3D Representing Curves and Surfaces

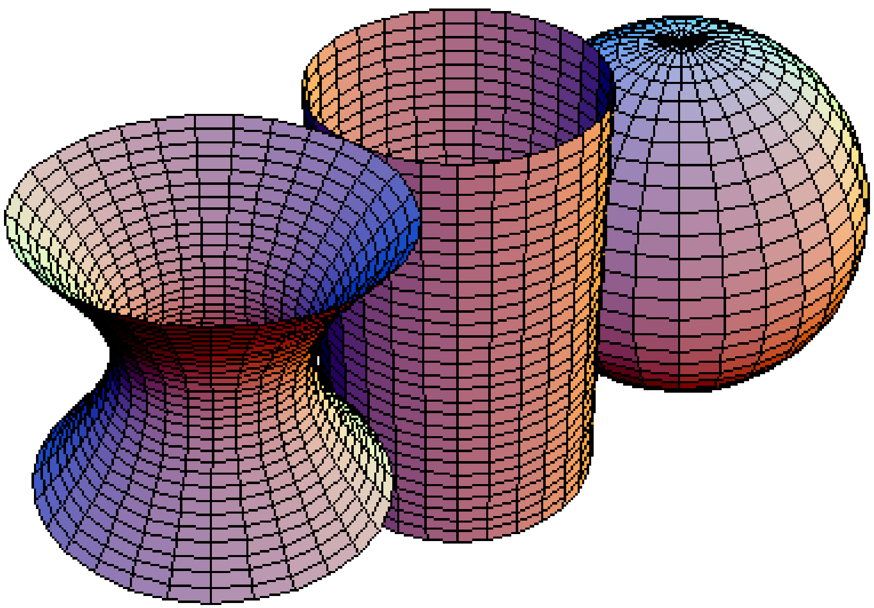
Representing Methods

- Polygon mesh
- Curve surface

Polygon Mesh

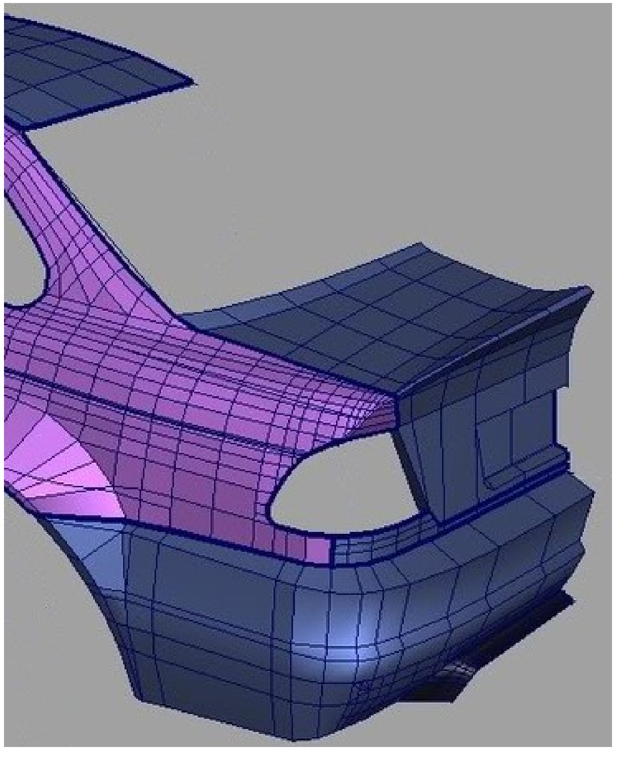
- 3D objects have surfaces that are planar polygons.

Curvature of a cylinder can be represented by many long narrow rectangles



Polygon Mesh

- Curved surfaces can be approximated by planar polyhedrons joined together.

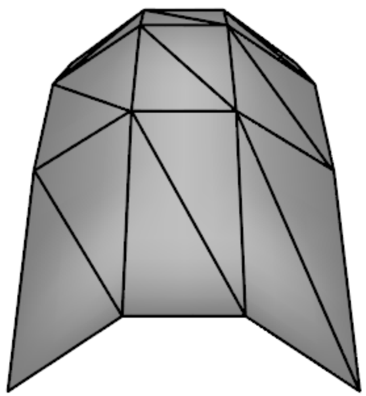


Polygon Mesh

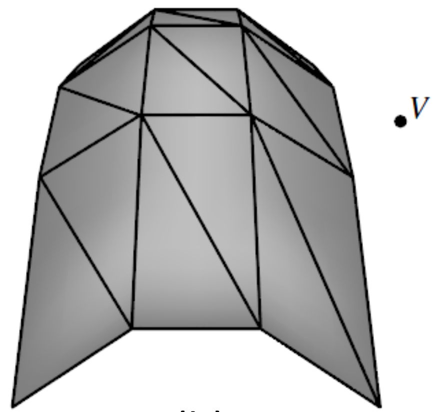
- Vertices are single points; Edges are line segments whose end points are vertices; and Faces are convex polygon in 3D space.
- A finite collection of vertices, edges, and faces is called a polygon mesh such that

- Each vertex must be shared by at least one edge.
- Each edge must be shared by at least one face.
- If two faces intersect, the vertex or edge of intersection must be a component in the mesh.

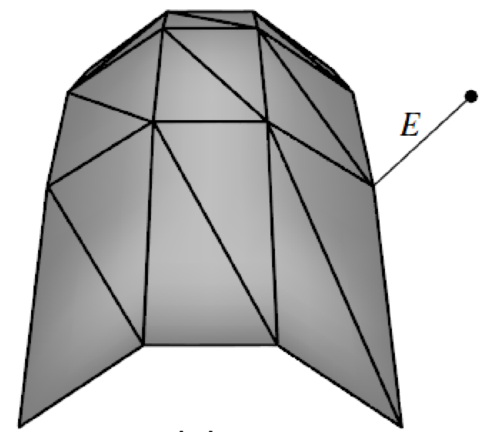
Polygon Mesh



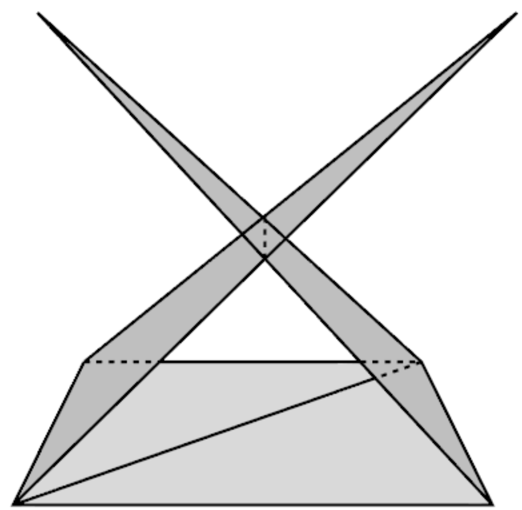
(a)



(b)



(c)



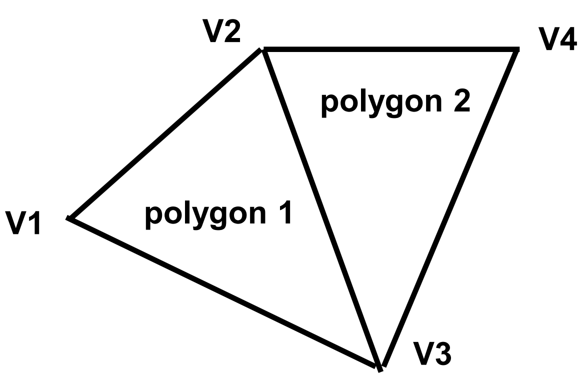
(d)

Polygon Mesh

- Expression 1: Vertex-based list of polygons

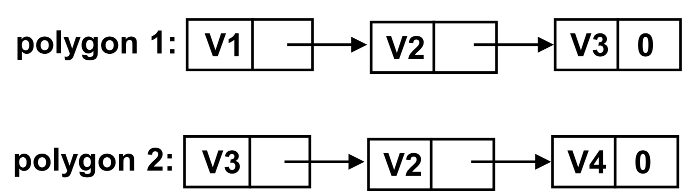
  - stores each vertex in a "vertex table"

  - defines a polygon as a sequence of vertices which can be realized by defining the polygons as linked lists of pointers into the vertex list



Vertex table

ID	x	y	z
V1			
V2			
V3			



Polygon Mesh

- Advantages of this approach

  - It requires the least amount of storage and easily allows for the mesh to be changed.

- Disadvantages of this approach

  - All polygons must be checked to see whether or not they share one specified edge

  - It is the same to find out one specified vertex.

Calculation slows down significantly when the size of polygon mesh increases.

Polygon Mesh

- Expression 2: Edge-based list of polygons

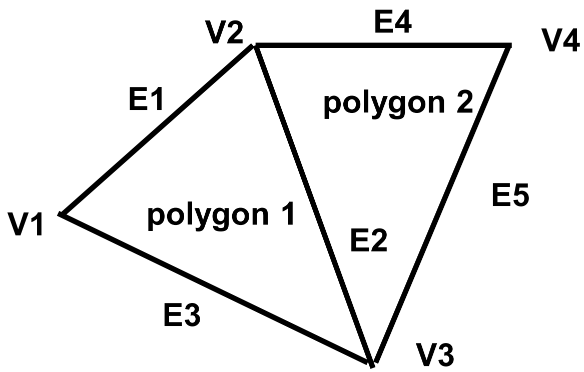
  Table of vertices:

ID	x	y	z
V1			
V2			
V3			

  Linked list of edges: where an edge is connected by two vertices where every edge has a pair of pointers to the vertex table, a pointer into the polygon list and a counter showing the # of polygons that share edge

  List of polygons: a linked list of pointers into the edge list which access (in the correct order) all of the edges that compose that particular polygon

Polygon Mesh



Polygon List

polygon 1: E1 → E2 → E3 0

polygon 2: E2 → E4 → E5 0

List of Edges

Edge 1: V1 V2 1 → P1 0

Edge 2: V2 V3 2 → P1 → P2 0

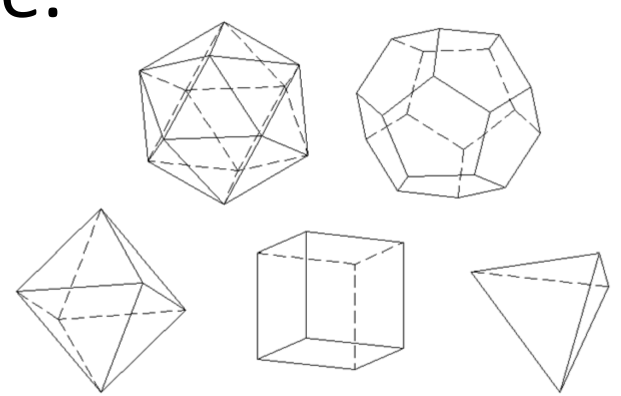
Edge 3: V3 V1 1 → P1 0

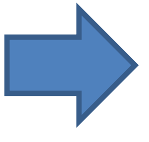
Edge 4: V2 V4 1 → P2 0

Edge 5: V4 V3 1 → P2 0

Polygon Mesh

- Advantage of using planar polygons is low computation by simply drawing the edges for wireframes or by simply filling the polygons
- Disadvantage is ambiguous because all edges may be shown that wouldn't be seen in real life.



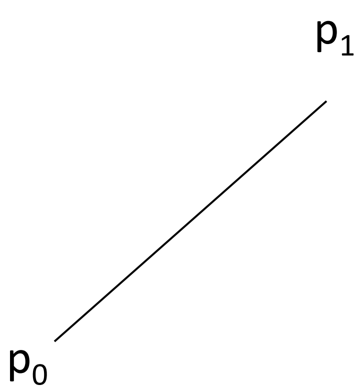


Curve surface

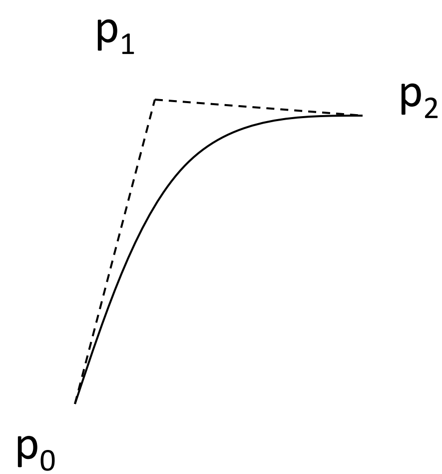
- Bezier curve and Bezier surface
- B-Spline curve and B-Spline surface

Bezier Curve

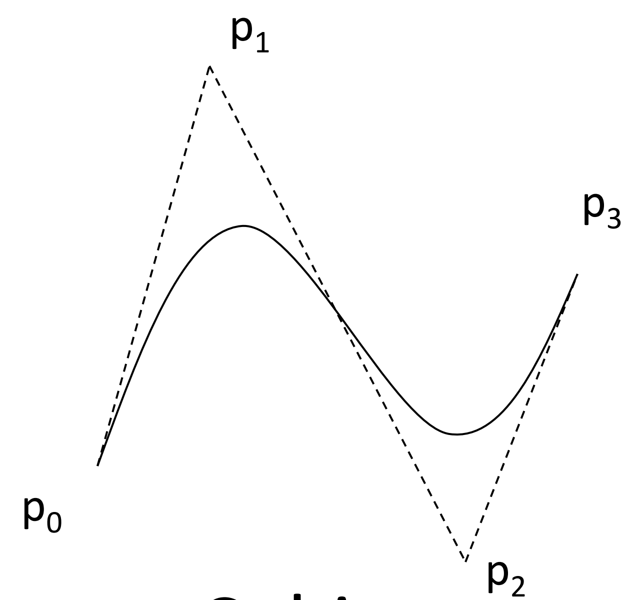
- Bezier curves is a higher order extension of linear interpolation



Linear



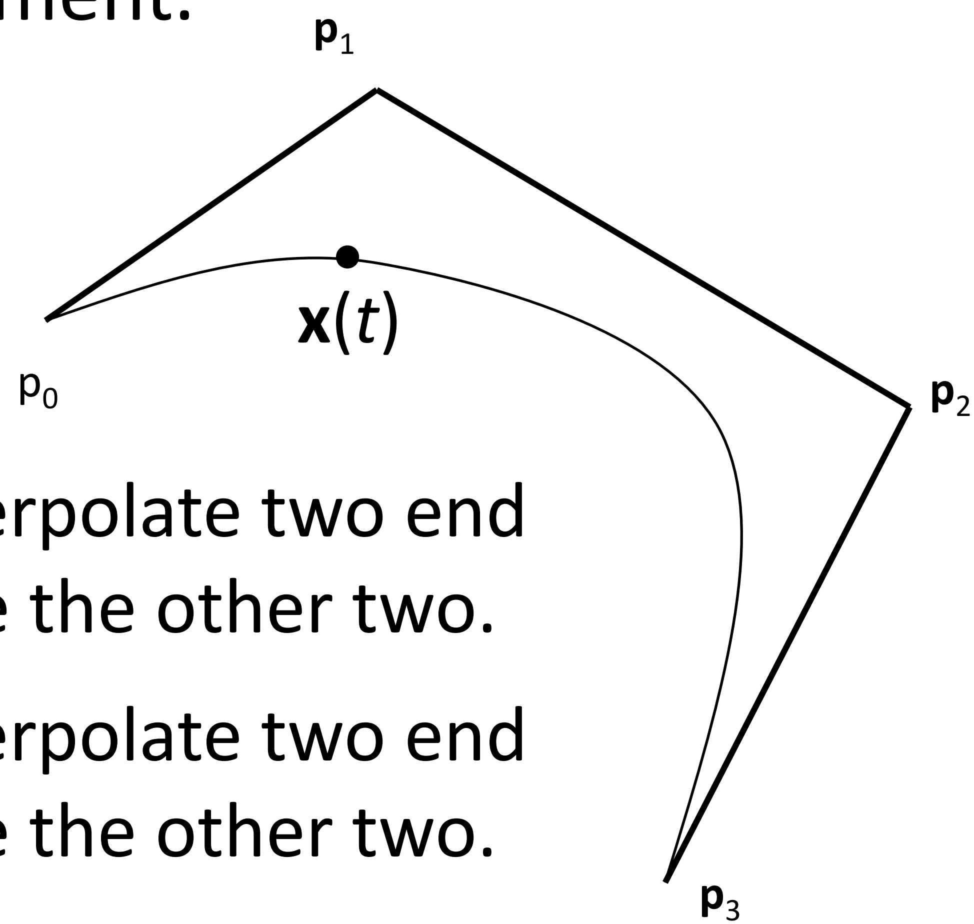
Quadratic



Cubic

Cubic Bezier Curve

- Cubic Bezier curves utilize four points to control one curve segment.

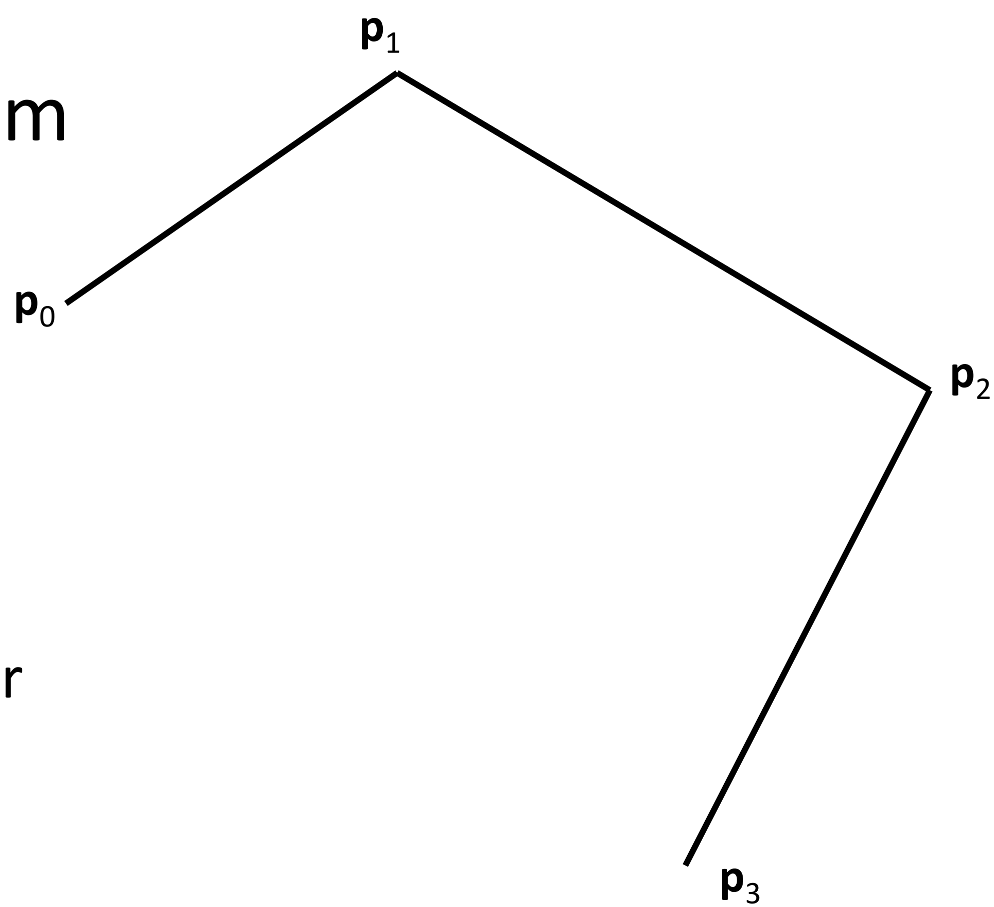


- Cubic Bezier curves interpolate two end points and approximate the other two.

- Cubic Bezier curves interpolate two end points and approximate the other two.

Cubic Bezier Curve

De Casteljau Algorithm



- We start with our original set of points
- In the case of a cubic Bezier curve, we start with four points

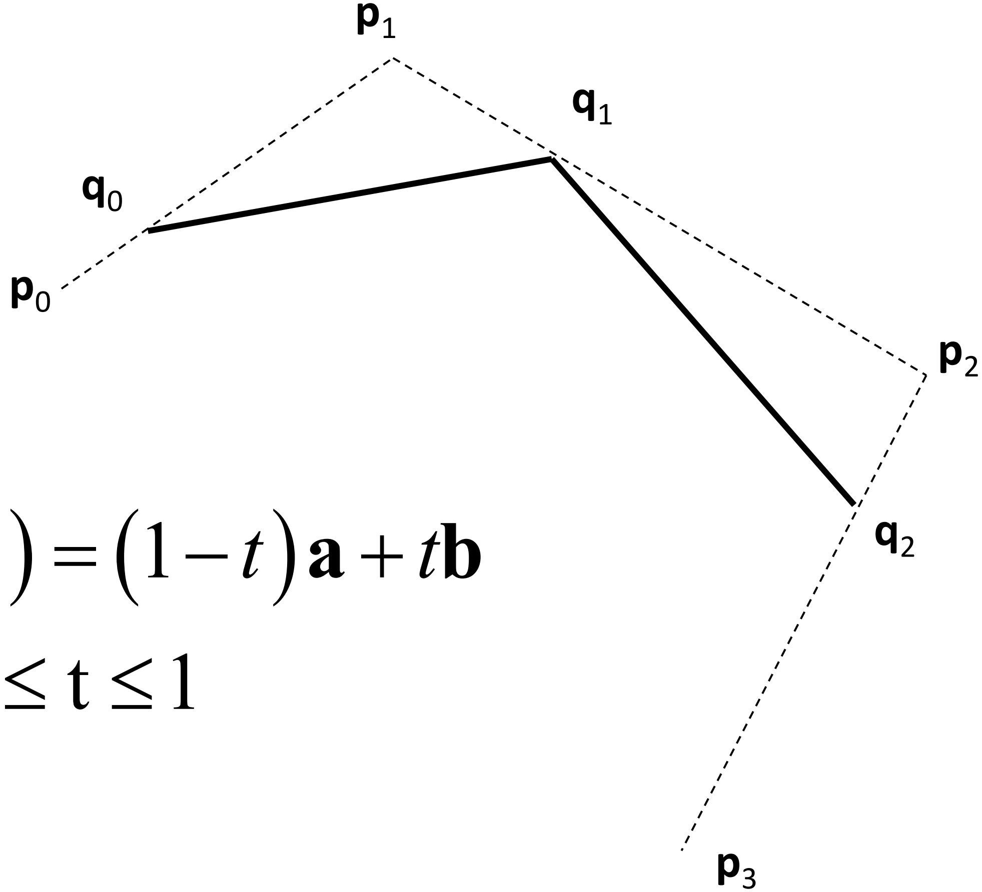
Cubic Bezier Curve

- De Casteljau Algorithm: Step 1

$$\mathbf{q}_0 = B\left(t, \mathbf{p}_0, \mathbf{p}_1\right)$$

$$\mathbf{q}_1 = B\left(t, \mathbf{p}_1, \mathbf{p}_2\right)$$

$$\mathbf{q}_2 = B\left(t, \mathbf{p}_2, \mathbf{p}_3\right)$$

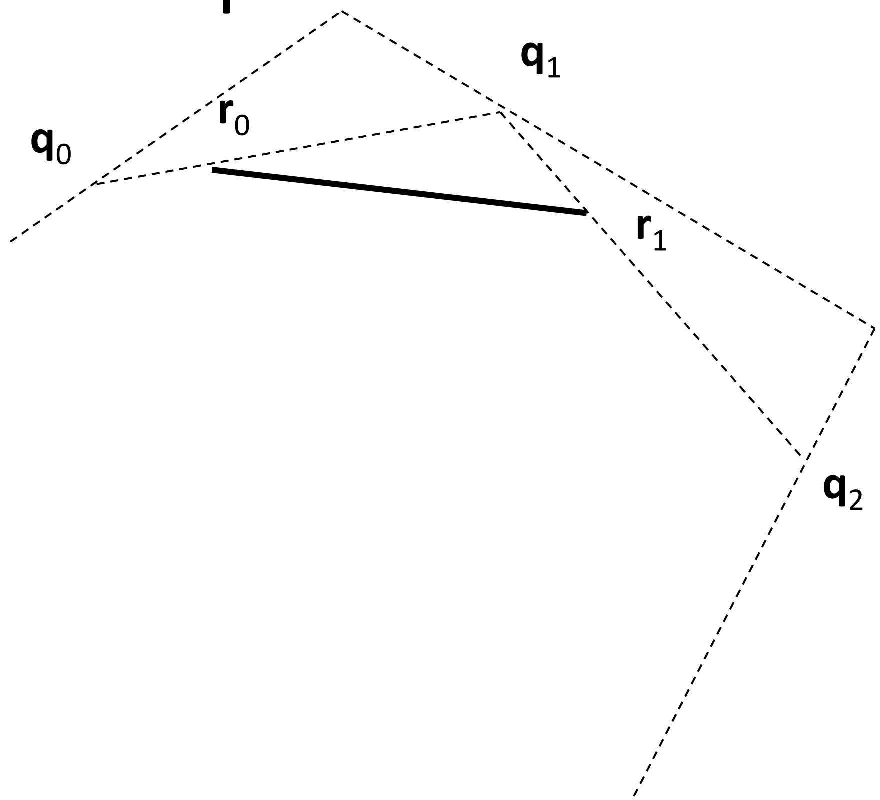


$$B\left(t, \mathbf{a}, \mathbf{b}\right) = \left(1 - t\right)\mathbf{a} + t\mathbf{b}$$

$$0 \le \mathrm{t} \le 1$$

Cubic Bezier Curve

- De Casteljau Algorithm: Step 2

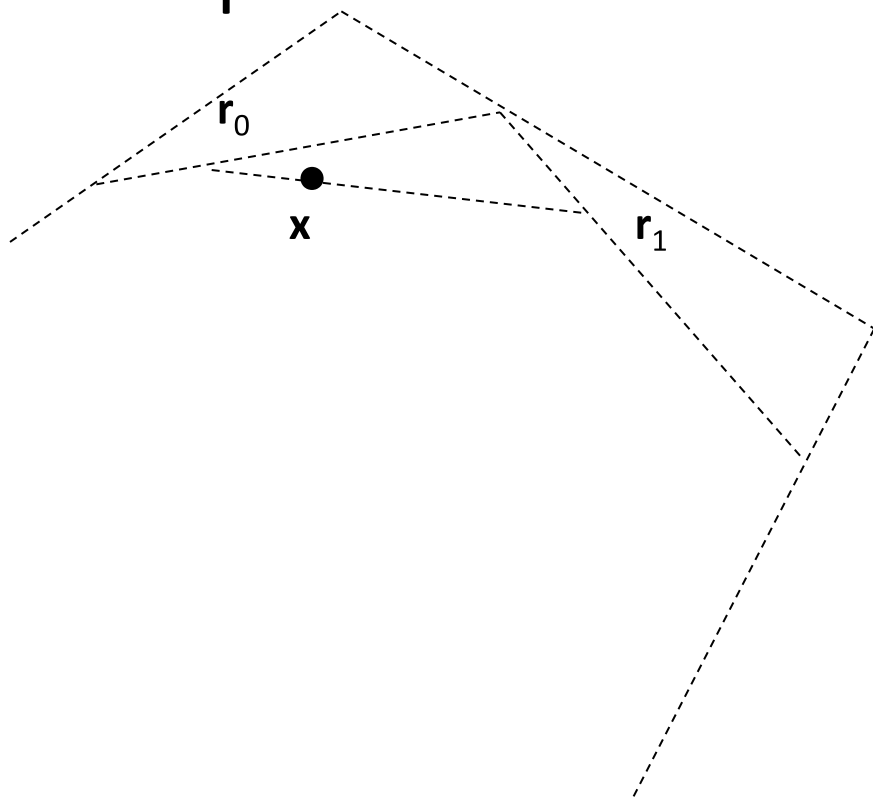


$$\mathbf{r}_0 = B\left(t, \mathbf{q}_0, \mathbf{q}_1\right)$$

$$\mathbf{r}_1 = B\left(t, \mathbf{q}_1, \mathbf{q}_2\right)$$

de Casteljau Algorithm

- De Casteljau Algorithm: Step 3



$$\mathbf{x} = B\left(t, \mathbf{r}_0, \mathbf{r}_1\right)$$

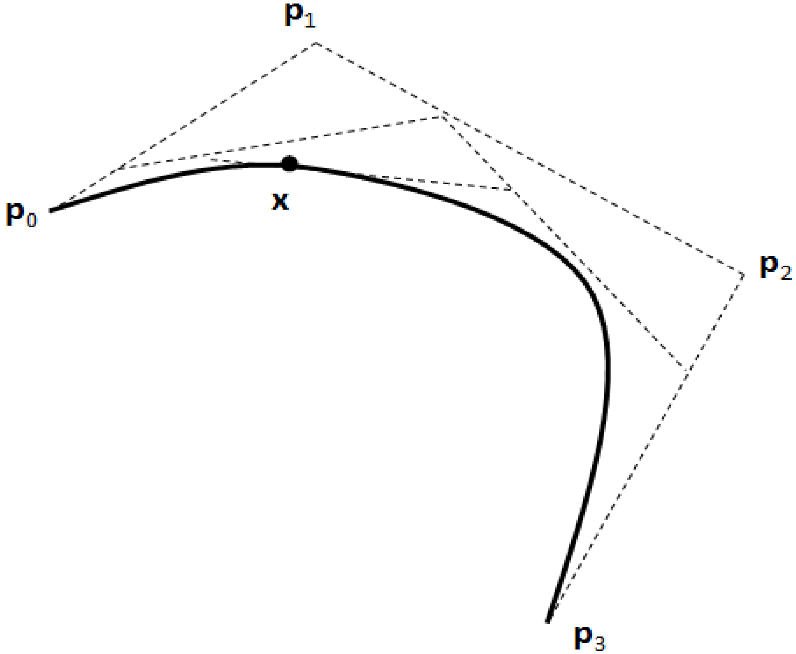
Recursive Linear Interpolation

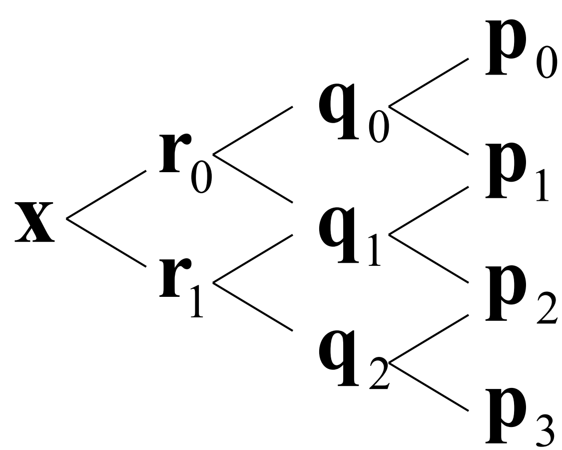
$$\mathbf{p}_0 \quad \mathbf{p}_1 \quad \mathbf{p}_2 \quad \mathbf{p}_3$$

$$\mathbf{q}_0 = B(t, \mathbf{p}_0, \mathbf{p}_1)$$
$$\mathbf{q}_1 = B(t, \mathbf{p}_1, \mathbf{p}_2)$$
$$\mathbf{q}_2 = B(t, \mathbf{p}_2, \mathbf{p}_3)$$

$$\mathbf{r}_0 = B(t, \mathbf{q}_0, \mathbf{q}_1)$$
$$\mathbf{r}_1 = B(t, \mathbf{q}_1, \mathbf{q}_2)$$

$$\mathbf{x} = B(t, \mathbf{r}_0, \mathbf{r}_1)$$





Expanding the B function

$$\mathbf{q}_0 = B\left(t,\mathbf{p}_0,\mathbf{p}_1\right) = \left(1-t\right)\mathbf{p}_0 + t\mathbf{p}_1$$
$$\mathbf{q}_1 = B\left(t,\mathbf{p}_1,\mathbf{p}_2\right) = \left(1-t\right)\mathbf{p}_1 + t\mathbf{p}_2$$
$$\mathbf{q}_2 = B\left(t,\mathbf{p}_2,\mathbf{p}_3\right) = \left(1-t\right)\mathbf{p}_2 + t\mathbf{p}_3$$

$$\mathbf{r}_0 = B\left(t,\mathbf{q}_0,\mathbf{q}_1\right) = \left(1-t\right)\left(\left(1-t\right)\mathbf{p}_0 + t\mathbf{p}_1\right) + t\left(\left(1-t\right)\mathbf{p}_1 + t\mathbf{p}_2\right)$$
$$\mathbf{r}_1 = B\left(t,\mathbf{q}_1,\mathbf{q}_2\right) = \left(1-t\right)\left(\left(1-t\right)\mathbf{p}_1 + t\mathbf{p}_2\right) + t\left(\left(1-t\right)\mathbf{p}_2 + t\mathbf{p}_3\right)$$

$$\begin{aligned}\mathbf{x} = B\left(t,\mathbf{r}_0,\mathbf{r}_1\right) &= \left(1-t\right)\left(\left(1-t\right)\left(\left(1-t\right)\mathbf{p}_0 + t\mathbf{p}_1\right) + t\left(\left(1-t\right)\mathbf{p}_1 + t\mathbf{p}_2\right)\right) \\ &\quad + t\left(\left(1-t\right)\left(\left(1-t\right)\mathbf{p}_1 + t\mathbf{p}_2\right) + t\left(\left(1-t\right)\mathbf{p}_2 + t\mathbf{p}_3\right)\right)\end{aligned}$$

Bernstein Polynomials

$$\mathbf{x}(t) = \left(-t^3 + 3t^2 - 3t + 1\right)\mathbf{p}_0 + \left(3t^3 - 6t^2 + 3t\right)\mathbf{p}_1 + \left(3t^3 - 6t^2 + 3t\right)\mathbf{p}_2 + \left(t^3\right)\mathbf{p}_3$$

$$\mathbf{x}(t) = \sum B_i^3(t)\mathbf{p}_i$$

$$B_0^3(t) = -t^3 + 3t^2 - 3t + 1$$

$$B_1^3(t) = 3t^3 - 6t^2 + 3t$$

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

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

$$B_i^n(t) = \begin{pmatrix} n \\ i \end{pmatrix} (1-t)^{n-i}(t)^i \qquad \begin{pmatrix} n \\ i \end{pmatrix} = \frac{n!}{i!(n-i)!}$$

Bernstein Polynomials

- The Bernstein polynomial form of a Bezier curve is

$$\mathbf{x}(t) = \sum_{i=0}^{n} B_i^n(t)\,\mathbf{p}_i$$

where

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

$$\binom{n}{i} = \frac{n!}{i!(n-i)!}$$

Cubic Equation Form

$$\mathbf{x} = \left(-\mathbf{p}_0 + 3\mathbf{p}_1 - 3\mathbf{p}_2 + \mathbf{p}_3\right)t^3 + \left(3\mathbf{p}_0 - 6\mathbf{p}_1 + 3\mathbf{p}_2\right)t^2$$
$$+ \left(-3\mathbf{p}_0 + 3\mathbf{p}_1\right)t + \left(\mathbf{p}_0\right)1$$

$$\boxed{\mathbf{x} = \mathbf{a}t^3 + \mathbf{b}t^2 + \mathbf{c}t + \mathbf{d} \qquad \begin{aligned} \mathbf{a} &= \left(-\mathbf{p}_0 + 3\mathbf{p}_1 - 3\mathbf{p}_2 + \mathbf{p}_3\right) \\ \mathbf{b} &= \left(3\mathbf{p}_0 - 6\mathbf{p}_1 + 3\mathbf{p}_2\right) \\ \mathbf{c} &= \left(-3\mathbf{p}_0 + 3\mathbf{p}_1\right) \\ \mathbf{d} &= \left(\mathbf{p}_0\right) \end{aligned}}$$

Cubic Matrix Form

$$\mathbf{x} = \mathbf{a}t^3 + \mathbf{b}t^2 + \mathbf{c}t + \mathbf{d}$$

$$\mathbf{a} = \left(-\mathbf{p}_0 + 3\mathbf{p}_1 - 3\mathbf{p}_2 + \mathbf{p}_3\right)$$
$$\mathbf{b} = \left(3\mathbf{p}_0 - 6\mathbf{p}_1 + 3\mathbf{p}_2\right)$$
$$\mathbf{c} = \left(-3\mathbf{p}_0 + 3\mathbf{p}_1\right)$$
$$\mathbf{d} = \left(\mathbf{p}_0\right)$$

$$\mathbf{x} = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \cdot \begin{bmatrix} \mathbf{a} \\ \mathbf{b} \\ \mathbf{c} \\ \mathbf{d} \end{bmatrix}$$

$$\begin{bmatrix} \mathbf{a} \\ \mathbf{b} \\ \mathbf{c} \\ \mathbf{d} \end{bmatrix} = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} \mathbf{p}_0 \\ \mathbf{p}_1 \\ \mathbf{p}_2 \\ \mathbf{p}_3 \end{bmatrix}$$

Cubic Matrix Form

$$\mathbf{x} = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \cdot \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} \mathbf{p}_0 \\ \mathbf{p}_1 \\ \mathbf{p}_2 \\ \mathbf{p}_3 \end{bmatrix}$$

$$\mathbf{x} = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \cdot \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} p_{0x} & p_{0y} & p_{0z} \\ p_{1x} & p_{1y} & p_{1z} \\ p_{2x} & p_{2y} & p_{2z} \\ p_{3x} & p_{3y} & p_{3z} \end{bmatrix}$$

Cubic Matrix Form

$$\mathbf{x} = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \cdot \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} p_{0x} & p_{0y} & p_{0z} \\ p_{1x} & p_{1y} & p_{1z} \\ p_{2x} & p_{2y} & p_{2z} \\ p_{3x} & p_{3y} & p_{3z} \end{bmatrix}$$

$$\mathrm{C}^0: \quad \mathbf{x} = \mathbf{t} \cdot \mathbf{B}_{Bez} \cdot \mathbf{G}_{Bez}$$

$$\Rightarrow C^1: \quad \frac{dx}{dt} \quad \text{and} \quad \mathrm{C}^2 = \frac{d}{dt}\left(\frac{dx}{dt}\right)$$

Derivatives

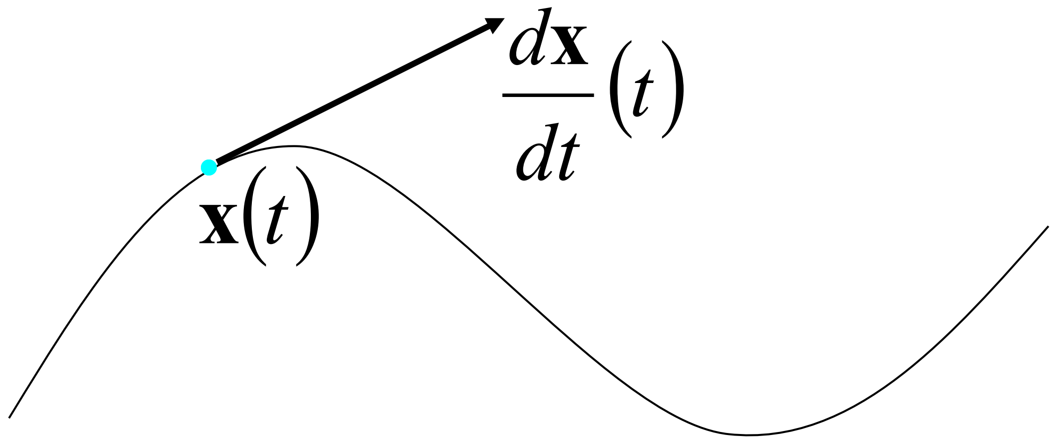
- The derivative (tangent) of a curve :

$$\mathbf{x} = \mathbf{a}t^3 + \mathbf{b}t^2 + \mathbf{c}t + \mathbf{d} \qquad \frac{d\mathbf{x}}{dt} = 3\mathbf{a}t^2 + 2\mathbf{b}t + \mathbf{c}$$

$$\mathbf{x} = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \cdot \begin{bmatrix} \mathbf{a} \\ \mathbf{b} \\ \mathbf{c} \\ \mathbf{d} \end{bmatrix} \qquad \frac{d\mathbf{x}}{dt} = \begin{bmatrix} 3t^2 & 2t & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} \mathbf{a} \\ \mathbf{b} \\ \mathbf{c} \\ \mathbf{d} \end{bmatrix}$$

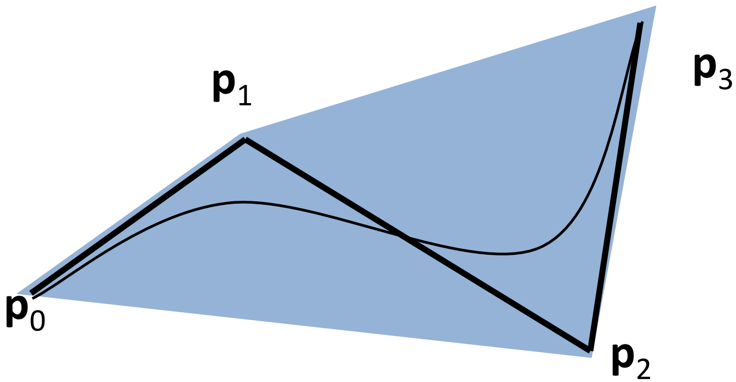
Tangents

- The derivative of a curve represents the tangent vector to the curve at some point



Convex Hull Property

- Convex hull of the curve is the polygon of control points.
- Every point on the curve is within the convex hull.



Continuity

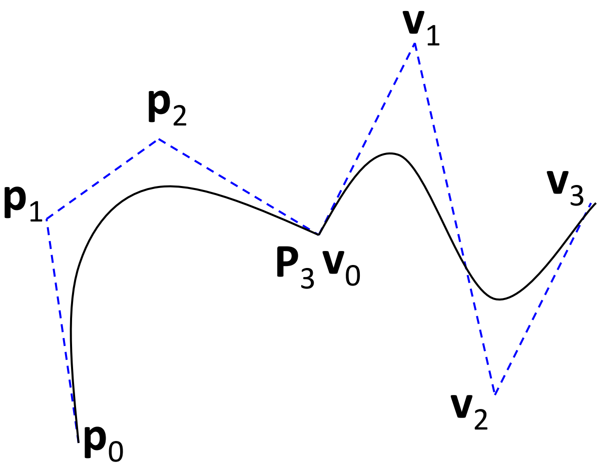
- A cubic curve is a single continuous curve and not have any gaps.
- We say that it has *geometric continuity*, or *C^0 continuity*
- The first derivative C^1 will be a continuous quadratic function and the second derivative will be a continuous linear function
- A cubic curve has *second derivative continuity* or *C^2 continuity*
- In general, the higher the continuity value, the 'smoother' the curve will be.

Piecewise Curves

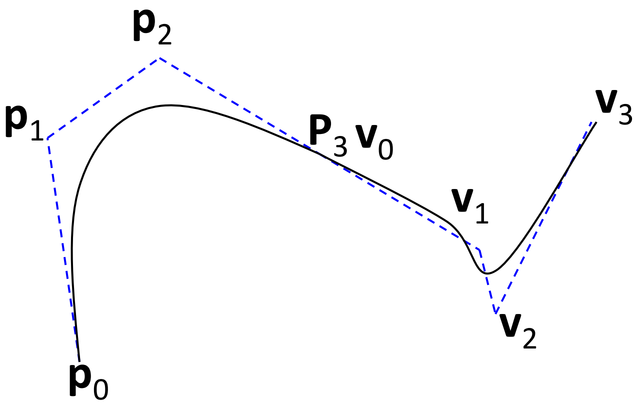
- Rather than use a very high degree curve to interpolate a large number of points, it is more common to break the curve up into several simple curves
- For example, a large complex curve could be broken into cubic curves, and would therefore be a *piecewise cubic curve*
- For the entire curve to look smooth and continuous, it is necessary to maintain C^1 continuity across segments, meaning that the position and tangents must match at the endpoints
- For smoother looking curves, it is best to maintain the C^2 continuity as well

Connecting Bezier Curves

- A simple way to make larger curves is to connect up Bezier curves
- Consider two Bezier curves defined by $\mathbf{p}_0 \ldots \mathbf{p}_3$ and $\mathbf{v}_0 \ldots \mathbf{v}_3$
- If $\mathbf{p}_3 = \mathbf{v}_0$, then they will have C^0 continuity
- If $(\mathbf{p}_3 - \mathbf{p}_2) = (\mathbf{v}_1 - \mathbf{v}_0)$, then they will have C^1 continuity
- C^2 continuity is more difficult…



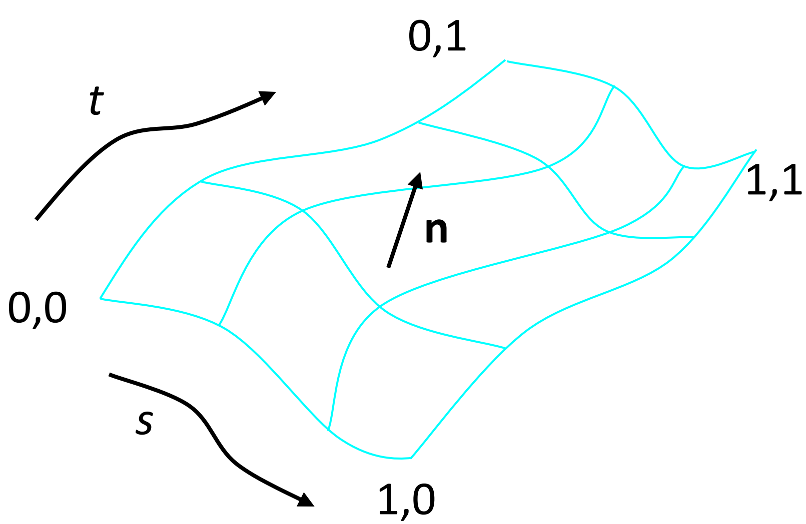
C^0 continuity



C^1 continuity

Bezier Surfaces

- Bezier surfaces are a straightforward extension to Bezier curves
- Instead of the curve being parameterized by a single variable *t*, we use two variables, *s* and *t*
- By definition, we choose to have *s* and *t* range from 0 to 1 and we say that an *s*-tangent crossed with a *t*-tangent will represent the normal for the front of the surface

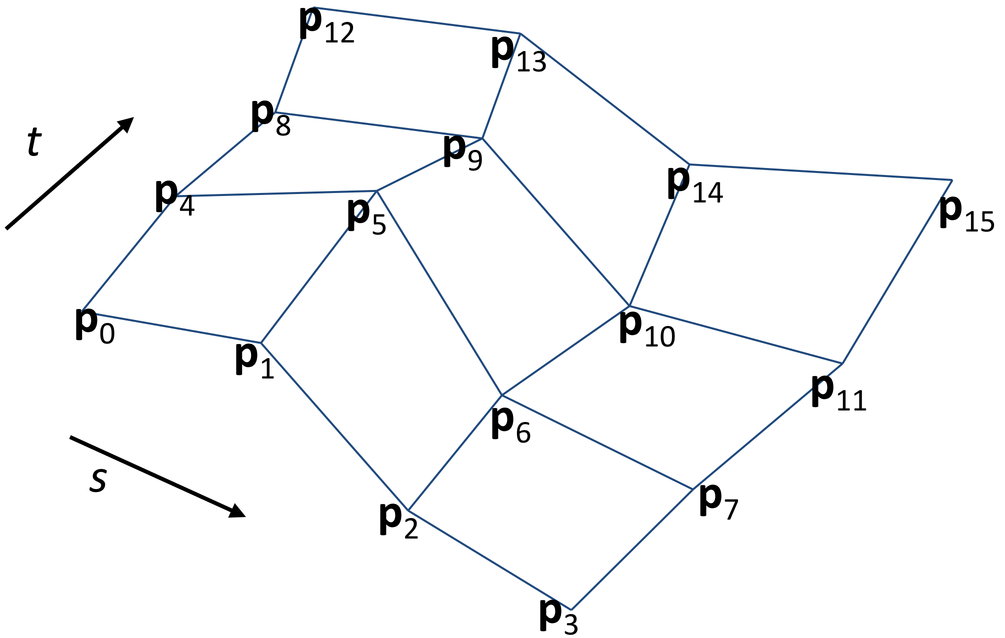


Curved Surfaces

- The Bezier surface is a type of *parametric surface*
- A parametric surface is a surface that can be parameterized by two variables, F(s, t)
- Parametric surfaces have a rectangular topology
- Parametric surfaces are sometimes called *patches*, *curved surfaces*, or just *surfaces*

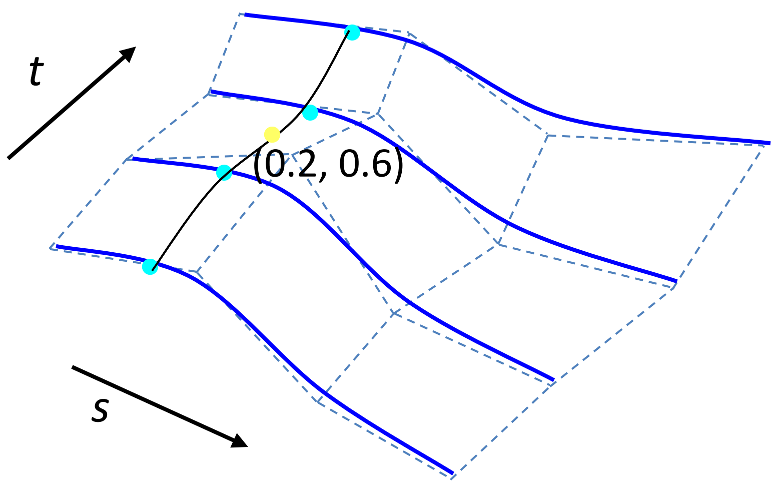
Control Mesh

- Consider a *bicubic* Bezier surface (bicubic means that it is a cubic function in both the *s* and *t* parameters)
- A cubic curve has 4 control points, and a bicubic surface has a grid of 4x4 control points, $\mathbf{p}_0$ through $\mathbf{p}_{15}$



Surface Evaluation

- The bicubic surface can be thought of as 4 curves along the *s* parameter (or alternately as 4 curves along the *t* parameter)
- To compute the location of the surface for some (*s*,*t*) pair, we can first solve each of the 4 *s*-curves for the specified value of *s*
- Those 4 points now make up a new curve which we evaluate at *t*
- Alternately, if we first solve the 4 t-curves and to create a new curve which we then evaluate at *s*, we will get the exact same answer
- This gives a pretty straightforward way to implement smooth surfaces with little more than what is needed to implement curves



Matrix Form

- To simplify notation for surfaces, we will define a matrix equation for each of the *x*, *y*, and *z* components, instead of combining them into a single equation as for curves
- For example, to evaluate the *x* component of a Bezier curve, we can use:

$$x = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \cdot \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} p_{0x} \\ p_{1x} \\ p_{2x} \\ p_{3x} \end{bmatrix}$$

$$x = \mathbf{t} \cdot \mathbf{B}_{Bez} \cdot \mathbf{g}_x$$

$$x = \mathbf{t} \cdot \mathbf{c}_x$$

Matrix Form

- To evaluate the *x* component of 4 curves simultaneously, we can combine 4 curves into a 4x4 matrix
- To evaluate a surface, we evaluate the 4 curves, and use them to make a new curve which is then evaluated
- This can be written in a compact matrix form:

$$x(s,t) = \mathbf{s} \cdot \mathbf{B}_{Bez} \cdot \mathbf{G}_x \cdot \mathbf{B}_{Bez}^T \cdot \mathbf{t}^T$$

$$\mathbf{s} = \begin{bmatrix} s^3 & s^2 & s & 1 \end{bmatrix}$$
$$\mathbf{t} = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix}$$

Matrix Form

$$\mathbf{x}(s,t)=\begin{bmatrix}\mathbf{s}\cdot\mathbf{C}_x\cdot\mathbf{t}^T\\ \mathbf{s}\cdot\mathbf{C}_y\cdot\mathbf{t}^T\\ \mathbf{s}\cdot\mathbf{C}_z\cdot\mathbf{t}^T\end{bmatrix}$$

$$\mathbf{B}_{Bez}=\begin{bmatrix}-1 & 3 & -3 & 1\\ 3 & -6 & 3 & 0\\ -3 & 3 & 0 & 0\\ 1 & 0 & 0 & 0\end{bmatrix}=\mathbf{B}_{Bez}^T$$

$$\mathbf{C}_x=\mathbf{B}_{Bez}\cdot\mathbf{G}_x\cdot\mathbf{B}_{Bez}^T$$

$$\mathbf{s}=\begin{bmatrix}s^3 & s^2 & s & 1\end{bmatrix}$$

$$\mathbf{t}=\begin{bmatrix}t^3 & t^2 & t & 1\end{bmatrix}$$

$$\mathbf{G}_x=\begin{bmatrix}p_{0x} & p_{4x} & p_{8x} & p_{12x}\\ p_{1x} & p_{5x} & p_{9x} & p_{13x}\\ p_{2x} & p_{6x} & p_{10x} & p_{14x}\\ p_{3x} & p_{7x} & p_{11x} & p_{15x}\end{bmatrix}$$

Matrix Form

- $\mathbf{C}_x$ stores the coefficients of the bicubic equation for *x*
- $\mathbf{G}_x$ stores the geometry (*x* components of the control points)
- $\mathbf{B}_{Bez}$ is the basis matrix (Bezier basis)
- **s** and **t** are the vectors formed from the exponents of *s* and *t*
- The matrix form leads to an efficient method of computation
- It can also take advantage of 4x4 matrix support which is built into modern graphics hardware

Tangents

- To compute the *s* and *t* tangent vectors at some (*s*,*t*) location, we can use:

$$\frac{\partial \mathbf{x}}{\partial s} = \begin{bmatrix} d\mathbf{s} \cdot \mathbf{C}_x \cdot \mathbf{t}^T \\ d\mathbf{s} \cdot \mathbf{C}_y \cdot \mathbf{t}^T \\ d\mathbf{s} \cdot \mathbf{C}_z \cdot \mathbf{t}^T \end{bmatrix}$$

$$\mathbf{s} = \begin{bmatrix} s^3 & s^2 & s & 1 \end{bmatrix}$$

$$\mathbf{t} = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix}$$

$$\frac{\partial \mathbf{x}}{\partial t} = \begin{bmatrix} \mathbf{s} \cdot \mathbf{C}_x \cdot d\mathbf{t}^T \\ \mathbf{s} \cdot \mathbf{C}_y \cdot d\mathbf{t}^T \\ \mathbf{s} \cdot \mathbf{C}_z \cdot d\mathbf{t}^T \end{bmatrix}$$

$$d\mathbf{s} = \begin{bmatrix} 3s^2 & 2s & 1 & 0 \end{bmatrix}$$

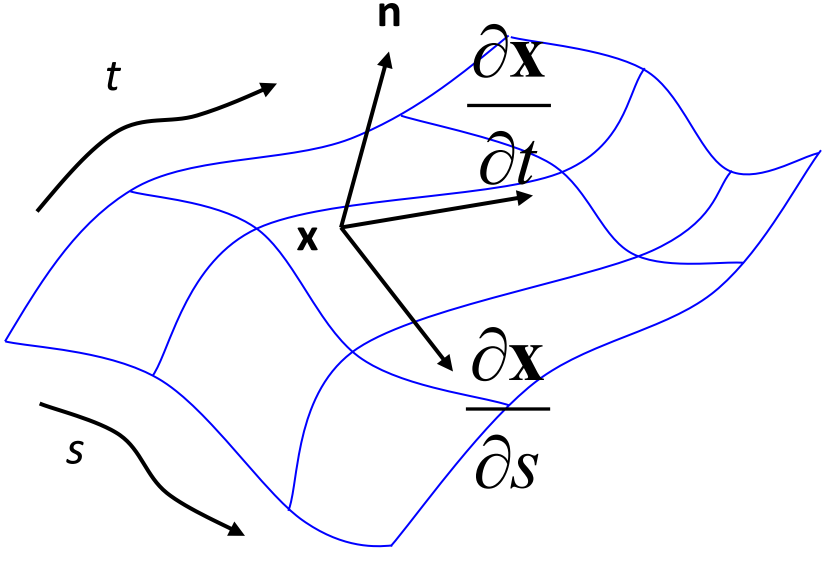
$$d\mathbf{t} = \begin{bmatrix} 3t^2 & 2t & 1 & 0 \end{bmatrix}$$

Normals

- To compute the normal of the surface at some location (*s*,*t*), we compute the two tangents at that location and then take their cross product
- Normalization

$$\mathbf{n}^* = \frac{\partial \mathbf{x}}{\partial s} \times \frac{\partial \mathbf{x}}{\partial t}$$

$$\mathbf{n} = \frac{\mathbf{n}^*}{\left|\mathbf{n}^*\right|}$$



Bezier Surface Properties

- Like Bezier curves, Bezier surfaces retain the convex hull property, so that any point on the actual surface will fall within the convex hull of the control points
- With Bezier curves, the curve will interpolate (pass through) the first and last control points, but will only approximate the other control points
- With Bezier surfaces, the 4 corners will interpolate, and the other 12 points in the control mesh are only approximated
- The 4 boundaries of the Bezier surface are just Bezier curves defined by the points on the edges of the surface
- By matching these points, two Bezier surfaces can be connected precisely

B-Spline Curves

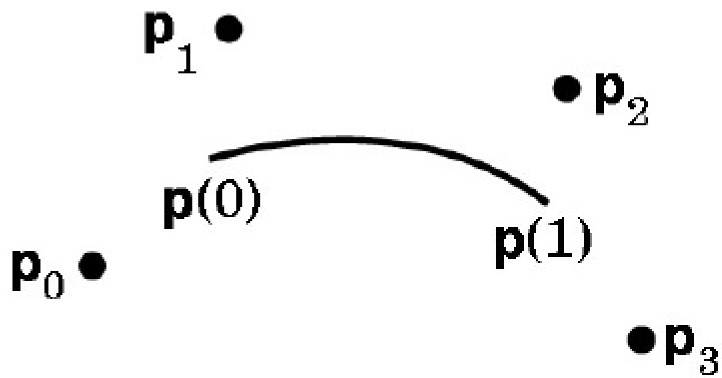
- Cubic polynomial form:

$$p(k) = \sum_{i=0}^{3} c_i k^i$$

- Given control point p_0 p_1 p_2 p_3, solve for c_k
- Cubic Bezier: interpolate 2 end points and approximate the two others.
- B-Spline: approximate more control points than Bezier does.

B-Spline Curves

- Need m+2 points to approximate m points
- For example, it utilizes 4 points to approximate two points



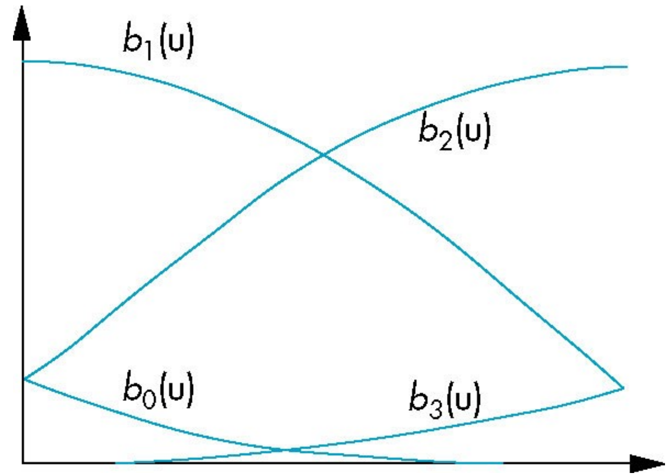
- Draw curve with overlapping control points : 0-1-2-3, 1-2-3-4, 2-3-4-5, 3-4-5-6, etc.

B-Splines

- Basis splines: use the data at $\mathbf{p}=[p_{i-2}\ p_{i-1}\ p_i\ p_{i-1}]^T$ to define curve only between p_{i-1} and p_i
- Allows us to apply more continuity conditions to each segment
- For cubic B-Spline, we can have continuity of function, first and second derivatives at join points
- Cost is 3 times as much work for curves
  - Add one new point each time rather than three
- For surfaces, we do 9 times as much work

Blending Functions

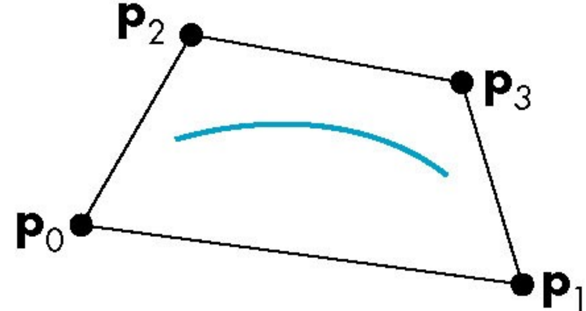
$$\mathbf{b}(u) = \frac{1}{6}\begin{bmatrix} (1-u)^3 \\ 4-6u^2+3u^3 \\ 1+3u+3u^2-3u^2 \\ u^3 \end{bmatrix}$$



$$x(t) = \sum_{i=0}^{3} P_i b_i(t)$$

$$= P_0 \frac{1}{6}\left(1-3t+3t^2-t^3\right) + P_1 \frac{1}{6}\left(4-6t^2+3t^3\right) + P_2 \frac{1}{6}\left(1+3t+3t^2-3t^3\right) + P_3 \frac{1}{6}\left(t^3\right)$$

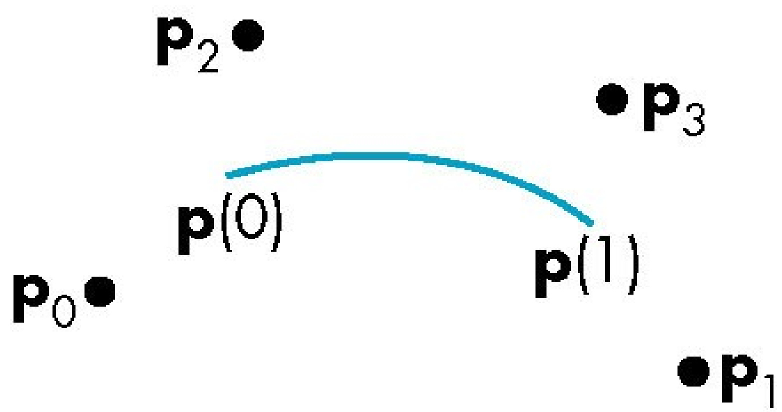
convex hull property



Cubic B-spline

$$p(u) = \mathbf{u}^T \mathbf{M}_S \mathbf{p} = \mathbf{b(}u\mathbf{)}^T \mathbf{p}$$

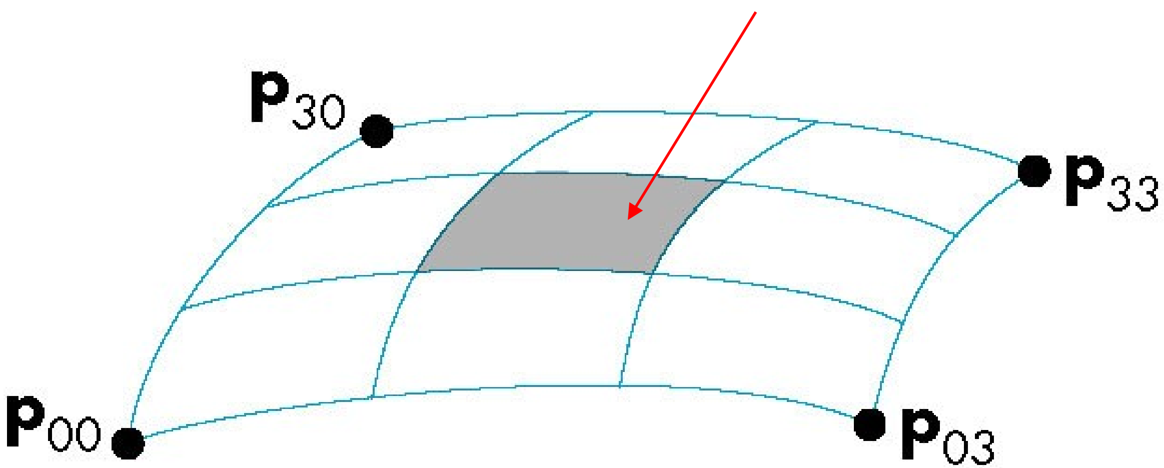
$$\mathbf{M}_S = \frac{1}{6}\begin{bmatrix} 1 & 4 & 1 & 0 \\ -3 & 0 & 3 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{bmatrix}$$



B-Spline Patches

$$p(u,v)=\sum_{i=0}^{3}\sum_{j=0}^{3} b_i(u)\,b_j(v)\,p_{ij}=u^T\,\mathbf{M}_S\,\mathbf{P}\,\mathbf{M}_S^T\,v$$

defined over only 1/9 of region



Splines and Basis

- If we examine the cubic B-spline from the perspective of each control (data) point, each interior point contributes (through the blending functions) to four segments
- We can rewrite p(u) in terms of the data points as

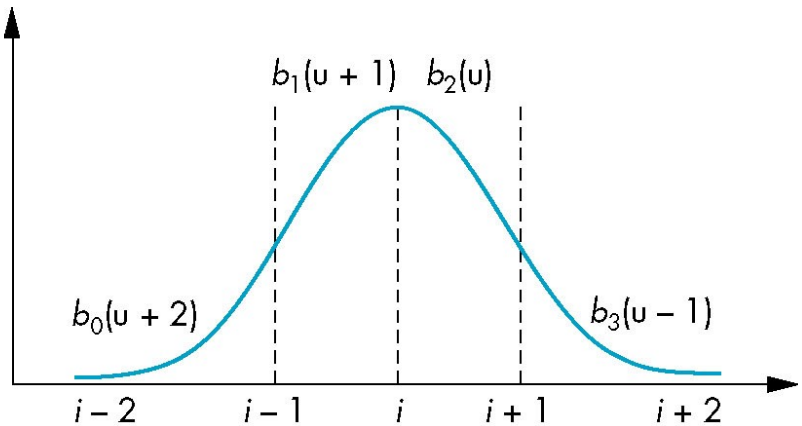
$$p(u) = \sum B_i(u)\, p_i$$

defining the basis functions $\{\mathrm{B_i(u)}\}$

Basis Functions

In terms of the blending polynomials

$$B_i(u) = \begin{cases} 0 & u < i-2 \\ b_0(u+2) & i-2 \le u < i-1 \\ b_1(u+1) & i-1 \le u < i \\ b_2(u) & i \le u < i+1 \\ b_3(u-1) & i+1 \le u < i+2 \\ 0 & u \ge i+2 \end{cases}$$



Generalizing Splines

- We can extend to any degree of Splines
- Data and conditions do not have to given at equally spaced values (the *knots*)
  - Nonuniform and uniform splines
  - Can have repeated knots
- Cox-de Boor recursion gives method of evaluation

$$b_{j,0}(t) = \begin{cases} 1, & \text{if } t_j \le t \le t_{j+1} \\ 0, & \text{otherwise} \end{cases}, j = 0, \ldots, m-2$$

$$b_{j,n}(t) = \frac{t - t_j}{t_{j+n} - t_j} b_{j,n-1}(t) + \frac{t_{j+n+1} - t}{t_{j+n+1} - t_{j+1}} b_{j+1,n-1}(t), j = 0, \ldots, m-n-2$$

References

[1] Foley, Van Dam, Feiner, Hughes, Computer Graphics - Principles and Practices 2nd Ed. In C, Addison Wesley, 1997.

[2] K. S. Fant, CG-Course Slide, Portland State University.

[3] D. James, CG-Course Slide, Carnegie Mellon University.

[4] D. Breen, et.al., CG-Course Slide, Drexel University.

[5] E. Catmull, J. Clark, Recursively generated B-Spline surfaces on abitrary topological meshes, Computer Aided Design, Vol. 10(6), 1978, pp. 350-354.