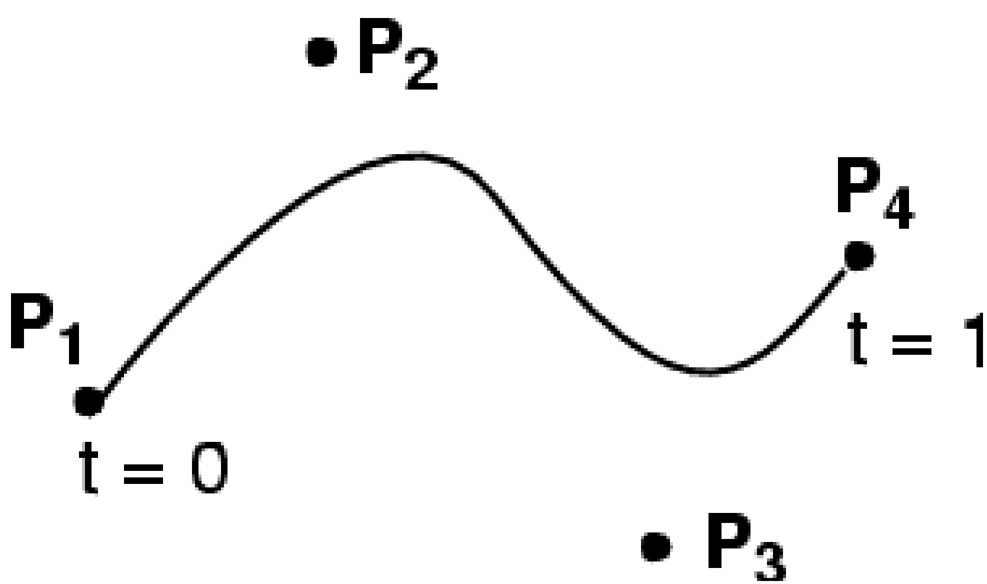
6.837 Computer Graphics

Curve Properties & Conversion, Surface Representations

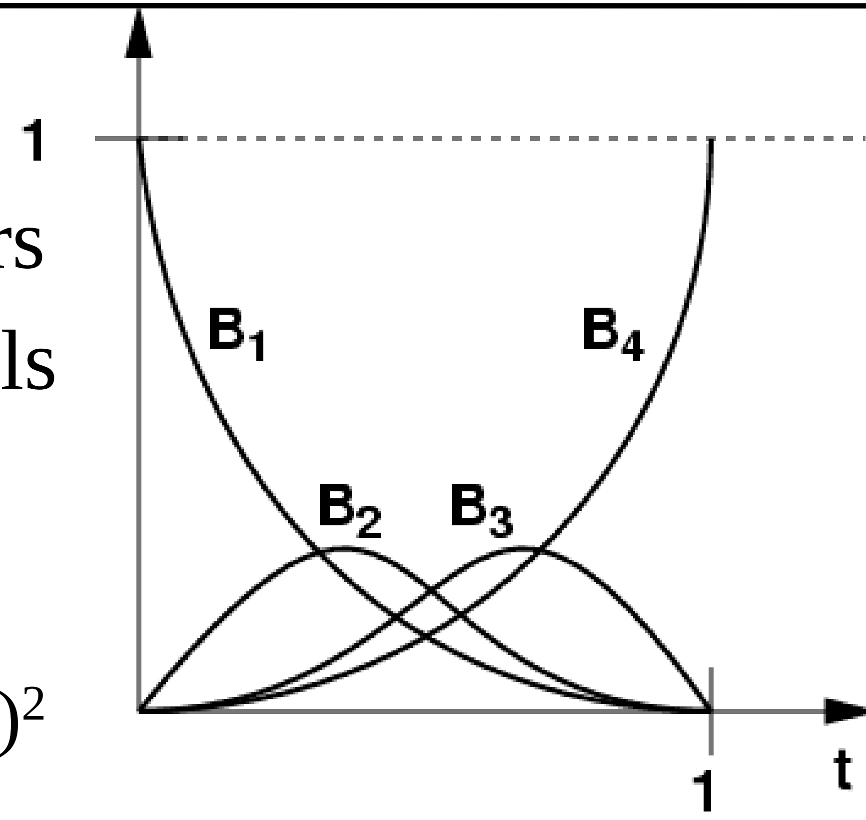
Cubic Bezier Splines

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



Bernstein Polynomials

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



- For cubic Bezier curve:

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

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

  (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:

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

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

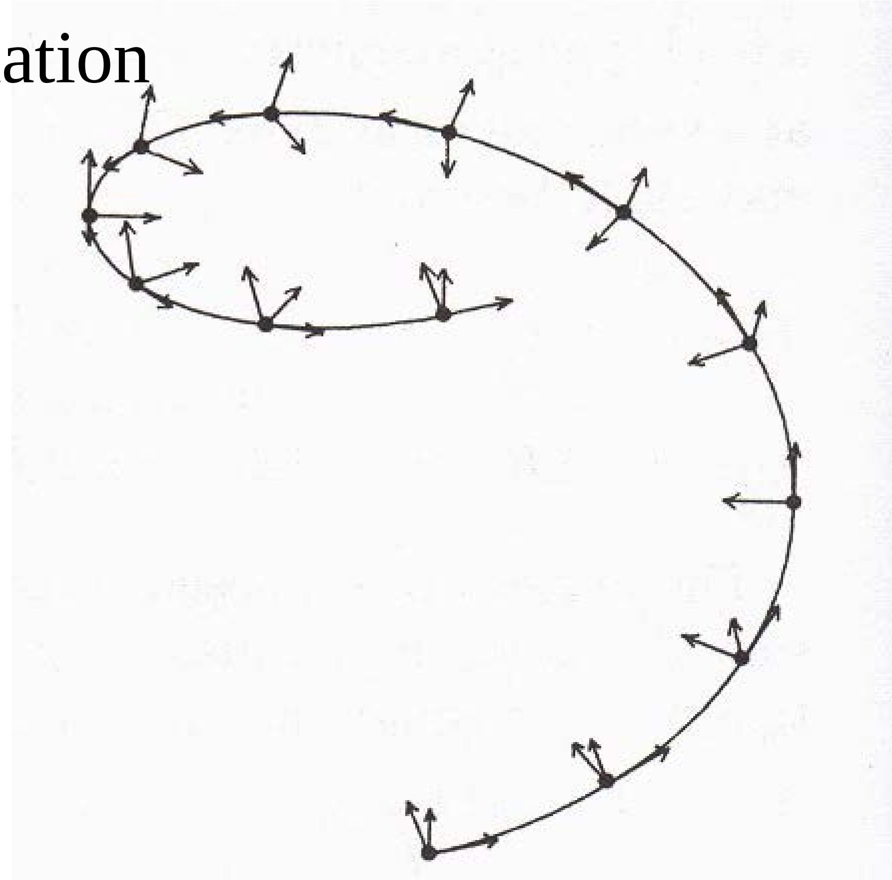
Questions?

The Plan for Today

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

Differential Properties of Curves

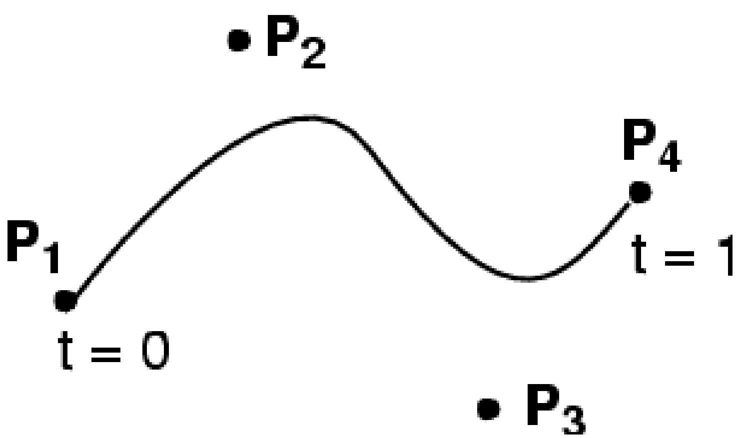
- Motivation
  - Compute normal for surfaces
  - Compute velocity for animation
  - Analyze smoothness



Velocity

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

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

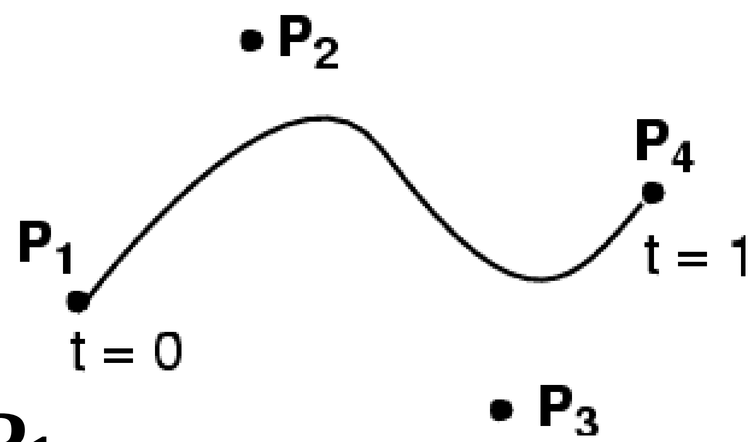


- You know how to differentiate polynomials...

Velocity

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

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



- $$
\begin{aligned}
\mathbf{P'(t)} = \quad & \mathbf{-3(1-t)^2} && \mathbf{P_1} \\
+ \quad & \mathbf{[3(1-t)^2 - 6t(1-t)]} && \mathbf{P_2} \\
+ \quad & \mathbf{[6t(1-t) - 3t^2]} && \mathbf{P_3} \\
+ \quad & \mathbf{3t^2} && \mathbf{P_4}
\end{aligned}
$$

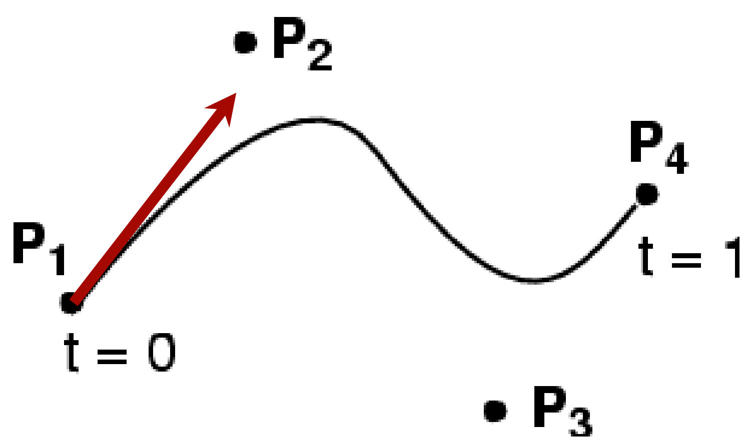
Sanity check: t=0; t=1

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.
- Can be done with matrices
  - Trivial in monomial basis
  - But get lower-order polynomials

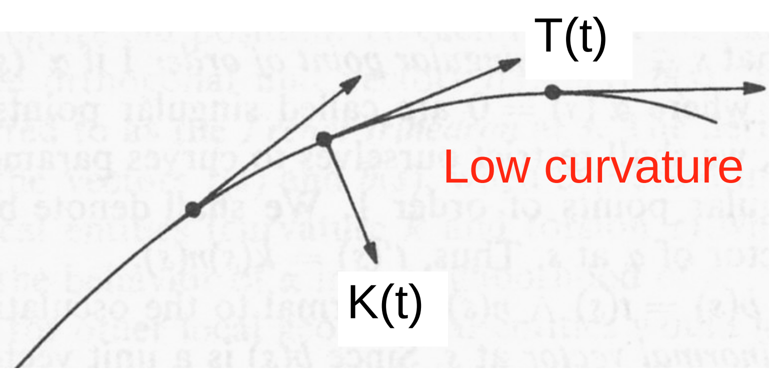
Tangent Vector

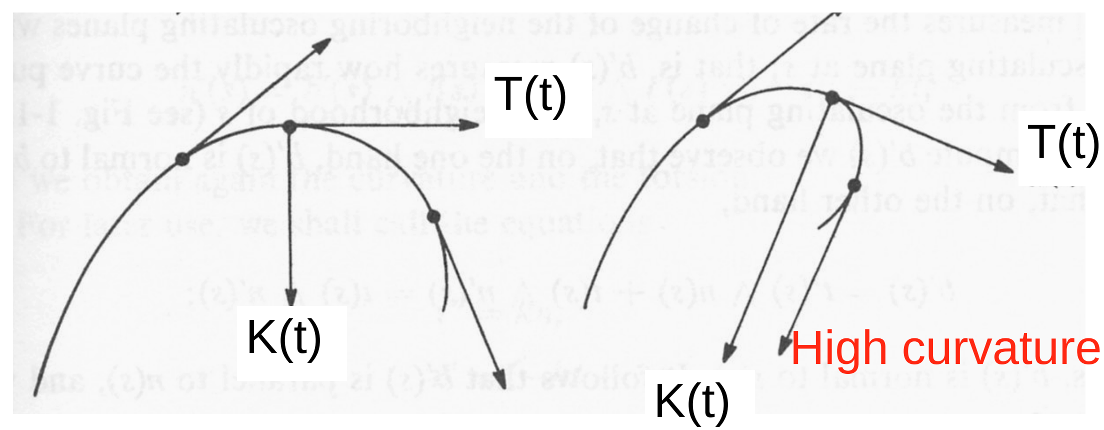
- The tangent to the curve P(t) can be defined as $T(t)=P'(t)/||P'(t)||$
  – normalized velocity, $||T(t)|| = 1$
- This provides us with one orientation for swept surfaces later



Curvature Vector

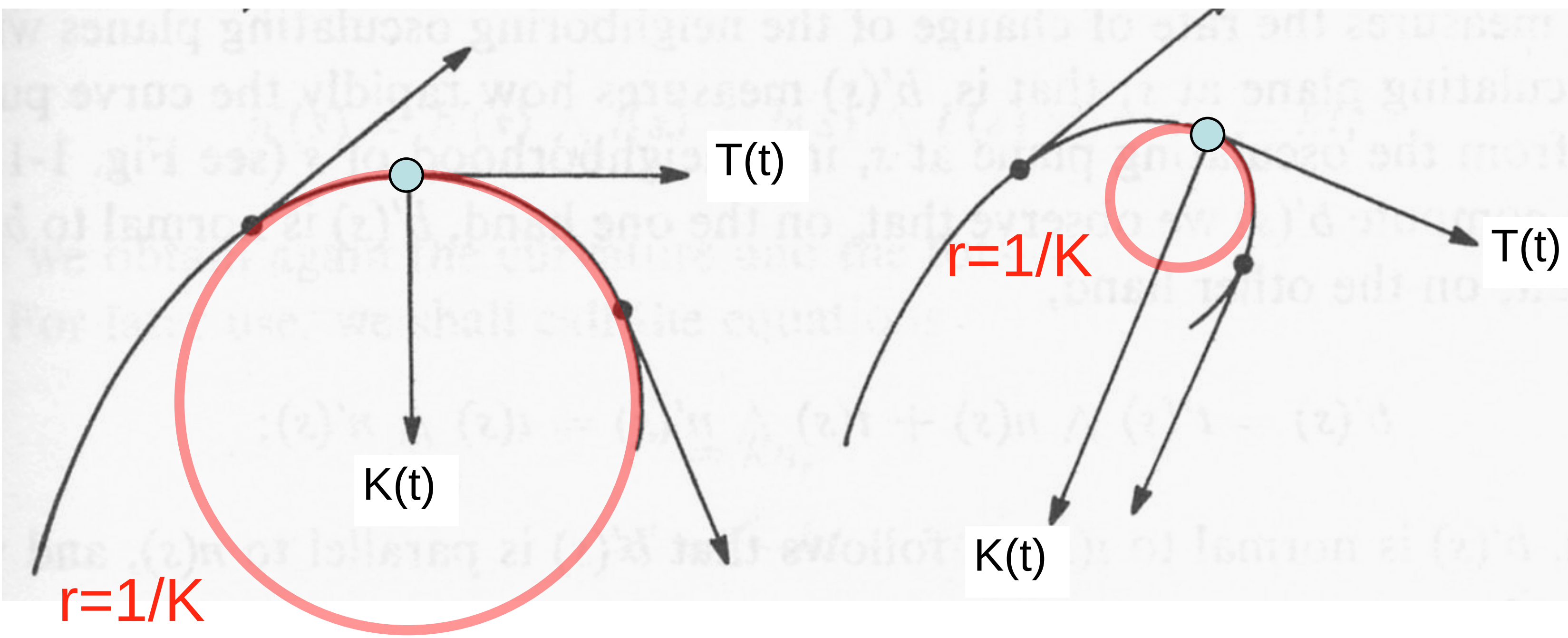
- Derivative of unit tangent
  - K(t)=T'(t)
  - Magnitude ||K(t)|| is constant for a circle
  - Zero for a straight line
- Always orthogonal to tangent, ie. $K \cdot T = 0$





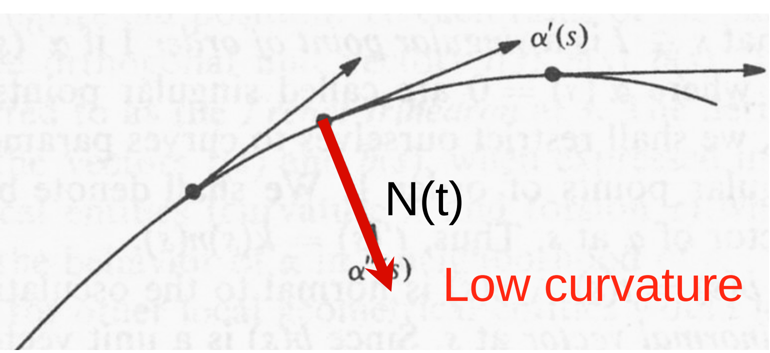
Geometric Interpretation

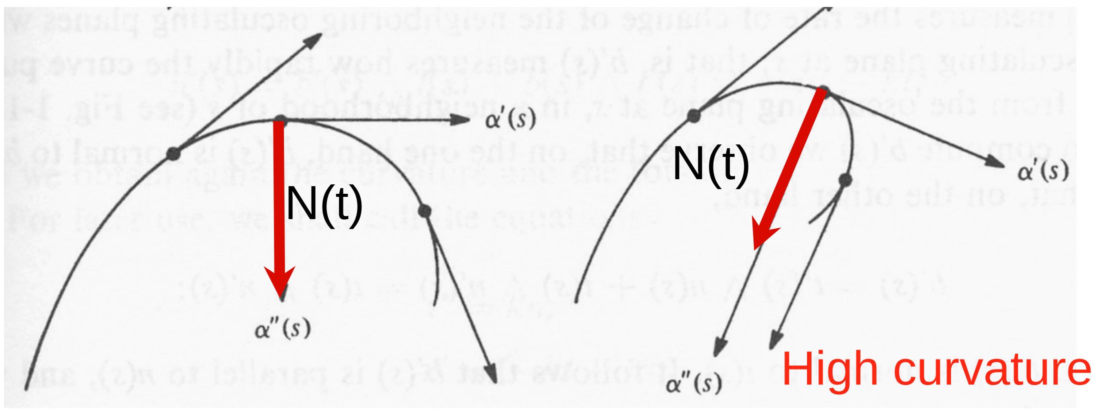
- K is zero for a line, constant for circle
  - What constant? 1/r
- 1/||K(t)|| is the radius of the circle that touches P(t) at *t* and has the same curvature as the curve



Curve Normal

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

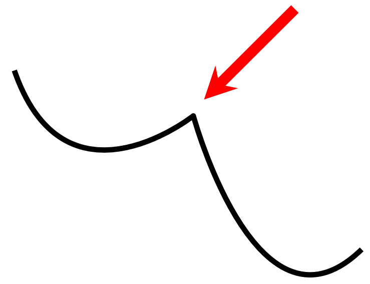




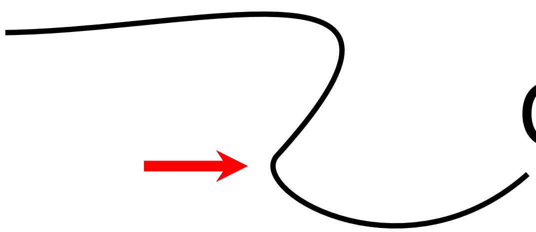
Questions?

Orders of Continuity

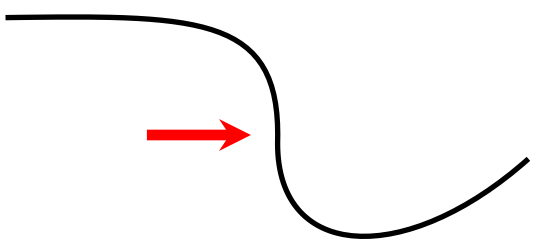
- C^0 = continuous
  - The seam can be a sharp kink
- G^1 = geometric continuity
  - Tangents **point to the same direction** at the seam
- C^1 = parametric continuity
  - Tangents **are the same** at the seam, implies G^1
- C^2 = curvature continuity
  - Tangents and their derivatives are the same



C^0



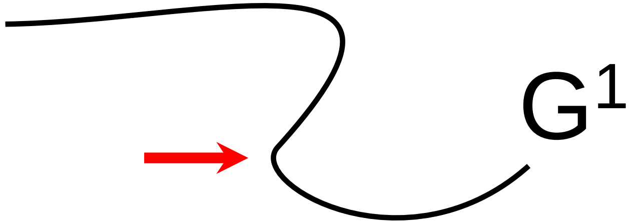
G^1

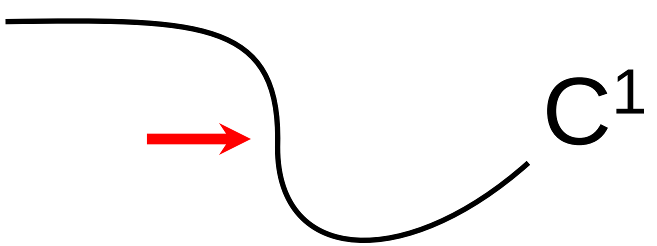


C^1

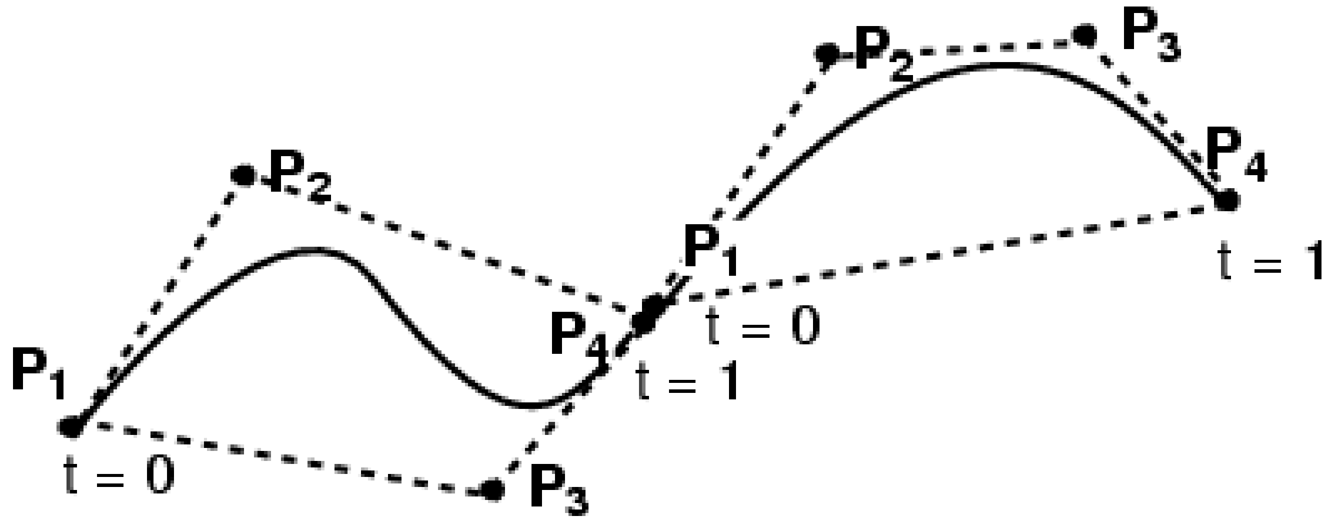
Orders of Continuity

- 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



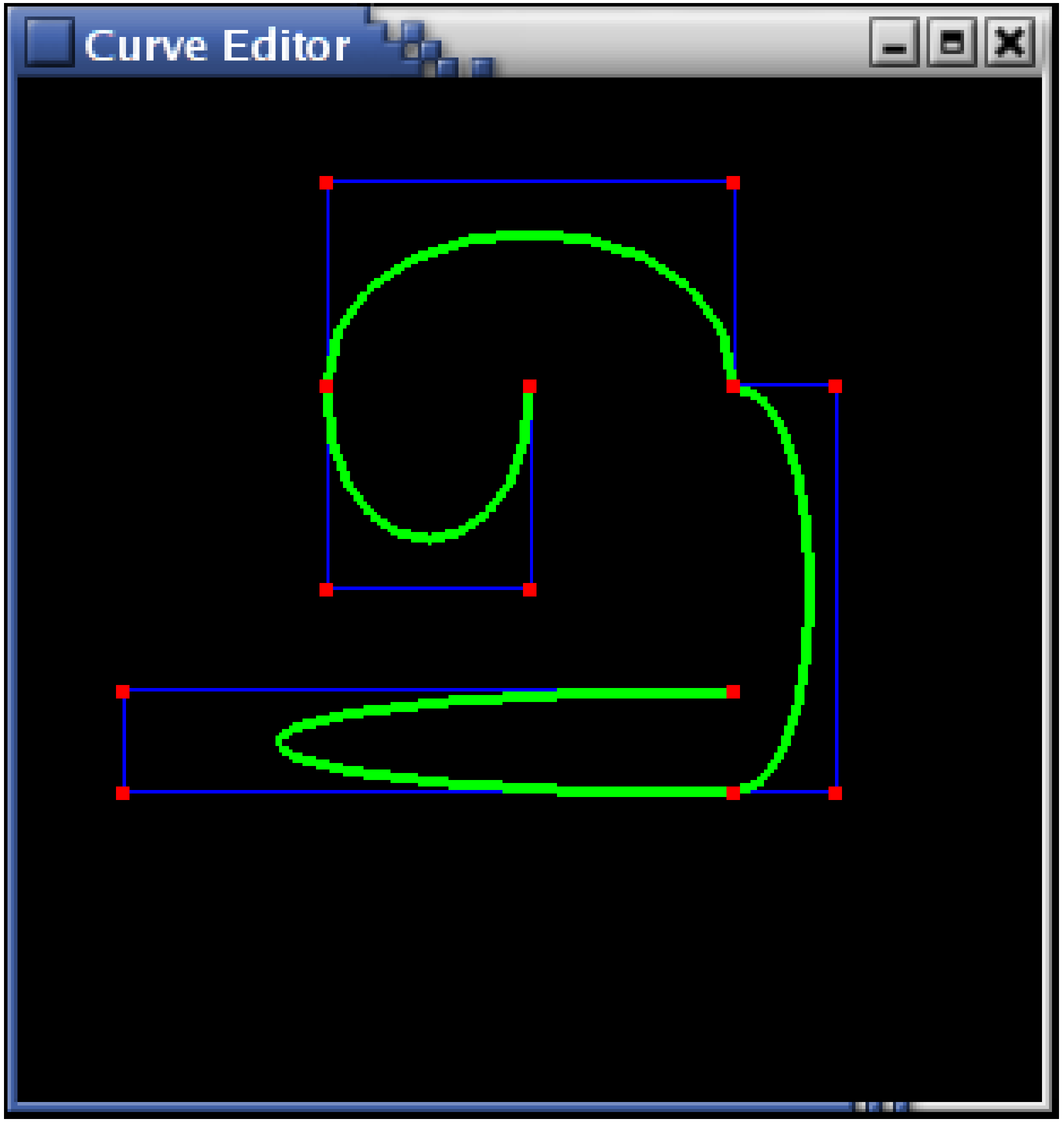


Connecting Cubic Bézier Curves



- 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

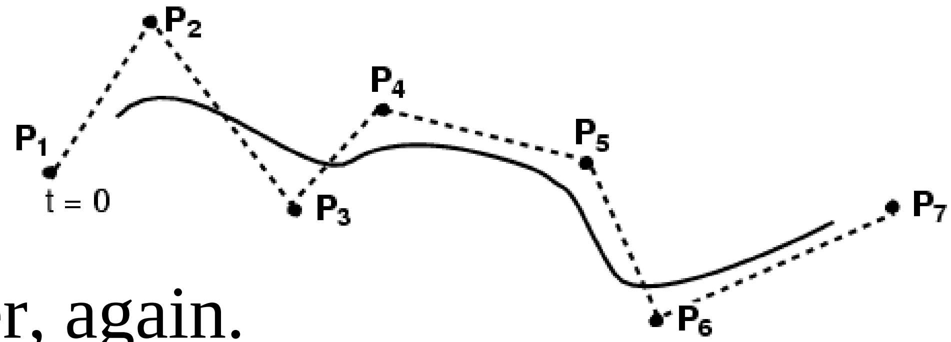


- Where is this curve
  - C^0 continuous?
  - G^1 continuous?
  - C^1 continuous?
- What's the relationship between:
  - the # of control points, and the # of cubic Bézier subcurves?

Questions?

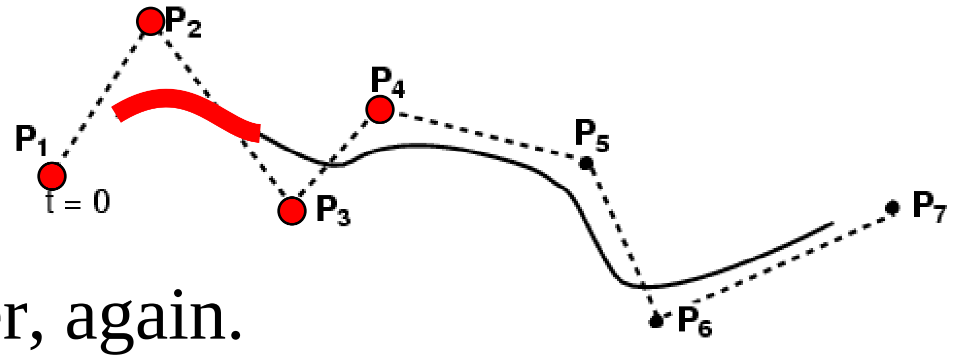
Cubic B-Splines

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



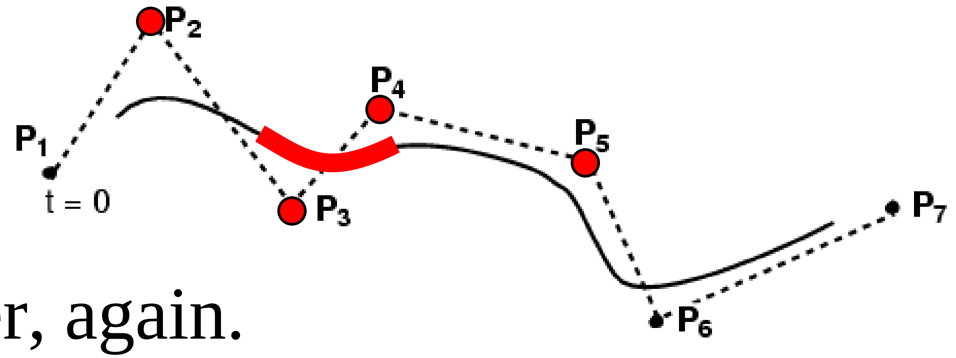
Cubic B-Splines

- ≥ 4 control points
- Locally cubic
  - Cubics chained together, again.



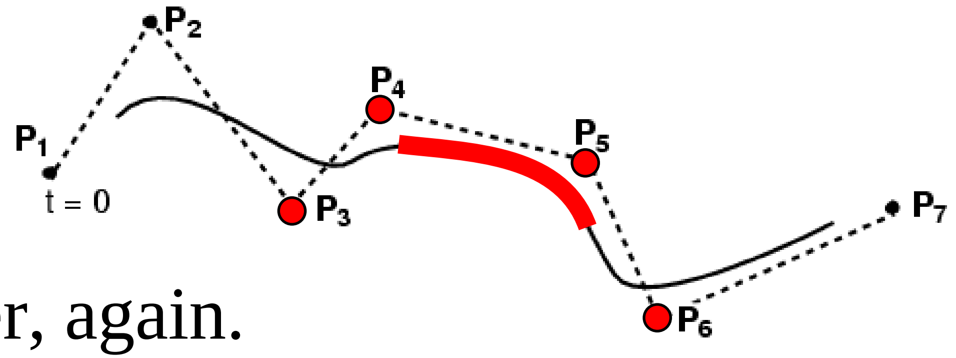
Cubic B-Splines

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



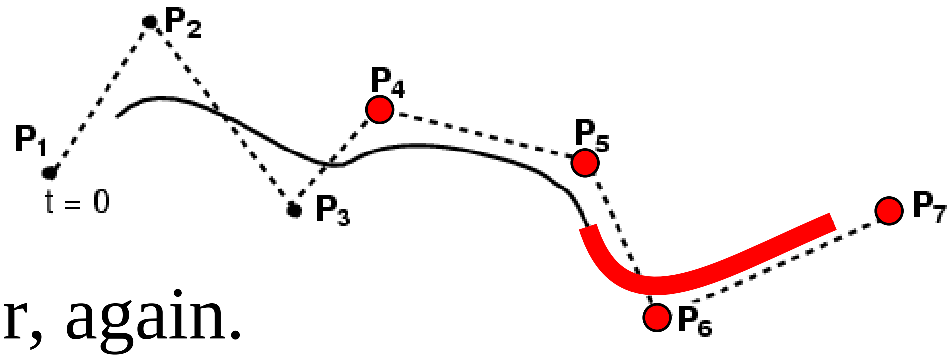
Cubic B-Splines

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

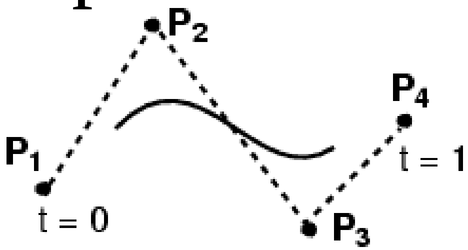


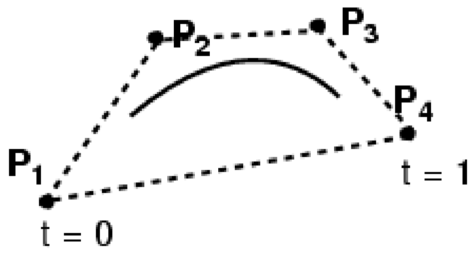
Cubic B-Splines

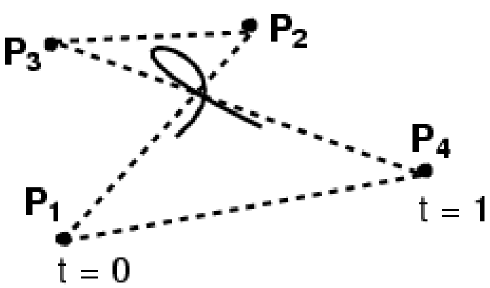
- ≥ 4 control points
- Locally cubic
  - Cubics chained together, again.



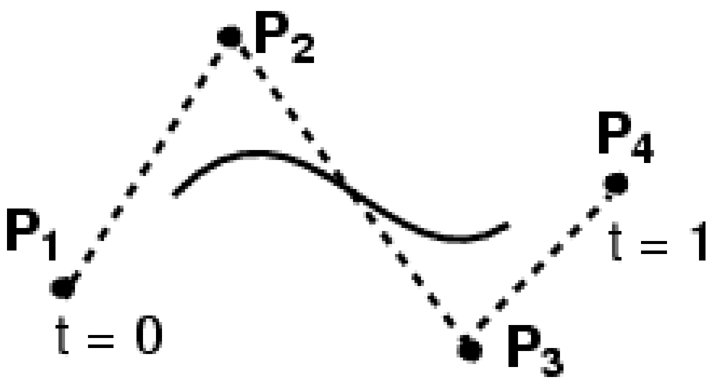
- Curve is not constrained to pass through any control points

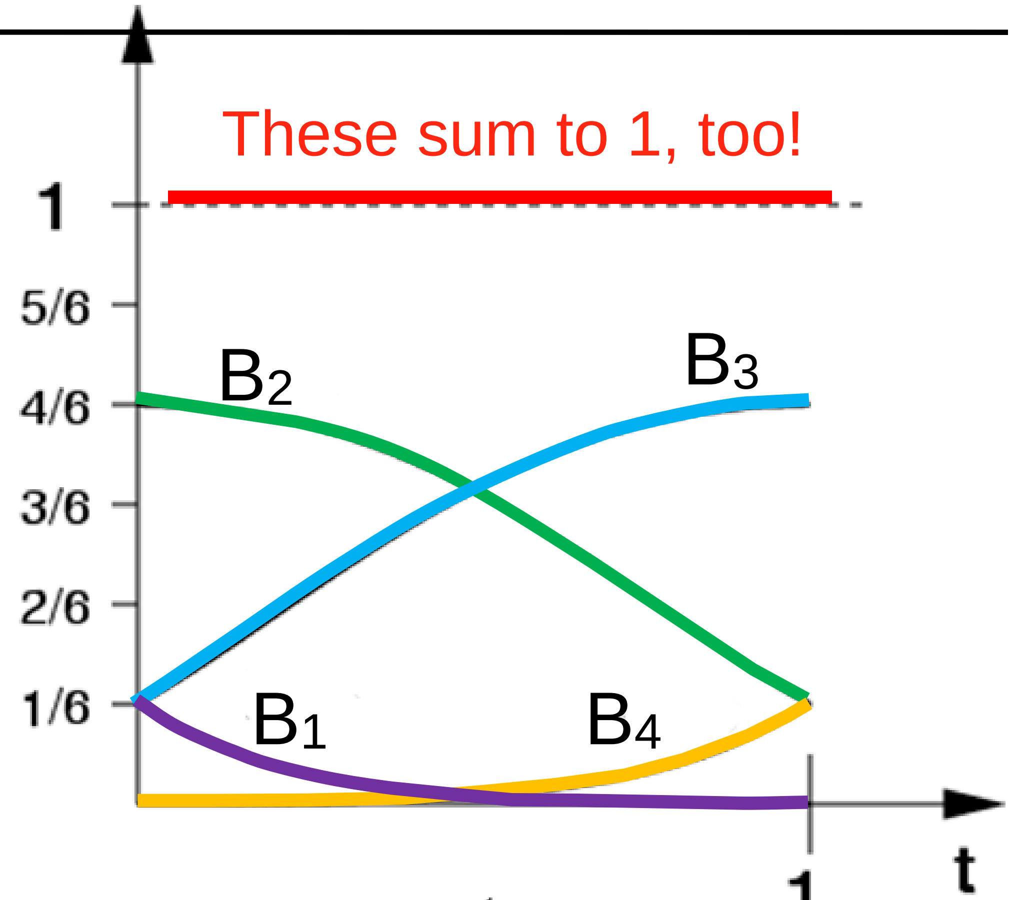






Cubic B-Splines: Basis





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

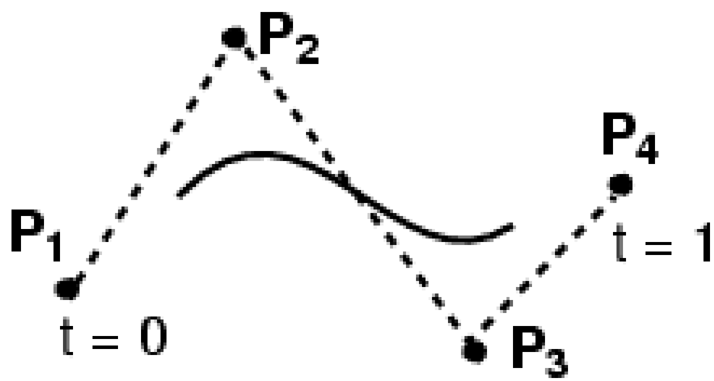
$$B_1(t) = \tfrac{1}{6}(1-t)^3$$

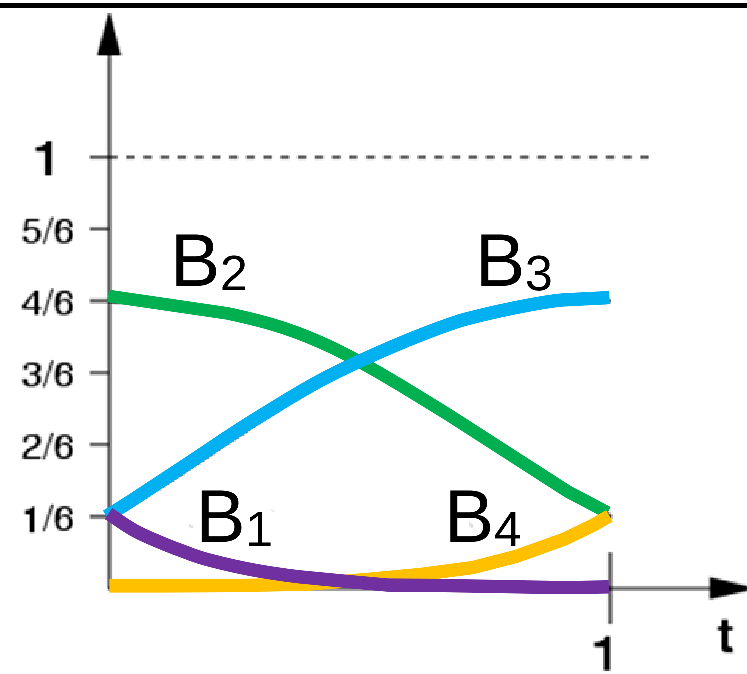
$$B_2(t) = \frac{1}{6}(3t^3 - 6t^2 + 4)$$

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

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

Cubic B-Splines: Basis





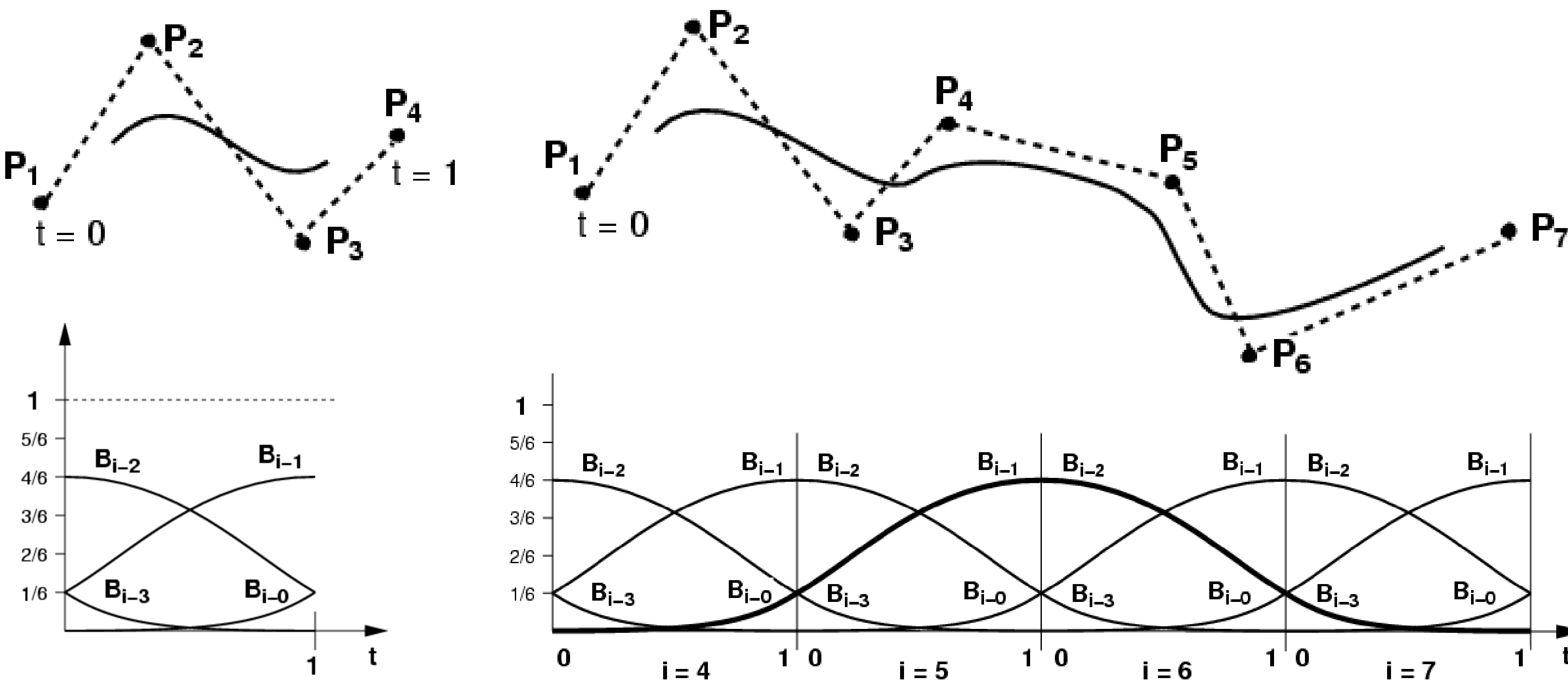
$$Q(t) = \frac{(1-t)^3}{6} P_1 \quad + \frac{3t^3 - 6t^2 + 4}{6} P_2 \quad + \frac{-3t^3 + 3t^2 + 3t + 1}{6} P_3 \quad + \frac{t^3}{6} P_4$$

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

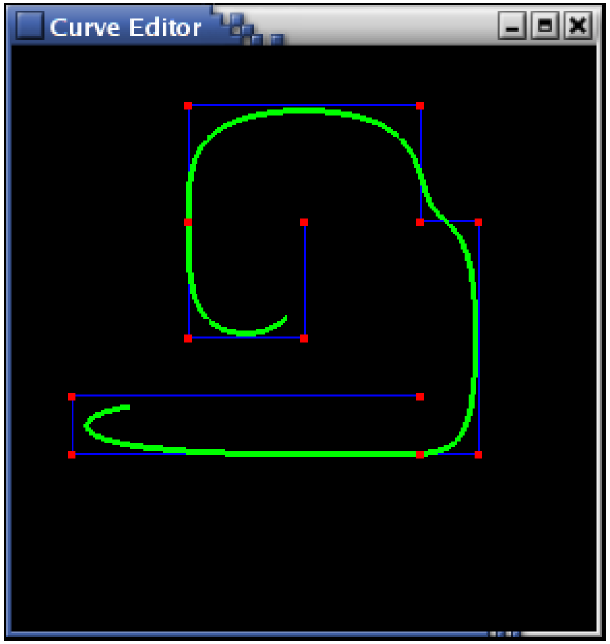
$$\boxed{B_{B-Spline} = \frac{1}{6} \begin{pmatrix} 1 & -3 & 3 & -1 \\ 4 & 0 & -6 & 3 \\ 1 & 3 & 3 & -3 \\ 0 & 0 & 0 & 1 \end{pmatrix}}$$

Cubic B-Splines

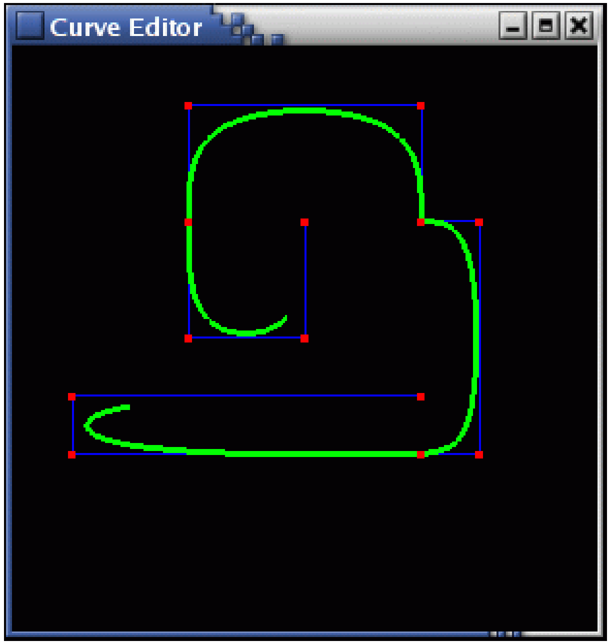
- Local control (windowing)
- Automatically C^2, and no need to match tangents!



B-Spline Curve Control Points

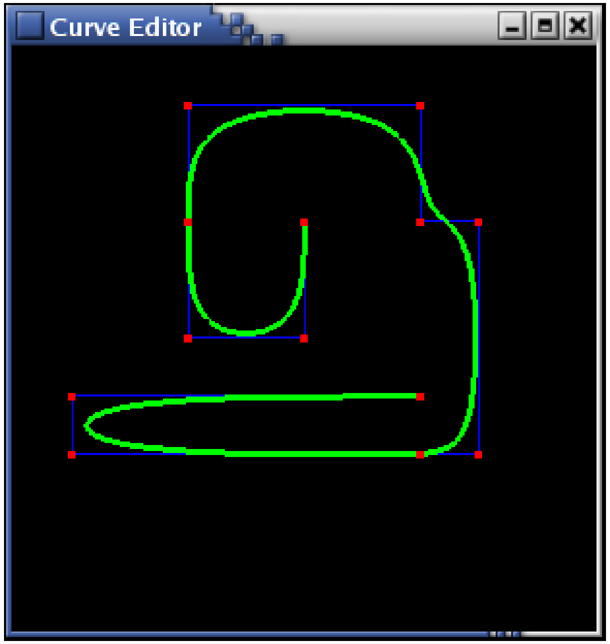


Default B-Spline



B-Spline with derivative discontinuity

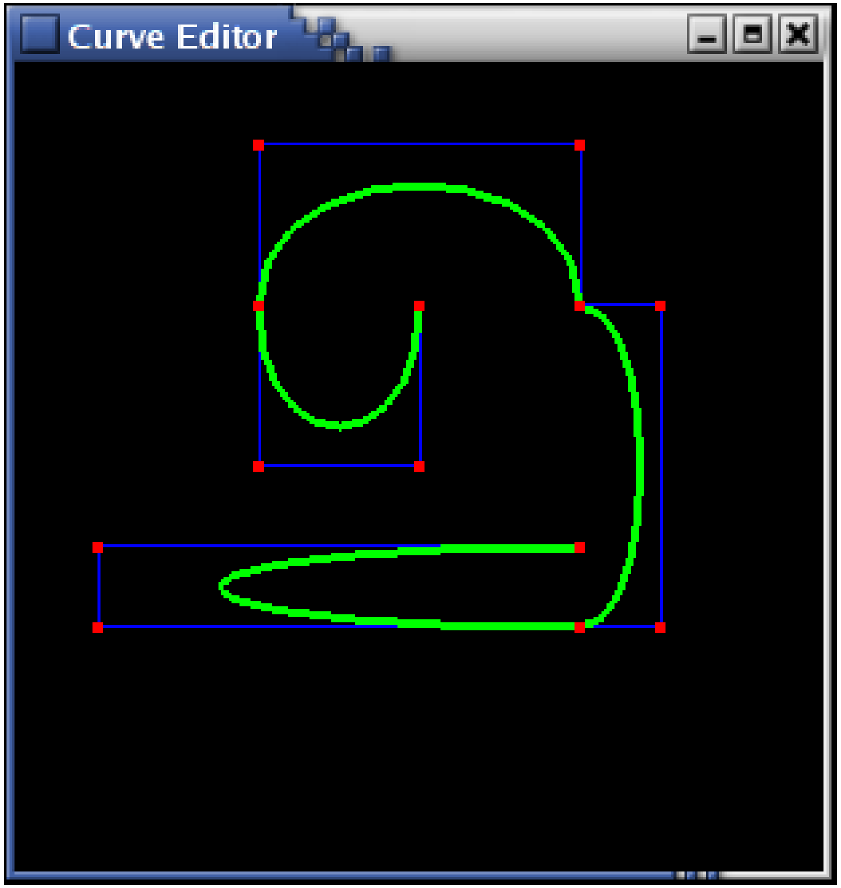
Repeat interior control point



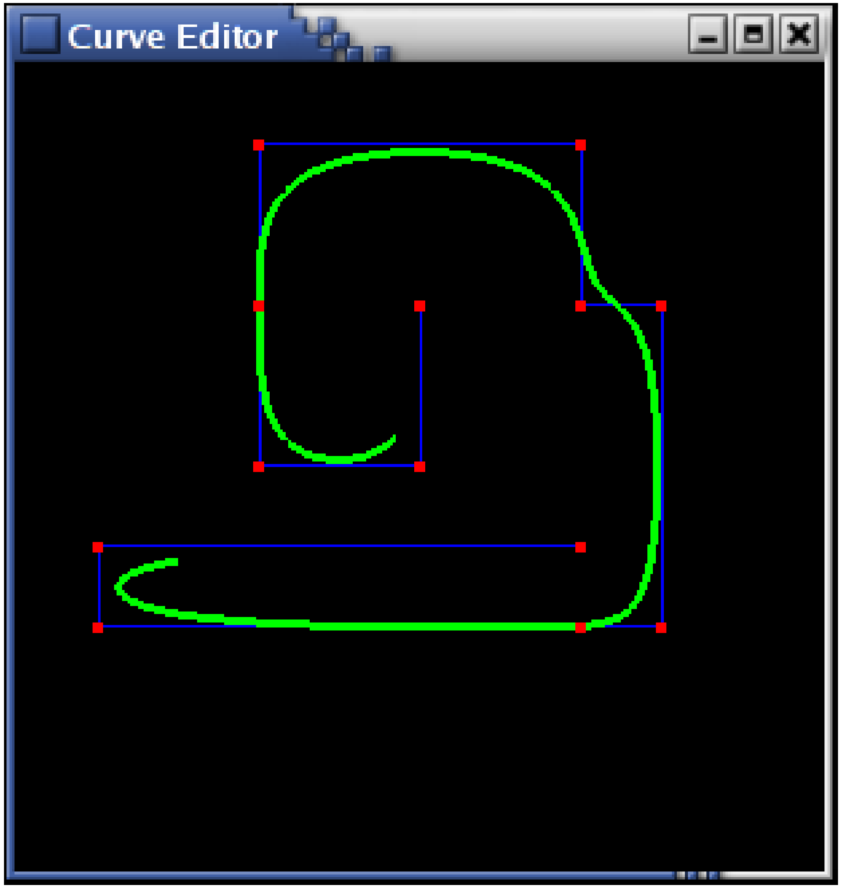
B-Spline which passes through end points

Repeat end points

Bézier ≠ B-Spline



Bézier



B-Spline

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

Converting between Bézier & BSpline

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

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

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

- $P(t) = \mathbf{G}\,\mathbf{B}_1\,\mathbf{T}(t) =$ $\mathbf{G}\,\mathbf{B}_1\,(\mathbf{B}_2^{-1}\mathbf{B}_2)\,\mathbf{T}(t) =$ $(\mathbf{G}\,\mathbf{B}_1\,\mathbf{B}_2^{-1})\,\mathbf{B}_2\,\mathbf{T}(t)$

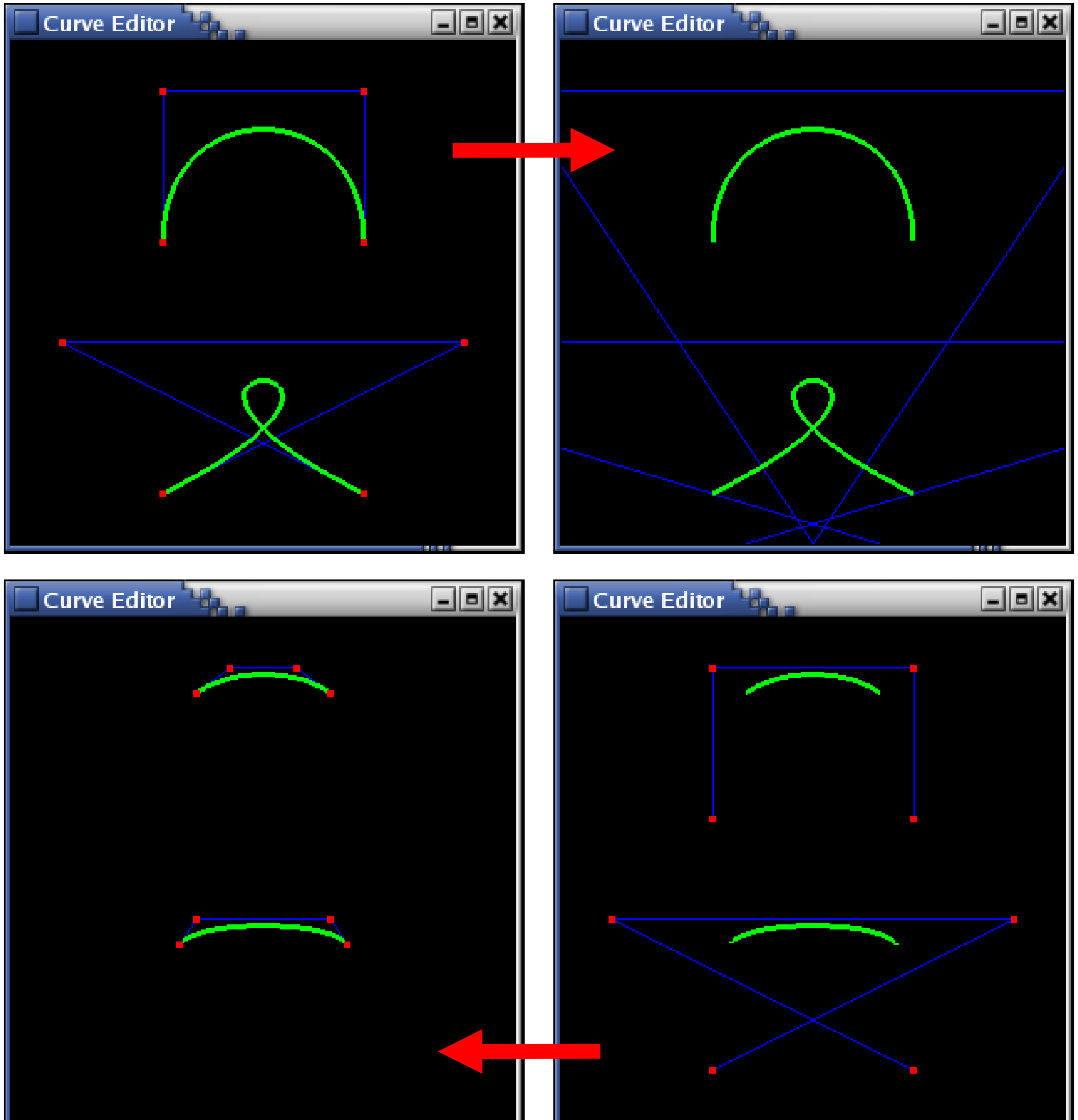
$$B_{B-Spline} = \frac{1}{6}\begin{pmatrix} 1 & -3 & 3 & -1 \\ 4 & 0 & -6 & 3 \\ 1 & 3 & 3 & -3 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- $\mathbf{G}\,\mathbf{B}_1\,\mathbf{B}_2^{-1}$ 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.

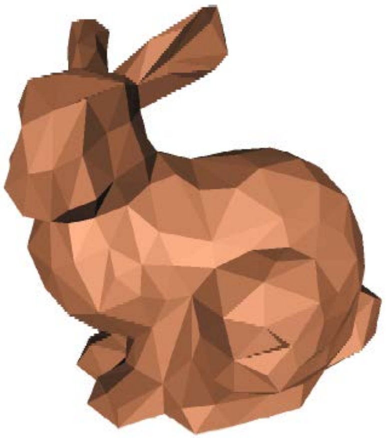
Questions?

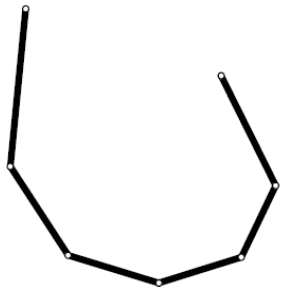
Representing Surfaces

- 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)



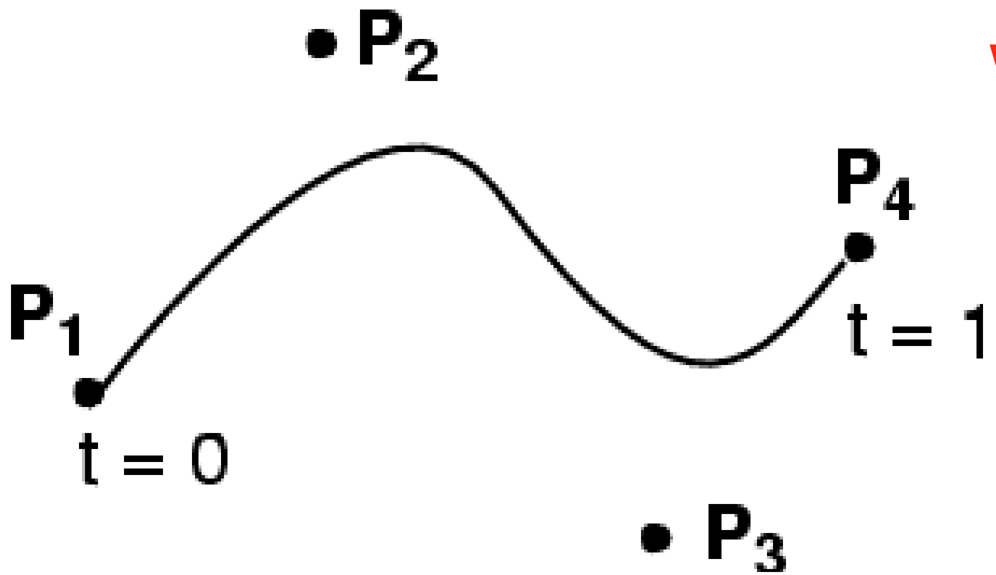


Smooth Surfaces?

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

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

What about a surface?

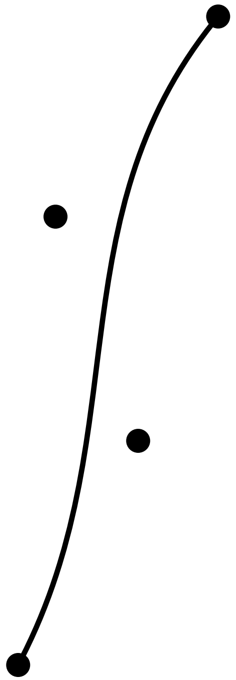


How to Build Them? Here's an Idea

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

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

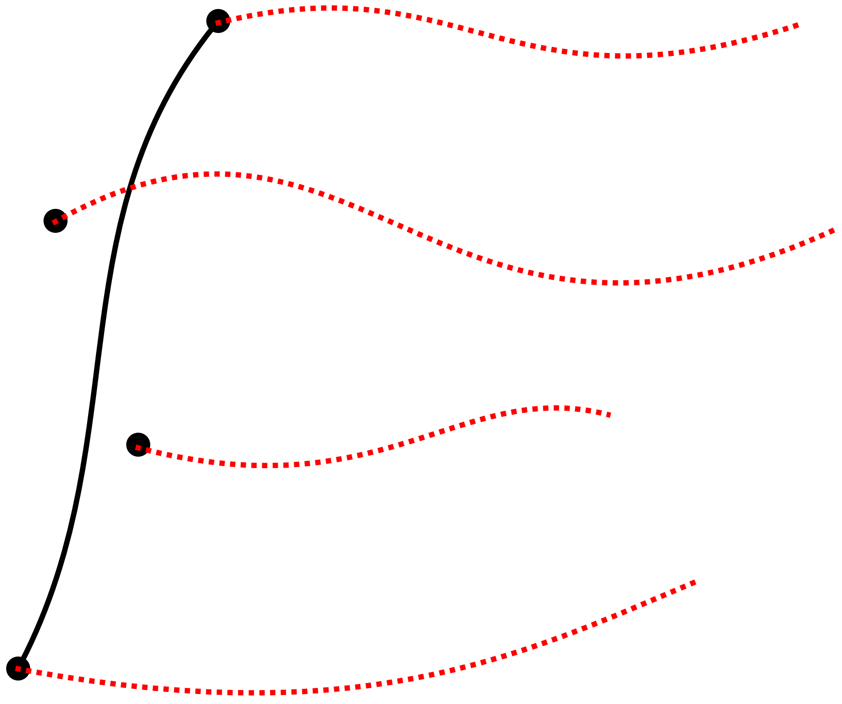
(Note! We relabeled *t* to *u*)



How to Build Them? Here's an Idea

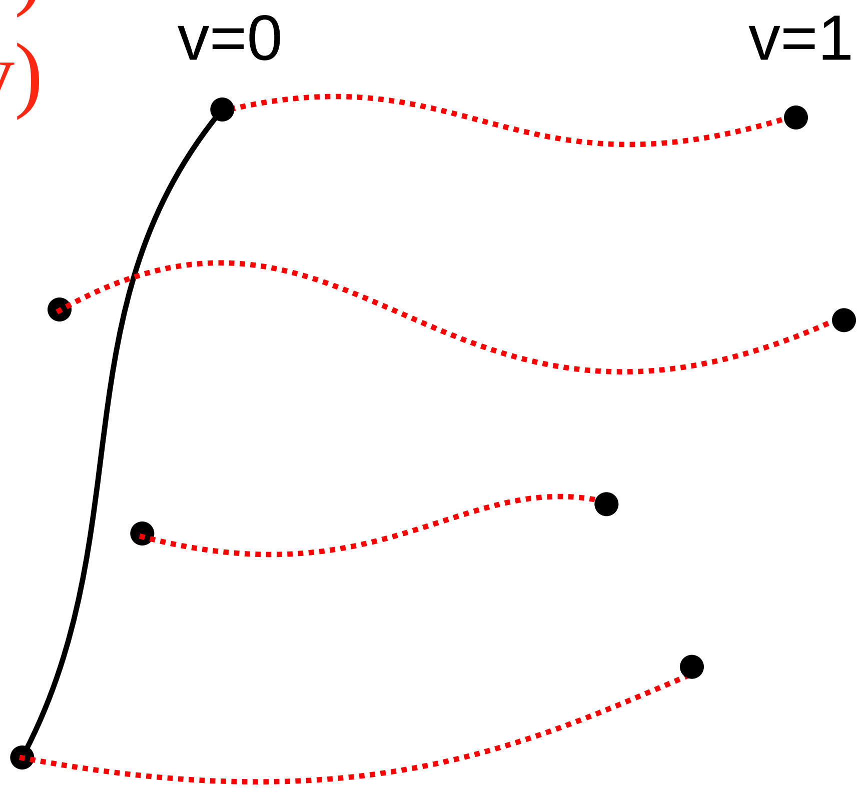
- $P(u) = (1-u)^3 \; P_1$
  $+ \; 3u(1-u)^2 \; P_2$
  $+ \; 3u^2(1-u) \; P_3$
  $+ \; u^3 \; P_4$

(Note! We relabeled *t* to *u*)



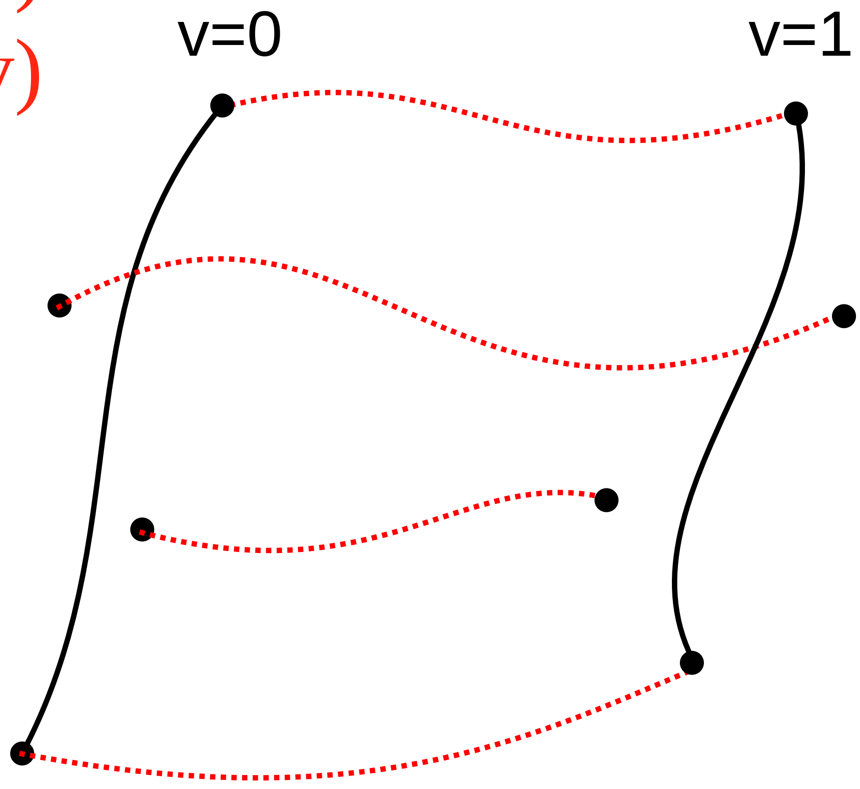
Here's an Idea

- $P(u, \textcolor{red}{v}) = (1-u)^3 \; P_1\textcolor{red}{(v)}$
  $+ \; 3u(1-u)^2 \; P_2\textcolor{red}{(v)}$
  $+ \; 3u^2(1-u) \; P_3\textcolor{red}{(v)}$
  $+ \; u^3 \; P_4\textcolor{red}{(v)}$
- Let's make
  the P_is 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_is move along curves!

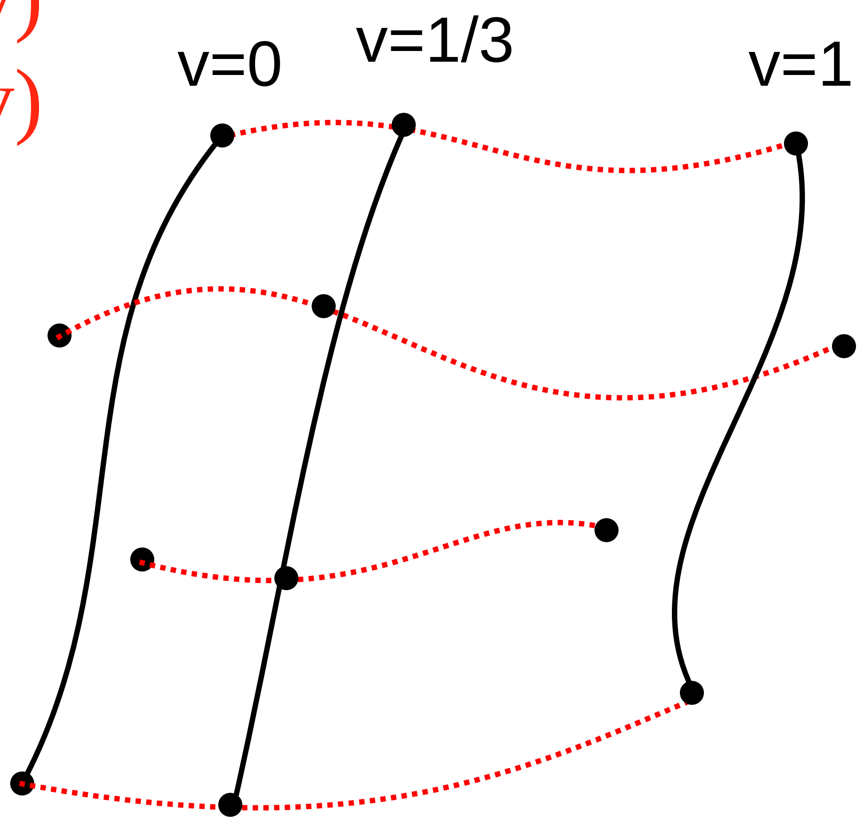


Here's an Idea

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

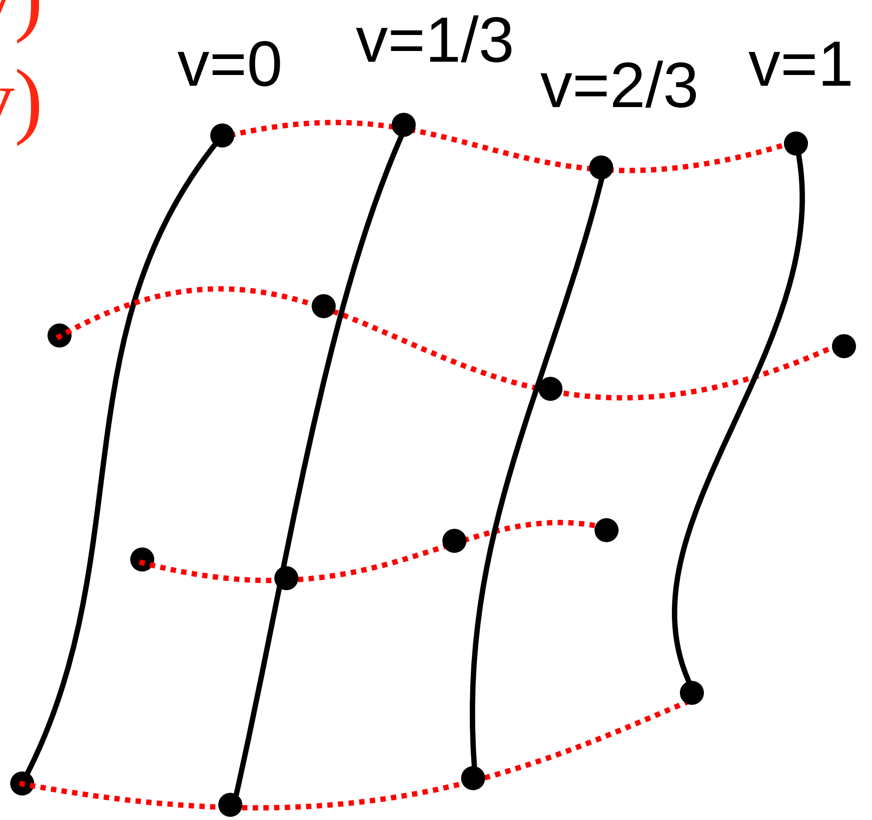
$$
\begin{aligned}
P(u, v) = \; & (1-u)^3 && P_1(v) \\
+ \; & 3u(1-u)^2 && P_2(v) \\
+ \; & 3u^2(1-u) && P_3(v) \\
+ \; & u^3 && P_4(v)
\end{aligned}
$$

- Let's make the P_is move along curves!



Here's an Idea

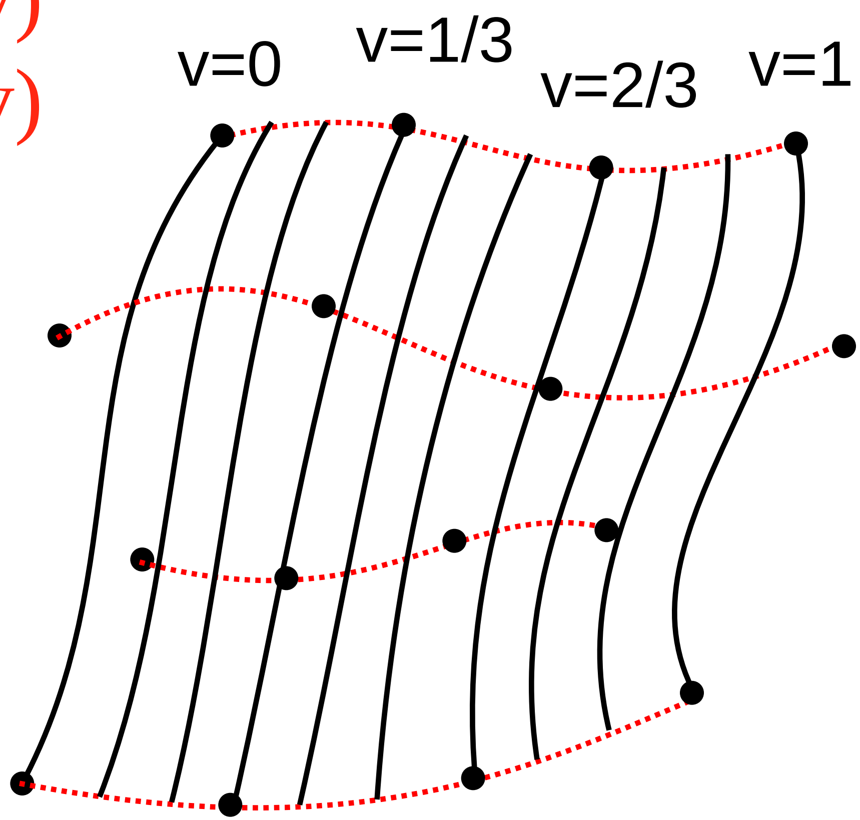
- $P(u, v) = (1-u)^3 \quad P_1(v)$
  $+ \quad 3u(1-u)^2 \quad P_2(v)$
  $+ \quad 3u^2(1-u) \quad P_3(v)$
  $+ \quad u^3 \quad P_4(v)$
- Let's make the P_is move along curves!



Here's an Idea

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

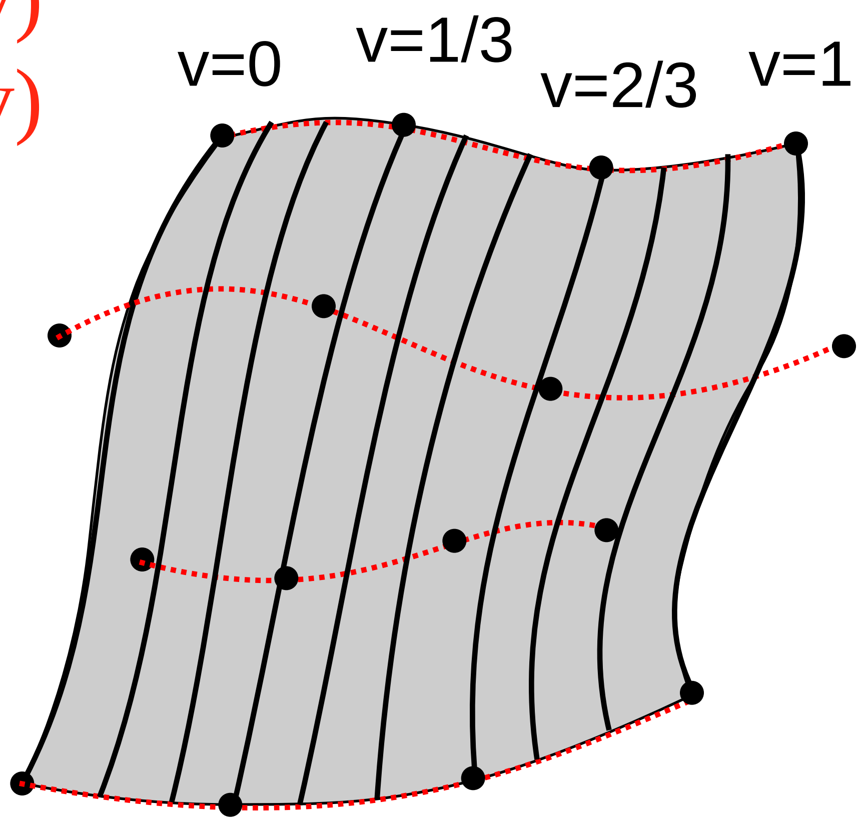
- Let's make the P_is move along curves!



Here's an Idea

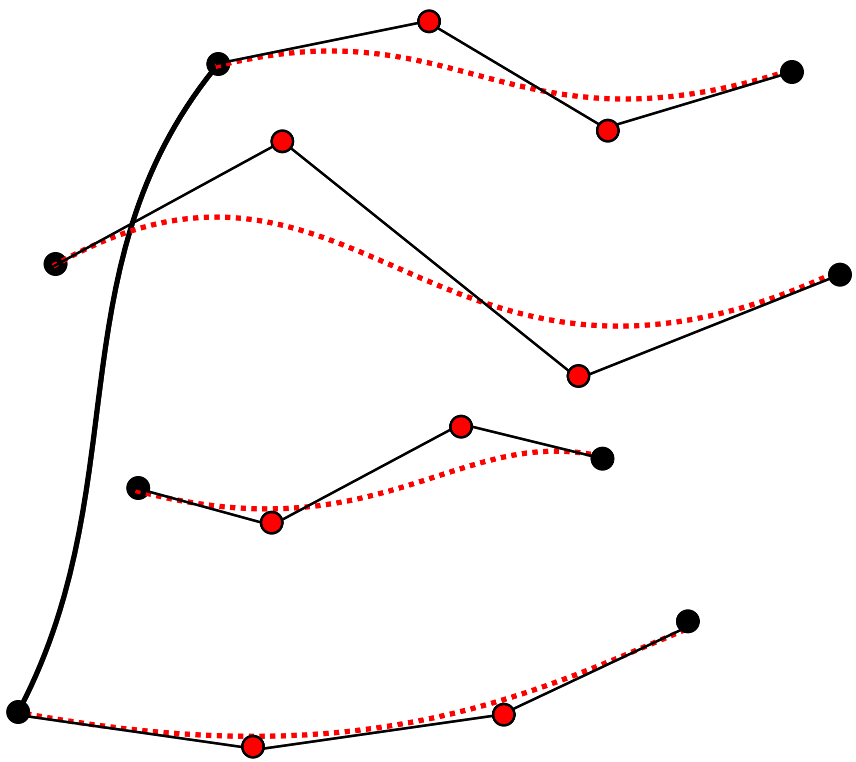
- $P(u, v) = (1-u)^3 \quad P_1(v)$
  $+ \quad 3u(1-u)^2 \quad P_2(v)$
  $+ \quad 3u^2(1-u) \quad P_3(v)$
  $+ \quad u^3 \quad P_4(v)$
- Let's make the P_is move along curves!

A 2D surface patch!



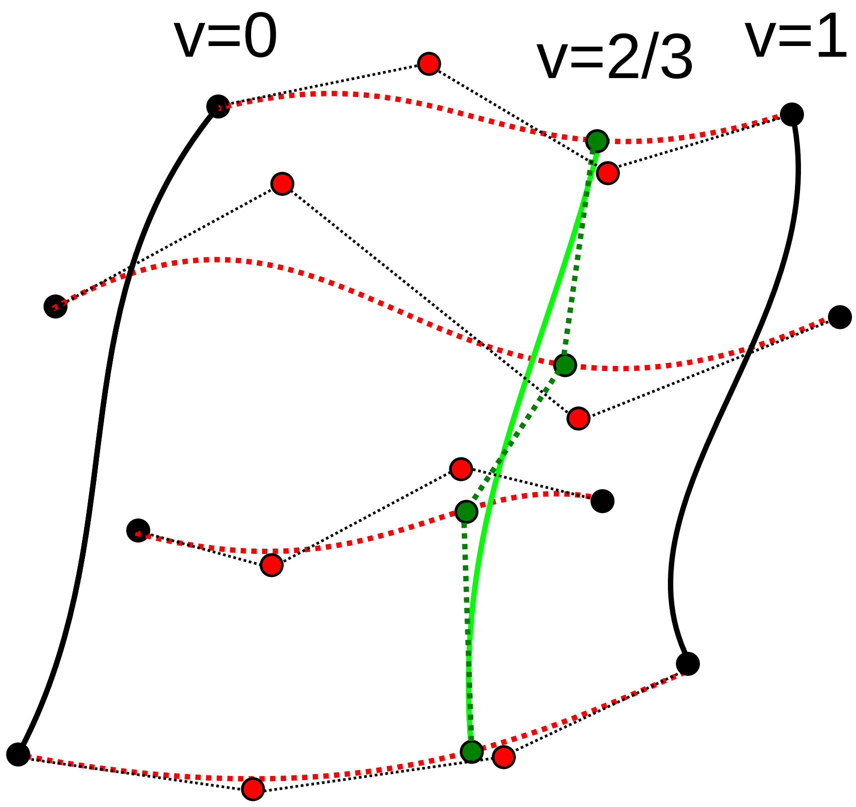
Tensor Product Bézier Patches

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



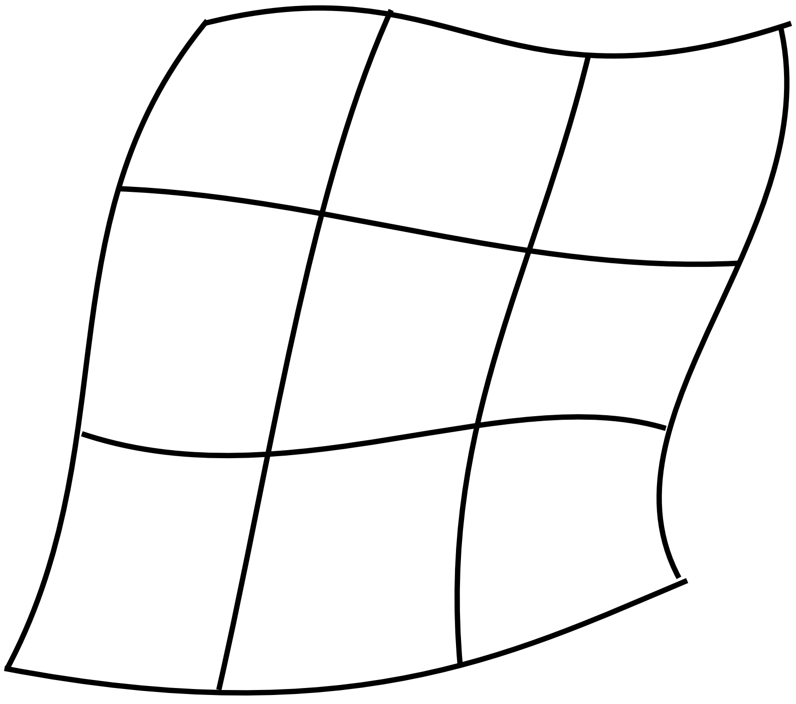
Tensor Product Bézier Patches

- In the previous, P_is 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

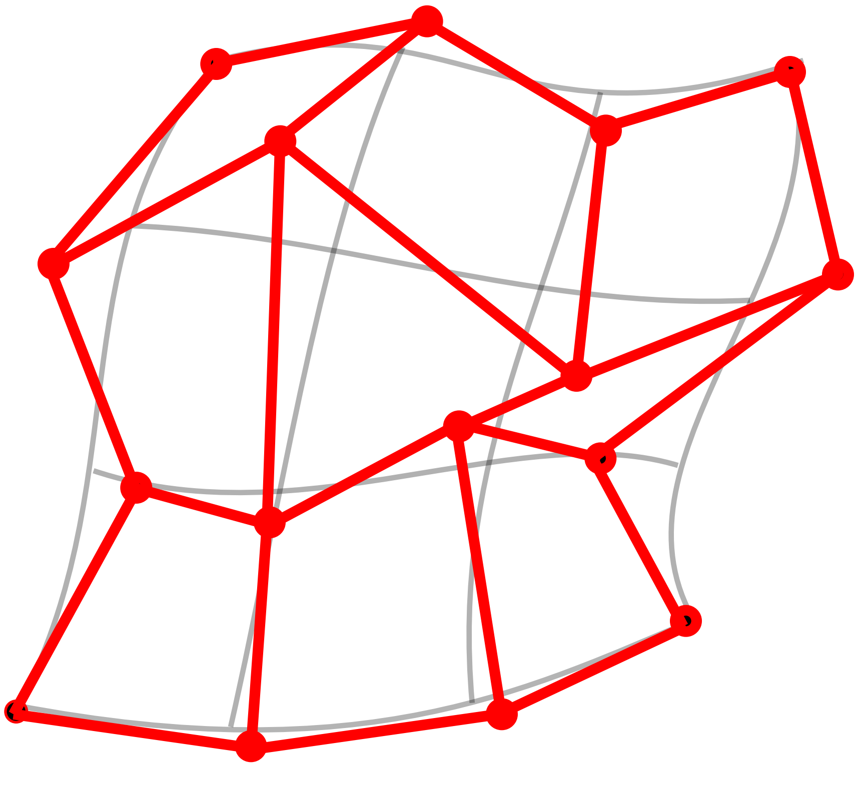
A **bicubic Bézier surface**



Tensor Product Bézier Patches

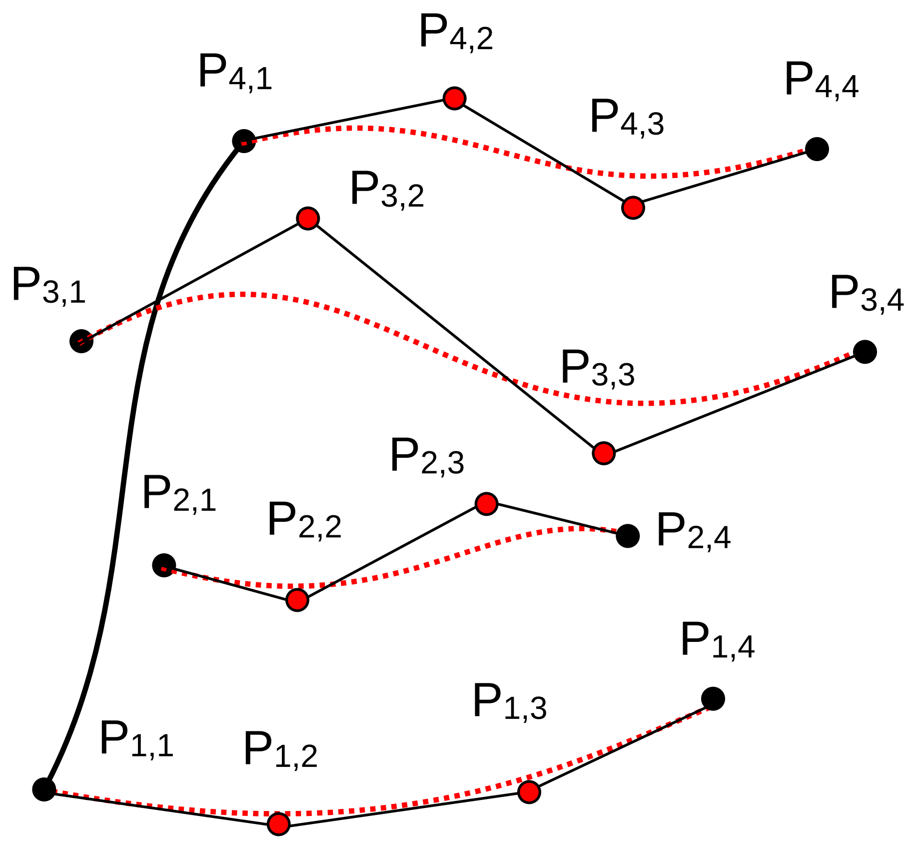
The “Control Mesh”

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

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

$$P(u,v) =$$

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

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

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

Bicubics, Tensor Product

- P(u,v) = B1(u) * P_1(v)
  - \+ B2(u) * P_2(v)
  - \+ B3(u) * P_3(v)
  - \+ B4(u) * P_4(v)
- Pi(v) = B1(v) * $P_{i,1}$
  - \+ B2(v) * $P_{i,2}$
  - \+ B3(v) * $P_{i,3}$
  - \+ B4(v) * $P_{i,4}$

$$P(u,v) =$$

16 control points $P_{i,j}$

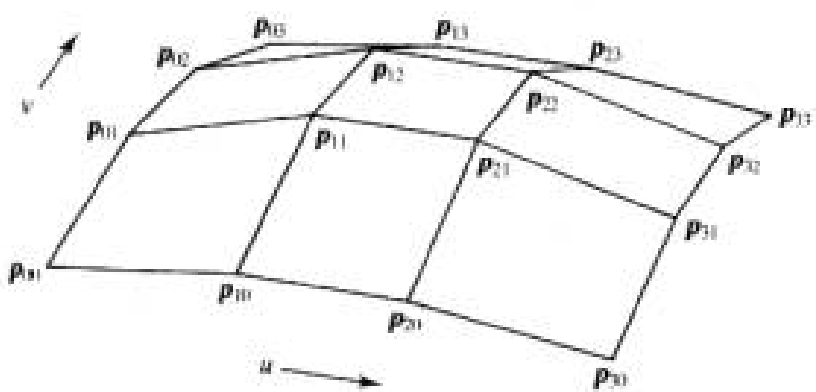
16 2D basis functions $B_{i,j}$

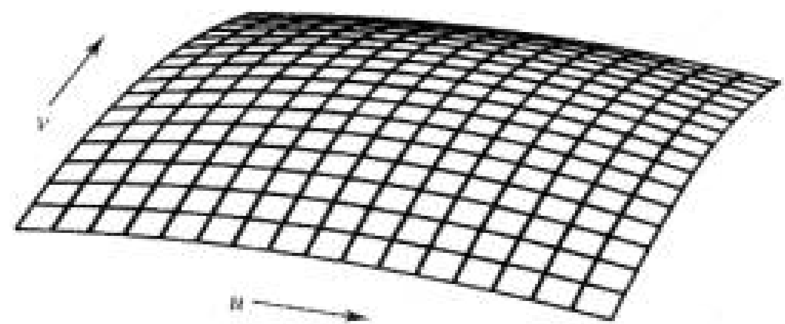
$$= \sum_{i=1}^{4} \sum_{j=1}^{4} \boxed{P_{i,j}} B_{i,j}(u,v)$$

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

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: $B_1(u)B_1(v)$; $B_1(u)B_2(v)$;... $B_4(u)B_4(v)$

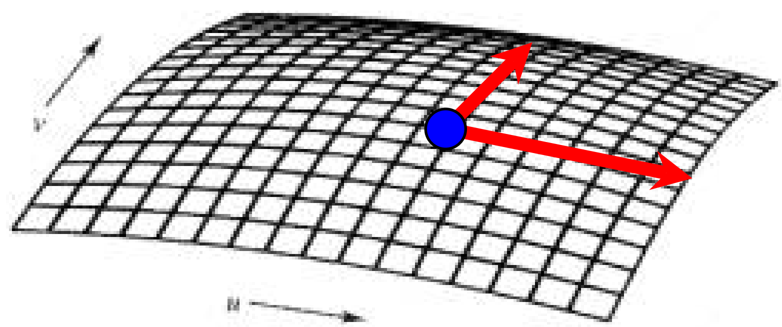




Questions?

Tangents and Normals for Patches

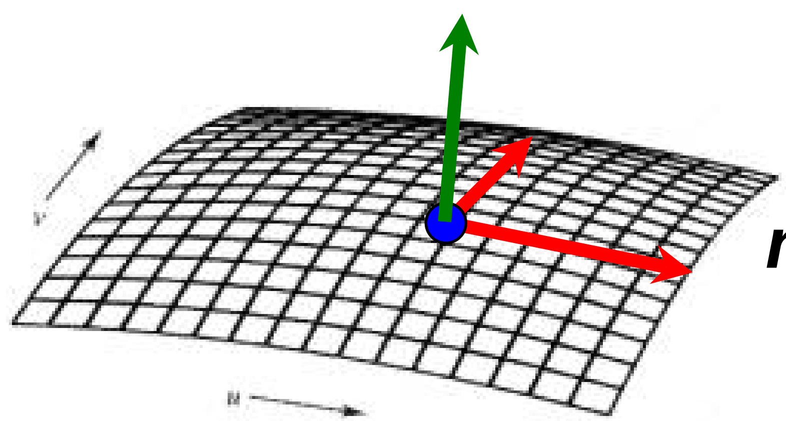
- P(u,v) is a 3D point specified by *u*, *v*
- The partial derivatives $\partial P/\partial u$ and $\partial P/\partial v$ are 3D vectors
  - Both are tangent to surface at P



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!**

Questions?

Recap: Matrix Notation for Curves

- Cubic Bézier in matrix notation

point on curve
(2x1 vector)

Canonical
"power basis"

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

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

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

"Spline matrix"
(Bernstein)

Hardcore: Matrix Notation for Patches

- Not required, but convenient!

x coordinate of surface at (*u*,*v*)

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

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

Column vector of basis functions (*v*)

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

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 $\boldsymbol{G}^x$, $\boldsymbol{G}^y$, $\boldsymbol{G}^z$ 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!

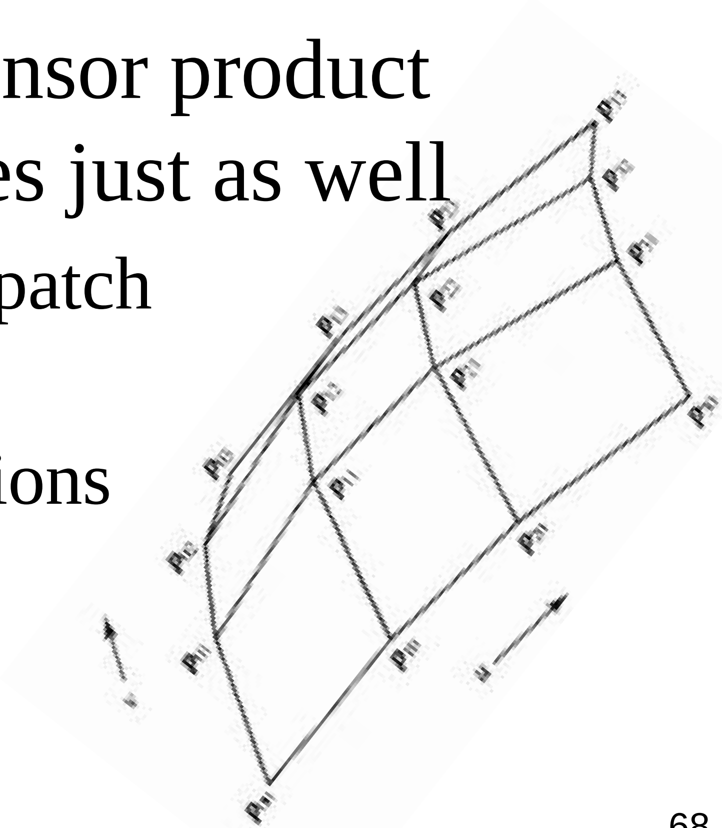
$$P^k(u,v) = \boldsymbol{T}^l(u)\,\boldsymbol{B}_l^i\,\boldsymbol{G}_{ij}^k\,\boldsymbol{B}_m^j\,\boldsymbol{T}^m(v)$$

- "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!

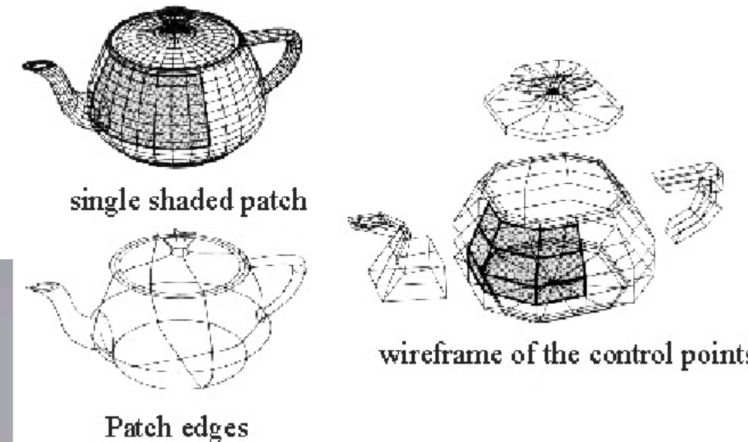


Tensor Product Spline Patches

- 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.

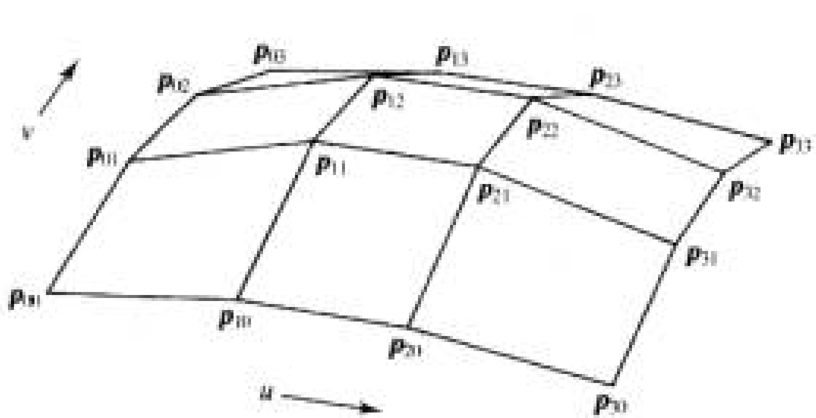
Utah Teapot: Tensor Bézier Splines

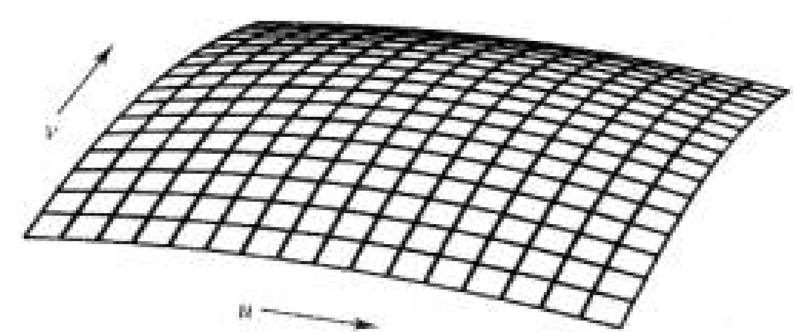
- Designed by Martin Newell



Cool: Displacement Mapping

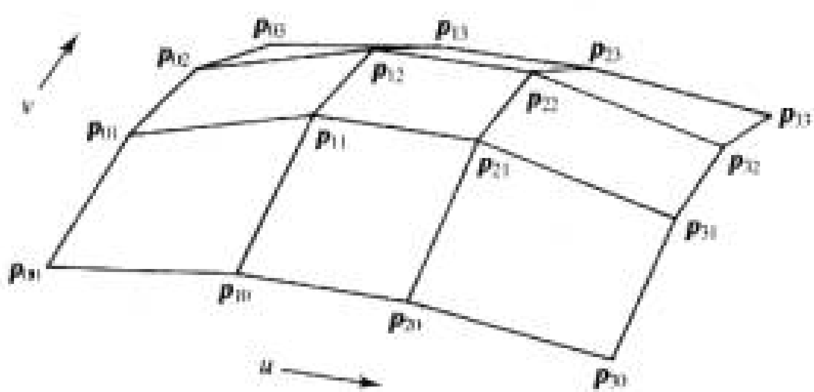
- Not all surfaces are smooth...

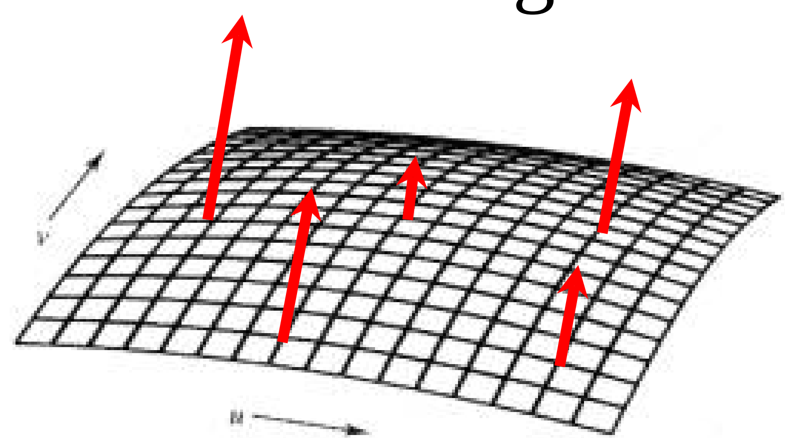




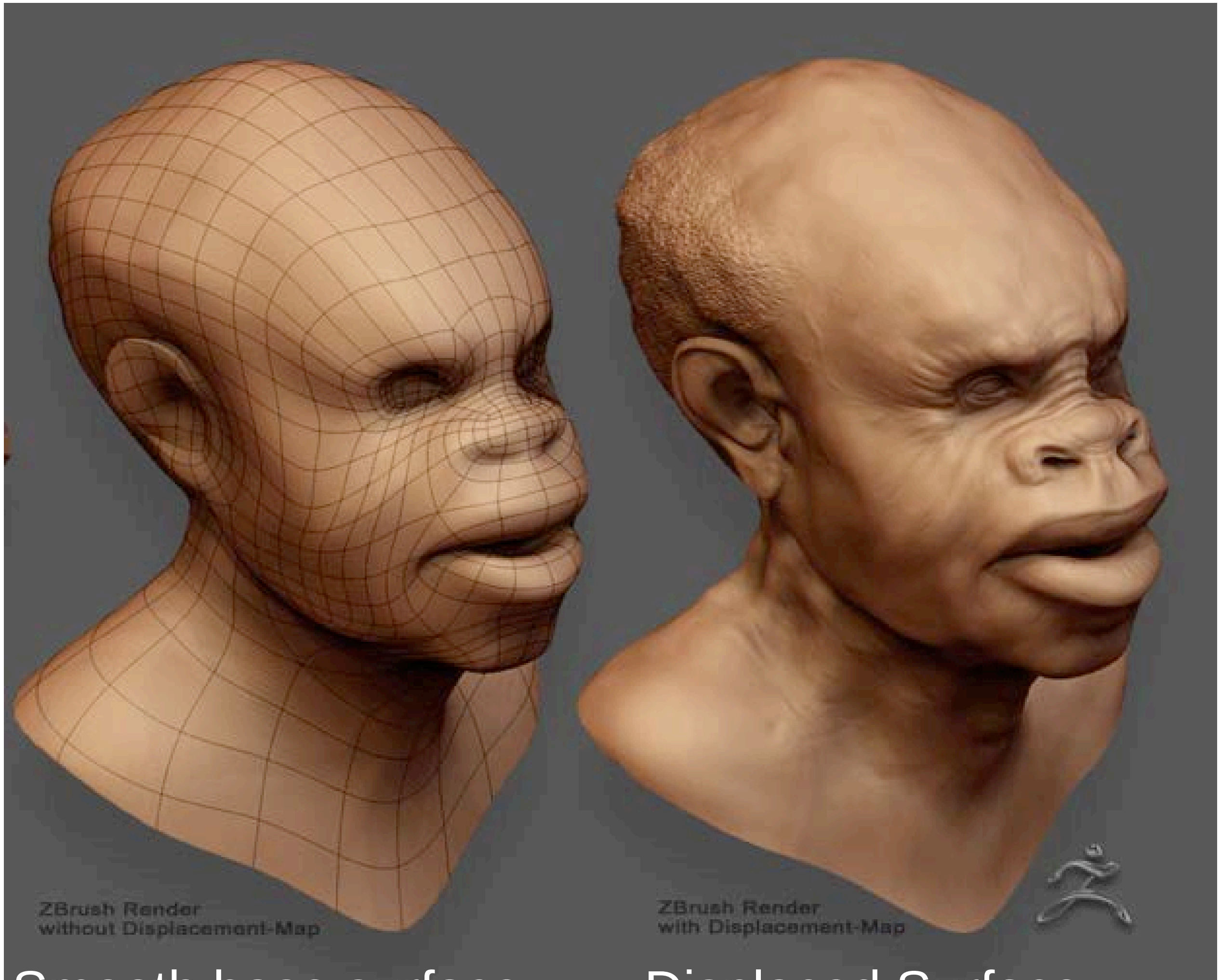
Cool: Displacement Mapping

- 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





Displacement Mapping Example



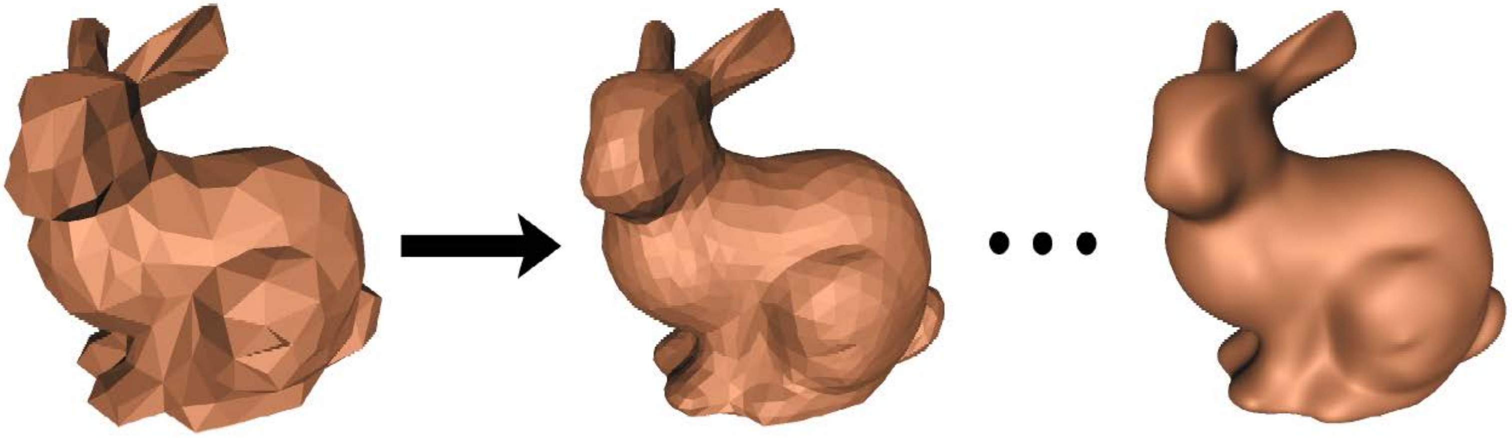
Smooth base surface Displaced Surface

zbrushcentral

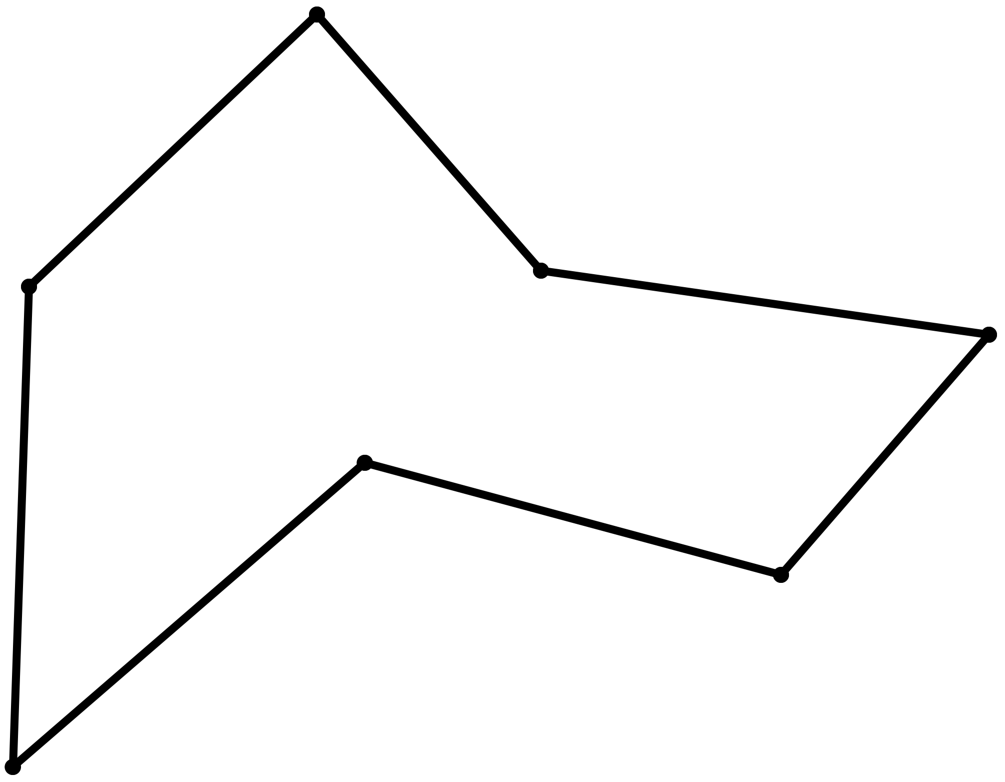
Questions?

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!

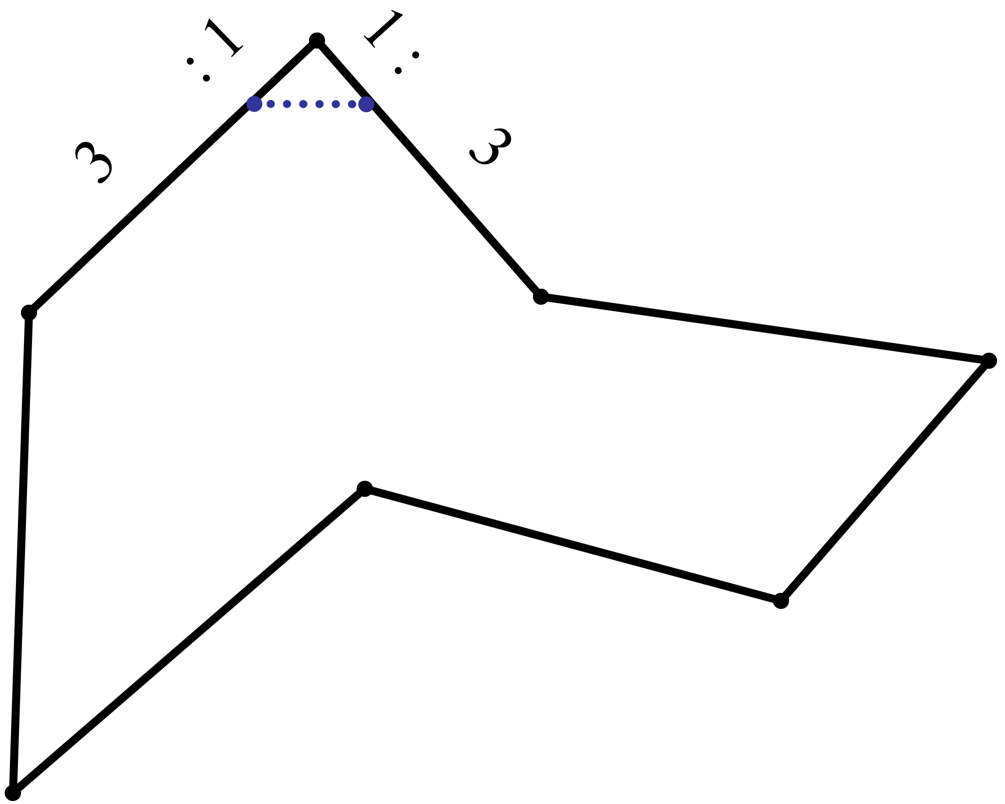


Corner Cutting



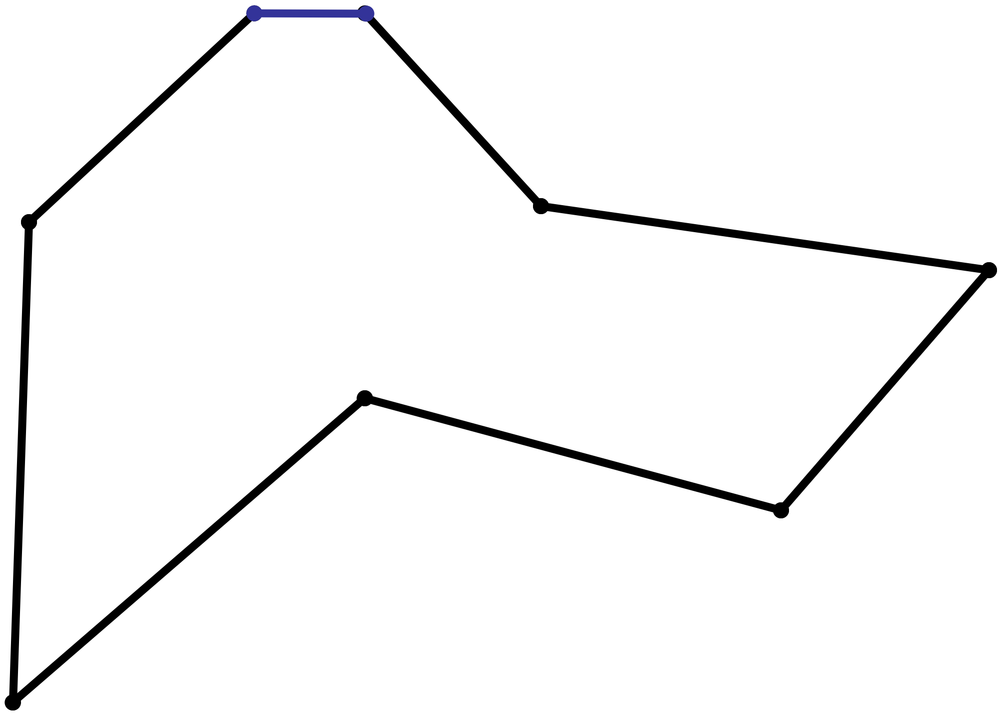
Slide by Adi Levin

Corner Cutting



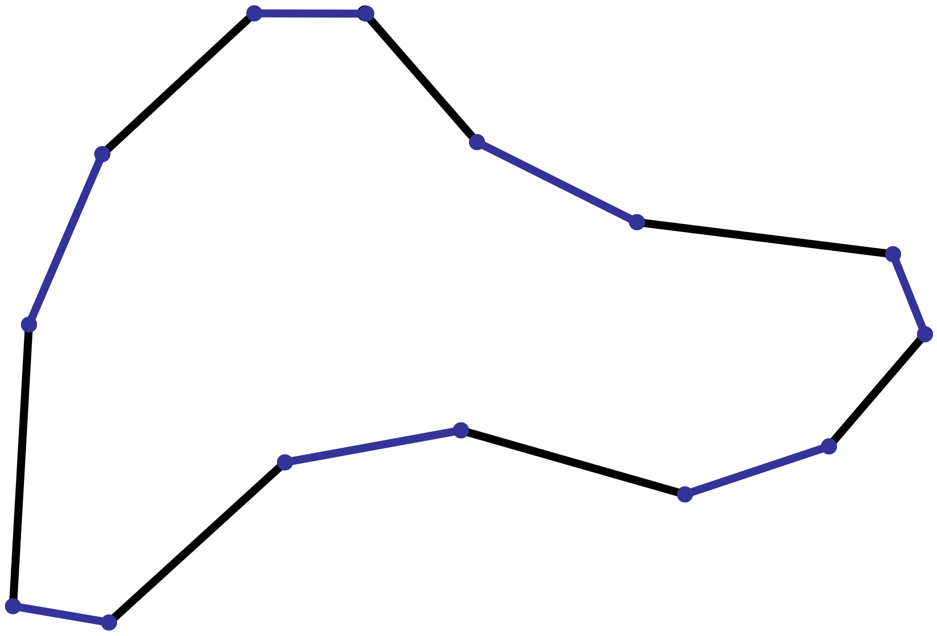
Slide by Adi Levin

Corner Cutting



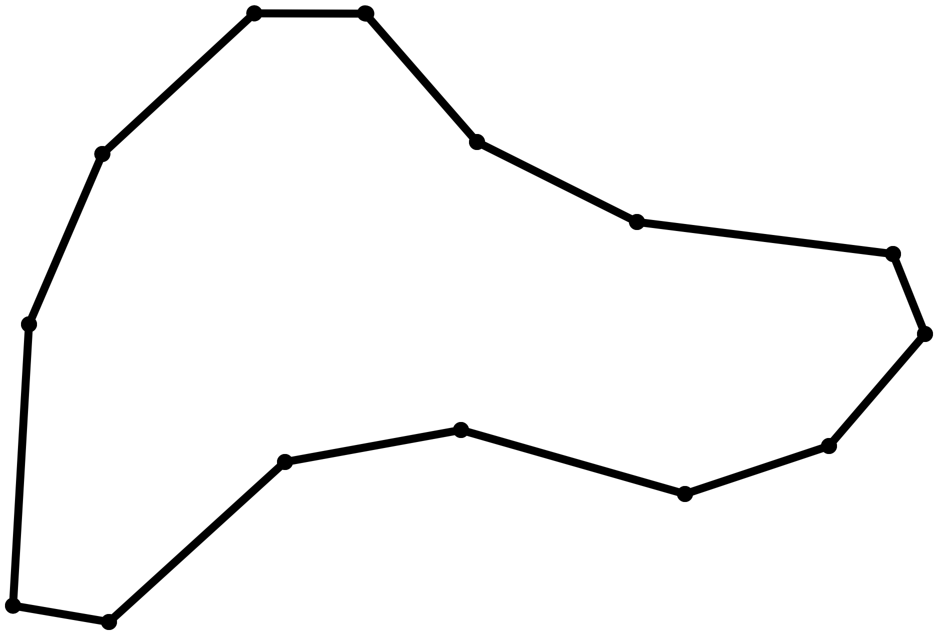
Slide by Adi Levin

Corner Cutting



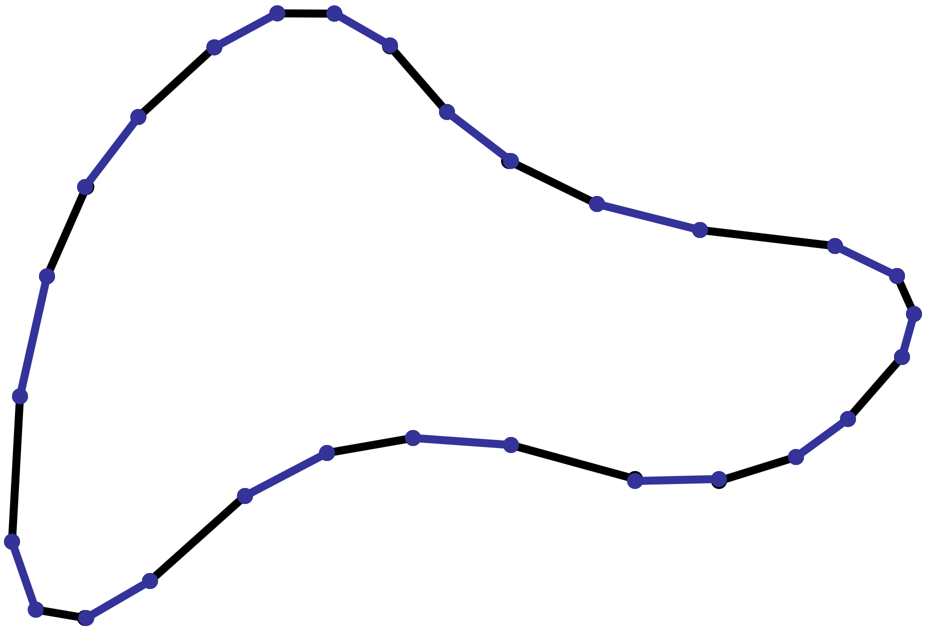
Slide by Adi Levin

Corner Cutting



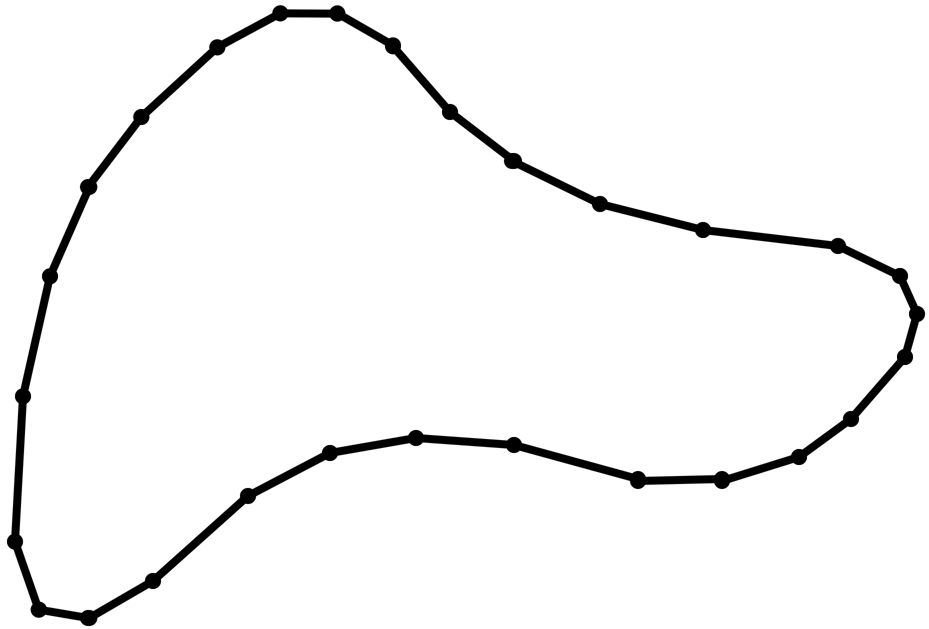
Slide by Adi Levin

Corner Cutting



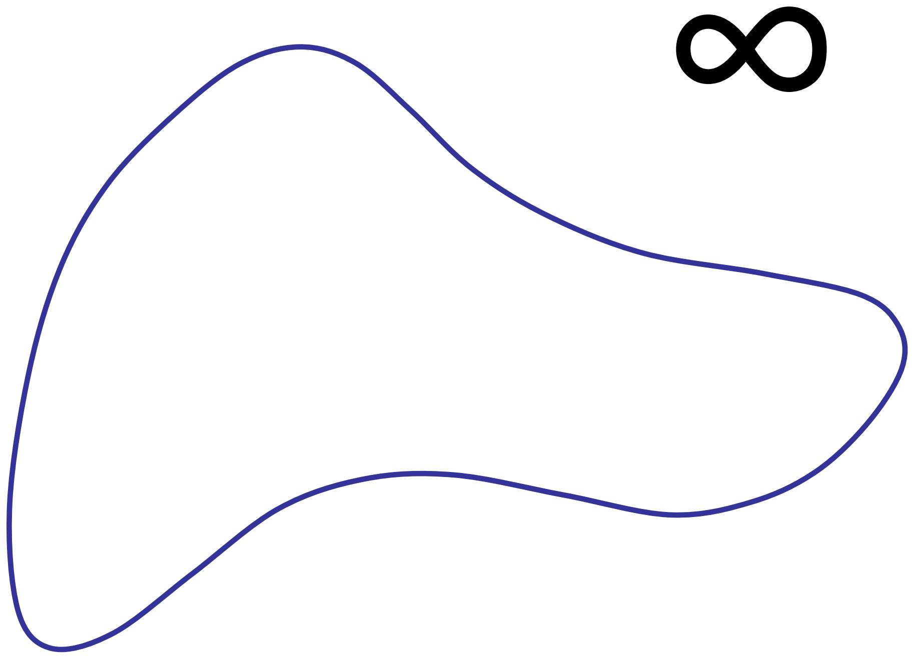
Slide by Adi Levin

Corner Cutting



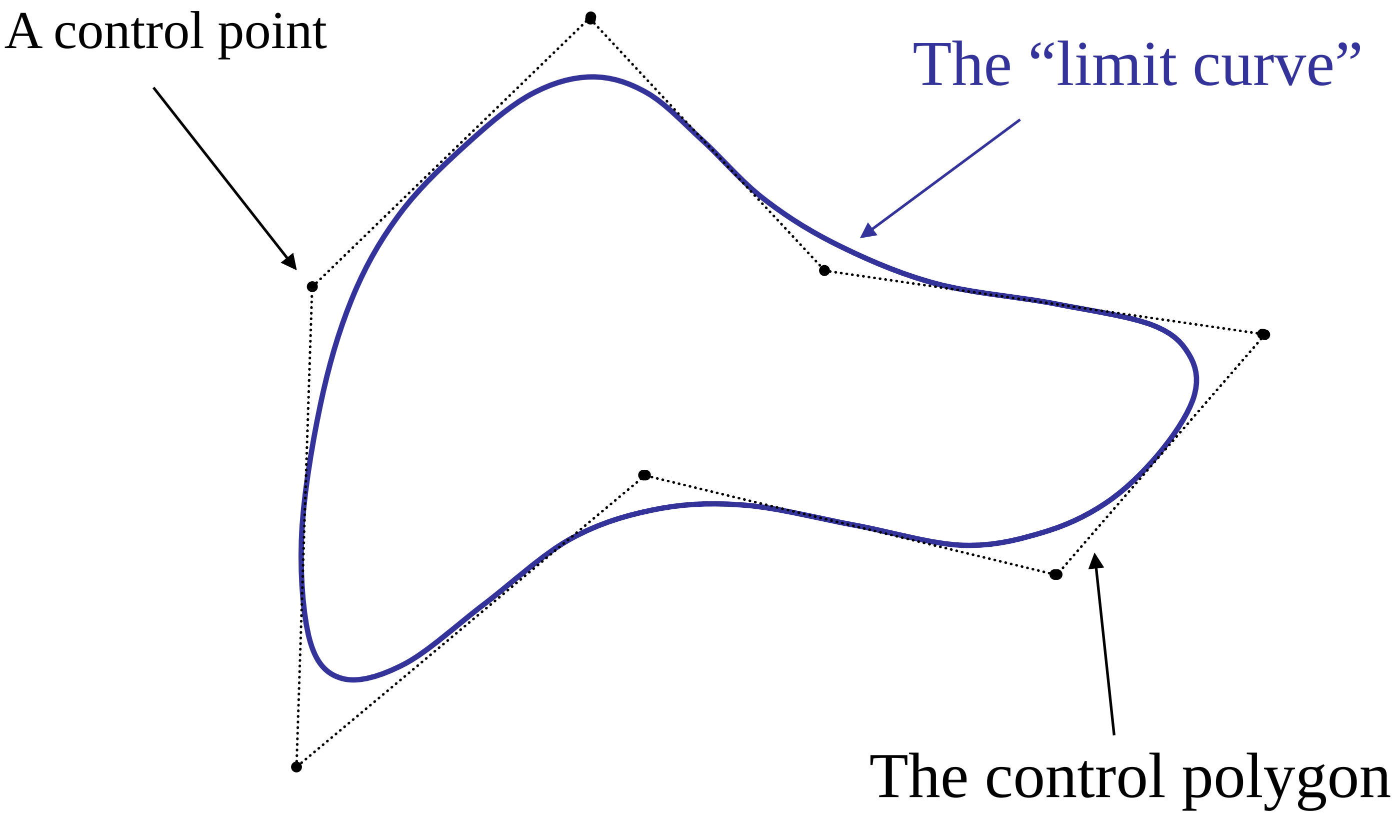
Slide by Adi Levin

Corner Cutting



Slide by Adi Levin

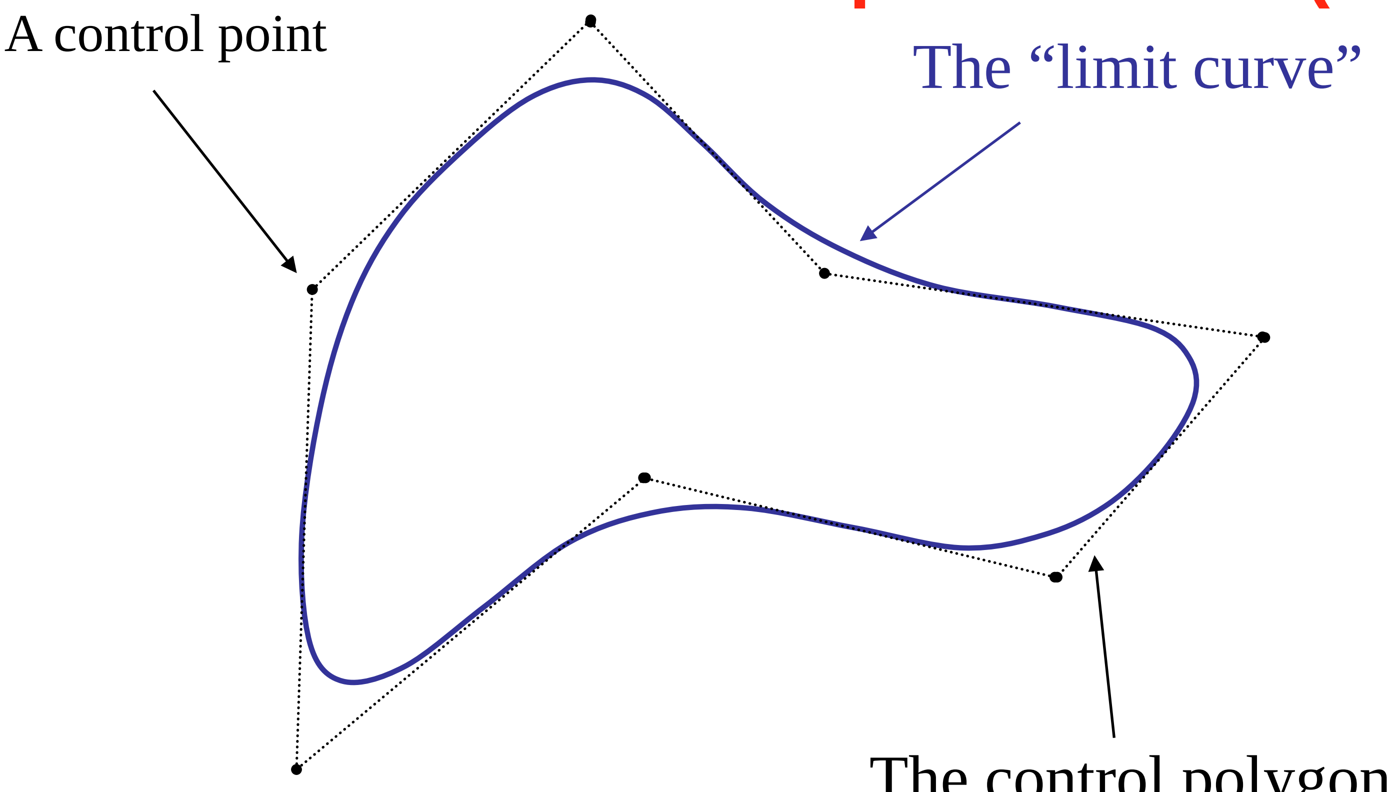
Corner Cutting



Slide by Adi Levin

Corner Cutting

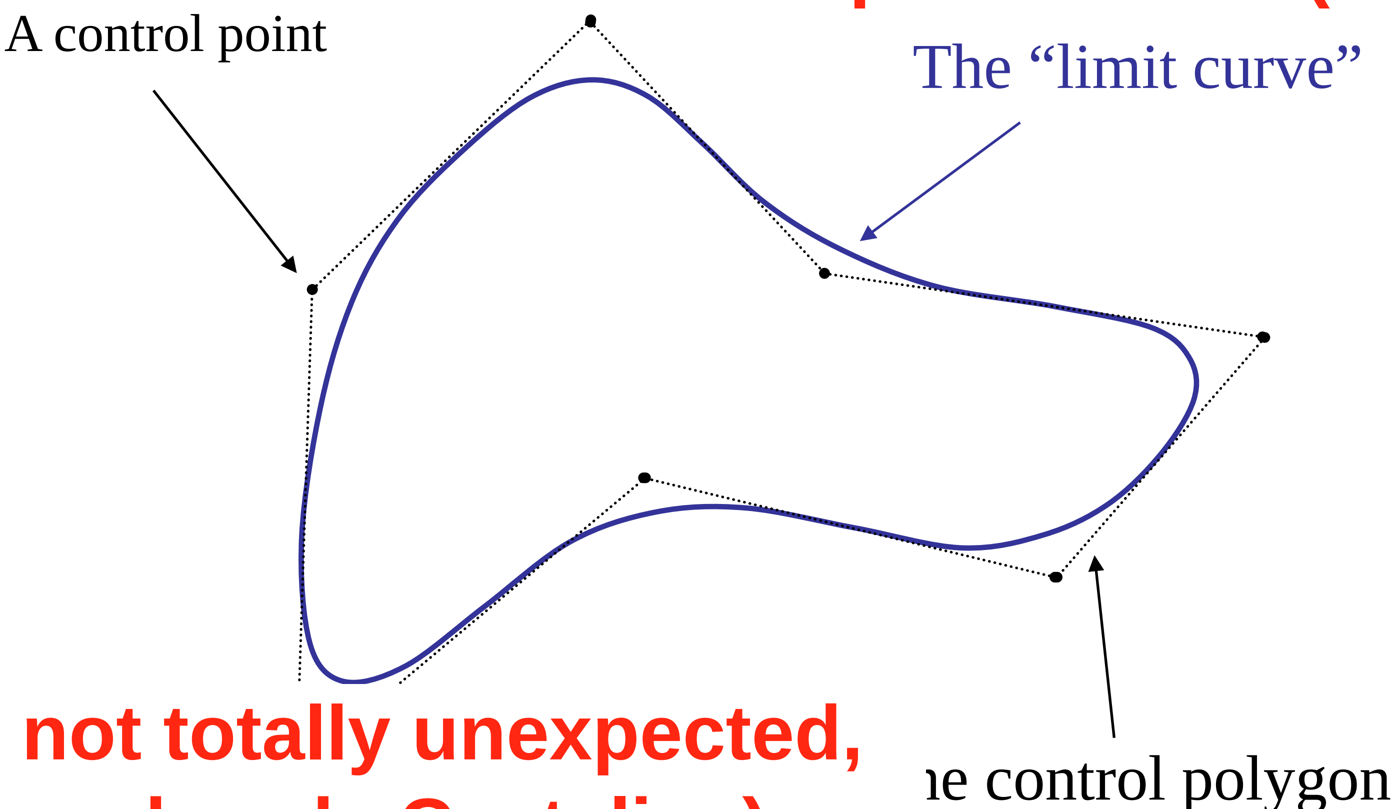
It turns out corner cutting (Chaikin's Algorithm) produces a quadratic B-Spline curve! (Magic!)



Slide by Adi Levin

Corner Cutting

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

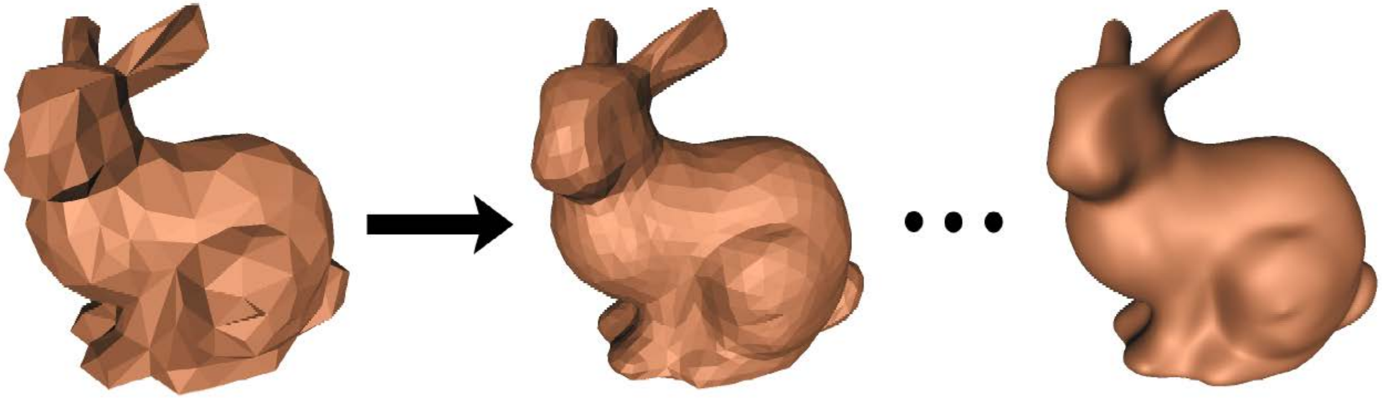


(Well, not totally unexpected, remember de Casteljau)

Slide by Adi Levin

Subdivision Curves and Surfaces

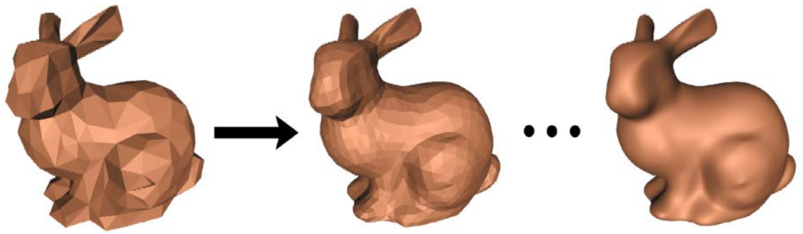
- Idea: cut corners to smooth
- Add points and compute weighted average of neighbors
- Same for surfaces
  - Special case for irregular vertices
    - vertex with more or less than 6 neighbors in a triangle mesh



Warren et al.

Subdivision Curves and Surfaces

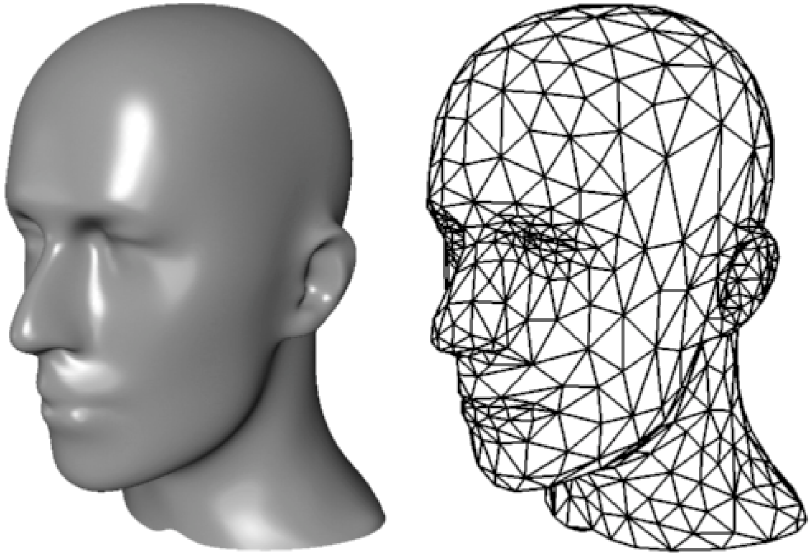
- Advantages
  - Arbitrary topology
  - Smooth at boundaries
  - Level of detail, scalable
  - Simple representation
  - Numerical stability, well-behaved meshes
  - Code simplicity
- Little disadvantage:
  - Procedural definition
  - Not parametric
  - Tricky at special vertices



Warren et al.

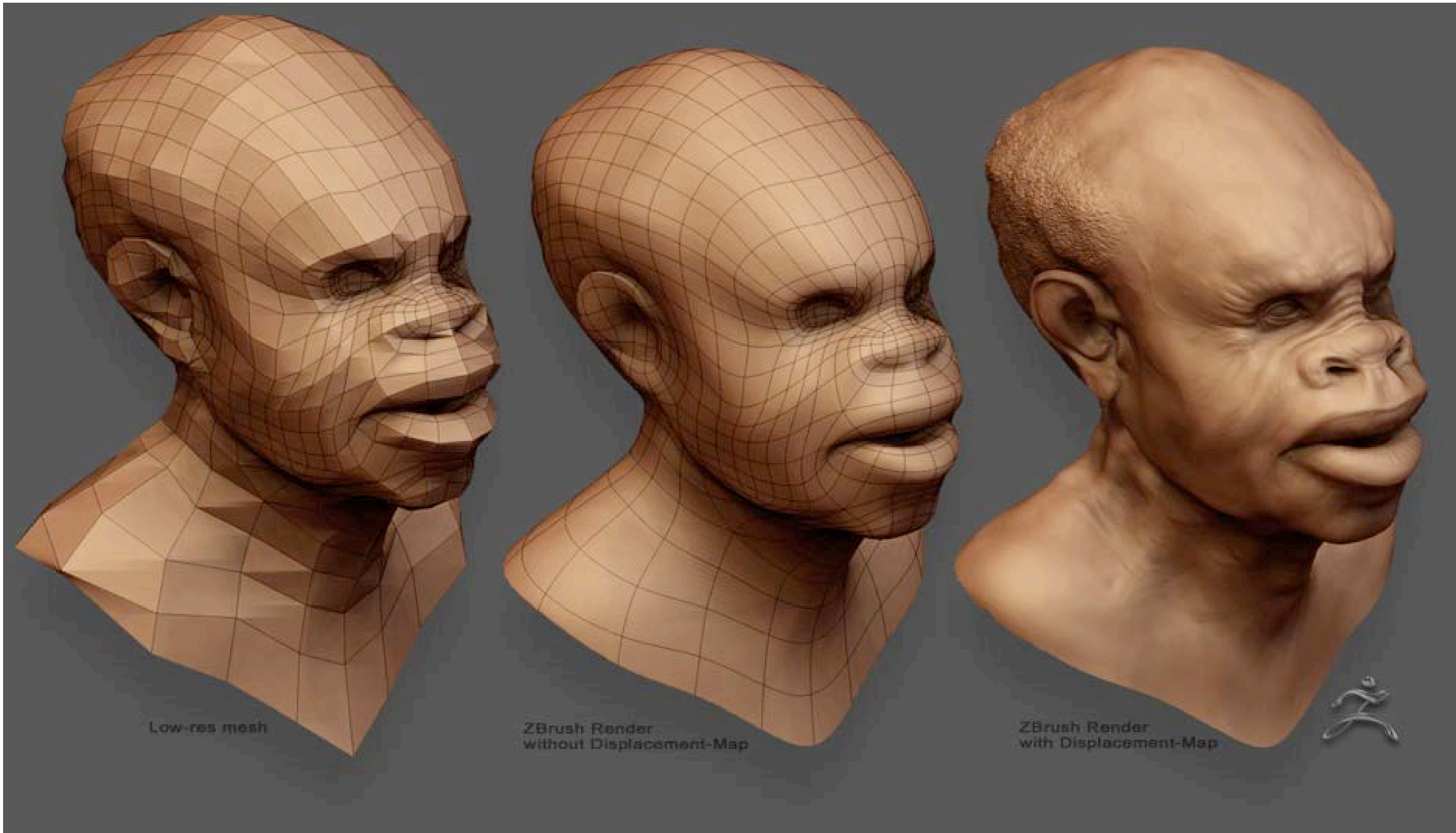
Flavors of Subdivision Surfaces

- 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

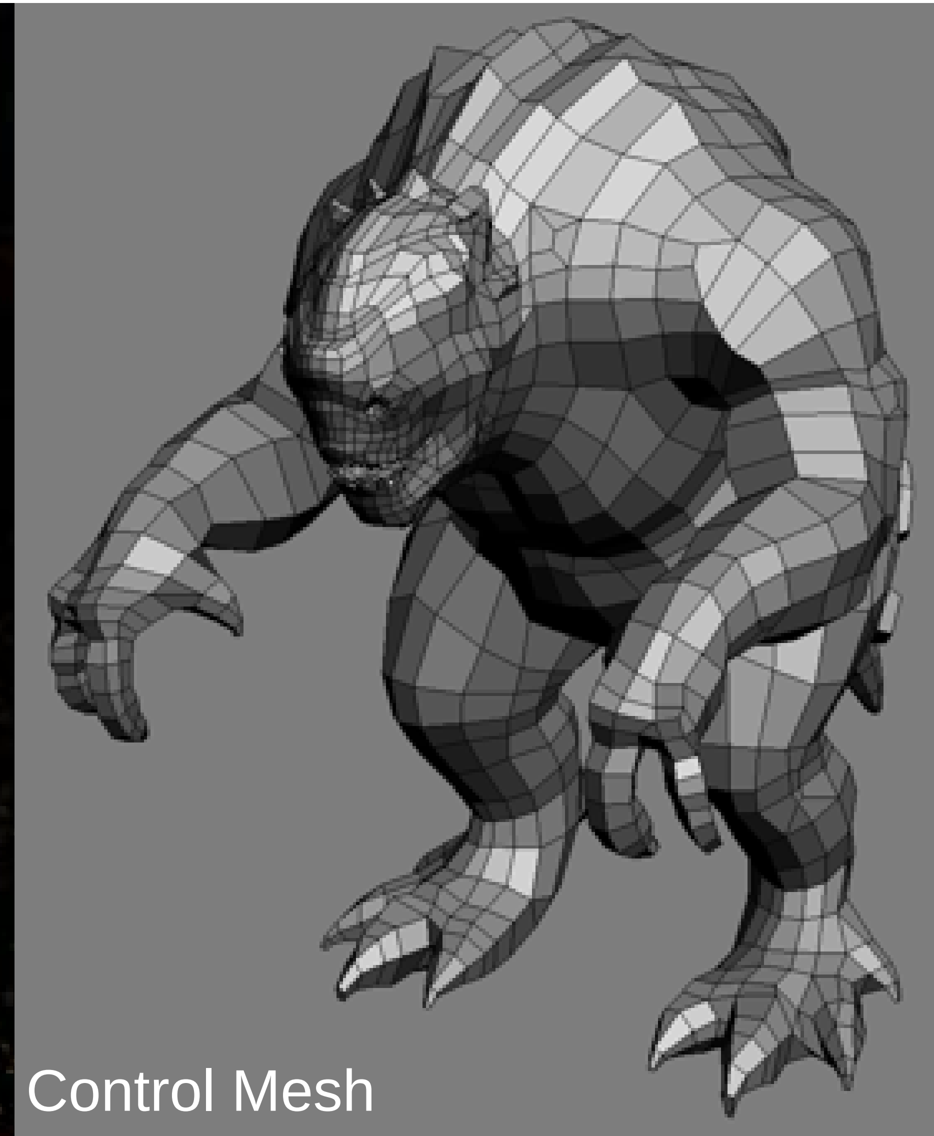
Subdivision + Displacement



zbrushcentral

Subdivision + Displacement

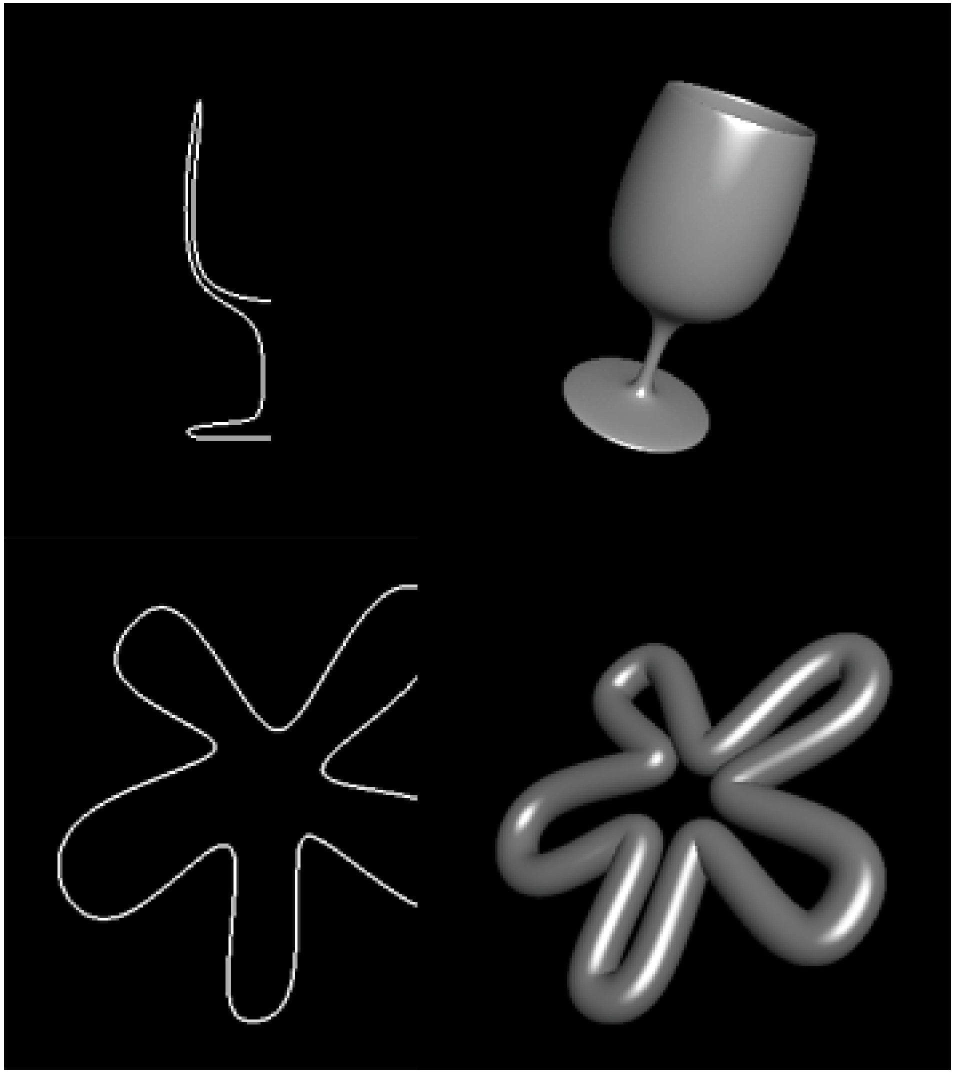
Epic Games



Questions?

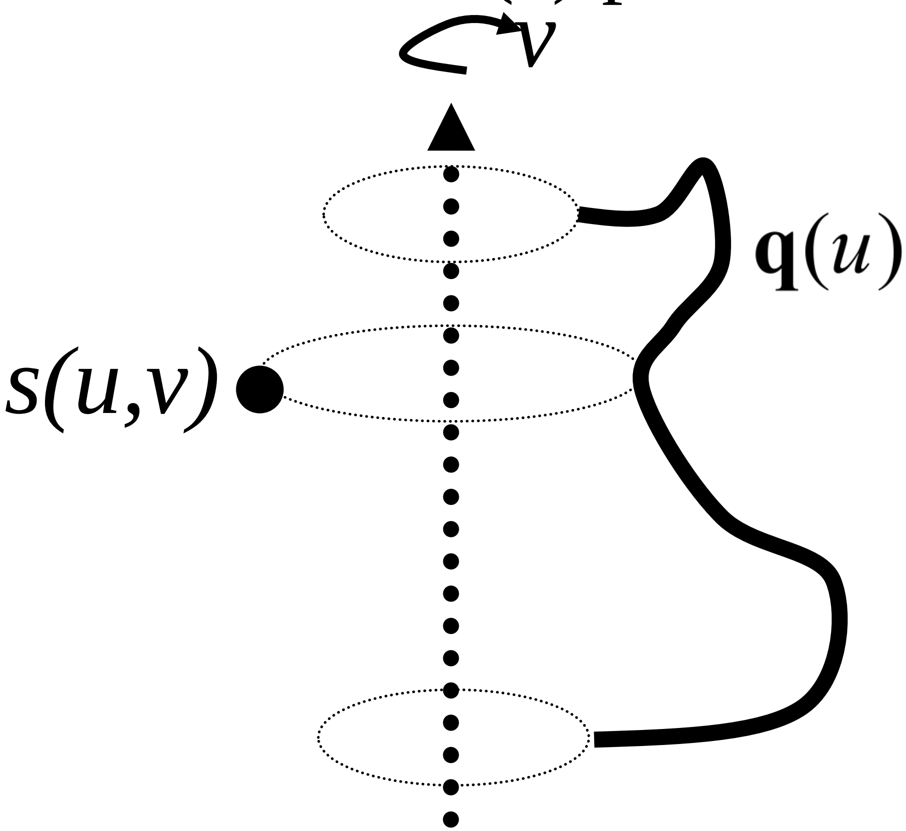
Specialized Procedural Definitions

- 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!**



Surface of Revolution

- 2D curve q(*u*) provides one dimension
  - Note: works also with 3D curve
- Rotation R(*v*) provides 2nd dimension



$\boldsymbol{s}(u,v)=\boldsymbol{R}(v)\boldsymbol{q}(u)$

where ***R*** is a matrix,

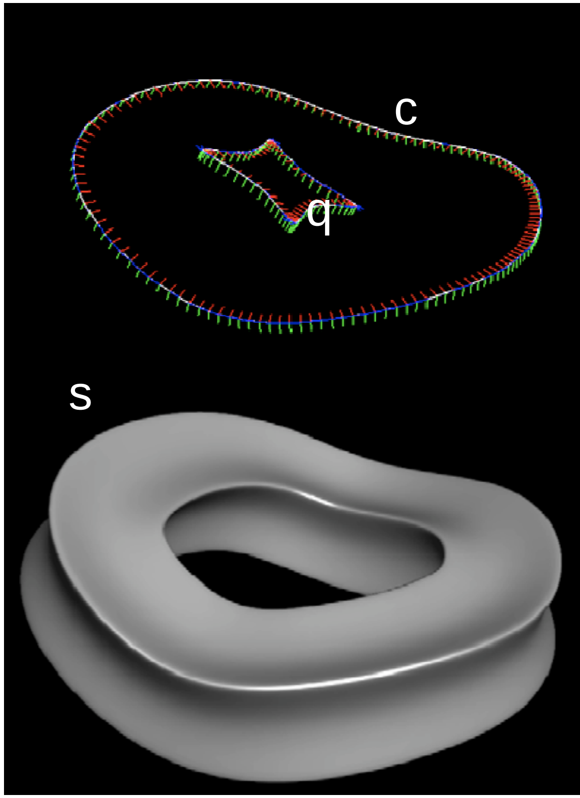
q a vector,

and ***s*** is a point on

the surface

General Swept Surfaces

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

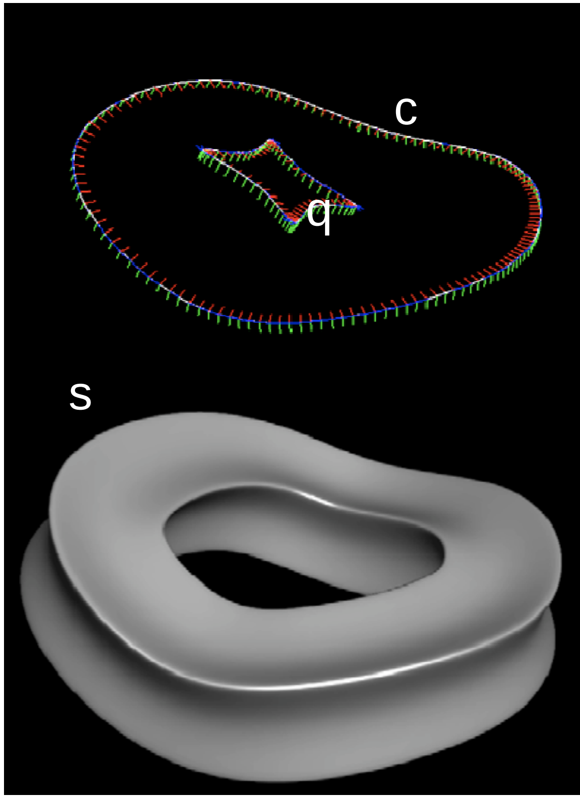


$$\mathbf{s}(u,v)=\mathbf{M}(\mathbf{c}(v))\mathbf{q}(u)$$

where ***M*** is a matrix that depends on the trajectory ***c***

General Swept Surfaces

- How do we get **M**?
  - Translation is easy, given by ***c****(v)*
  - What about orientation?
- Orientation options:
  - Align profile curve with an axis.
  - **Better**: Align profile curve with frame that "follows" the curve

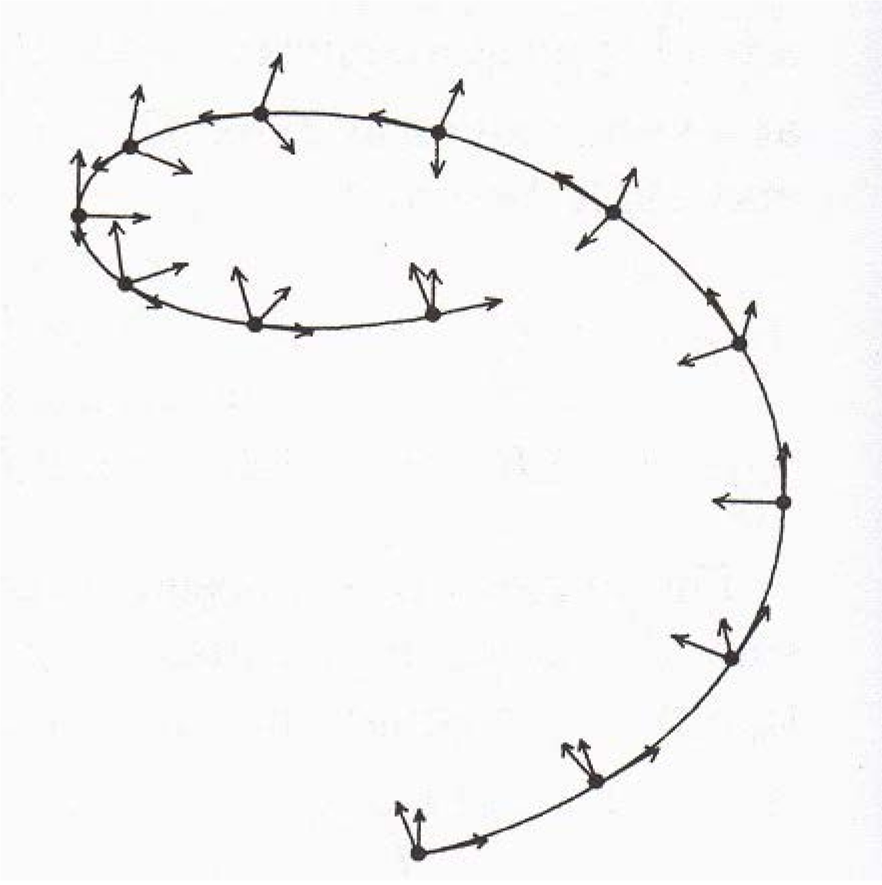


$$\mathbf{s}(u,v)=\mathbf{M}(\mathbf{c}(v))\mathbf{q}(u)$$

where ***M*** is a matrix that depends on the trajectory ***c***

Frames on Curves: Frenet Frame

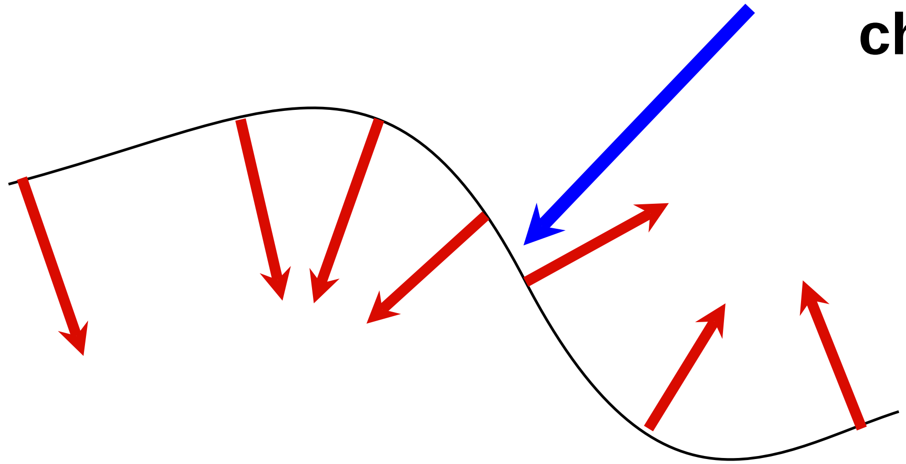
- Frame defined by 1st (tangent), 2nd and 3rd derivatives of a 3D curve
- Looks like a good idea for swept surfaces...



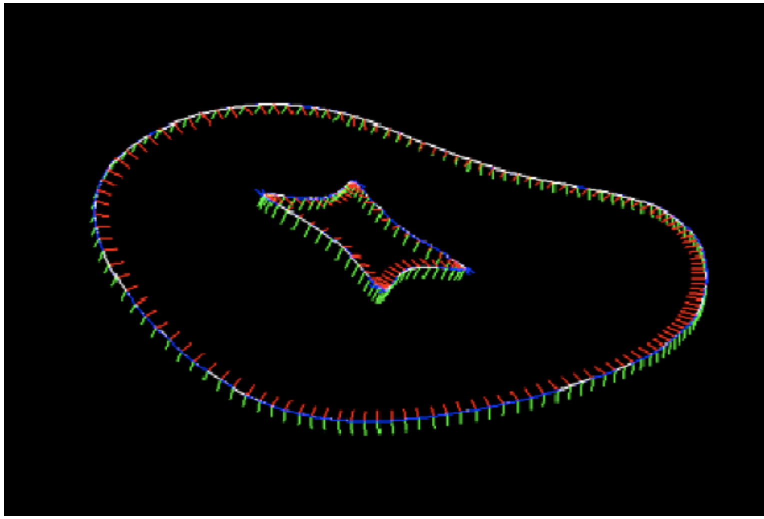
Frenet: Problem at Inflection!

- Normal flips!
- Bad to define a smooth swept surface

An inflection is a point where curvature changes sign



Smooth Frames on Curves



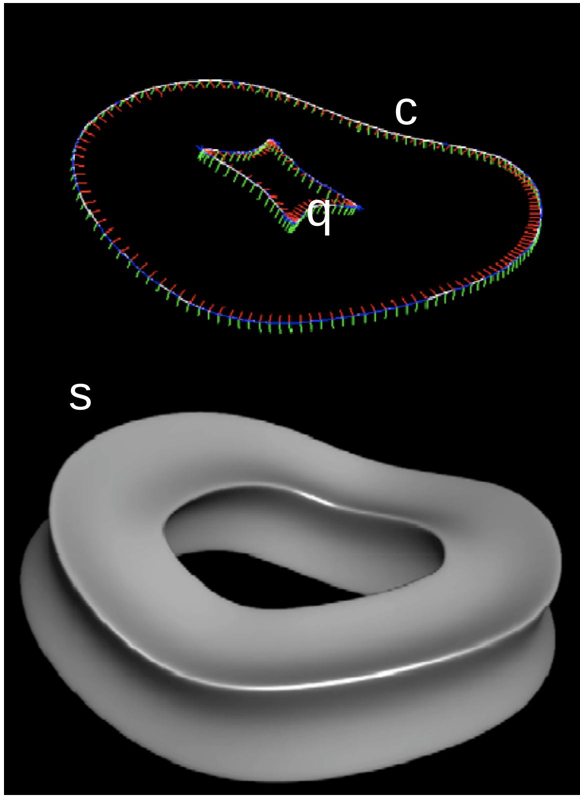
- Build triplet of vectors
  - include tangent (it is reliable)
  - orthonormal
  - coherent over the curve
- Idea:
  - use cross product to create orthogonal vectors
  - exploit discretization of curve
  - use previous frame to bootstrap orientation
  - **See Assignment 1 instructions!**

Normals for Swept Surfaces

- Need partial derivatives w.r.t. both *u* and *v*

$$n = (\partial \mathbf{s}/\partial u) \times (\partial \mathbf{s}/\partial v)$$

  - *Remember to normalize!*
- One given by tangent of profile curve, the other by tangent of trajectory



$$\mathbf{s}(u,v)=\mathbf{M}(\mathbf{c}(v))\mathbf{q}(u)$$

where ***M*** is a matrix that depends on the trajectory ***c***

Questions?

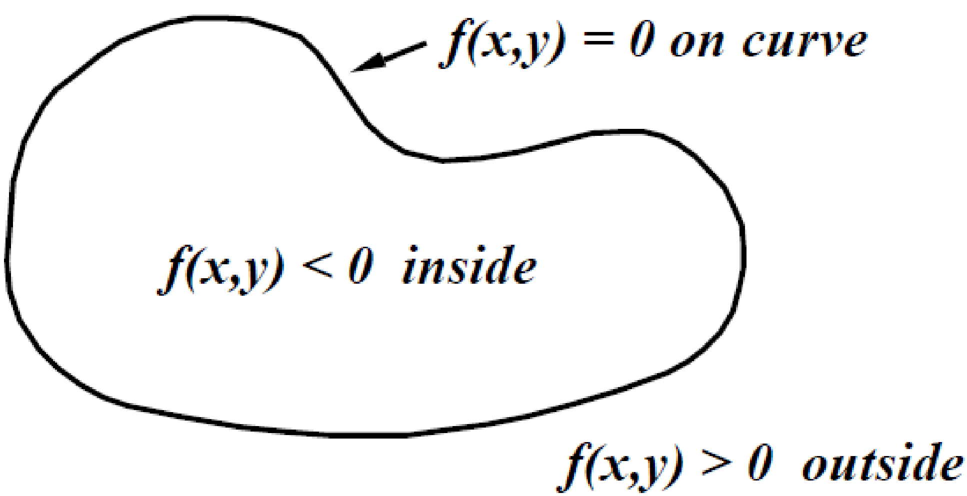
Implicit Surfaces

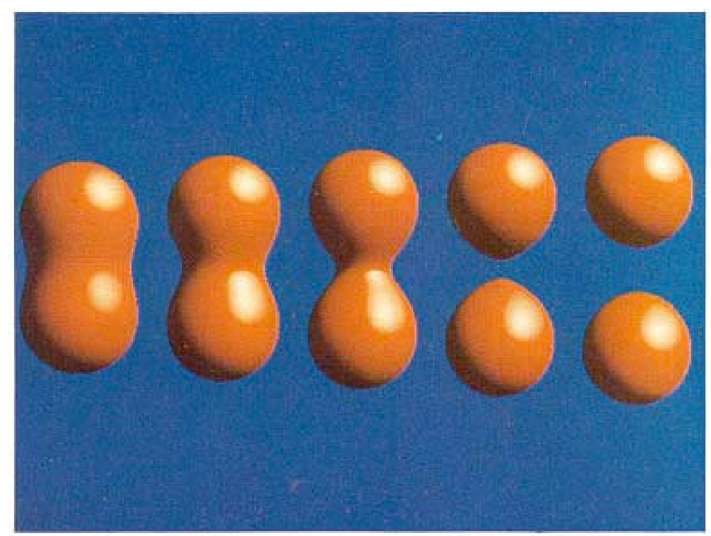
- Surface defined implicitly by a function

  $f(x, y, z) = 0$ (on surface)

  $f(x, y, z) < 0$ (inside)

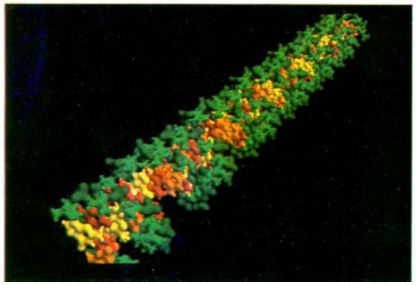
  $f(x, y, z) > 0$ (outside)





From Blinn 1982

Implicit Surfaces

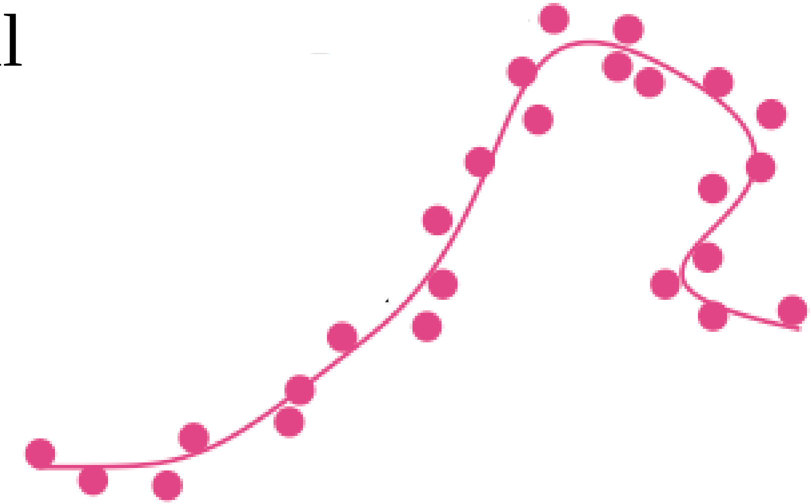


- Pros:
  - Efficient check whether point is inside
  - Efficient Boolean operations
  - Can handle weird topology for animation
  - Easy to do sketchy modeling
- Cons:
  - Does not allow us to easily generate a point on the surface

Questions?

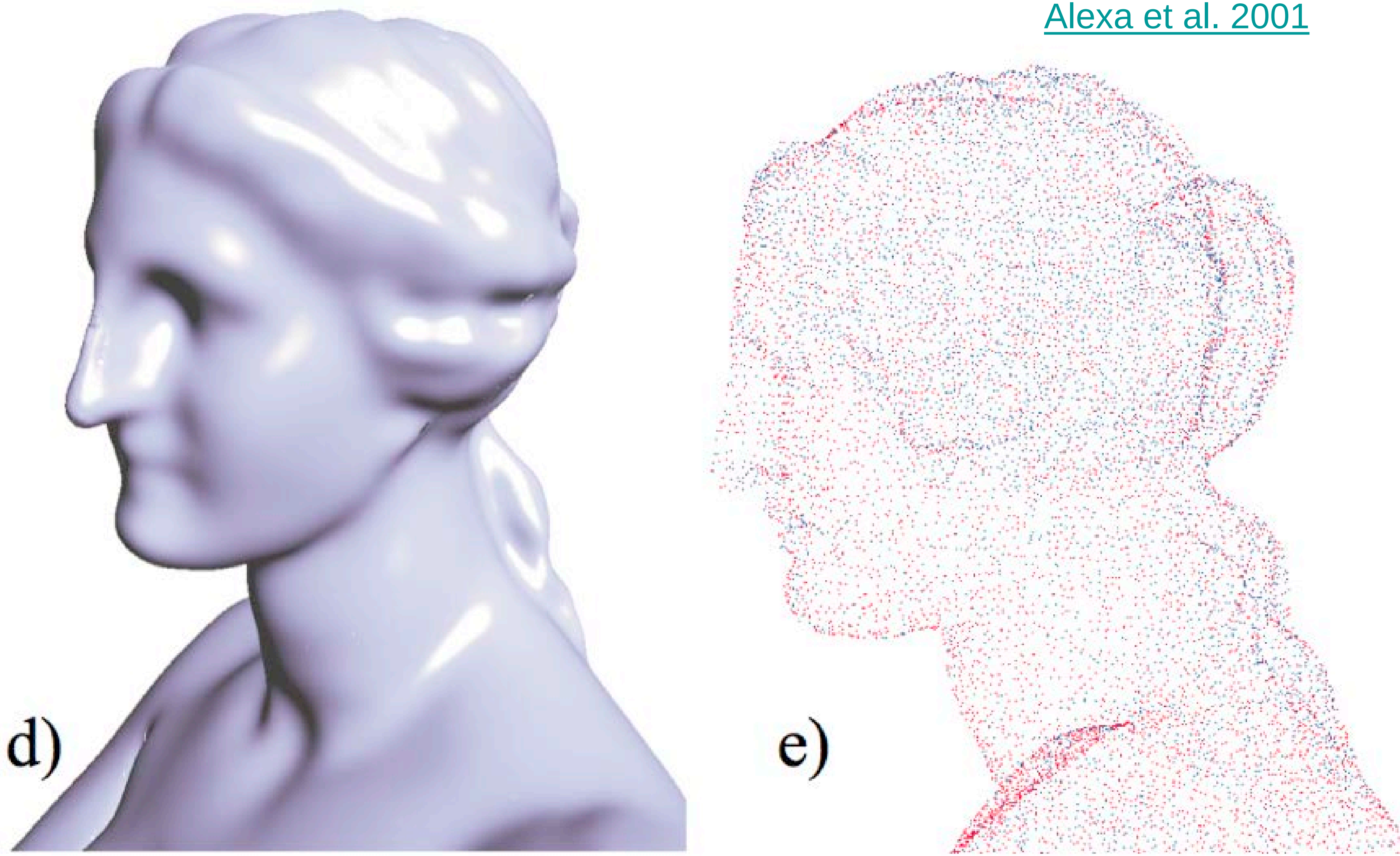
Point Set Surfaces

- 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



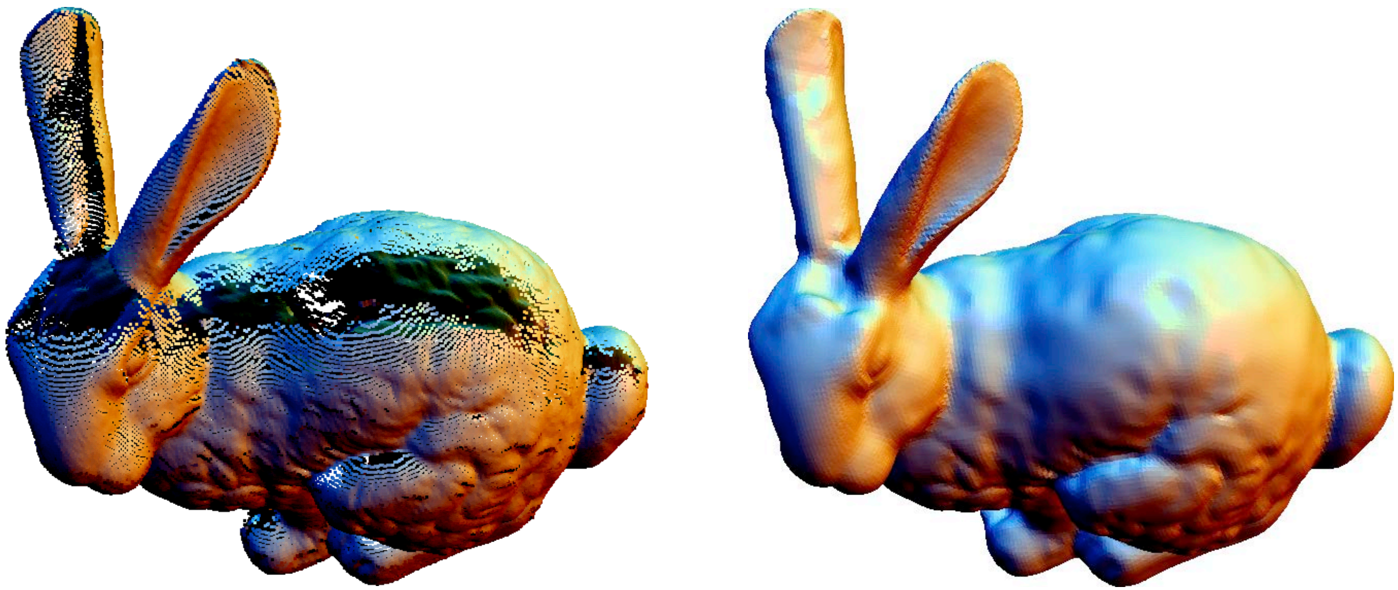
Point Set Surfaces

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.

Questions?

That's All for Today

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