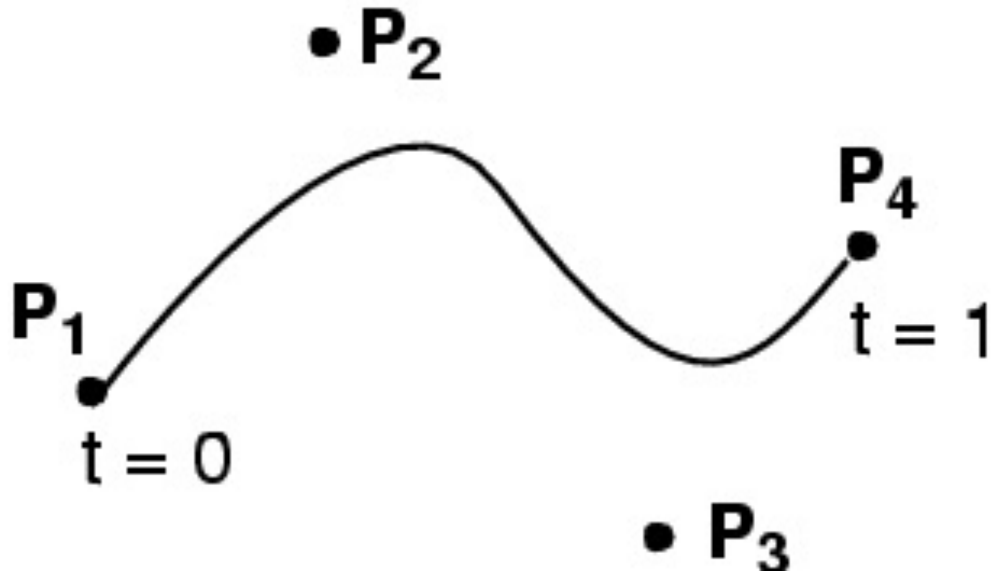
## Advanced Digital Design

# Curve Properties & Conversion, Surface Representations

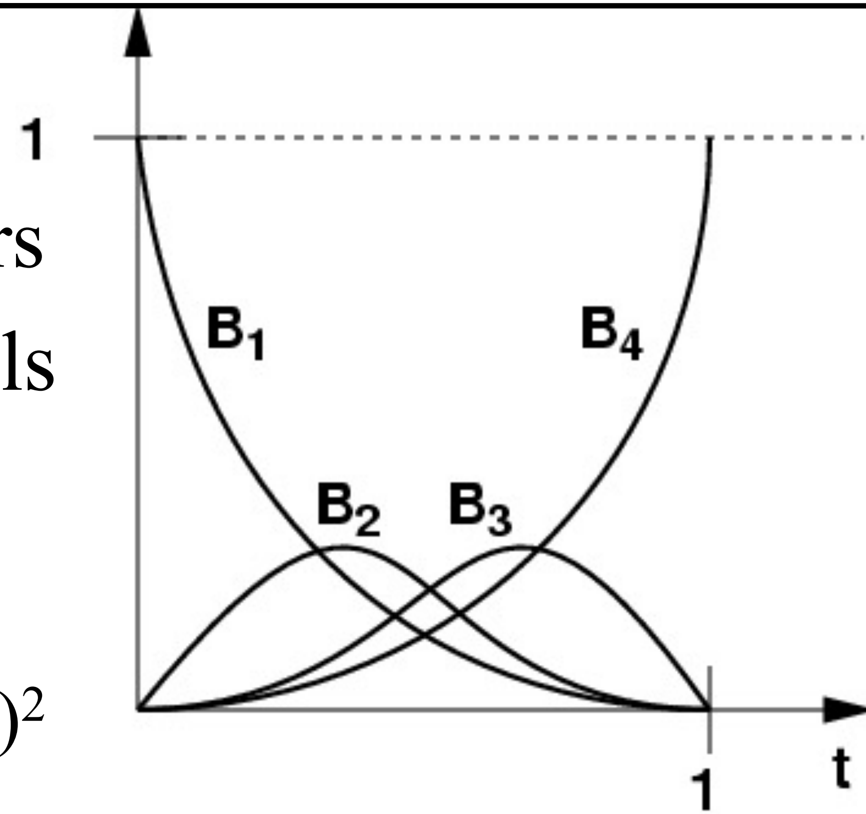
Sai-Kit Yeung
HKUST ISD and CSE

# Cubic Bezier Splines

- $$\begin{aligned} P(t) = \quad & (1-t)^3 && P_1 \\ + \quad & 3t(1-t)^2 && P_2 \\ + \quad & 3t^2(1-t) && P_3 \\ + \quad & t^3 && P_4 \end{aligned}$$



# Bernstein Polynomials




- For Bézier curves, the basis polynomials/vectors are Bernstein polynomials

- For cubic Bezier curve:

  $B_1(t)=(1-t)^3$ $\quad$ $B_2(t)=3t(1-t)^2$

  $B_3(t)=3t^2(1-t)$ $\quad$ $B_4(t)=t^3$

  (careful with indices, many authors start at 0)

- Defined for any degree

# General Spline Formulation

$$Q(t) = \mathbf{GBT(t)} = \text{Geometry } \mathbf{G} \cdot \text{Spline Basis } \mathbf{B} \cdot \text{Power Basis } \mathbf{T(t)}$$

- Geometry: control points coordinates assembled into a matrix ($P_1$, $P_2$, …, $P_{n+1}$)
- Power basis: the monomials 1, t, $t^2$, ...
- Cubic Bézier:

$$\mathrm{P(t)} = \begin{pmatrix} x(t) \\ y(t) \end{pmatrix} =$$

$$\begin{pmatrix} x_1 & x_2 & x_3 & x_4 \\ y_1 & y_2 & y_3 & y_4 \end{pmatrix} \begin{pmatrix} 1 & -3 & 3 & -1 \\ 0 & 3 & -6 & 3 \\ 0 & 0 & 3 & -3 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ t \\ t^2 \\ t^3 \end{pmatrix}$$

# General Spline Formulation

$$Q(t) = \mathbf{GBT(t)} = \text{Geometry } \mathbf{G} \cdot \text{Spline Basis } \mathbf{B} \cdot \text{Power Basis } \mathbf{T(t)}$$

- Geometry: control points coordinates assembled into a matrix ($P_1$, $P_2$, …, $P_{n+1}$)
- Power basis: the monomials 1, t, $t^2$, ...
- Cubic Bézier:

$$\mathrm{P}(\mathrm{t}) = \begin{pmatrix} x(t) \\ y(t) \end{pmatrix} =$$

$$\begin{pmatrix} x_1 & x_2 & x_3 & x_4 \\ y_1 & y_2 & y_3 & y_4 \end{pmatrix} \begin{pmatrix} 1 & -3 & 3 & -1 \\ 0 & 3 & -6 & 3 \\ 0 & 0 & 3 & -3 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ t \\ t^2 \\ t^3 \end{pmatrix}$$

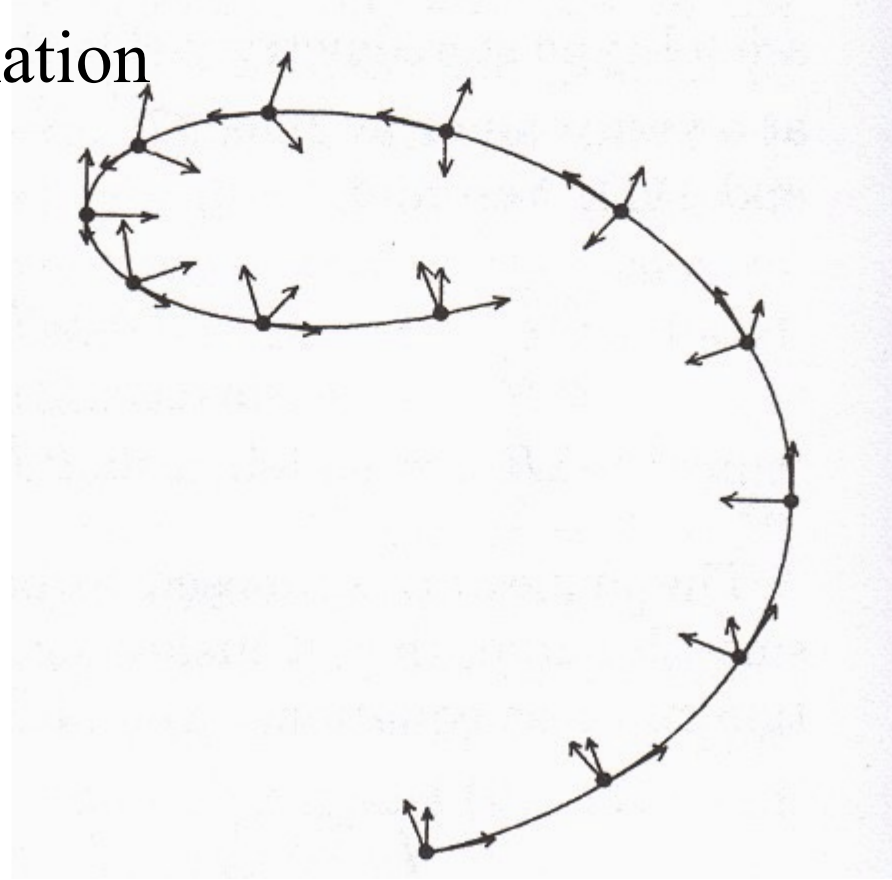
Questions?

# The Plan for Today

- Differential Properties of Curves & Continuity
- B-Splines
- Surfaces
  - Tensor Product Splines
  - Subdivision Surfaces
  - Procedural Surfaces
  - Other

# Differential Properties of Curves

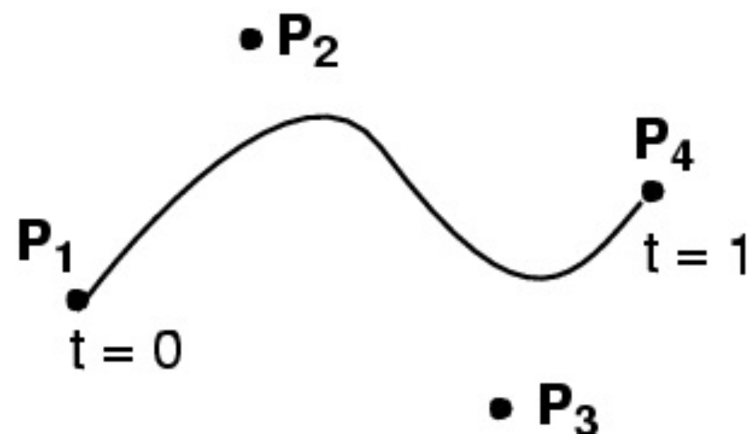
- Motivation
  - Compute normal for surfaces
  - Compute velocity for animation
  - Analyze smoothness



# Velocity

- First derivative w.r.t. t
- Can you compute this for Bezier curves?

$$P(t) = (1-t)^3 \, P_1 + 3t(1-t)^2 \, P_2 + 3t^2(1-t) \, P_3 + t^3 \, P_4$$

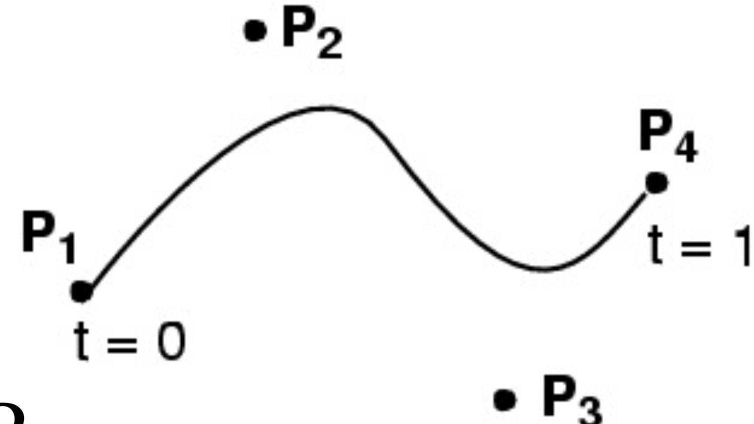



- You know how to differentiate polynomials...

# Velocity

- First derivative w.r.t. t
- Can you compute this for Bezier curves?

$$P(t) = (1-t)^3 P_1 + 3t(1-t)^2 P_2 + 3t^2(1-t) P_3 + t^3 P_4$$




- $$P'(t) = -3(1-t)^2 P_1 + [3(1-t)^2 - 6t(1-t)] P_2 + [6t(1-t) - 3t^2] P_3 + 3t^2 P_4$$

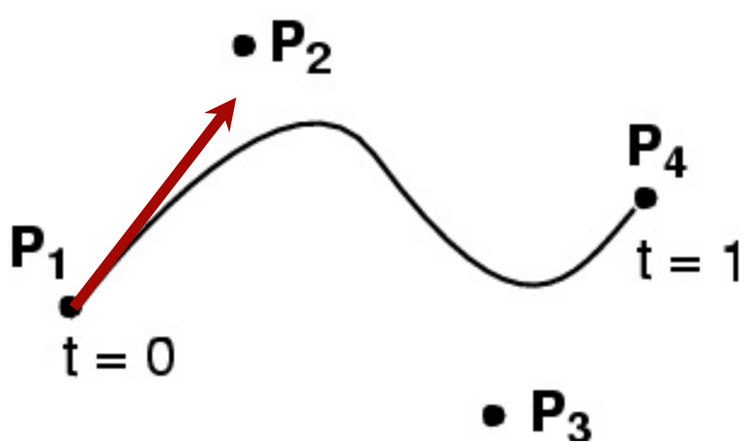
Sanity check: t=0; t=1

# Linearity?

- Differentiation is a linear operation
  - (f+g)'=f'+g'
  - (af)'=a f'
- This means that the derivative of the basis is enough to know the derivative of any spline with this basis.
  - Control points are just scalar
- Can be done with matrices
  - Trivial in monomial basis
  - But get lower-order polynomials

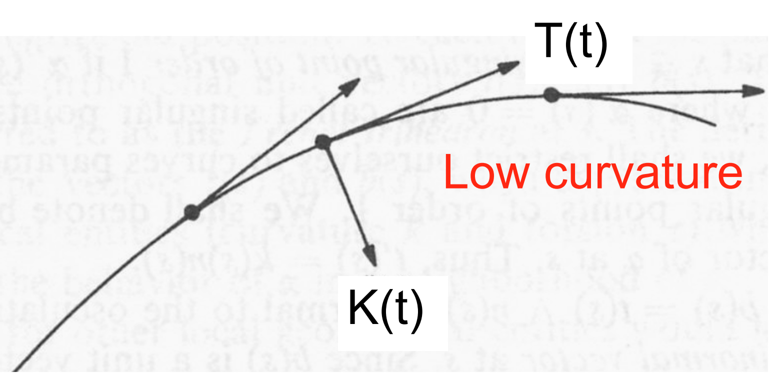
# Tangent Vector

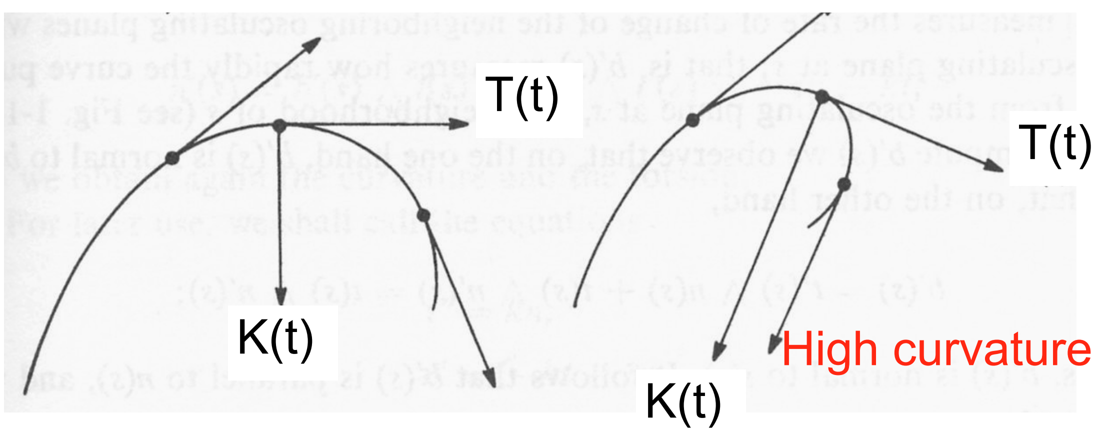
- The tangent to the curve P(t) can be defined as $T(t)=P'(t)/||P'(t)||$
  - normalized velocity, $||T(t)|| = 1$
- This provides us with one orientation for swept surfaces later



# Curvature Vector

- Derivative of unit tangent
  - Amount by which a geometric object deviates from being *straight*
  - Defined as K(t)=T'(t)
  - Magnitude ||K(t)|| is constant for a circle
  - Zero for a straight line
- Always orthogonal to tangent, ie. $K \cdot T = 0$

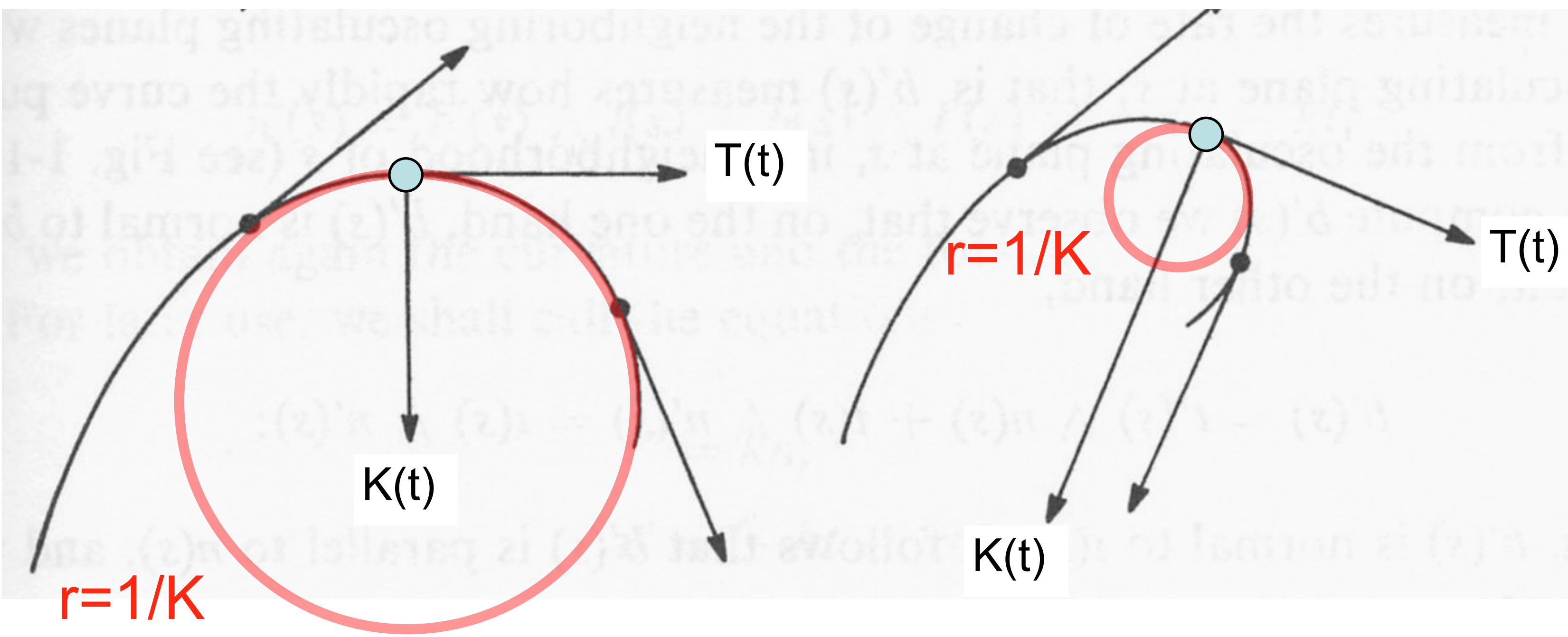


# Geometric Interpretation

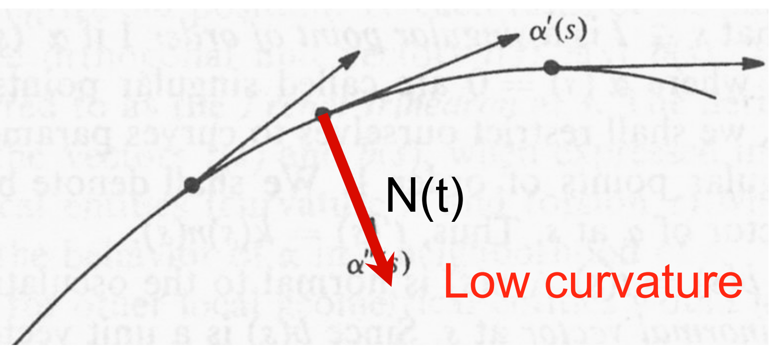
- $1/||K(t)||$ is the radius of the circle that touches $P(t)$ at t and has the same curvature as the curve

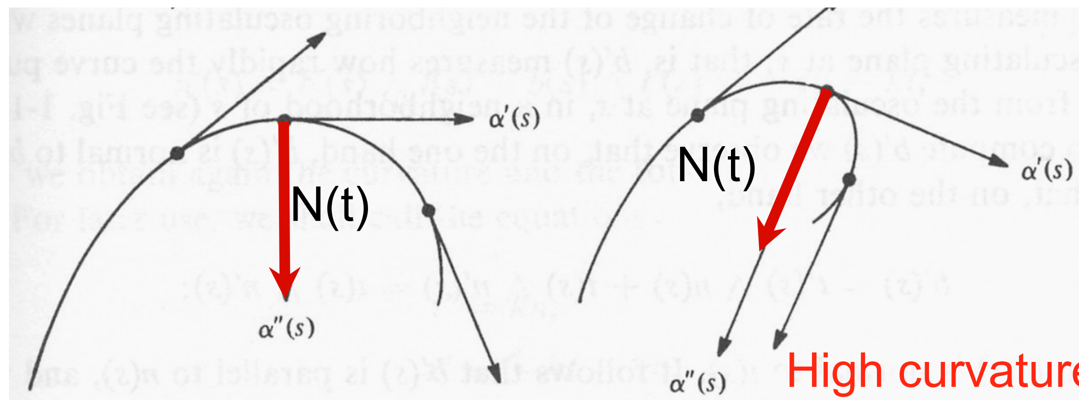
You may prove it



# Curve Normal

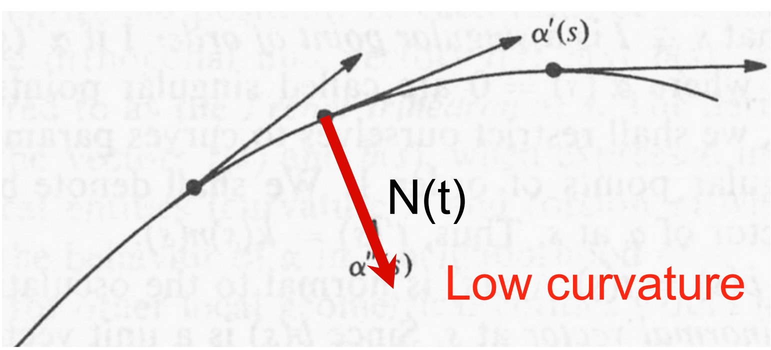
- Normalized curvature: T'(t)/||T'(t)||

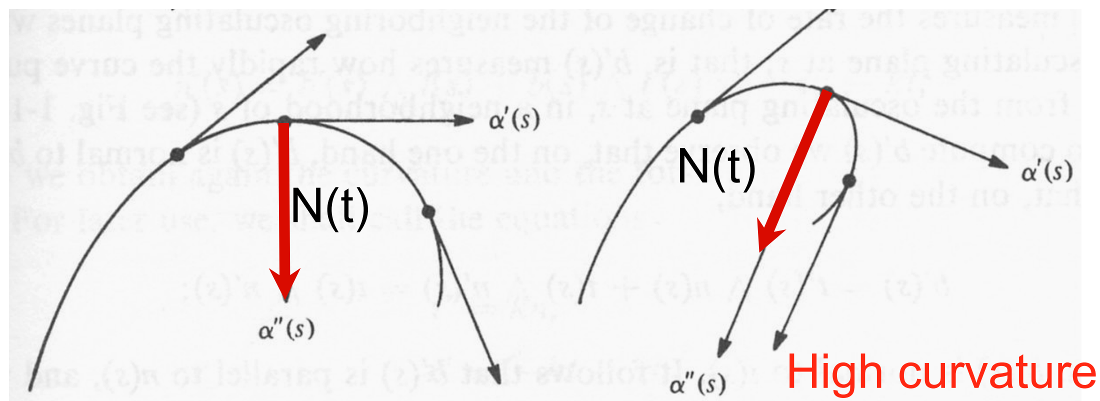


# Curve Normal

- Normalized curvature: T'(t)/||T'(t)||

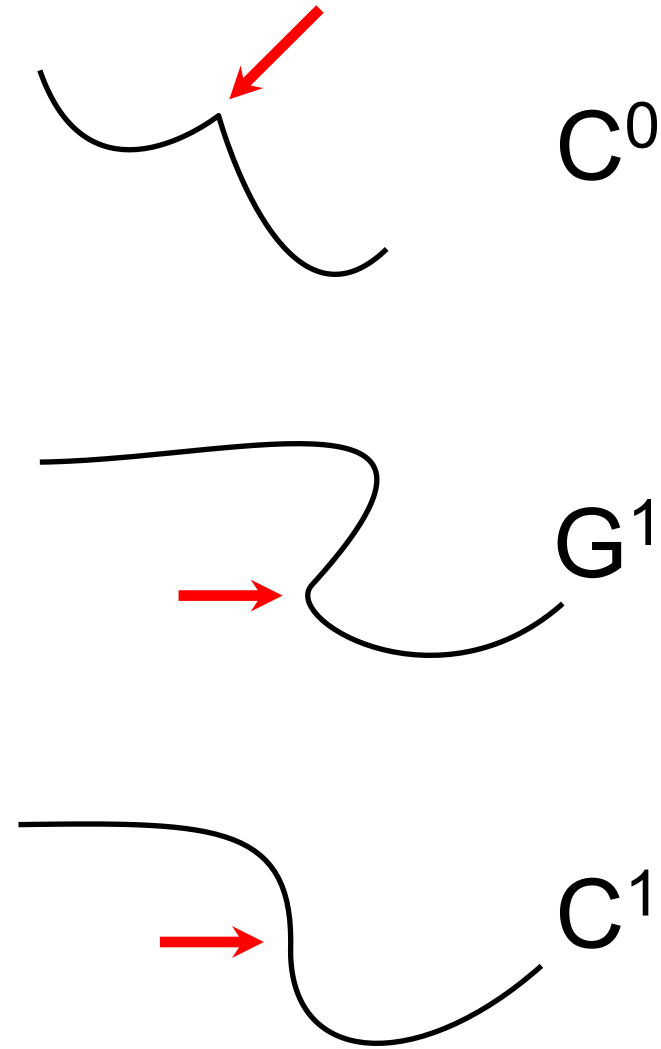



Questions?

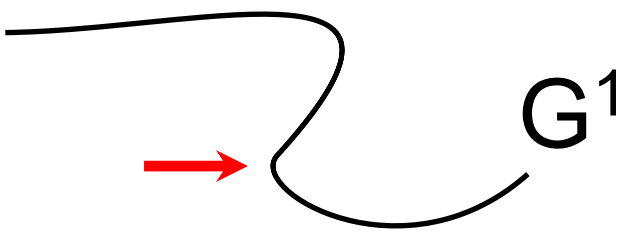
# Orders of Continuity

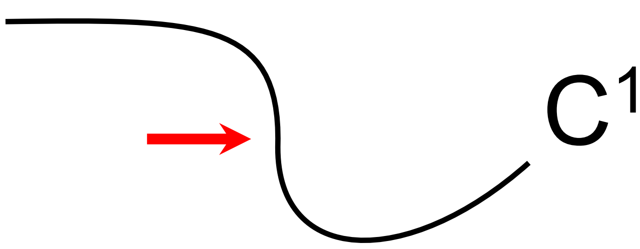
- $C^0$ = continuous
  - The seam can be a sharp kink
- $G^1$ = geometric continuity
  - Tangents point to the same direction at the seam
- $C^1$ = parametric continuity
  - Tangents are the same at the seam, implies $G^1$
- $C^2$ = curvature continuity
  - Tangents and their derivatives are the same



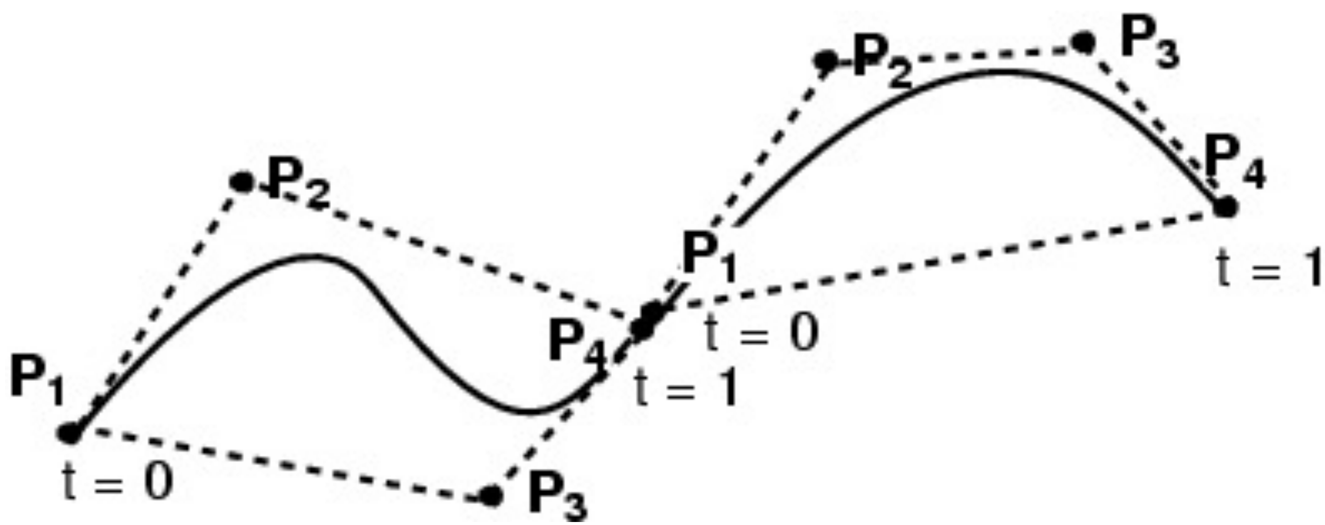
# Orders of Continuity

- $G^1$ = geometric continuity
  - Tangents point to the same direction at the seam
  - good enough for modeling
- $C^1$ = parametric continuity
  - Tangents are the same at the seam, implies $G^1$
  - often necessary for animation



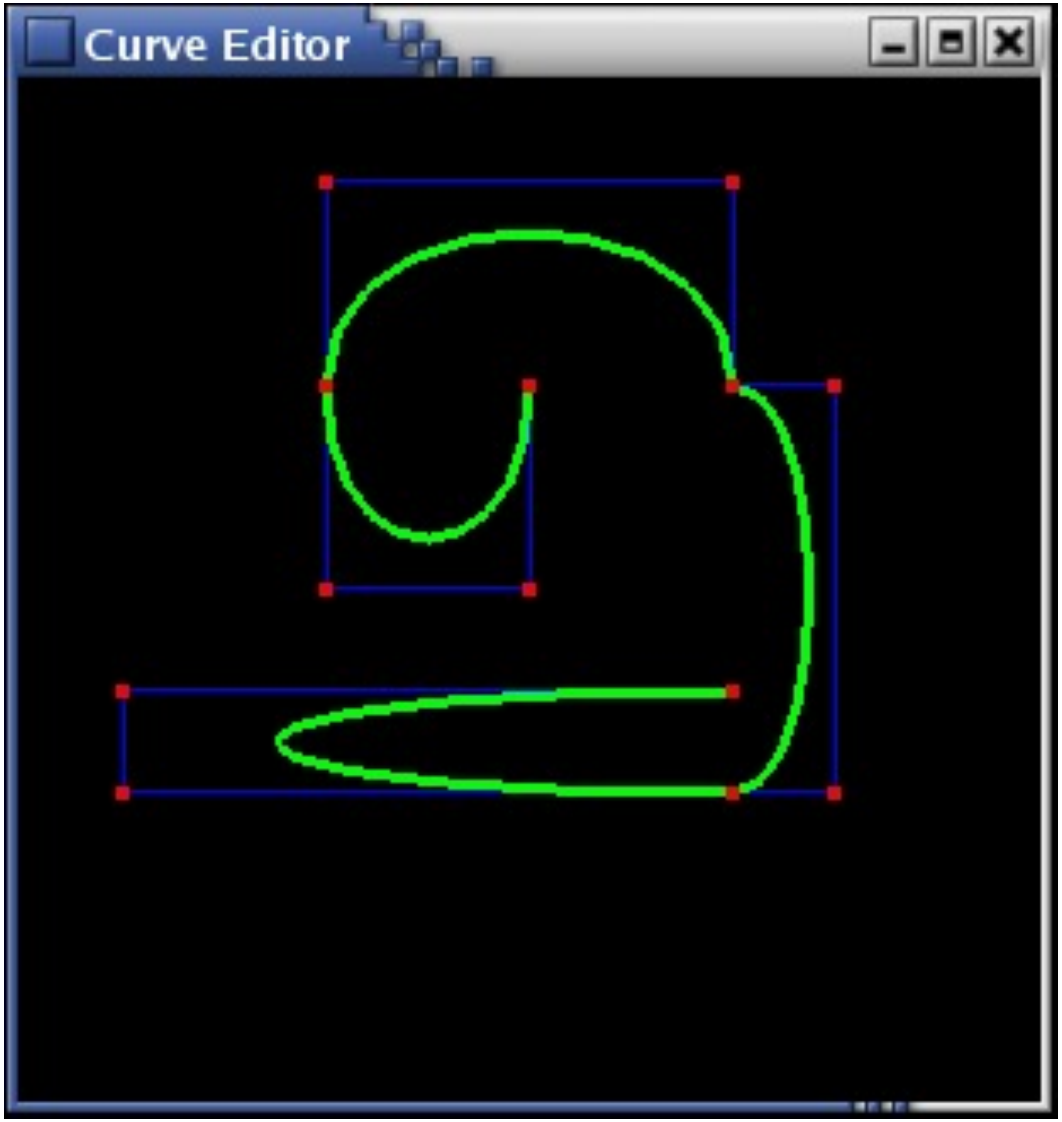



# Connecting Cubic Bézier Curves



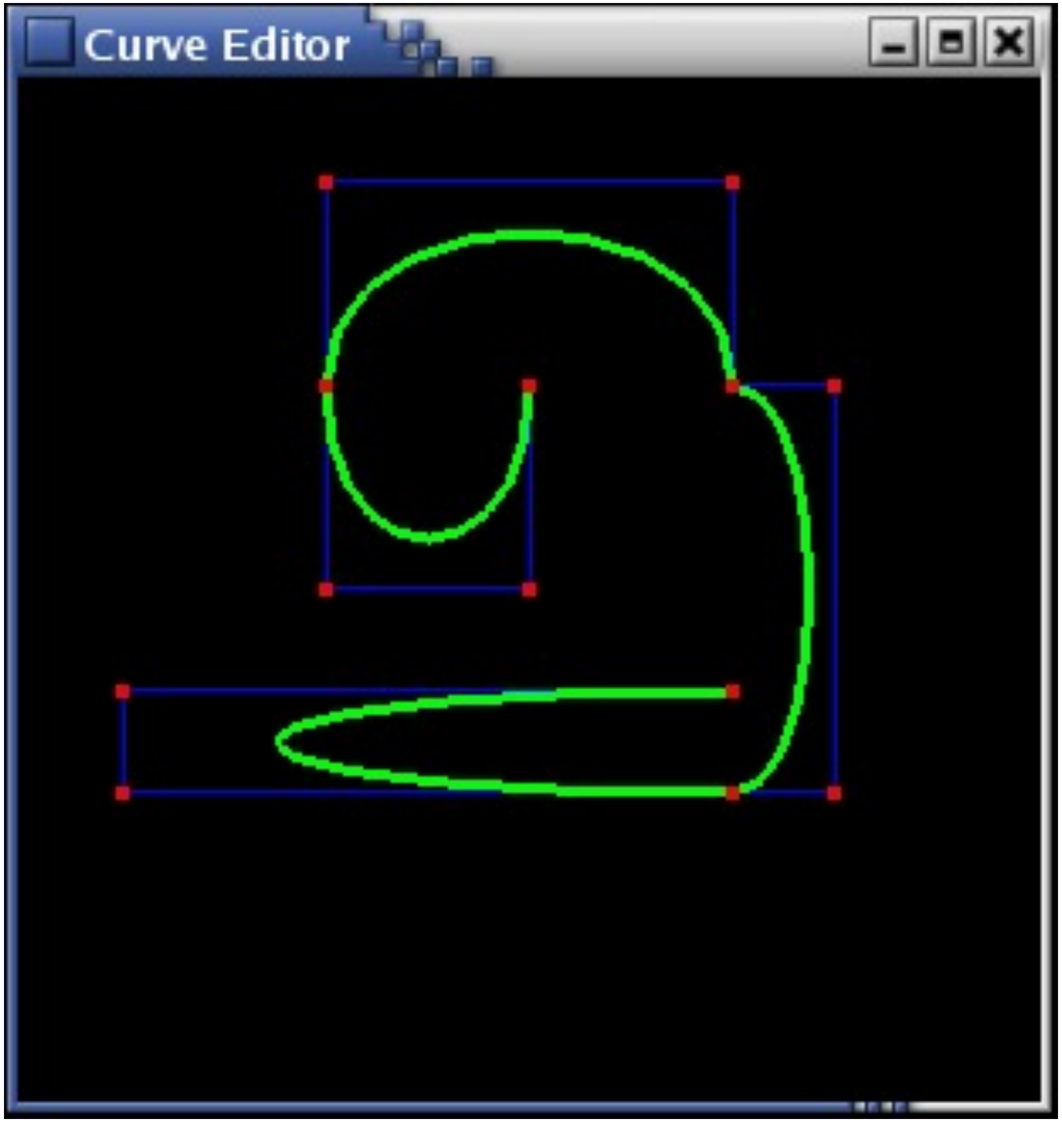

- How can we guarantee $C^0$ continuity?
- How can we guarantee $G^1$ continuity?
- How can we guarantee $C^1$ continuity?
- $C^2$ and above gets difficult

# Connecting Cubic Bézier Curves




- In-class exercise
- Where is this curve
  - $C^0$ continuous?
  - $G^1$ continuous?
  - $C^1$ continuous?
- What's the relationship between:
  - the # of control points, and the # of cubic Bézier subcurves?

# Connecting Cubic Bézier Curves

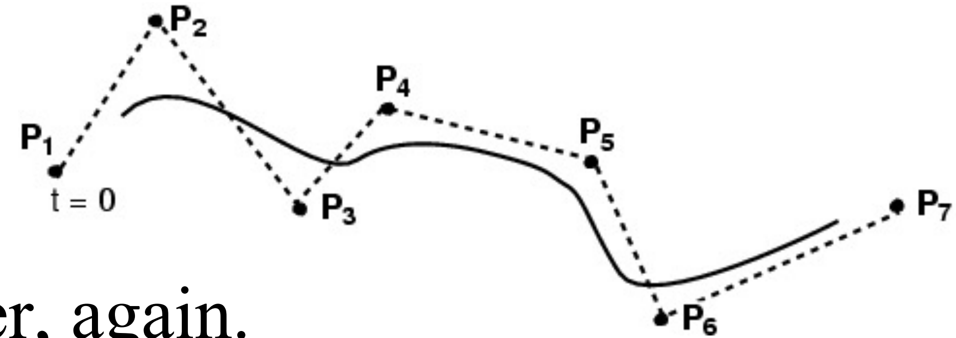



- In-class exercise
- Where is this curve
  - $C^0$ continuous?
  - $G^1$ continuous?
  - $C^1$ continuous?
- What's the relationship between:
  - the # of control points, and the # of cubic Bézier subcurves?

Questions?

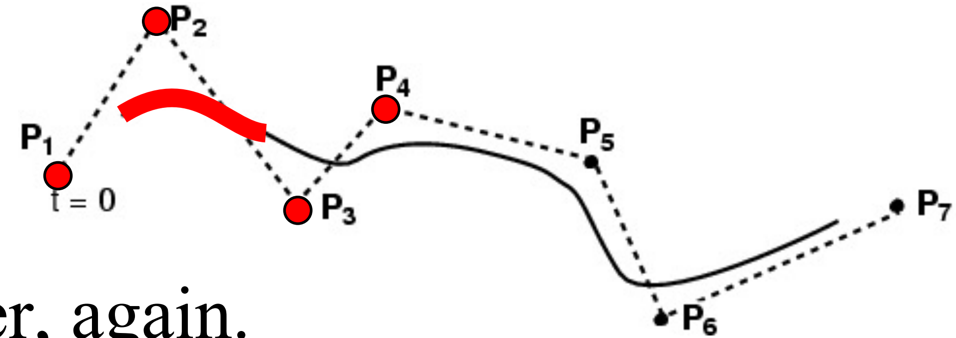
# Cubic B-Splines

- $\geq$ 4 control points
- Locally cubic
  - Cubics chained together, again.



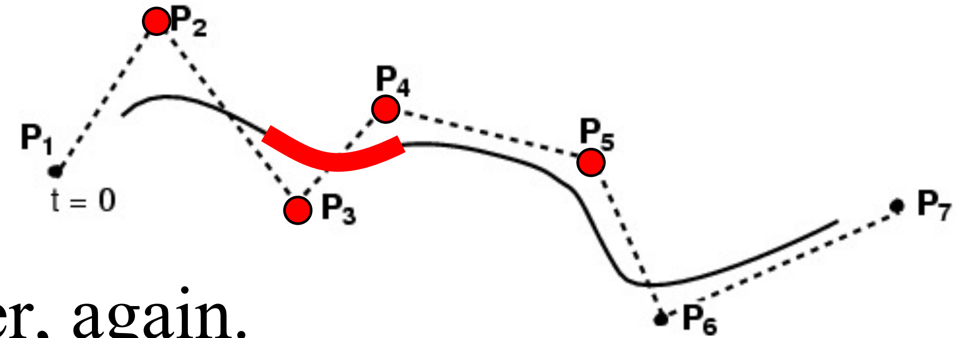
# Cubic B-Splines

- $\geq$ 4 control points
- Locally cubic
  - Cubics chained together, again.



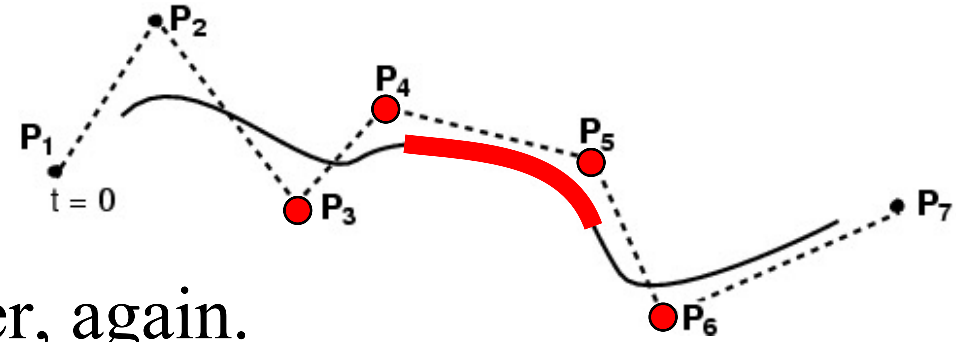
# Cubic B-Splines

- $\geq$ 4 control points
- Locally cubic
  - Cubics chained together, again.



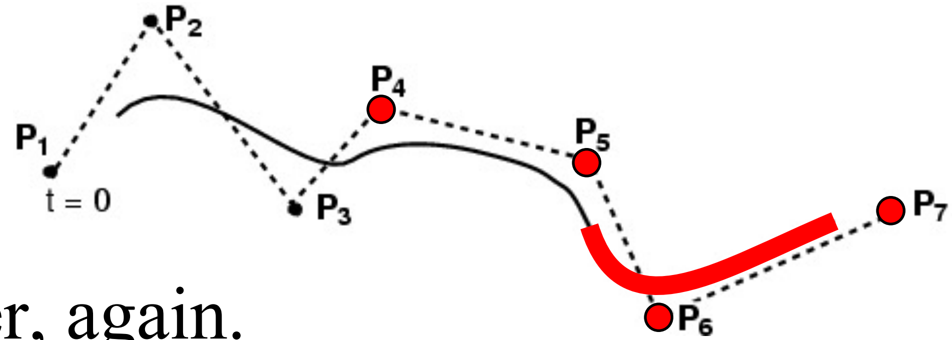
# Cubic B-Splines

- $\geq$ 4 control points
- Locally cubic
  - – Cubics chained together, again.

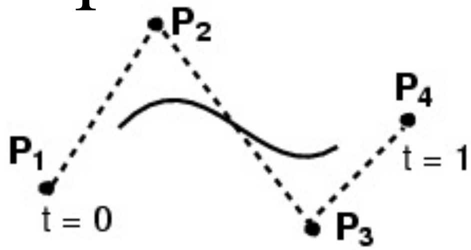


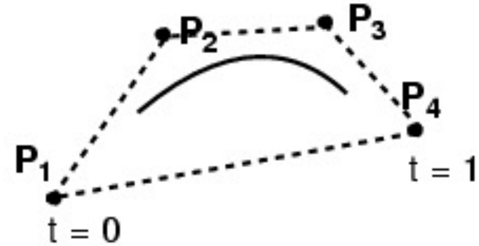
# Cubic B-Splines

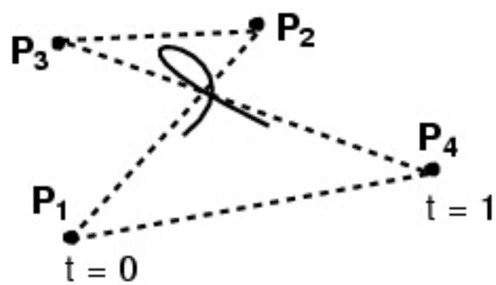
- $\geq$ 4 control points
- Locally cubic
  – Cubics chained together, again.



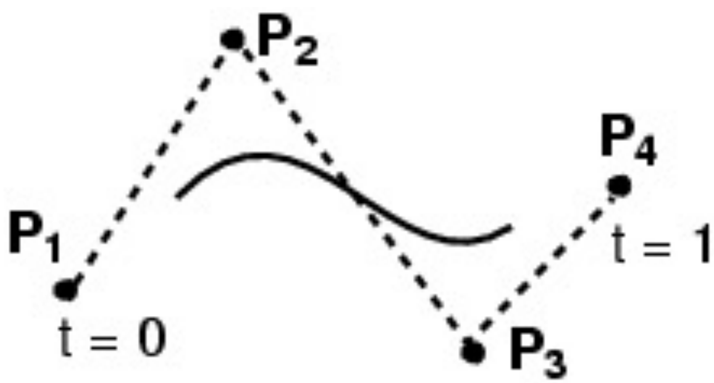

- Curve is not constrained to pass through any control points



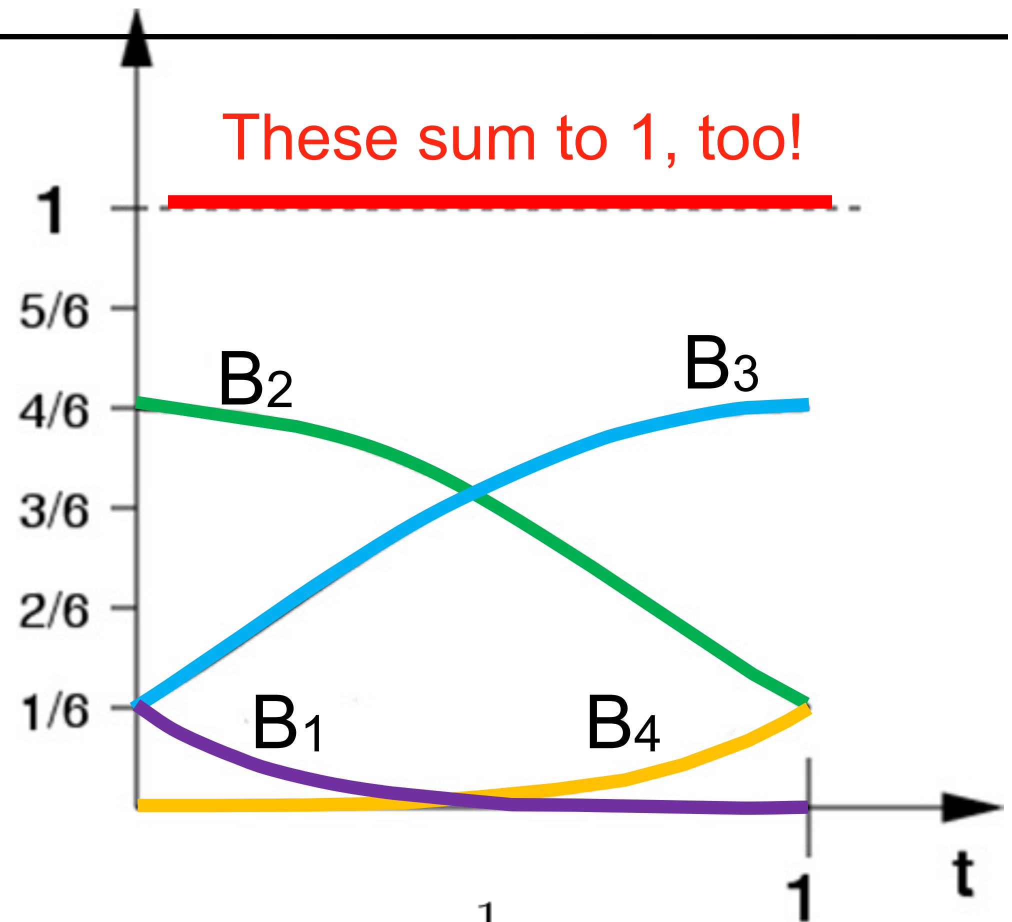



# Cubic B-Splines: Basis




A B-Spline curve is also bounded by the convex hull of its control points.



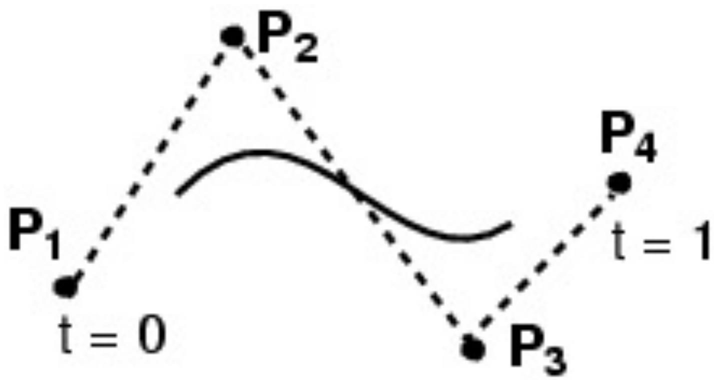

$$B_1(t) = \tfrac{1}{6}(1-t)^3$$

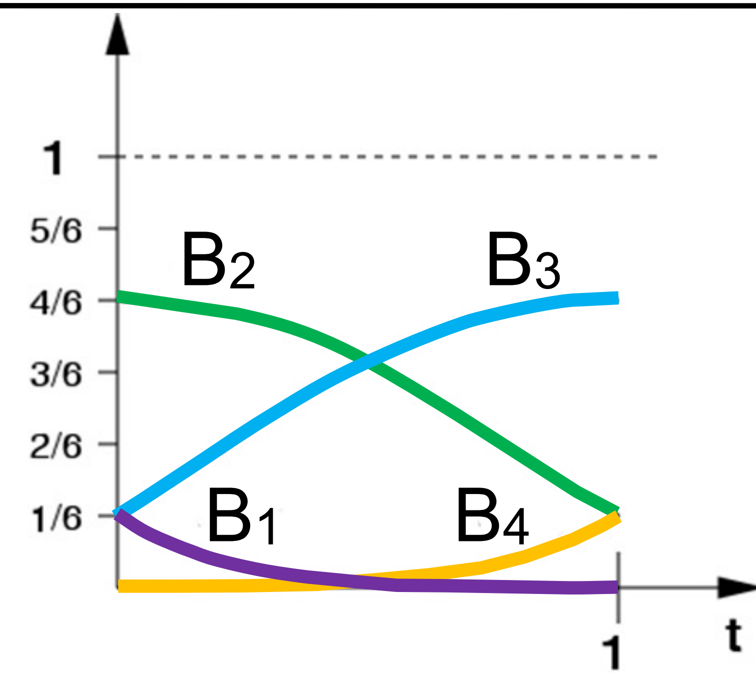
$$B_2(t) = \frac{1}{6}(3t^3 - 6t^2 + 4)$$

$$B_3(t) = \tfrac{1}{6}(-3t^3 + 3t^2 + 3t + 1)$$

$$B_4(t) = \frac{1}{6}t^3$$

# Cubic B-Splines: Basis




$$Q(t) = \frac{(1-t)^3}{6} P_1 + \frac{3t^3 - 6t^2 + 4}{6} P_2 + \frac{-3t^3 + 3t^2 + 3t + 1}{6} P_3 + \frac{t^3}{6} P_4$$

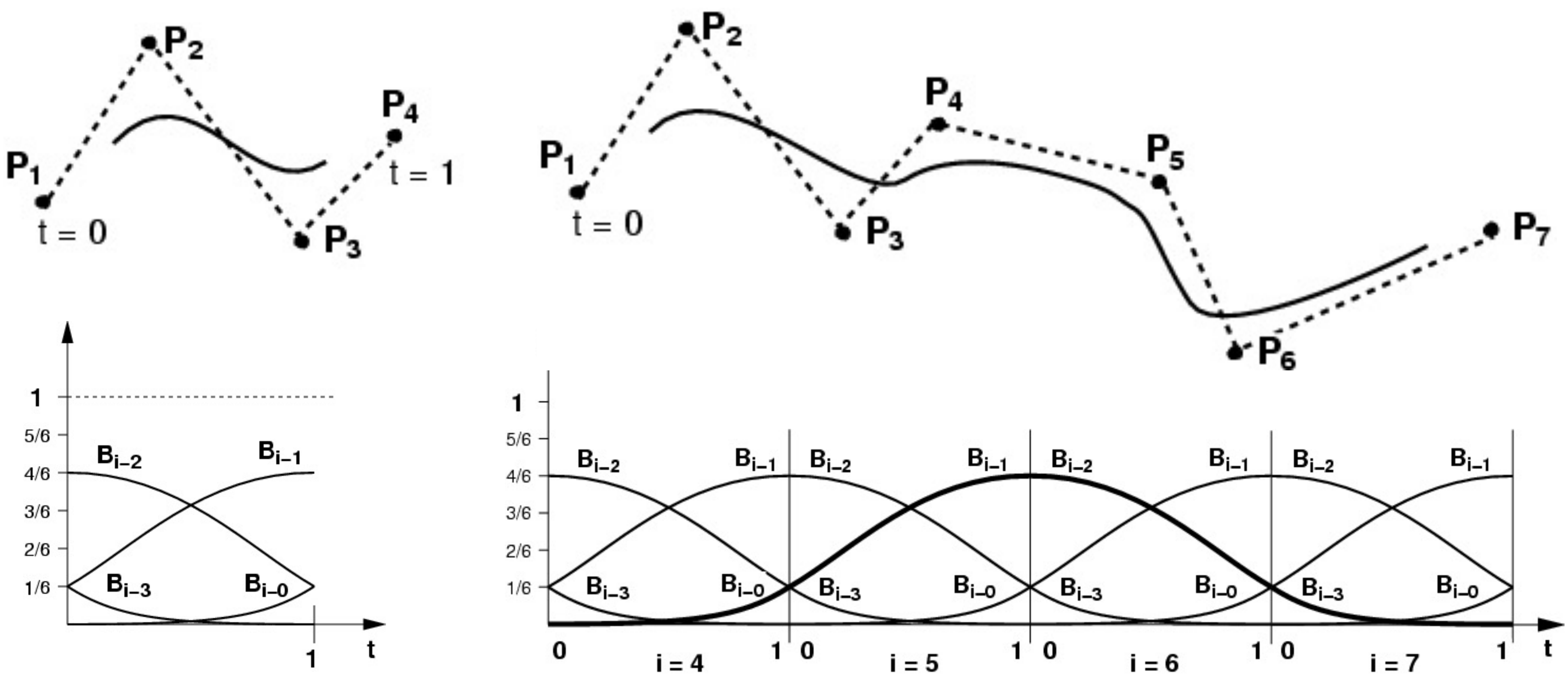
$$Q(t) = \mathbf{GBT(t)}$$

$$\boxed{B_{B-Spline} = \frac{1}{6} \begin{pmatrix} 1 & -3 & 3 & -1 \\ 4 & 0 & -6 & 3 \\ 1 & 3 & 3 & -3 \\ 0 & 0 & 0 & 1 \end{pmatrix}}$$

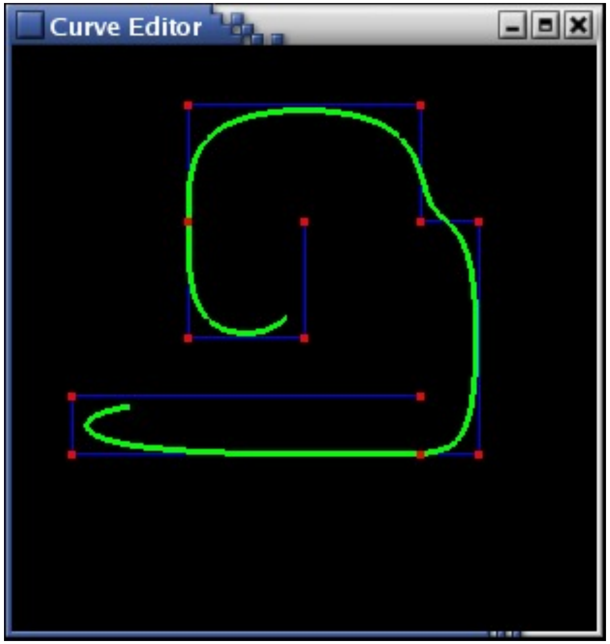
# Cubic B-Splines

- Local control (windowing)
- Automatically $C^2$, and no need to match tangents!

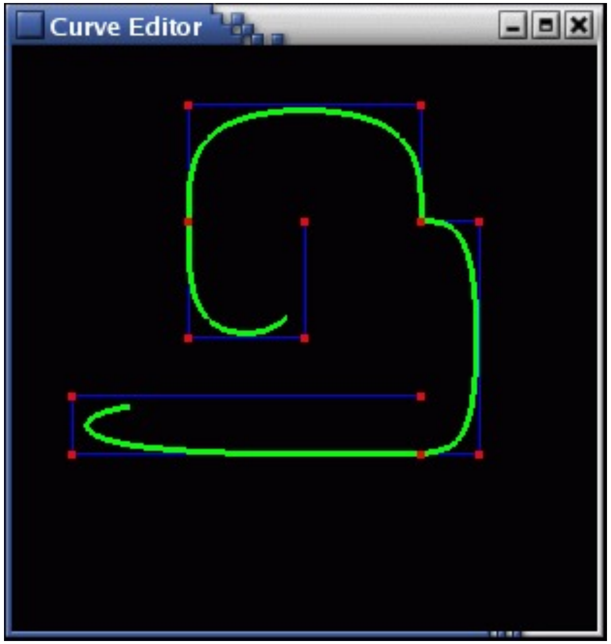
verify



# B-Spline Curve Control Points

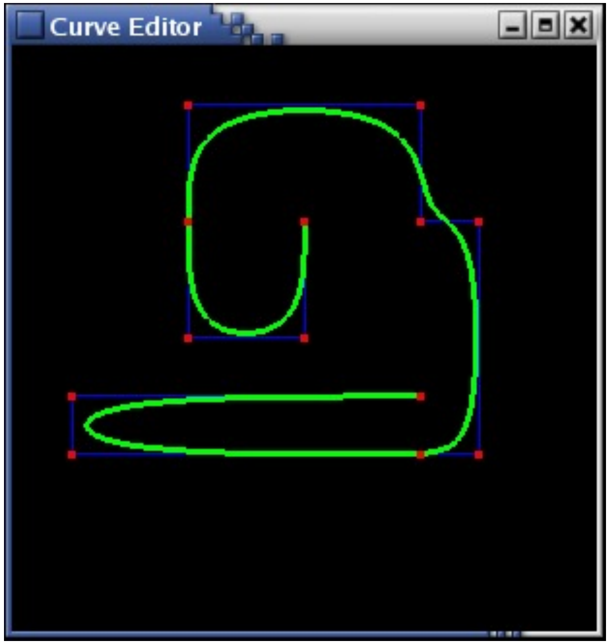



Default B-Spline




B-Spline with derivative discontinuity

Repeat interior control point

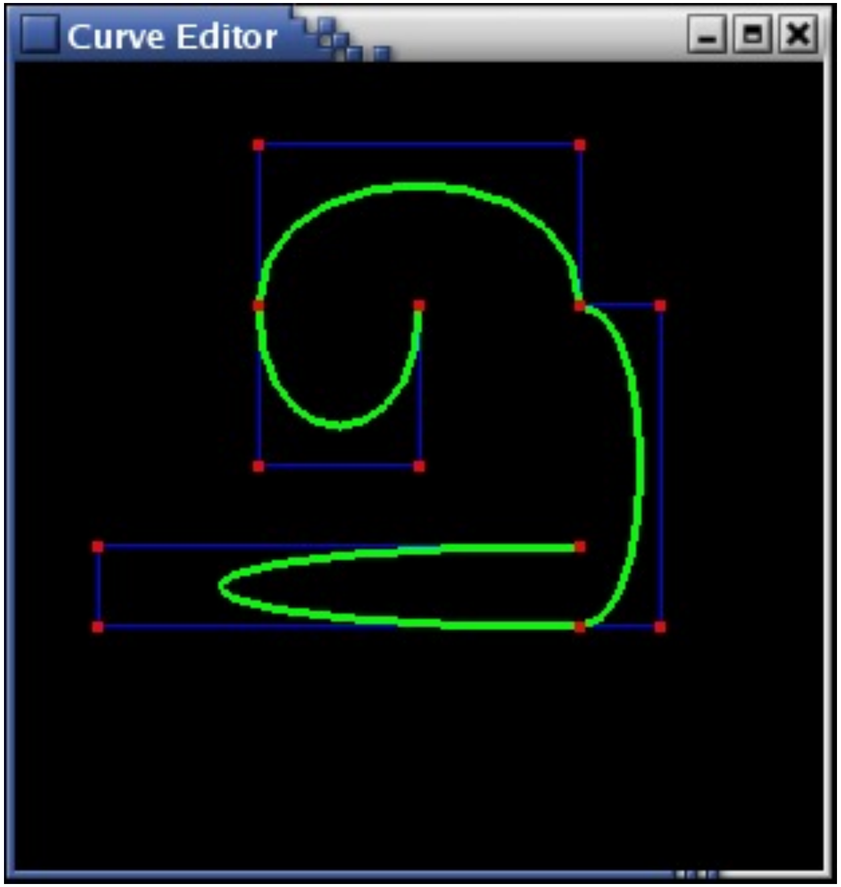



B-Spline which passes through end points

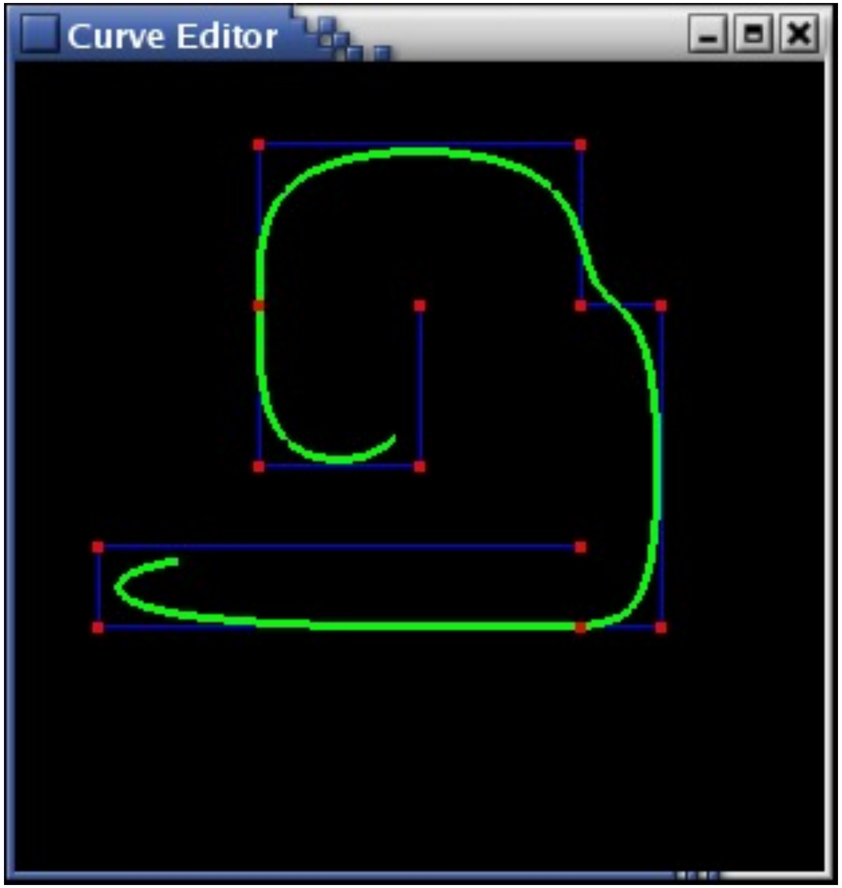
Repeat end points

Try it on Assignment 1!

# Bézier ≠ B-Spline




Bézier




B-Spline

But both are cubics, so one can be converted into the other!

# Converting between Bézier & BSpline

$$Q(t) = \mathbf{GBT(t)} = \text{Geometry } \mathbf{G} \cdot \text{Spline Basis } \mathbf{B} \cdot \text{Power Basis } \mathbf{T(t)}$$

- Simple with the basis matrices!
  - Note that this only works for a single segment of 4 control points

$$B_{Bezier} = \begin{pmatrix} 1 & -3 & 3 & -1 \\ 0 & 3 & -6 & 3 \\ 0 & 0 & 3 & -3 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- $P(t) = G\, B_1\, T(t) = G\, B_1\, (B_2^{-1} B_2)\, T(t) = (G\, B_1\, B_2^{-1})\, B_2\, T(t)$

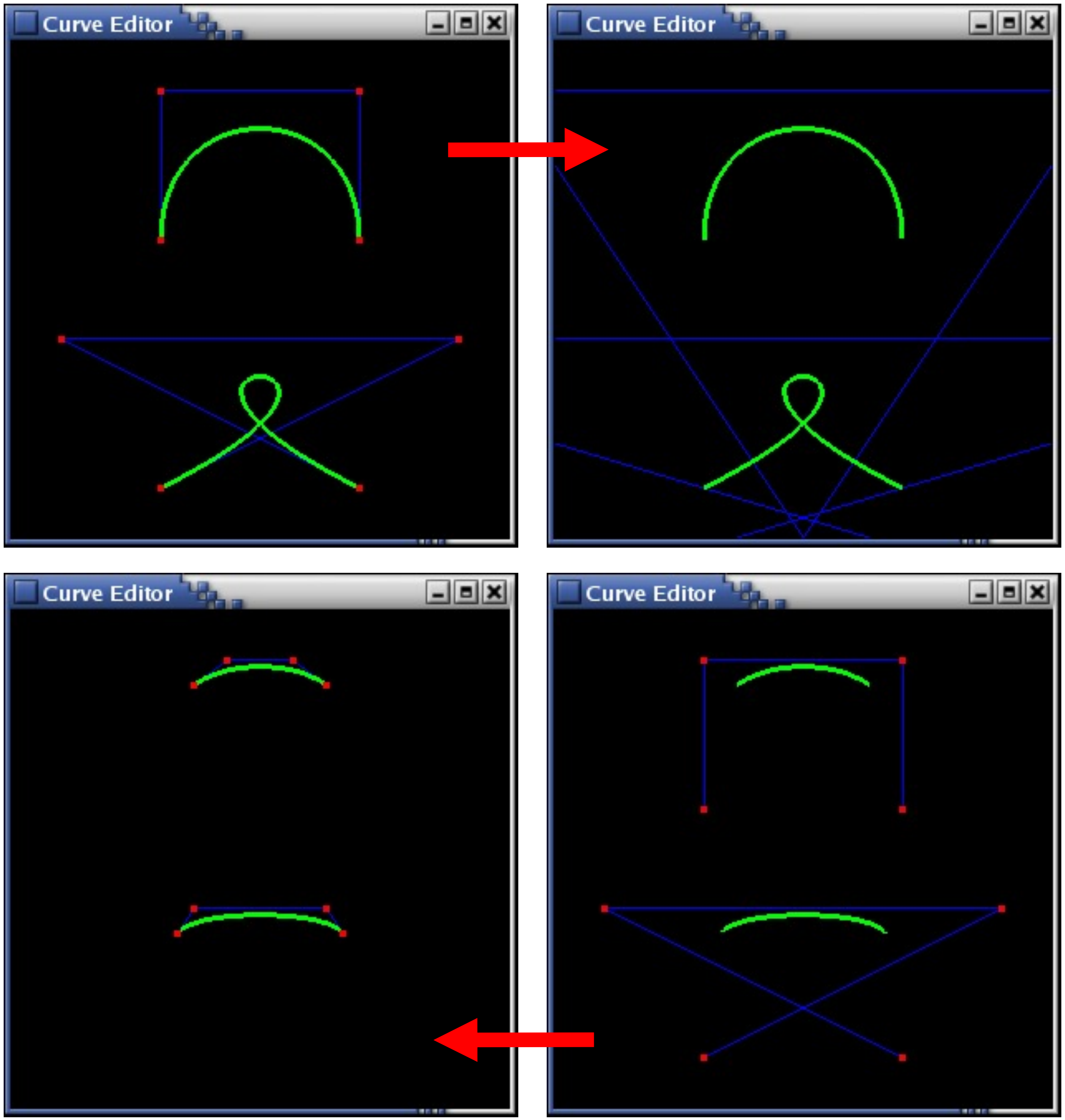
$$B_{B-Spline} = \frac{1}{6} \begin{pmatrix} 1 & -3 & 3 & -1 \\ 4 & 0 & -6 & 3 \\ 1 & 3 & 3 & -3 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- $G\, B_1\, B_2^{-1}$ are the control points for the segment in new basis.

# Converting between Bézier & B-Spline

original control points as Bézier

new BSpline control points to match Bézier




new Bézier control points to match B-Spline

original control points as B-Spline

# NURBS (Generalized B-Splines)

- Rational cubics
  - Use homogeneous coordinates, just add w !
    - Provides an extra weight parameter to control points

- NURBS:  Non-Uniform Rational B-Spline
  - non-uniform = different spacing between the blending functions, a.k.a. “knots”
  - rational = ratio of cubic polynomials (instead of just cubic)
    - implemented by adding the homogeneous coordinate w into the control points.
- Not required in this class

# NURBS (Generalized B-Splines)

- Rational cubics
  - Use homogeneous coordinates, just add w !
    - Provides an extra weight parameter to control points

- NURBS: Non-Uniform Rational B-Spline
  - non-uniform = different spacing between the blending functions, a.k.a. “knots”
  - rational = ratio of cubic polynomials (instead of just cubic)
    - implemented by adding the homogeneous coordinate w into the control points.
- Not required in this class

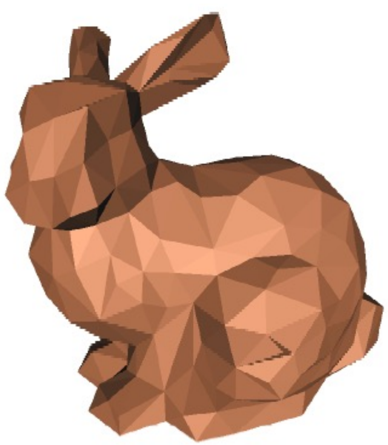
Questions?

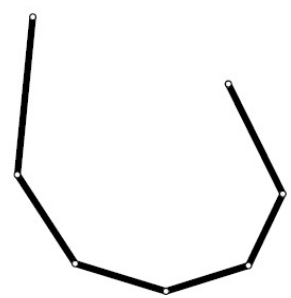
# Representing Surfaces

- Triangle meshes
  - Surface analogue of polylines, this is what GPUs draw
- Tensor Product Splines
  - Surface analogue of spline curves
- Subdivision surfaces
- Implicit surfaces
  - $f(x,y,z)=0$
- Procedural
  - e.g. surfaces of revolution, generalized cylinder
- From volume data (medical images, etc.)

# Triangle Meshes

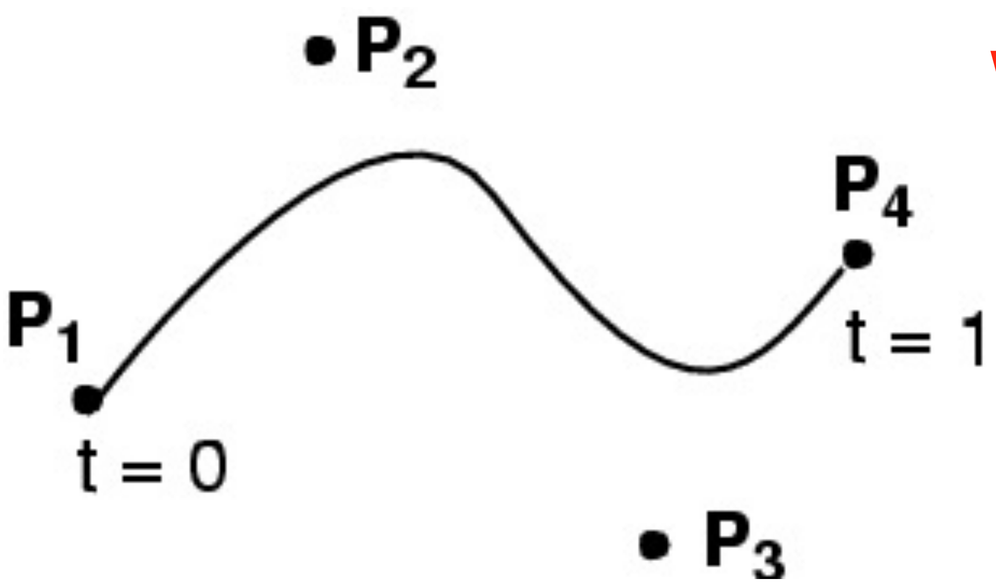
- What you've used so far in Assignment 0
- Triangle represented by 3 vertices
- **Pro**: simple, can be rendered directly
- **Cons**: not smooth, needs many triangles to approximate smooth surfaces (tessellation)





# Smooth Surfaces?

- $P(t) = (1-t)^3 P_1 + 3t(1-t)^2 P_2 + 3t^2(1-t) P_3 + t^3 P_4$




What's the dimensionality of a curve? 1D!

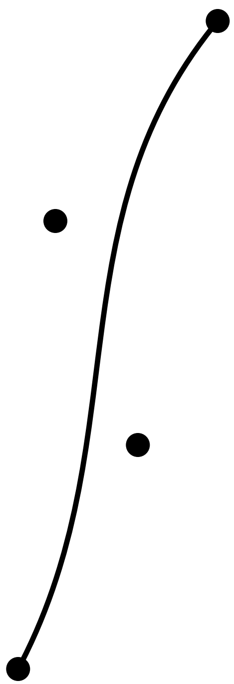
What about a surface?

# How to Build Them? Here’s an Idea

- $P(u) = (1-u)^3 P_1 + 3u(1-u)^2 P_2 + 3u^2(1-u) P_3 + u^3 P_4$

$$
\begin{aligned}
P(u) = &\quad (1-u)^3 && P_1 \\
+ &\quad 3u(1-u)^2 && P_2 \\
+ &\quad 3u^2(1-u) && P_3 \\
+ &\quad u^3 && P_4
\end{aligned}
$$

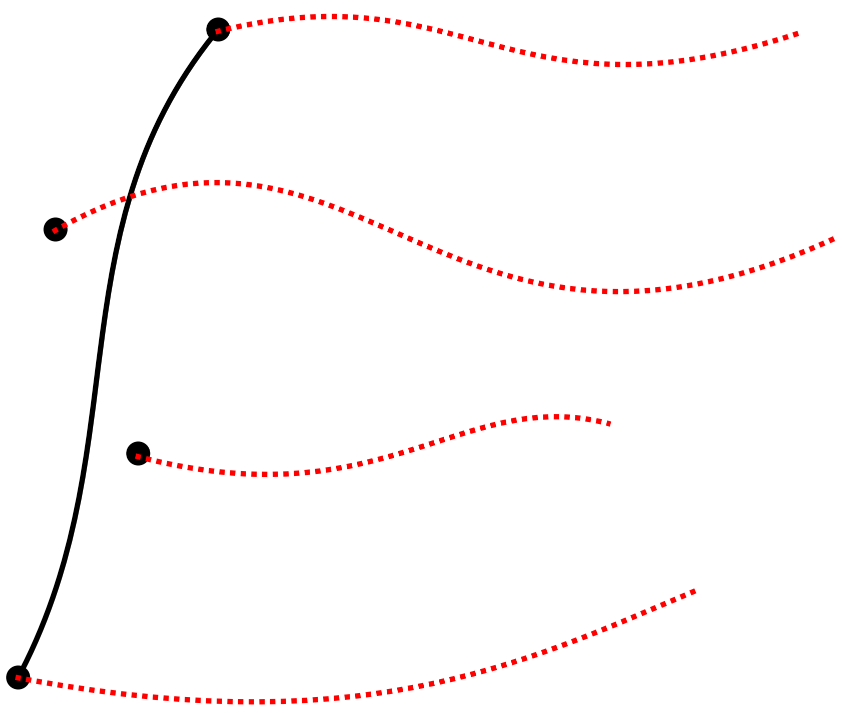
(Note! We relabeled t to u)



# How to Build Them? Here's an Idea

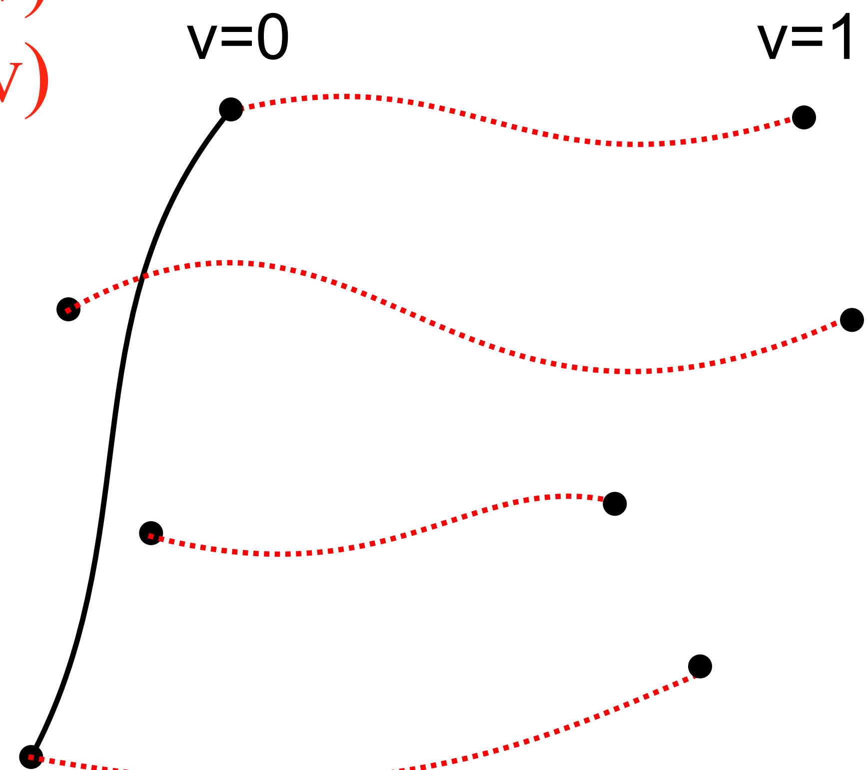
- $$\begin{aligned} P(u) = \quad & (1-u)^3 && P_1 \\ + \quad & 3u(1-u)^2 && P_2 \\ + \quad & 3u^2(1-u) && P_3 \\ + \quad & u^3 && P_4 \end{aligned}$$

(Note! We relabeled t to u)



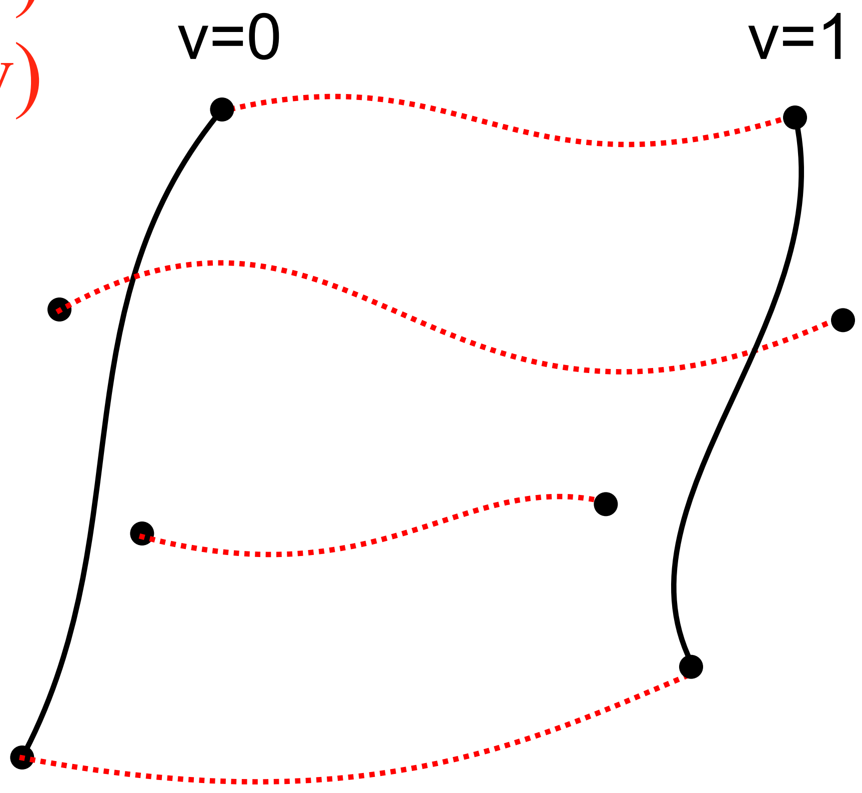
# Here's an Idea

- $$P(u, v) = (1-u)^3 P_1(v) + 3u(1-u)^2 P_2(v) + 3u^2(1-u) P_3(v) + u^3 P_4(v)$$
- Let's make the $P_i$s move along curves!



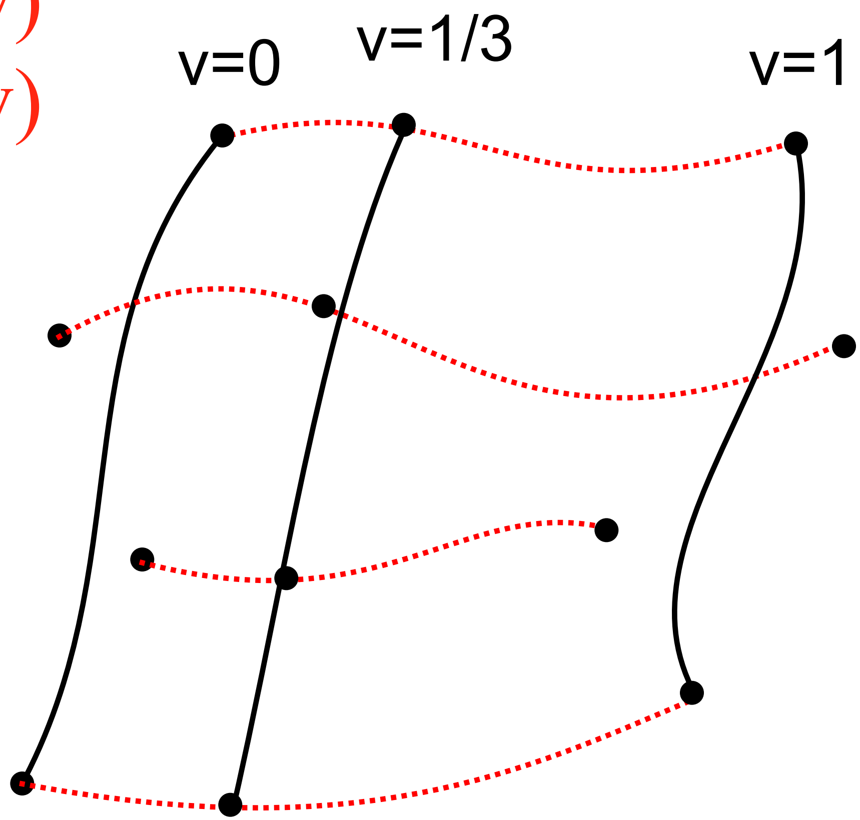
# Here's an Idea

- $P(u, v) = (1\text{-}u)^3 \quad P_1(v)$
  $+ \quad 3u(1\text{-}u)^2 \quad P_2(v)$
  $+ \quad 3u^2(1\text{-}u) \quad P_3(v)$
  $+ \quad u^3 \quad P_4(v)$
- Let's make the $P_i$s move along curves!



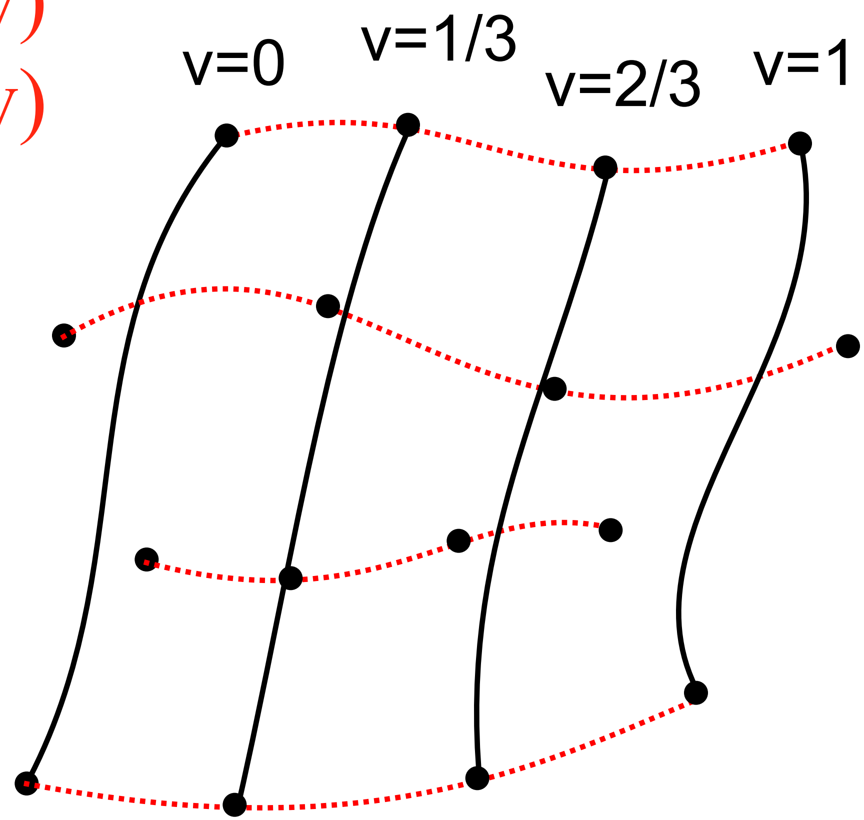
# Here's an Idea

- $$P(u, v) = (1-u)^3 P_1(v) + 3u(1-u)^2 P_2(v) + 3u^2(1-u) P_3(v) + u^3 P_4(v)$$
- Let's make the $P_i$s move along curves!



# Here's an Idea

- $P(u, v) = (1-u)^3 \quad P_1(v)$
  $+ \quad 3u(1-u)^2 \quad P_2(v)$
  $+ \quad 3u^2(1-u) \quad P_3(v)$
  $+ \quad u^3 \quad P_4(v)$
- Let's make the $P_i$s move along curves!

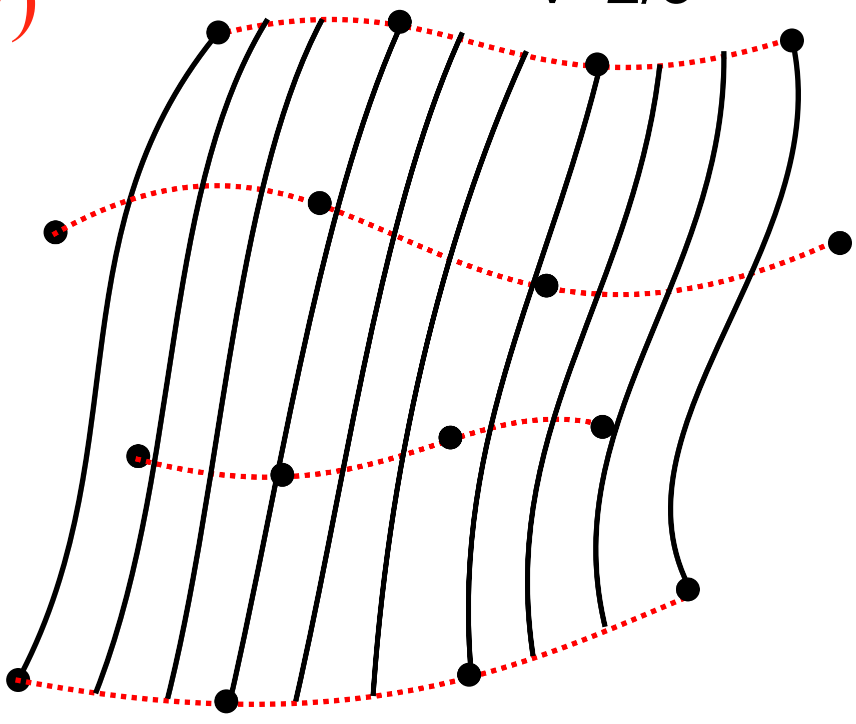


# Here's an Idea

- $P(u, v) = (1-u)^3 \, P_1(v) + 3u(1-u)^2 \, P_2(v) + 3u^2(1-u) \, P_3(v) + u^3 \, P_4(v)$

$$
\begin{aligned}
P(u, v) = \; & (1-u)^3 && P_1(v) \\
+ \; & 3u(1-u)^2 && P_2(v) \\
+ \; & 3u^2(1-u) && P_3(v) \\
+ \; & u^3 && P_4(v)
\end{aligned}
$$

- Let's make the $P_i$s move along curves!

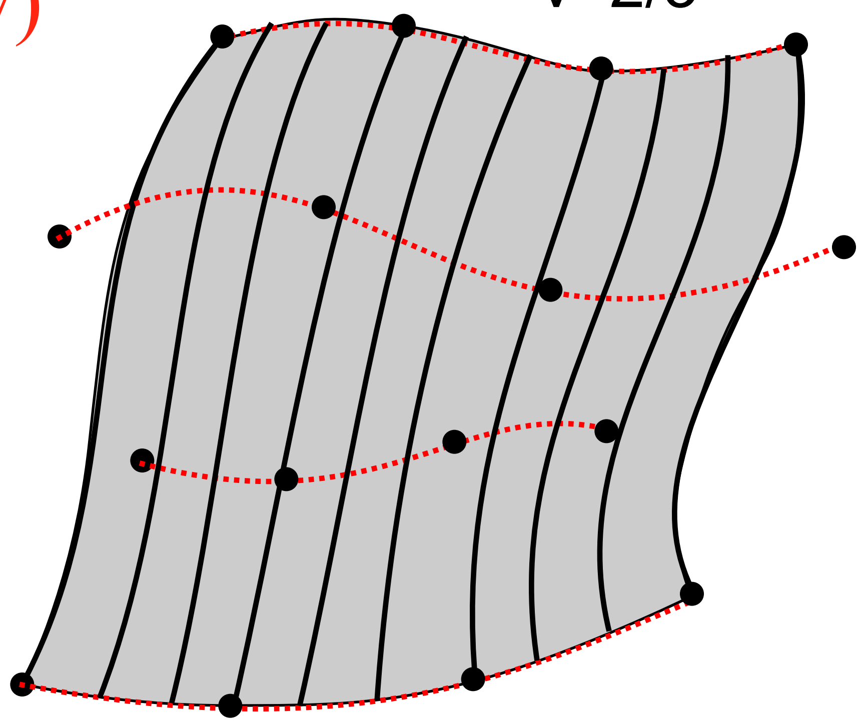



We can define the curve for any value of v from 0 to 1

# Here's an Idea

- $$\begin{aligned} P(u, v) = \ & (1-u)^3 && P_1(v) \\ + \ & 3u(1-u)^2 && P_2(v) \\ + \ & 3u^2(1-u) && P_3(v) \\ + \ & u^3 && P_4(v) \end{aligned}$$
- Let's make the $P_i$s move along curves!

A 2D surface patch!

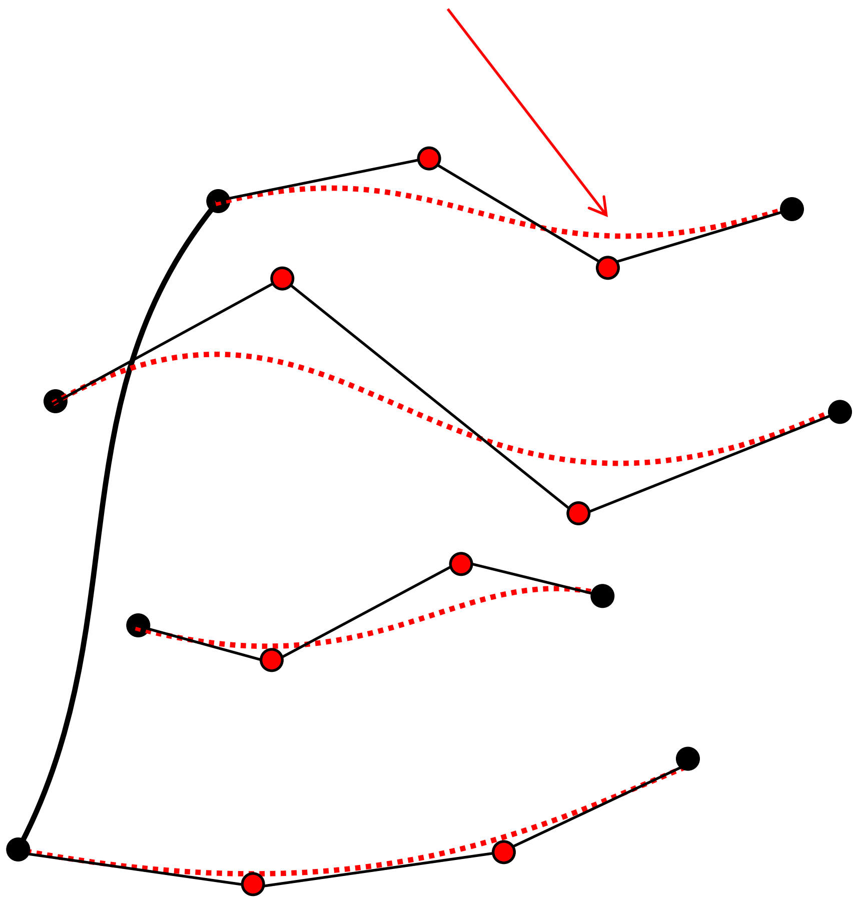



We can define the curve for any value of v from 0 to 1

Function of 2 variables

# Tensor Product Bézier Patches

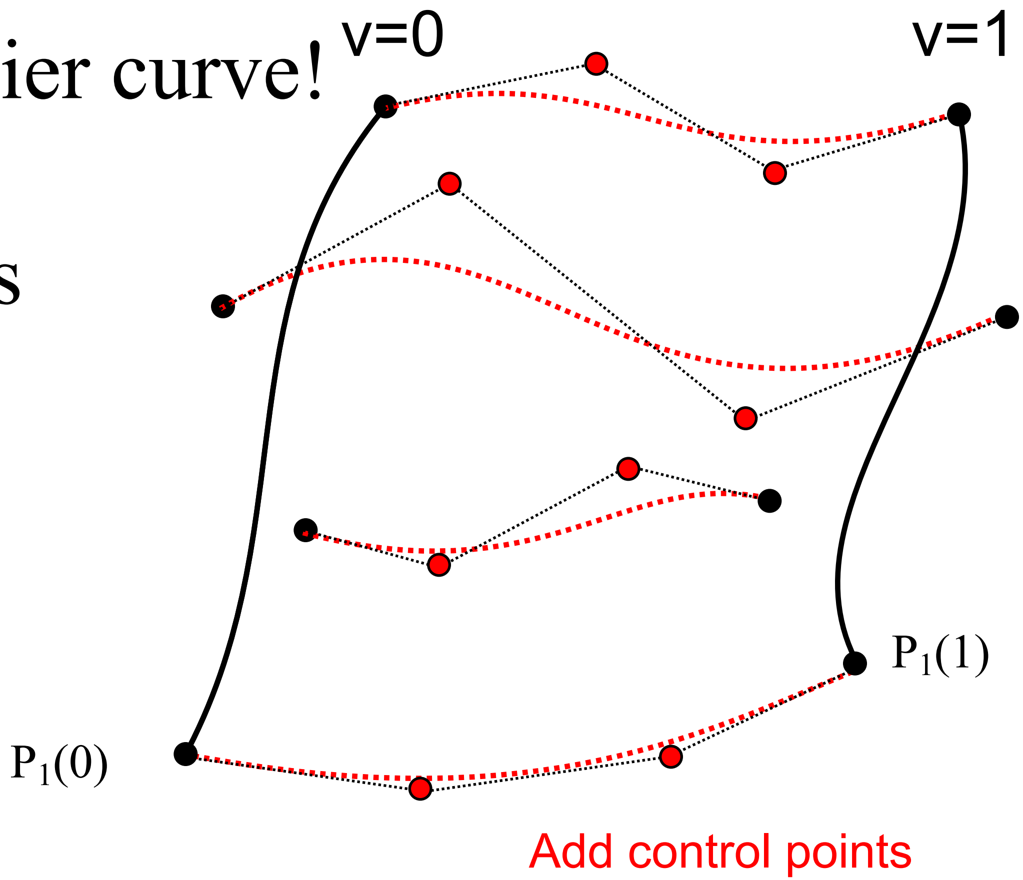
- In the previous, $P_i$s were just some curves
- What if we make **them** Bézier curves?



Add control points

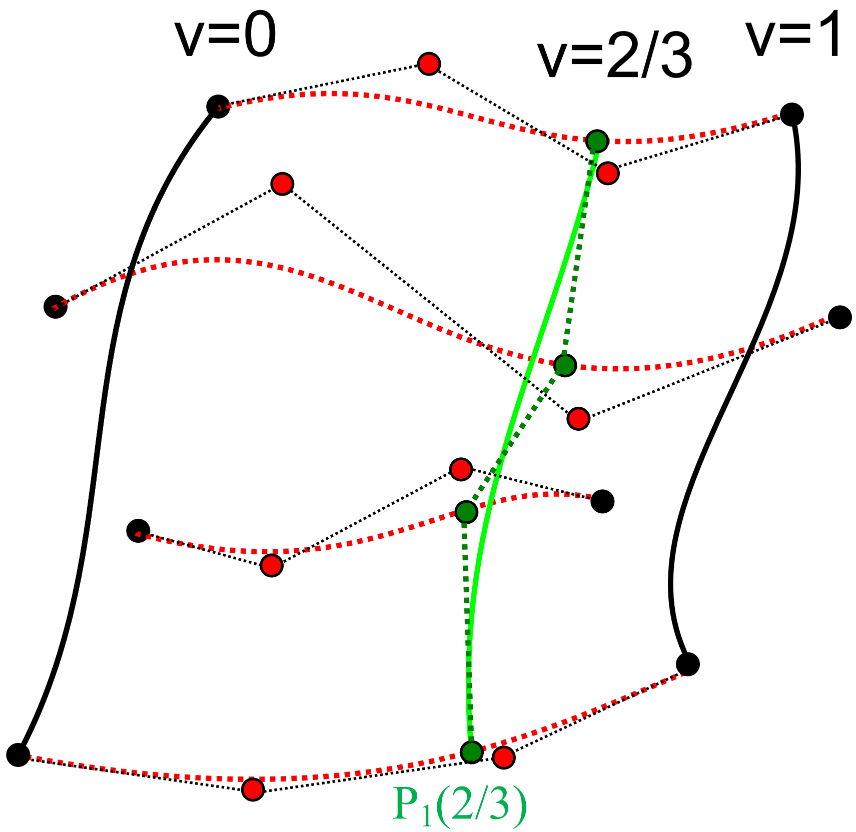
# Tensor Product Bézier Patches

- In the previous, $P_i$s were just some curves
- What if we make **them** Bézier curves?
- Each u=const. **and** v=const. curve is also a Bézier curve!
- E.g., v=0, v=1
  => 4 control points



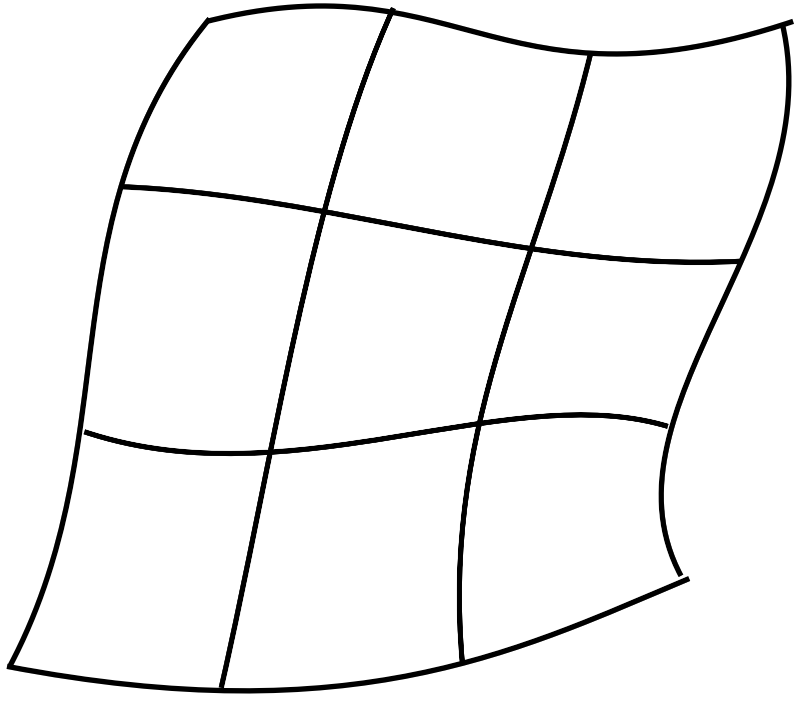
# Tensor Product Bézier Patches

- In the previous, $P_i$s were just some curves
- What if we make **them** Bézier curves?
- Each u=const. **and** v=const. curve is a Bézier curve!
- Note that the boundary control points (except corners) are NOT interpolated!



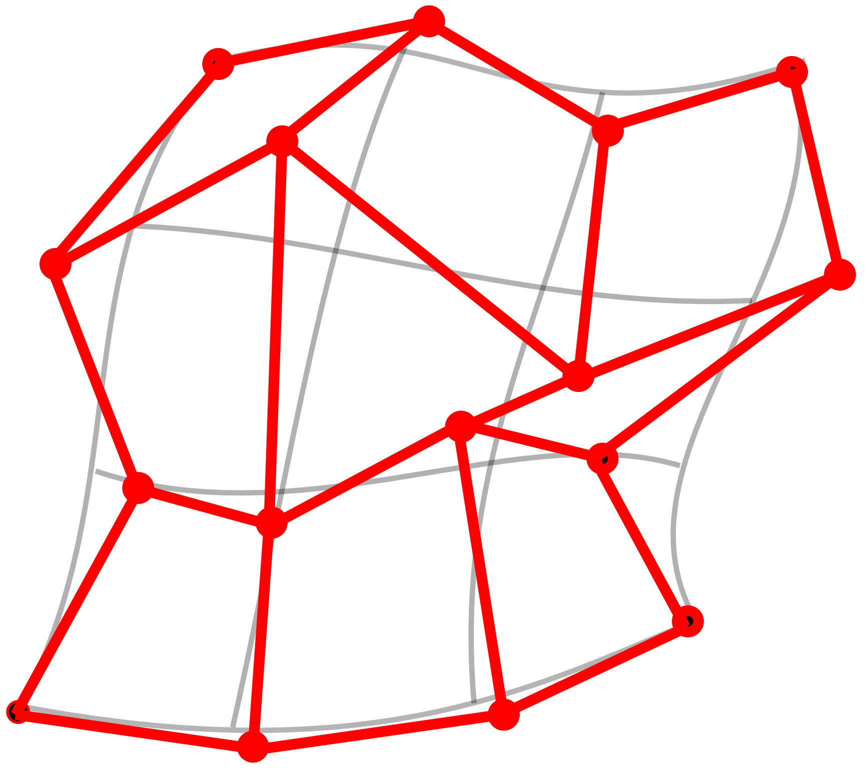
# Tensor Product Bézier Patches

A bicubic Bézier surface



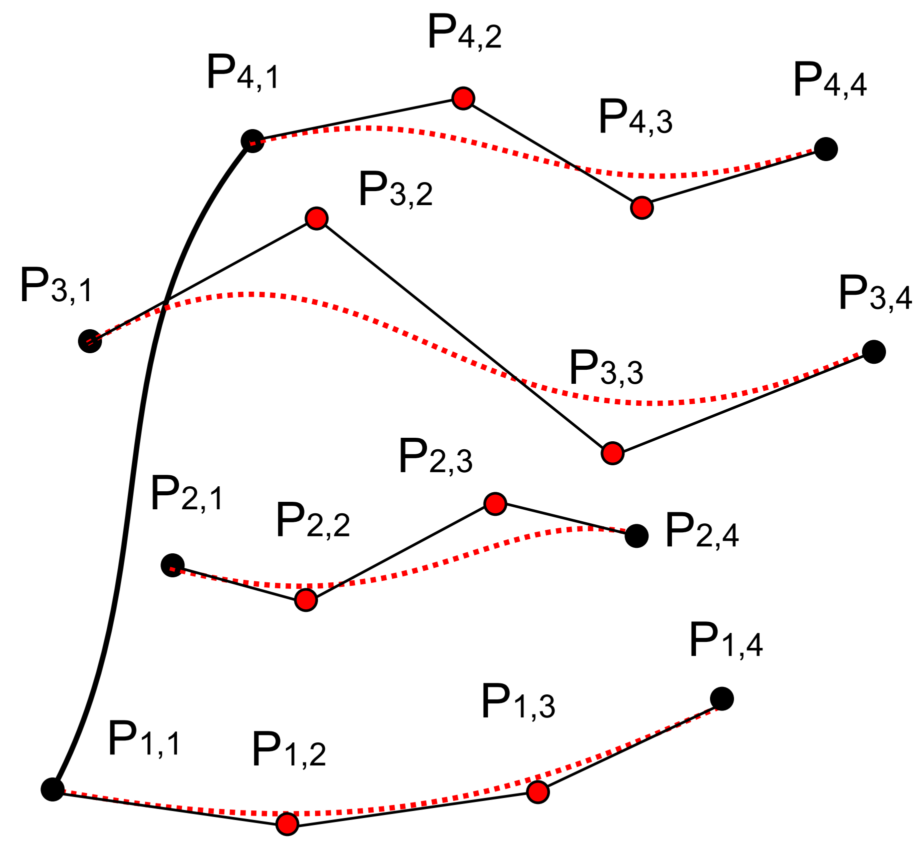
# Tensor Product Bézier Patches

The “Control Mesh”
16 control points



# Bicubics, Tensor Product

- $P(u,v) = B_1(u) * P_1(v) + B_2(u) * P_2(v) + B_3(u) * P_3(v) + B_4(u) * P_4(v)$
- $P_i(v) = B_1(v) * P_{i,1} + B_2(v) * P_{i,2} + B_3(v) * P_{i,3} + B_4(v) * P_{i,4}$



# Bicubics, Tensor Product

- $P(u,v) = B_1(u) * P_1(v)$
  $+ \quad B_2(u) * P_2(v)$
  $+ \quad B_3(u) * P_3(v)$
  $+ \quad B_4(u) * P_4(v)$
- $P_i(v) = B_1(v) * P_{i,1}$
  $+ \quad B_2(v) * P_{i,2}$
  $+ \quad B_3(v) * P_{i,3}$
  $+ \quad B_4(v) * P_{i,4}$

$$P(u,v) =$$

$$\sum_{i=1}^{4} B_i(u) \left[ \sum_{j=1}^{4} P_{i,j} \, B_j(v) \right]$$

$$= \sum_{i=1}^{4} \sum_{j=1}^{4} P_{i,j} \, B_{i,j}(u,v)$$

$$B_{i,j}(u,v) = B_i(u) B_j(v)$$

# Bicubics, Tensor Product

- $P(u,v) = B_1(u) * P_1(v)$
  $+ B_2(u) * P_2(v)$
  $+ B_3(u) * P_3(v)$
  $+ B_4(u) * P_4(v)$
- $P_i(v) = B_1(v) * P_{i,1}$
  $+ B_2(v) * P_{i,2}$
  $+ B_3(v) * P_{i,3}$
  $+ B_4(v) * P_{i,4}$

$$P(u,v) =$$

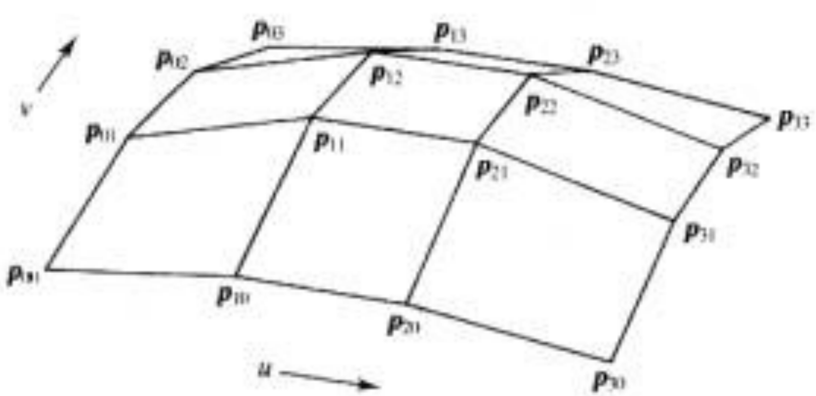
16 control points $P_{i,j}$
16 2D basis functions $B_{i,j}$

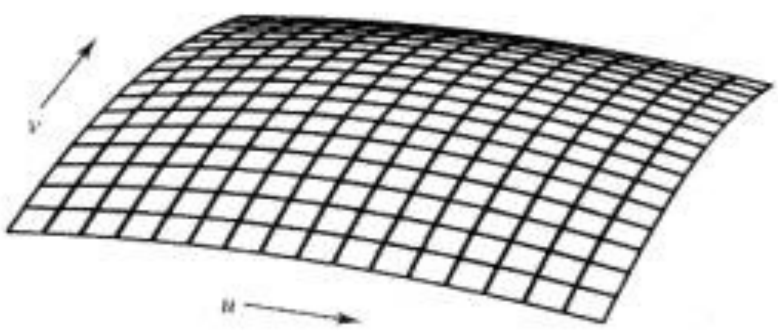
$$= \sum_{i=1}^{4} \sum_{j=1}^{4} \boxed{P_{i,j}} B_{i,j}(u,v)$$

$$\boxed{B_{i,j}(u,v) = B_i(u) B_j(v)}$$

# Recap: Tensor Bézier Patches

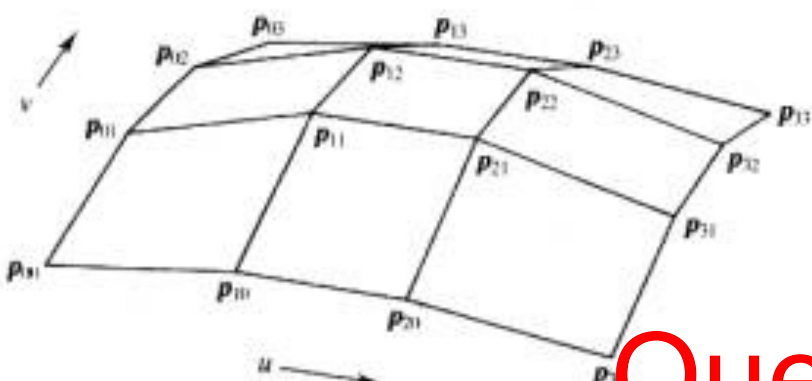
- Parametric surface $P(u,v)$ is a bicubic polynomial of two variables u & v
- Defined by 4x4=16 control points $P_{1,1}$, $P_{1,2}$.... $P_{4,4}$
- Interpolates 4 corners, approximates others
- Basis are product of two Bernstein polynomials: $B_1(u)B_1(v)$; $B_1(u)B_2(v)$;... $B_4(u)B_4(v)$

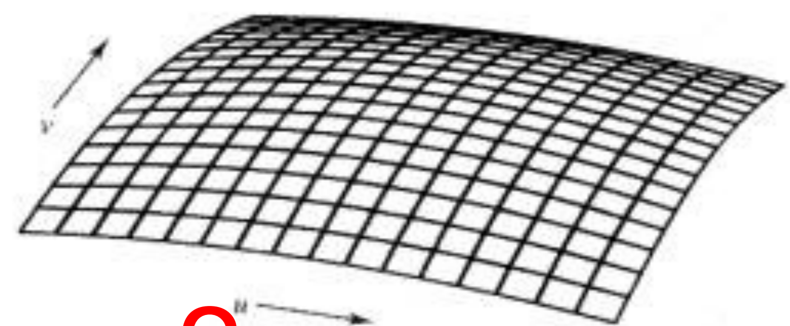


# Recap: Tensor Bézier Patches

- Parametric surface P(u,v) is a bicubic polynomial of two variables u & v
- Defined by 4x4=16 control points $P_{1,1}$, $P_{1,2}$.... $P_{4,4}$
- Interpolates 4 corners, approximates others
- Basis are product of two Bernstein polynomials: $B_1(u)B_1(v)$; $B_1(u)B_2(v)$;... $B_4(u)B_4(v)$

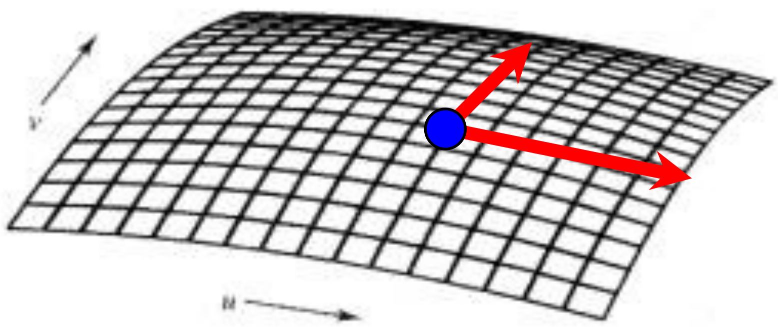



Questions?

# Tangents and Normals for Patches

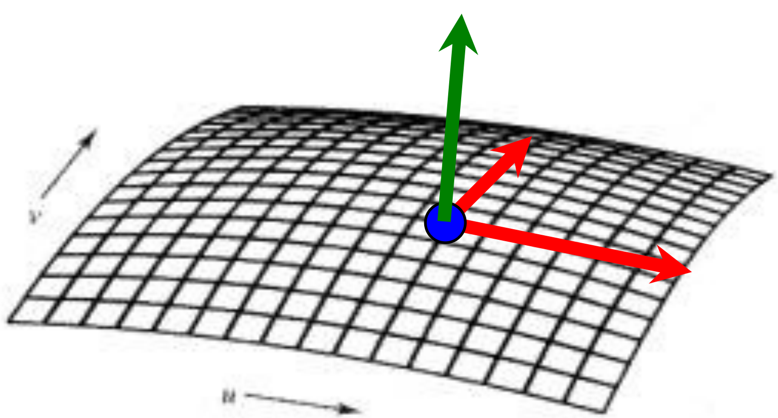
- P(u,v) is a 3D point specified by *u*, *v*
- The partial derivatives $\partial P/\partial u$ and $\partial P/\partial v$ are 3D vectors
  - Both are tangent to surface at P



# Tangents and Normals for Patches

- P(u,v) is a 3D point specified by *u*, *v*
- The partial derivatives $\partial P/\partial u$ and $\partial P/\partial v$ are 3D vectors
  - Both are tangent to surface at P
  - Normal is perpendicular to both, i.e.,

$$n = (\partial P/\partial u) \times (\partial P/\partial v)$$

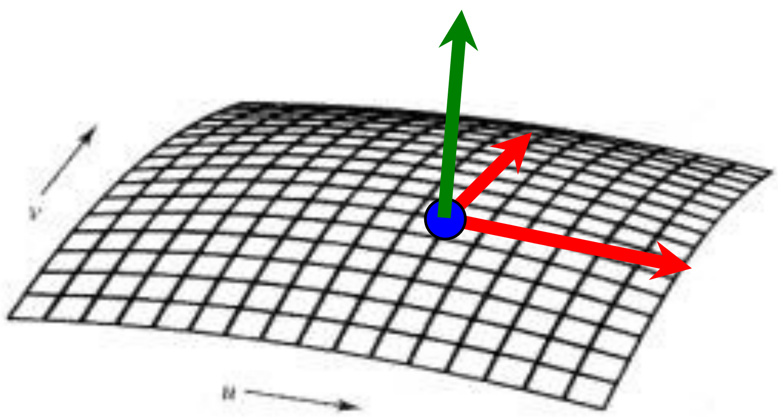


n is usually not unit, so must normalize!

# Tangents and Normals for Patches

- P(u,v) is a 3D point specified by *u*, *v*
- The partial derivatives $\partial P/\partial u$ and $\partial P/\partial v$ are 3D vectors
  - Both are tangent to surface at P
  - Normal is perpendicular to both, i.e.,

$$n = (\partial P/\partial u) \times (\partial P/\partial v)$$



n is usually not unit, so must normalize!

Questions?

# Recap: Matrix Notation for Curves

- Cubic Bézier in matrix notation

point on curve
(2x1 vector)

Canonical
"power basis"

$$\mathrm{P(t)} = \begin{pmatrix} x(t) \\ y(t) \end{pmatrix} =$$

$$\begin{pmatrix} x_1 & x_2 & x_3 & x_4 \\ y_1 & y_2 & y_3 & y_4 \end{pmatrix} \begin{pmatrix} 1 & -3 & 3 & -1 \\ 0 & 3 & -6 & 3 \\ 0 & 0 & 3 & -3 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ t \\ t^2 \\ t^3 \end{pmatrix}$$

"Geometry matrix"
of control points $P_1..P_4$
(2 x 4)

"Spline matrix"
(Bernstein)

# Hardcore: Matrix Notation for Patches

- Not required, but convenient!

$$P(u,v) = \sum_{i=1}^{4} B_i(u) \left[ \sum_{j=1}^{4} P_{i,j} B_j(v) \right]$$

x coordinate of surface at (u,v)

$$P^x(u,v) =$$

Column vector of basis functions (v)

$$(B_1(u), \ldots, B_4(u)) \begin{pmatrix} P^x_{1,1} & \cdot\cdot & P^x_{1,4} \\ \vdots & & \vdots \\ P^x_{4,1} & \cdot\cdot & P^x_{4,4} \end{pmatrix} \begin{pmatrix} B_1(v) \\ \vdots \\ B_4(v) \end{pmatrix}$$

Row vector of basis functions (u)

4x4 matrix of x coordinates of the control points

# Hardcore: Matrix Notation for Patches

- Curves:

$$P(t) = \boldsymbol{G}\,\boldsymbol{B}\,\boldsymbol{T}(t)$$

- Surfaces:

$$P^x(u,v) = \boldsymbol{T}(u)^{\mathrm{T}}\,\boldsymbol{B}^{\mathrm{T}}\,\boldsymbol{G}^x\,\boldsymbol{B}\,\boldsymbol{T}(v)$$

A separate 4x4 geometry matrix for x, y, z

- T = power basis
  B = spline matrix
  G = geometry matrix

# Super Hardcore: Tensor Notation

- You can stack the $G^x$, $G^y$, $G^z$ matrices into a geometry tensor of control points
  - I.e., $G^k{}_{i,j}$ = the k$^{th}$ coordinate of control point $P_{i,j}$
  - A cube of numbers!

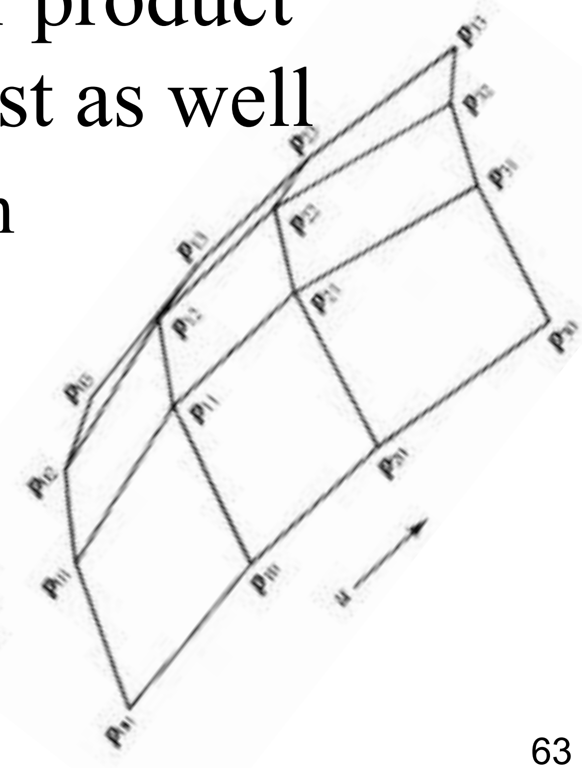
$$P^k(u,v) = \boldsymbol{T}^l(u)\,\boldsymbol{B}_l^i\,\boldsymbol{G}_{ij}^k\,\boldsymbol{B}_m^j\,\boldsymbol{T}^m(v)$$

- "Definitely not required, but nice!
  - See http://en.wikipedia.org/wiki/Multilinear_algebra

# Tensor Product B-Spline Patches

- Bézier and B-Spline curves are both cubics
  - Can change between representations using matrices

- Consequently, you can build tensor product surface patches out of B-Splines just as well
  - Still 4x4 control points for each patch
  - 2D basis functions are pairwise products of B-Spline basis functions
  - Yes, simple!

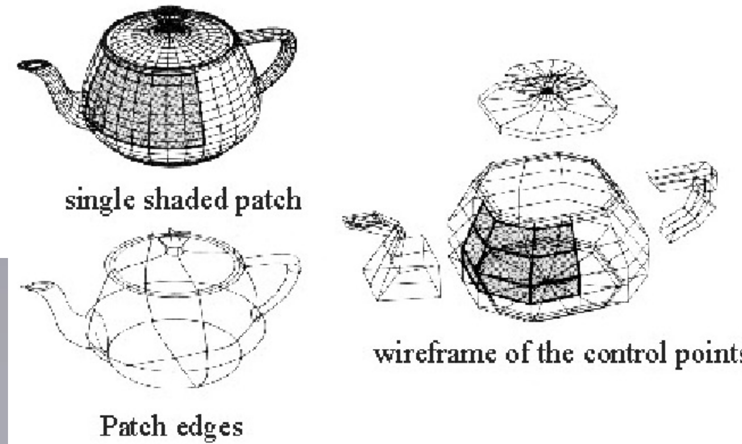


# Tensor Product Spline Patches

- Pros
  - Smooth
  - Defined by reasonably small set of points
- Cons
  - Harder to render (usually converted to triangles)
  - Tricky to ensure continuity at patch boundaries
- Extensions
  - Rational splines: Splines in homogeneous coordinates
  - NURBS: Non-Uniform Rational B-Splines
    - Like curves: ratio of polynomials, non-uniform location of control points, etc.

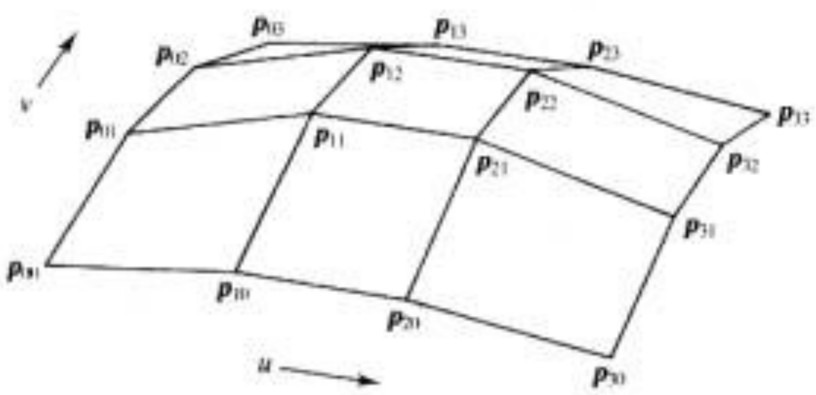
# Utah Teapot: Tensor Bézier Splines

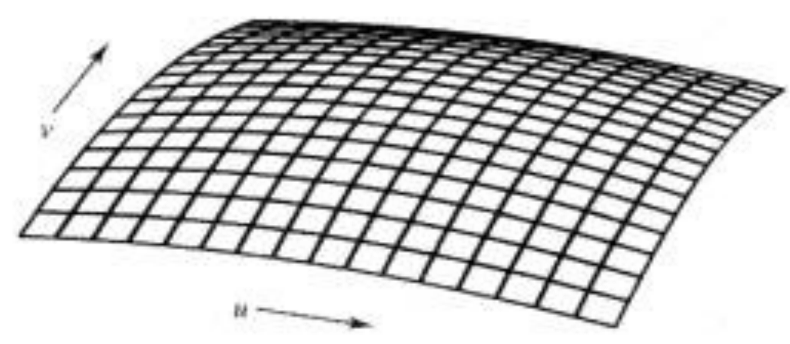
- Designed by Martin Newell 1975




# Cool: Displacement Mapping

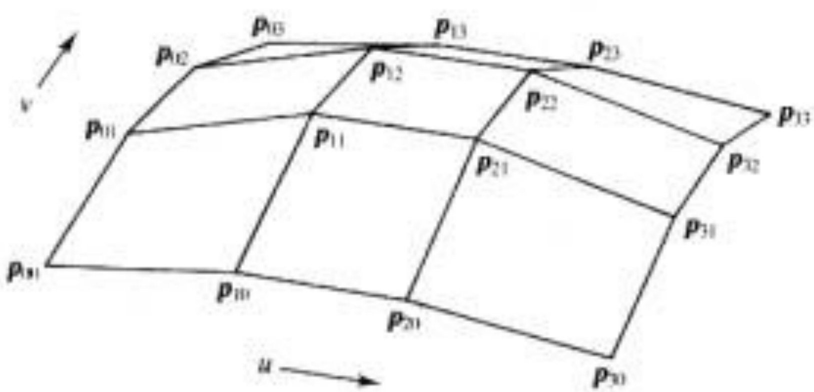
- Not all surfaces are smooth...

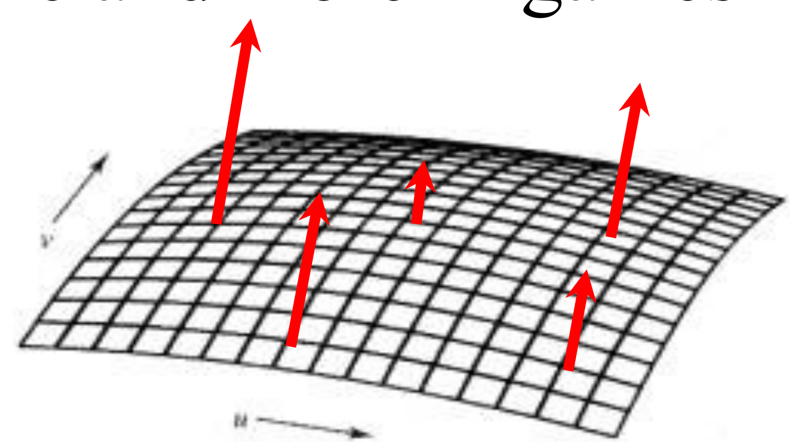


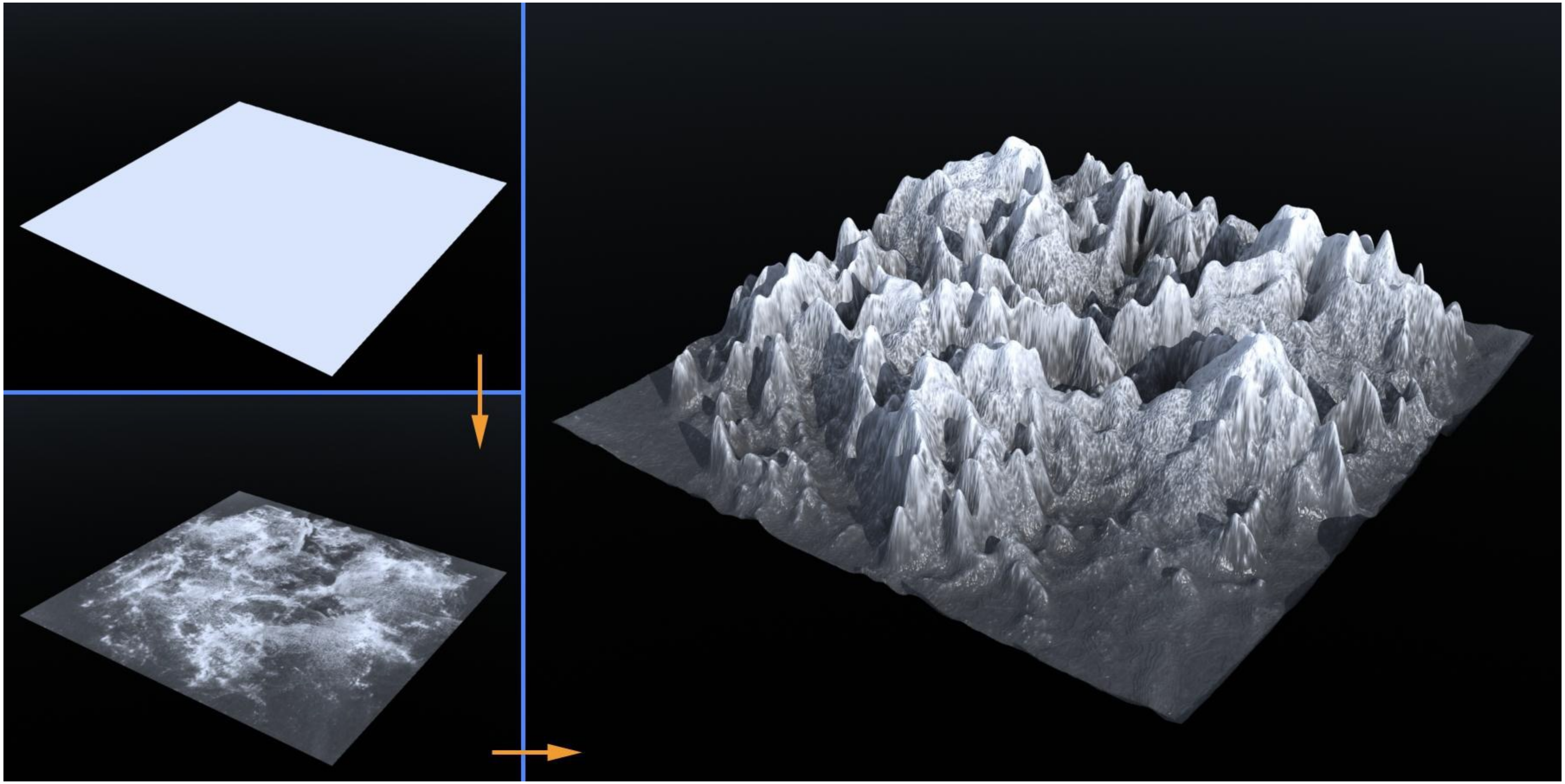
# Cool: Displacement Mapping

- Not all surfaces are smooth...
- “Paint” displacements on a smooth surface
  - For example, in the direction of normal
- Tessellate smooth patch into fine grid, then add displacement D(u,v) to vertices
- Heavily used in movies, more and more in games

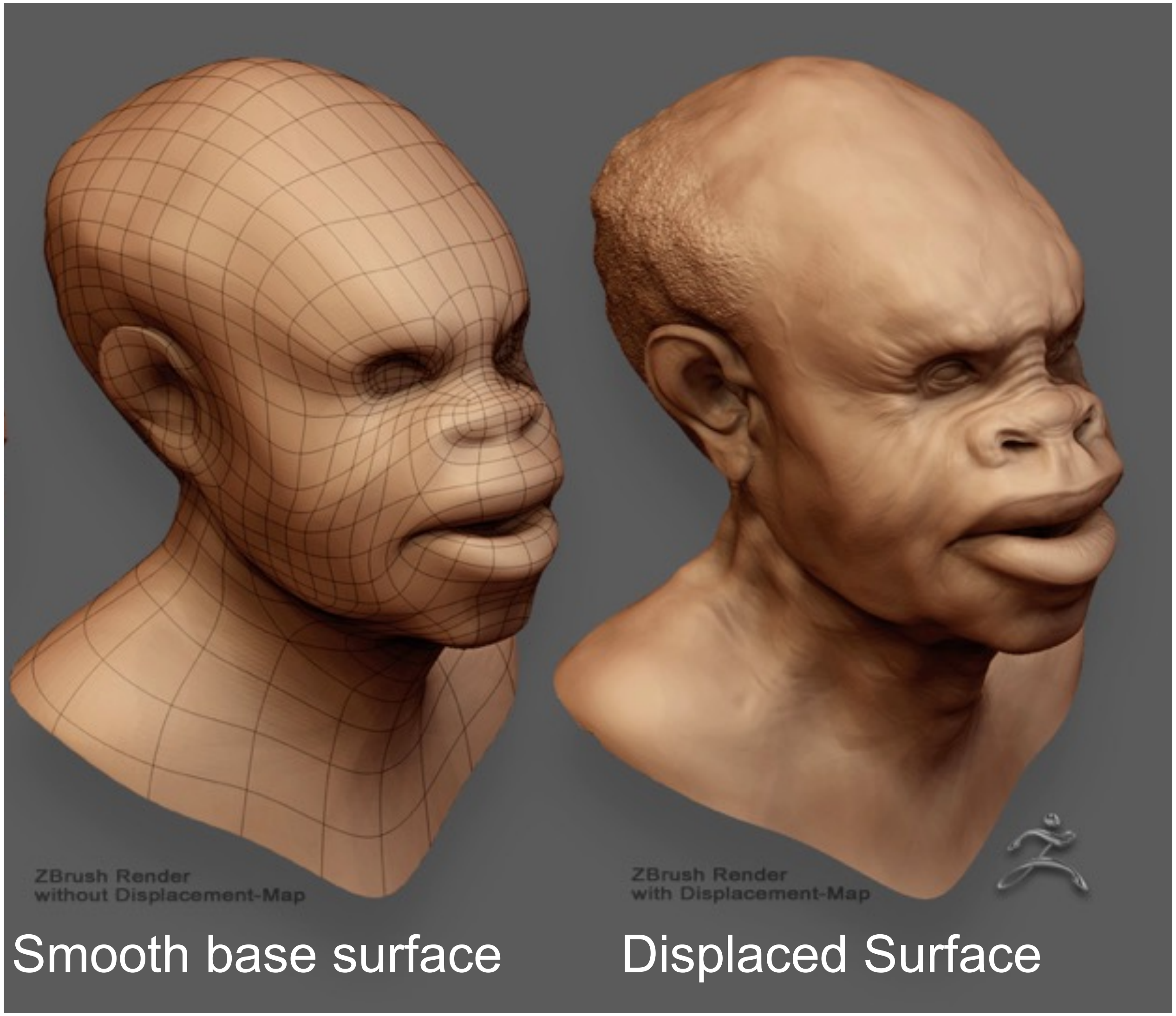




# Cool: Displacement Mapping

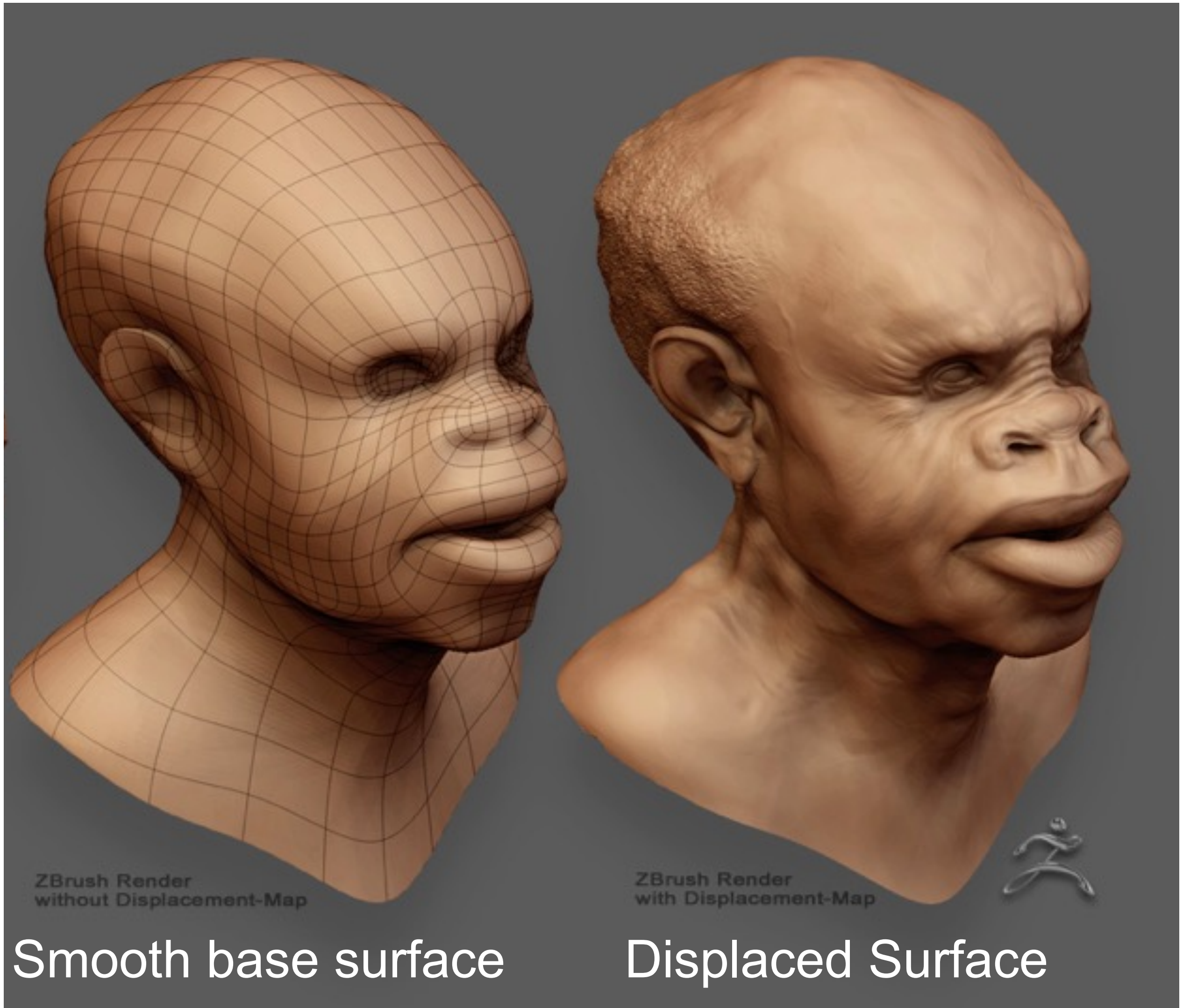


# Displacement Mapping Example




zbrushcentral

# Displacement Mapping Example



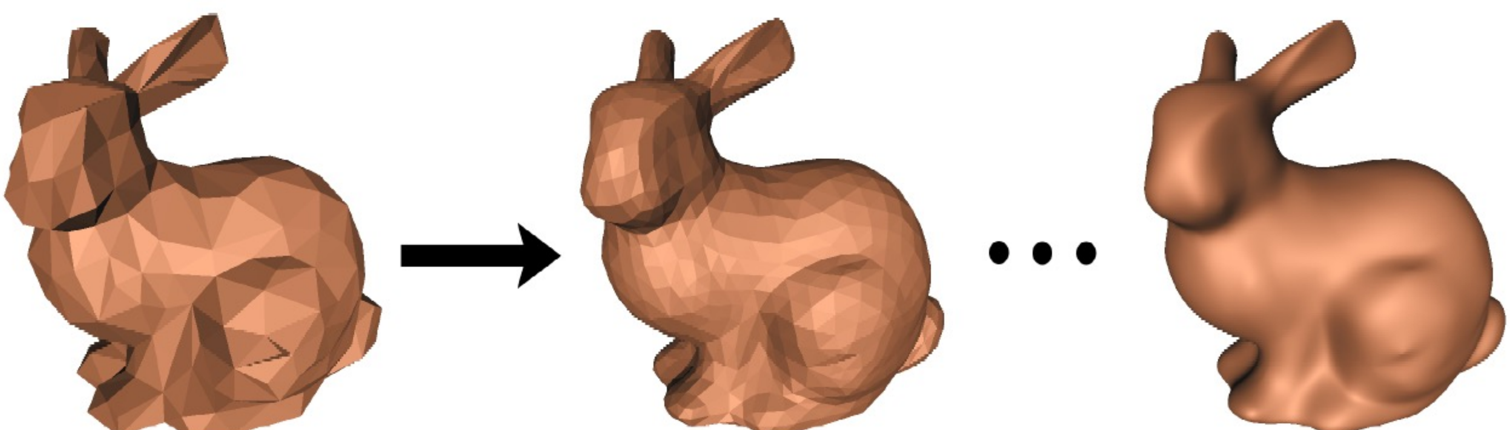

Smooth base surface Displaced Surface

zbrushcentral

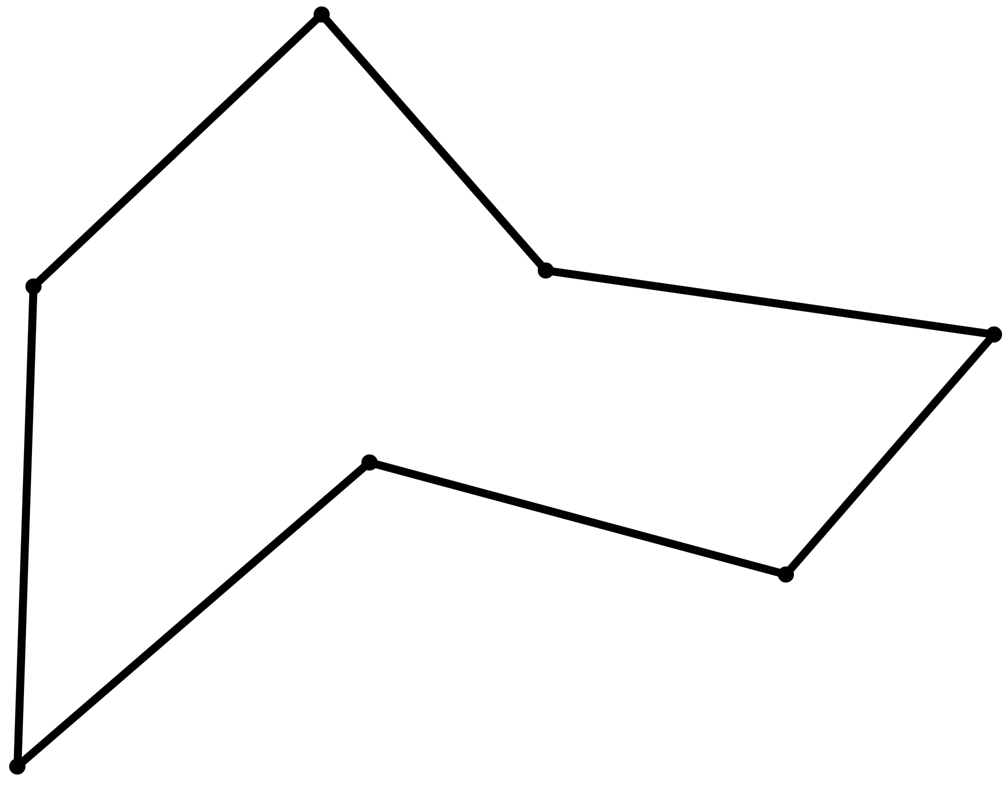
Questions?

# Subdivision Surfaces

- Start with polygonal mesh
- Subdivide into larger number of polygons, smooth result after each subdivision
  - Lots of ways to do this.
- The limit surface is smooth!

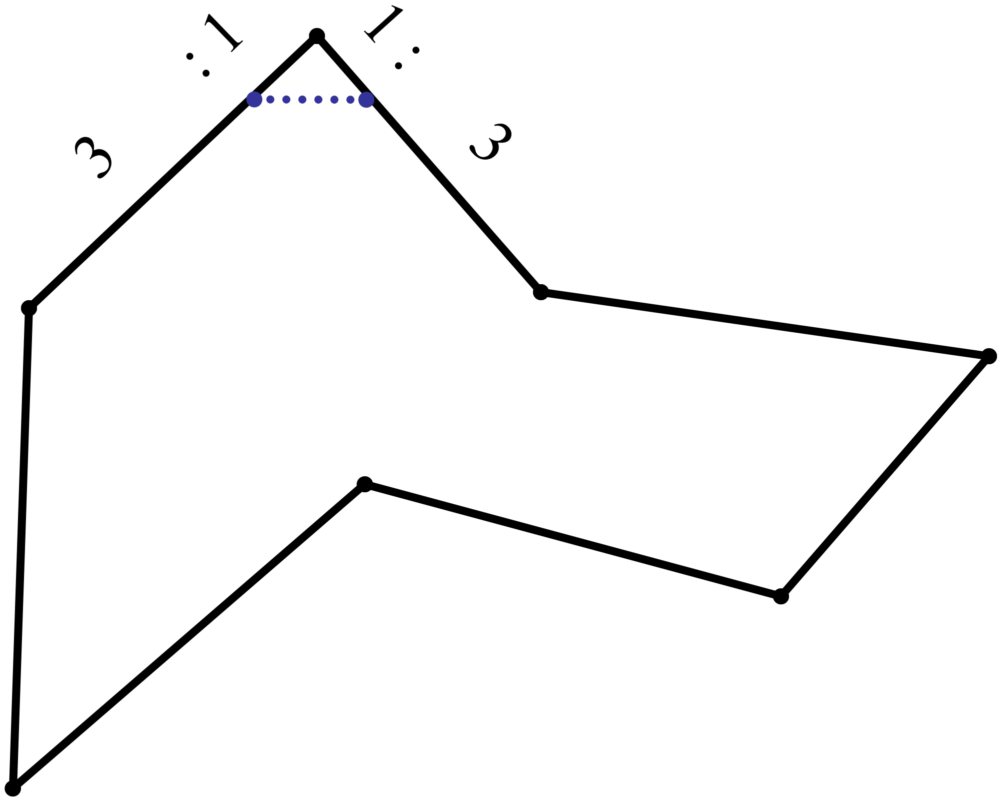


# Corner Cutting



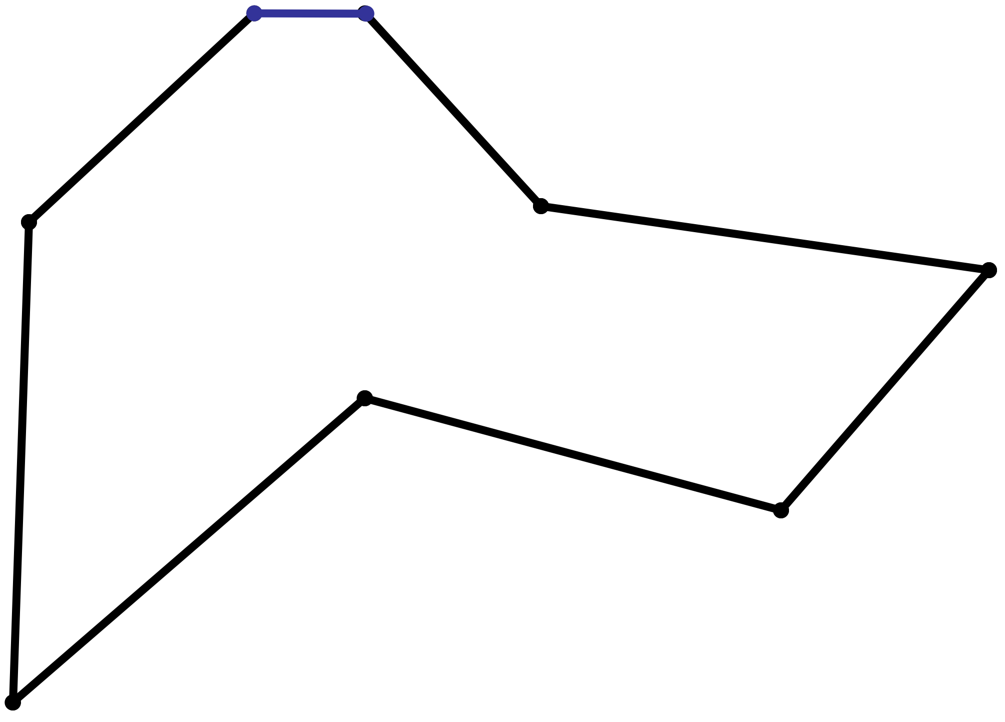
Slide by Adi Levin

# Corner Cutting



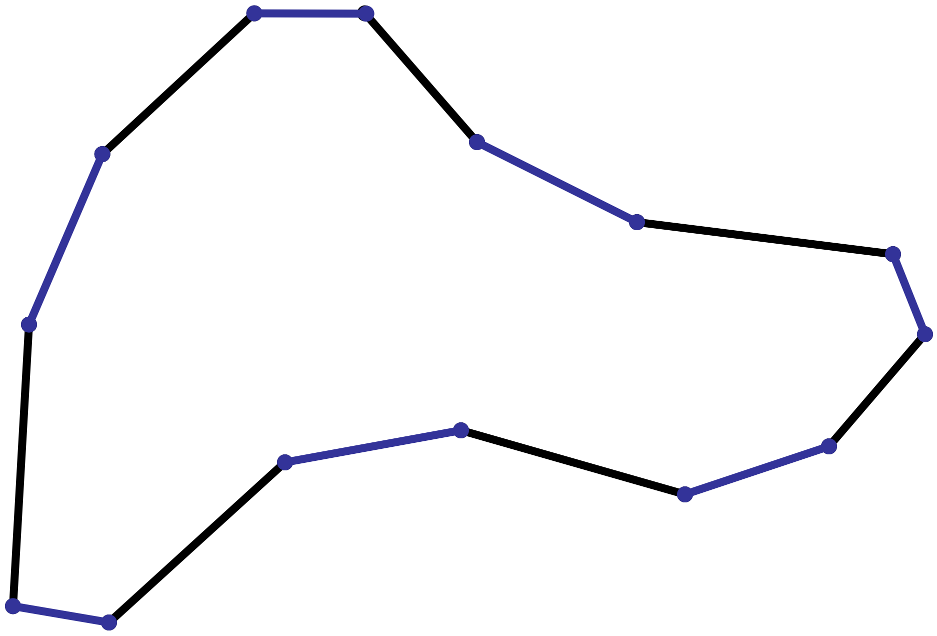

Slide by Adi Levin

# Corner Cutting



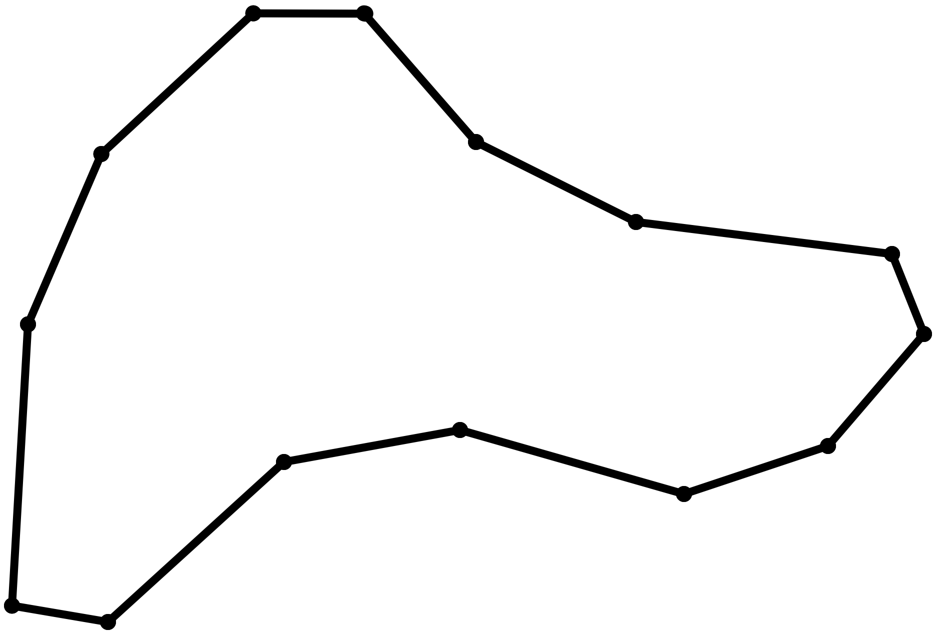
Slide by Adi Levin

# Corner Cutting



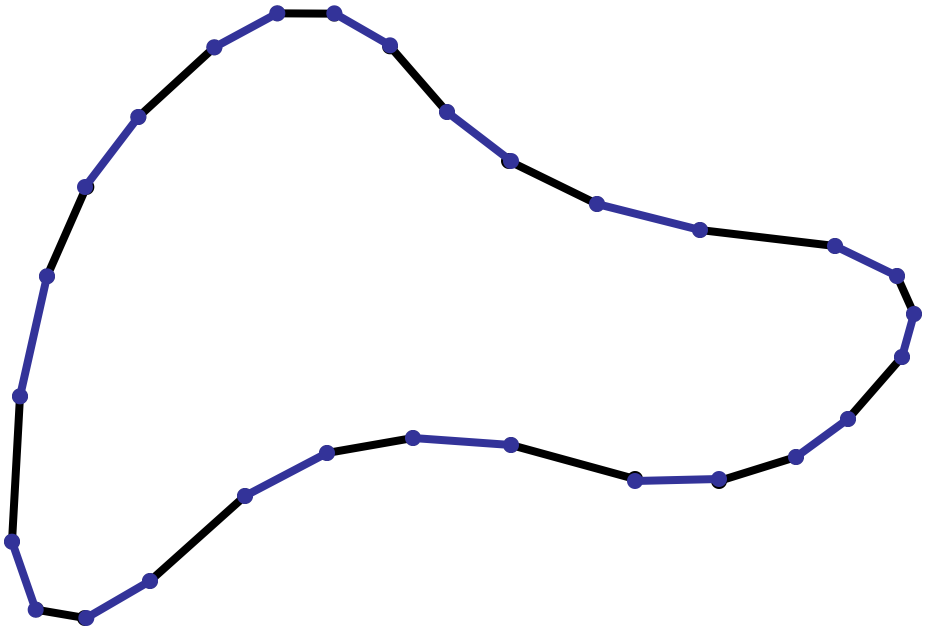
Slide by Adi Levin

# Corner Cutting



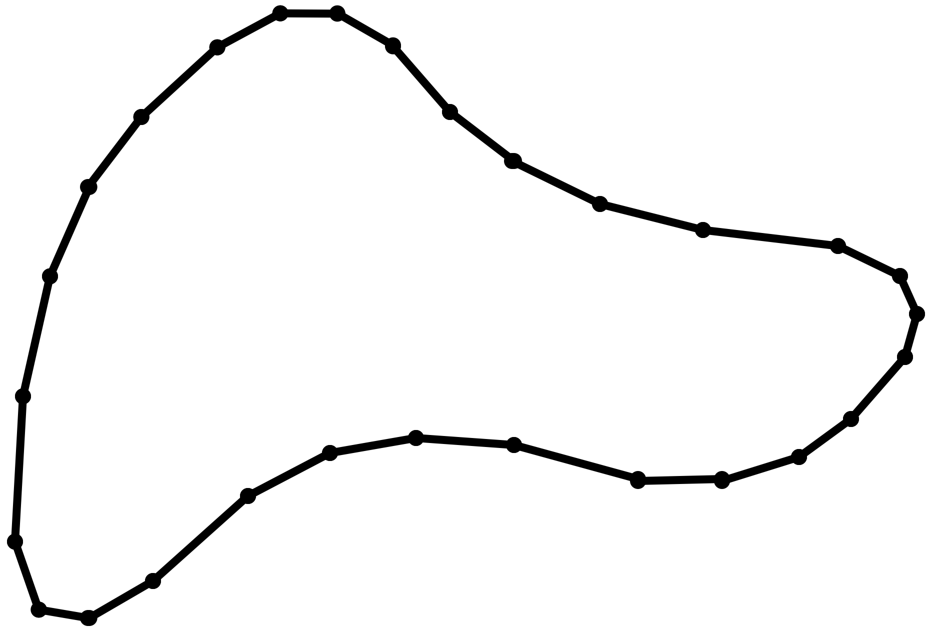
Slide by Adi Levin

# Corner Cutting



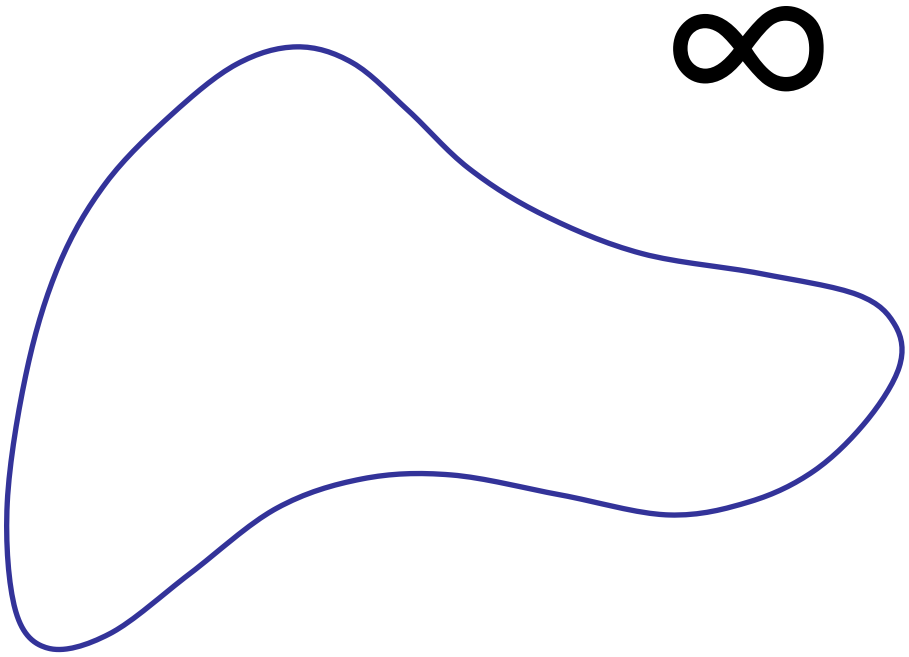
Slide by Adi Levin

# Corner Cutting



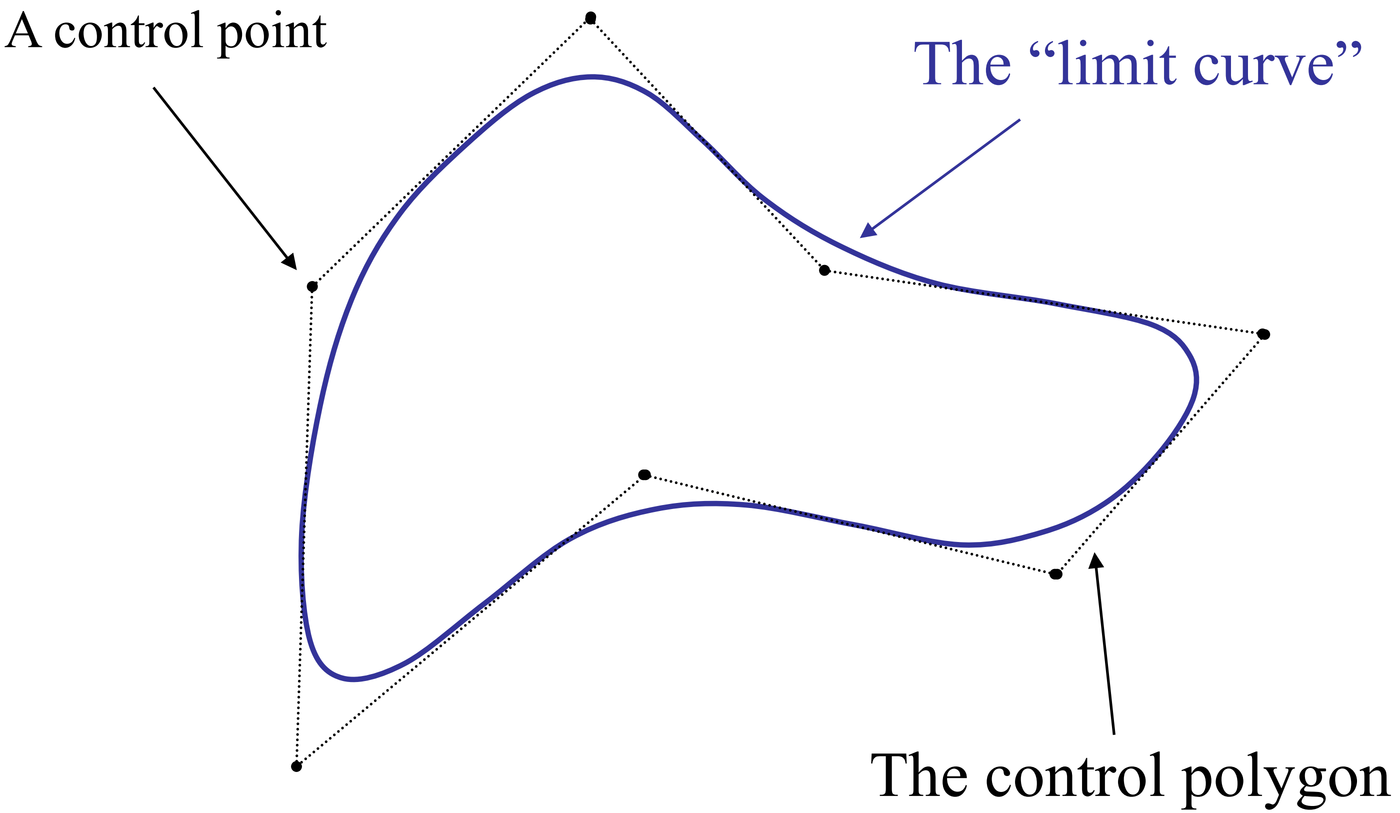
Slide by Adi Levin

# Corner Cutting




Slide by Adi Levin

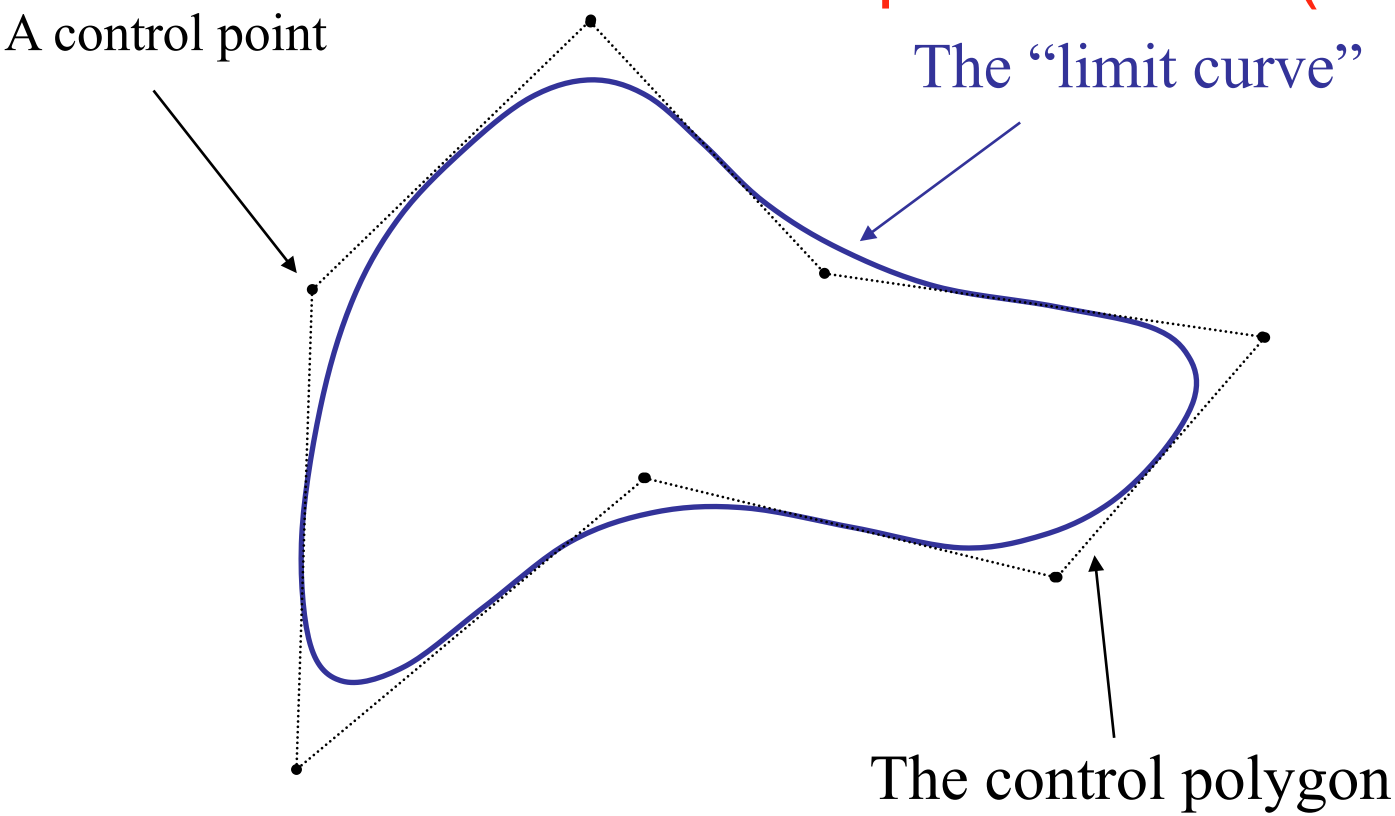
# Corner Cutting




Slide by Adi Levin

# Corner Cutting

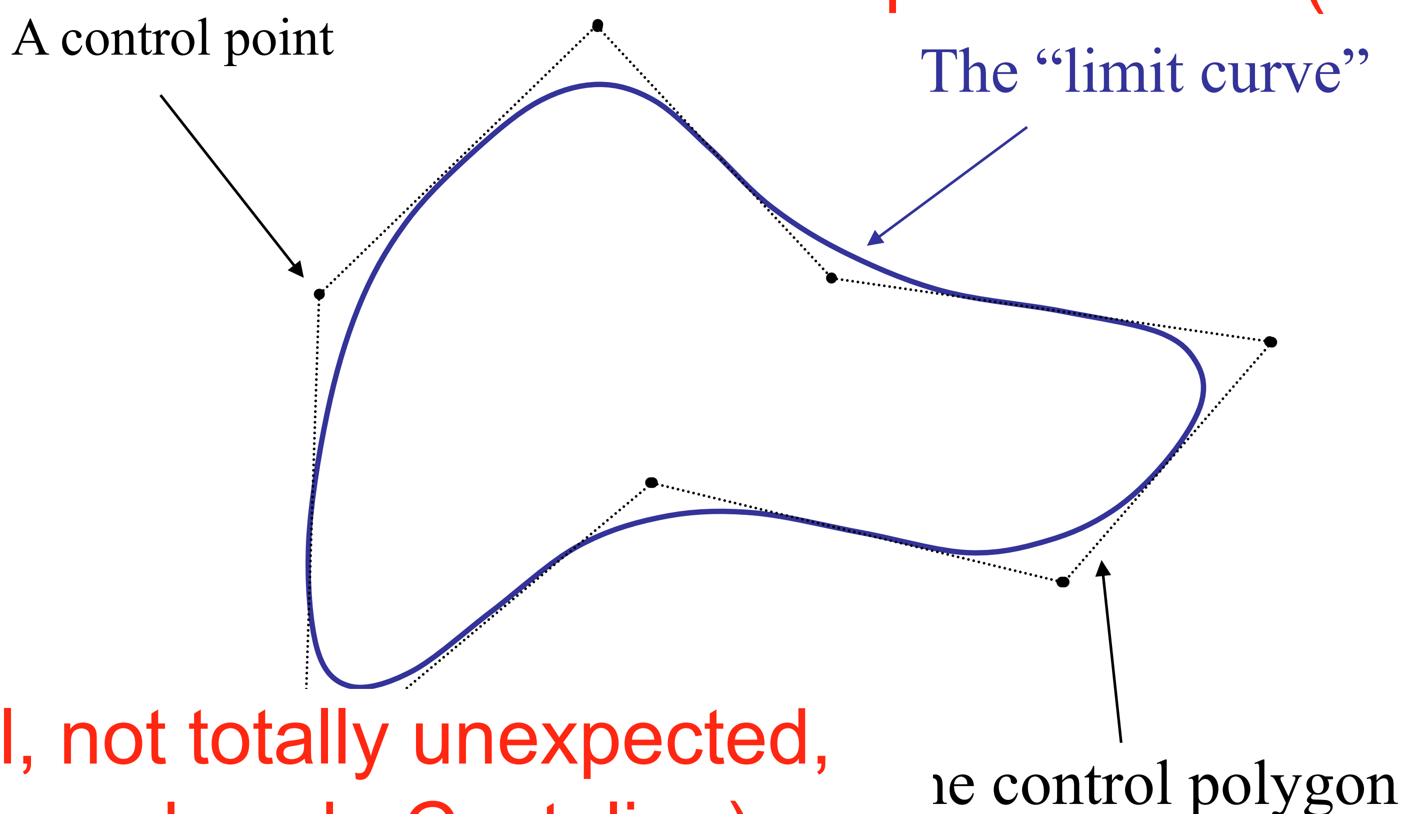
It turns out corner cutting (Chaikin's Algorithm) produces a quadratic B-Spline curve! (Magic!)




Slide by Adi Levin

# Corner Cutting

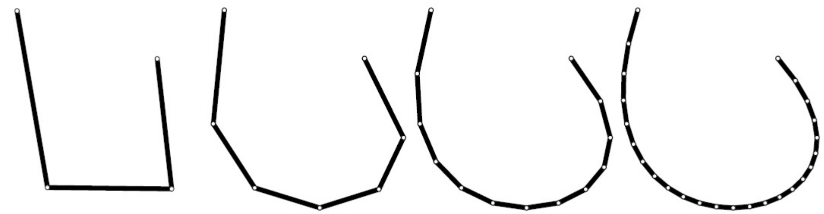
It turns out corner cutting (Chaikin's Algorithm) produces a quadratic B-Spline curve! (Magic!)



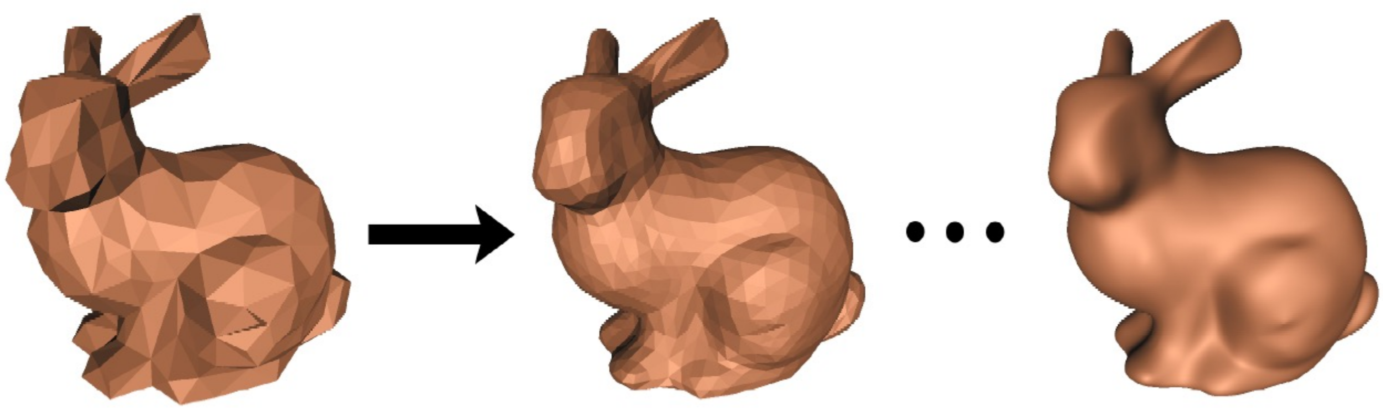

(Well, not totally unexpected, remember de Casteljau)

Slide by Adi Levin

# Subdivision Curves and Surfaces



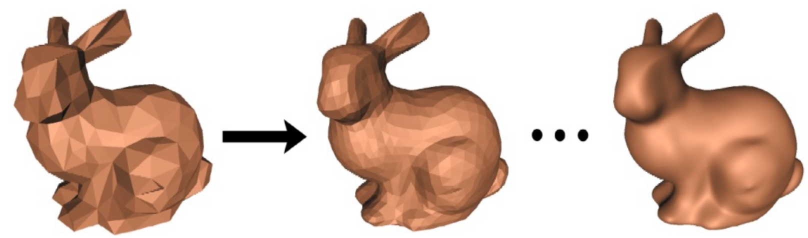
- Idea: cut corners to smooth
- Add points and compute weighted average of neighbors
- Same for surfaces
  - Special case for irregular vertices
    - vertex with more or less than 6 neighbors in a triangle mesh



Warren et al.

# Subdivision Curves and Surfaces

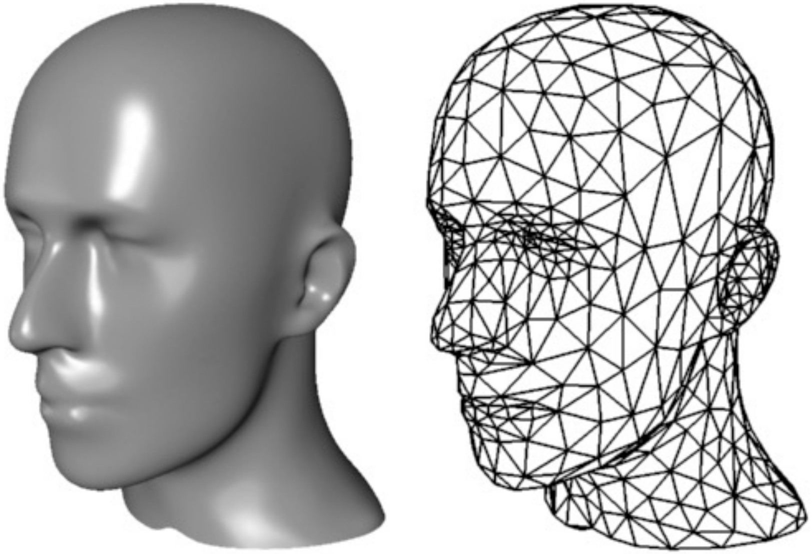
- Advantages
  - Arbitrary topology
  - Smooth at boundaries
  - Level of detail, scalable
  - Simple representation
  - Numerical stability, well-behaved meshes
  - Code simplicity
- Little disadvantage:
  - Procedural definition
  - Not parametric
  - Tricky at special vertices



Warren et al.

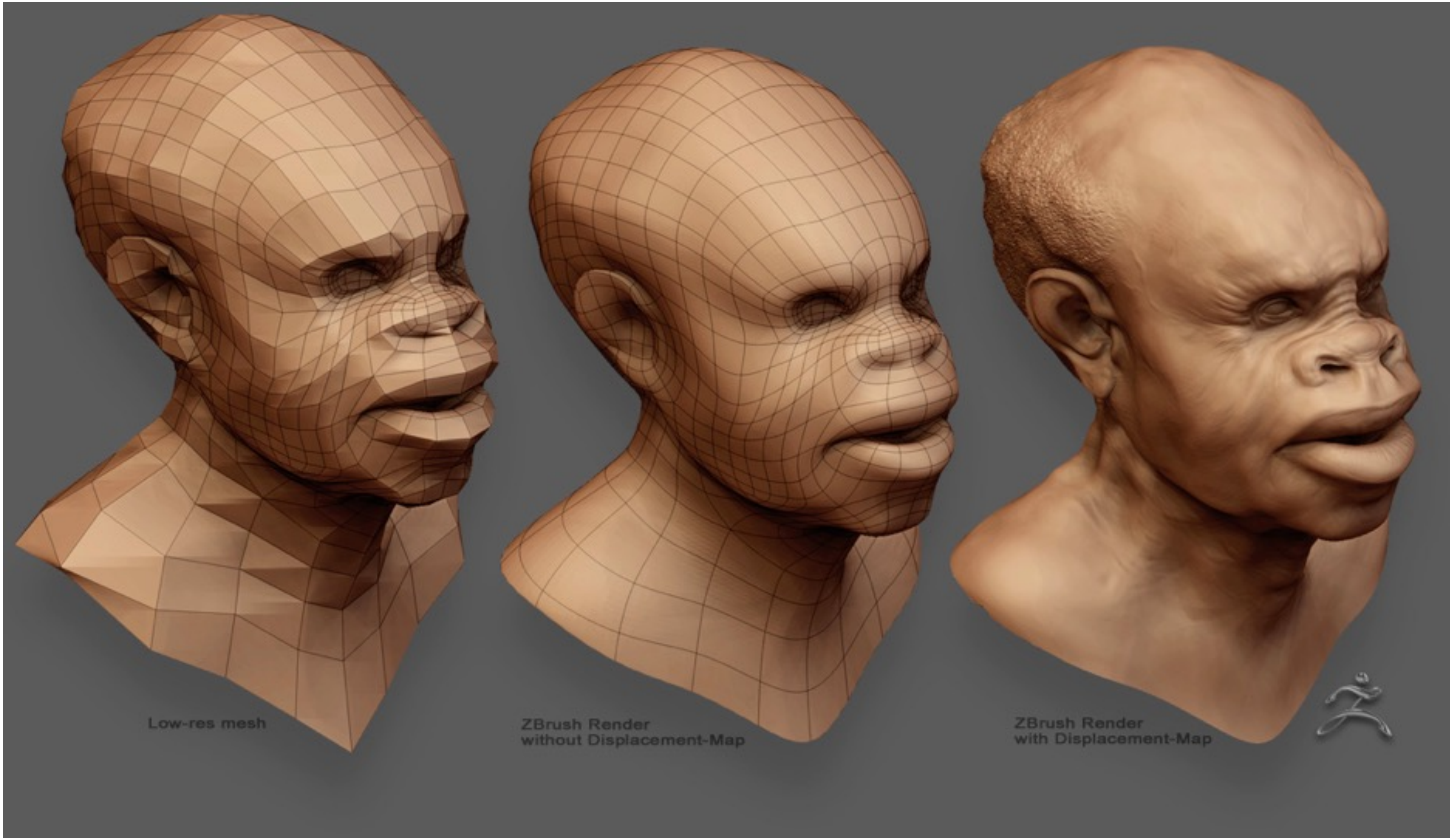
# Flavors of Subdivision Surfaces

- Catmull-Clark
  - Quads and triangles
  - Generalizes bicubics to arbitrary topology!
- Loop, Butterfly
  - Triangles
- Doo-Sabin, sqrt(3), biquartic...
  - and a whole host of others
- Used everywhere in movie and game modeling!
- See http://www.cs.nyu.edu/~dzorin/sig00course/



Leif Kobbelt

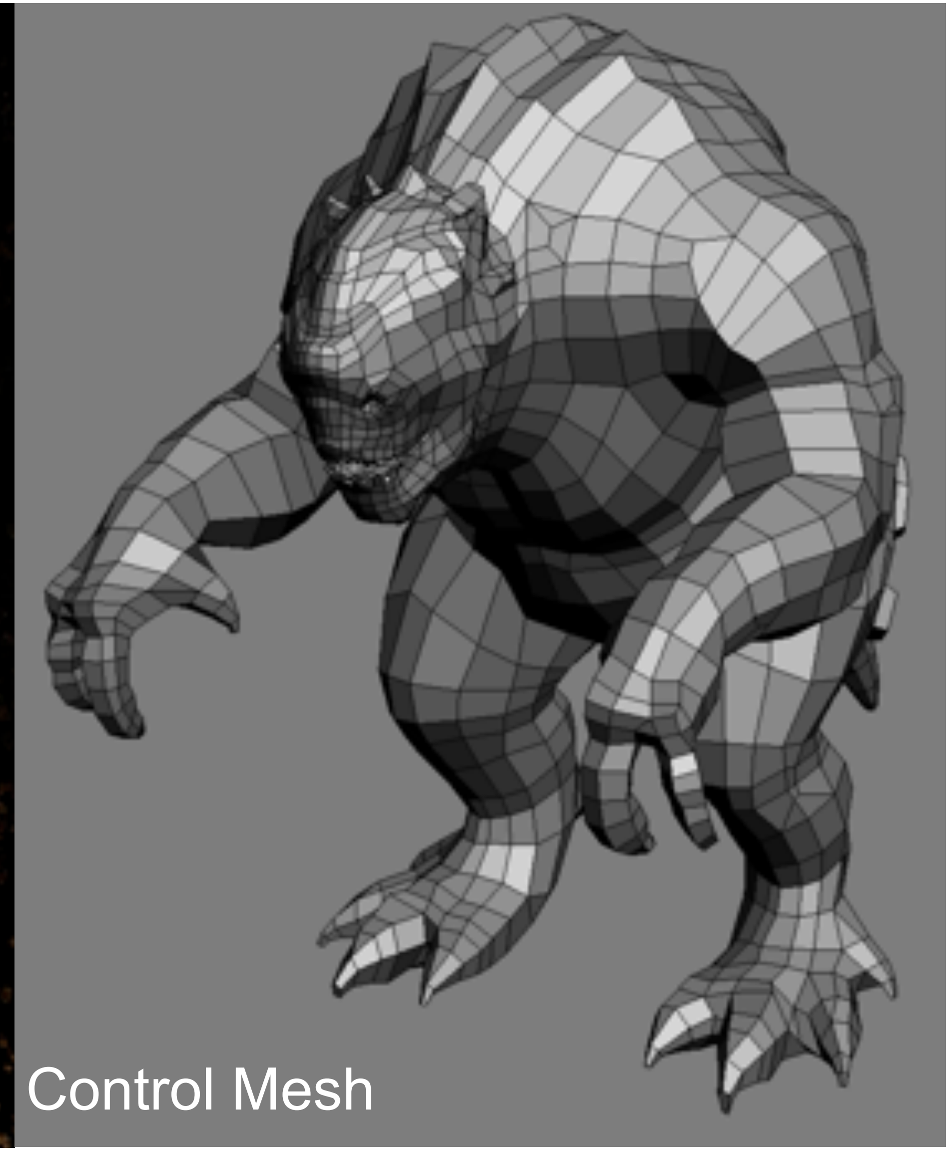
# Subdivision + Displacement




zbrushcentral

# Subdivision + Displacement

Epic Games

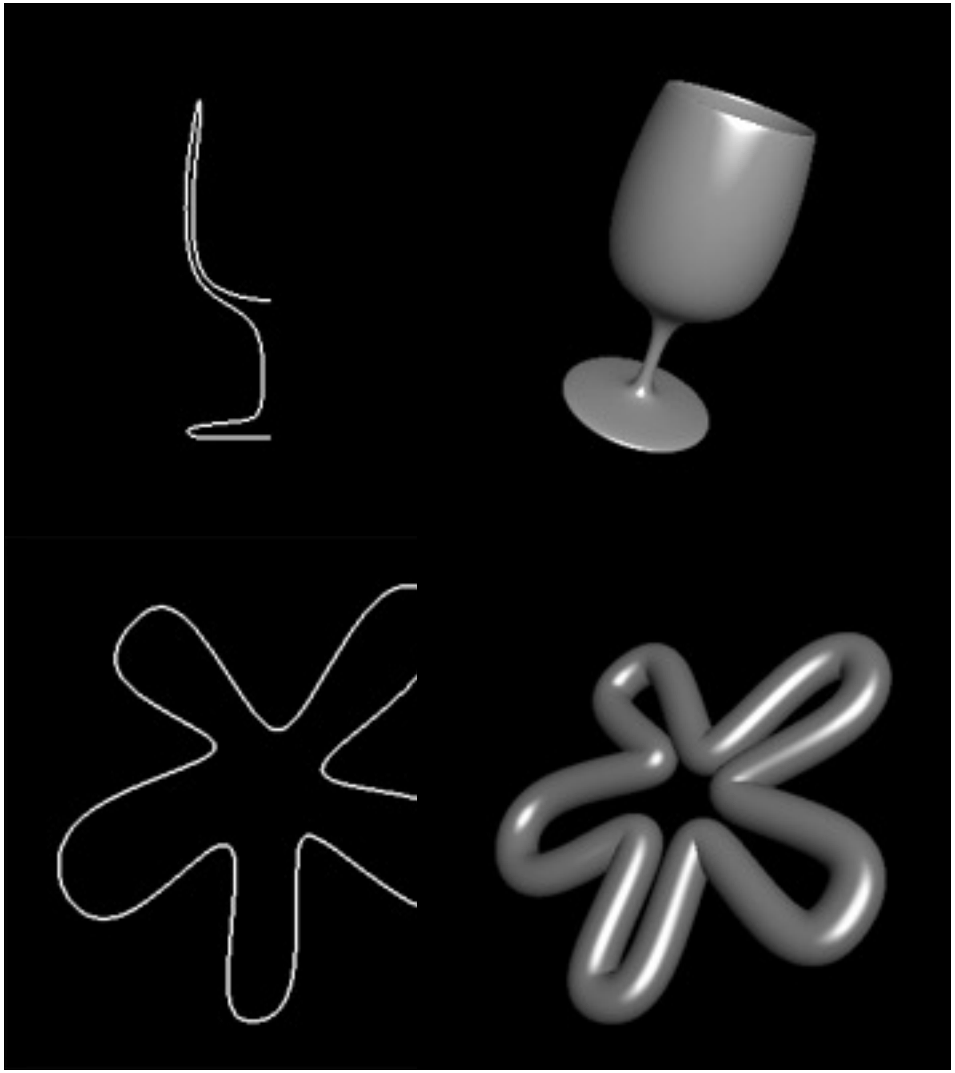
# Subdivision + Displacement

Epic Games


Questions?

# Specialized Procedural Definitions

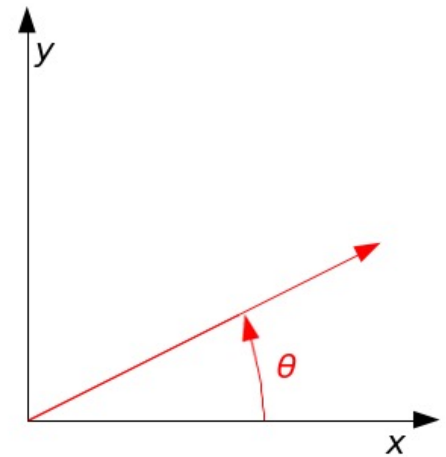
- Surfaces of revolution
  - Rotate given 2D profile curve
- Generalized cylinders
  - Given 2D profile and 3D curve, sweep the profile along the 3D curve
- Assignment 1!



# Rotation Matrix – Quick review

- 2D rotation

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$



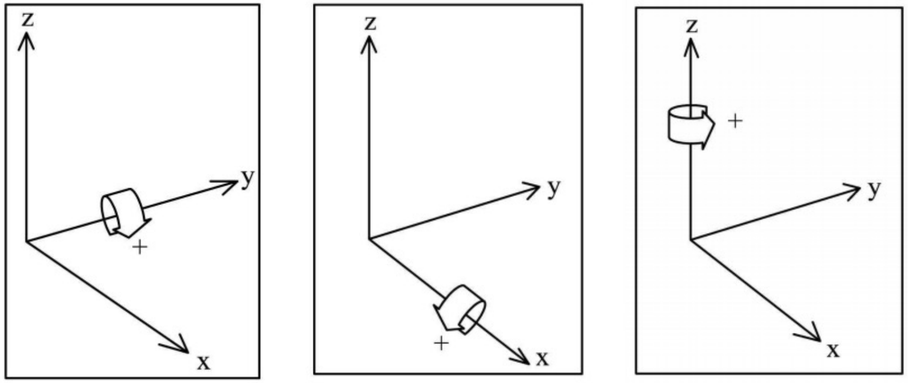

Verify

# Rotation Matrix – Quick review

- 3D Rotation about coordinate axes

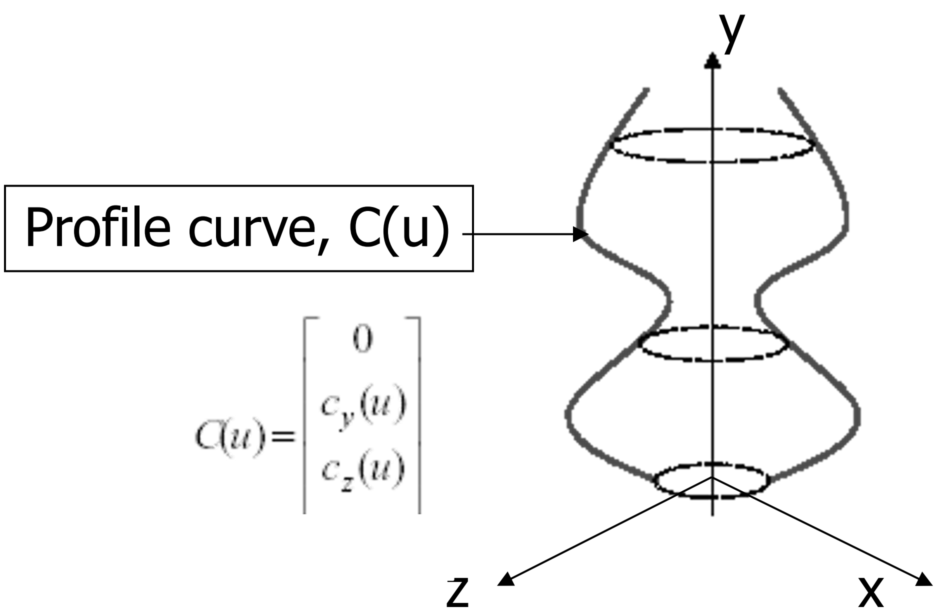
$$R_z = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \qquad R_x = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{pmatrix}$$

$$R_y = \begin{pmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix}$$



# Surface of Revolution

- Given a curve C(u) in the *yz*-plane:



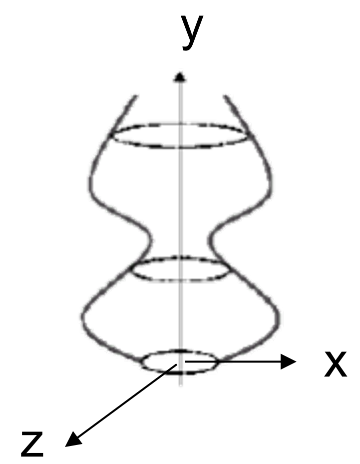

- Let $R_y(\theta)$ be a rotation about the *y*-axis
- **Find**: A surface S(u,v) obtained by applying $R_y(\theta)$ on C(u)

$$S(u,v) = R_y(v)\ [0, C_y(u), C_z(u)]^t$$

# Surface of Revolution

- Profile curve on yz plane:

$$C(u) = (\ 0,\ C_y(u),\ C_z(u)\ )$$




- The surface S(u,v) obtained by rotating C(u) about the y-axis is?

In class exercise

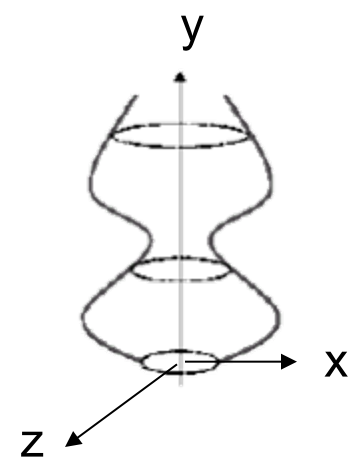
$$R_y = \begin{pmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix}$$

# Surface of Revolution

- Profile curve on yz plane:

$$C(u) = (\ 0, C_y(u), C_z(u)\ )$$

- The surface S(u,v) obtained by rotating C(u) about the y-axis is

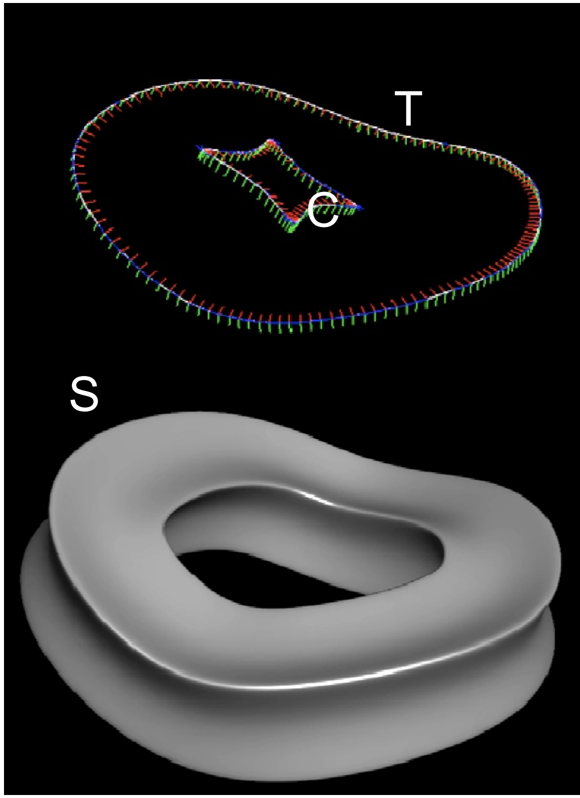



$$S(u,v) = (\ -C_z(u)\sin(v), C_y(u), C_z(u)\cos(v))^t$$

$$R_y = \begin{pmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix}$$

# General Swept Surfaces

- Trace out surface by moving a profile curve along a trajectory.
  - profile curve C(u) provides one dim
  - Trajectory T(v) provides the other
- Surface of revolution can be seen as a special case where trajectory is a circle

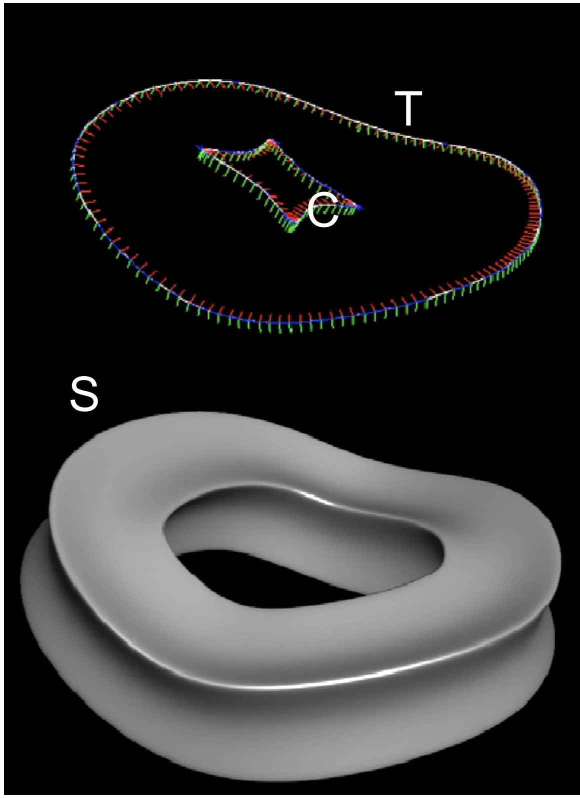



$$S(u,v)=M(T(v))C(u)$$

where M is a matrix that depends on the trajectory T

# General Swept Surfaces

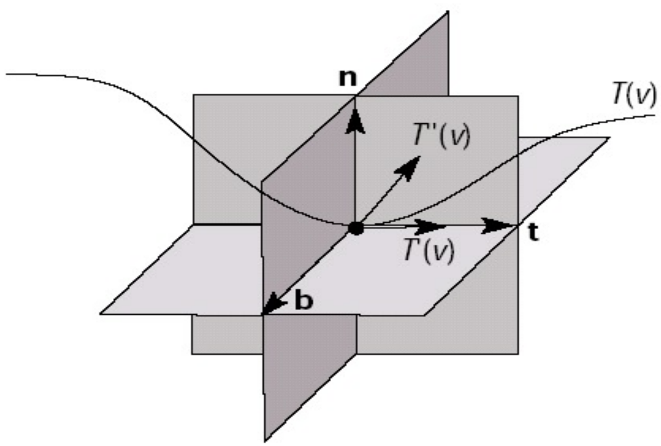
- What is the matrix M about???
  - Transformation:
    - Orientation + Translation
  - Translation is easy, given T(v)
  - What about Orientation??
    - Align profile curve with frame that "follows" the trajectory curve




***Note: C(u) is planar, T(v) is a space curve

# Orientating C(u)

- Define a local coordinate frame at any point along the trajectory
- The Frenet frame (b,n,t)

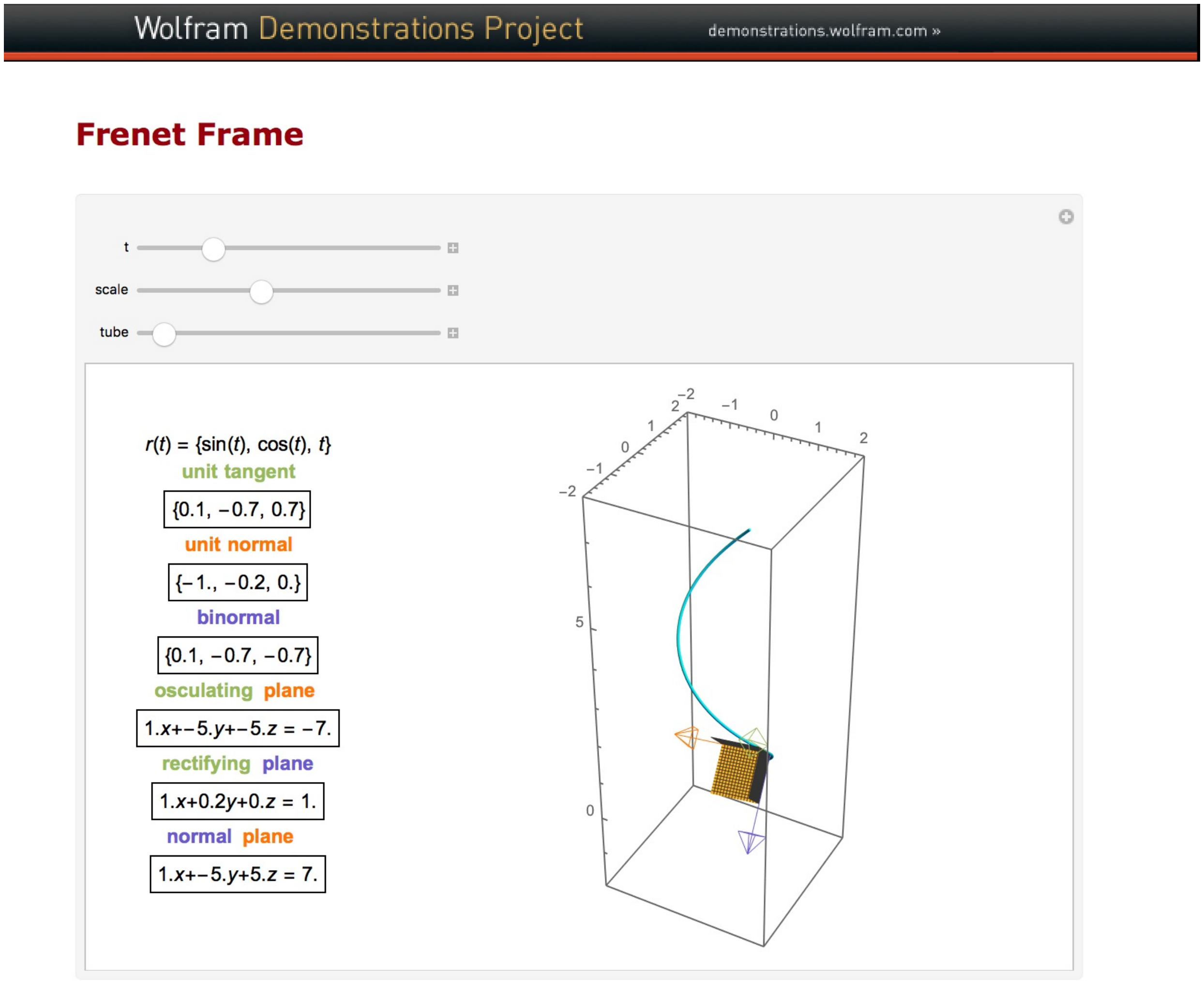



$$\mathbf{t}(v) = \text{normalize}[T'(v)]$$
$$\mathbf{b}(v) = \text{normalize}[T'(v) \times T''(v)]$$
$$\mathbf{n}(v) = \mathbf{b}(v) \times \mathbf{t}(v)$$

- As we move along T(v), the Frenet frame (b,n,t) varies smoothly (inflection points where curvature goes to zero needs special treatment)

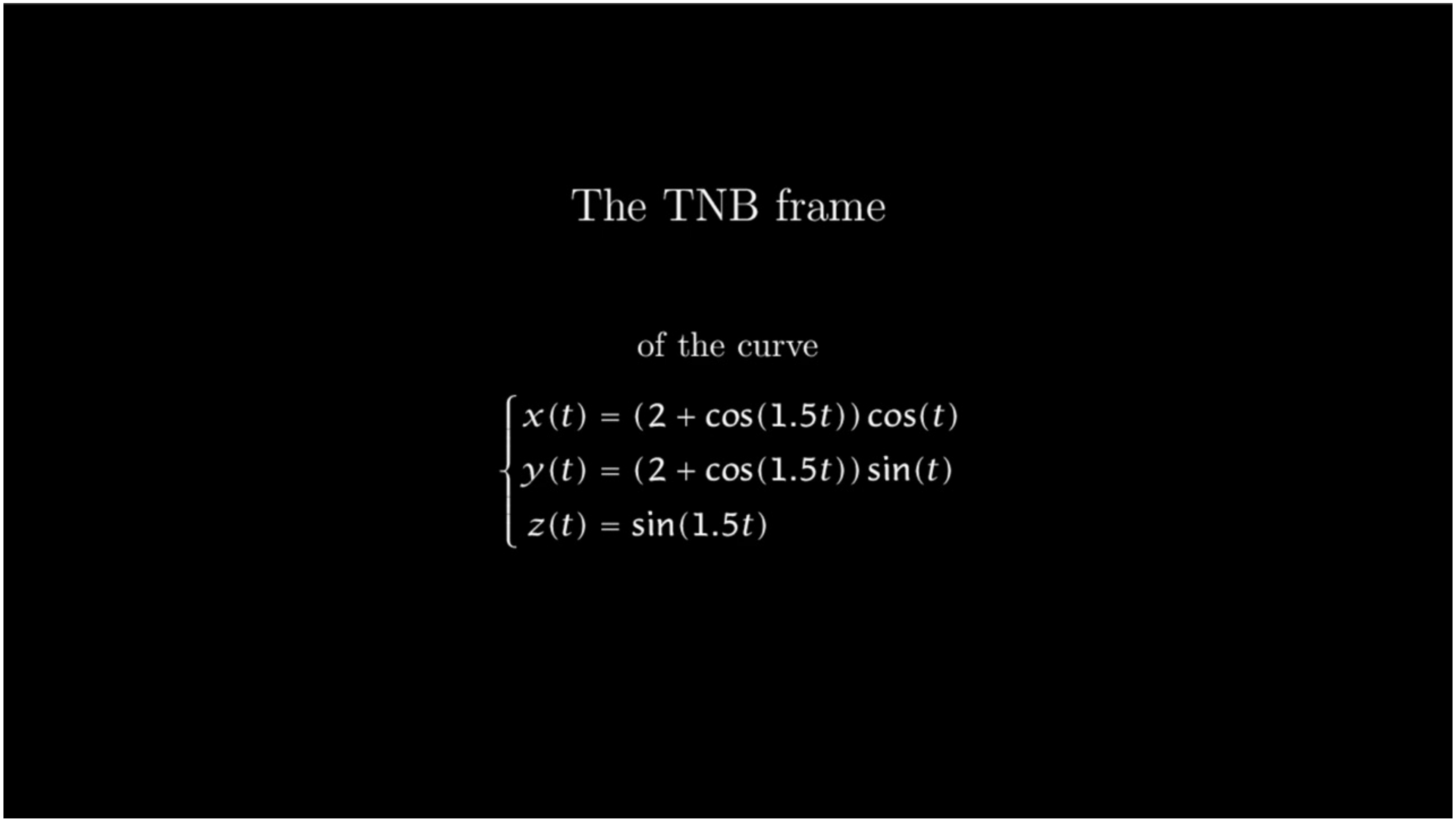
# Demo

- http://demonstrations.wolfram.com/FrenetFrame/



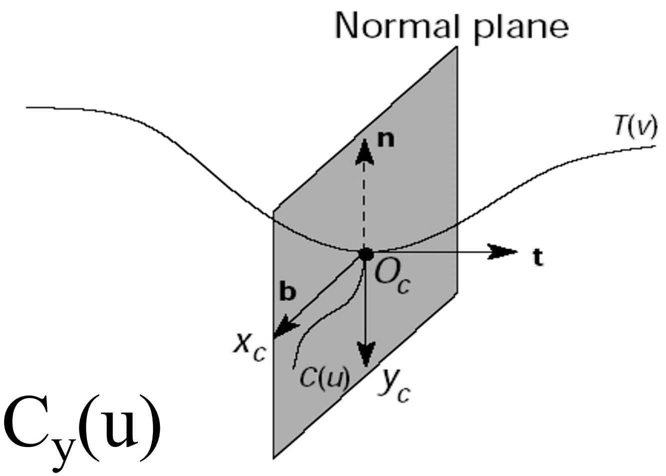
# Demo

- https://www.youtube.com/watch?v=JZGFcwipHYY



# Sweep Surfaces

- Orient the profile curve C(u) using the Frenet frame of the trajectory T(v):
  - Put C(u) in the normal plane.
  - Place $O_c$ at T(v).
  - Align $x_c$ of C(u) with b.
  - Align $y_c$ of C(u) with n.



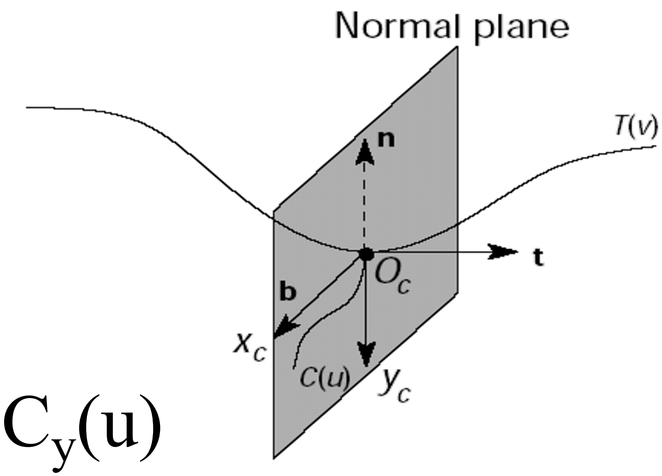

- Sweep surface:
  - $S(u,v) = T(v) + b(v)C_x(u) + n(v)\ C_y(u)$

Coordinates of the trajectory curve at v

b(v), n(v) are unit vector

# Sweep Surfaces

- Orient the profile curve C(u) using the Frenet frame of the trajectory T(v):
  - Put C(u) in the normal plane.
  - Place $O_c$ at T(v).
  - Align $x_c$ of C(u) with b.
  - Align $y_c$ of C(u) with n.




- Sweep surface:
  - $S(u,v) = T(v) + b(v)C_x(u) + n(v)\ C_y(u)$

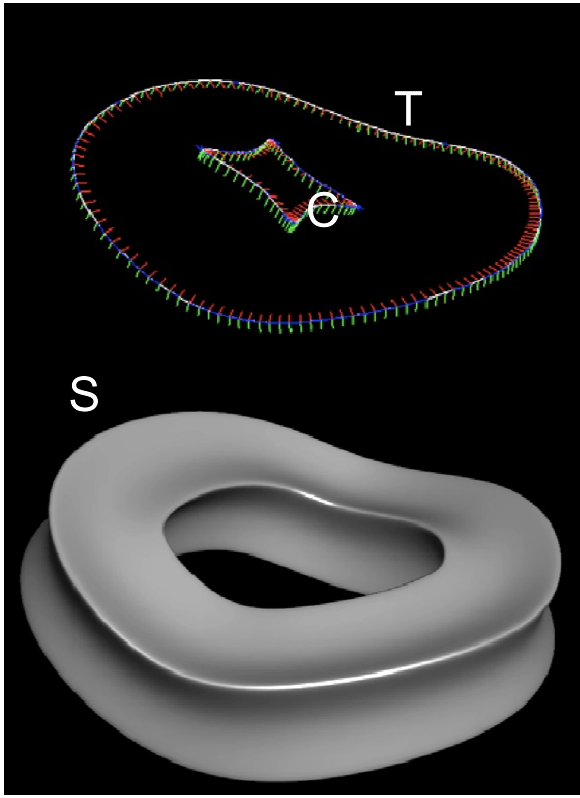
$S(u,v) = [b(v)\ n(v)\ t(v)]\ [C_x(u)\ C_y(u)\ 0]' + T(v)$

$\Rightarrow R(T(v))C(u) + T(v)$

M is related to R and T (next class!)

# General Swept Surfaces

- Trace out surface by moving a profile curve along a trajectory.
  - profile curve C(u) provides one dim
  - Trajectory T(v) provides the other
- Surface of revolution can be seen as a special case where trajectory is a circle




$$S(u,v)=M(T(v))C(u)$$

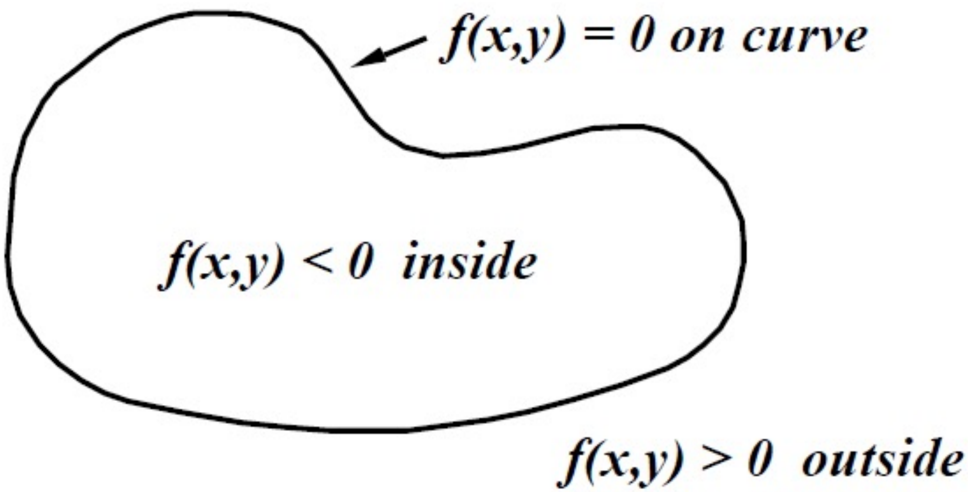
where M is a matrix that depends on the trajectory C

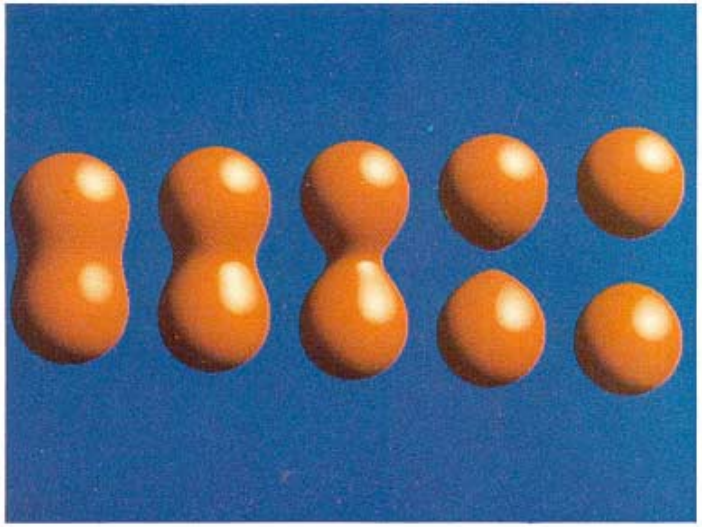
Questions?

# Implicit Surfaces

- Surface defined implicitly by a function

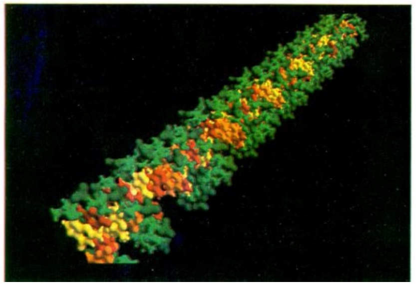
$f(x, y, z) = 0$ (on surface)
$f(x, y, z) < 0$ (inside)
$f(x, y, z) > 0$ (outside)



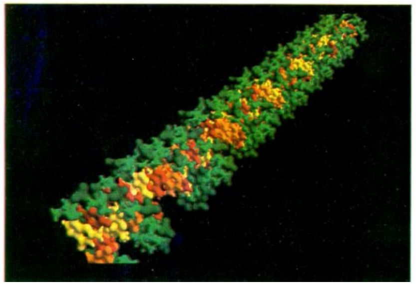
From Blinn 1982

# Implicit Surfaces



- Pros:
  - Efficient check whether  point is inside
  - Efficient Boolean operations
  - Can handle weird topology for animation
  - Easy to do sketchy modeling
- Cons:
  - Does not allow us to easily generate a point on the surface

# Implicit Surfaces

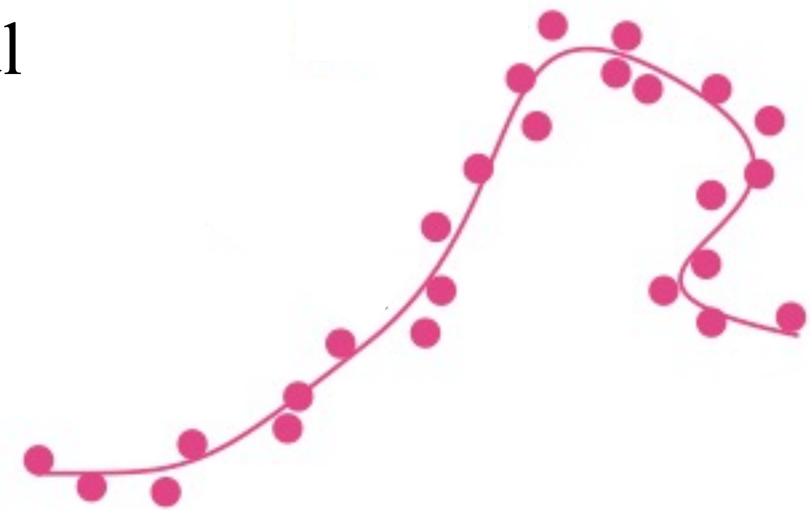


- Pros:
  – Efficient check whether  point is inside
  – Efficient Boolean operations
  – Can handle weird topology for animation
  – Easy to do sketchy modeling
- Cons:
  – Does not allow us to easily generate a point on the surface

Questions?

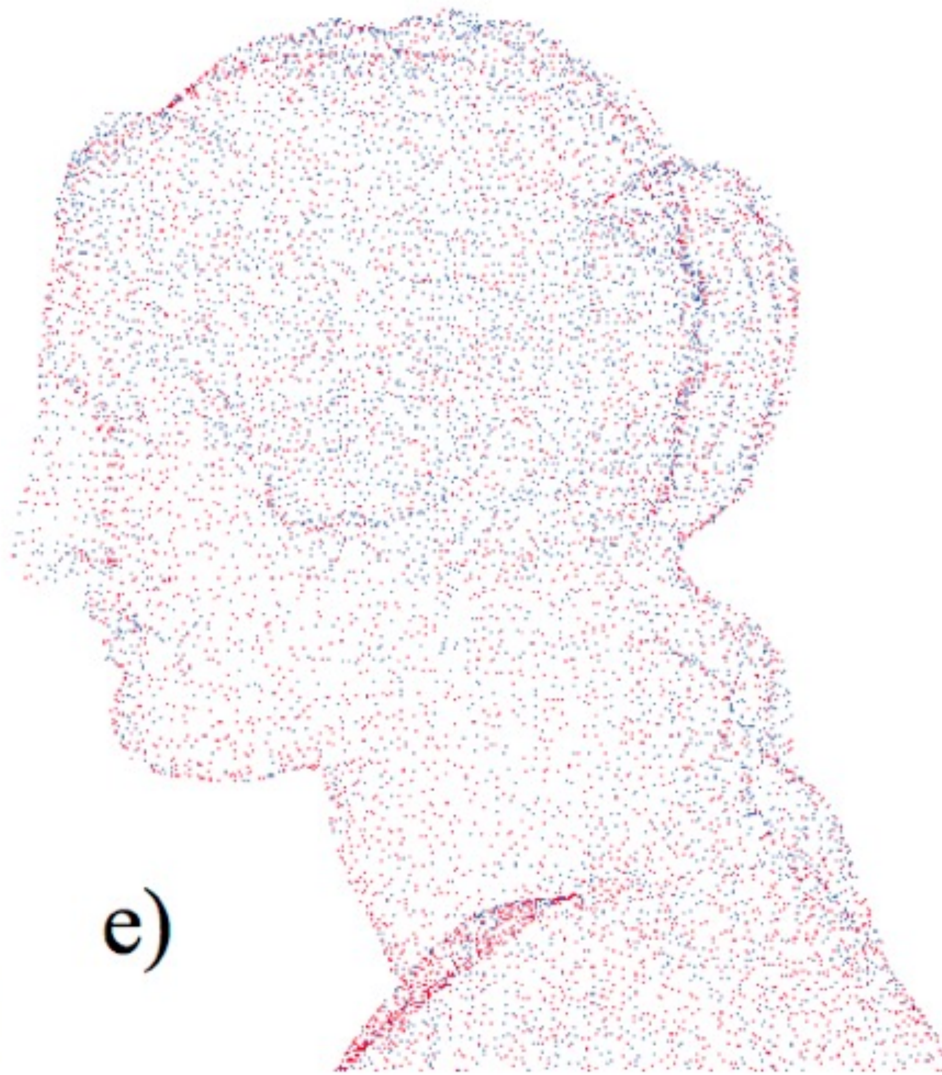
# Point Set Surfaces

- Given only a noisy 3D point cloud (no connectivity), can you define a reasonable surface using only the points?
  - Laser range scans only give you points, so this is potentially useful



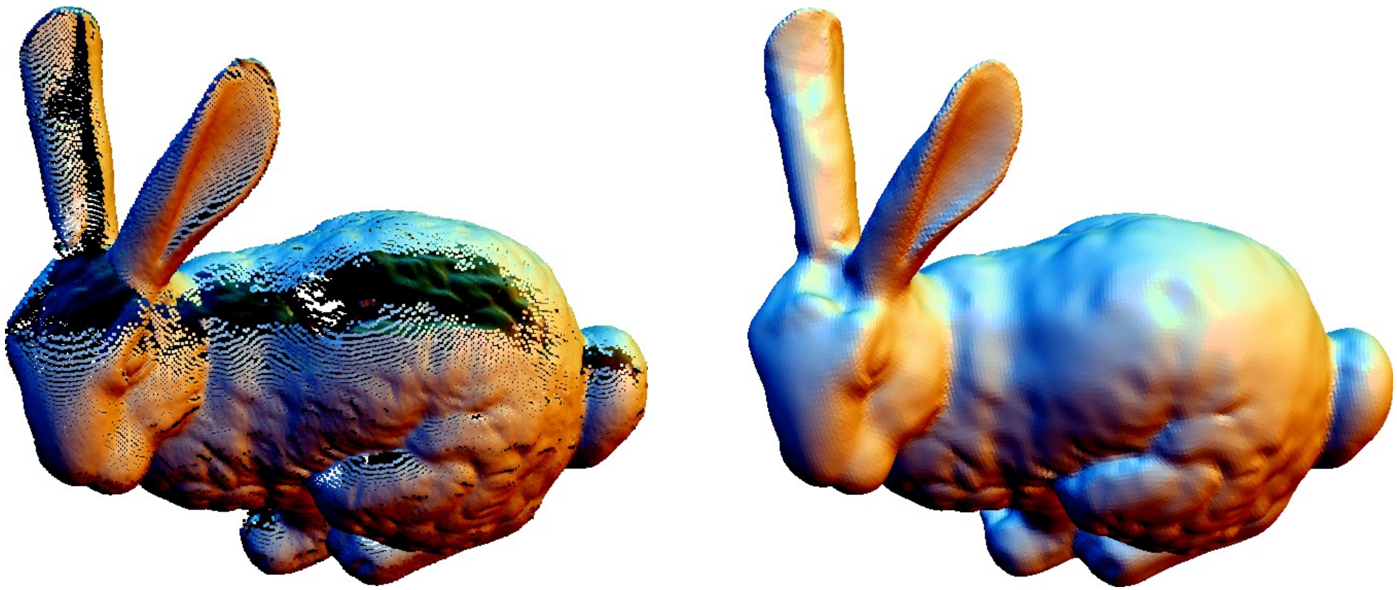
# Point Set Surfaces

Alexa et al. 2001

# Point Set Surfaces

- Modern take on implicit surfaces
- Cool math: Moving Least Squares (MLS), partitions of unity, etc.

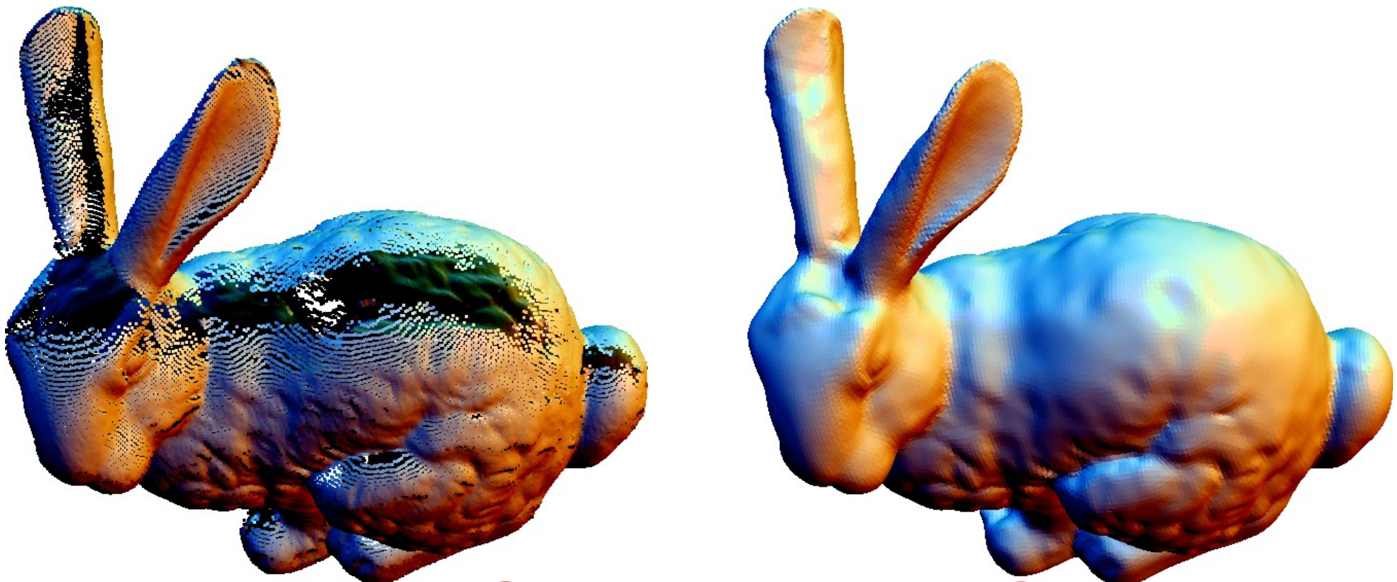


Ohtake et al. 2003

- Not required in this class, but nice to know.

# Point Set Surfaces

- Modern take on implicit surfaces
- Cool math: Moving Least Squares (MLS), partitions of unity, etc.



Ohtake et al. 2003

Questions?

- Not required in this class, but nice to know.

# That’s All for Today

- Further reading
  - Buss, Chapters 7 & 8
- Subvision curves and surfaces
  - http://www.cs.nyu.edu/~dzorin/sig00course/