Bezier Surfaces

Steve Rotenberg
CSE168: Rendering Algorithms
UCSD, Spring 2014

Curved Surfaces

- There are a variety of curved surface categories used in computer graphics:
  - Parametric (explicit) surfaces
  - Subdivision surfaces
  - Implicit surfaces

Parametric Surface

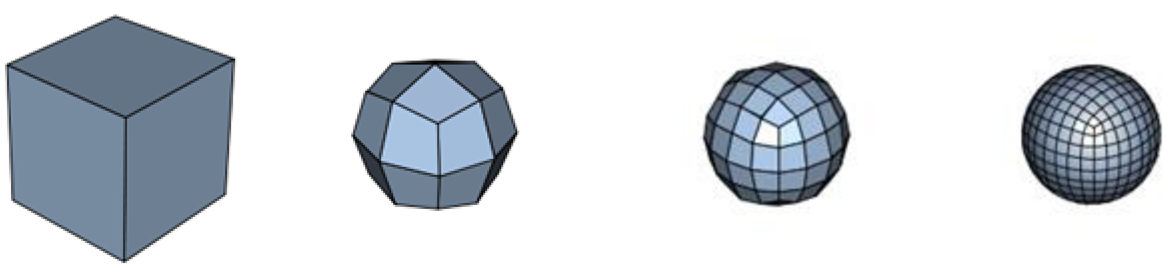
- A *parametric surface* (or *explicit surface*) is a surface that is explicitly defined mathematically as a function of two parameters:

$$\mathbf{x} = \mathbf{f}(s, t)$$

- For some domain of *s* & *t* (typically from 0…1)
- Parametric surfaces have a rectangular topology
- Popular surfaces of this type include:
  - Bezier surfaces
  - B-splines
  - NURBS (non-uniform rational B-splines)

Subdivision Surfaces

- *Subdivision surfaces* are a category of curved surfaces that are constructed by applying repeated smoothing operations on a polygonal mesh
- There are a variety of mathematical formulations for these such as:
  - Catmull-Clark
  - Loop
  - sqrt(3)
  - Butterfly



Implicit Surfaces

- *Implicit surfaces* are non-parametric surfaces that are implicitly defined…
- For example, we could start with some geometric object and implicitly define a surface as the set of points that are exactly 1 unit away from the original object
- Another example could be the set of points where some 3D function equals zero

Bezier Curves

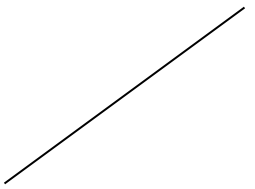
Bezier

- Pierre Bézier (1910-1999) was a French engineer who made many early contributions to computational geometry and CAD modeling systems
- Much of his work was done while at Renault, where he worked for 42 years (1933-1975)
- He developed much of the mathematical theory and notation of Bezier curves, although the original algorithm was invented by Paul de Casteljau 1959, while working at Citroën

Polynomial Functions

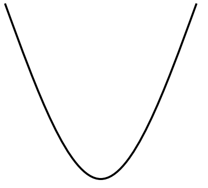
- Linear:

$$f(t) = at + b$$



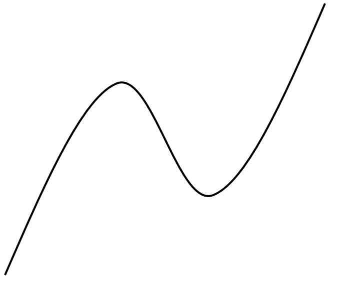
- Quadratic:

$$f(t) = at^2 + bt + c$$



- Cubic:

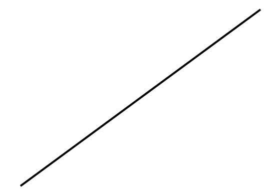
$$f(t) = at^3 + bt^2 + ct + d$$



Vector Polynomials (Curves)

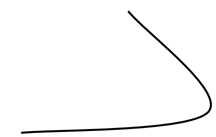
- Linear:

$$\mathbf{f}(t) = \mathbf{a}t + \mathbf{b}$$



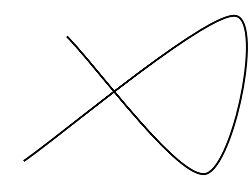
- Quadratic:

$$\mathbf{f}(t) = \mathbf{a}t^2 + \mathbf{b}t + \mathbf{c}$$



- Cubic:

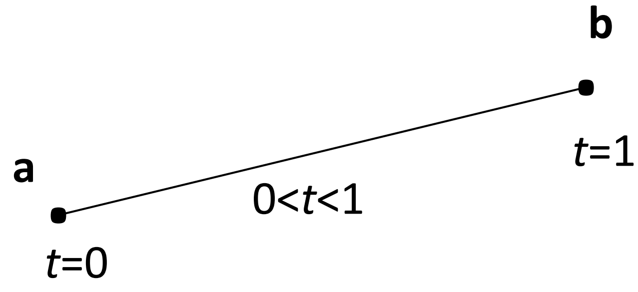
$$\mathbf{f}(t) = \mathbf{a}t^3 + \mathbf{b}t^2 + \mathbf{c}t + \mathbf{d}$$



We usually define the curve for $0 \le t \le 1$

Linear Interpolation

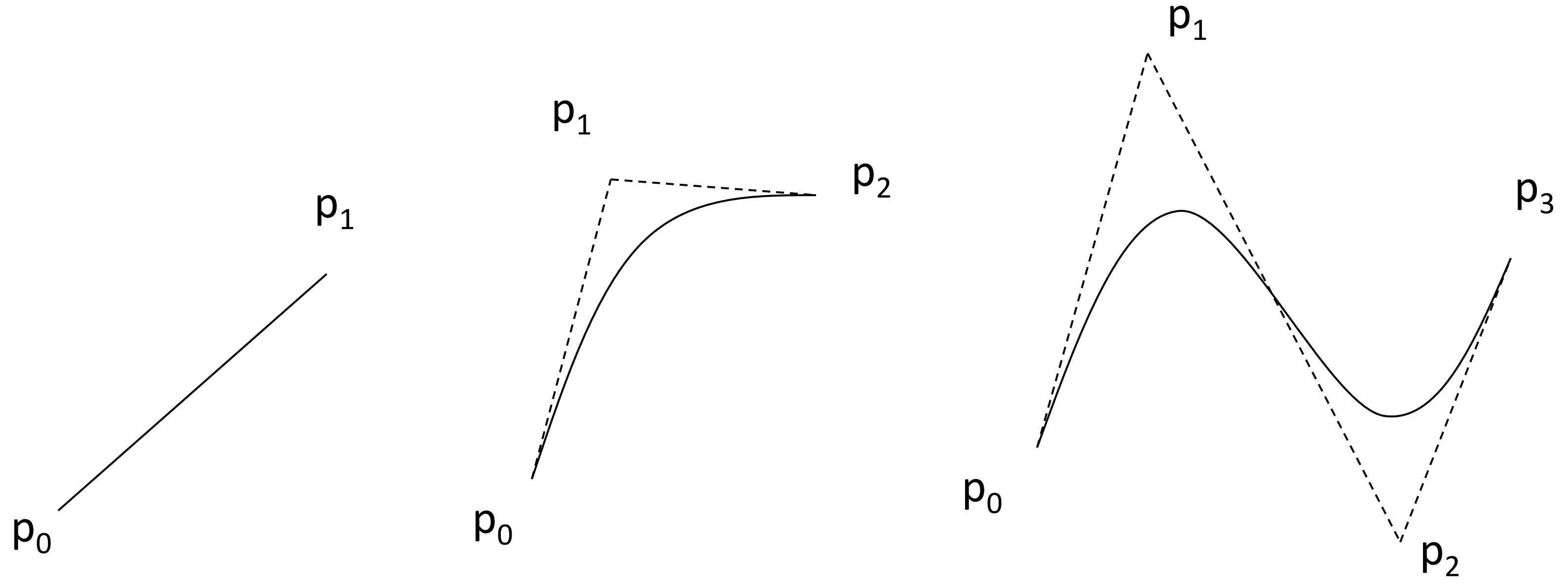
- Linear interpolation (Lerp) is a common technique for generating a new value that is somewhere in between two other values
- A ‘value’ could be a number, vector, color, or even something more complex like an entire 3D object…
- Consider interpolating between two points **a** and **b** by some parameter *t*



$$Lerp(t, \mathbf{a}, \mathbf{b}) = (1-t)\mathbf{a} + t\mathbf{b}$$

Bezier Curves

- Bezier curves can be thought of as a higher order extension of linear interpolation

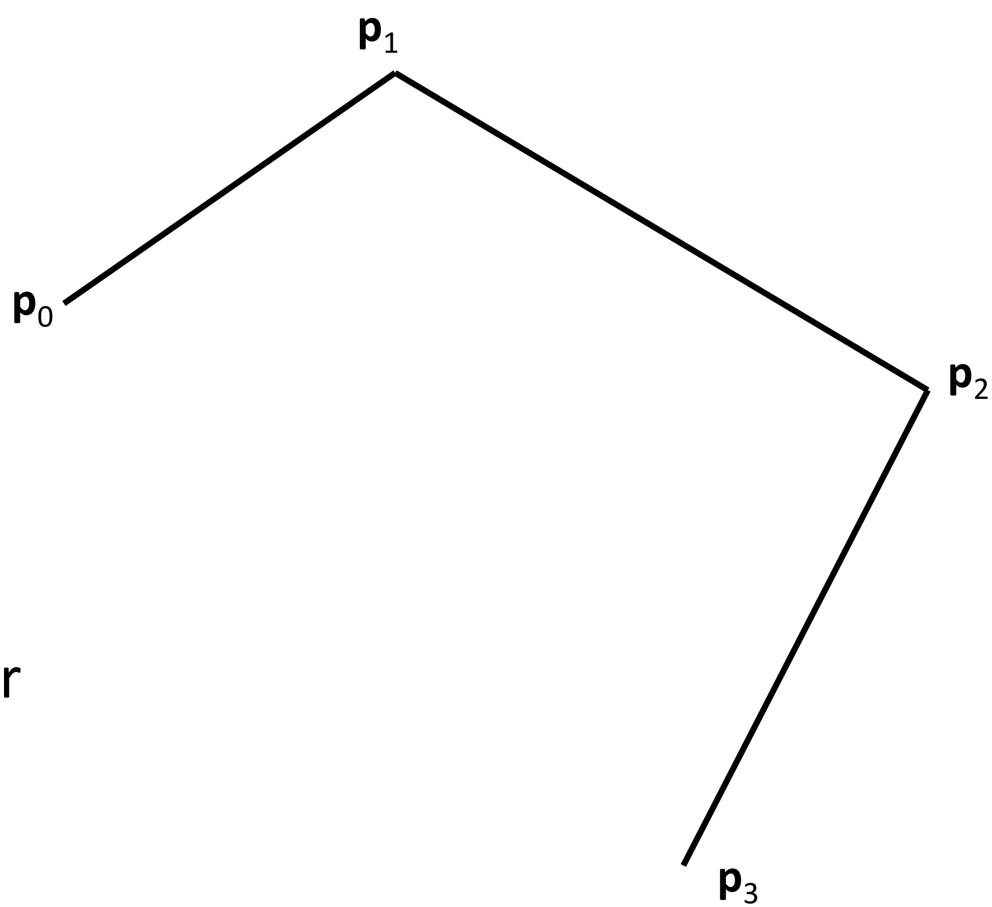


Linear Quadratic Cubic

de Casteljau Algorithm

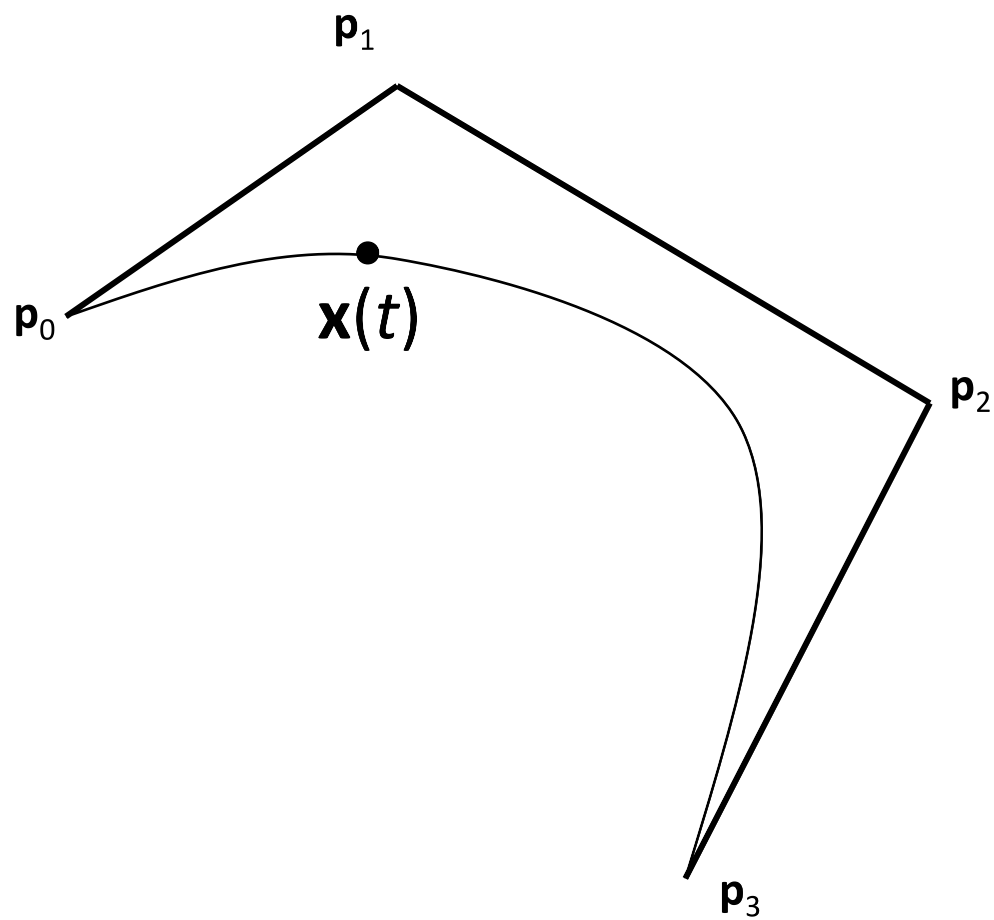
- The de Casteljau algorithm describes the curve as a recursive series of linear interpolations
- This form is useful for providing an intuitive understanding of the geometry involved, and is very numerically stable, but it is not the most efficient form

de Casteljau Algorithm



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

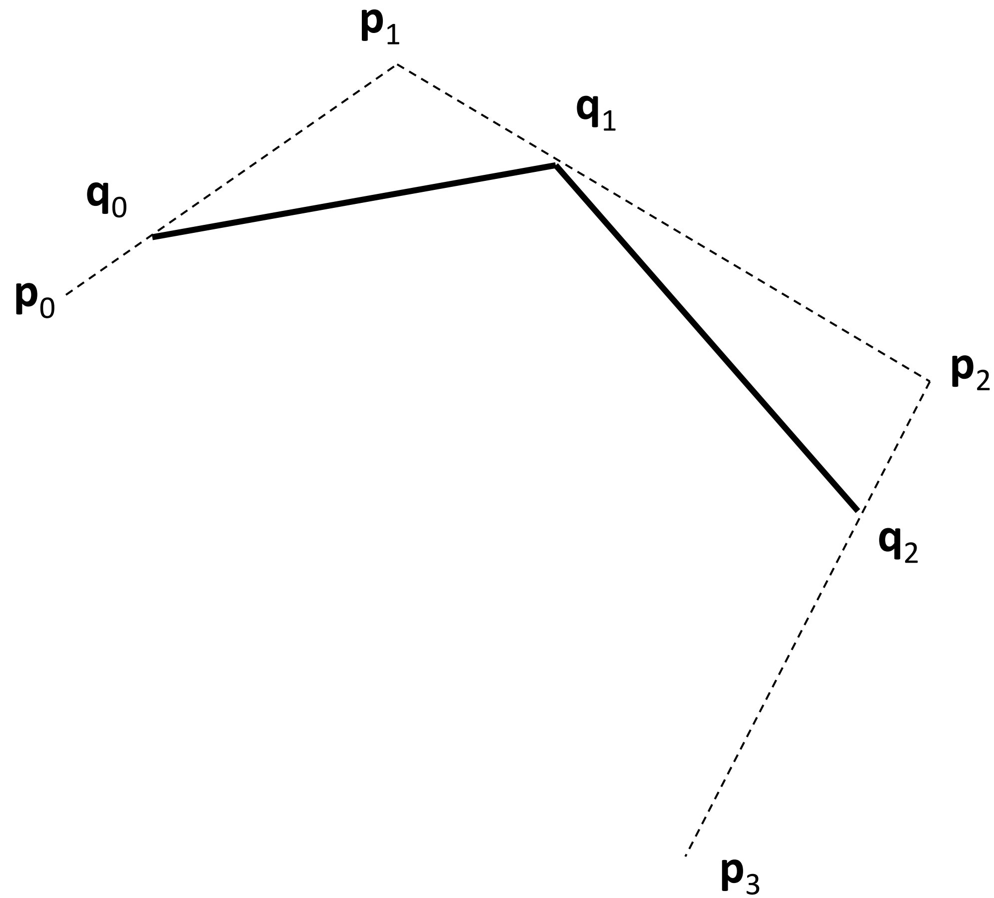
de Casteljau Algorithm



- We want to find the point $\mathbf{x}$ on the curve as a function of parameter t

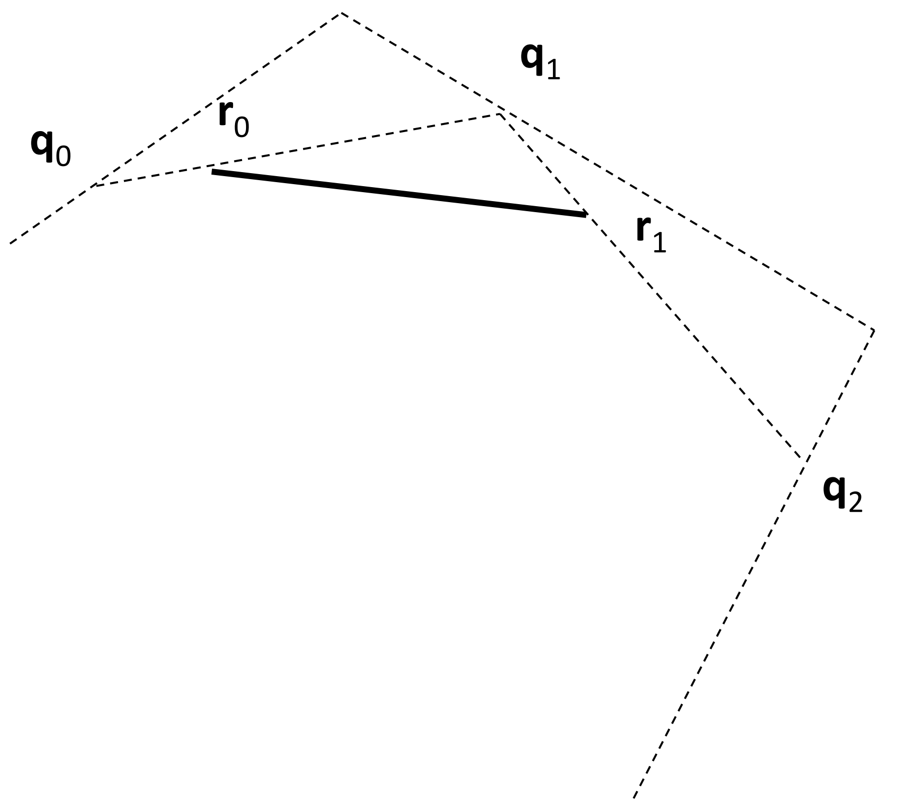
de Casteljau Algorithm

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

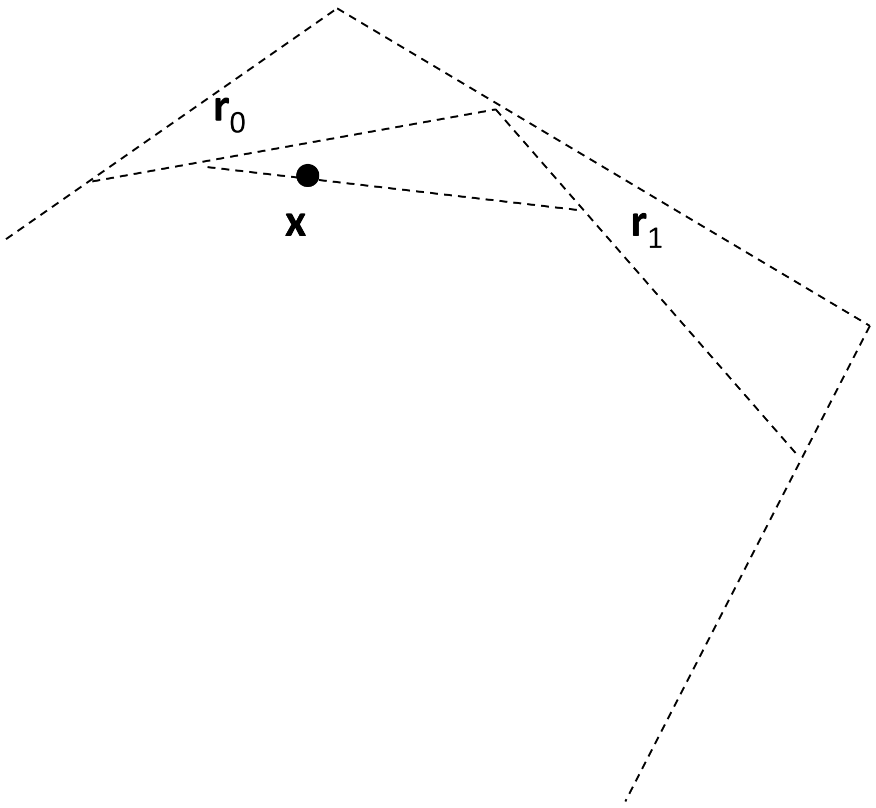


de Casteljau Algorithm

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

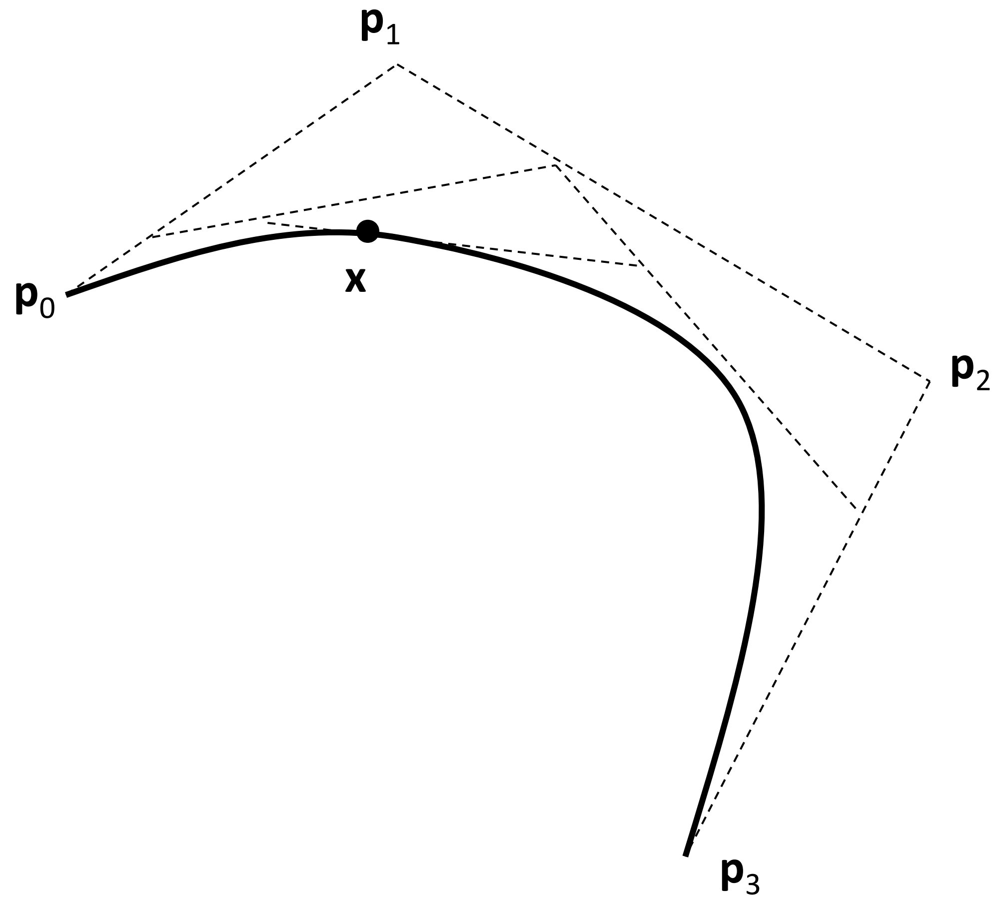


de Casteljau Algorithm



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

Bezier Curve



Recursive Linear Interpolation

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

$$\mathbf{r}_0 = Lerp\left(t, \mathbf{q}_0, \mathbf{q}_1\right)$$
$$\mathbf{r}_1 = Lerp\left(t, \mathbf{q}_1, \mathbf{q}_2\right)$$

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

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

$$\begin{tikzcd}
 & & & \mathbf{p}_0 \\
 & & \mathbf{q}_0 \arrow[ur, dash] \arrow[dr, dash] & \\
 & \mathbf{r}_0 \arrow[ur, dash] \arrow[dr, dash] & & \mathbf{p}_1 \\
\mathbf{x} \arrow[ur, dash] \arrow[dr, dash] & & \mathbf{q}_1 \arrow[ur, dash] \arrow[dr, dash] & \\
 & \mathbf{r}_1 \arrow[ur, dash] \arrow[dr, dash] & & \mathbf{p}_2 \\
 & & \mathbf{q}_2 \arrow[ur, dash] \arrow[dr, dash] & \\
 & & & \mathbf{p}_3
\end{tikzcd}$$

Expanding the Lerps

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

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

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

Cubic Equation Form

$$\mathbf{x} = (1-t)\big((1-t)((1-t)\mathbf{p}_0 + t\mathbf{p}_1) + t((1-t)\mathbf{p}_1 + t\mathbf{p}_2)\big)$$
$$+ t\big((1-t)((1-t)\mathbf{p}_1 + t\mathbf{p}_2) + t((1-t)\mathbf{p}_2 + t\mathbf{p}_3)\big)$$

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

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

Cubic Equation Form

- If we regroup the equation by terms of exponents of *t*, we get it in the standard cubic form
- This form is very good for fast evaluation, as all of the constant terms (**a**,**b**,**c**,**d**) can be precomputed
- The cubic equation form obscures the input geometry ($\mathbf{p}_0$,$\mathbf{p}_1$,$\mathbf{p}_2$,$\mathbf{p}_3$), but there is a one-to-one mapping between the two and so the geometry can always be extracted out of the cubic coefficients

Matrix Form

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

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

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

$$\mathbf{c} = \left(-3\mathbf{p}_0 + 3\mathbf{p}_1\right)$$

$$\mathbf{d} = \left(\mathbf{p}_0\right)$$

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

Matrix Form

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

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

Matrix Form

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

$$\boxed{\begin{aligned} \mathbf{x} &= \mathbf{t} \cdot \mathbf{B}_{Bez} \cdot \mathbf{G}_{Bez} \\ \mathbf{x} &= \mathbf{t} \cdot \mathbf{C} \end{aligned}}$$

Matrix Form

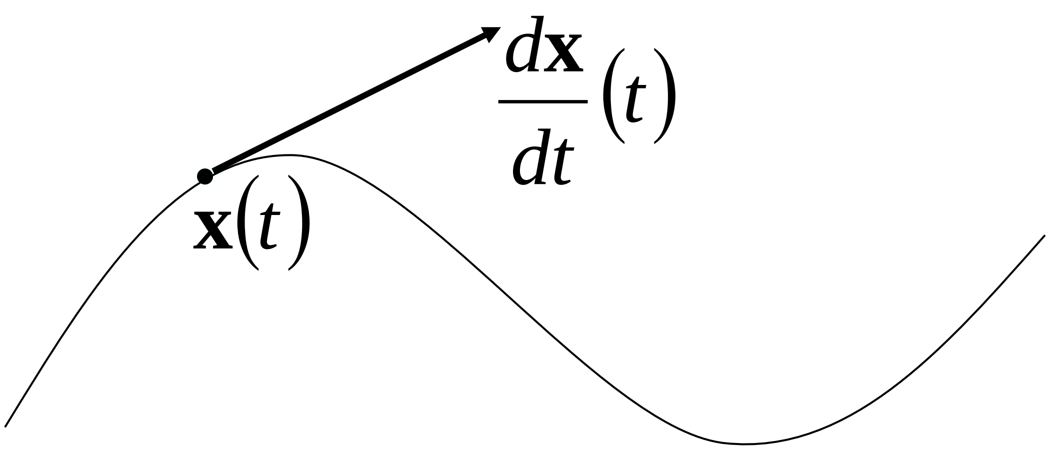
- We can rewrite the equations in matrix form
- This gives us a compact notation and shows how different forms of cubic curves can be related
- It also is a very efficient form as it can take advantage of existing 4x4 matrix hardware support…

Bezier Curves & Cubic Curves

- By adjusting the 4 control points of a cubic Bezier curve, we can represent any cubic curve
- Likewise, any cubic curve can be represented uniquely by a cubic Bezier curve
- There is a one-to-one mapping between the 4 Bezier control points ($\mathbf{p}_0,\mathbf{p}_1,\mathbf{p}_2,\mathbf{p}_3$) and the pure cubic coefficients ($\mathbf{a},\mathbf{b},\mathbf{c},\mathbf{d}$)
- The Bezier basis matrix $\mathbf{B}_{Bez}$ (and it's inverse) perform this mapping
- There are other common forms of cubic curves that also retain this property (Hermite, Catmull-Rom, B-Spline)

Tangents

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



Derivatives

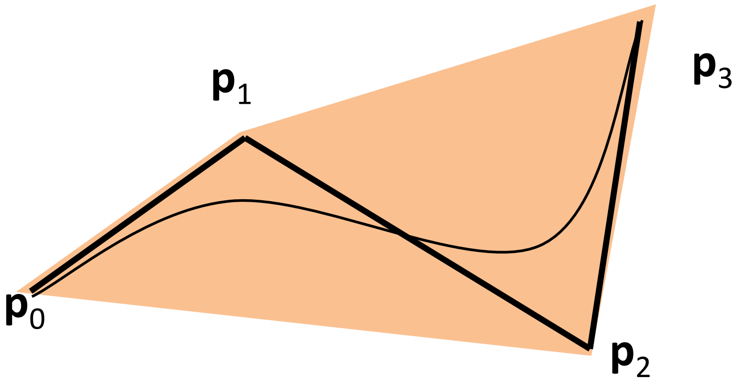
- Finding the derivative (tangent) of a curve is easy:

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

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

Convex Hull Property

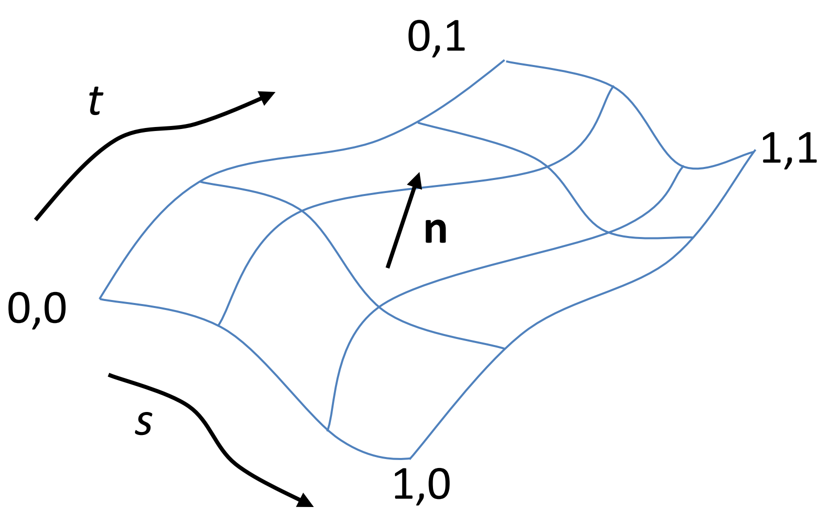
- If we take all of the control points for a Bezier curve and construct a convex polygon around them, we have the *convex hull* of the curve
- An important property of Bezier curves is that every point on the curve itself will be somewhere within the convex hull of the control points



Bezier Surfaces

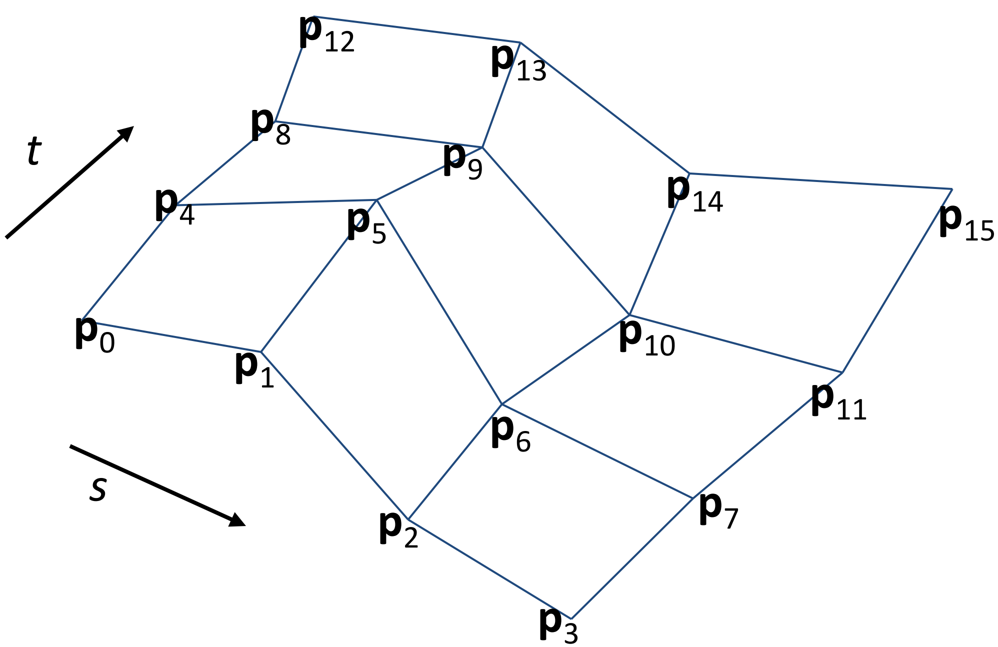
Bezier Surfaces

- 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 the corresponding *t*-tangent will represent the normal for the front of the surface at that location



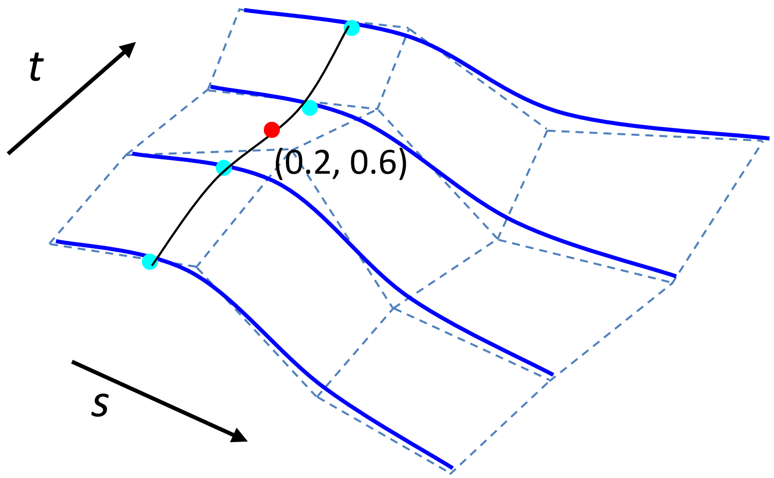
Control Mesh

- 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 equivalently 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

- We saw the matrix form for a 3D Bezier curve is

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

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

$$\mathbf{x} = \mathbf{t} \cdot \mathbf{C}$$

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:

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

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

Matrix Form

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

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

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

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

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

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

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 is a nice and compact notation and 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:

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

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

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

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

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

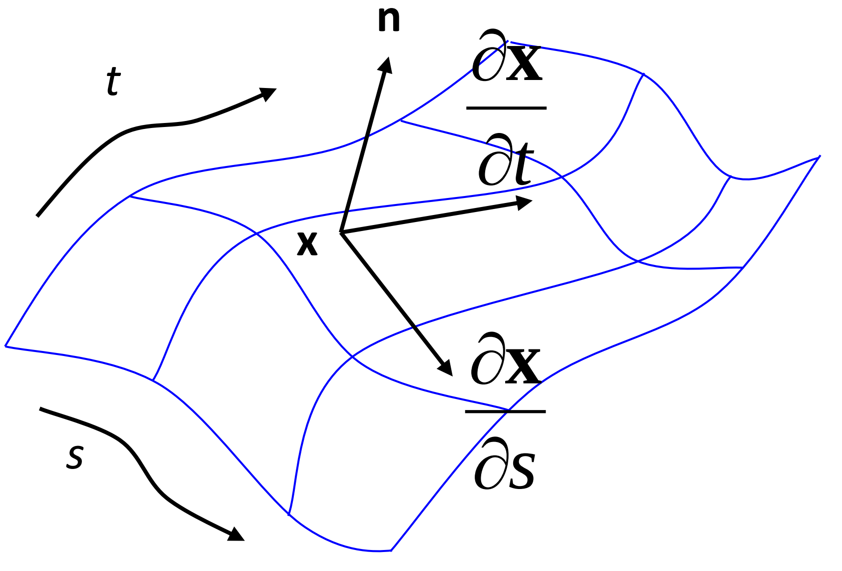
$$d\mathbf{t} = \begin{bmatrix} 3t^2 & 2t & 1 & 0 \end{bmatrix}$$

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
- Usually, it is normalized as well

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

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



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 each edge of the surface
- By matching these points, two Bezier surfaces can be connected precisely

Tessellation

- *Tessellation* is the process of triangulating a curved surface
- If we triangulate the surface before rendering, we can just insert those triangles into a spatial data structure and render as usual
- This is usually sufficient, as surfaces can be automatically tessellated to triangles that are the size of a single pixel or even smaller
- We will look at tessellation and displacement mapping of curved surfaces in the next lecture