Computer Graphics 2016

10. Spline and Surfaces

Hongxin Zhang
State Key Lab of CAD&CG, Zhejiang University

2016-12-05

Outline

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

classification of curves

$y = x^2 + 5x + 3 \longrightarrow y = f(x)$

(explicit curve)

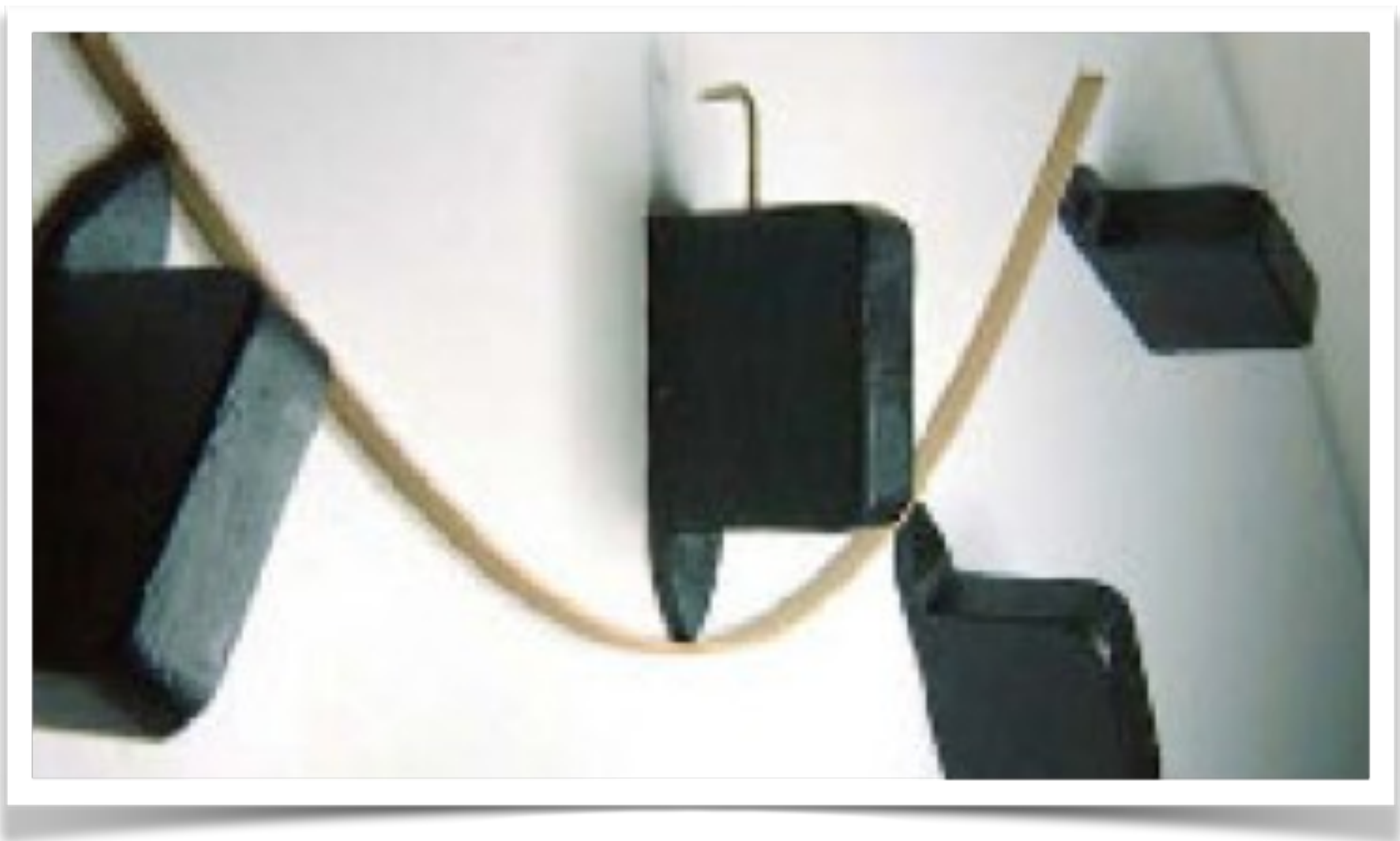
$(x - x_c)^2 + (y - y_c)^2 - r^2 = 0 \longrightarrow g(x,y) = 0$

(implicit curve)

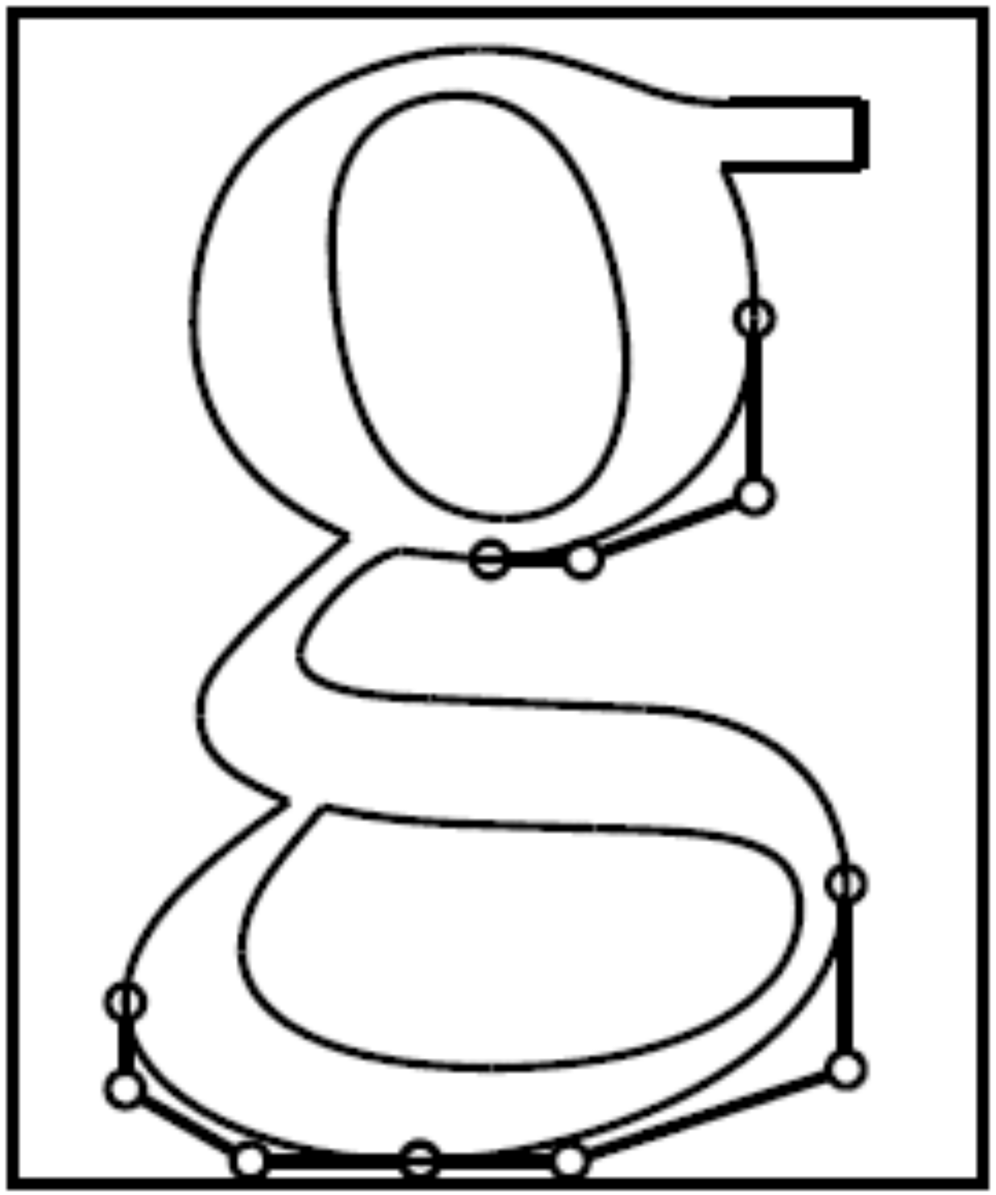
$$\begin{aligned} x &= x_c + r \cdot \cos\theta \\ y &= y_c + r \cdot \sin\theta \end{aligned} \longrightarrow \begin{cases} x = x(t) \\ y = y(t) \end{cases}$$

(parametric curve)

Splines



Bézier curve



Pierre Étienne Bézier
an engineer at Renault

Bézier curve

Bézier curve

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

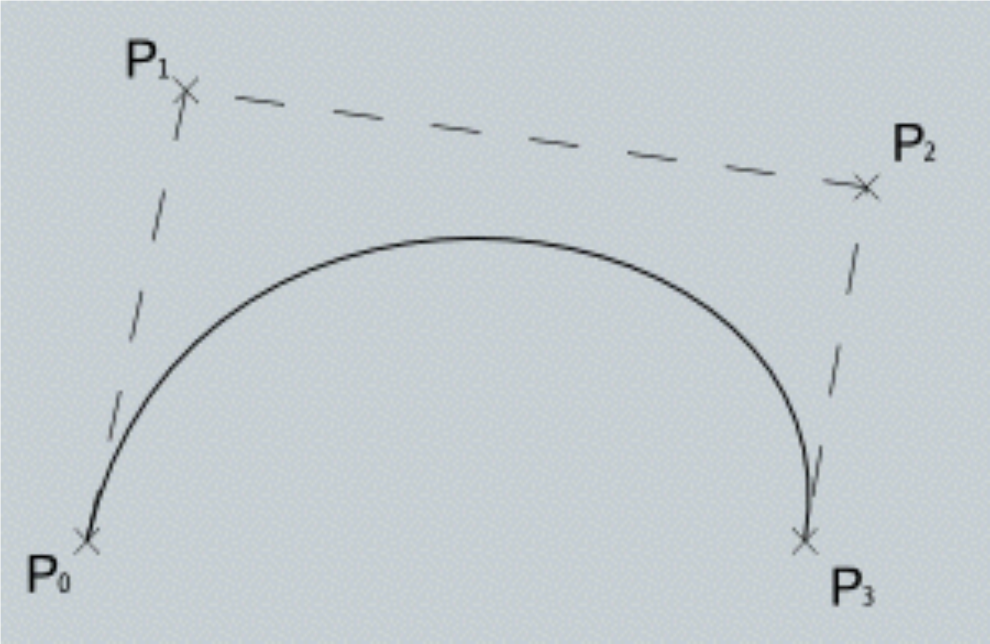
where, P_i (i=0,1,…,n) are control points.

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

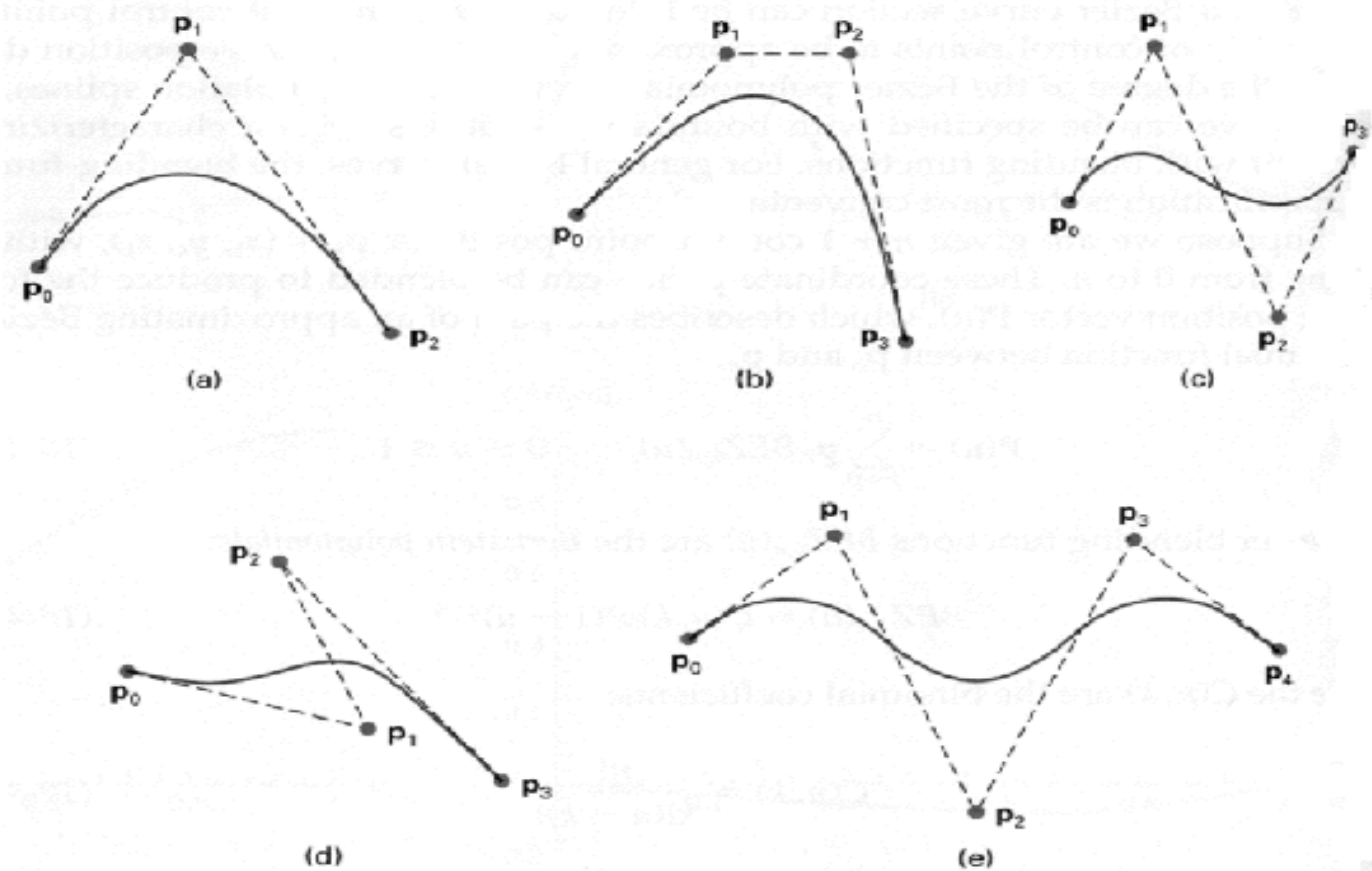
Bernstein basis

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

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



Bézier curve



General spline curves

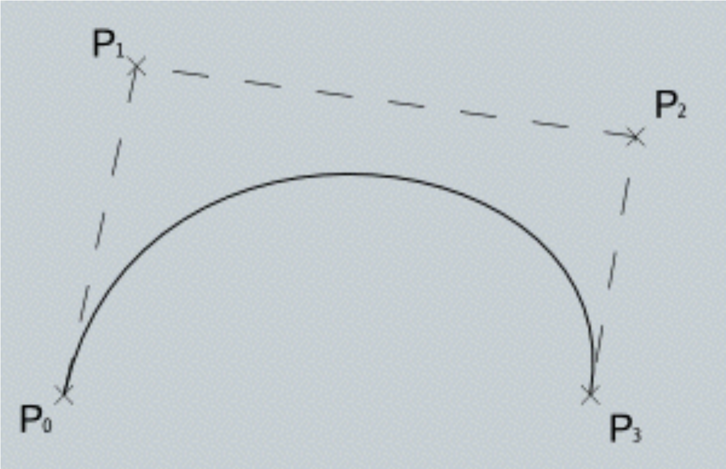
parametric curve

basis functions

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

$$t \in [t_0, t_1)$$

control pints



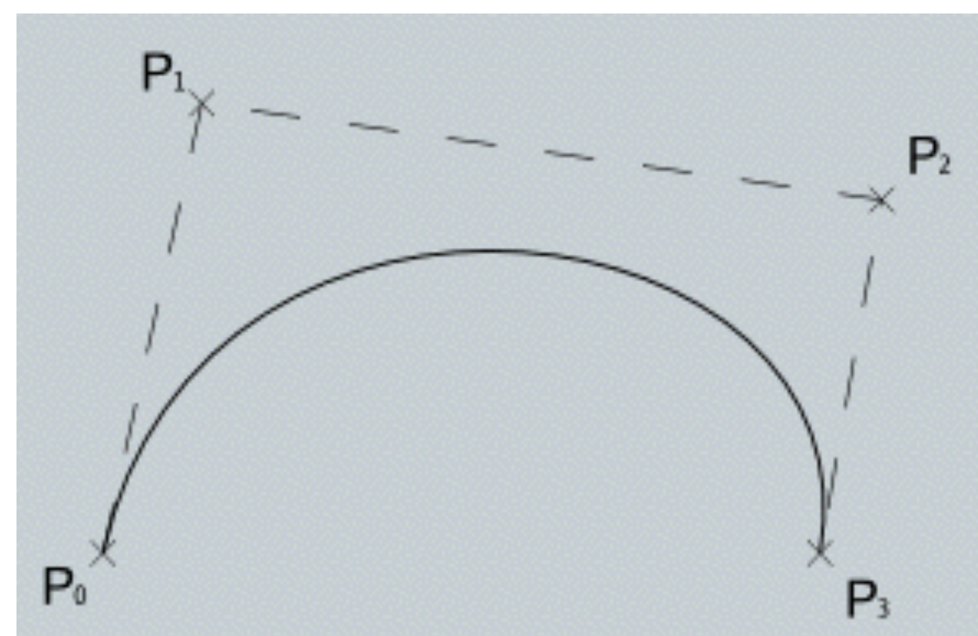
Bézier curve

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

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

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

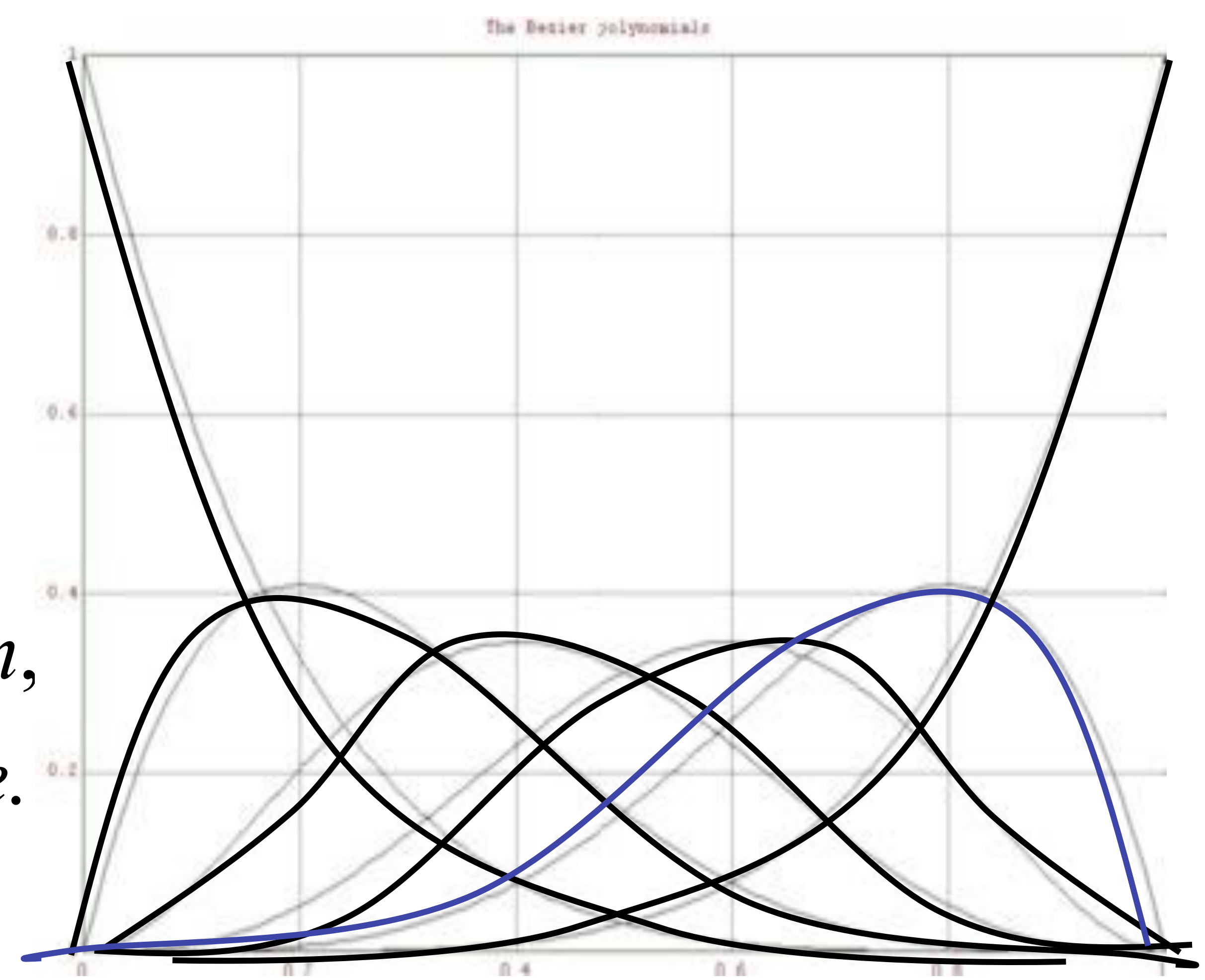
$$C(t) = \begin{pmatrix} X(t) \\ Y(t) \end{pmatrix}, \quad P_i = \begin{pmatrix} x_i \\ y_i \end{pmatrix}$$



Bézier curve

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

1. $B_{i,n}(t) \geq 0, \ i = 0,1,\ldots,n, \ t \in [0,1].$
2. $$\sum_{i=0}^{n} B_{i,n}(t) = 1, \ t \in [0,1].$$
3. $$B_{i,n}(t) = B_{n-i,n}(1-t),$$ $$i = 0,1,\ldots,n, \ t \in [0,1].$$
4. $$B_{i,n}(0) = \begin{cases} 1, & i = 0, \\ 0, & else; \end{cases} \quad B_{i,n}(1) = \begin{cases} 1, & i = n, \\ 0, & else. \end{cases}$$



Bézier curve

Properties of Bernstein **basis**

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

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

7. $$(1-t)B_{i,n}(t) = (1-\frac{i}{n+1})B_{i,n+1}(t);$$

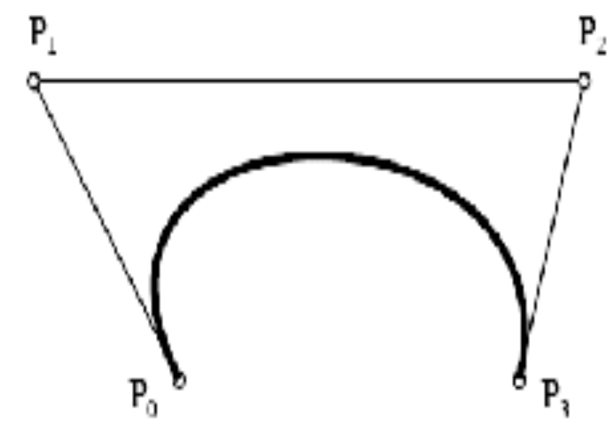
$$tB_{i,n}(t) = \frac{i+1}{n+1}B_{i+1,n+1}(t);$$

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

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

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

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

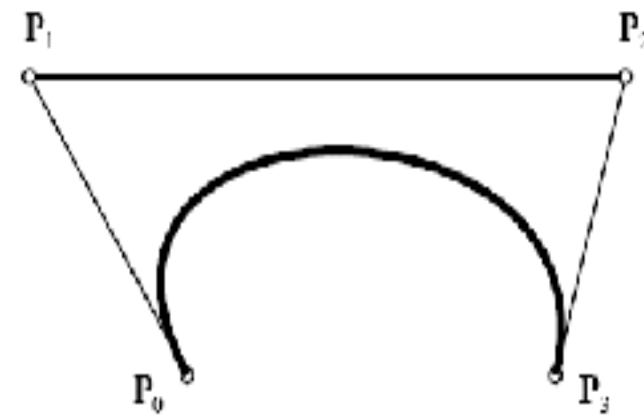
$$C'(t)=n\sum_{i=0}^{n-1}(P_{i+1}-P_i)B_{i,n-1}(t),\ t\in[0,1];\ \ C'(0)=n(P_1-P_0),C'(1)=n(P_n-P_{n-1}).$$

3. **Symmetry:** Let two Bezier curves be generated by ordered Bezier (control) points labelled by {p0,p1,...,pn} and {pn, pn-1,..., 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.

Bézier curve

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 {p0, p1,..., pn} compute the curve and then apply an affine map to it;

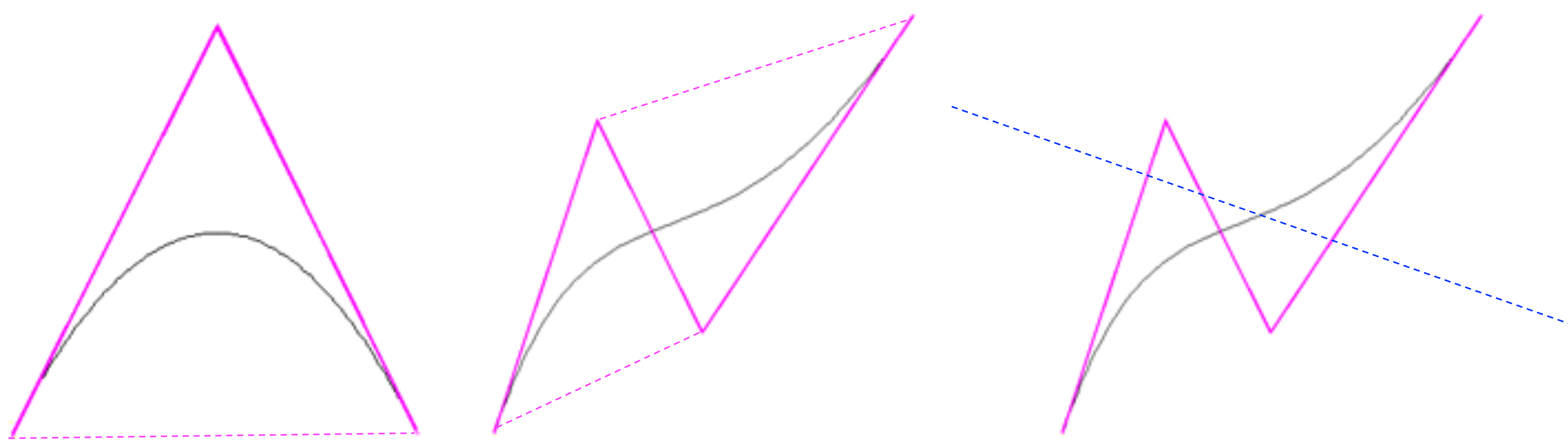
(2) first apply an affine map to the control points {p0, p1,..., pn} to obtain new control points {F(p0),...,F(pn)} and then find the curve with these new control points.

Bézier curve

properties of Bézier curves

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

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



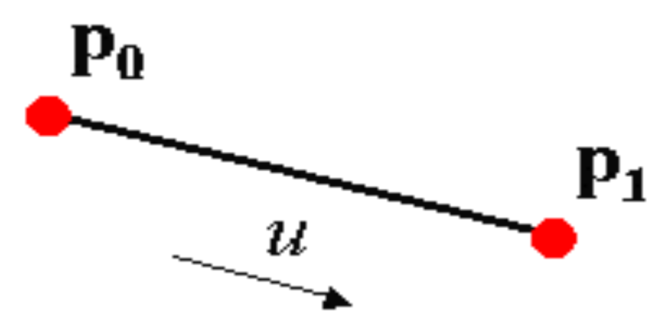
浙江大学计算机学院
数字媒体与网络技术

Bézier curve

Bézier curves

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

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



2. quadratic

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

Degree 2

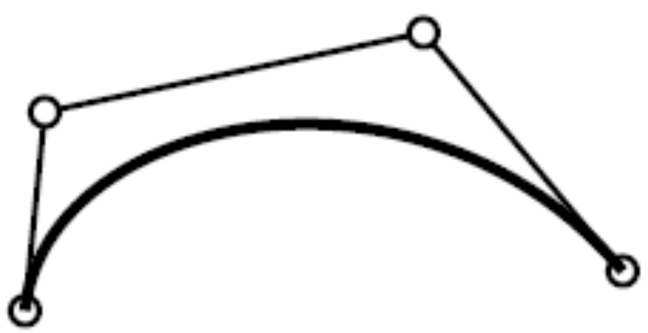
$$C(t) = [t^2 \quad t \quad 1]\begin{bmatrix} 1 & -2 & 1 \\ -2 & 2 & 0 \\ 1 & 0 & 0 \end{bmatrix}\begin{bmatrix} P_0 \\ P_1 \\ P_2 \end{bmatrix}$$

Bézier curve

3. cubic:

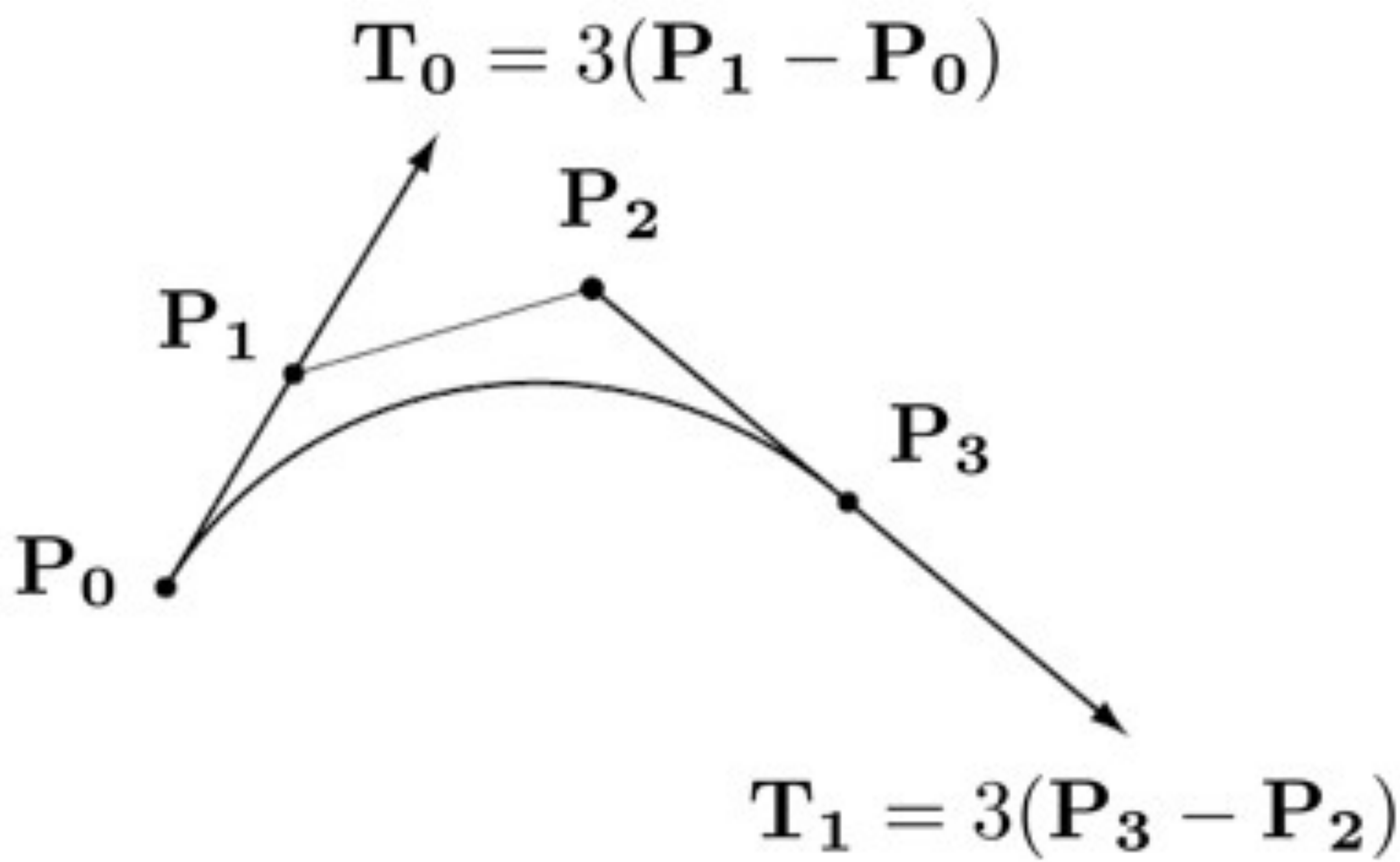
$$\boldsymbol{C}(t) = (1-t)^3 \boldsymbol{P}_0 + 3t(1-t)^2 \boldsymbol{P}_1 + 3t^2(1-t)\boldsymbol{P}_2 + t^3 \boldsymbol{P}_3$$

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



Degree 3

Bezier Curve



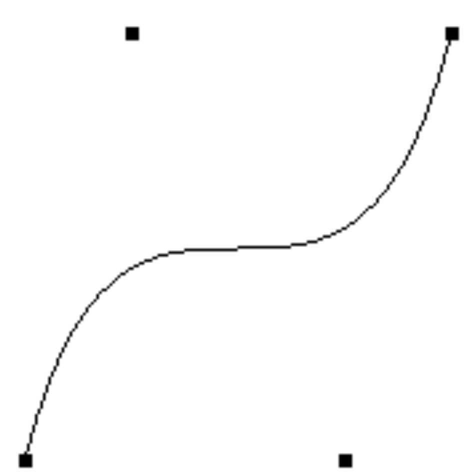
Bezier Curve in OpenGL

- glMap1*(GL_MAP1_VERTEX_3,
  uMin, uMax, stride, nPts, *ctrlPts);
- glEnable/glDisable(GL_MAP1_VERTEX_3);

- glBegin(GL_LINE_STRIP);
  for (...) {
  glEvalCoord1*(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);
   glMap1f(GL_MAP1_VERTEX_3,
0.0, 1.0, 3, 4, &ctrlpoints[0][0]);
   glEnable(GL_MAP1_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++)
         glEvalCoord1f((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();
}
```

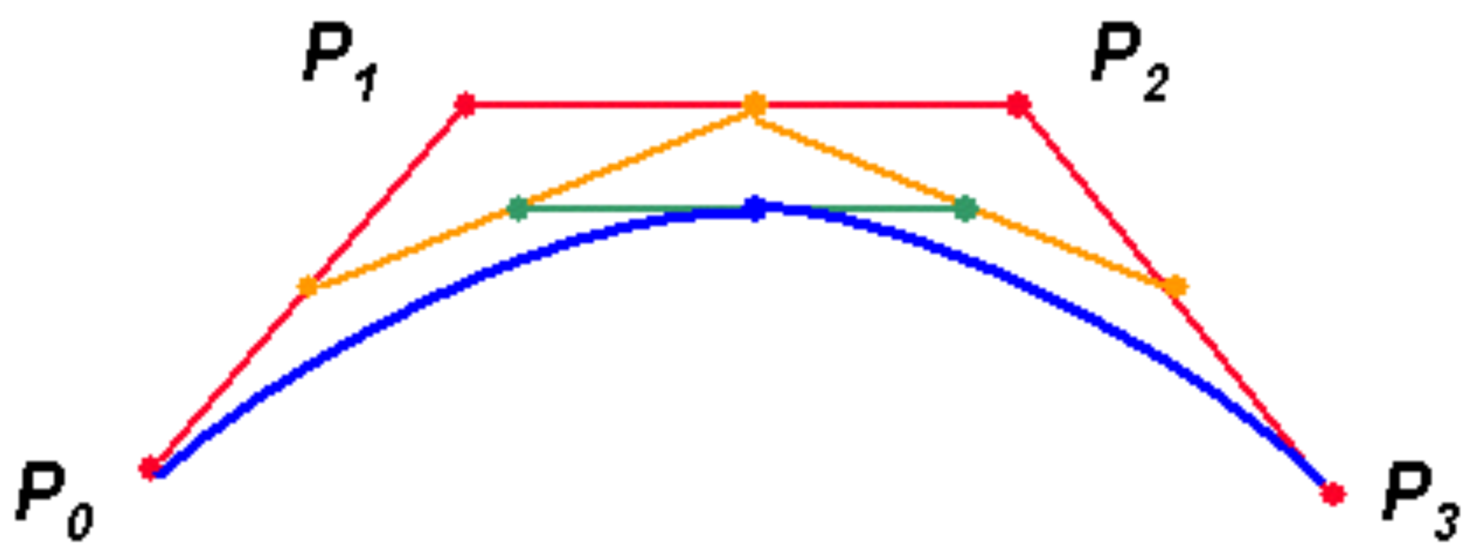
Bézier curve

de Casteljau algorithm

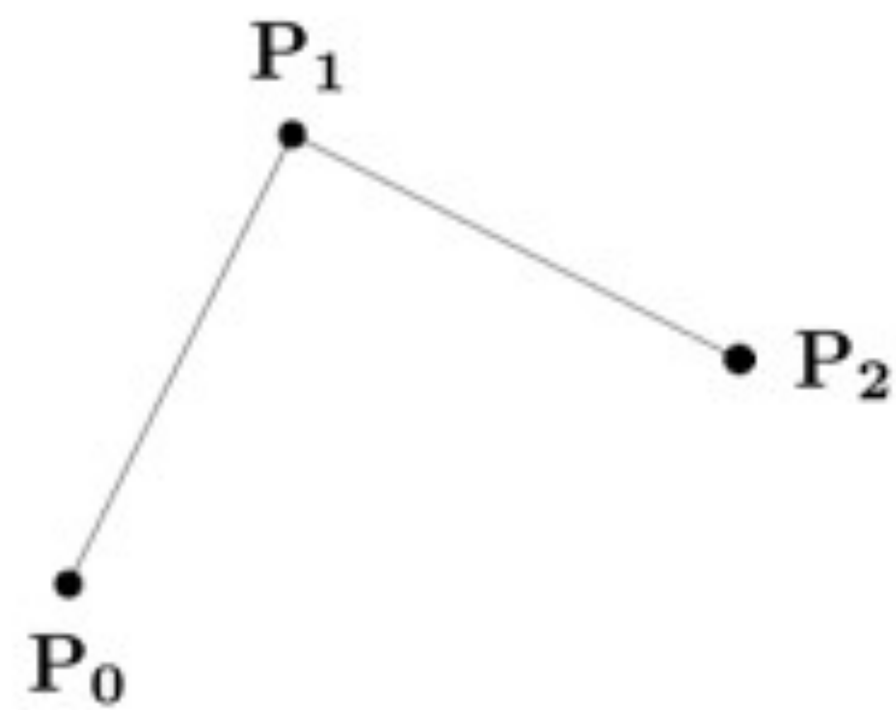
given the control points $\boldsymbol{P}_0, \boldsymbol{P}_1, \ldots, \boldsymbol{P}_n$, and t of Bézier curve, let:

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

then $\boldsymbol{P}_0^n\left(t\right)$ =C(t).

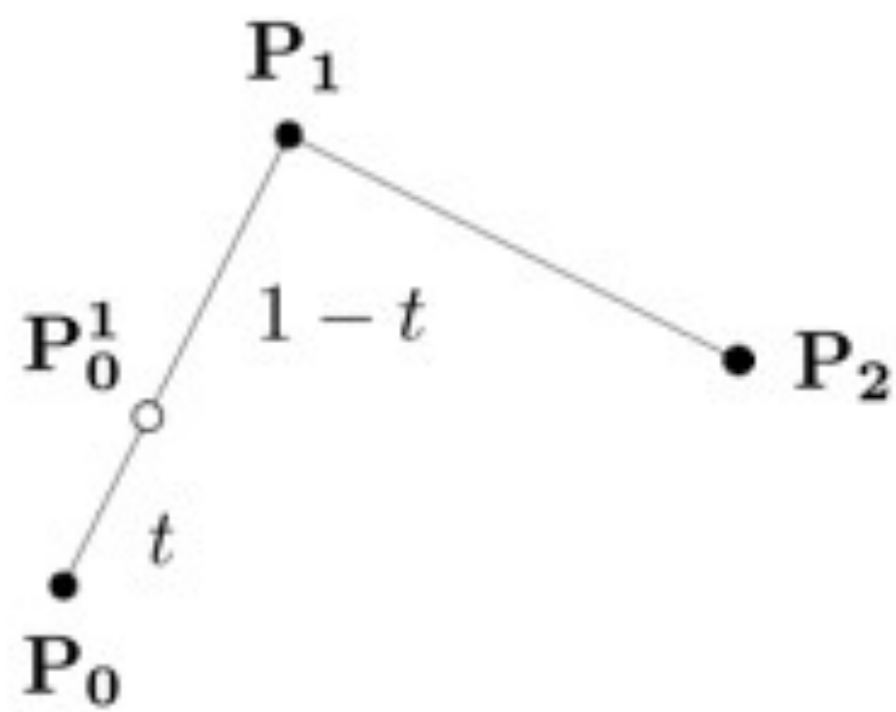


Consider Three Points



Insert Point Using Linear Interpolation

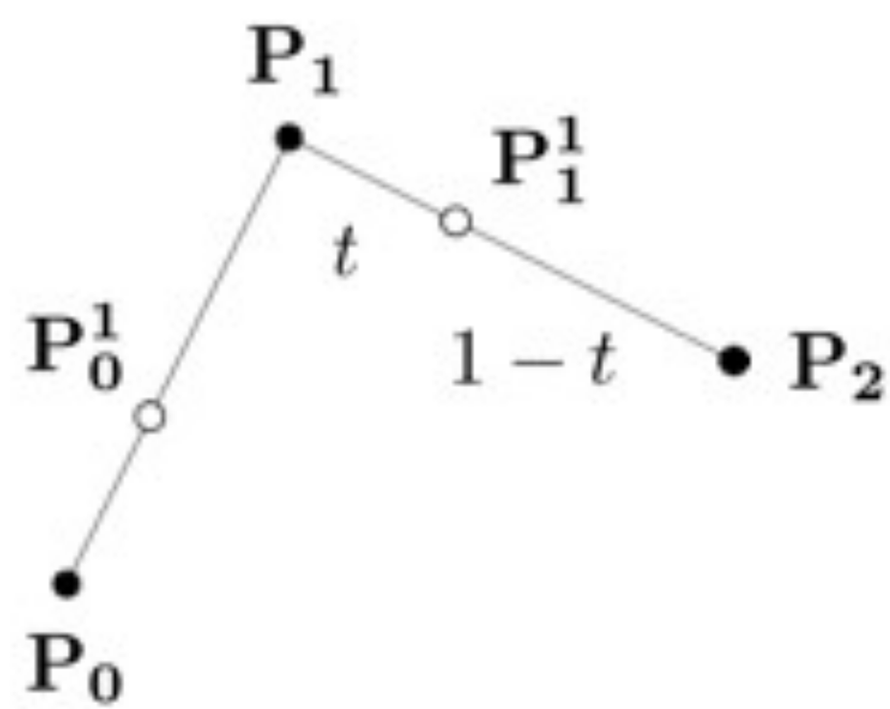
$$\mathbf{P}_0^1 = (1-t)\mathbf{P}_0 + t\mathbf{P}_1$$



Insert Points on Both Edges

$$\mathbf{P_0^1} = (1-t)\mathbf{P_0} + t\mathbf{P_1}$$

$$\mathbf{P_1^1} = (1-t)\mathbf{P_1} + t\mathbf{P_2}$$

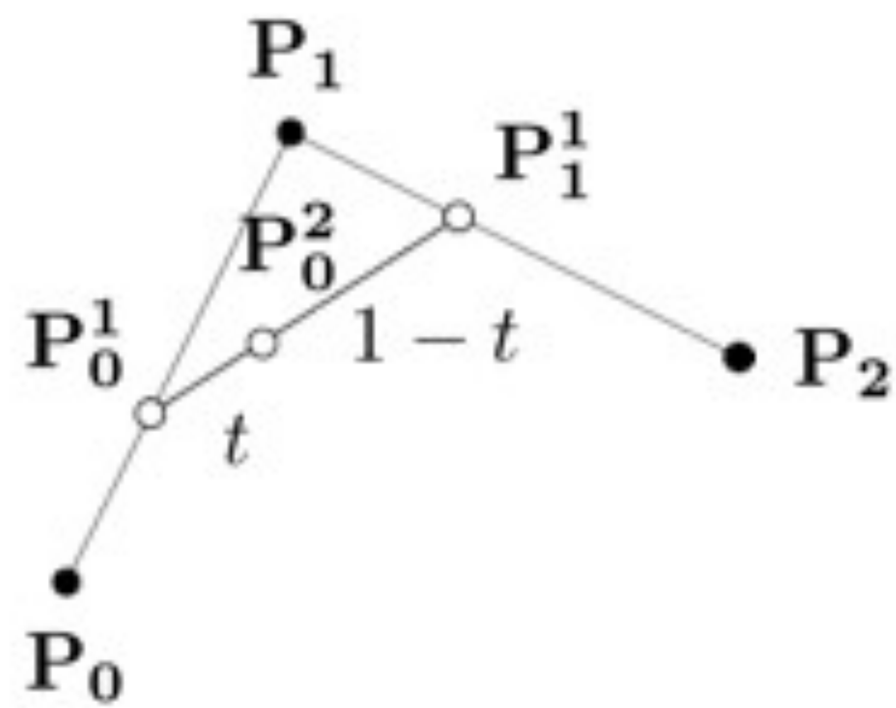


Repeat Recursively

$$\mathbf{P}_0^1 = (1-t)\mathbf{P_0} + t\mathbf{P_1}$$

$$\mathbf{P}_1^1 = (1-t)\mathbf{P_1} + t\mathbf{P_2}$$

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

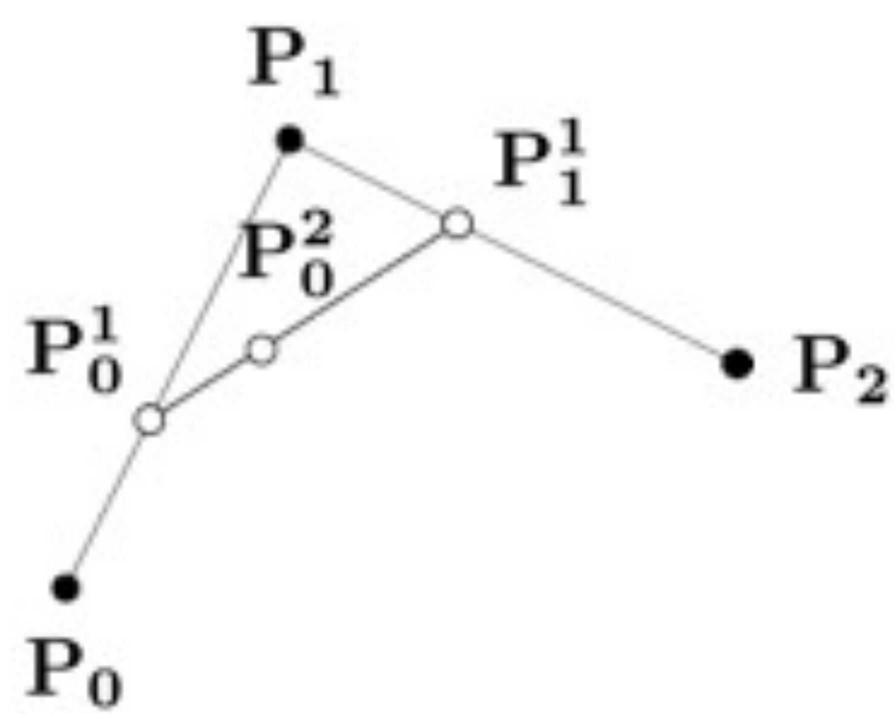


Algorithm Defines Curve

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

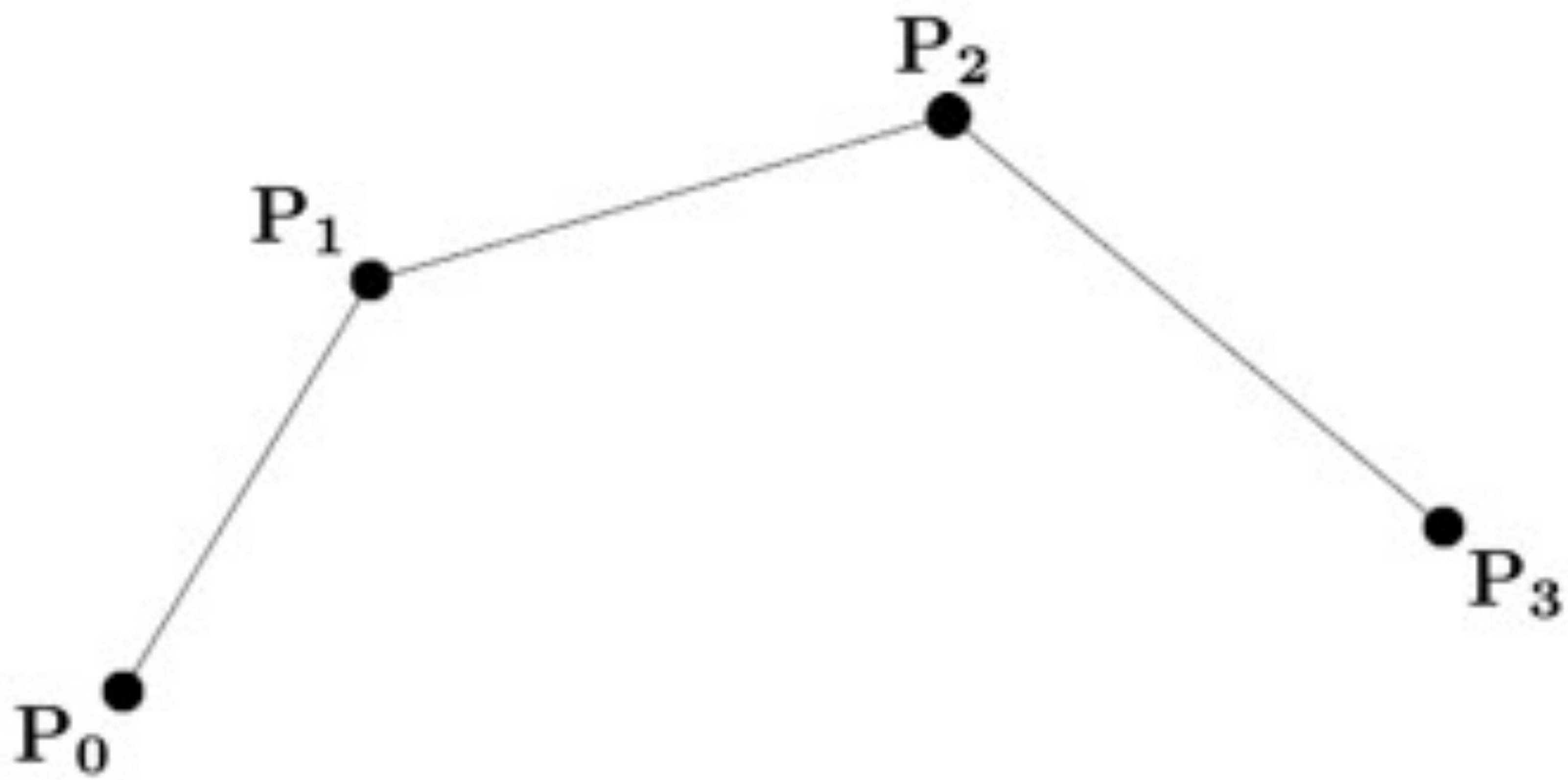
$$\mathbf{P}_1^1 = (1-t)\mathbf{P}_1 + t\mathbf{P}_2$$

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

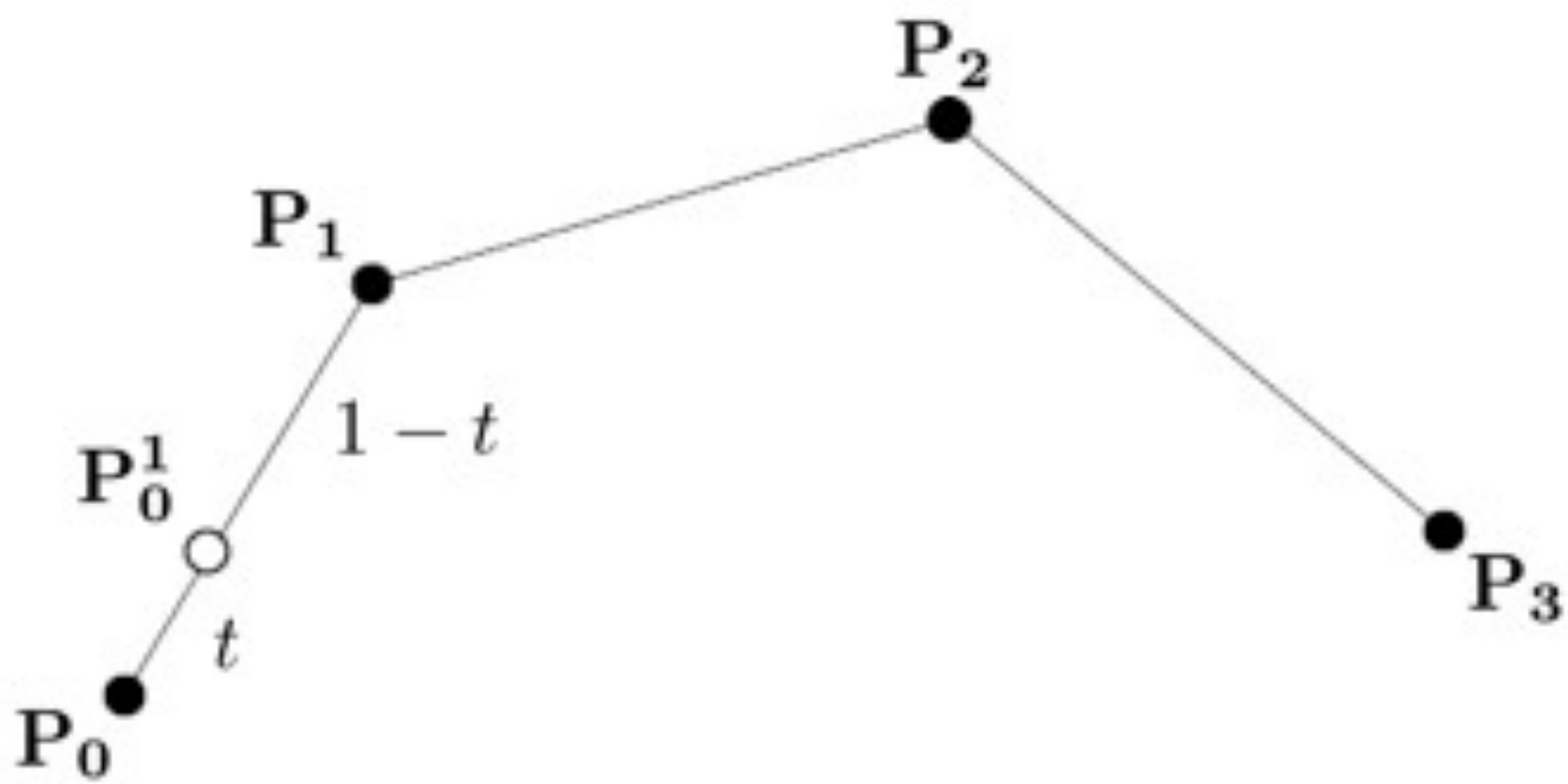


Resulting point $\mathrm{P}(t) = \mathrm{P}_0^2$

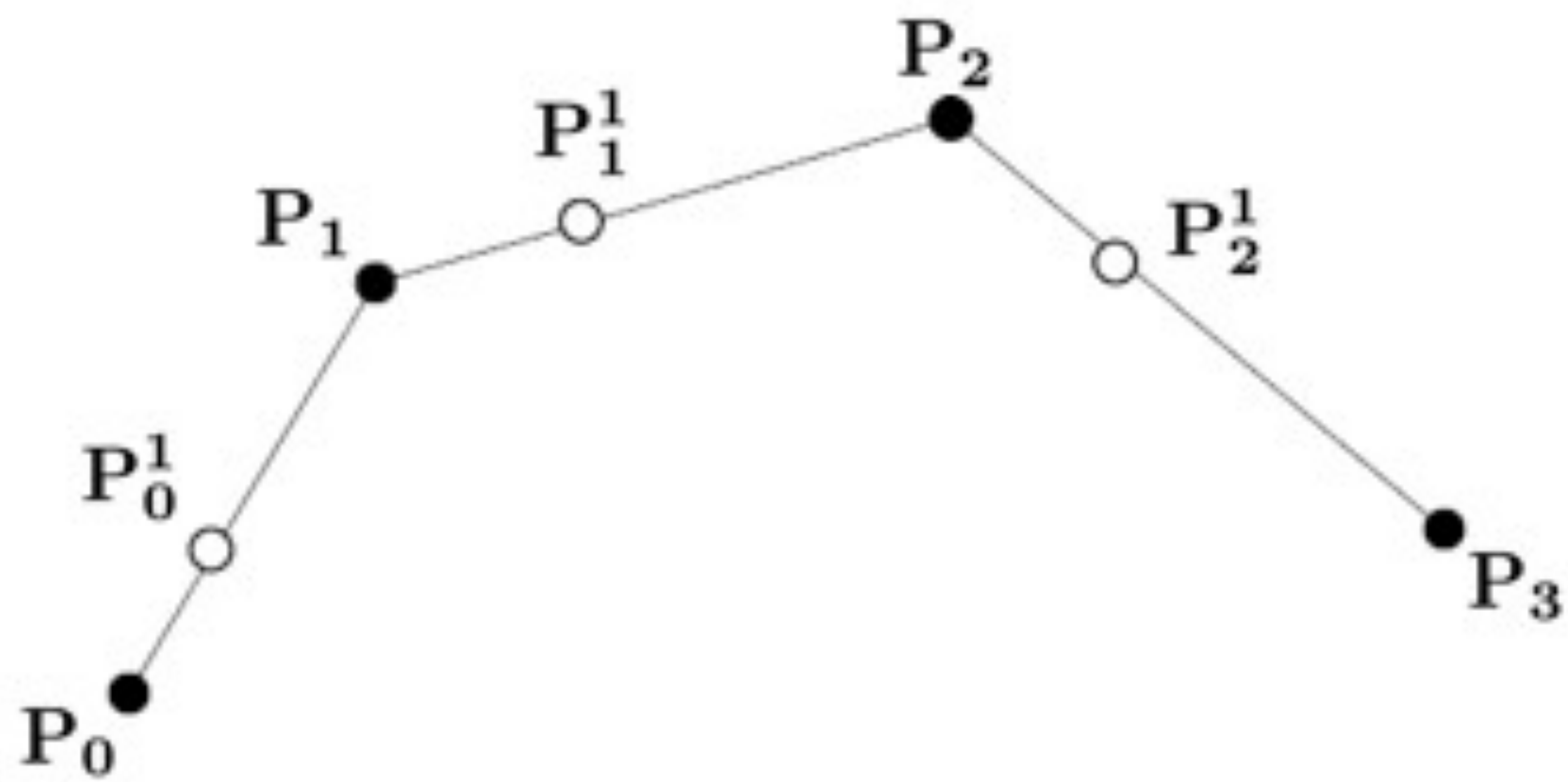
Consider Four Points



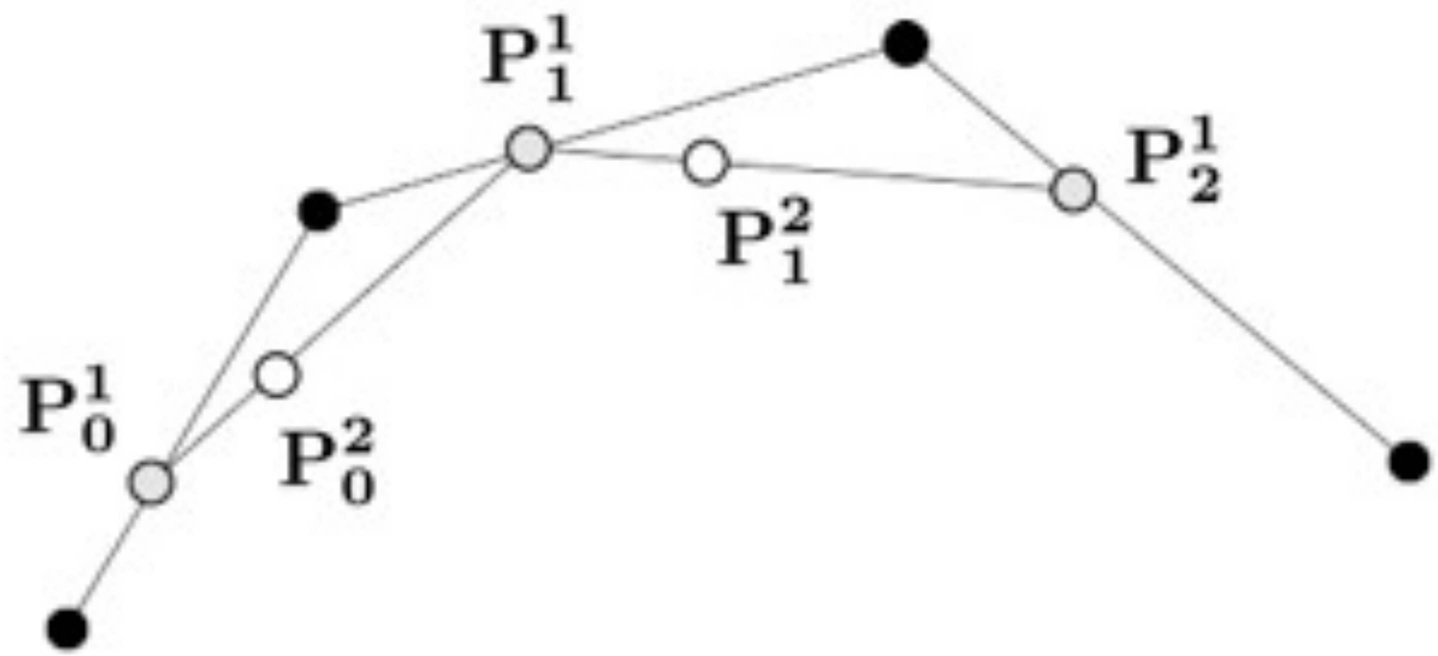
Linear Interpolation



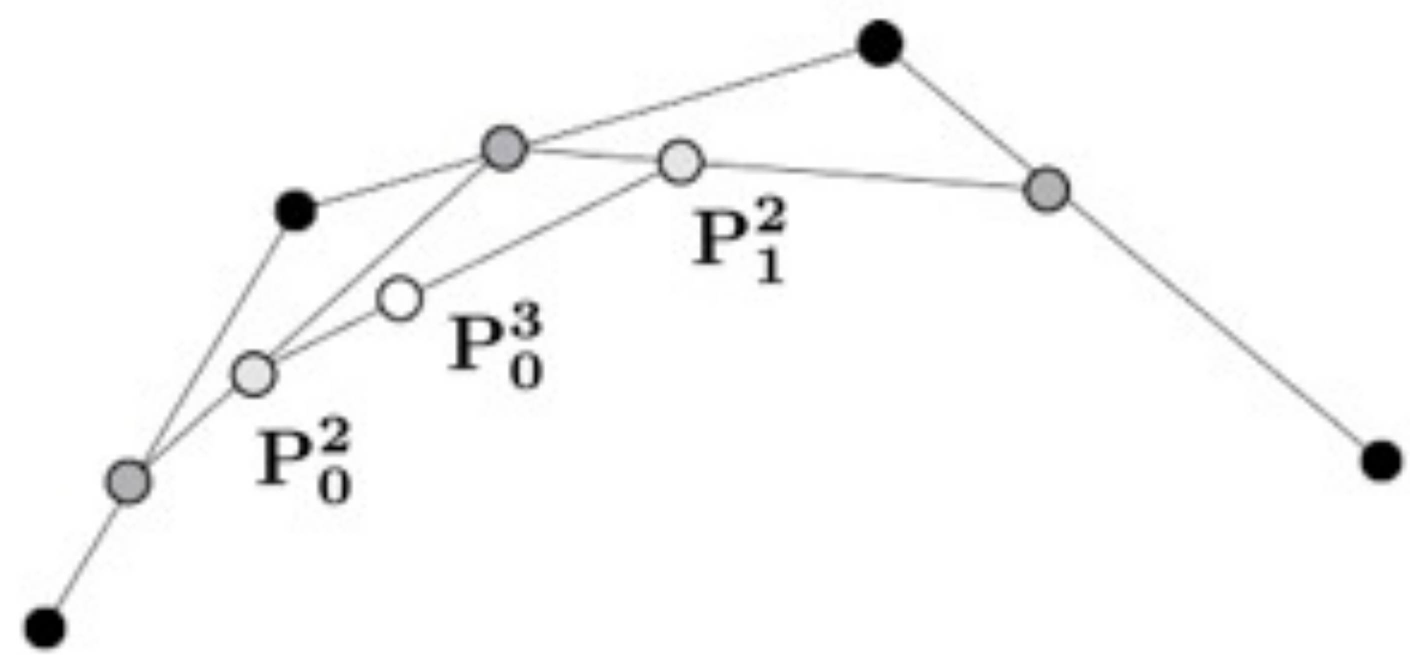
On All Edge Segments



Repeat Recursively

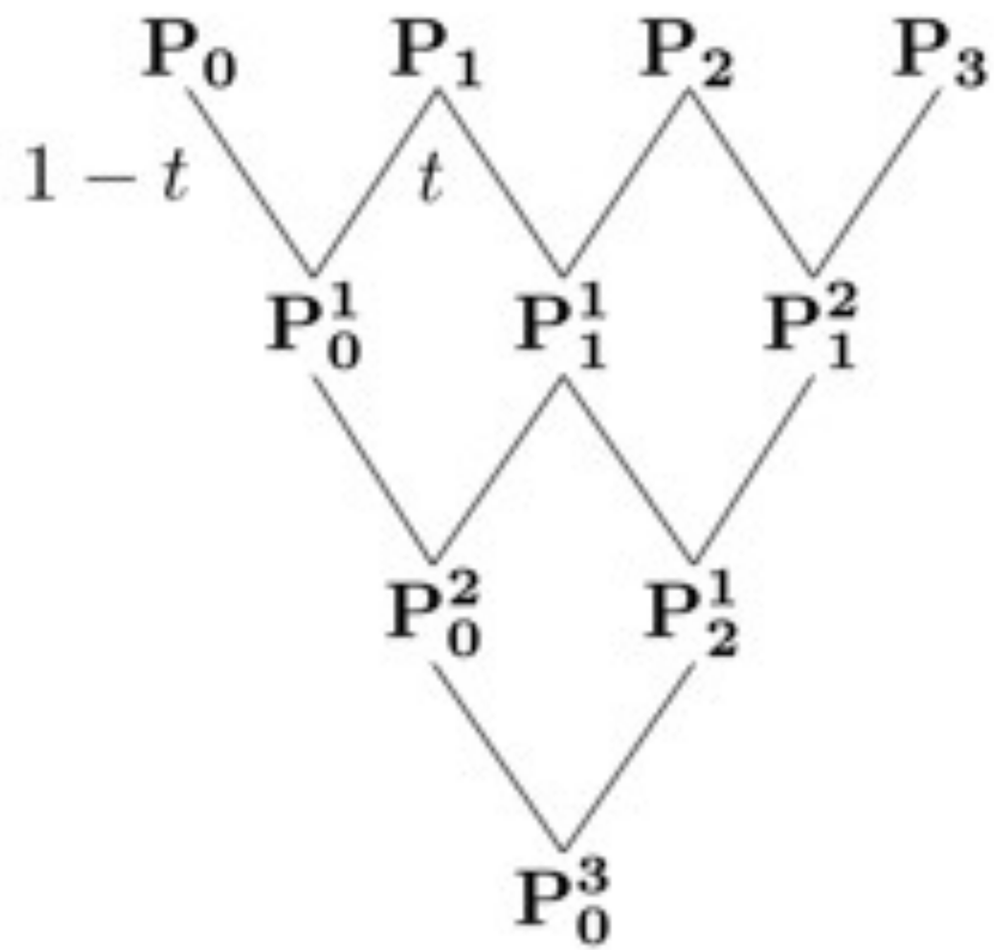


Algorithm Defines Curve



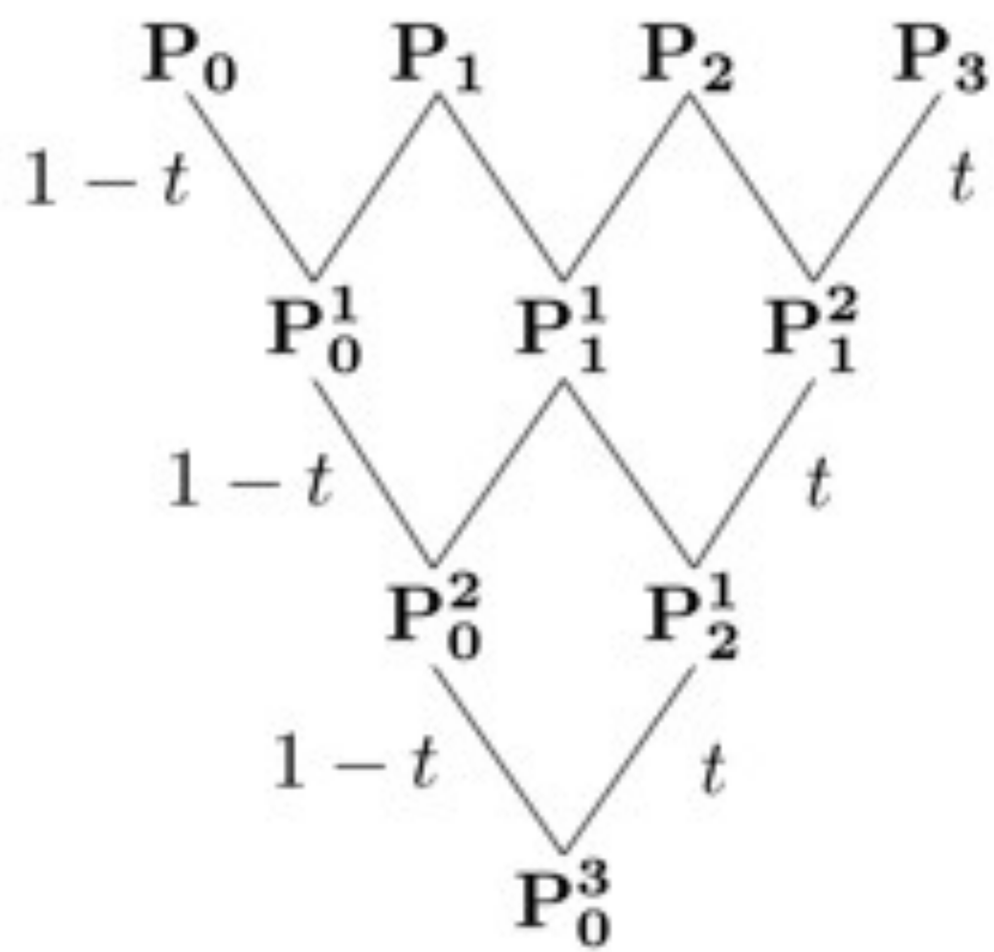
$$\mathbf{P}(t) = \mathbf{P}_0^3$$

Pyramid Algorithm



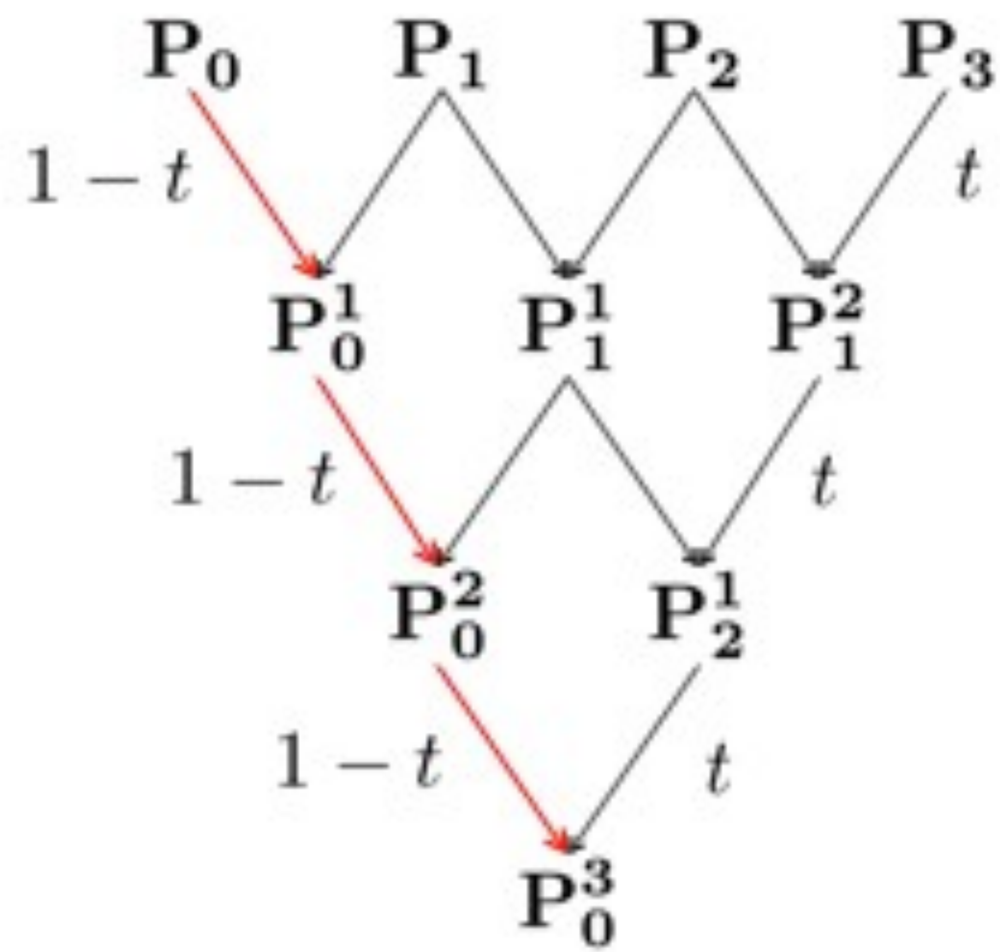
$$\mathbf{P}(t) = \sum_{i=0}^{3} \mathbf{P}_i B_i(t)$$

Pyramid Algorithm



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

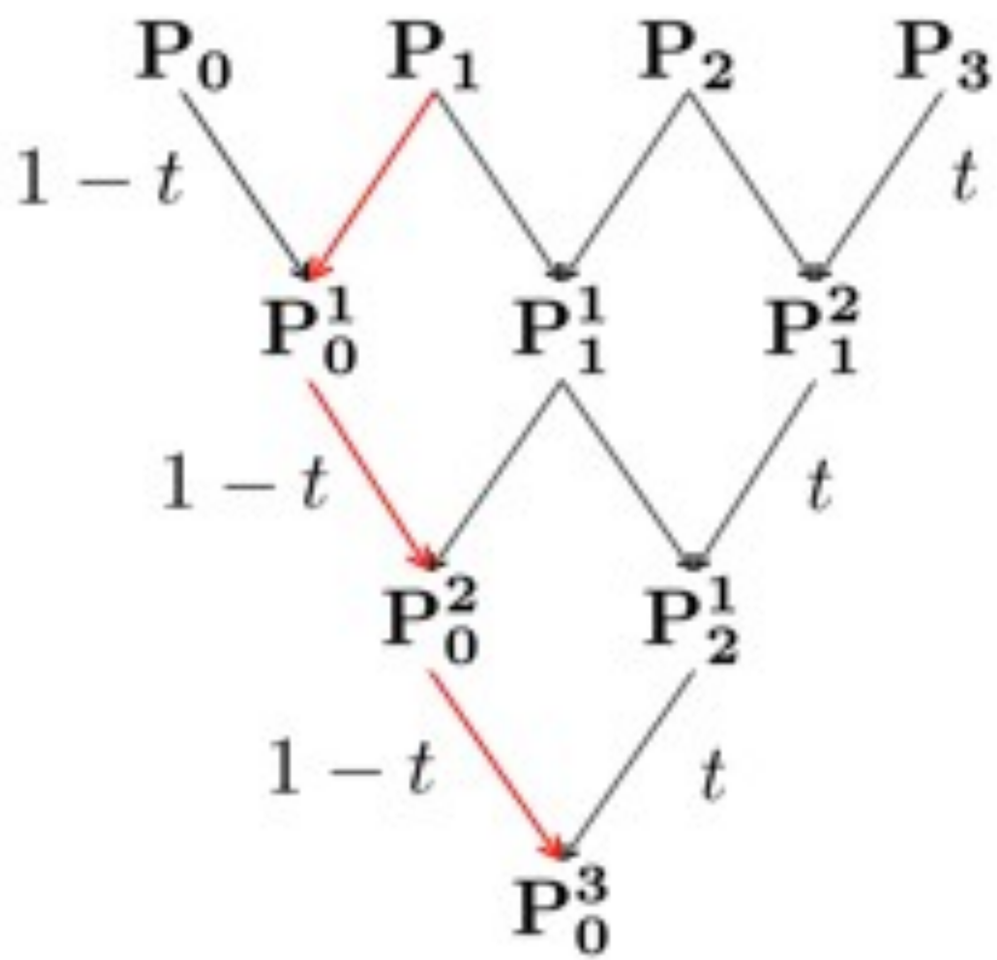
Pyramid Algorithm



Path

$(1-t)^3\mathbf{P}_0$

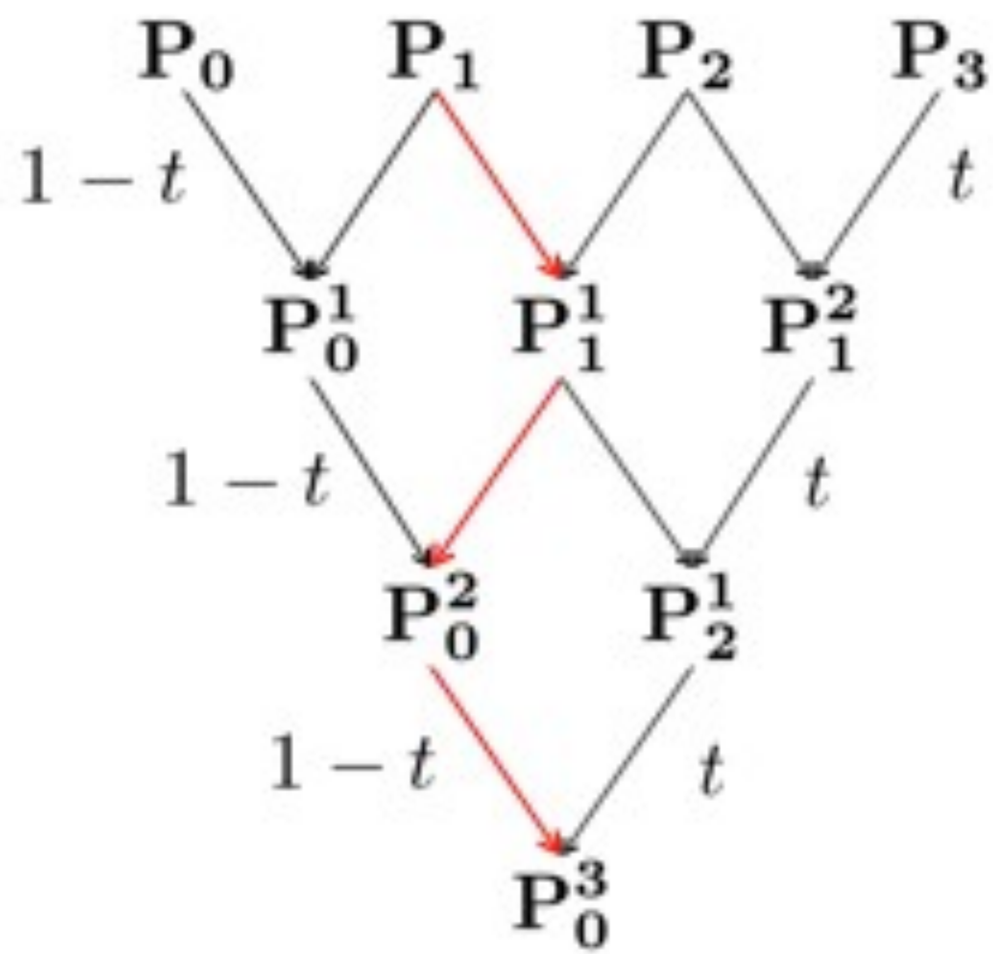
Pyramid Algorithm



Path

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

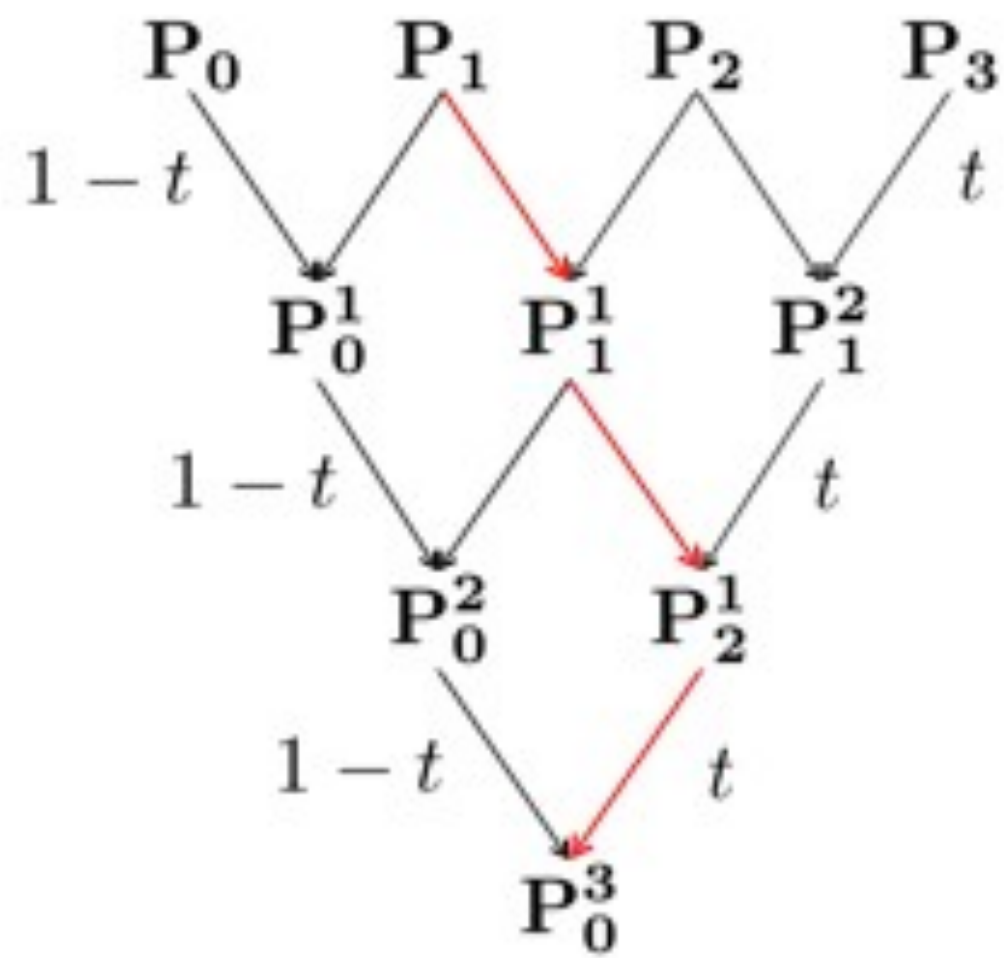
Pyramid Algorithm



Path

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

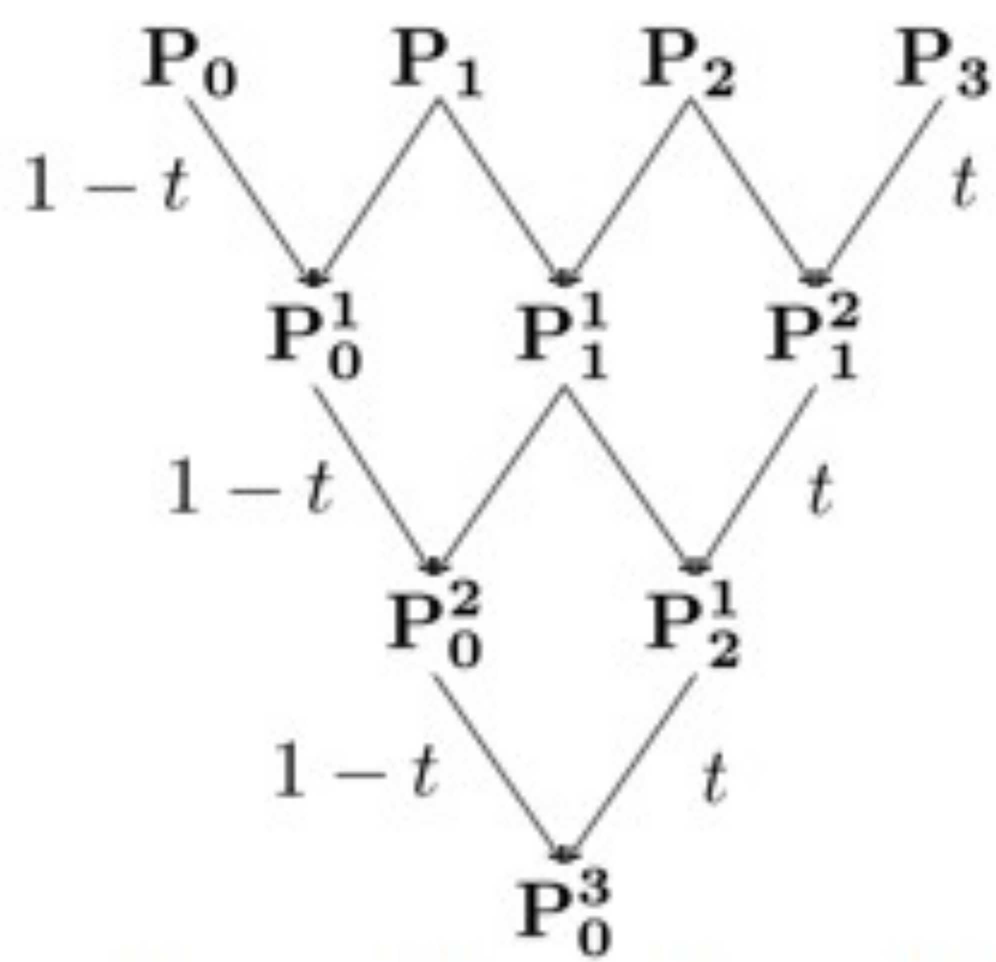
Pyramid Algorithm



Three paths total

$$3t(1-t)^2\mathbf{P}_0^1$$

Leads to a Cubic Polynomial Curve



$$P(t) = \sum_i^n P_i\, B_i^n(t)$$

$$B_0^3(t) = (1-t)^3$$

$$B_1^3(t) = 3t(1-t)^2$$

$$B_2^3(t) = 3t^2(1-t)$$

$$B_3^3(t) = t^3$$

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

Bézier curve

Rational Bézier Curve

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

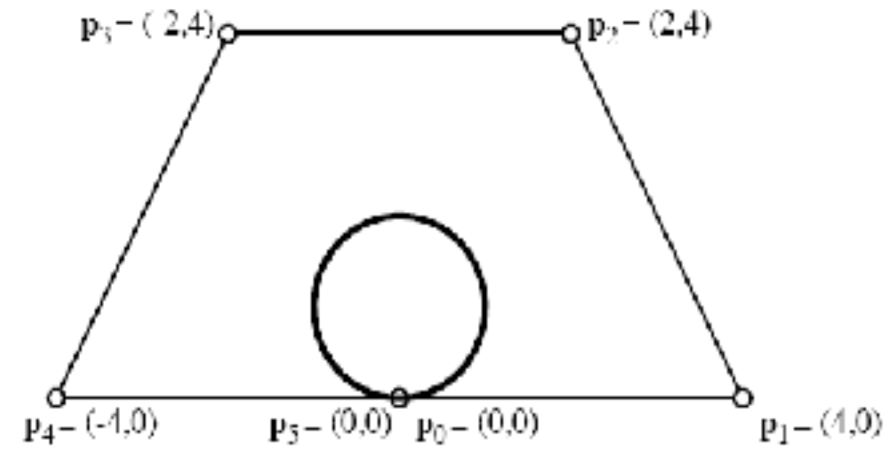


Figure 2.19: Circle as Degree 5 Rational Bézier Curve.

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: $\boldsymbol{R}(0) = \boldsymbol{P}_0$; $\boldsymbol{R}(1) = \boldsymbol{P}_n$

2. tangent of endpoints:

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

3. **Convex Hull Property**

......

5.

6. Influence of the weights

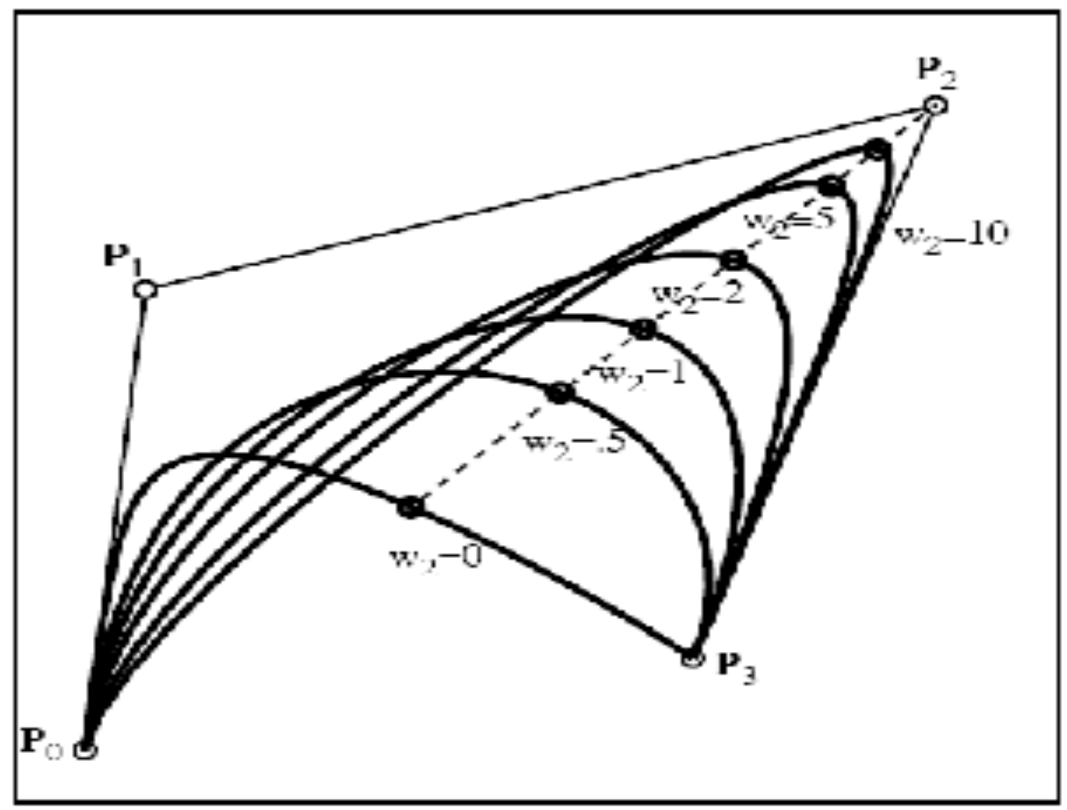


Figure 2.16: Rational Bézier curve.

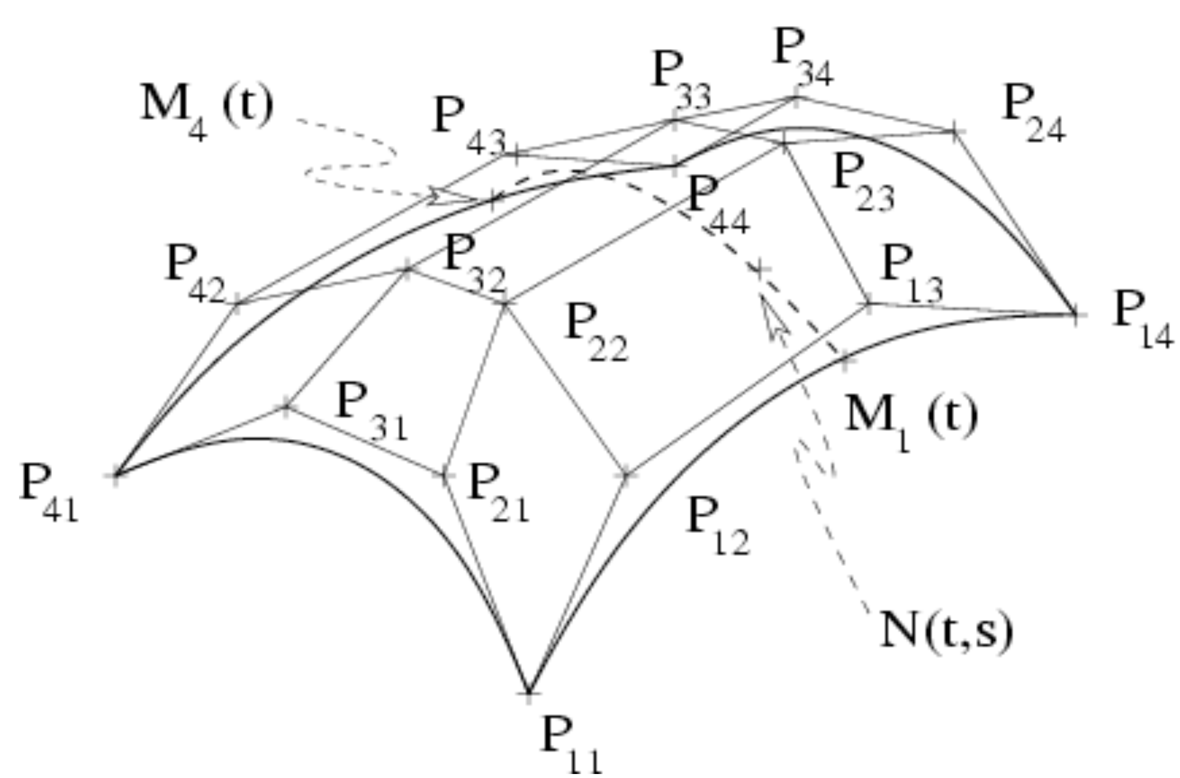
Bézier surface

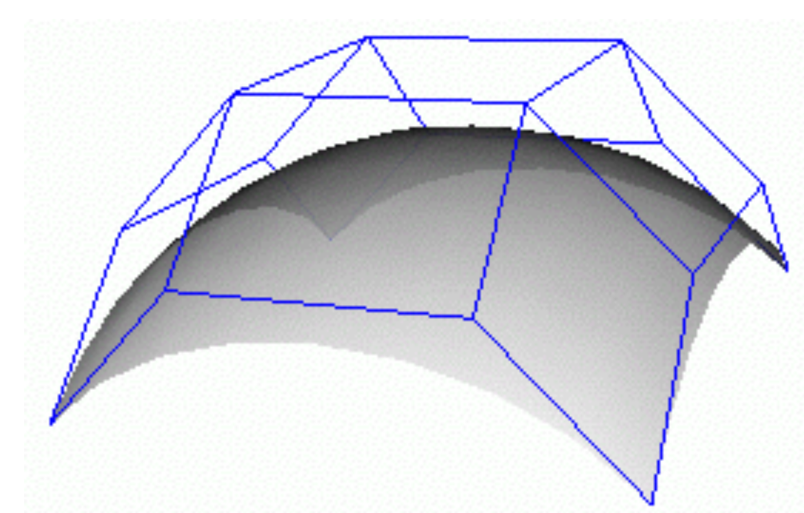
Bézier surface

Bézier surface:

$$\boldsymbol{S}(u,v)=\sum_{i=0}^{n}\sum_{j=0}^{m}\boldsymbol{P}_{ij}B_{i,n}(u)B_{j,m}(v),\qquad 0\le u,v\le 1$$

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





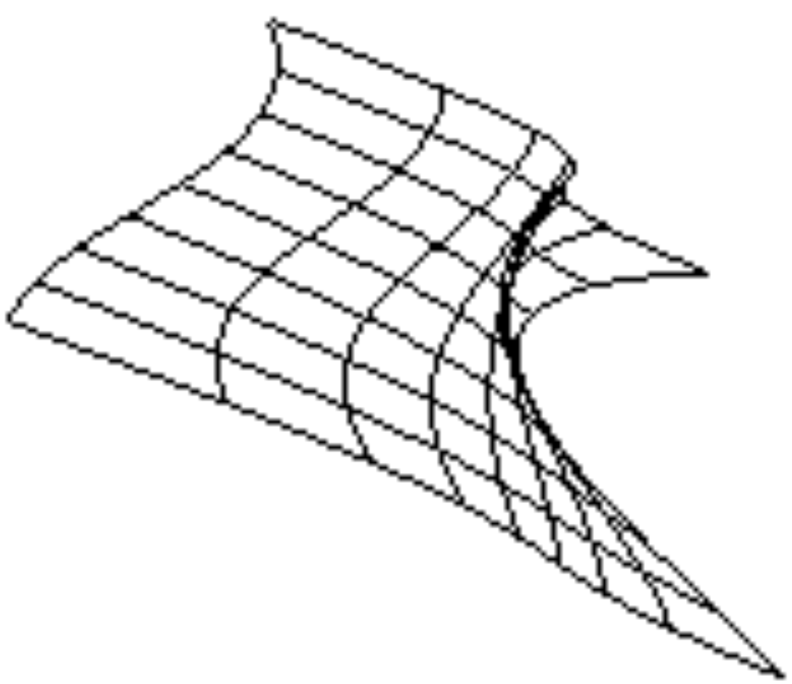
Bezier Surface in OpenGL

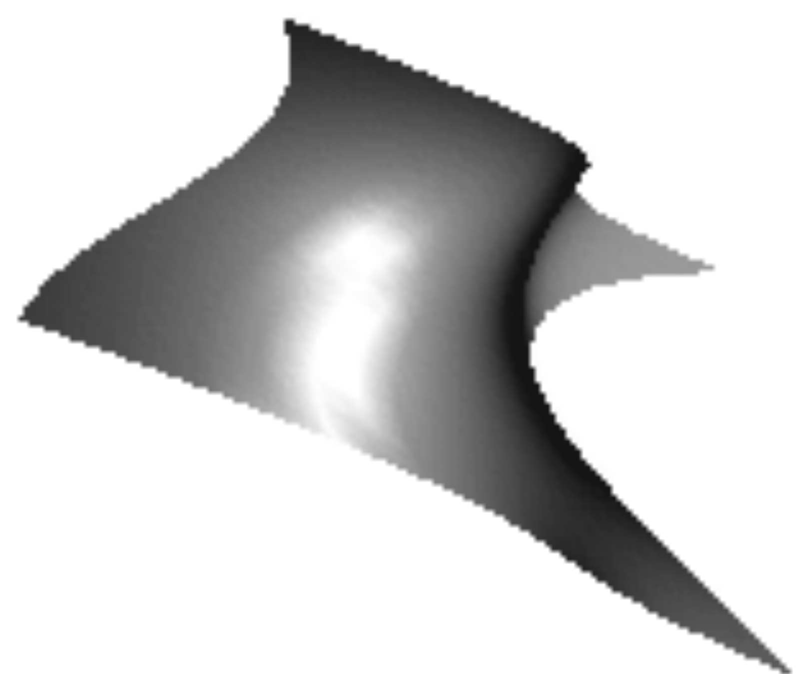
```
- glMap2*(GL_MAP2_VERTEX_3,
          uMin, uMax, uStride, nuPts,
          vMin, vMax, vStride, nvPts,*ctrlPts);
- glEnable/glDisable(GL_MAP2_VERTEX_3);

- 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();
```

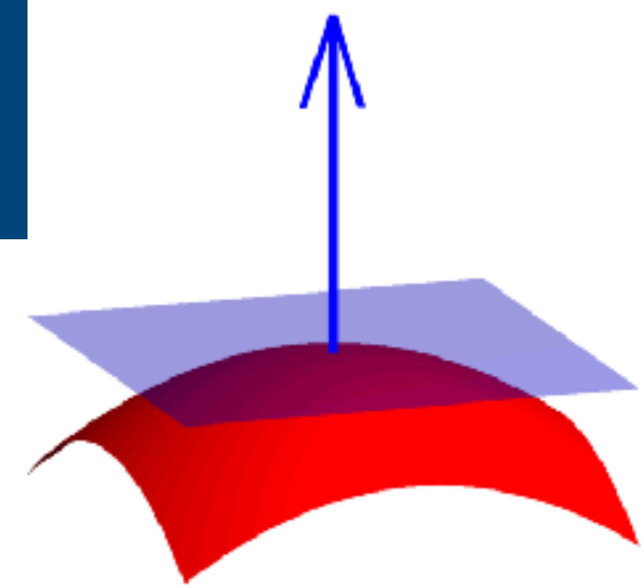
```
GLfloat ctrlpoints[4][4][3] = {
   {{-1.5, -1.5, 4.0}, {-0.5, -1.5, 2.0},
    {0.5, -1.5, -1.0}, {1.5, -1.5, 2.0}},
   {{-1.5, -0.5, 1.0}, {-0.5, -0.5, 3.0},
    {0.5, -0.5, 0.0}, {1.5, -0.5, -1.0}},
   {{-1.5, 0.5, 4.0}, {-0.5, 0.5, 0.0},
    {0.5, 0.5, 3.0}, {1.5, 0.5, 4.0}},
   {{-1.5, 1.5, -2.0},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



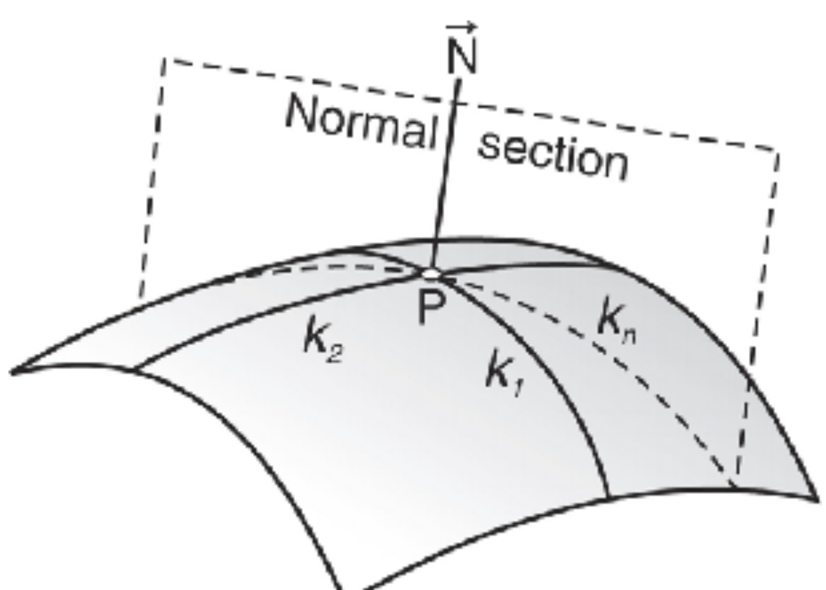
partial derivation of Bézier surface $\boldsymbol{S}(u,v)$:

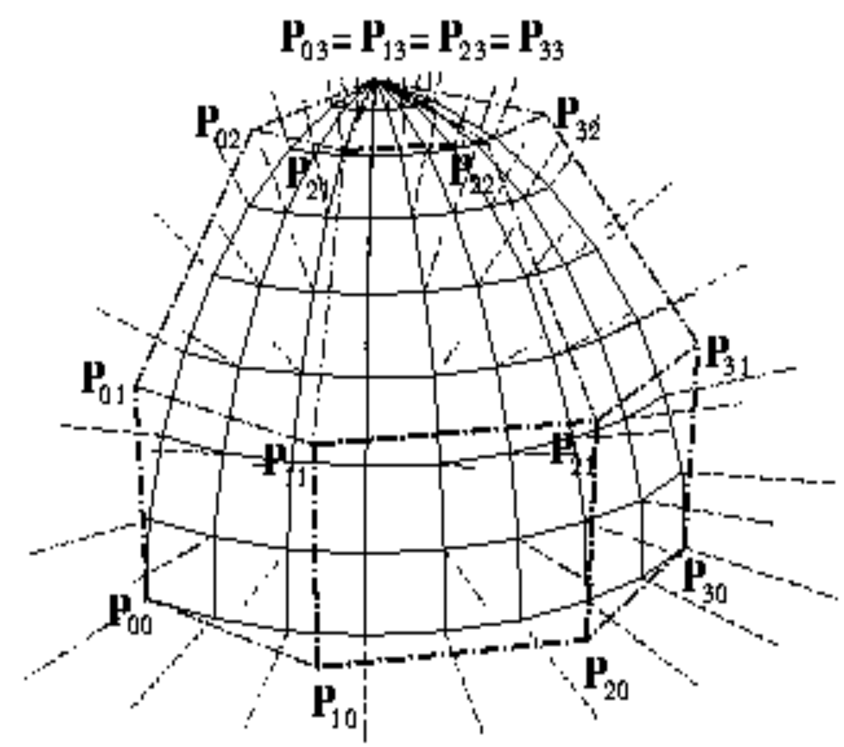
$$\frac{\partial}{\partial u}\boldsymbol{S}(u,v)=\frac{\partial}{\partial u}\sum_{i=0}^{n}\sum_{j=0}^{m}\boldsymbol{P}_{ij}B_{i,n}(u)B_{j,m}(v)=n\sum_{i=0}^{n-1}\sum_{j=0}^{m}(\boldsymbol{P}_{i+1,j}-\boldsymbol{P}_{ij})B_{i,n-1}(u)B_{j,m}(v)$$

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

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

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

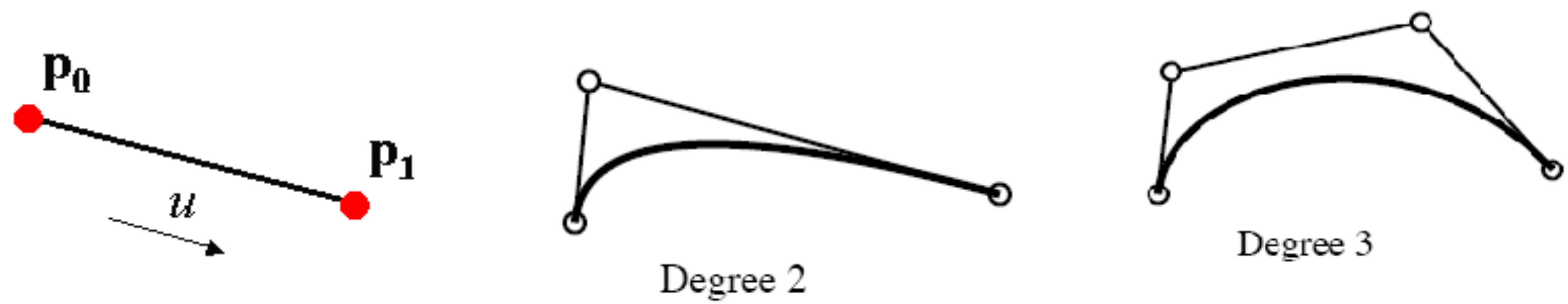




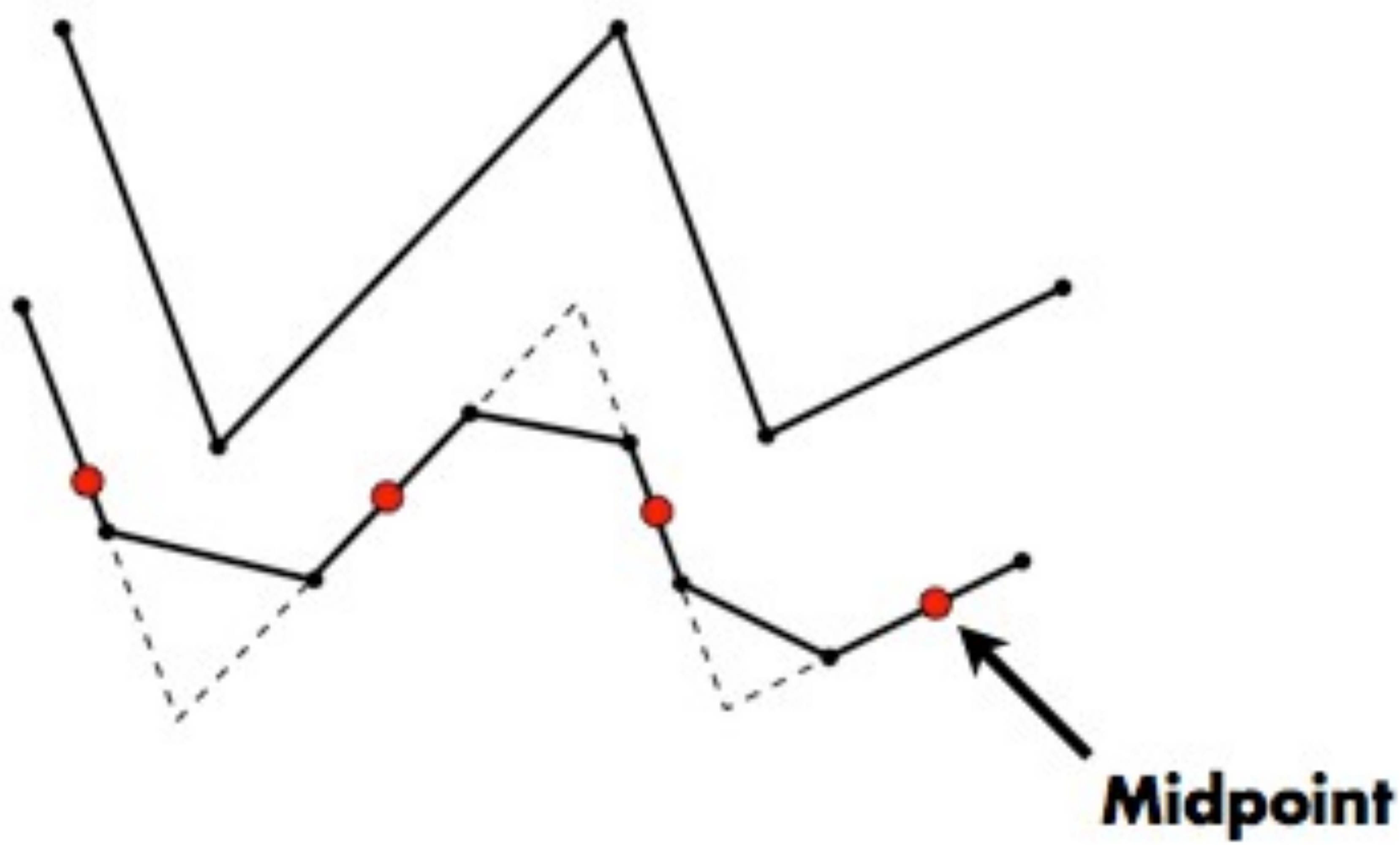
NURBS curve

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.

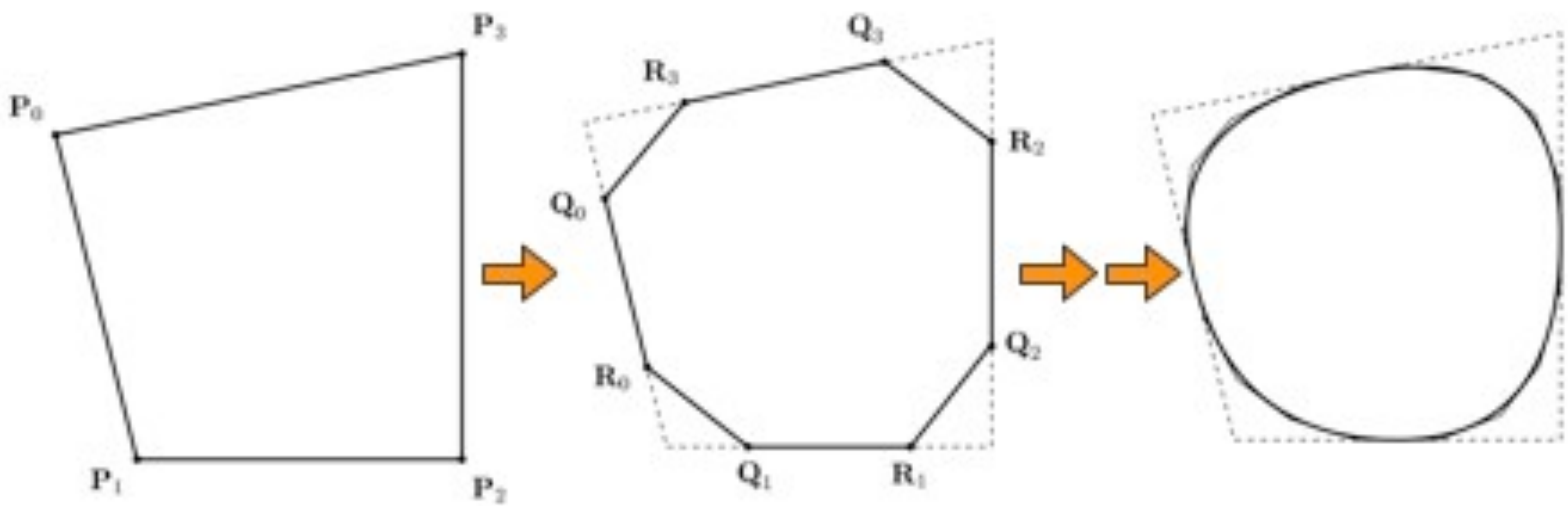


Corner Cutting Algorithm



Chaiken (1974)

Procedural Curve



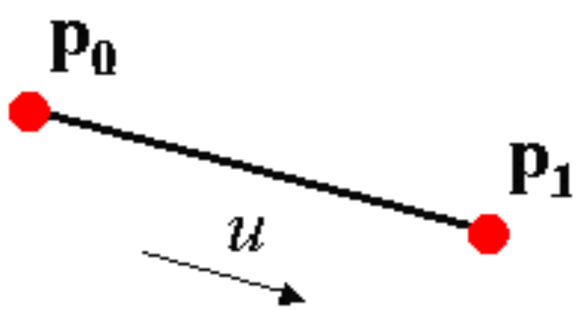
Repeatedly cutting corners generates a limit curve
1. Interpolates midpoints
2. Tangent preserved at midpoints

NURBS curve

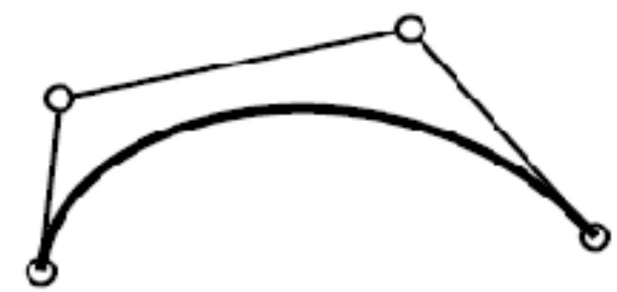
B-spline curve

- 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.



Degree 2



Degree 3

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

NURBS curve

B-spline curve:

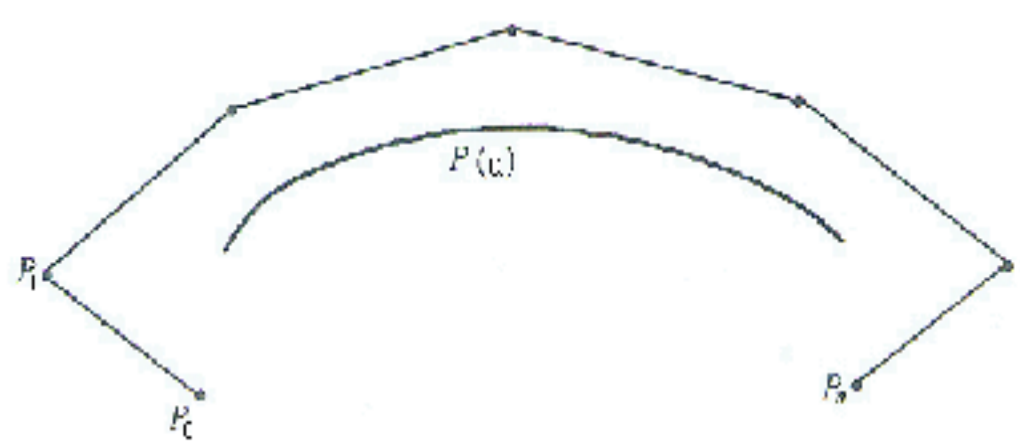
$$C(u) = \sum_{i=0}^{n} P_i N_{i,p}(u) \qquad a \le u \le b$$

Where $\boldsymbol{P}_0, \boldsymbol{P}_1, \ldots, \boldsymbol{P}_n$ are control points, $\mathbf{u}=[u_0=a, u_1, \ldots, u_i, \ldots, u_{n+k+1}=b]$.

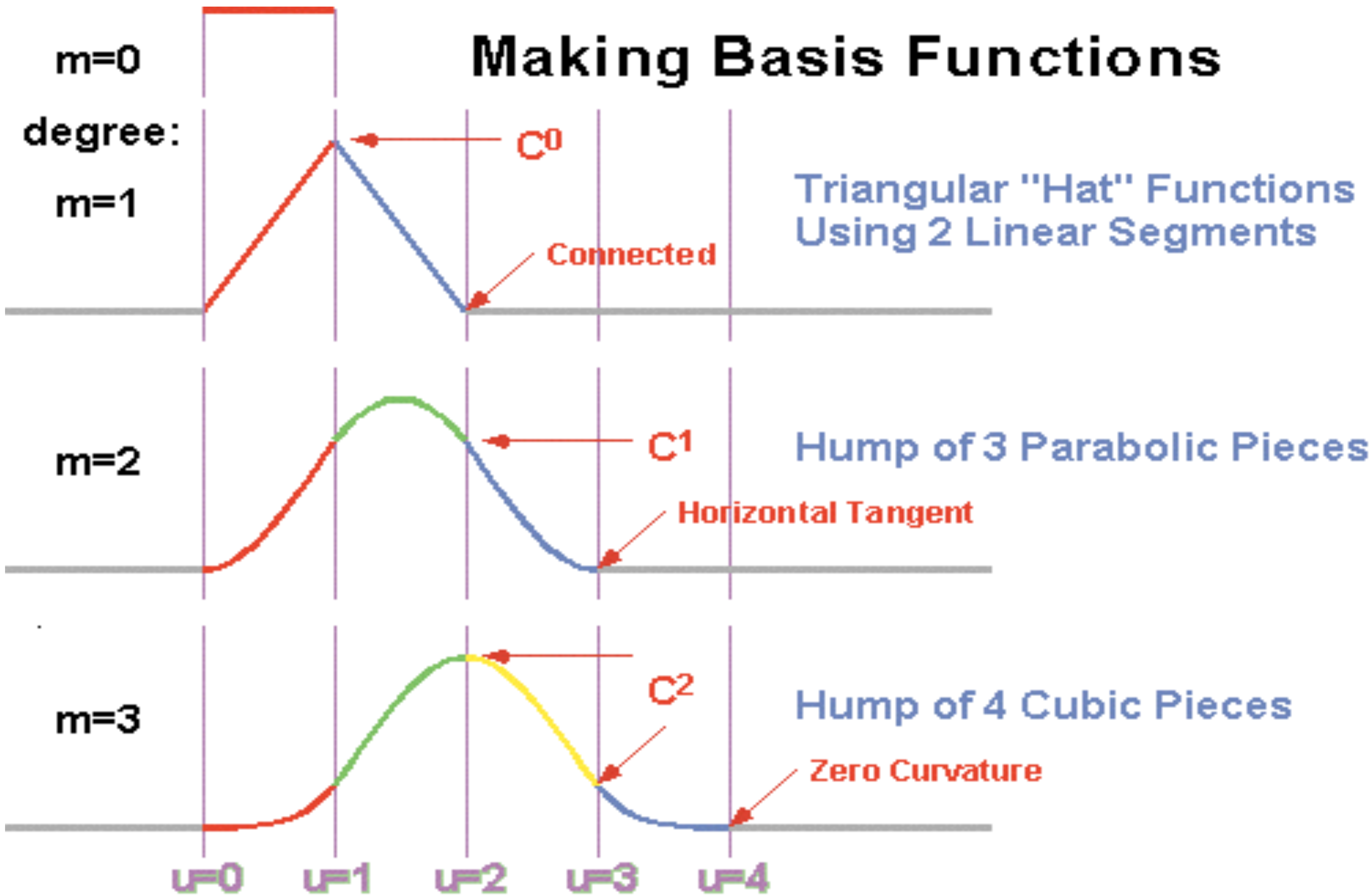
$$N_{i,0}(u) = \begin{cases} 1 & u_i \le u < u_{i+1} \\ 0 & otherwise \end{cases}$$

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

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



B-spline basis



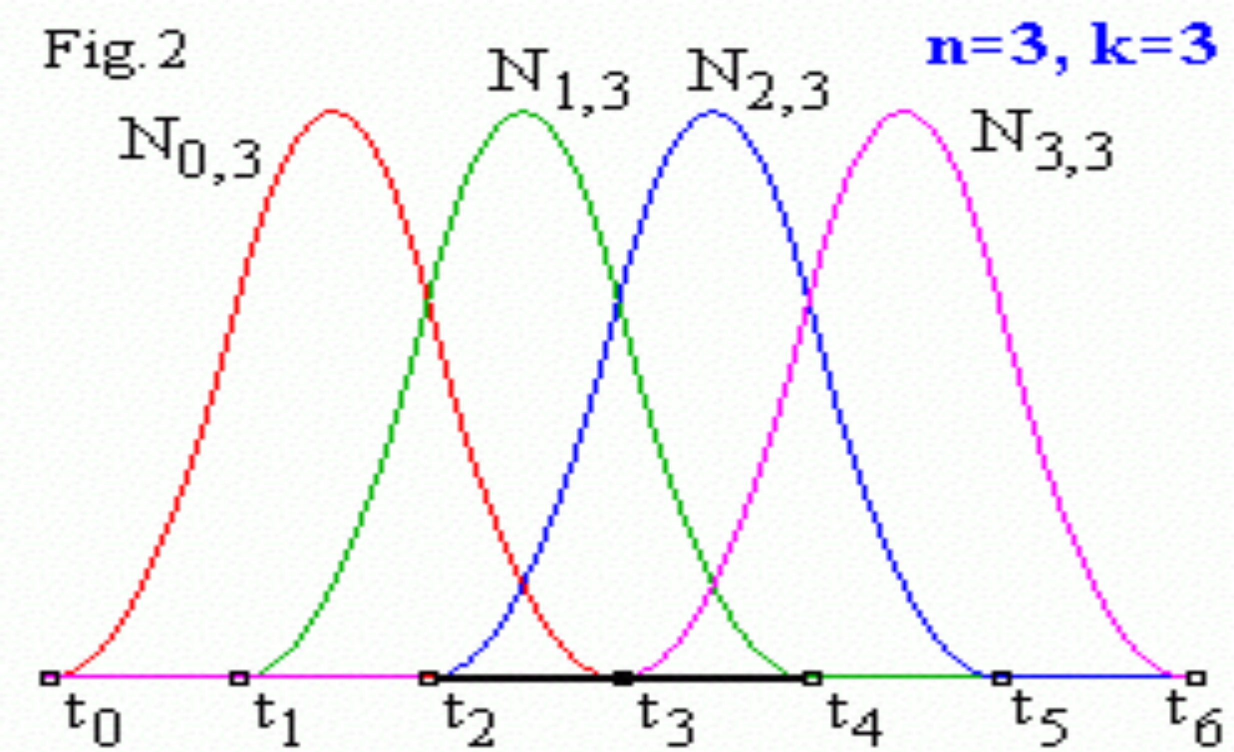
$$U = \{u_i\}_{i=-\infty}^{\infty}$$

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

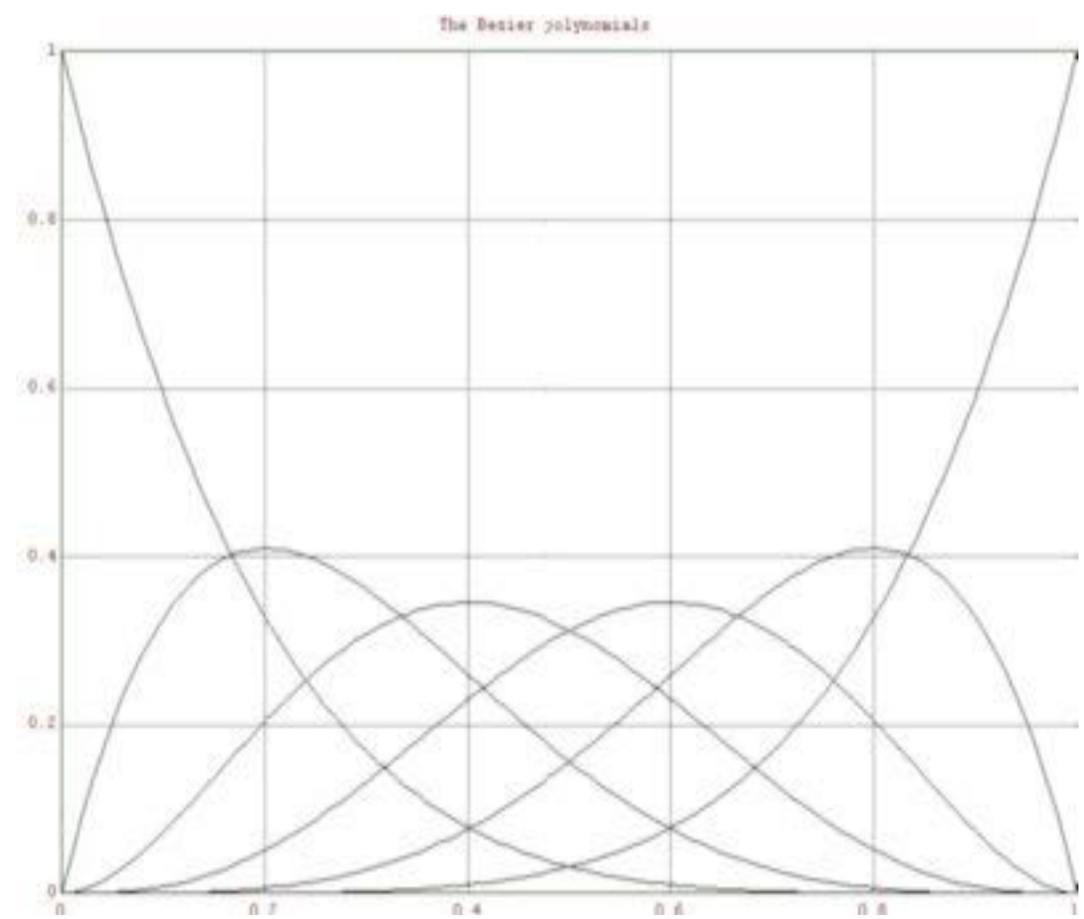
$$N_{i,p}(u) = \frac{u - u_i}{u_{i+p} - u_i} N_{i,p-1}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}(u),$$

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

B-spline basis v.s. Bernstein ~



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



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

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

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

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

NURBS curve

properties of B-spline basis

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

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

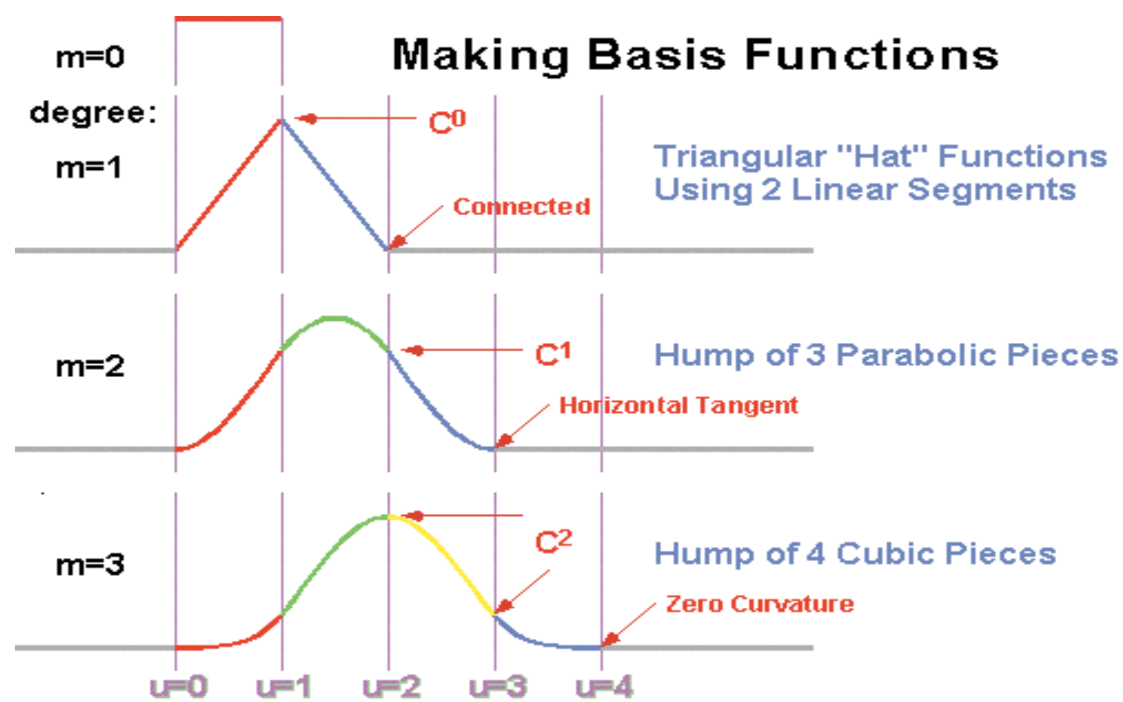
2. normalization:

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

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:

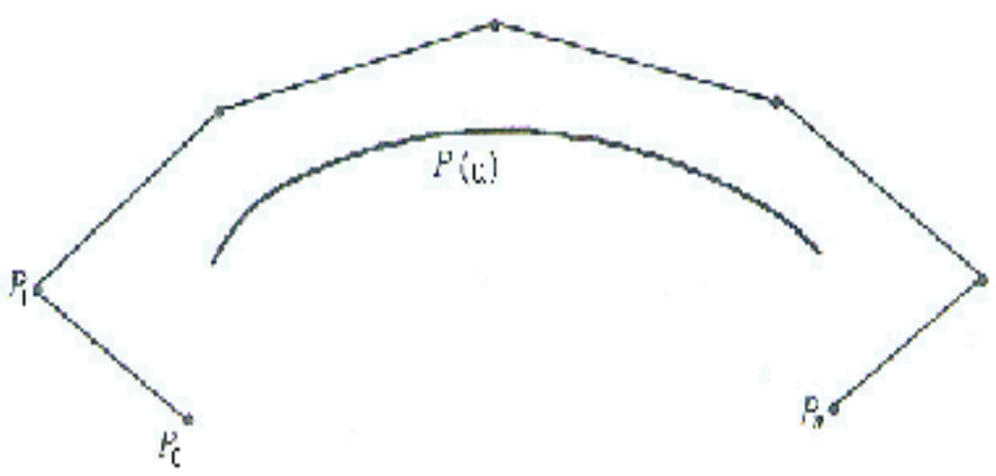
1. **Convex Hull Property**

2. **variation diminishing property.**

3. **Affine Invariance**

4. **local**

5. **piecewise polynomial**



浙江大学计算机学院
数字媒体与网络技术

NURBS curve

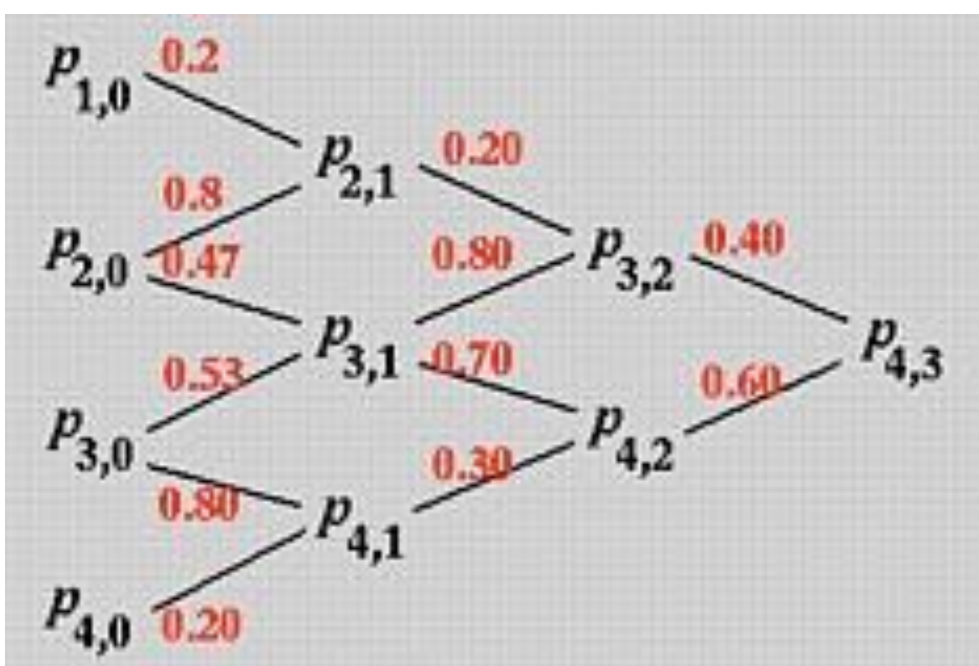
B-spline---de Boor algorithm

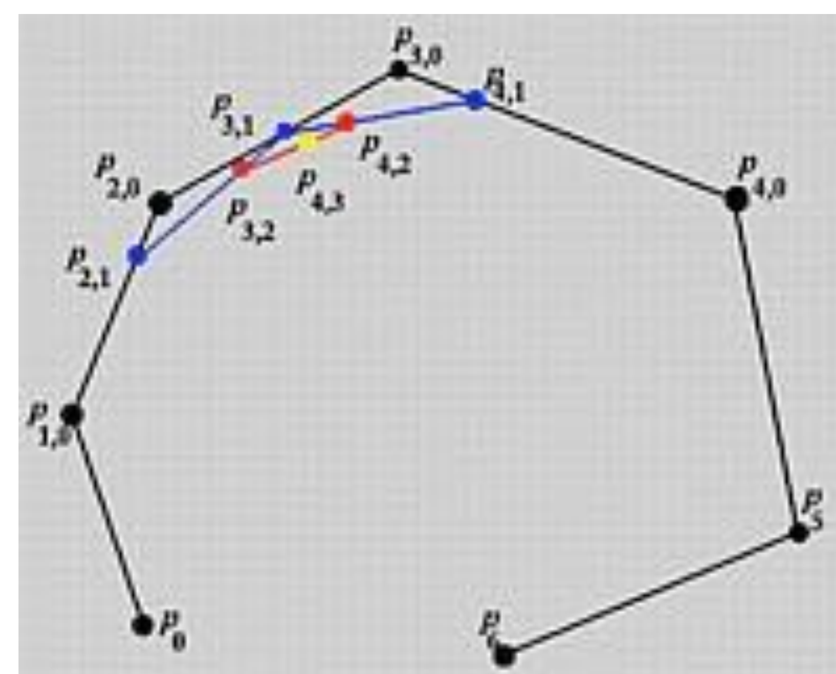
to calculate the point of B-spline curve $\boldsymbol{C}(u)$ at u:

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 $\boldsymbol{P}_{j-p}, \boldsymbol{P}_{j-p+1}, \ldots, \boldsymbol{P}_j$;
3. calculate

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

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





NURBS curve

Catmull-Clark and Doo-Sabin subdivision

Start from

$$P^i = \left(\ldots, p_{-1}^i, p_0^i, p_1^i, p_2^i, \ldots\right)$$

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:

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

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

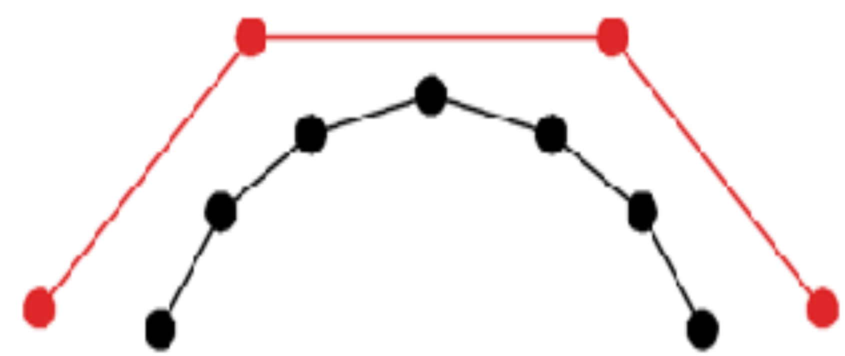


Figure 3: Subdividing an initial set of control points (upper, red) results in additional control points (lower, black), that more closely approximate a curve.

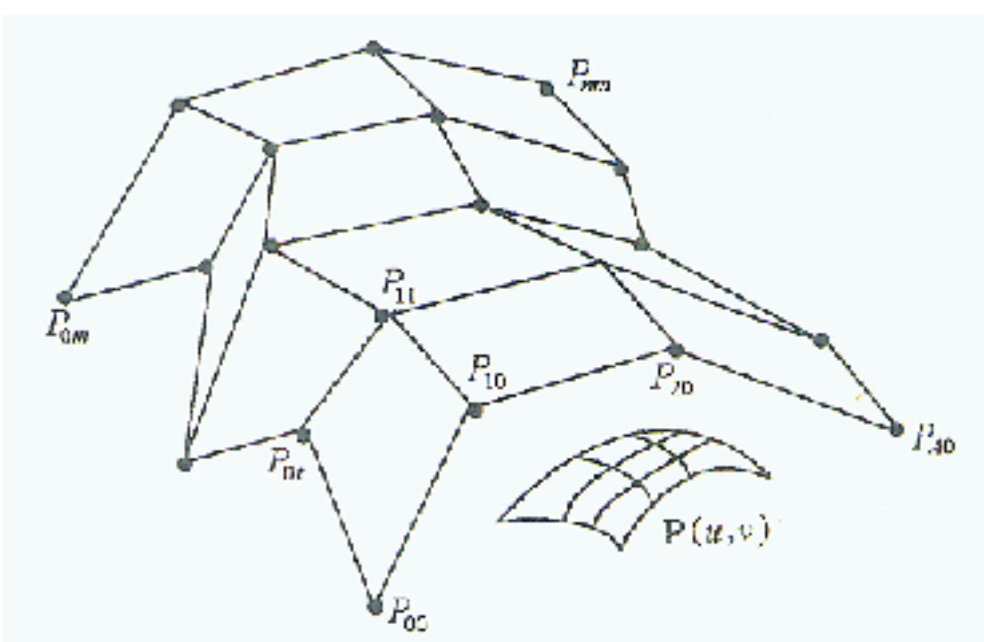
B-spline surface

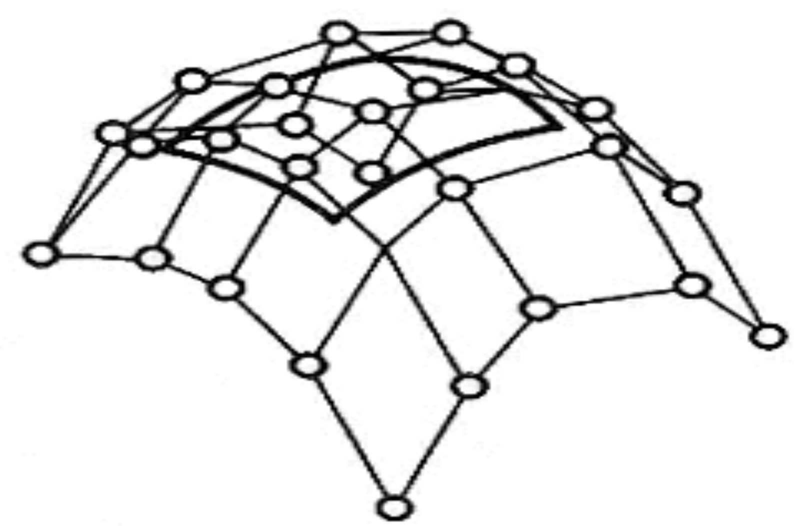
(n+1)×(m+1) control points: $\boldsymbol{P}_{i,j}$ (Degrees of u, v: p, q);

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

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

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





NURBS surface

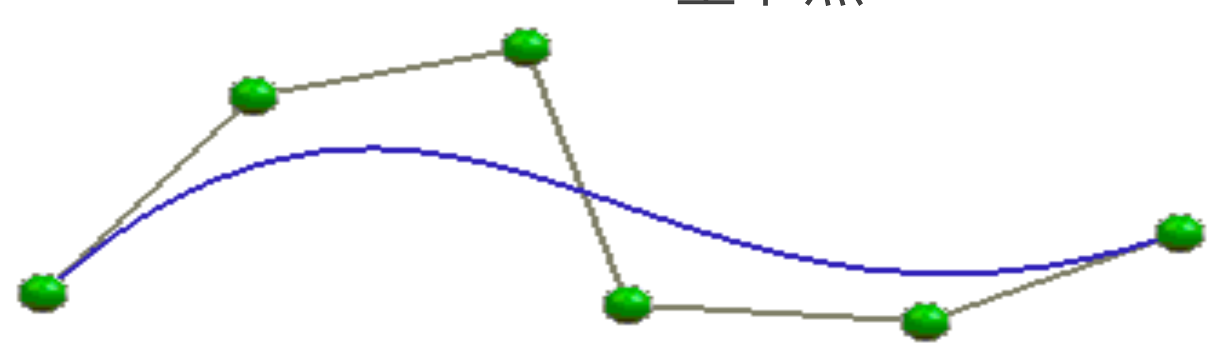
NURBS (Non-uniform **Rational** B-spline)

NURBS curves:

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

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

重节点



NURBS surface

NURBS surface

$$S(u,v)=\frac{\sum_{i=0}^{n}\sum_{j=0}^{m}N_{i,p}(u)N_{j,q}(v)\omega_{i,j}\boldsymbol{P}_{i,j}}{\sum_{i=0}^{n}\sum_{j=0}^{m}N_{i,p}(u)N_{j,q}(v)\omega_{i,j}}\qquad 0\le u,v\le 1$$

ω_{ij}: weights

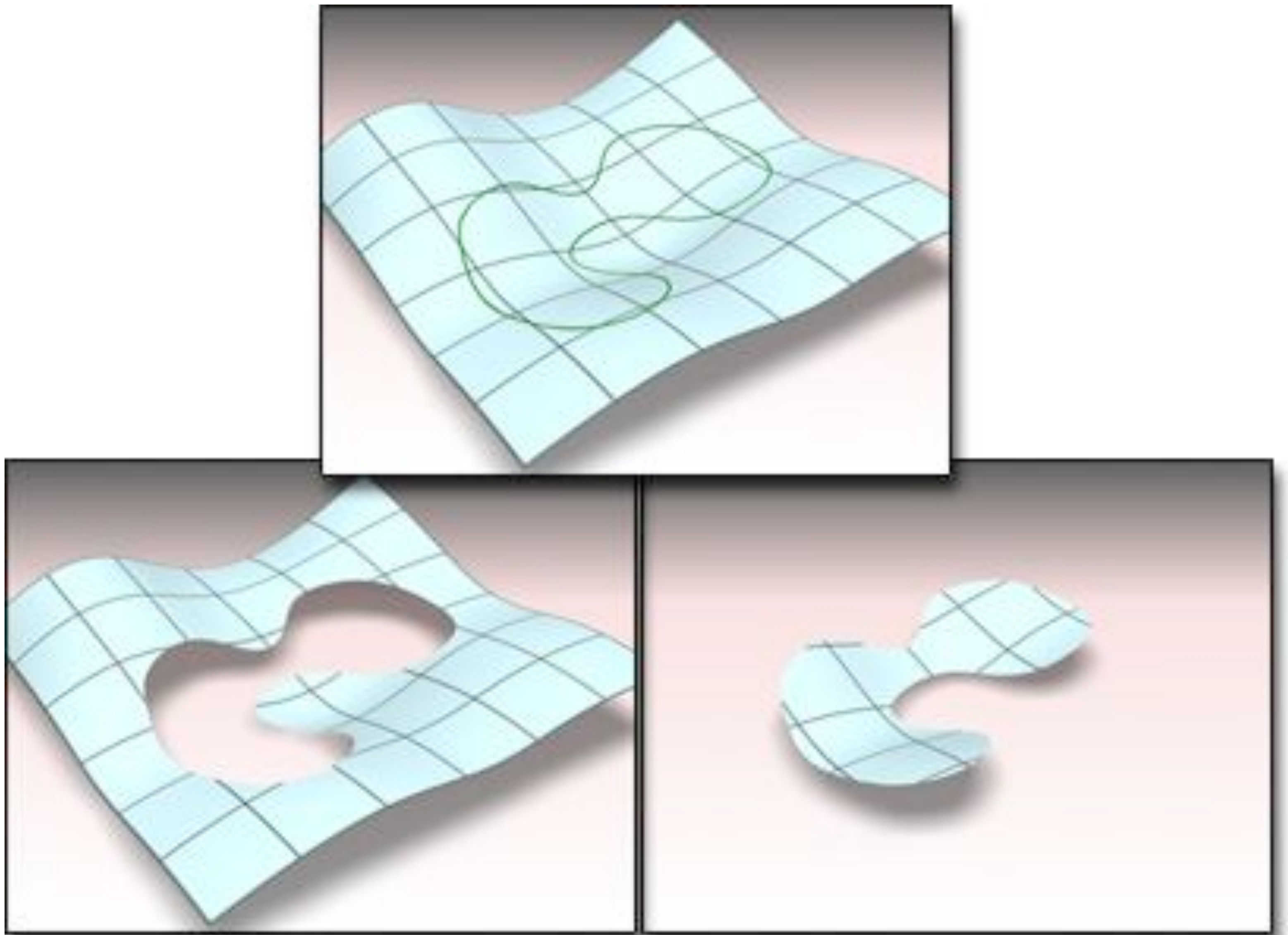
$$U=\{\underbrace{0,\ldots,0}_{p+1},u_{p+1},\ldots,u_{r-p-1},\underbrace{1,\ldots,1}_{p+1}\}$$

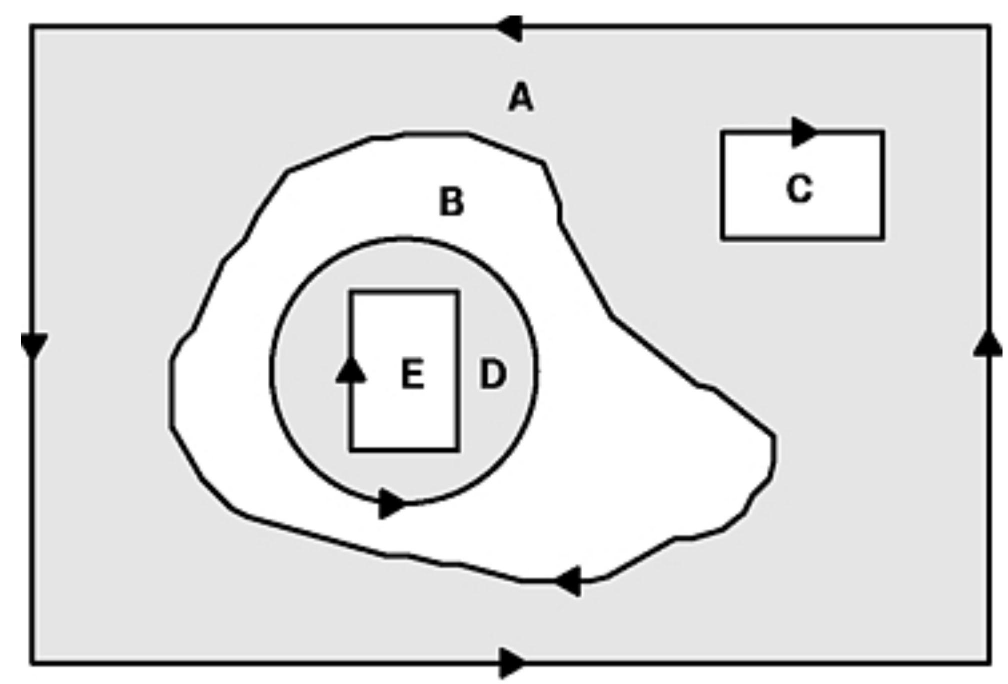
$$V=\{\underbrace{0,\ldots,0}_{q+1},v_{q+1},\ldots,v_{s-q-1},\underbrace{1,\ldots,1}_{q+1}\}$$

NURBS in OpenGL

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

  gluNurbsCurve(curveName, nknots,  *knotVector,
  stride,*ctrlPts, order, GL_MAP1_VERTEX_3);

  gluEndCurve(curveName);
```

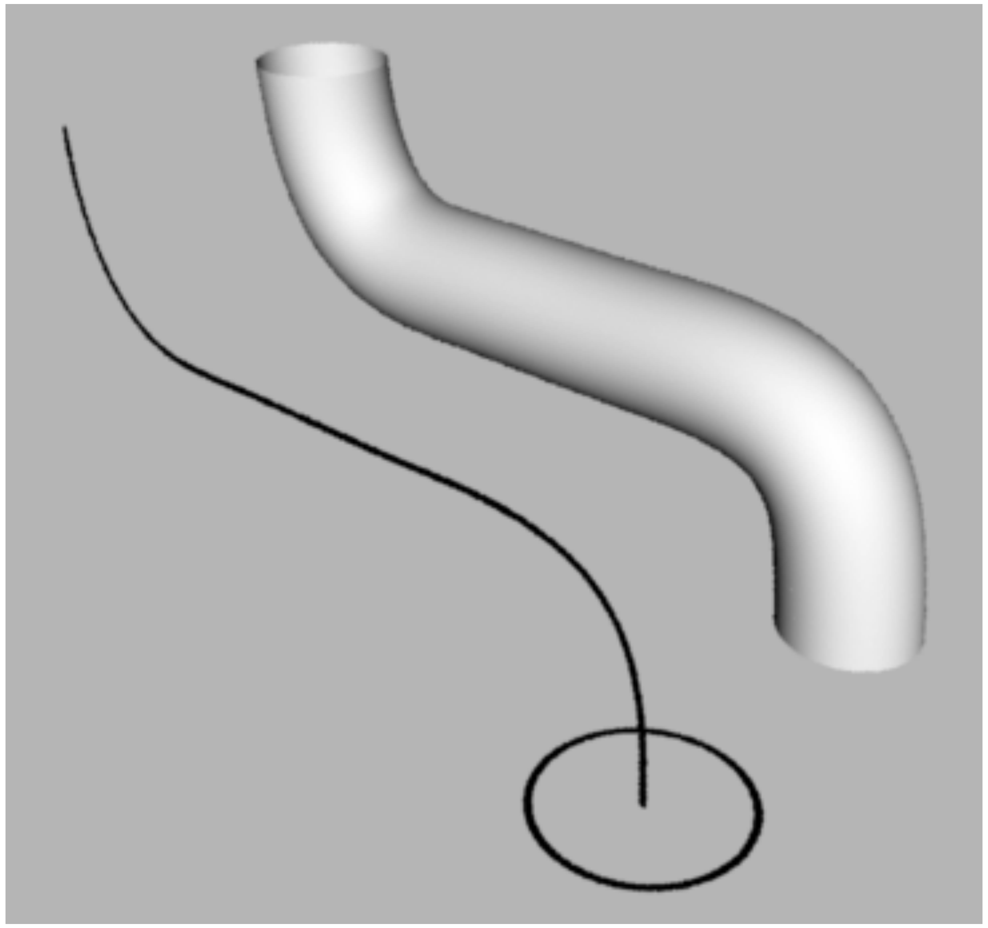
Surface trimming

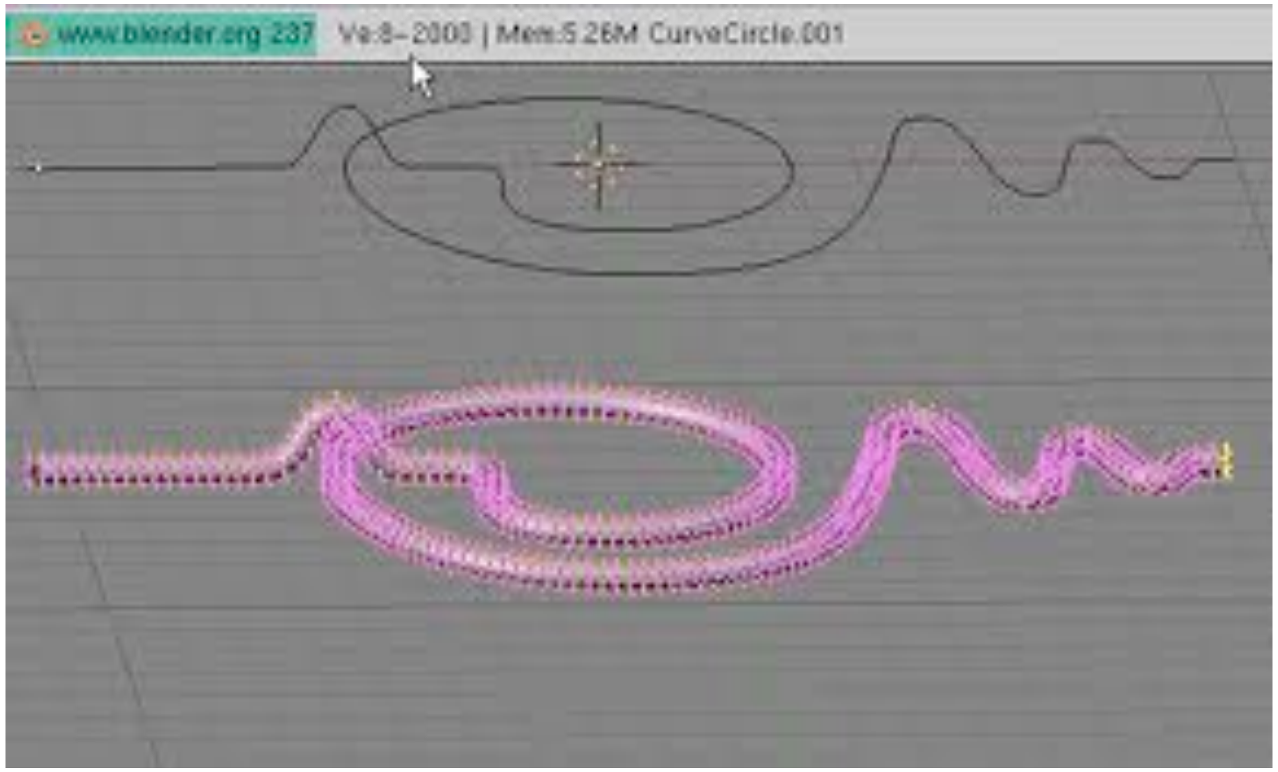




Trimming Loops

Sweeping

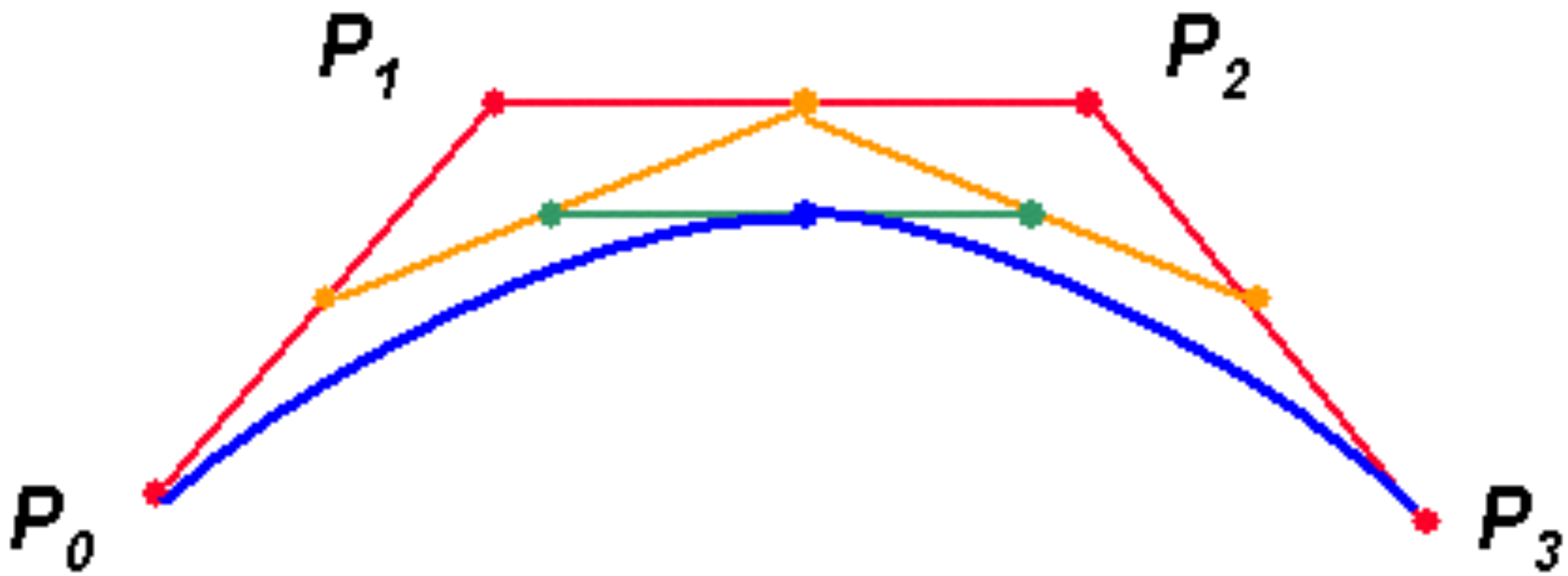




subdivision surface

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