TIEA311
Tietokonegrafiikan perusteet
kevät 2019

(“Principles of Computer Graphics” – Spring 2019)

TIEA311 Tietokonegrafiikan perusteet – kevät 2019 (“Principles of Computer Graphics” – Spring 2019)

Adapted from: *Wojciech Matusik*, and *Frédo Durand*: 6.837 Computer Graphics. Fall 2012. Massachusetts Institute of Technology: MIT OpenCourseWare, https://ocw.mit.edu/.

Frontpage of the local course version, held during Spring 2019 at the Faculty of Information technology, University of Jyväskylä:
`http://users.jyu.fi/~nieminen/tgp19/`

TIEA311 - Today in Jyväskylä

Plan for today:

- Usual warm-up.
- Continue from yesterday
- Go through theory
- Remember to have a break!
- The teacher will try to remember and make use of the fact that we have groups of 2-3 students with pen and paper.

TIEA311 - Local plan for today

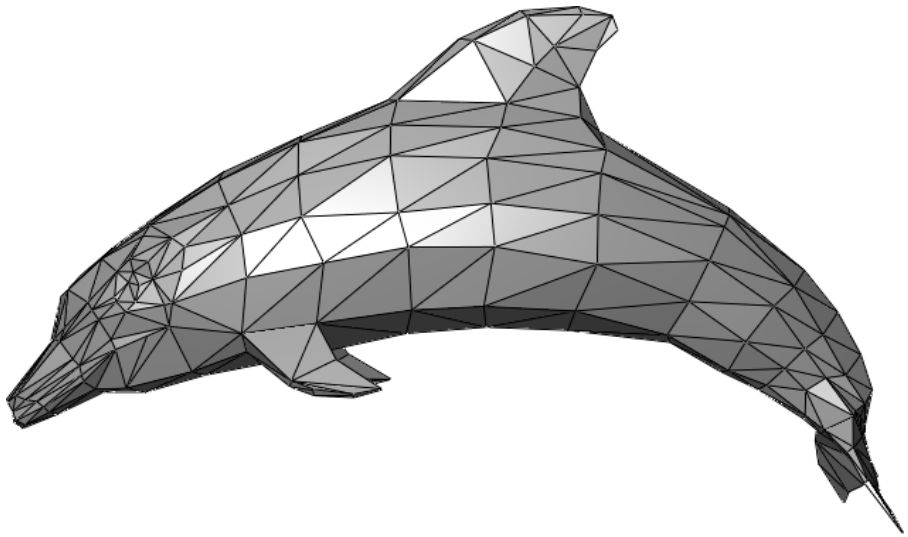
- Maybe some things I forgot to mention yesterday?
- Very brief recap of what went on previously.
- Then forward, with full speed!

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, e.g.** 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)



>> (“fast-forward!”)

On our local course (TIEA311), we skim through the following slides, grabbing ideas and keywords without detail. Note to self: apply **great speed** with the “next slide” button!

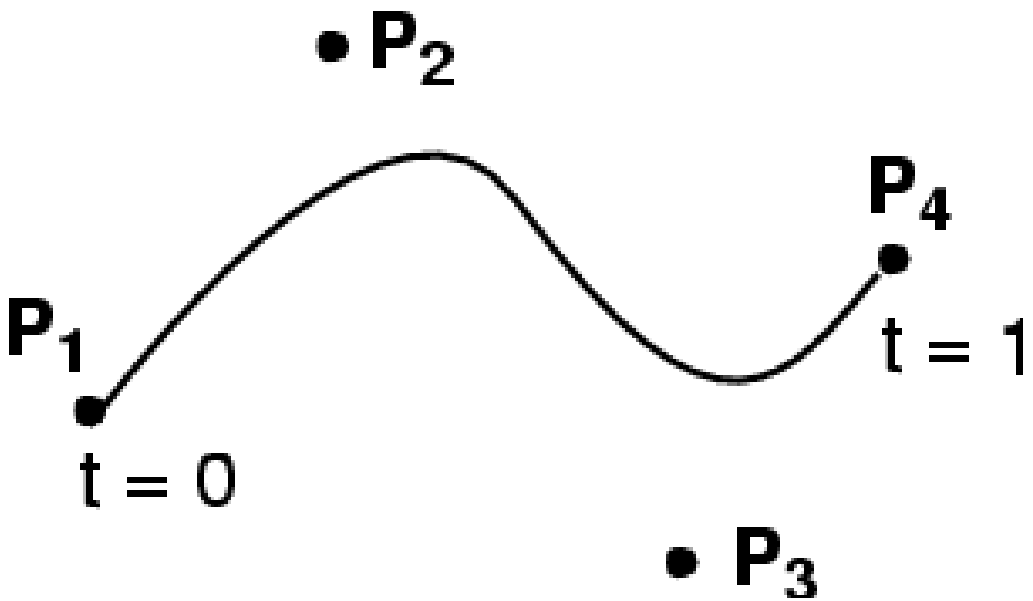
Rationale:

- We **need to know** about what is possible.
- These things are omnipresent in real-world graphics libraries, and CAD and CGI software, so we must **understand what they do** in order to **apply them more knowingly**.
- Examples to **motivate further math studies** – the ultimate goal of a computer science student should be the skills to **build** and **improve** the said libraries and software for the artists and engineers to use.
- If we need some of the concepts or notations again on this course, **we’ll return to them** with further explanation.

Smooth Surfaces?

- $P(t) = (1-t)^3$ P1
  $+ \quad 3t(1-t)^2$ P2
  $+ \quad 3t^2(1-t)$ P3
  $+ \quad t^3$ P4

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

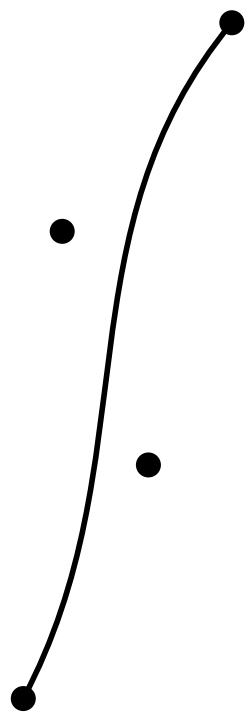


What about a surface?

How to Build Them? Here's an Idea

- $P(u) = (1-u)^3$ P1
  $+ \quad 3u(1-u)^2$ P2
  $+ \quad 3u^2(1-u)$ P3
  $+ \quad u^3$ P4

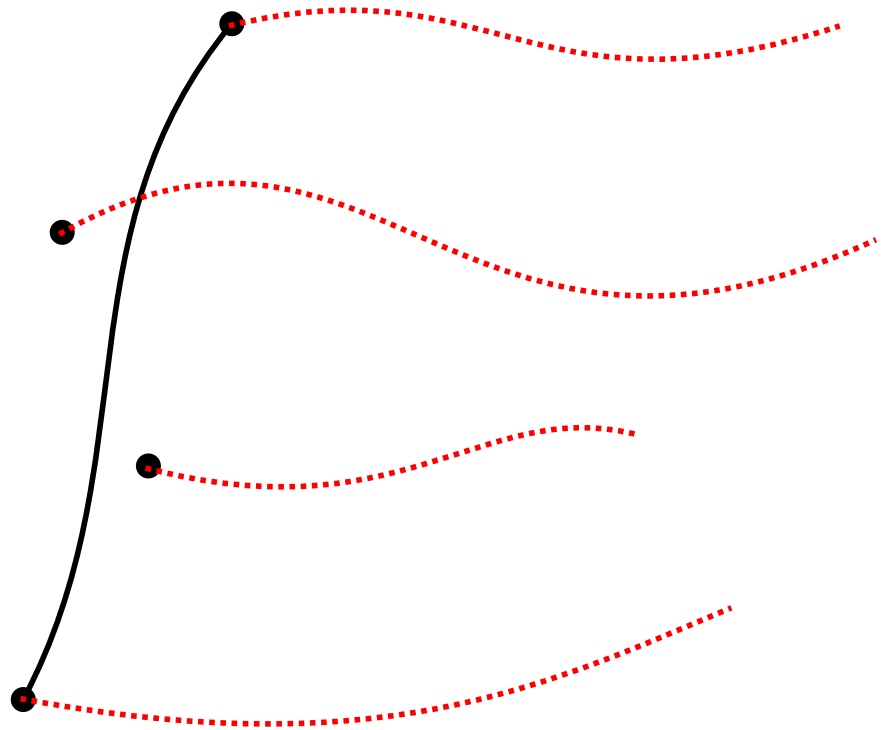
(Note! We relabeled *t* to *u*)



How to Build Them? Here's an Idea

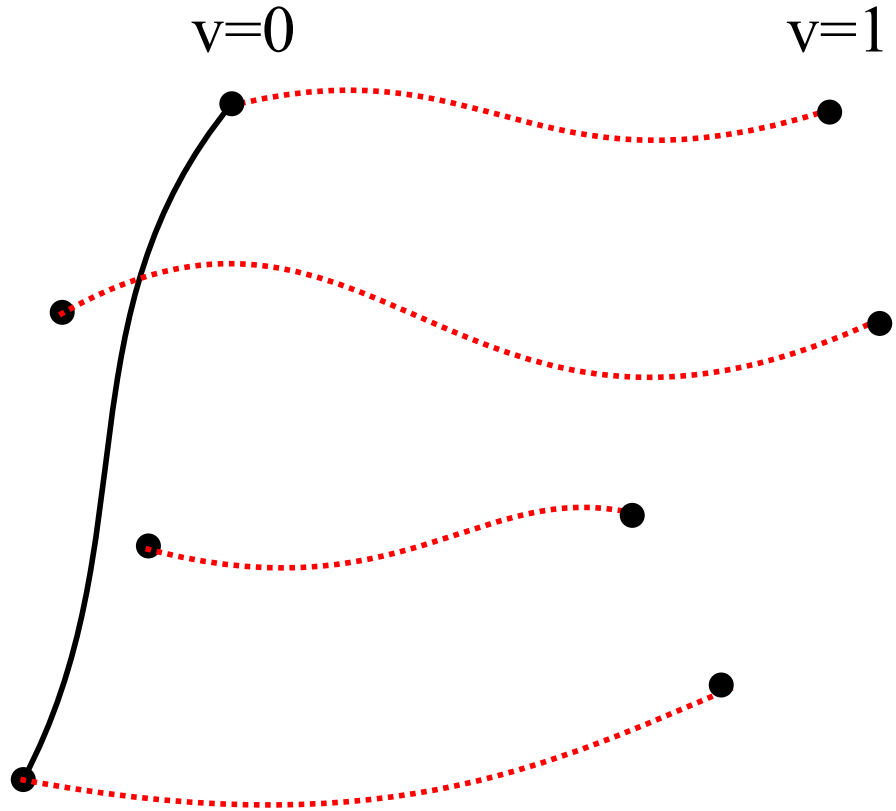
- $P(u) = (1-u)^3$ P1
  $+ \quad 3u(1-u)^2$ P2
  $+ \quad 3u^2(1-u)$ P3
  $+ \quad u^3$ P4

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



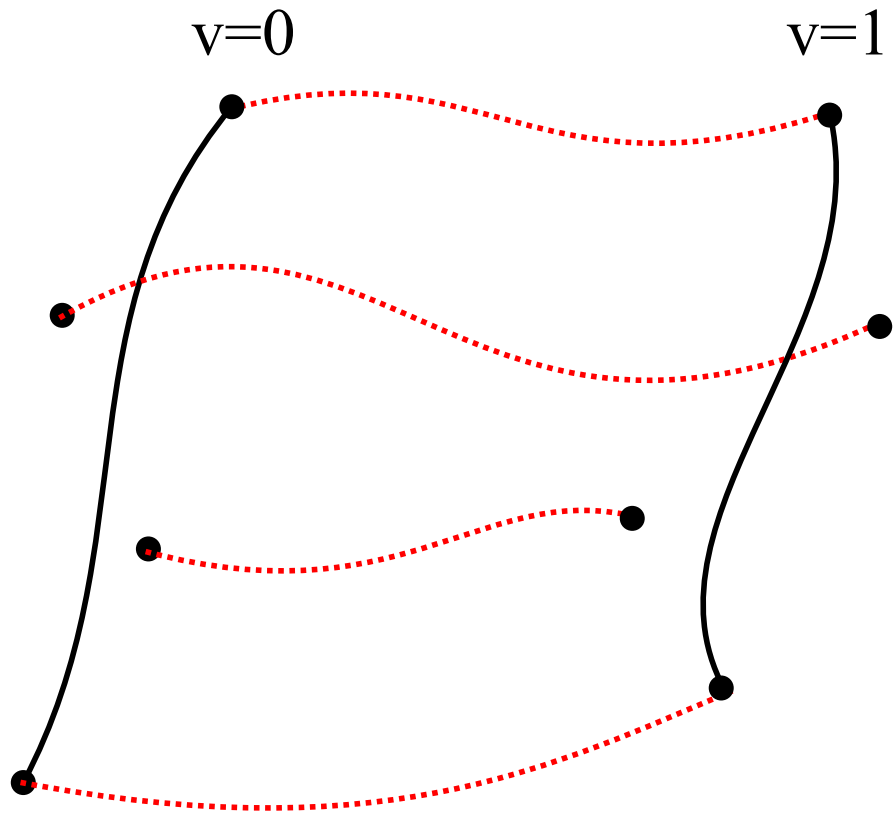
Here's an Idea

- $P(u, v) = (1-u)^3 \; P1(v)$
  $+ \; 3u(1-u)^2 \; P2(v)$
  $+ \; 3u^2(1-u) \; P3(v)$
  $+ \; u^3 \; P4(v)$
- Let's make the Pis move along curves!



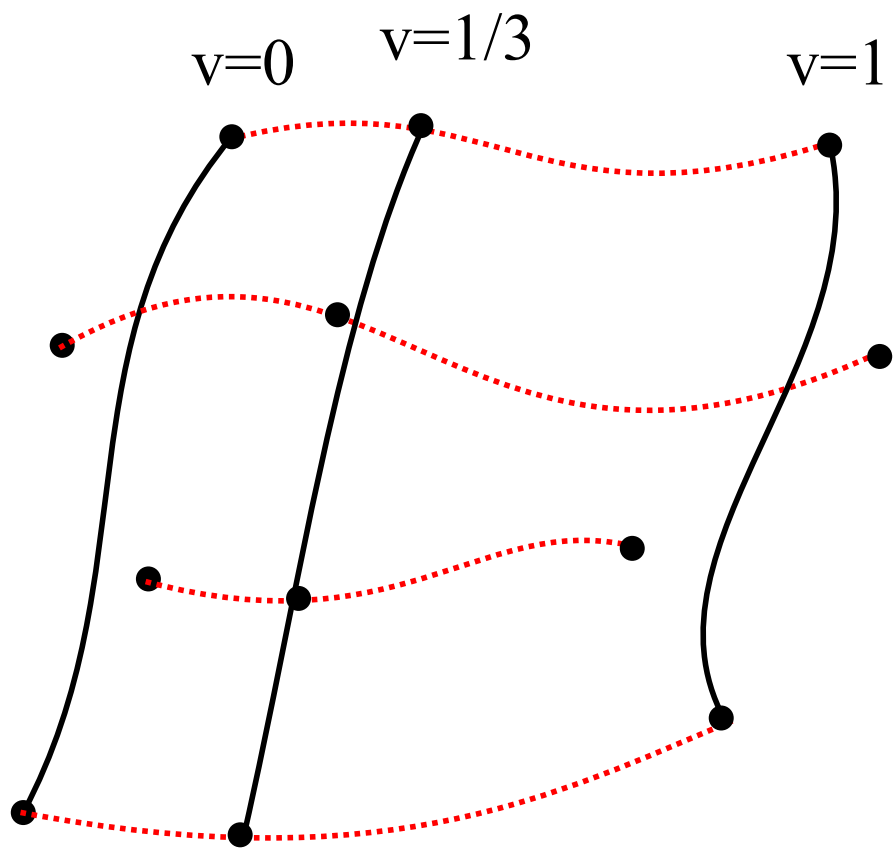
Here's an Idea

- $P(u, v) = (1-u)^3 \; P1(v)$
  $+ \; 3u(1-u)^2 \; P2(v)$
  $+ \; 3u^2(1-u) \; P3(v)$
  $+ \; u^3 \; P4(v)$
- Let's make the Pis move along curves!



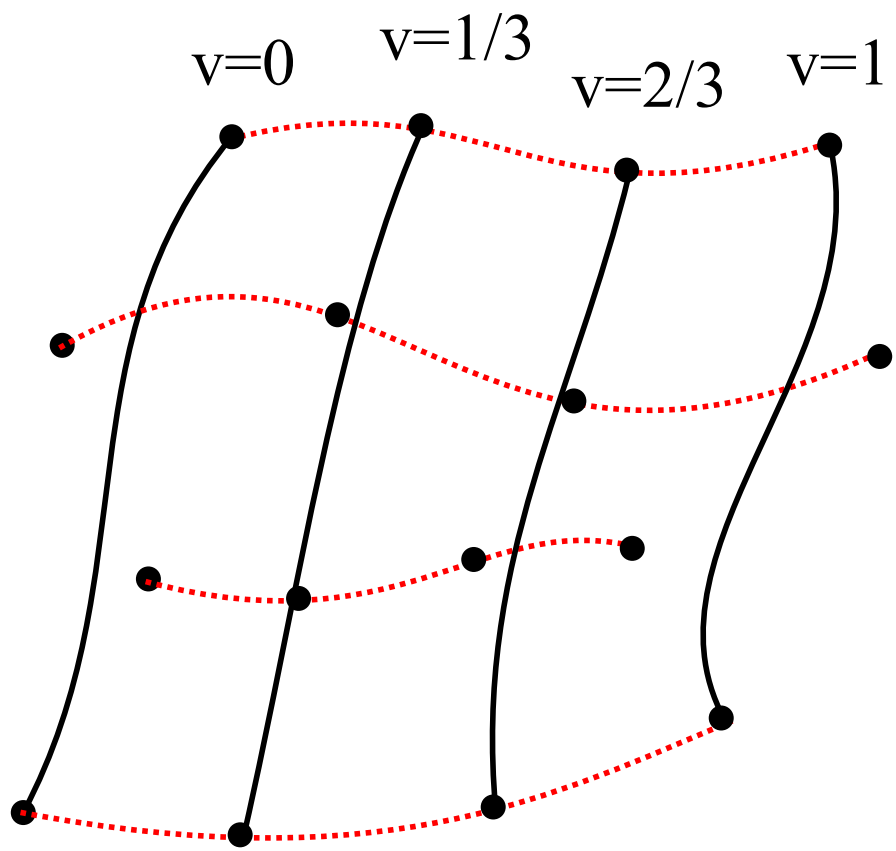
Here's an Idea

- $P(u, v) = (1-u)^3 \; P1(v)$
  $+ \; 3u(1-u)^2 \; P2(v)$
  $+ \; 3u^2(1-u) \; P3(v)$
  $+ \; u^3 \; P4(v)$
- Let's make the Pis move along curves!



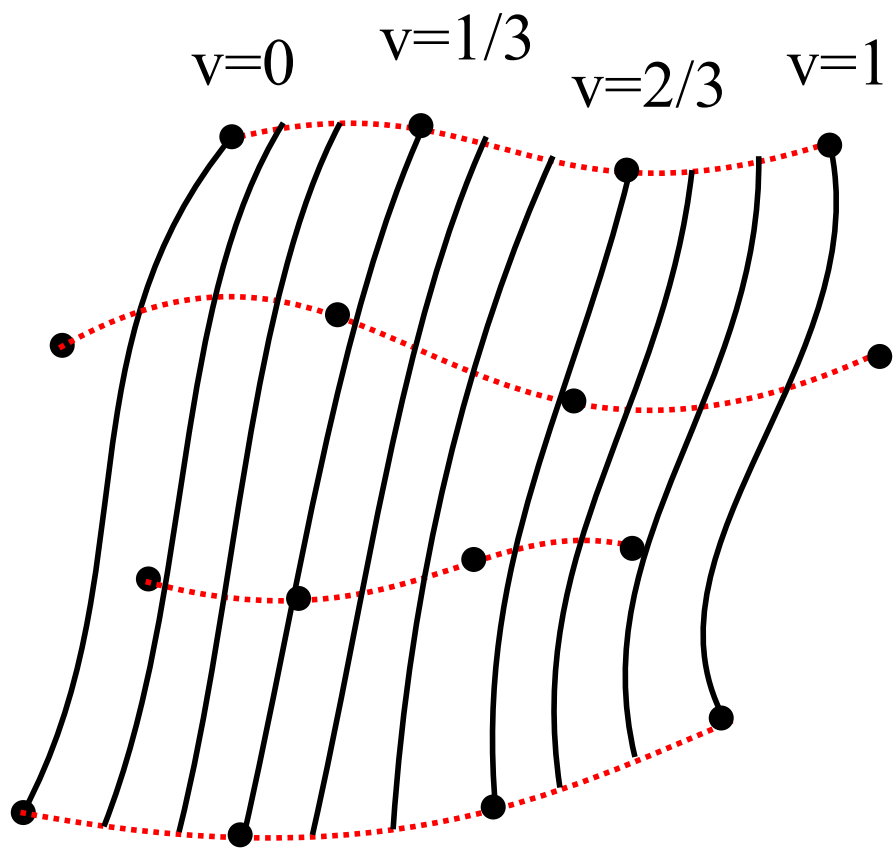
Here's an Idea

- $P(u, v) = (1-u)^3 \; P1(v)$
  $+ \; 3u(1-u)^2 \; P2(v)$
  $+ \; 3u^2(1-u) \; P3(v)$
  $+ \; u^3 \; P4(v)$
- Let's make the Pis move along curves!



Here's an Idea

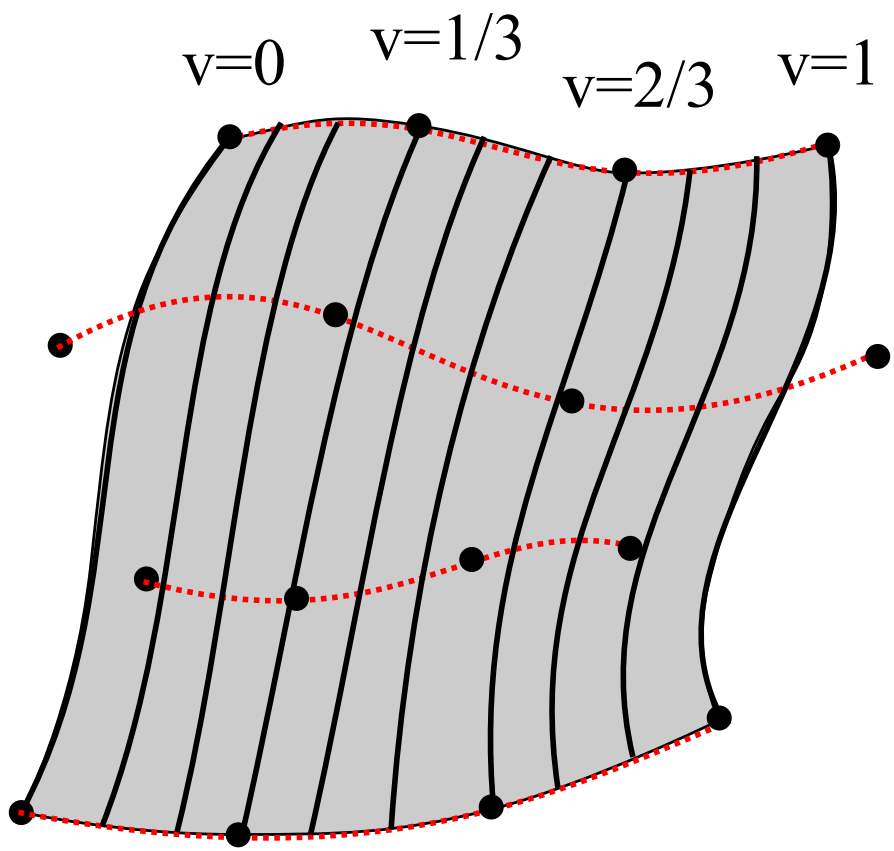
- $P(u, v) = (1-u)^3 \; P1(v)$
  $+ \; 3u(1-u)^2 \; P2(v)$
  $+ \; 3u^2(1-u) \; P3(v)$
  $+ \; u^3 \; P4(v)$
- Let's make the Pis move along curves!



Here's an Idea

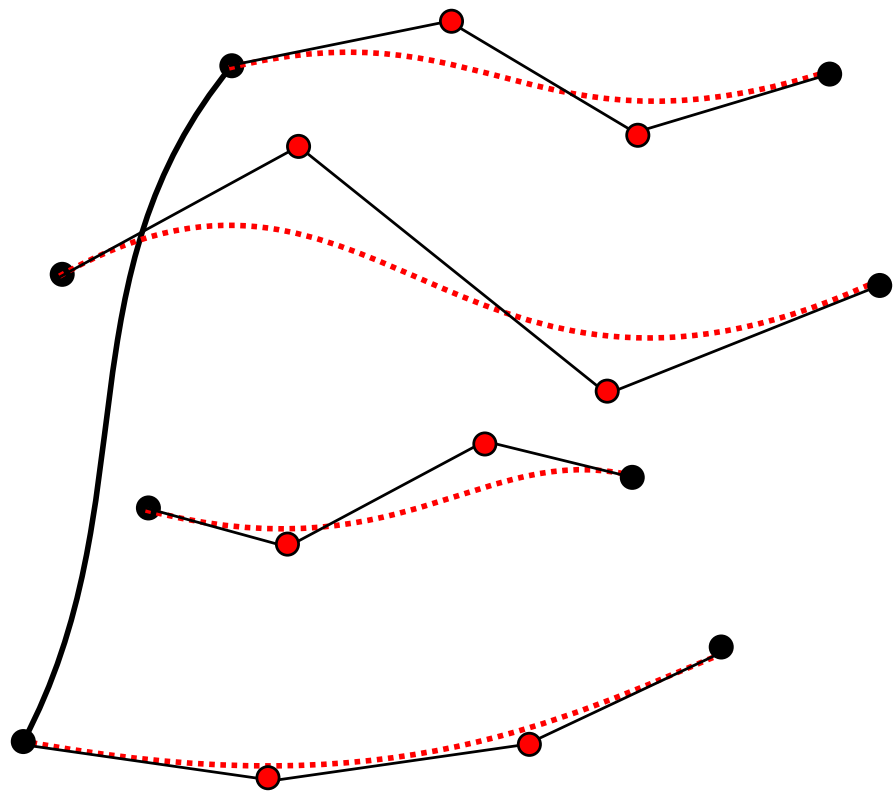
- $P(u, v) = (1-u)^3 \; P1(v)$
  $+ \; 3u(1-u)^2 \; P2(v)$
  $+ \; 3u^2(1-u) \; P3(v)$
  $+ \; u^3 \; P4(v)$
- Let's make the Pis move along curves!

A 2D surface patch!



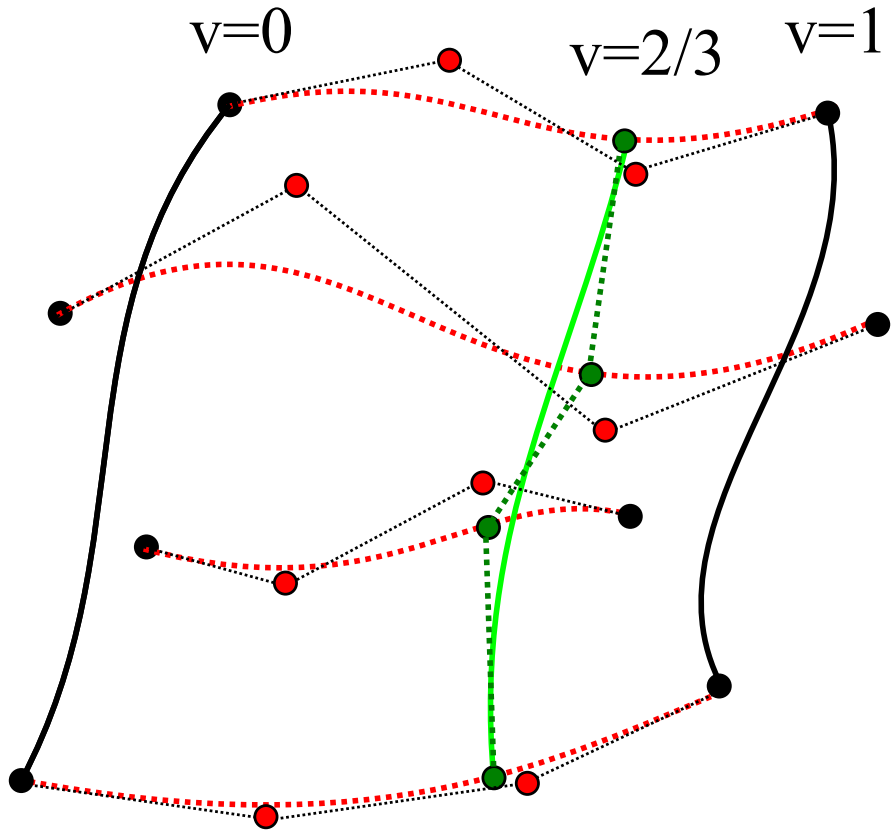
Tensor Product Bézier Patches

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



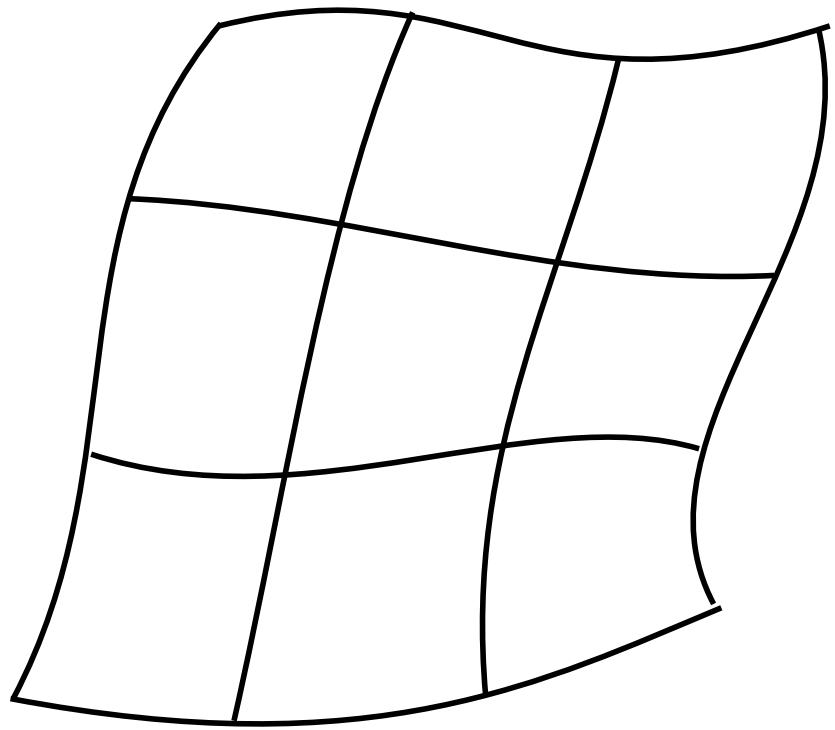
Tensor Product Bézier Patches

- 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!



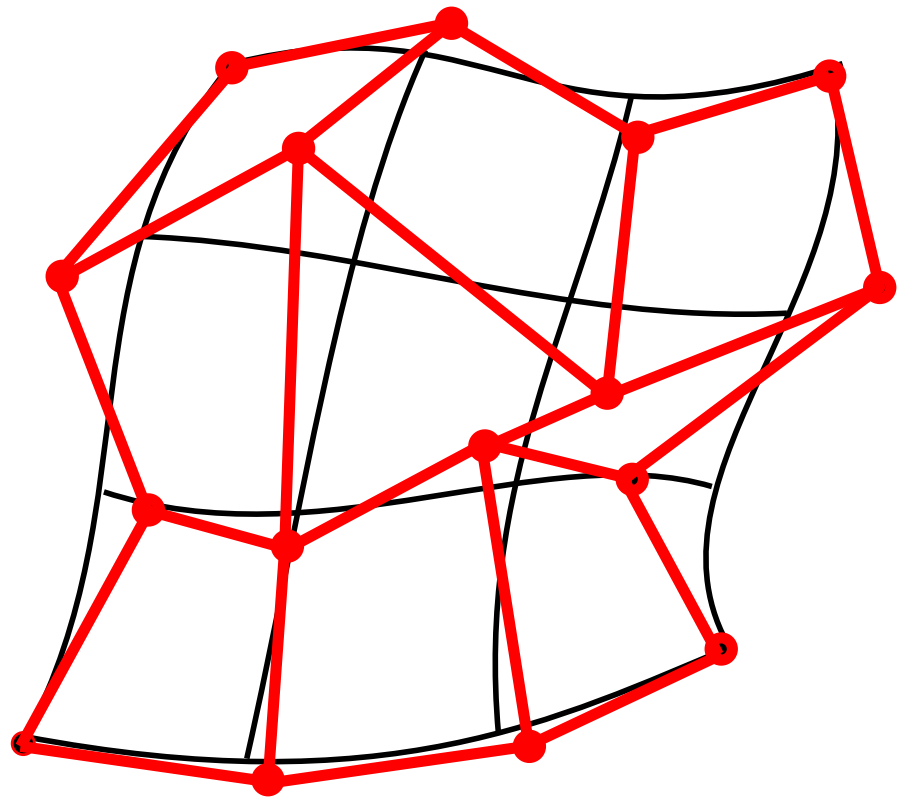
Tensor Product Bézier Patches

A **bicubic Bézier surface**



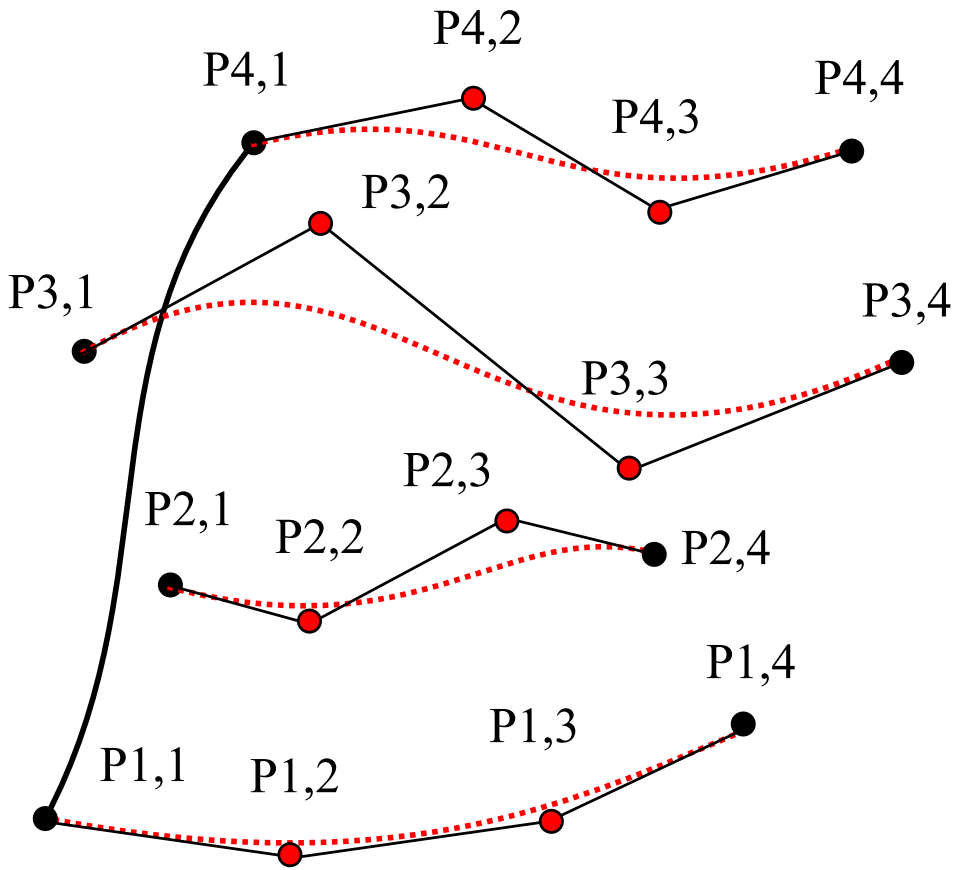
Tensor Product Bézier Patches

The “Control Mesh”
16 control points



Bicubics, Tensor Product

- P(u,v) = B1(u) * P1(v)
  + B2(u) * P2(v)
  + B3(u) * P3(v)
  + B4(u) * P4(v)
- Pi(v) = B1(v) * Pi,1
  + B2(v) * Pi,2
  + B3(v) * Pi,3
  + B4(v) * Pi,4



Bicubics, Tensor Product

- P(u,v) = B1(u) * P1(v)
  + B2(u) * P2(v)
  + B3(u) * P3(v)
  + B4(u) * P4(v)
- Pi(v) = B1(v) * Pi,1
  + B2(v) * Pi,2
  + B3(v) * Pi,3
  + B4(v) * Pi,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) * P1(v)
  + B2(u) * P2(v)
  + B3(u) * P3(v)
  + B4(u) * P4(v)
- Pi(v) = B1(v) * Pi,1
  + B2(v) * Pi,2
  + B3(v) * Pi,3
  + B4(v) * Pi,4

$$P(u,v) =$$

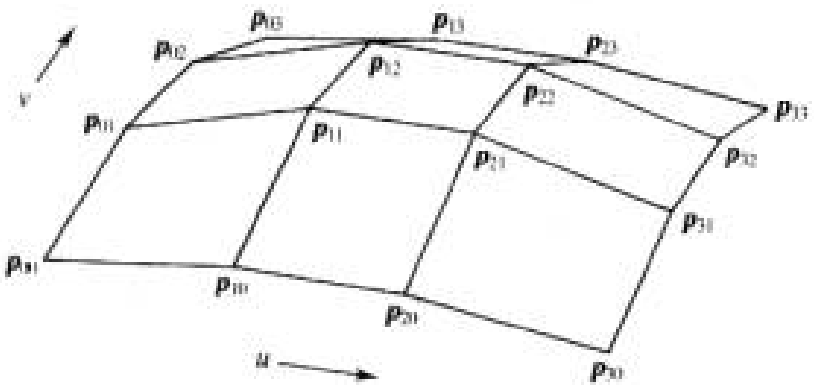
16 control points Pi,j
16 2D basis functions Bi,j

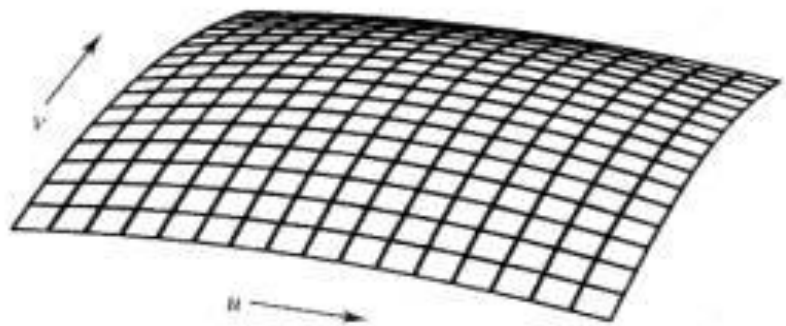
$$= \sum_{i=1}^{4} \sum_{j=1}^{4} P_{i,j} B_{i,j}(u,v)$$

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

Recap: Tensor Bézier Patches

- Parametric surface P(u,v) is a bicubic polynomial of two variables u & v
- Defined by 4x4=16 control points P1,1, P1,2.... P4,4
- Interpolates 4 corners, approximates others
- Basis are product of two Bernstein polynomials: B1(u)B1(v); B1(u)B2(v);... B4(u)B4(v)

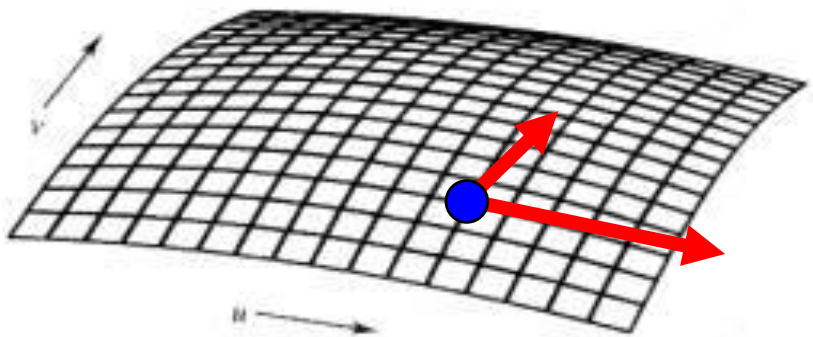




Questions?

Tangents and Normals for Patches

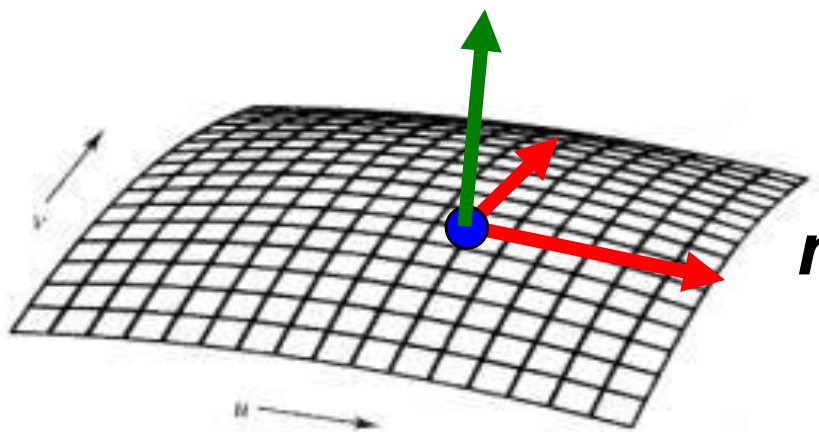
- P(u,v) is a 3D point specified by *u*, *v*
- The partial derivatives $\partial P/\partial u$ and $\partial P/\partial v$ are 3D vectors
  - Both are tangent to surface at P



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)

$$\mathrm{P(t)} = \begin{pmatrix} x(t) \\ y(t) \end{pmatrix} = \begin{pmatrix} x_1 & x_2 & x_3 & x_4 \\ y_1 & y_2 & y_3 & y_4 \end{pmatrix} \begin{pmatrix} 1 & -3 & 3 & -1 \\ 0 & 3 & -6 & 3 \\ 0 & 0 & 3 & -3 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ t \\ t^2 \\ t^3 \end{pmatrix}$$

Canonical
"power basis"

"Geometry matrix"
of control points P1..P4
(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*)

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

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 ***G***x, ***G***y, ***G***z matrices into a geometry **tensor** of control points
  - I.e., Gki,j = the *kth* coordinate of control point Pi,j
  - A cube of numbers!

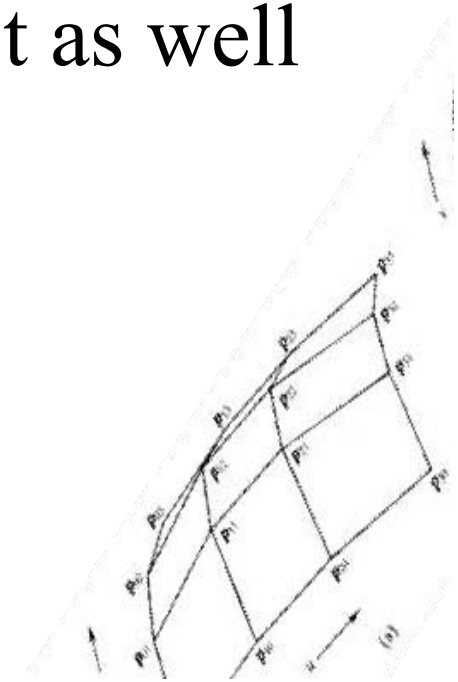
$$P^k(u,v) = \boldsymbol{T}^l(u)\, \boldsymbol{B}^i_l\, \boldsymbol{G}^k_{ij}\, \boldsymbol{B}^j_m\, \boldsymbol{T}^m(v)$$

- "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

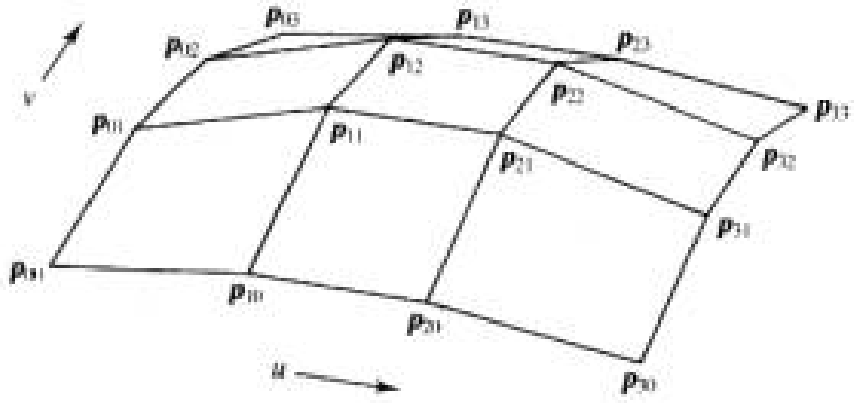
- Pros
  - Smooth
  - Defined by reasonably small set of points
- Cons
  - Harder to render (usually converted to triangles)
  - Tricky to ensure continuity at patch boundaries
- Extensions
  - Rational splines: Splines in homogeneous coordinates
  - NURBS: Non-Uniform Rational B-Splines
    - Like curves: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

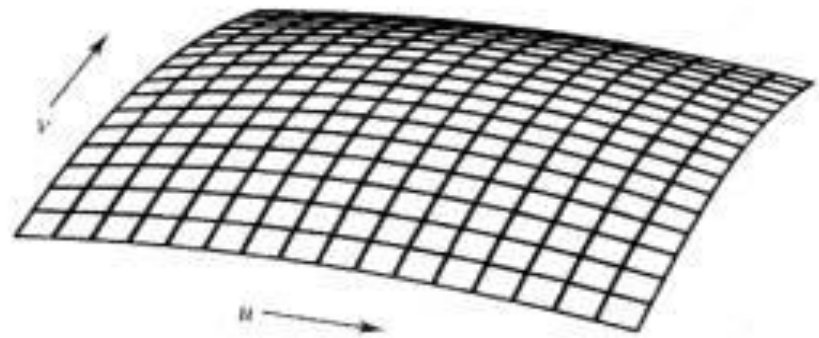
Utah Teapot: Tensor Bézier Splines

- Designed by Martin Newell

Cool: Displacement Mapping

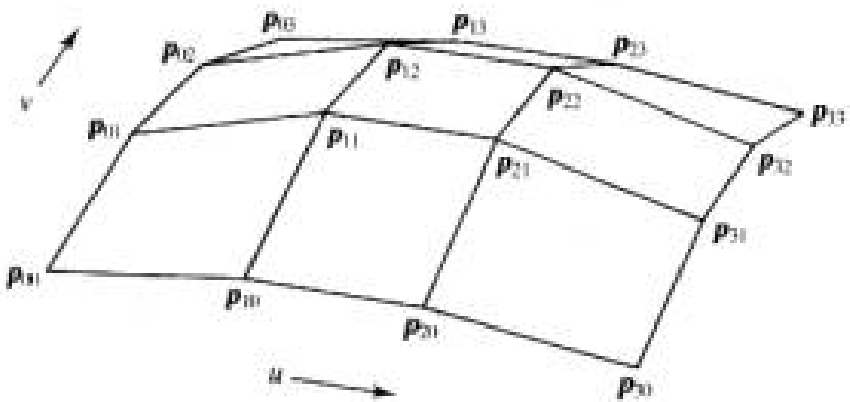
- Not all surfaces are smooth...

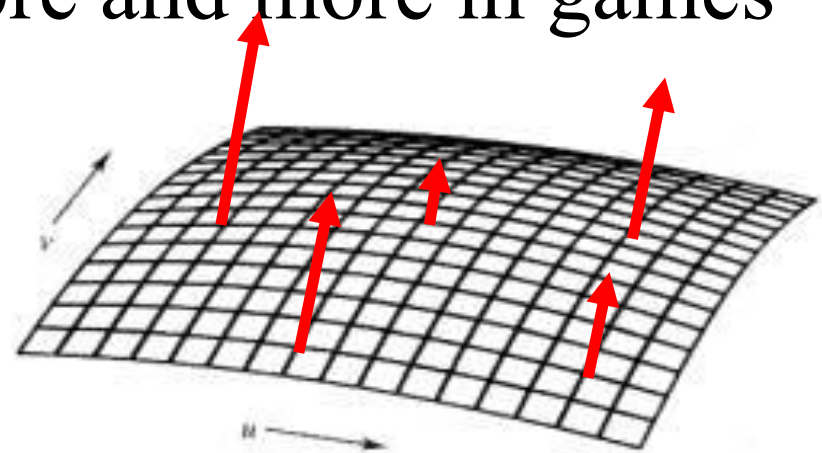




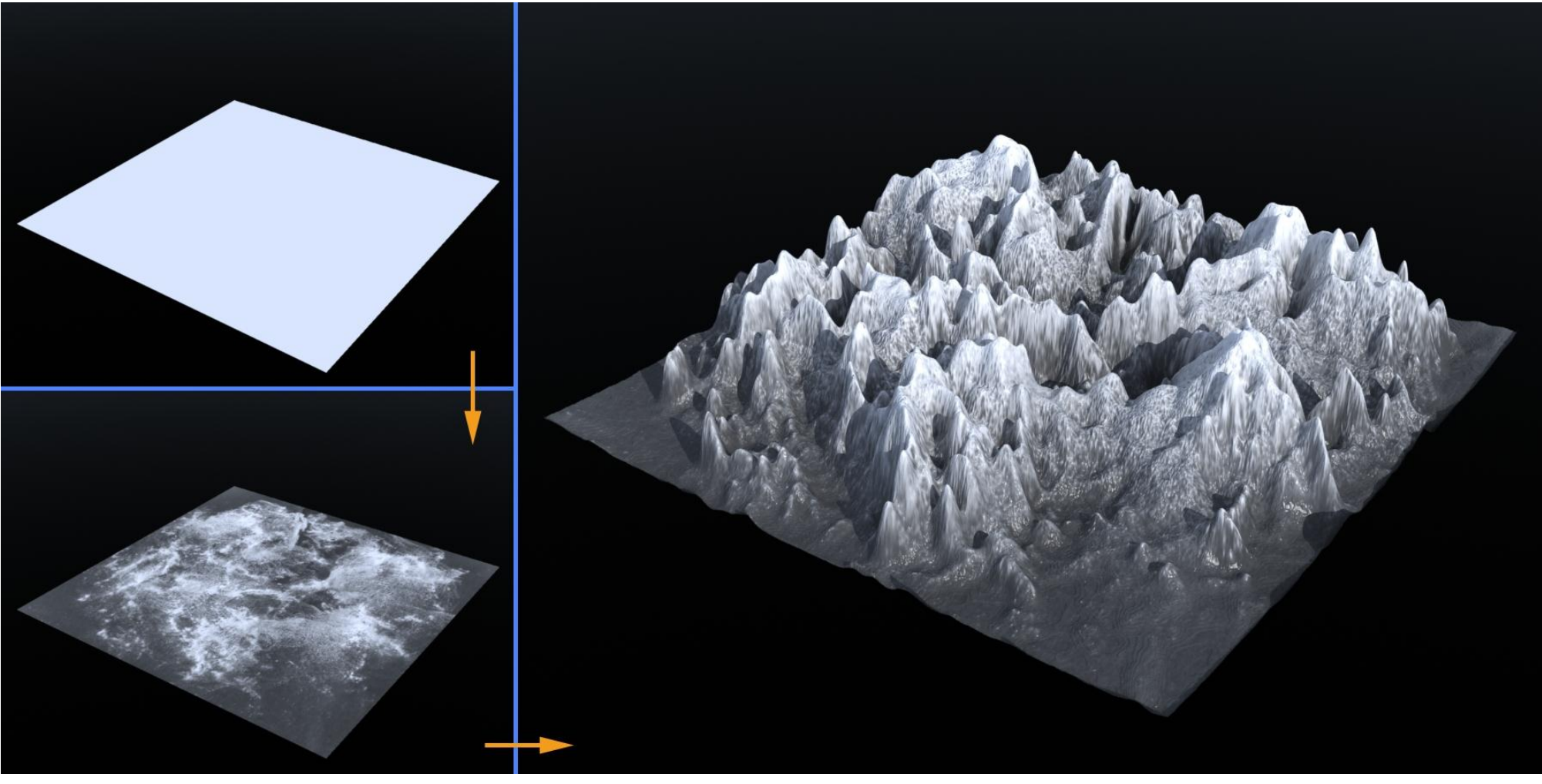
Cool: Displacement Mapping

- 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





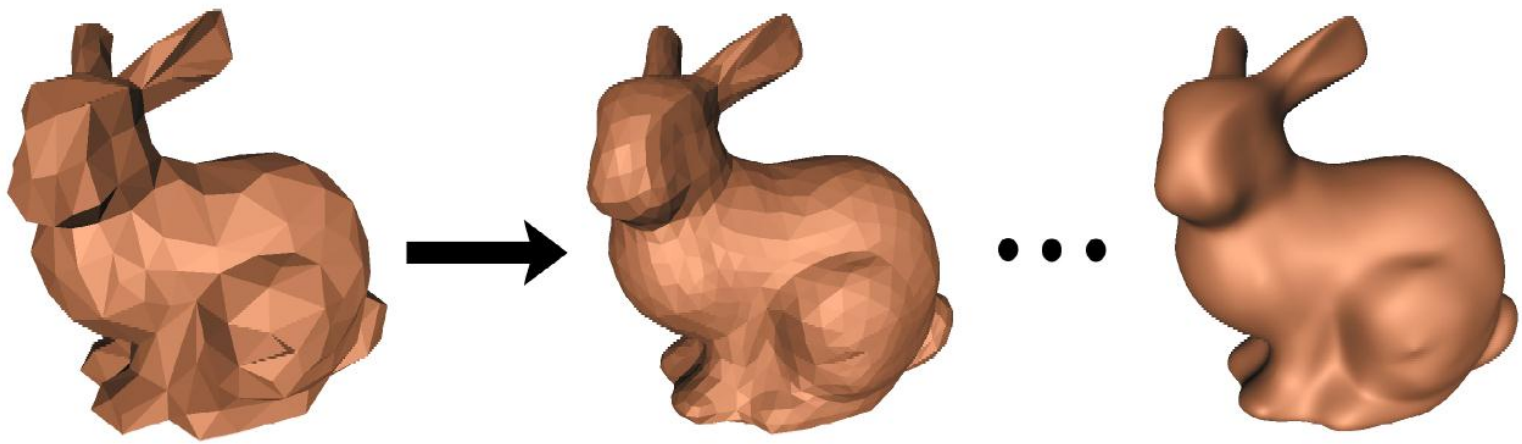
Displacement Mapping Example



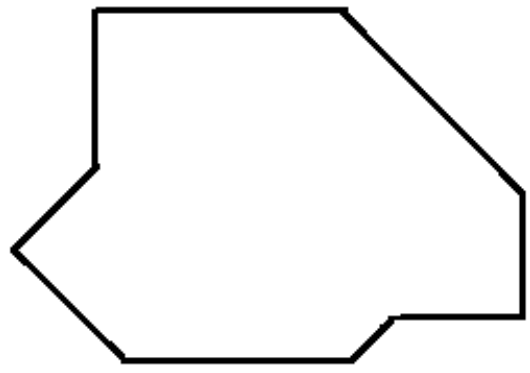
Questions?

Subdivision Surfaces

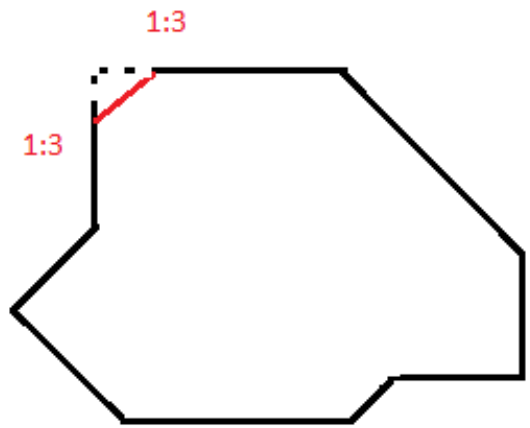
- Start with polygonal mesh
- Subdivide into larger number of polygons, smooth result after each subdivision
  - Lots of ways to do this.
- The limit surface is smooth!



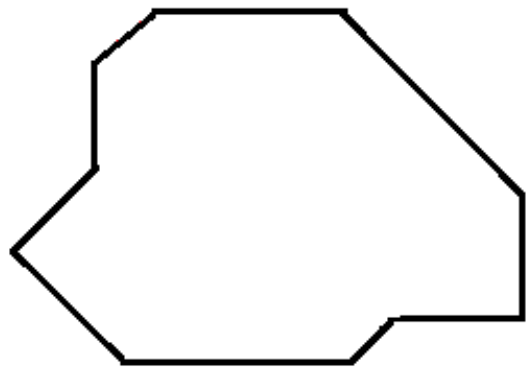
Corner Cutting



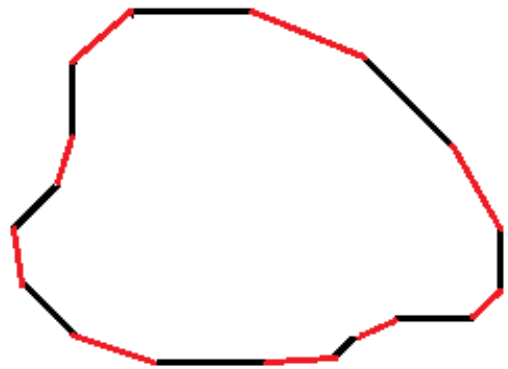
Corner Cutting



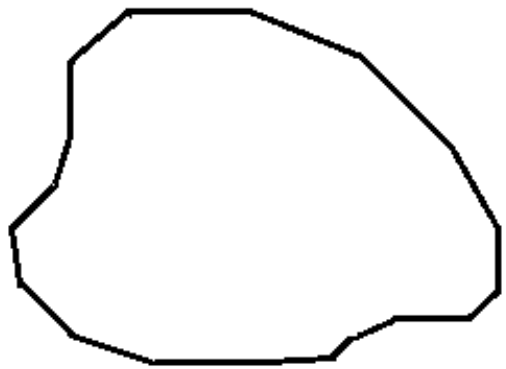
Corner Cutting



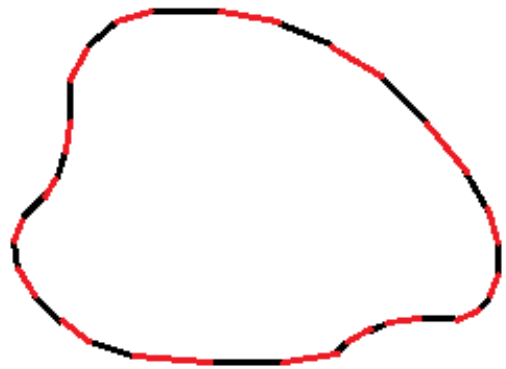
Corner Cutting



Corner Cutting



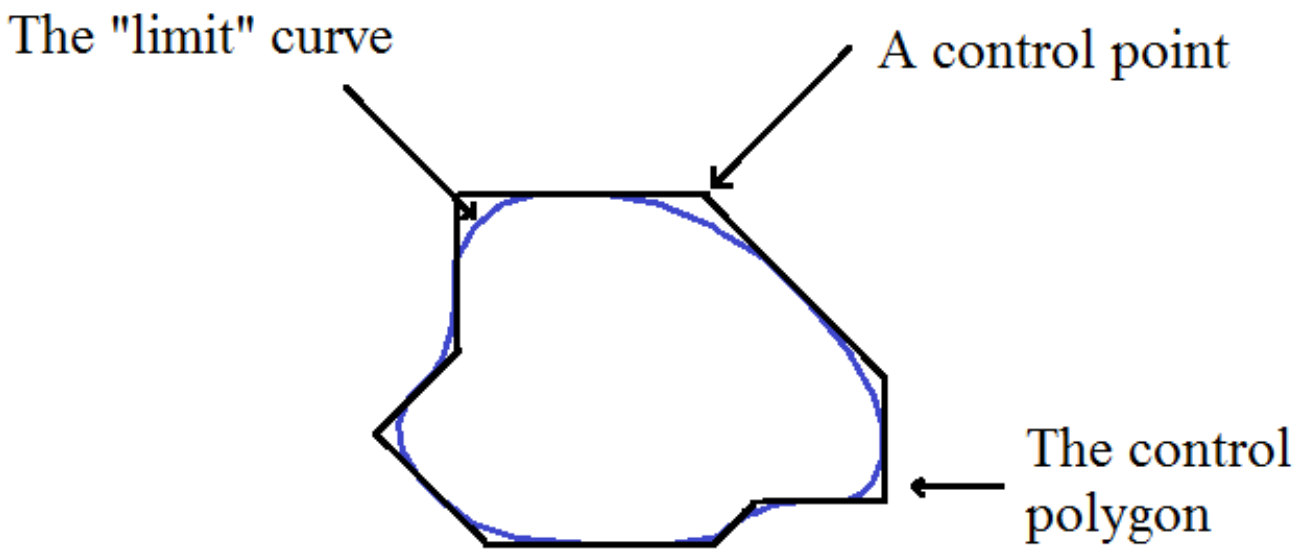
Corner Cutting



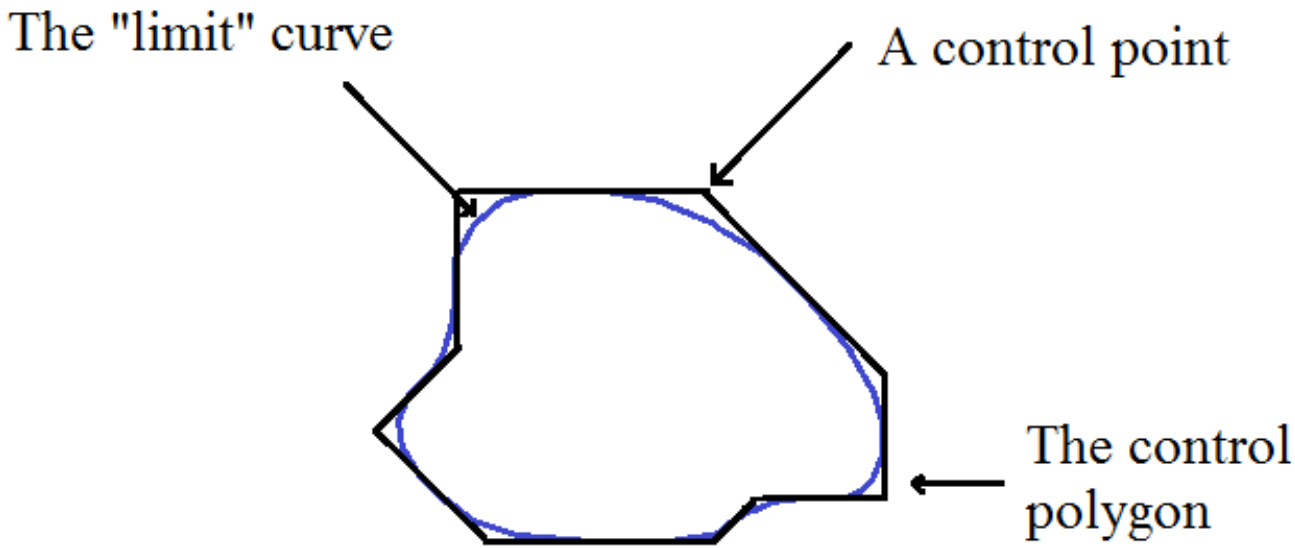
Corner Cutting

Corner Cutting

Corner Cutting

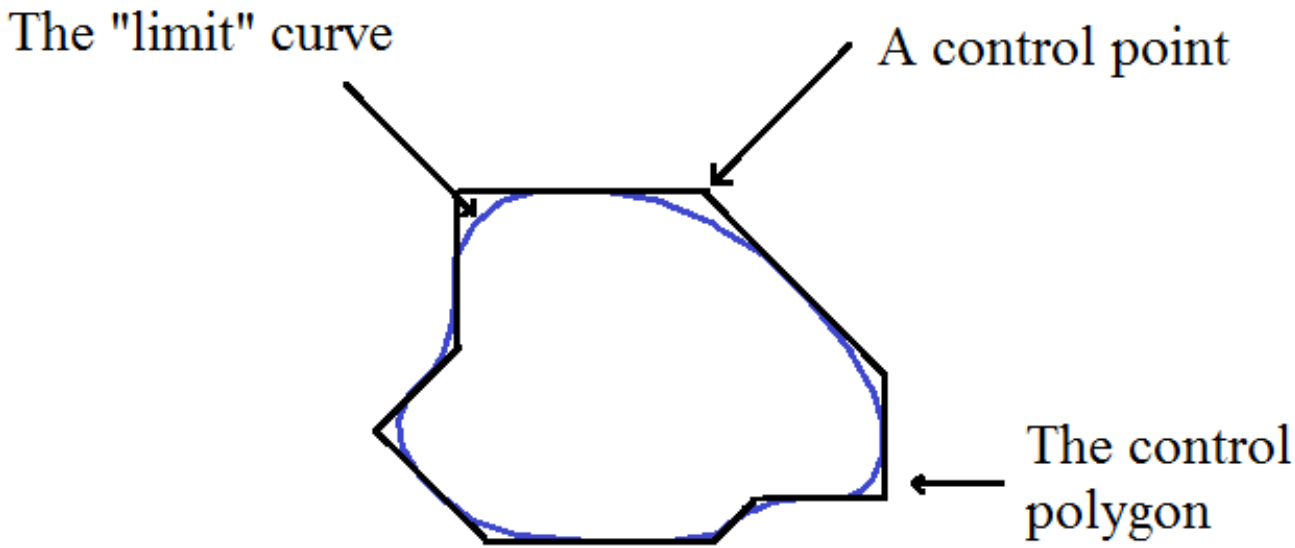


Corner Cutting



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

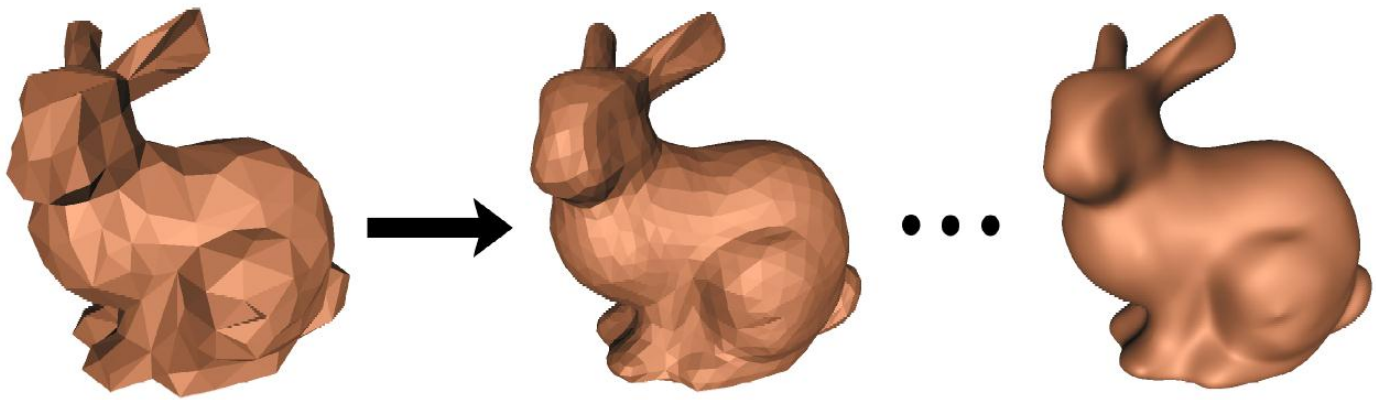
Corner Cutting



(Well, not totally unexpected, remember de Casteljau)

Subdivision Curves and Surfaces

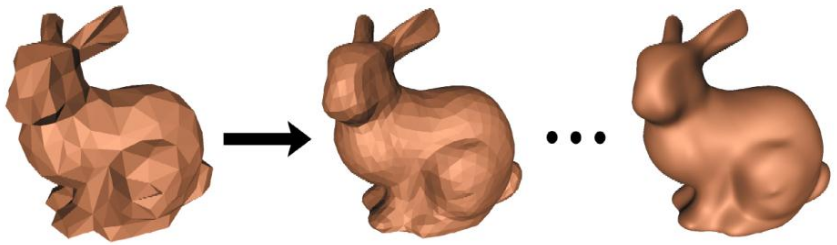
- Idea: cut corners to smooth
- Add points and compute weighted average of neighbors
- Same for surfaces
  - Special case for irregular vertices
    - vertex with more or less than 6 neighbors in a triangle mesh



Warren et al.

Subdivision Curves and Surfaces

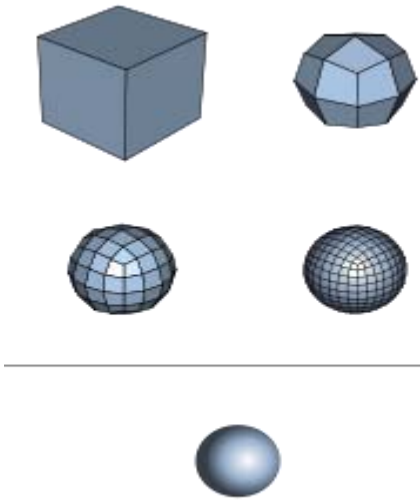
- Advantages
  - Arbitrary topology
  - Smooth at boundaries
  - Level of detail, scalable
  - Simple representation
  - Numerical stability, well-behaved meshes
  - Code simplicity
- Little disadvantage:
  - Procedural definition
  - Not parametric
  - Tricky at special vertices



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/



Subdivision + Displacement

Original rough mesh

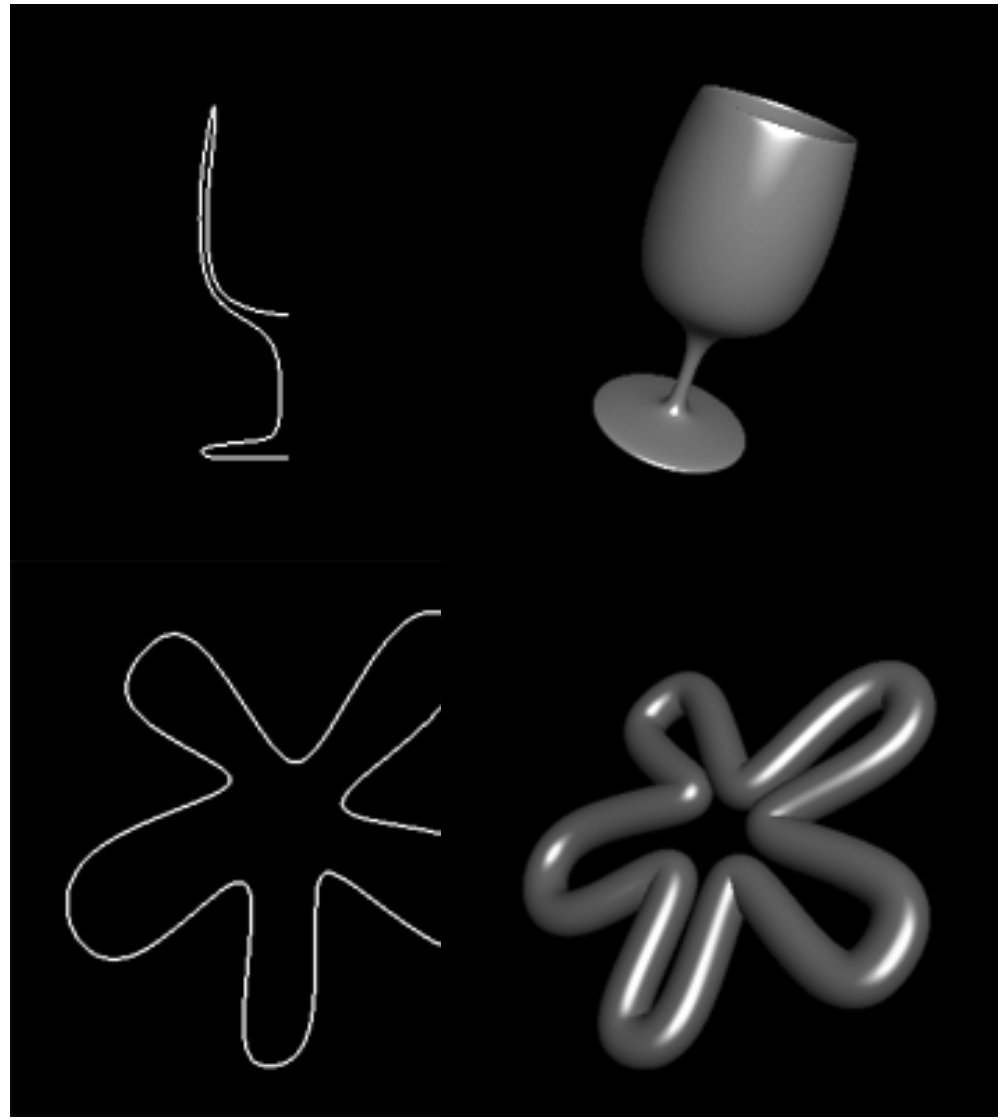
Original mesh with subdivision

Original mesh with subdivision and displacement

Questions?

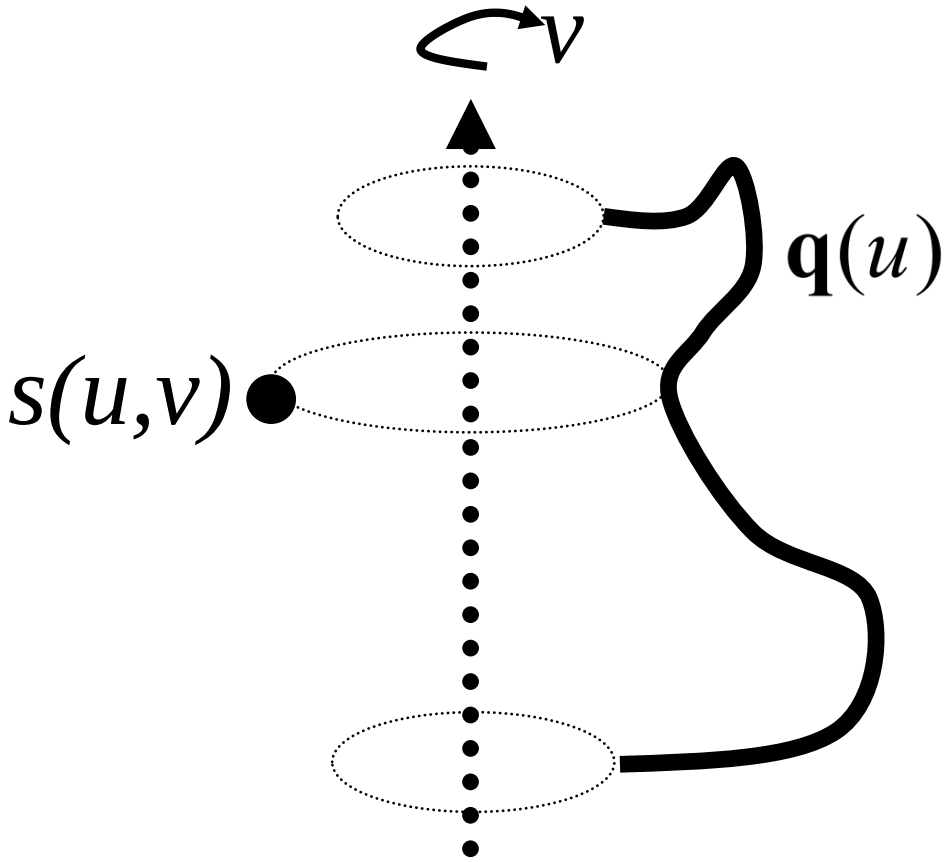
Specialized Procedural Definitions

- 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

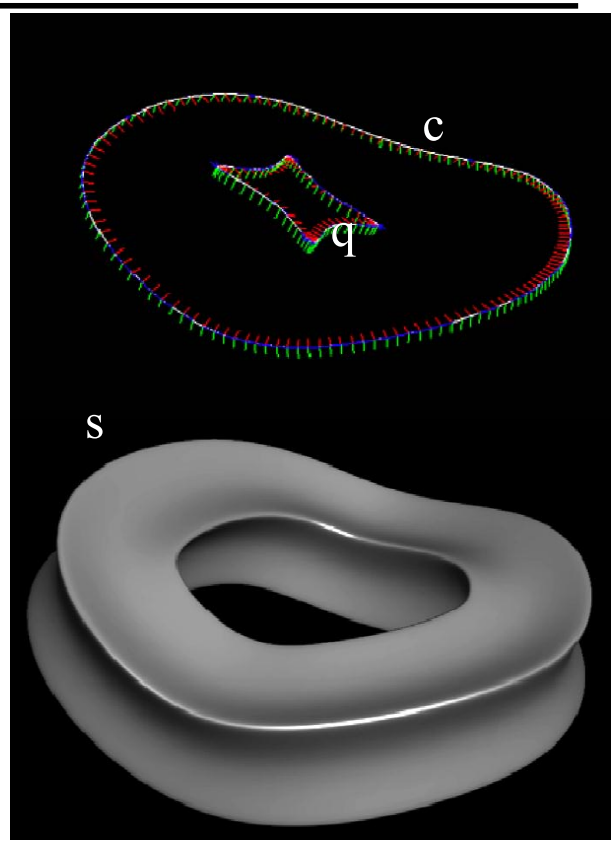
- 2D curve q(u) provides one dimension
  - Note: works also with 3D curve
- Rotation R(v) provides 2nd dimension



$\boldsymbol{s}(u,v)=\boldsymbol{R}(v)\boldsymbol{q}(u)$
where $\boldsymbol{R}$ is a matrix,
$\boldsymbol{q}$ a vector,
and $\boldsymbol{s}$ is a point on the surface

General Swept Surfaces

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

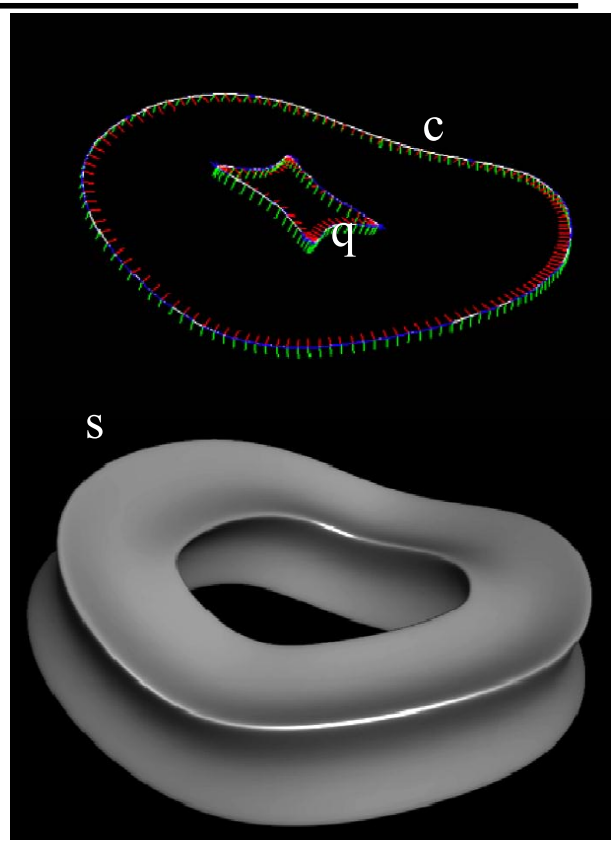


$$\boldsymbol{s}(u,v)=\boldsymbol{M}(\boldsymbol{c}(v))\boldsymbol{q}(u)$$

where $\boldsymbol{M}$ is a matrix that depends on the trajectory $\boldsymbol{c}$

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

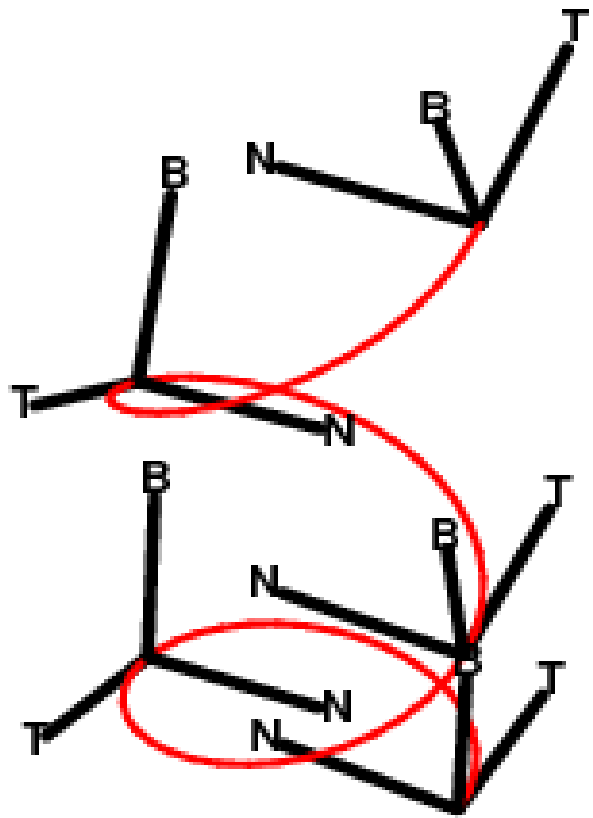


$$\boldsymbol{s}(u,v)=\boldsymbol{M}(\boldsymbol{c}(v))\boldsymbol{q}(u)$$

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

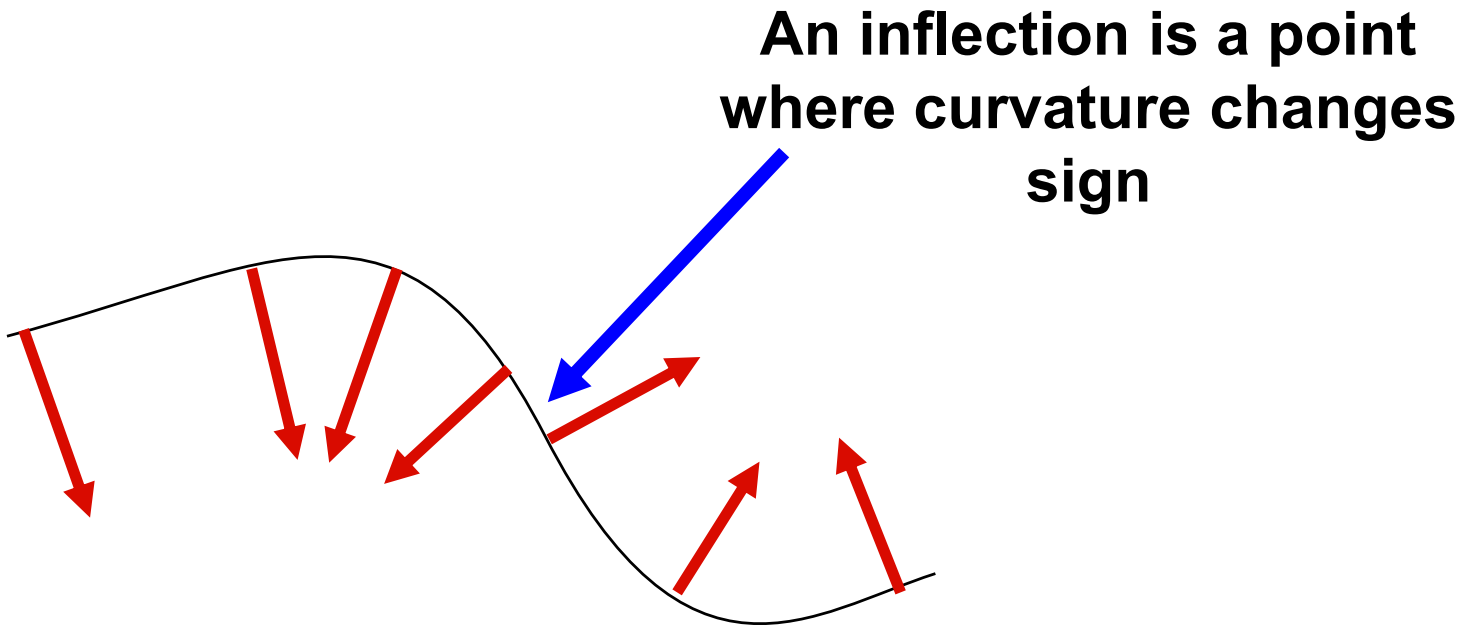
Frames on Curves: Frenet Frame

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

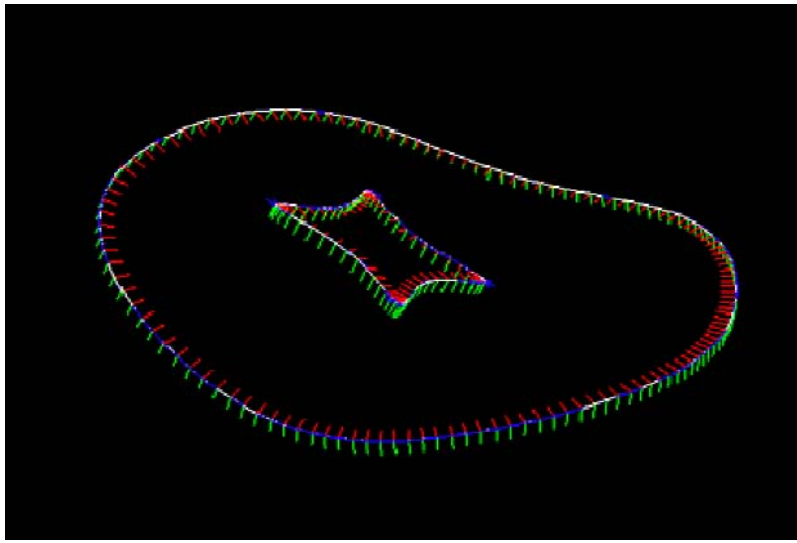


Frenet: Problem at Inflection!

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



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!**

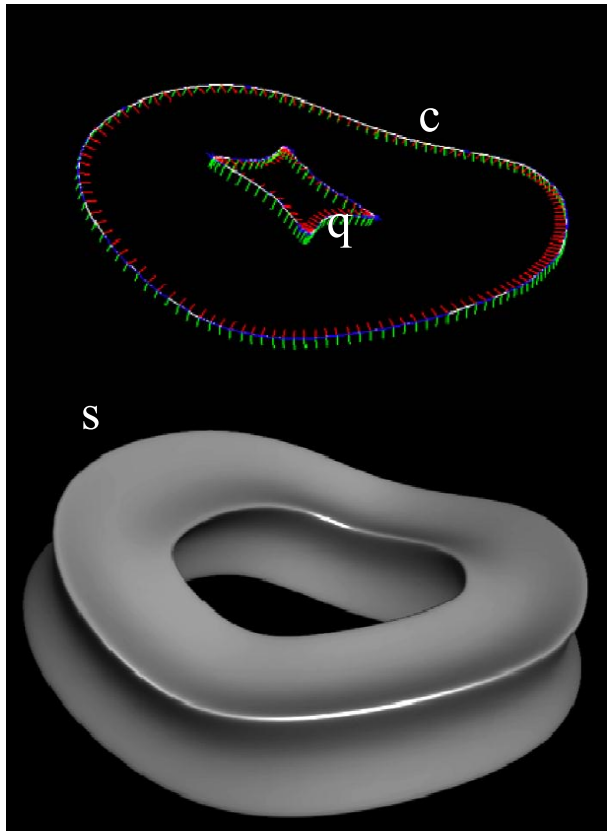
Normals for Swept Surfaces

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

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

  – *Remember to normalize!*

- One given by tangent of profile curve, the other by tangent of trajectory



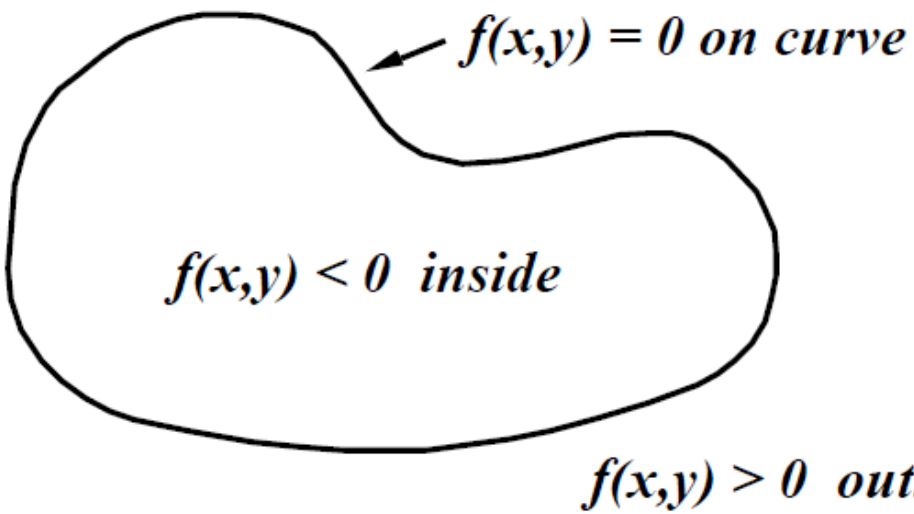
$$\boldsymbol{s}(u,v)=\boldsymbol{M}(\boldsymbol{c}(v))\boldsymbol{q}(u)$$

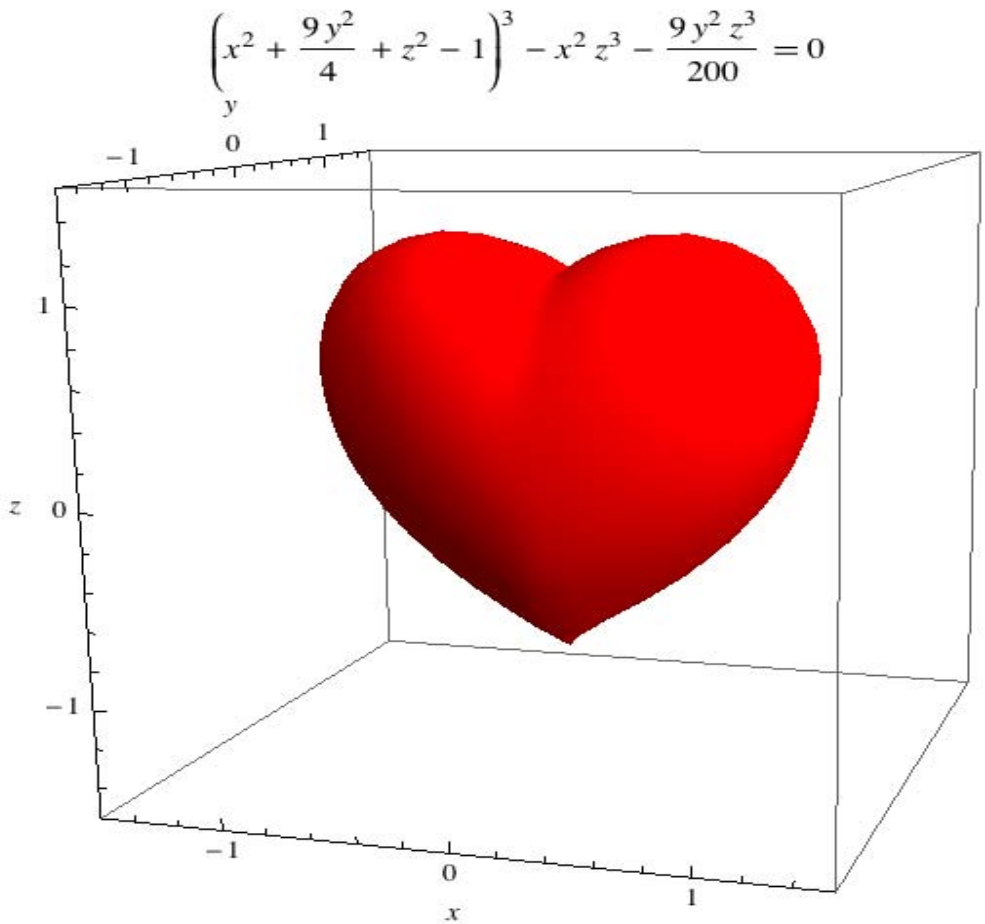
where $\boldsymbol{M}$ is a matrix that depends on the trajectory $\boldsymbol{c}$

Implicit Surfaces

- Surface defined implicitly by a function

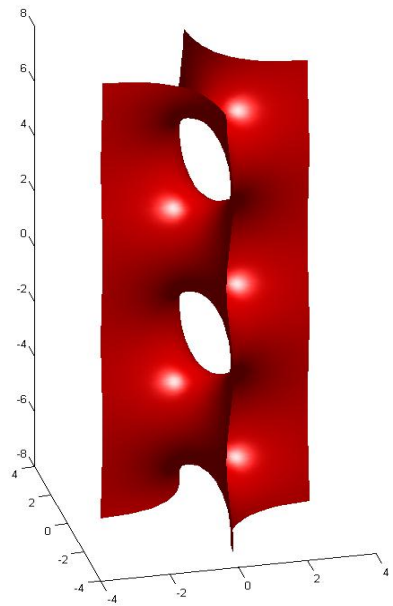
f (x, y, z) = 0 (on surface)
f (x, y, z) < 0 (inside)
f (x, y, z) > 0 (outside)





Implicit Surfaces

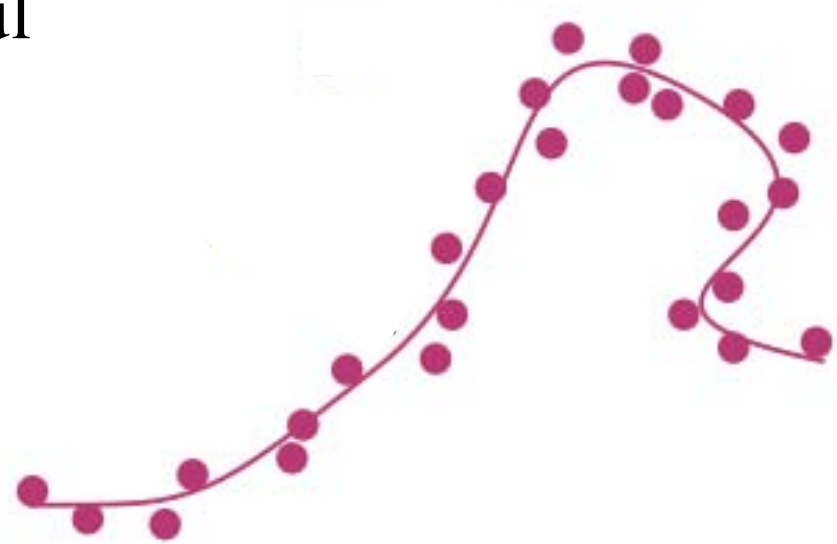
- Pros:
  - Efficient check whether point is inside
  - Efficient Boolean operations
  - Can handle weird topology for animation
  - Easy to do sketchy modeling
- Cons:
  - Does not allow us to easily generate a point on the surface



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

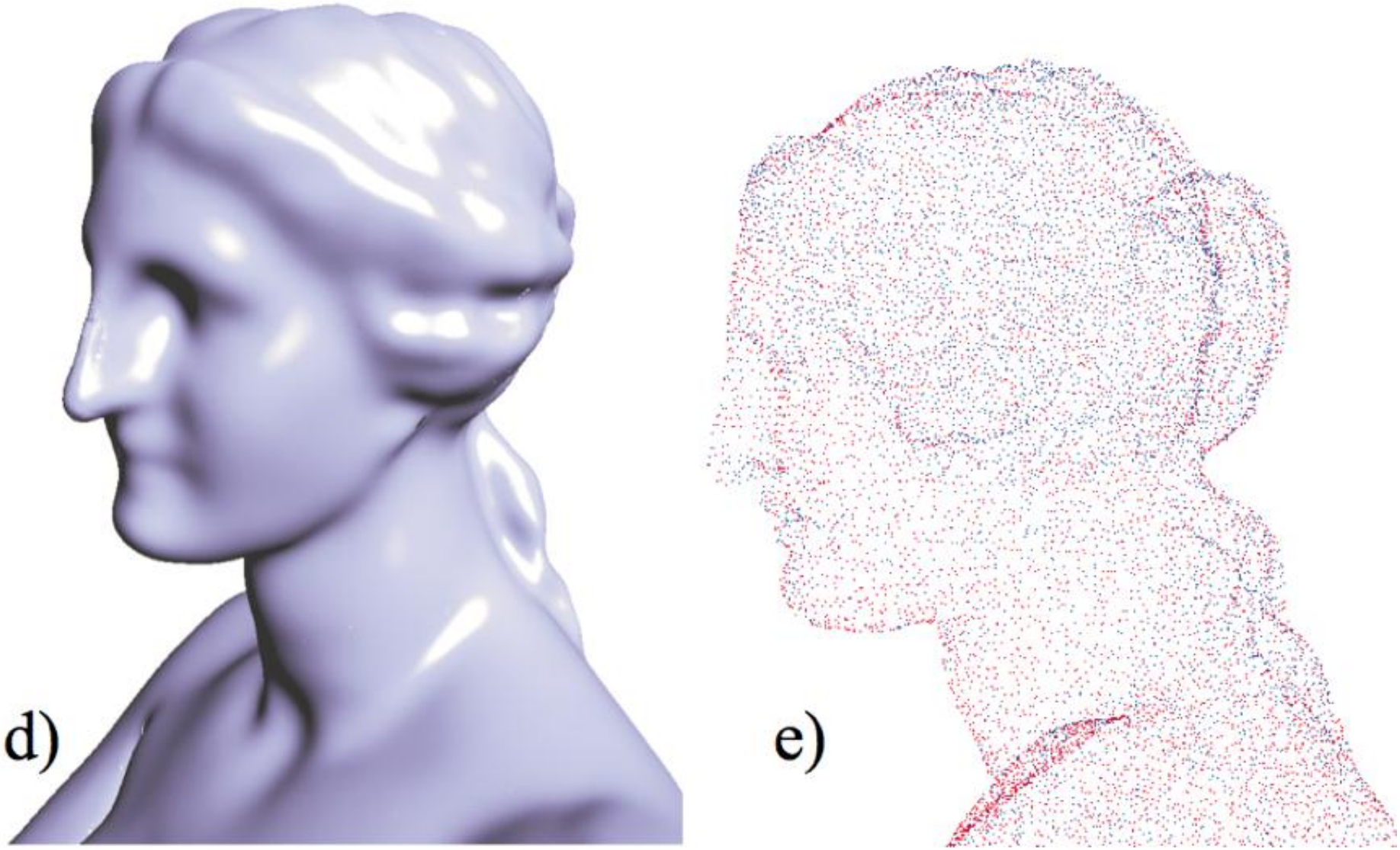


From Point Set Surfaces, (Alexa et al. 2001).

Point Set Surfaces

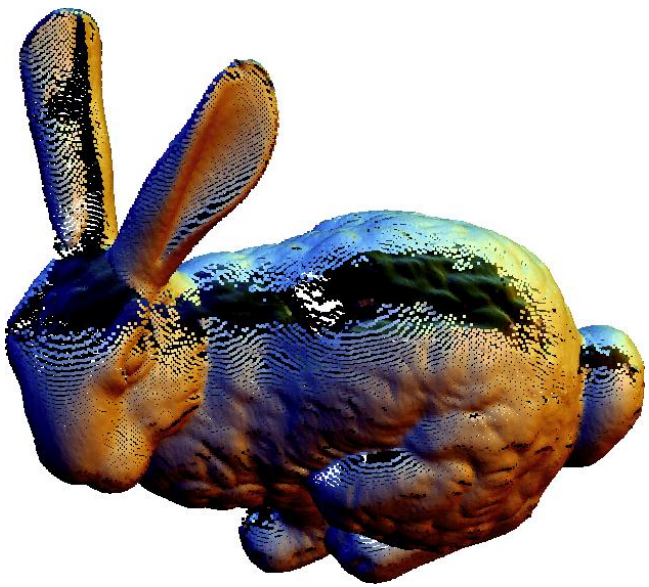
From Point Set Surfaces, used with permission from ACM, Inc

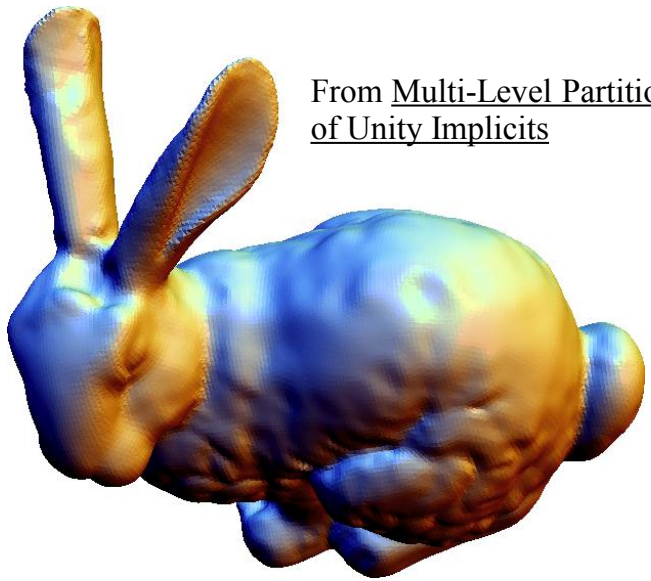
Alexa et al. 2001



Point Set Surfaces

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





From Multi-Level Partition of Unity Implicits

Ohtake et al. 2003

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

Questions?

|| (“pause”)

Ok, OMG, what? Further questions:

- Tensor products? Do I need to be a theoretical physicist like Einstein to do computer graphics?
- Well. . . not really, although it would help :)
- In the long run, **the more math you can fit** in your personal study plan, **the better you will become in computing**, including graphics programming and many other wonderful things that “the guy next door” can’t do.
- On this course, as you saw, we did a fast-forward.

Fast-forward ends here. We’ll come back to a first course in graphics.

This **story continues in the practical Assignment 1 handout**.

Play with this: http://nurbscalculator.in/

More math details (if you are interested): `http://www.cs.mtu.edu/~shene/COURSES/cs3621/NOTES/`

Form **groups**, **ask others** for help. **Help** your coursemates – you'll learn more while helping others. Ultimately **think and code by yourself** – otherwise learning is unlikely to happen.

TIEA311

Note:

- Surfaces in Assignment 1 can be done by applying a suitable affine transform (4x4 matrix) to each vertex and normal of the profile curve.
- (Not the whole story! Especially about the normal vector, but quite enough for Assignment 1...)
- So you should go and complete it right about now.

TIEA311 - Today in Jyväskylä

The time allotted for this week's graphics lectures is now over.

Next lecture happens in 6 days and 4 hours.

The teacher will now tell his view about what **could be useful activities** for you during that time period.

$\rightarrow$ see lecture video.

Make notes, if you have to.

Even if he forgets to say it, **remember to rest, too**!