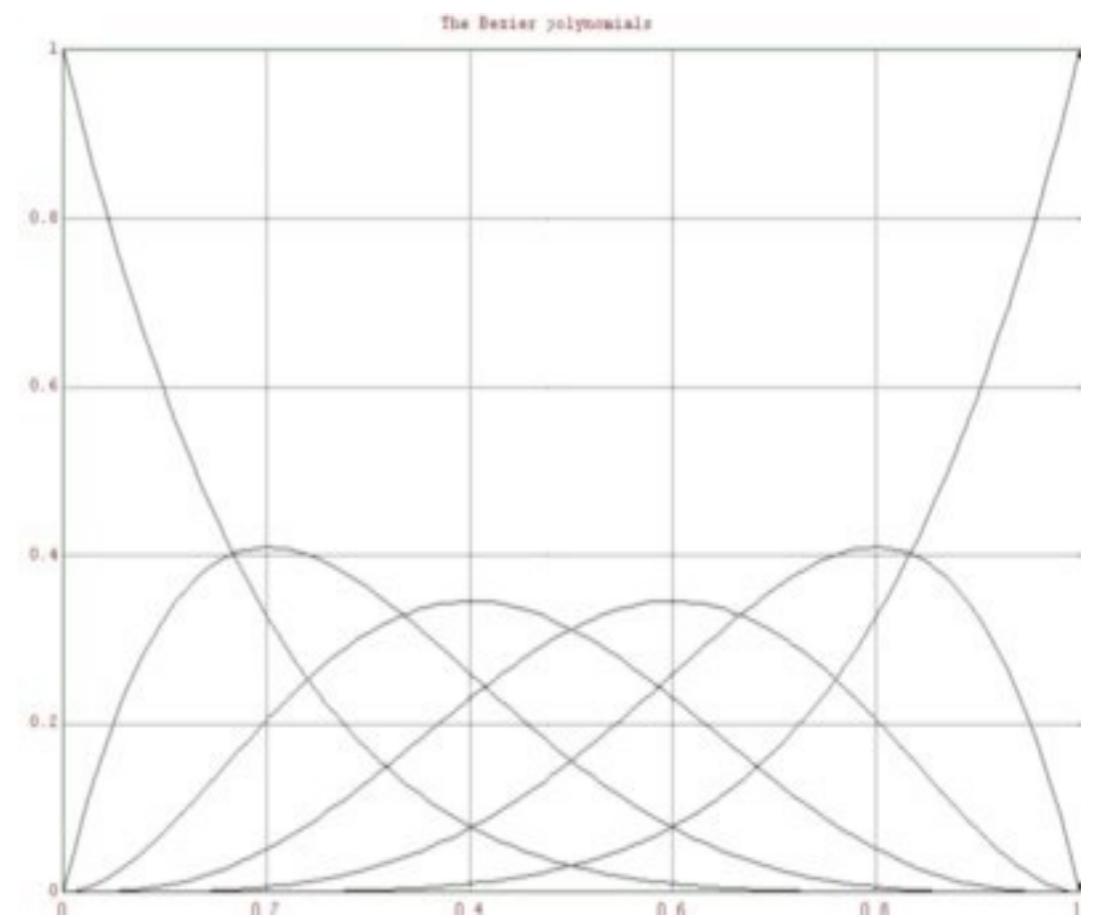
$$\frac{0}{0} = 0$$

B-spline basis v.s. Bernstein ~

Fig.2



$$B_{i,n}(t) = (1-t)B_{i,n-1}(t) + tB_{i-1,n-1}(t), \quad i = 0, 1, \dots, n.$$



$$N_{i,0}(u) = \begin{cases} 1 & u_i \leq u < u_{i+1} \\ 0 & \text{else} \end{cases}$$

$$U = \{u_i\}_{i=-\infty}^{\infty}$$

$$N_{i,p}(u) = \frac{u - u_i}{u_{i+p} - u_i} N_{i,p-1}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}(u),$$

$$\frac{0}{0} = 0$$

NURBS curve

properties of B-spline basis

1. localization: $N_{i,p}(u) > 0$ only when $u \in [u_i, u_{i+p+1}]$.

$$N_{i,p}(u) = \begin{cases} > 0, & u_i \leq u < u_{i+p+1} \\ = 0, & u < u_i \text{ or } u > u_{i+p+1} \end{cases}$$

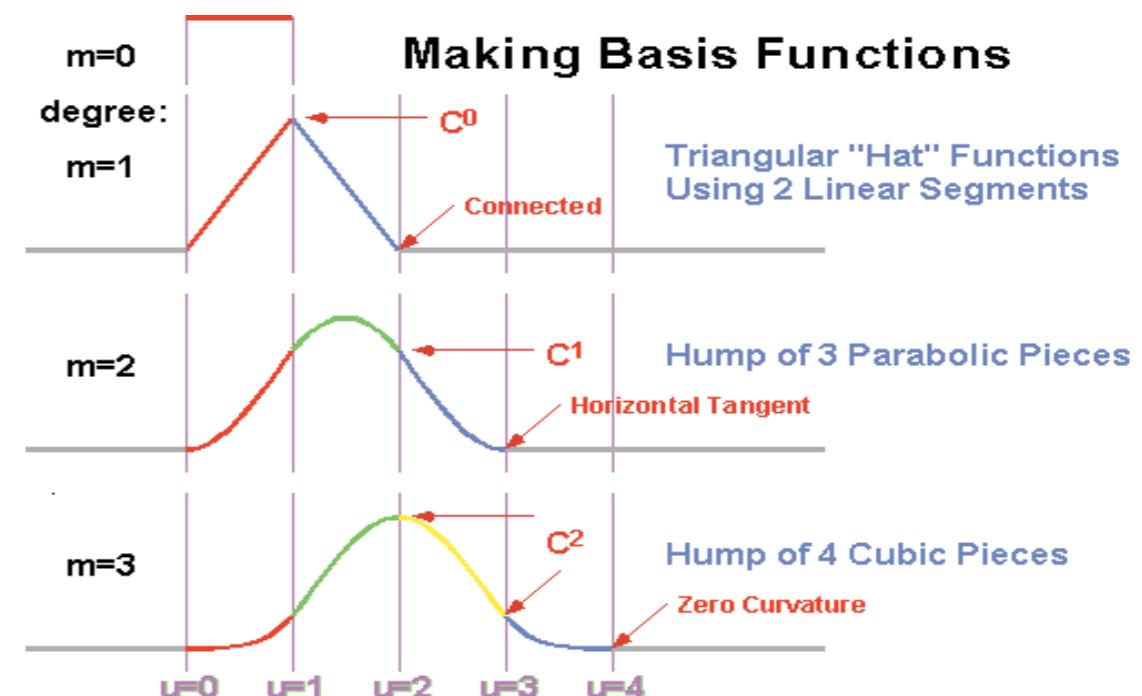
2. normalization:

$$\sum_{j=-\infty}^{\infty} N_{j,p}(u) = \sum_{j=i-p}^i N_{j,p}(u) = 1, u \in [u_i, u_{i+1})$$

3. piecewise polynomial: $N_{i,p}(u)$ is a polynomial with degree $< p$, in every $[u_j, u_{j+1})$

4. differential:

$$N'_{i,p}(u) = \frac{p}{u_{i+p} - u_i} N_{i,p-1}(u) - \frac{p}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}(u)$$



NURBS curve

Properties of B-spline curve:

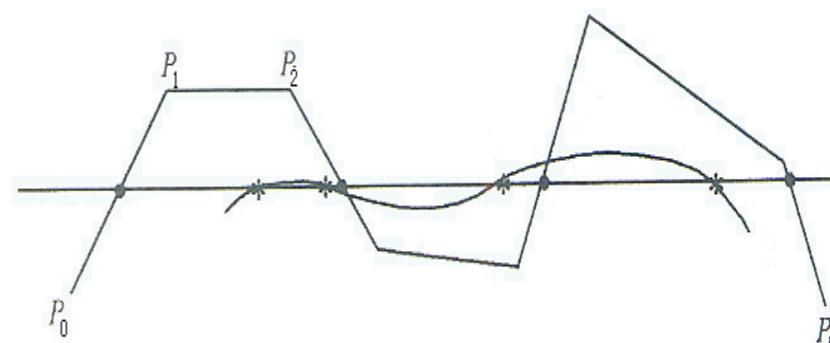
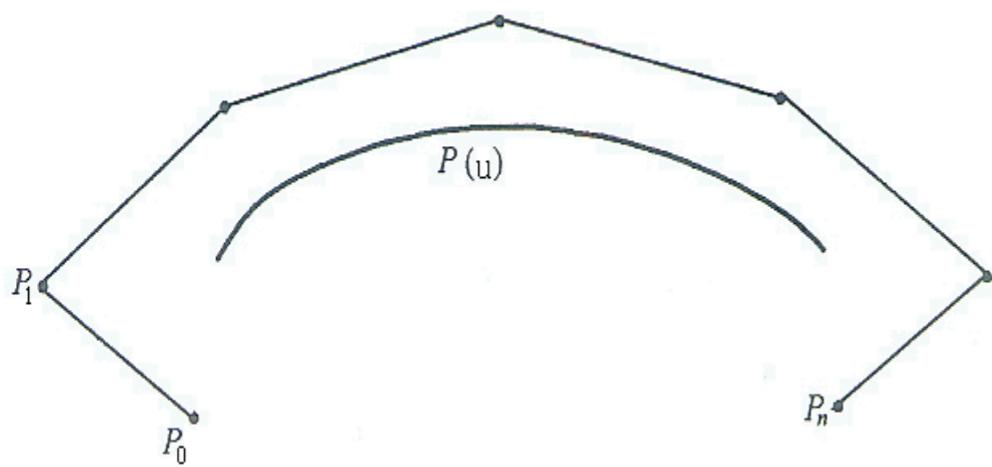
1. Convex Hull Property

2. variation diminishing property.

3. Affine Invariance

4. local

5. piecewise polynomial



NURBS curve

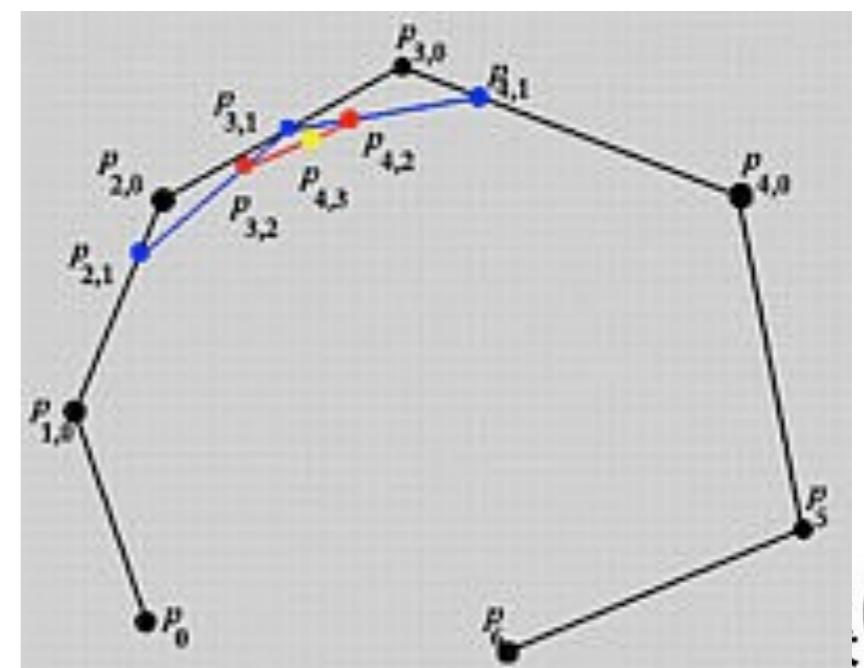
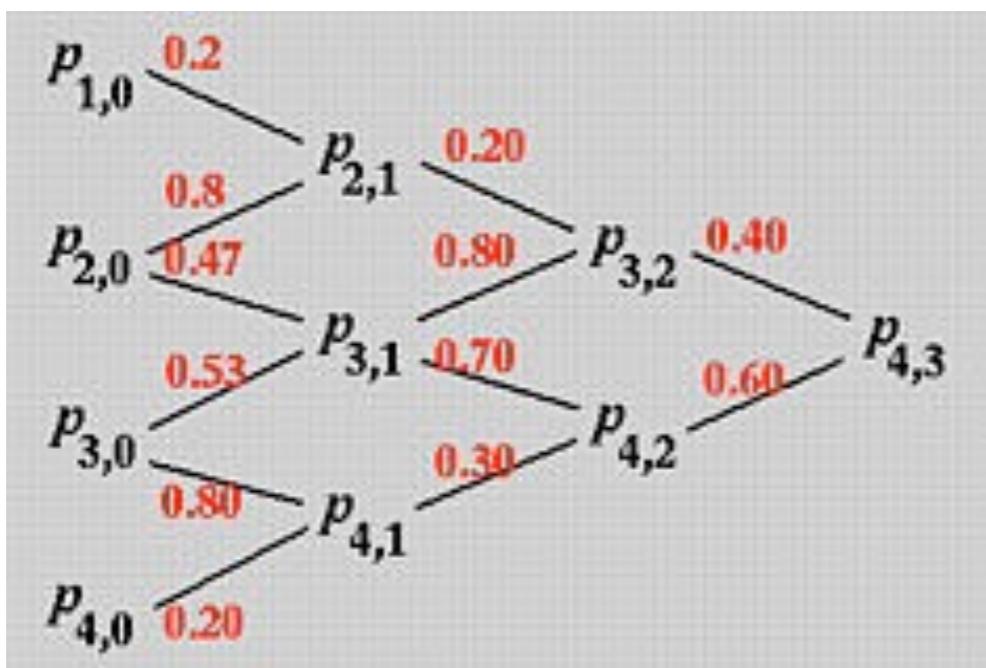
B-spline---de Boor algorithm

to calculate the point of B-spline curve $\mathbf{C}(u)$ at u :

1. find the interval where u lies in : $u \in [u_j, u_{j+1})$;
2. curve in $u \in [u_j, u_{j+1})$ is only determined by $\mathbf{P}_{j-p}, \mathbf{P}_{j-p+1}, \dots, \mathbf{P}_j$;
3. calculate

$$\mathbf{P}_i^r(u) = \begin{cases} \mathbf{P}_i & r = 0, i = j - p; j - p + 1, L, j; \\ \frac{u - u_i}{u_{i+k-r} - u_i} \mathbf{P}_i^{r-1}(u) + \frac{u_{i+k-r} - u}{u_{i+k-r} - u_i} \mathbf{P}_{i-1}^{r-1}(u), & r = 1, 2, L, k - 1; i = j - p + r, j - p + r + 1, L, j. \end{cases}$$

4. $\mathbf{P}_j^{k-1}(u) = C(u)$



NURBS curve

Catmull-Clark and Doo-Sabin subdivision

Start from

$$P^i = (\mathbf{L}^-, p_{-1}^i, p_0^i, p_1^i, p_2^i, \mathbf{L}^+)$$

Catmull-Clark rules

$$p_{2j}^{i+1} = \frac{1}{8} p_{j-1}^i + \frac{6}{8} p_j^i + \frac{1}{8} p_{j+1}^i$$

$$p_{2j+1}^{i+1} = \frac{4}{8} p_j^i + \frac{4}{8} p_{j+1}^i$$

Doo-Sabin rules:

$$p_{2j}^{i+1} = \frac{3}{4} p_j^i + \frac{1}{4} p_{j+1}^i$$

$$p_{2j+1}^{i+1} = \frac{1}{4} p_j^i + \frac{3}{4} p_{j+1}^i$$

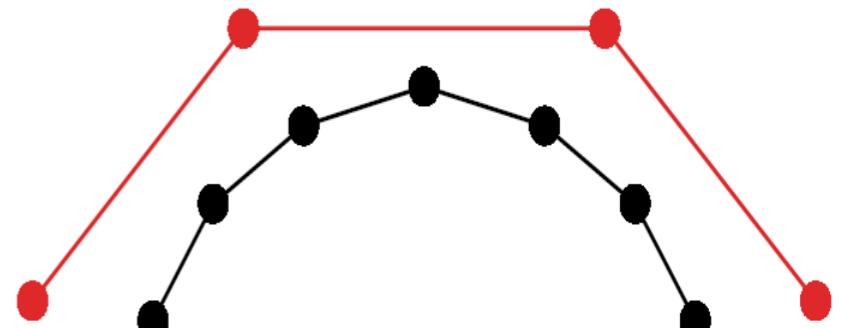


Figure 3: Subdividing an initial set of control points (upper, red) results in additional control points (lower, black), that more closely approximate a curve.

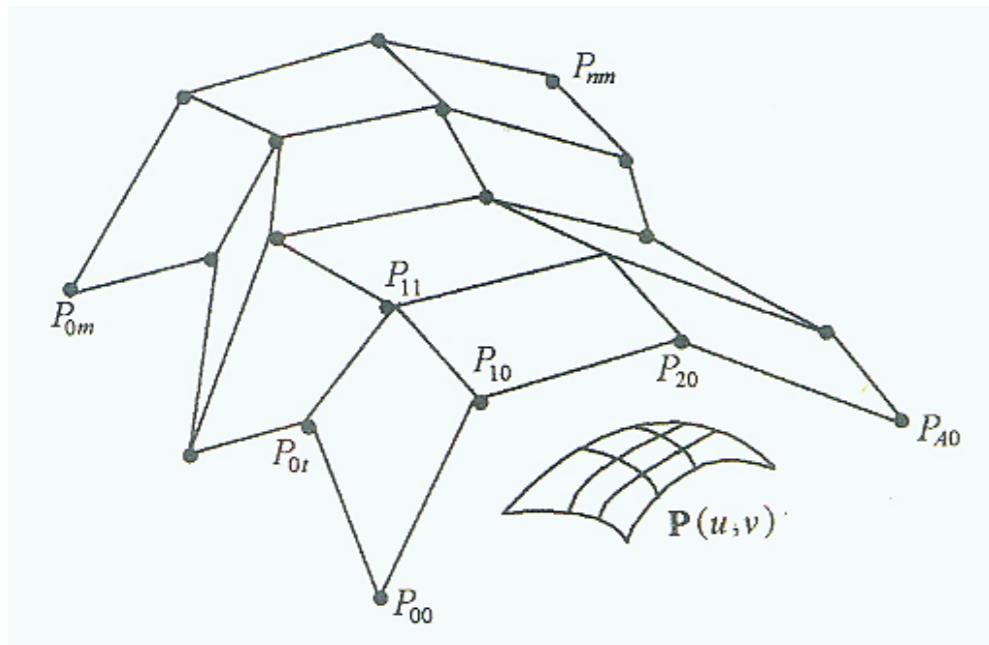
B-spline surface

$(n+1) \times (m+1)$ control points: $\mathbf{P}_{i,j}$ (Degrees of u, v : p, q);

nodes: $U = [u_0, u_1, \dots, u_{n+p+1}]$, $V = [v_0, v_1, \dots, v_{m+q+1}]$,

Then a tensor B-spline surface with degree $p \times q$:

$$S(u, v) = \sum_{i=0}^n \sum_{j=0}^m N_{i,p}(u) N_{j,q}(v) \mathbf{P}_{i,j}$$



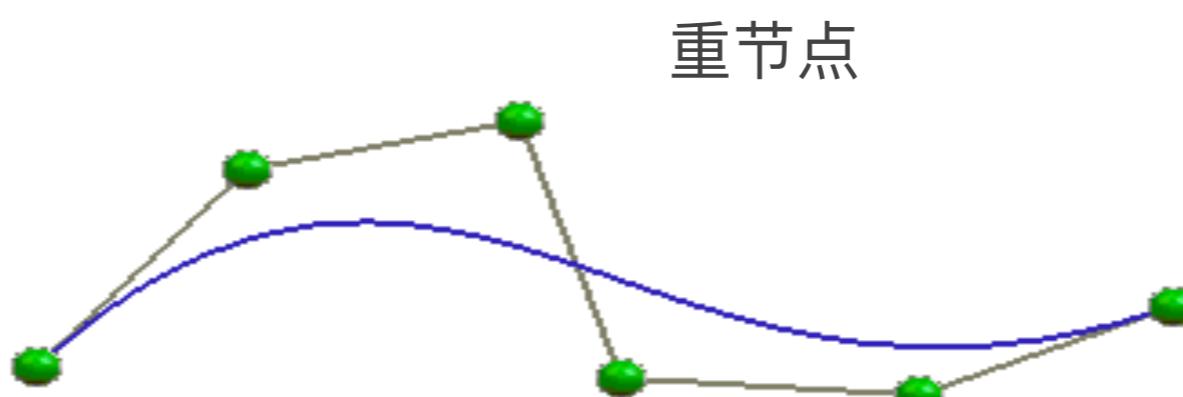
NURBS surface

NURBS (Non-uniform **Rational** B-spline)

NURBS curves:

$$\mathbf{C}(u) = \frac{\sum_{i=0}^n N_{i,p}(u) \omega_i \mathbf{P}_i}{\sum_{i=0}^n N_{i,p}(u) \omega_i}, \quad a \leq u \leq b$$

$$U = \{ \underbrace{a, \dots, a}_{p+1}, u_{p+1}, \dots, u_{m-p-1}, \underbrace{b, \dots, b}_{p+1} \}$$



NURBS surface

NURBS surface

$$S(u, v) = \frac{\sum_{i=0}^n \sum_{j=0}^m N_{i,p}(u) N_{j,q}(v) \omega_{i,j} P_{i,j}}{\sum_{i=0}^n \sum_{j=0}^m N_{i,p}(u) N_{j,q}(v) \omega_{i,j}} \quad 0 \leq u, v \leq 1$$

ω_{ij} : weights

$$U = \{0, \dots, 0, u_{p+1}, \dots, u_{r-p-1}, 1, \dots, 1\}$$

$$\underbrace{p+1}_{p+1} \qquad \qquad \qquad \underbrace{p+1}_{p+1}$$

$$V = \{0, \dots, 0, v_{q+1}, \dots, v_{s-q-1}, 1, \dots, 1\}$$

$$\underbrace{q+1}_{q+1} \qquad \qquad \qquad \underbrace{q+1}_{q+1}$$

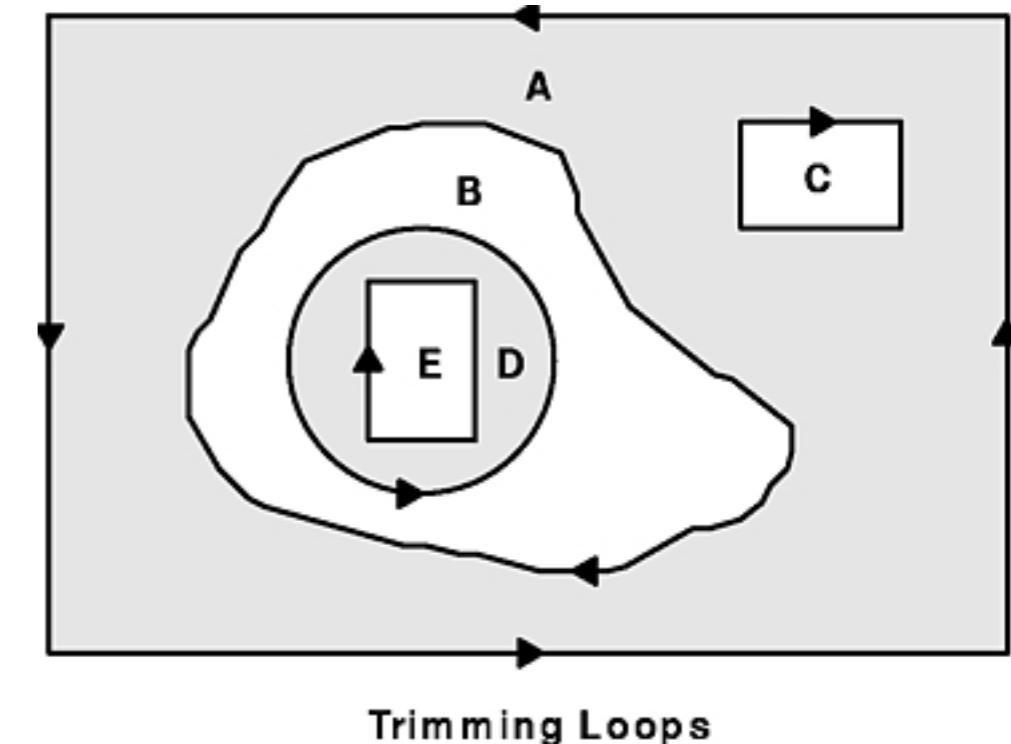
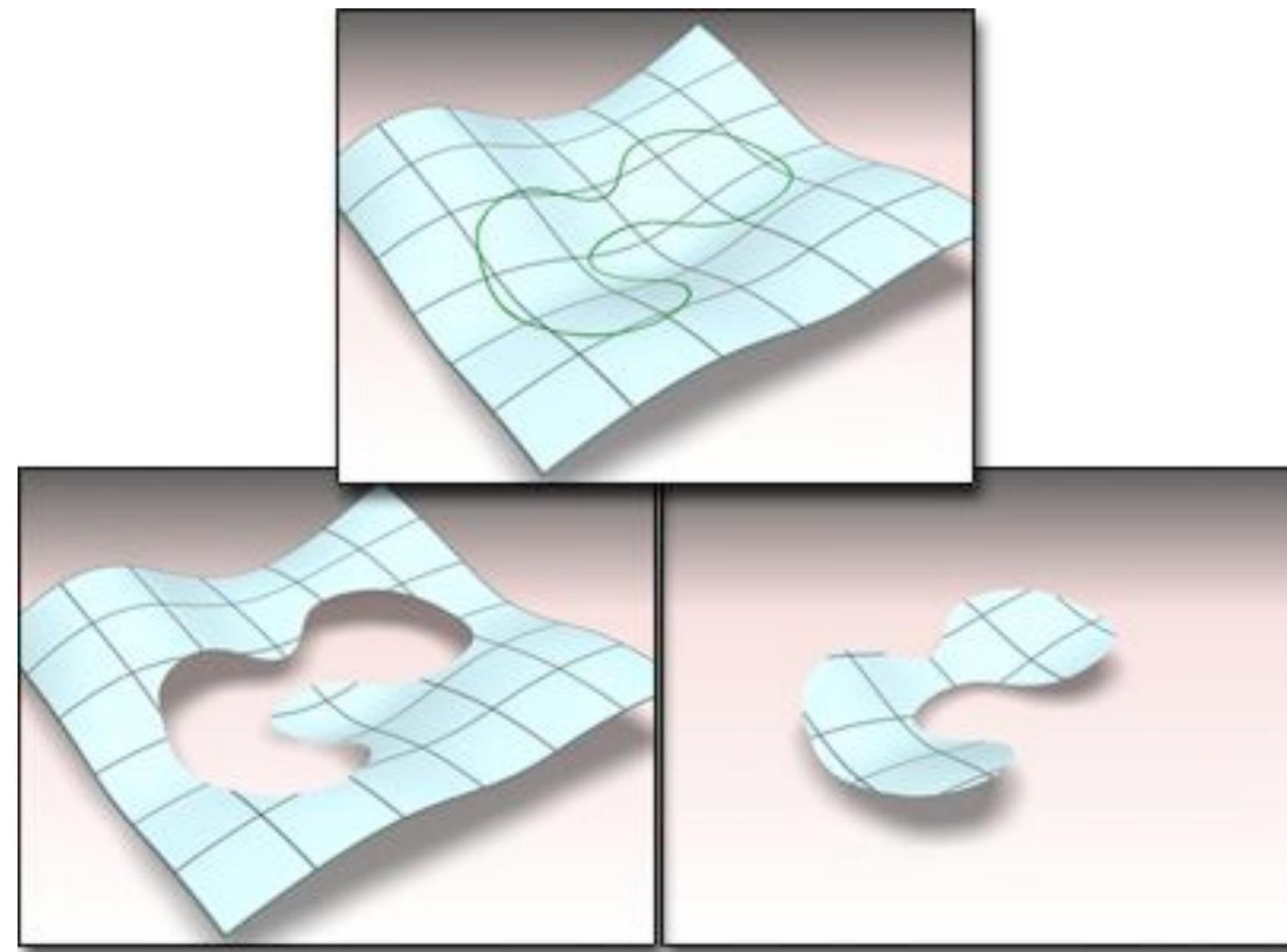
NURBS in OpenGL

- `curveName = gluNewNurbsRenderer();
gluBeginCurve (curveName);`

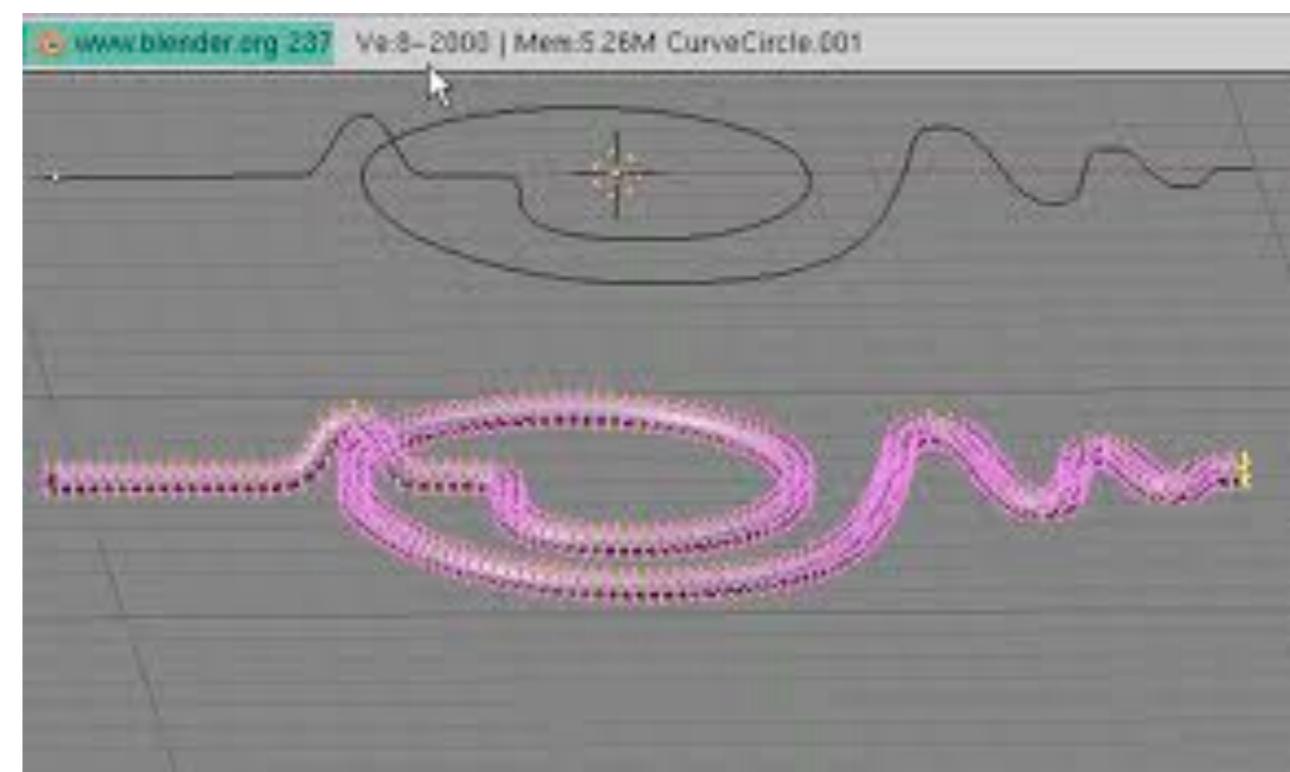
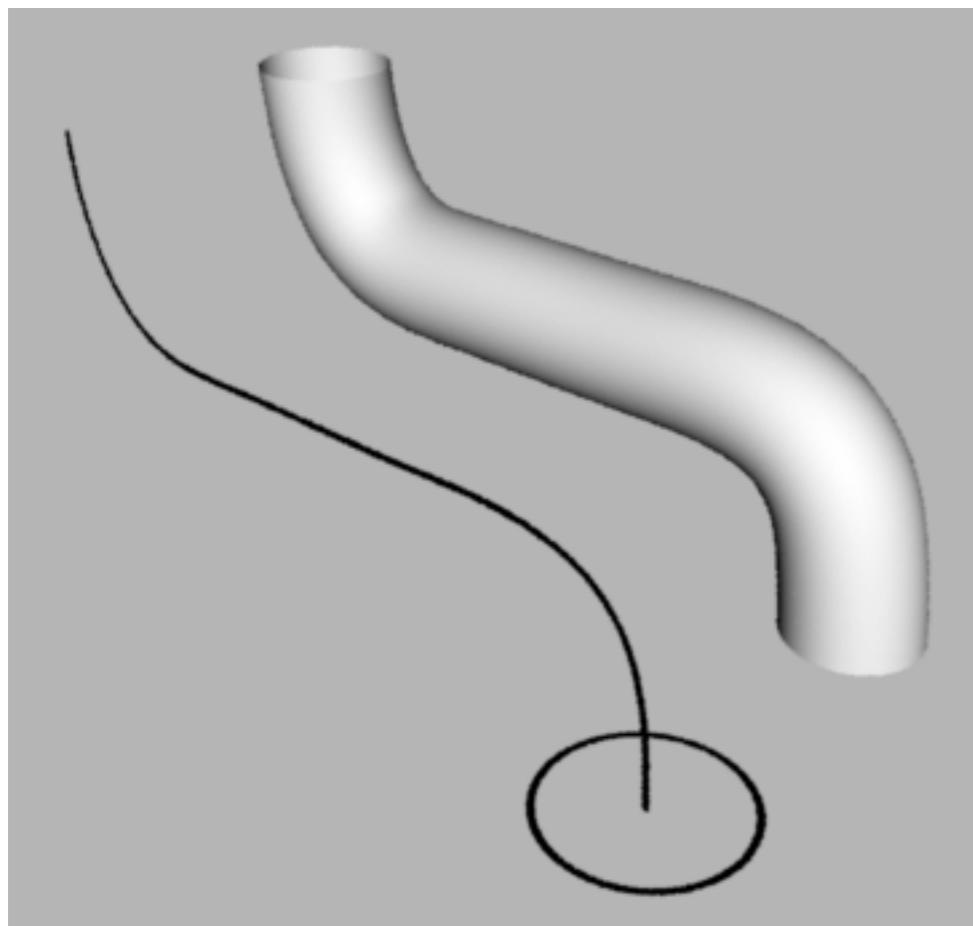
```
gluNurbsCurve(curveName, nknots, *knotVector,  
stride,*ctrlPts, order, GL_MAPI_VERTEX_3);
```

```
gluEndCurve(curveName);
```

Surface trimming



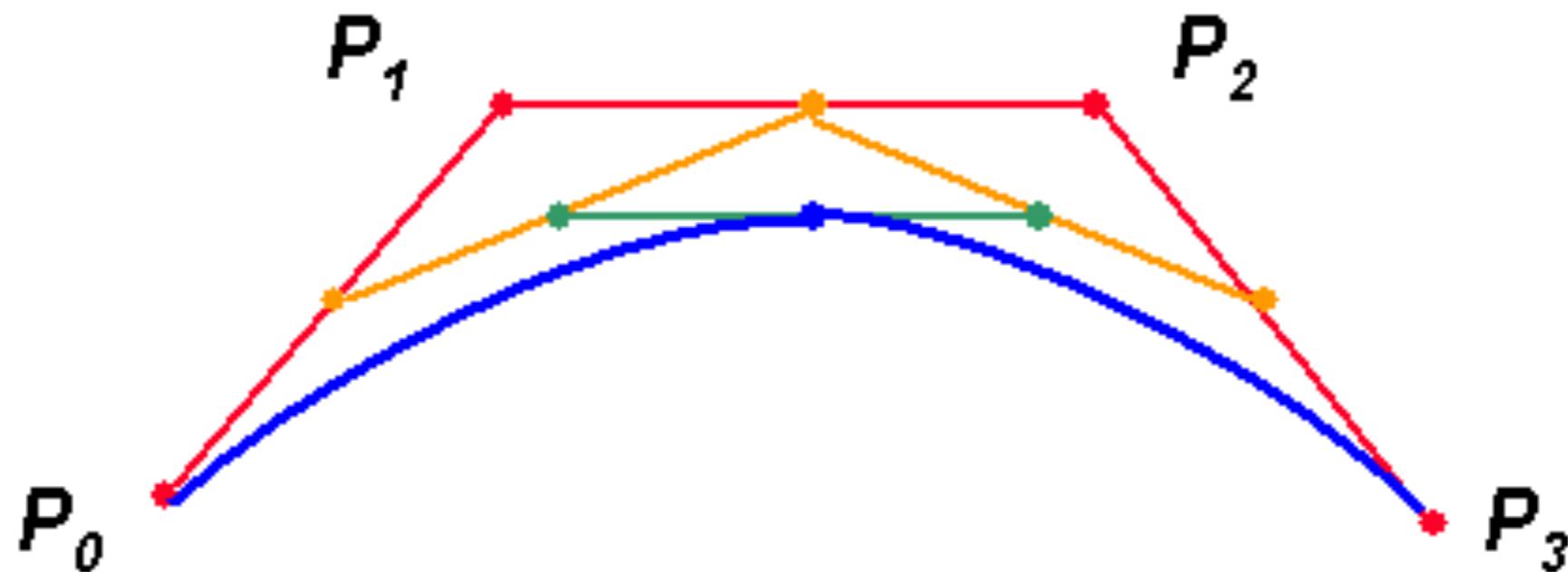
Sweeping



subdivision surface

subdivision curves:

- starting from a set of points, generate new points in every step under some rules, when such step goes on infinitely, the points will be convergent to a smooth curve.



subdivision surface

