MSc Computer Games and Entertainment

Maths & Graphics Unit 2011/12

Lecturer: Gareth Edwards

Subdivision Of Triangular Terrain Mesh

Breckon, Chenney, Hobbs, Hoppe, Watts

Parametric Curves & Surfaces

Based on lecture notes by Greg Humphreys, UoV

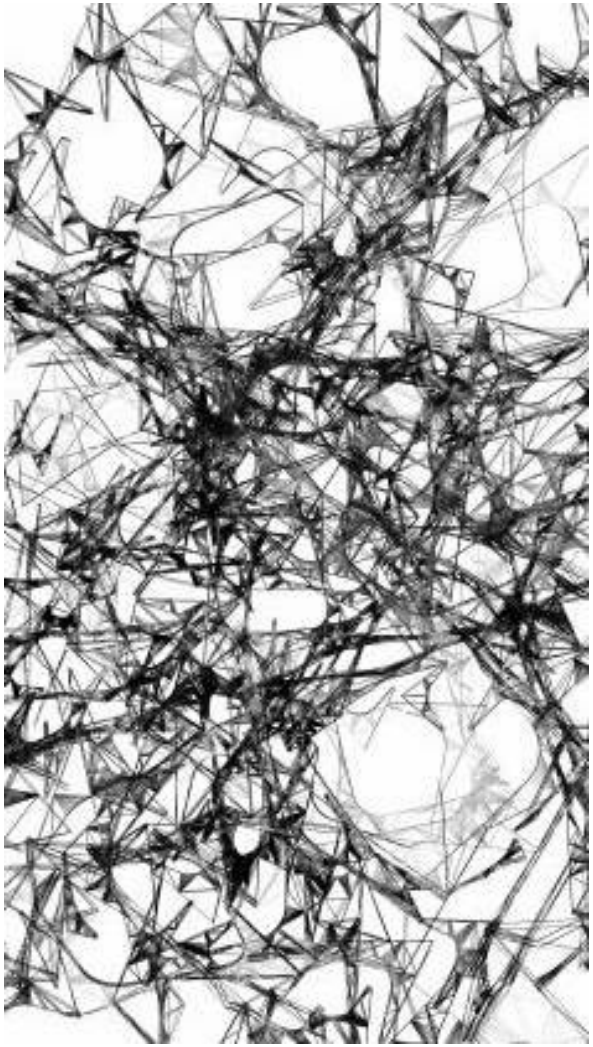
Curved Surfaces

- **Motivation**
  - Exact boundary representation for some objects
  - More concise representation than polygonal mesh

CONTEXT

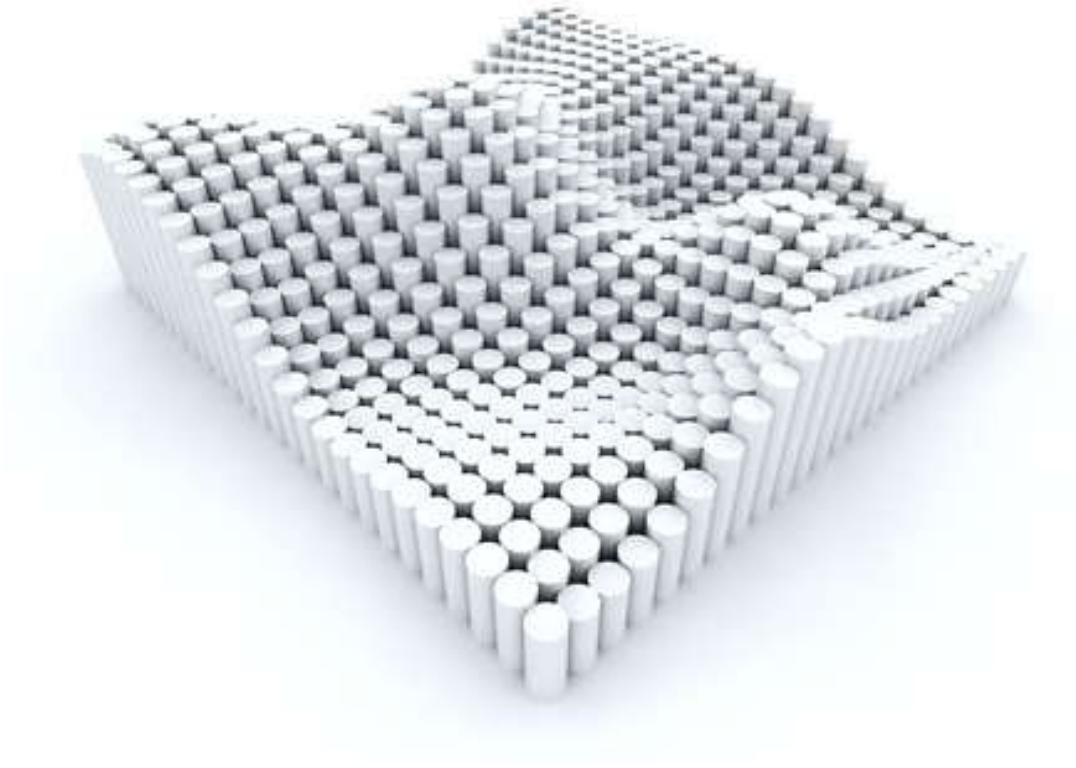
Parametric Surface

- **Archimorph on facebook**
  - an image from a parametric model which tracks solar position to maximize algal bloom cultivation within a module. at this stage of the design, the modules are populated across the entire surface, but the next iteration will allow other external influences (information from program, context, and climate) to effect the surface and distribution of the module based on the program. other means of algae production and materiality are simultaneously being explored with aggregations



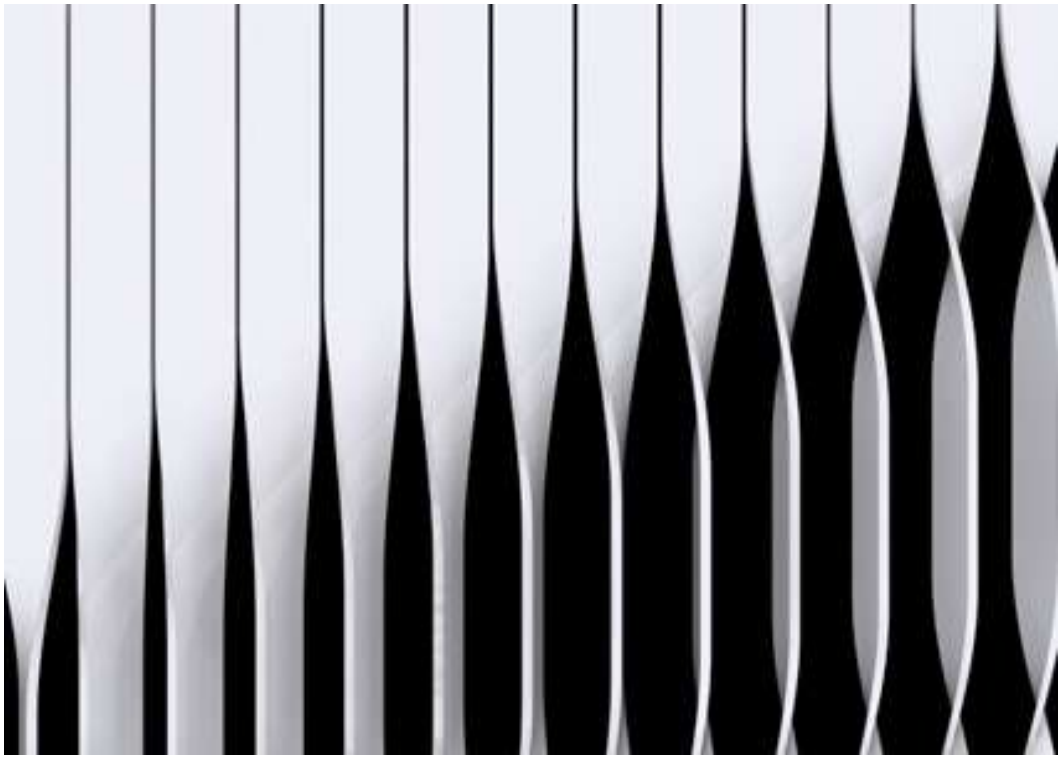
Controlling scalar properties in two dimensional array by a surface

- **3ds MAX Parametric Array**



Creating complex and simple data sets

- **3ds MAX Parametric Array**



The Torus

- **NATHAN MILLER** using Grasshopper
  - “Here is a strategy I am working on in Grasshopper for creating rationalized doubly curved surfaces. The example below uses a torus (defined parametrically using a GH function component). The use of a toroidal "slice" to drive the geometry allows for flat panels, repetitive panel sizes, and quadrilateral panel shapes.”

Nathan Miller

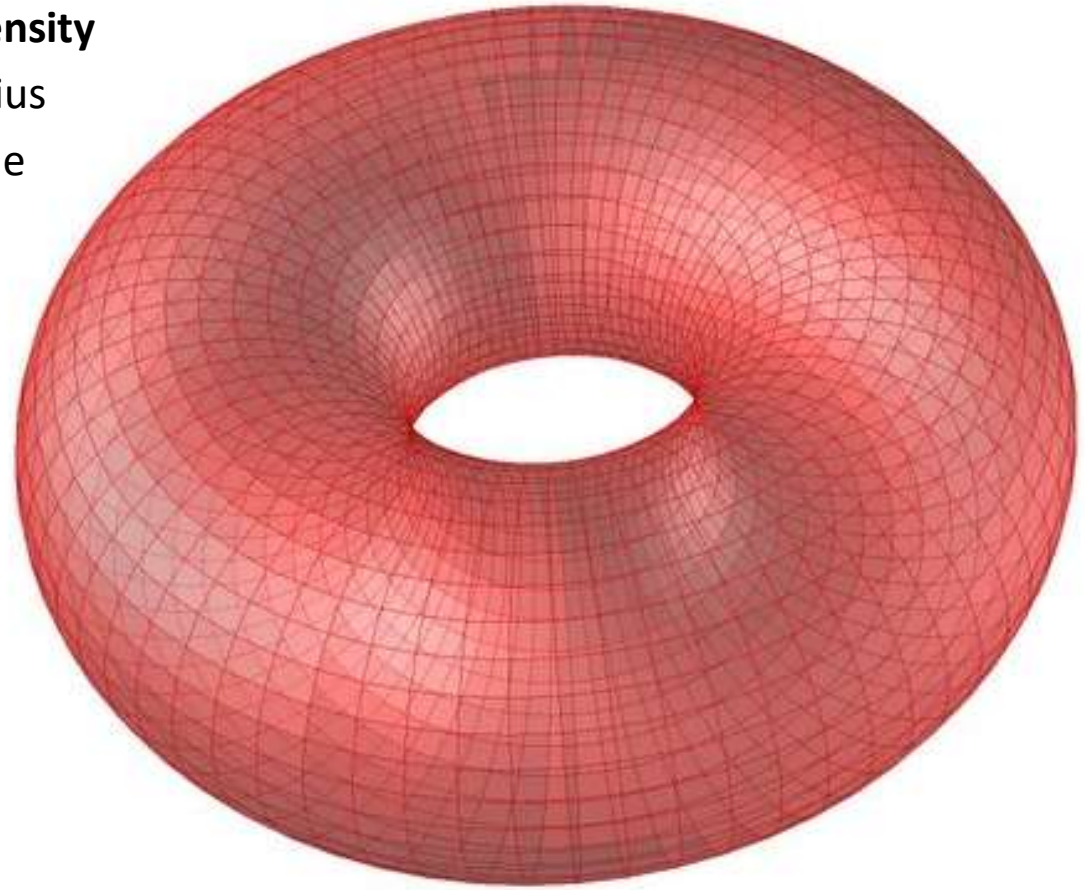
The Torus

- **Generators**
  - Radius
  - Circle

Nathan Miller

The Torus

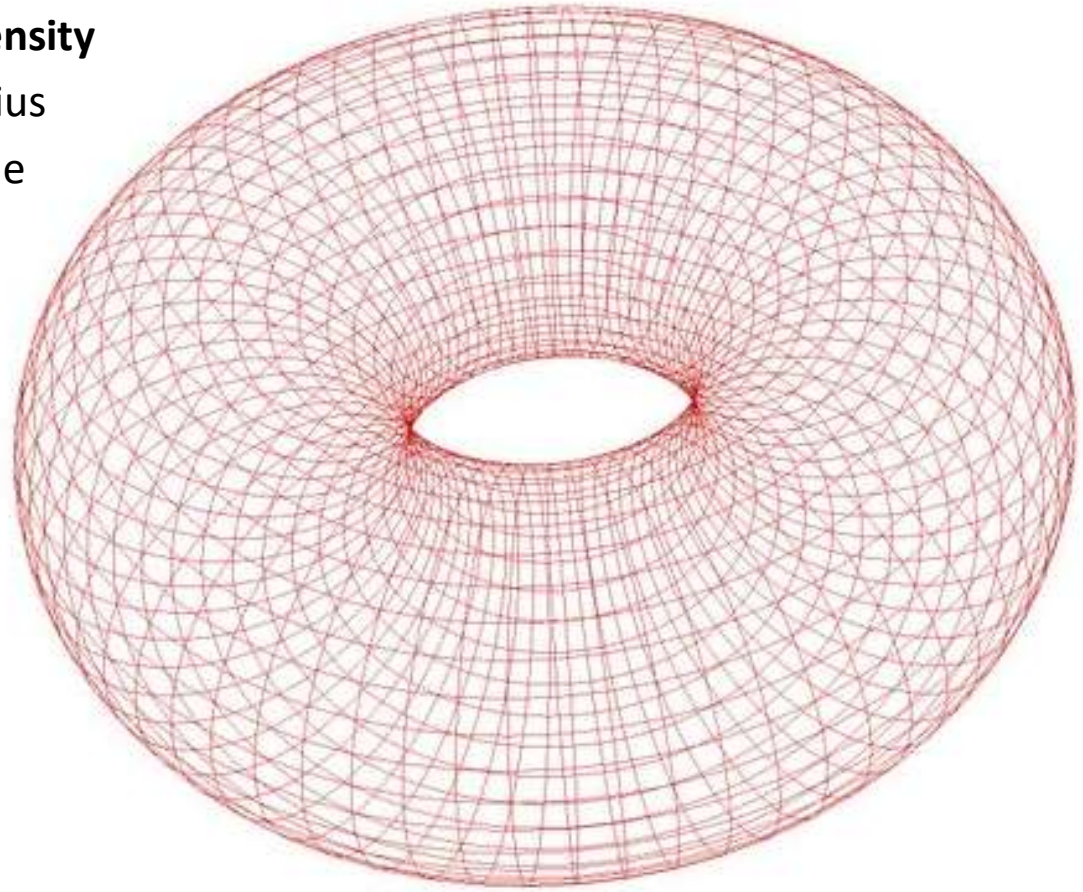
- **Mesh Density**
  - Radius
  - Circle



Nathan Miller

The Torus

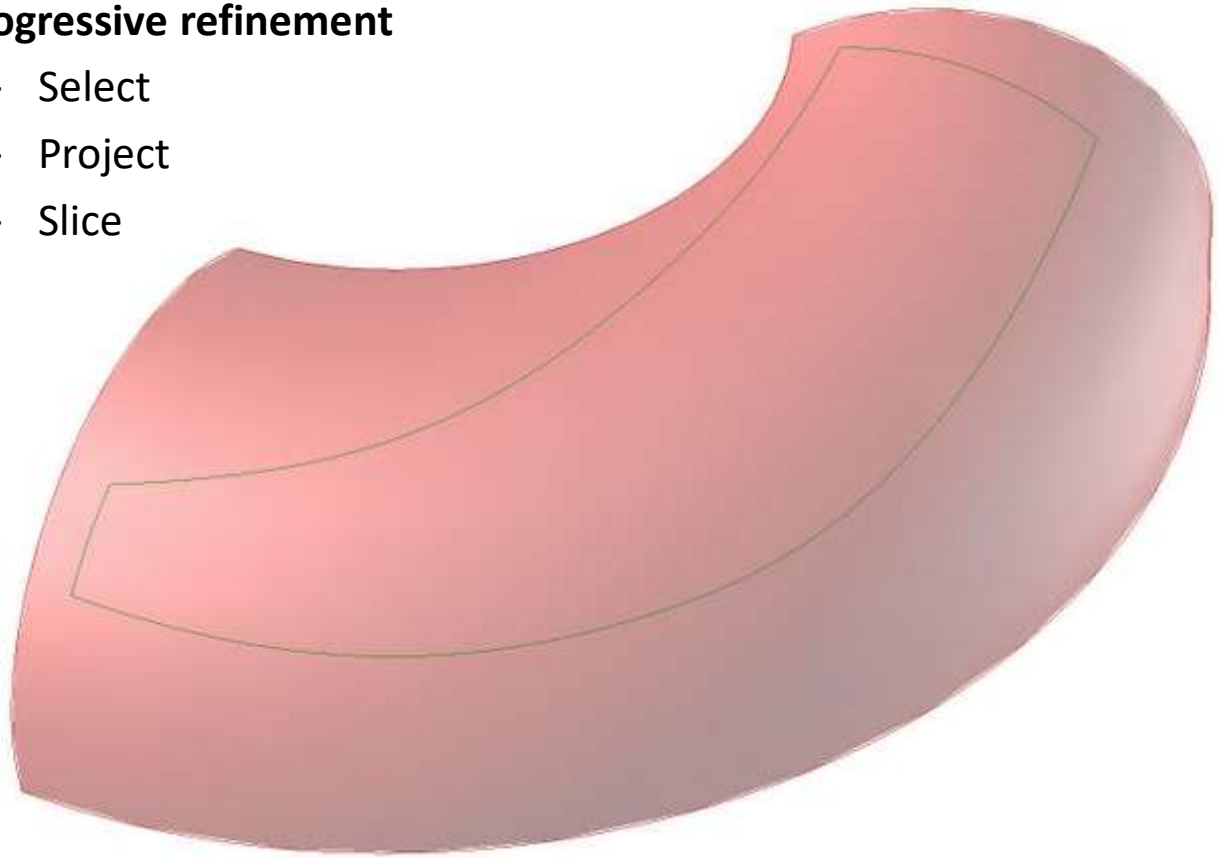
- **Mesh Density**
  - Radius
  - Circle



Nathan Miller

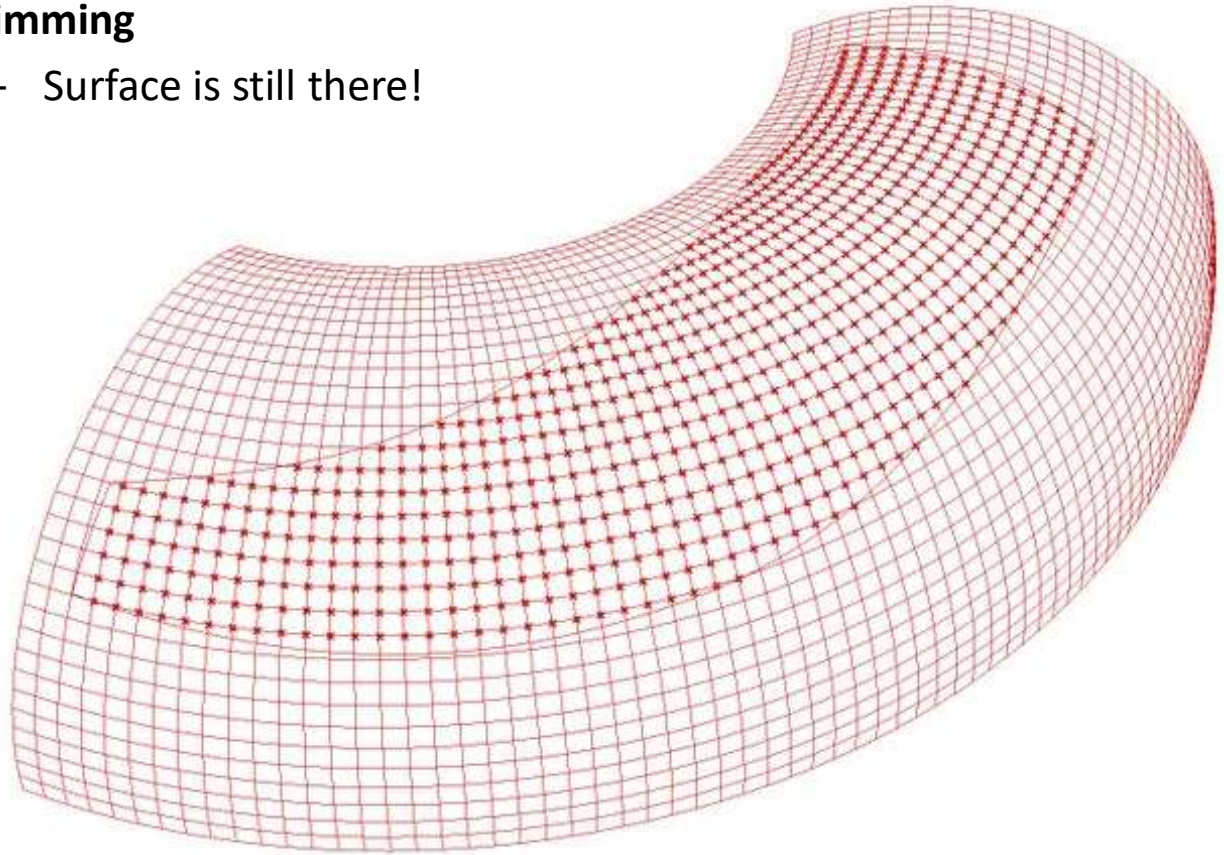
The Torus

- **Progressive refinement**
  - Select
  - Project
  - Slice



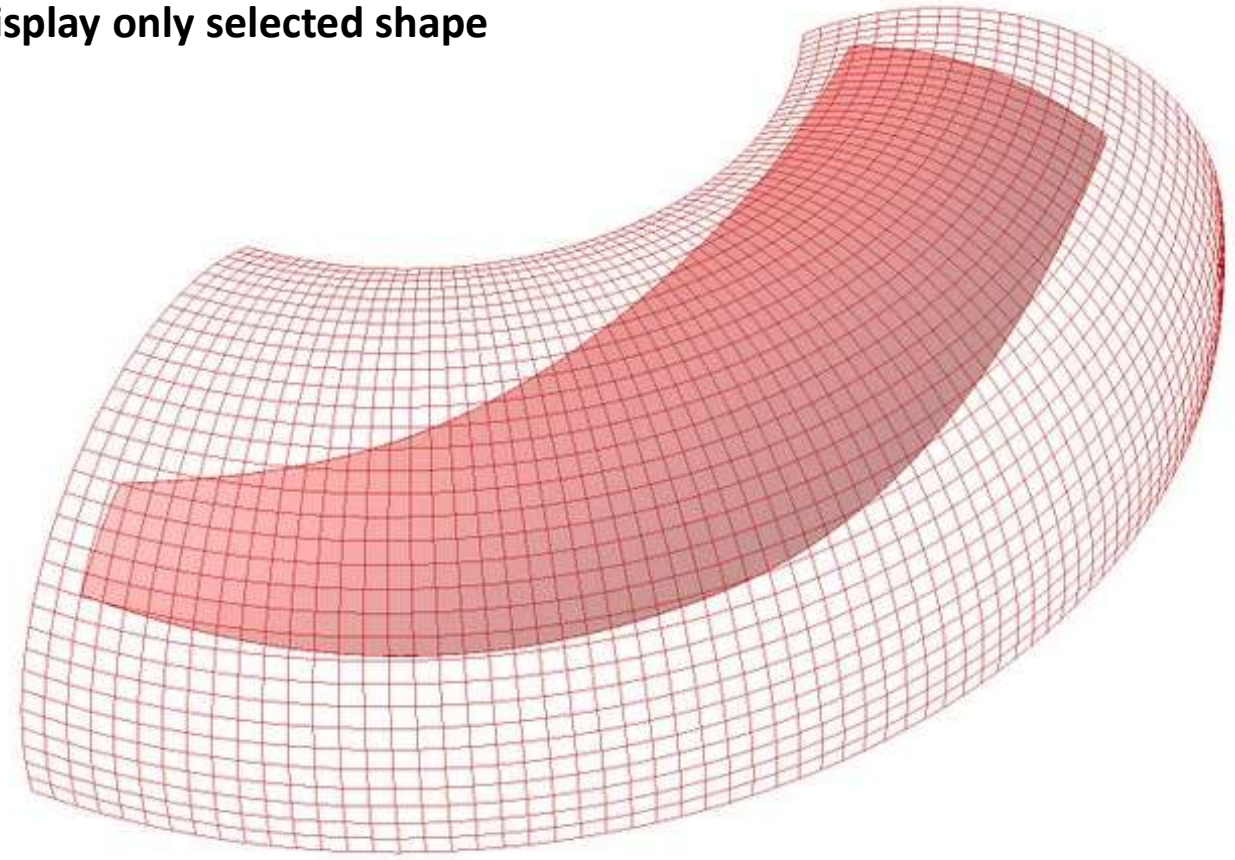
The Torus

- **Trimming**
  - Surface is still there!



The Torus

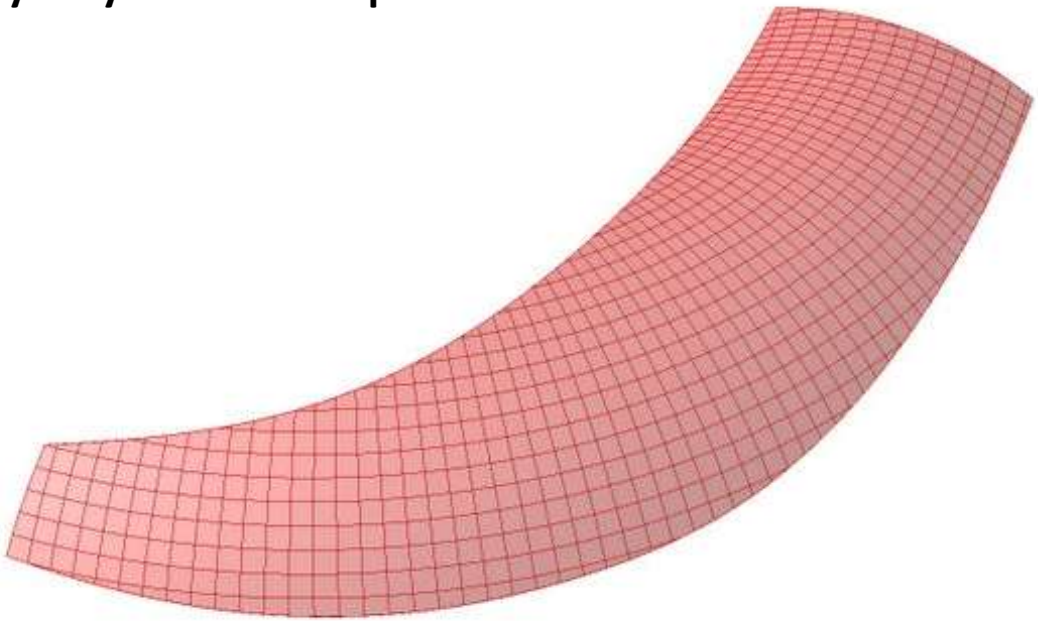
- **Display only selected shape**



Nathan Miller

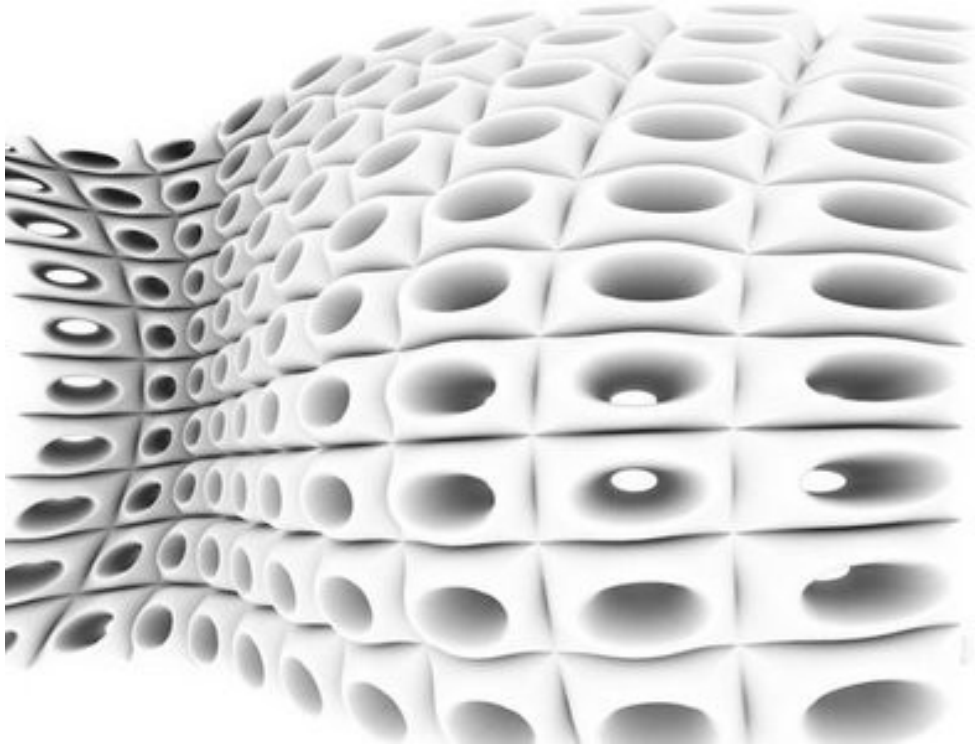
The Torus

- **Display only selected shape**



Propagating objects across a curved surface

- A surface which changed shape and size of an object in relation to a point object

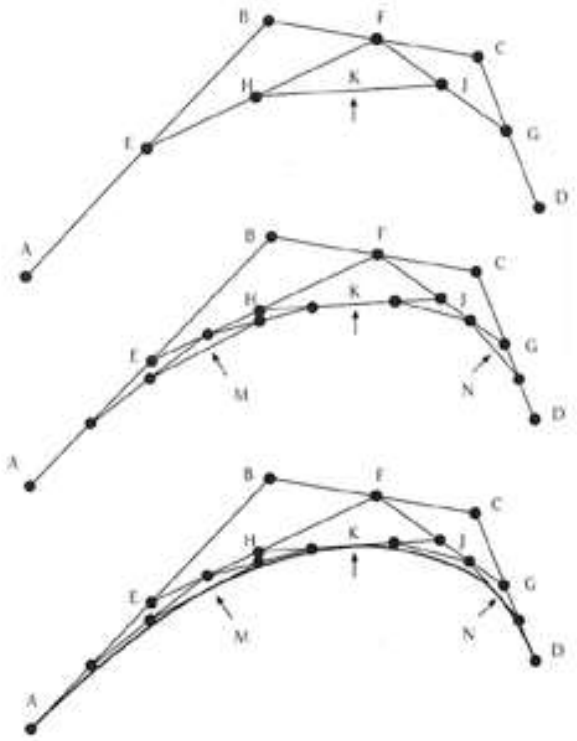


Nathan Miller

In Design

- **ROBIN CARPENTER**
  - “Trying to draw curves on screen, I had to learn about the bezier curve and it’ mathematical construction. I was amazed how simple it actually was”.
  - “And although we use it daily, we know so little about it. I wanted to make its beauty visible”.
  - “In honour of Pierre Etienne Bezier. And to his wife”.
  - “BEZIER - a silver pearl necklace”.

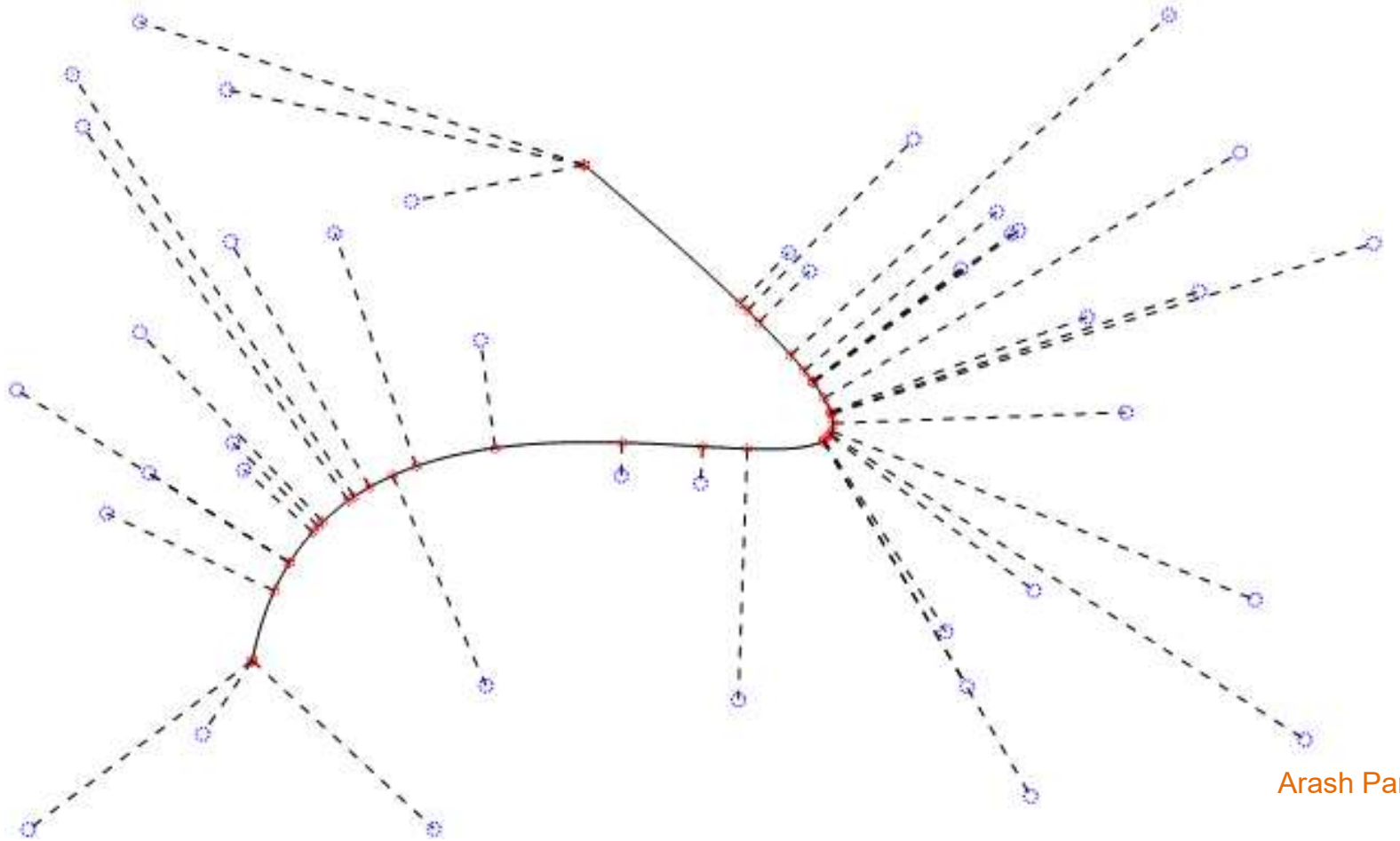
In Design



Robin Carpenter

Closest Point On Quadratic Bezier From External Points

- **Visual Intelligence**



Arash Partow

Closest Point On Quadratic Bezier From External Points

```
const std::size_t max_points = 50;
wykobi::quadratic_bezier<T,2> bezier;
bezier[0] = wykobi::generate_random_point<T>(width - 1.0, height - 1.0);
bezier[1] = wykobi::generate_random_point<T>(width - 1.0, height - 1.0);
bezier[2] = wykobi::generate_random_point<T>(width - 1.0, height - 1.0);
   std::vector<wykobi::point2d<T>> point_list;
point_list.reserve(max_points);
wykobi::generate_random_points(0.0,0.0,width - 1.0,height -
   1.0,max_points,std::back_inserter(point_list)); graphic.draw(bezier,100);

for(std::size_t i = 0; i < point_list.size(); ++i) {
   wykobi::point2d<T> closest_point =
         wykobi::closest_point_on_bezier_from_point(bezier,point_list[i]);
   if (wykobi::distance(closest_point,point_list[i]) > T(1.0)) {
         graphic.draw(wykobi::make_segment(closest_point,point_list[i]));
   }
   graphic.draw_circle(point_list[i],3);
   graphic.draw_circle(closest_point,2);
}
```

Arash Partow

Curved Surfaces

- **Motivation**
  - Exact boundary representation for some objects
  - More concise representation than polygonal mesh

Curved Surfaces

- **What makes a good surface representation?**
  - Accurate
  - Concise
  - Intuitive specification
  - Local support
  - Affine invariant
  - Arbitrary topology
  - Guaranteed continuity
  - Natural parameterization
  - Efficient display
  - Efficient intersections

Parametric Surfaces

- **Boundary defined by parametric functions:**

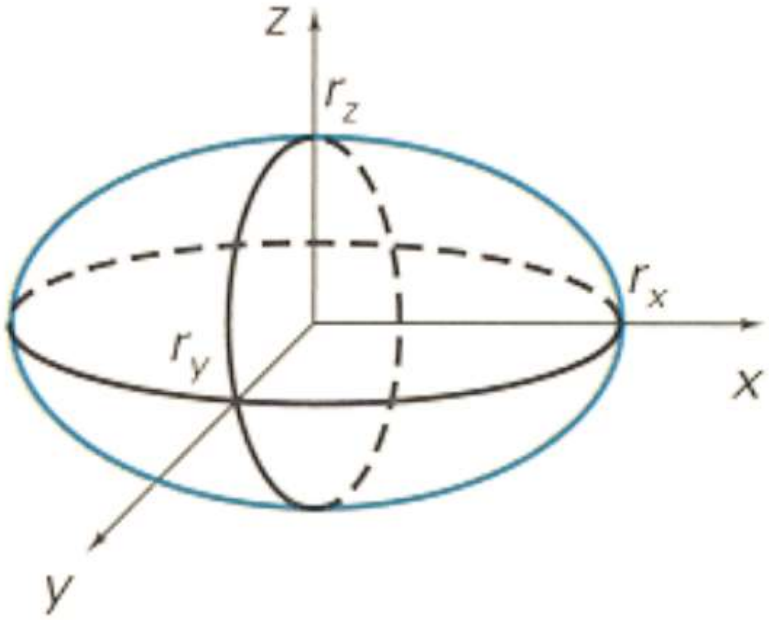
  x = fx(u,v)

  y = fy(u,v)

  z = fz(u,v)

- **Example: ellipsoid**

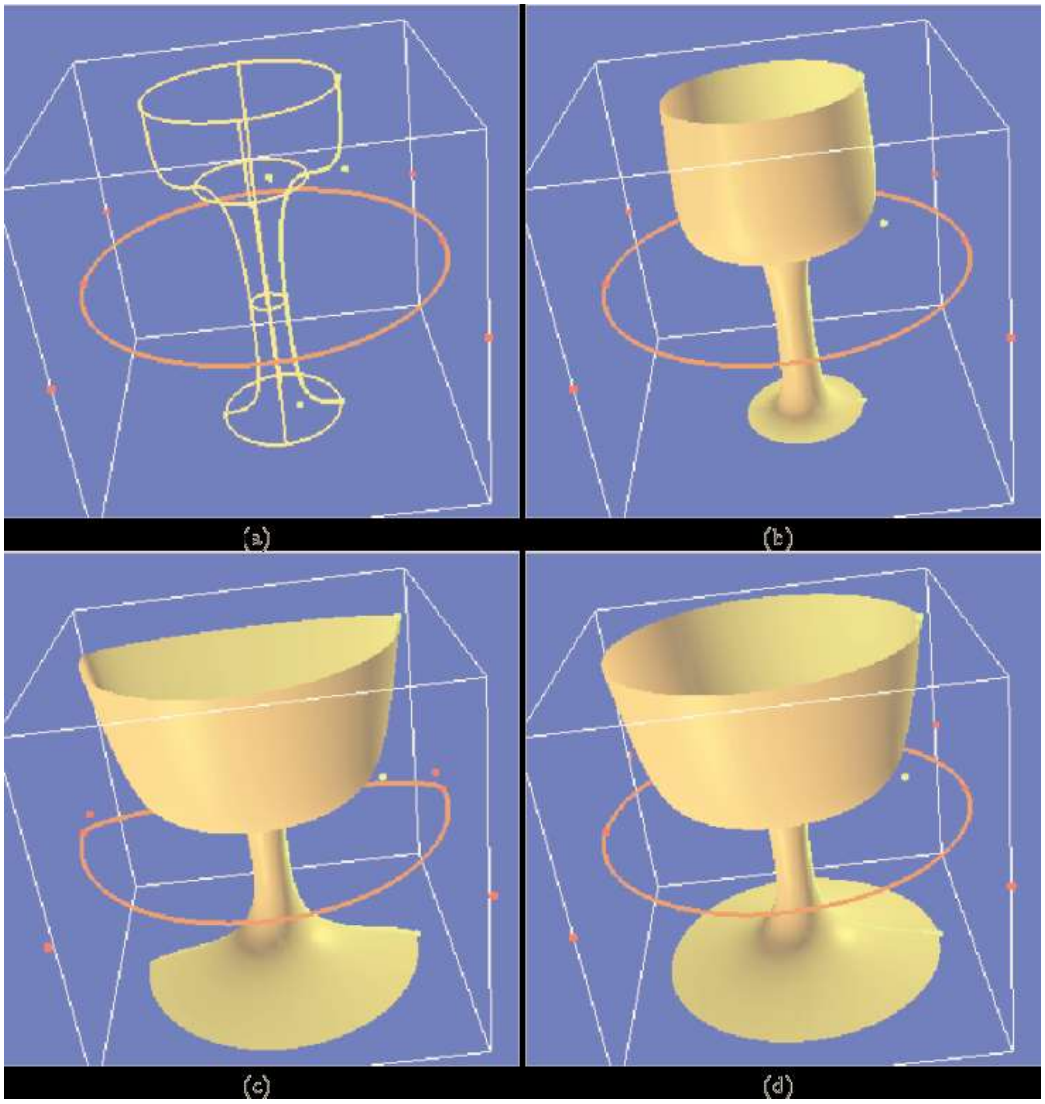
$$x = r_x \cos\phi \cos\theta$$
$$y = r_y \cos\phi \sin\theta$$
$$z = r_z \sin\phi$$



H&B Figure 10.10

Surface of revolution

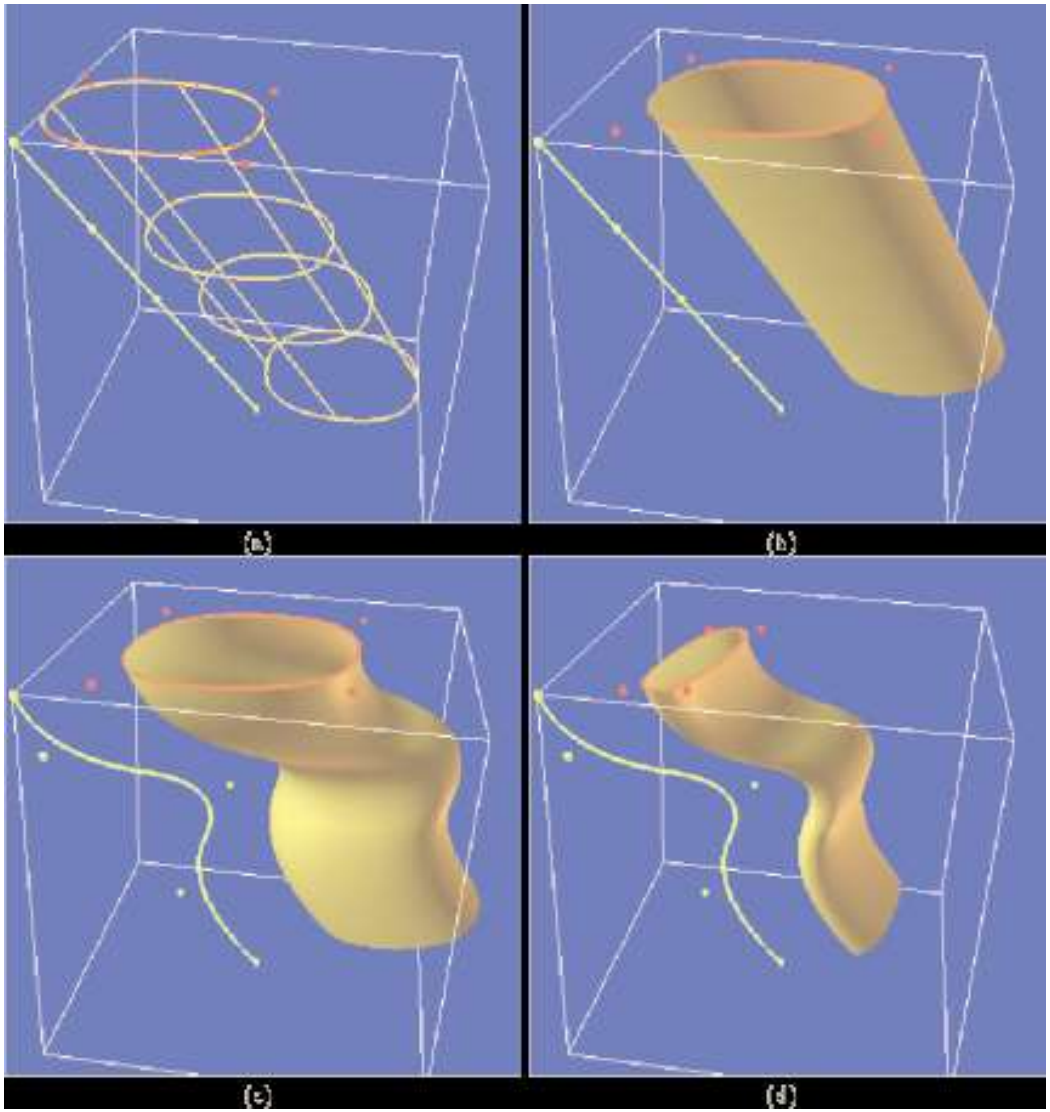
- **Idea: take a curve and rotate it about an axis**



Demetri Terzopoulos

Swept surface

- **Idea: sweep one curve along path of another curve**

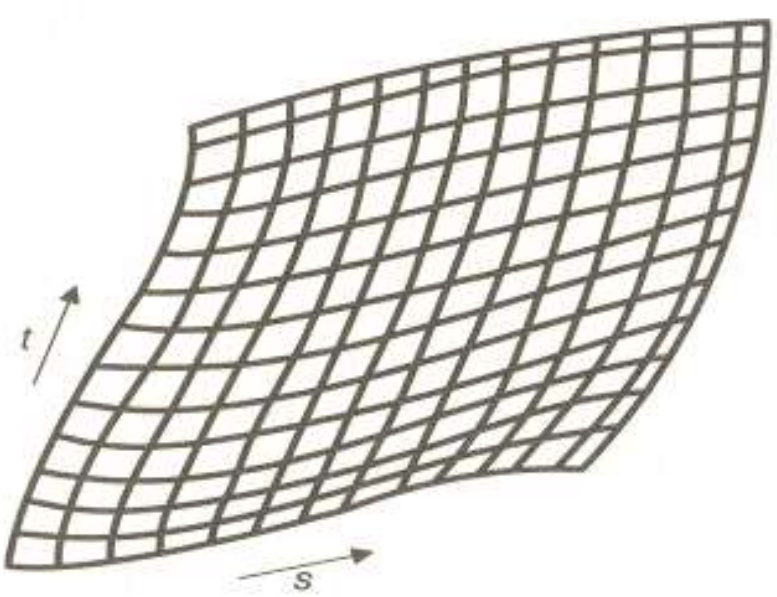


Demetri Terzopoulos

Parametric Surfaces

- **Advantage**:
  - easy to enumerate points on surface

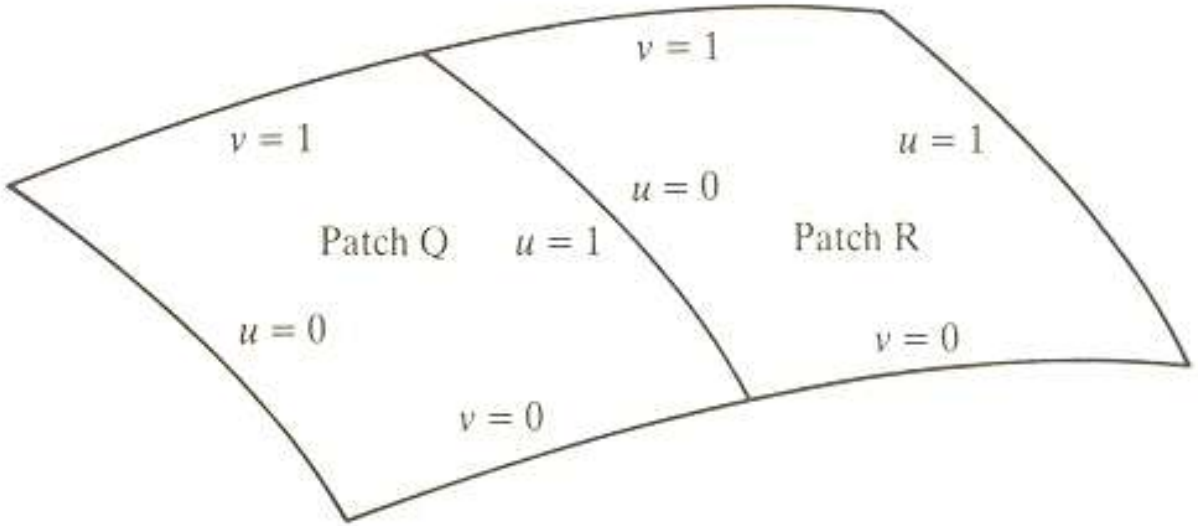
- **Disadvantage:**
  - need piecewise-parametric surfa describe complex shape



FvDFH Figure 11.42

Piecewise Parametric Surfaces

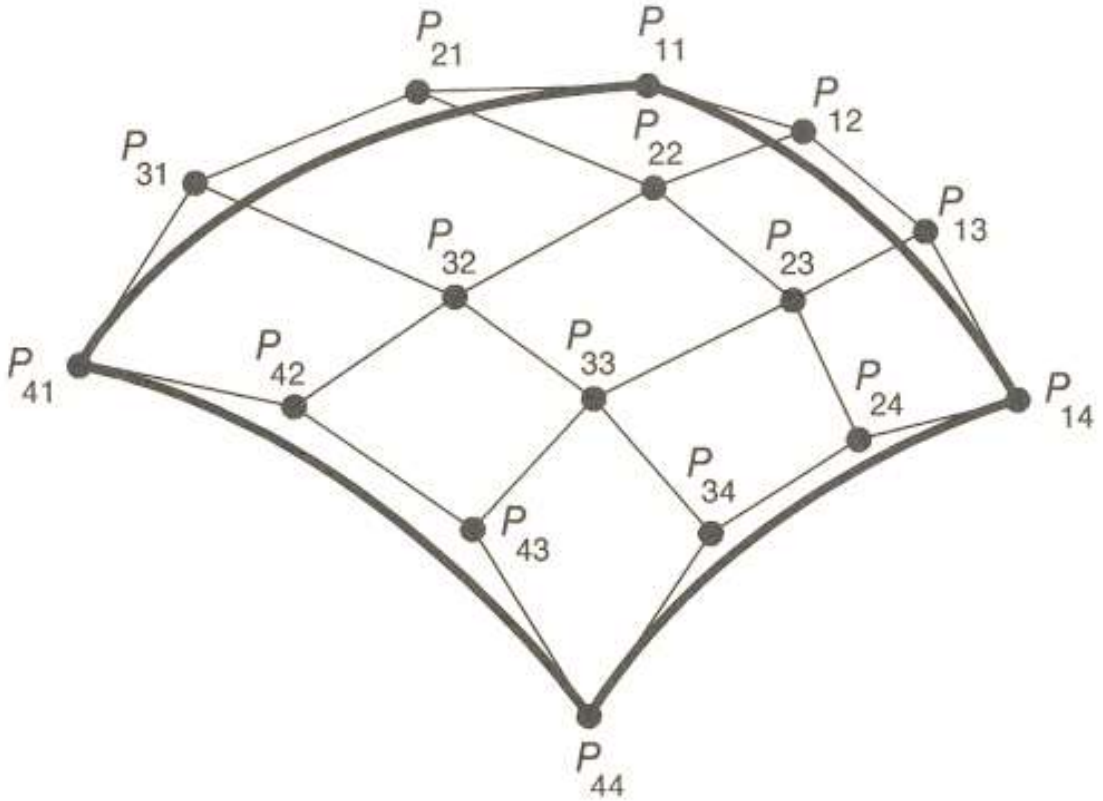
- **Each patch is defined by blending control points**
  - Same ideas as parametric splines!



Watt Figure 6.25

Parametric Patches

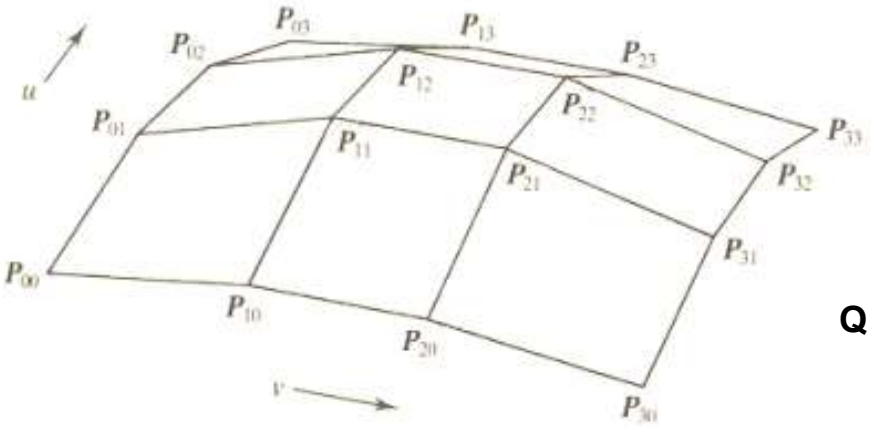
- **Each patch is defined by blending control points**
  - Same ideas as parametric curves!

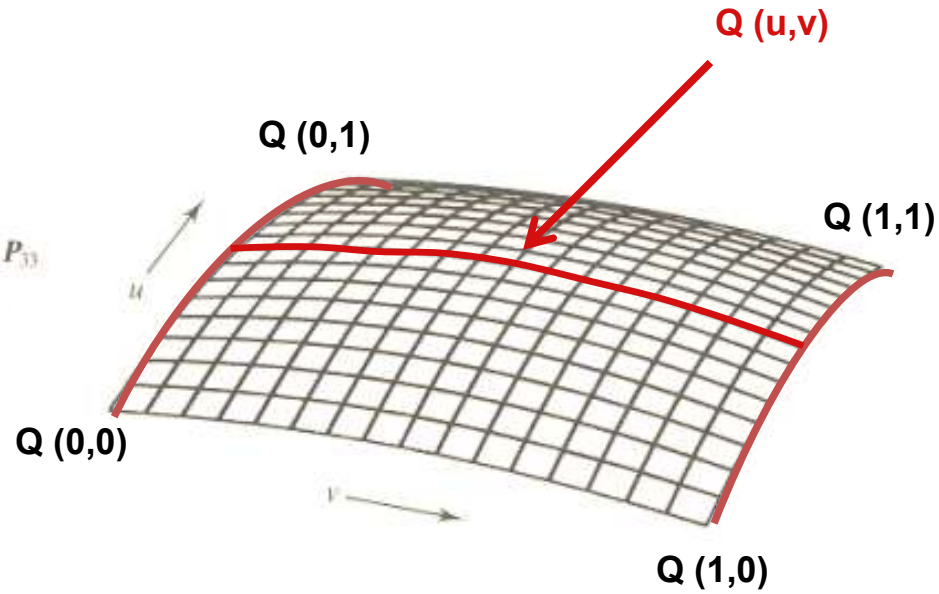


FvDFH Figure 11.44

Parametric Patches

- Point **Q(u,v)** on the patch is the tensor product of parametric curves defined by the control points





Watt Figure 6.21

Let the Control Points Move

- Call the Bézier parameter *v, and let the N+1 control* points depend on some other parameter *u:*

$$p(u,v)=\sum_{k=0}^{N} p_k(u)B_k^N(v)$$

- Each “u-contour” is a normal Bézier curve, but at different *u values, the control points are at different* positions

- Think of the surface as a changing Bézier curve sweeping through space

- How do the control points change?

Bézier Patches

- Let's allow the control points to move along their own Bézier curves:

$$p_k(u) = \sum_{i=0}^{N} p_{i,k} B_i^N(u)$$

- Putting this together with our original definition of our surface, we get the *tensor product form for the Bézier patch:*

$$p(u,v) = \sum_{i=0}^{M} \sum_{k=0}^{N} p_{i,k} B_i^M(u) B_k^N(v)$$

Parametric Bicubic Patches

- Point **Q(u,v)** on any patch is defined by combining control points with polynomial blending functions:

$$Q(u,v) = \mathbf{U}\mathbf{M}\begin{bmatrix} P_{1,1} & P_{1,2} & P_{1,3} & P_{1,4} \\ P_{2,1} & P_{2,2} & P_{2,3} & P_{2,4} \\ P_{3,1} & P_{3,2} & P_{3,3} & P_{3,4} \\ P_{4,1} & P_{4,2} & P_{4,3} & P_{4,4} \end{bmatrix}\mathbf{M}^{\mathbf{T}}\mathbf{V}^{\mathbf{T}}$$

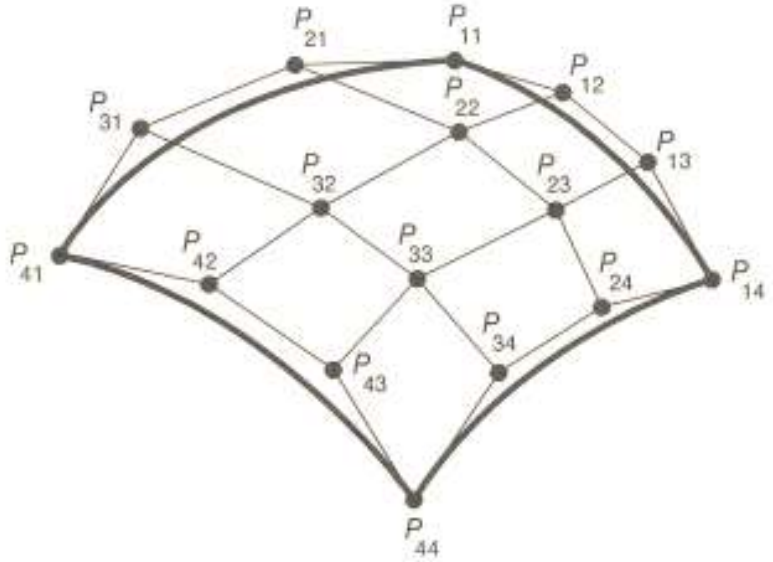
$$\mathbf{U} = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \qquad \mathbf{V} = \begin{bmatrix} v^3 & v^2 & v & 1 \end{bmatrix}$$

- Where M is a matrix describing the blending functions for a parametric cubic curve (e.g., Bezier, B-spline, etc.)

Bezier Patches

$$Q(u,v) = \mathbf{U}\mathbf{M}_{\mathbf{Bezier}} \begin{bmatrix} P_{1,1} & P_{1,2} & P_{1,3} & P_{1,4} \\ P_{2,1} & P_{2,2} & P_{2,3} & P_{2,4} \\ P_{3,1} & P_{3,2} & P_{3,3} & P_{3,4} \\ P_{4,1} & P_{4,2} & P_{4,3} & P_{4,4} \end{bmatrix} \mathbf{M}_{\mathbf{Bezier}}^{\mathbf{T}} \mathbf{V}$$

$$\mathbf{M}_{\mathbf{Bezier}} = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$



FvDFH Figure 11.42

B-Spline Patches

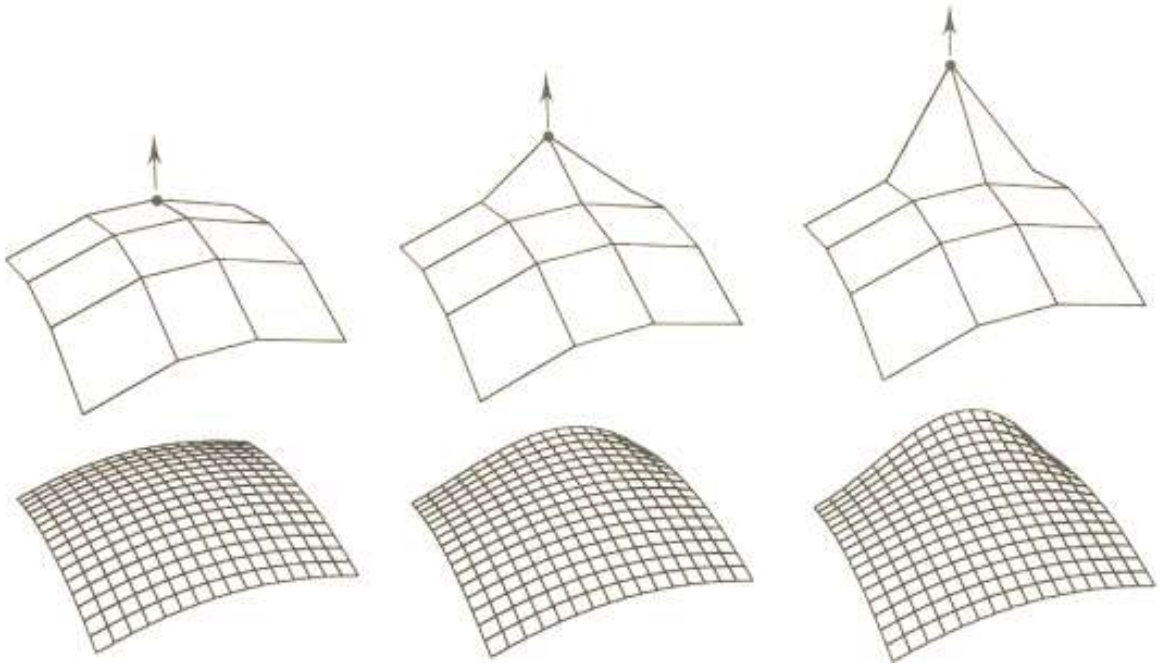
$$Q(u,v) = \mathbf{U}\mathbf{M}_{\mathbf{B-Spline}} \begin{bmatrix} P_{1,1} & P_{1,2} & P_{1,3} & P_{1,4} \\ P_{2,1} & P_{2,2} & P_{2,3} & P_{2,4} \\ P_{3,1} & P_{3,2} & P_{3,3} & P_{3,4} \\ P_{4,1} & P_{4,2} & P_{4,3} & P_{4,4} \end{bmatrix} \mathbf{M}^{\mathbf{T}}_{\mathbf{B-Spline}} \mathbf{V}$$

$$\mathbf{M}_{\mathbf{B-Spline}} = \begin{bmatrix} -1/6 & 1/2 & -1/2 & 1/6 \\ 1/2 & -1 & 1/2 & 0 \\ -1/2 & 0 & 1/2 & 0 \\ 1/6 & 2/3 & 1/6 & 0 \end{bmatrix}$$

Watt Figure 6.28

Bezier Patches

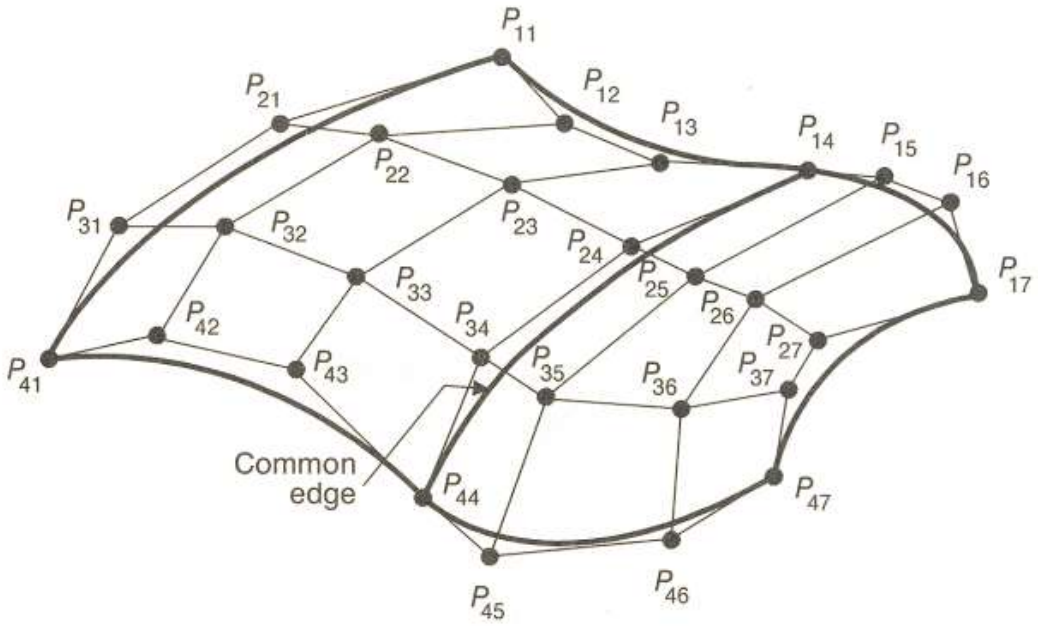
- **Properties**:
  - Interpolates four corner points
  - Convex hull
  - Local control



Watt Figure 6.22

Bezier Surfaces

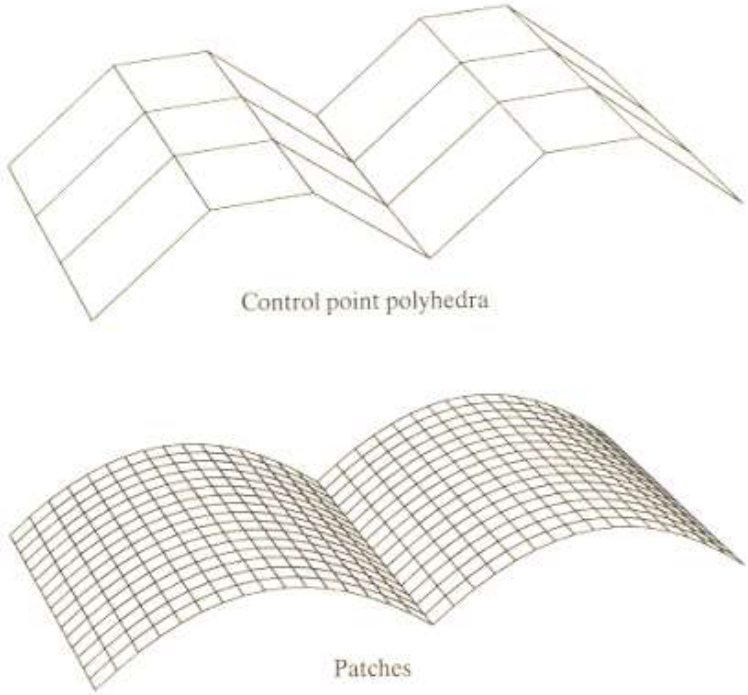
- **Continuity constraints are similar to the contraints for Bezier splines**



FvDFH Figure 11.43

Bezier Surfaces

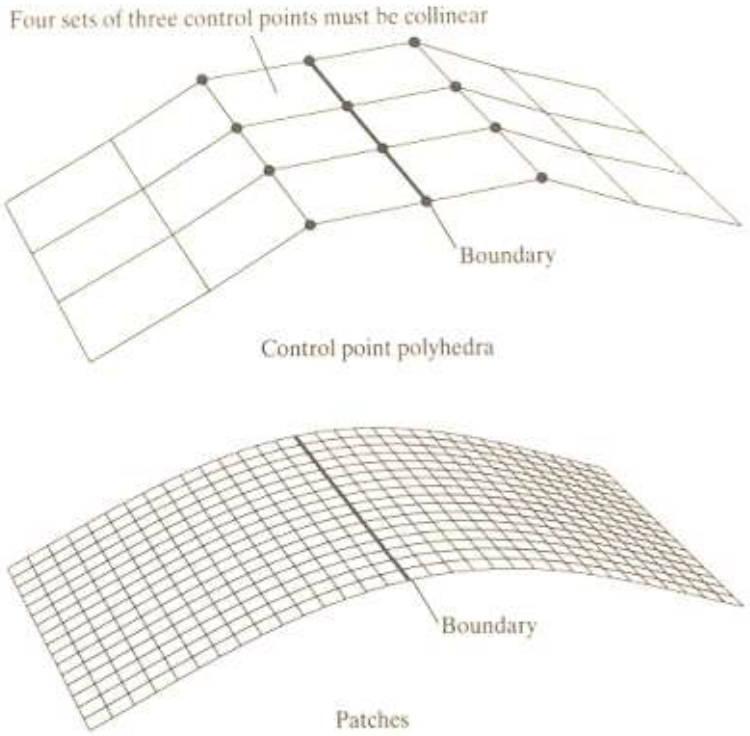
- **C0** continuity requires aligning boundary curves



Watt Figure 6.26a

Bezier Surfaces

- **C1** continuity requires aligning boundary curves and derivatives (a reason to prefer subdivision surf.)

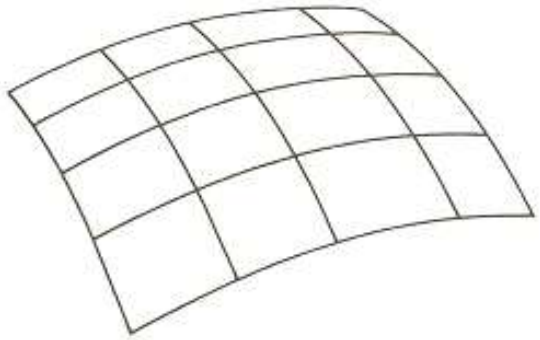


Watt Figure 6.26b

Drawing Bezier Surfaces

- **Simple approach is to loop through uniformly spaced increments of u and v**

```
DrawSurface(void)
{
    for (int i = 0; i < imax; i++)
    {
        float u = umin + i * ustep;
        for (int j = 0; j < jmax; j++)
        {
          float v = vmin + j * vstep;
          DrawQuadrilateral(...);
        }
    }
}
```

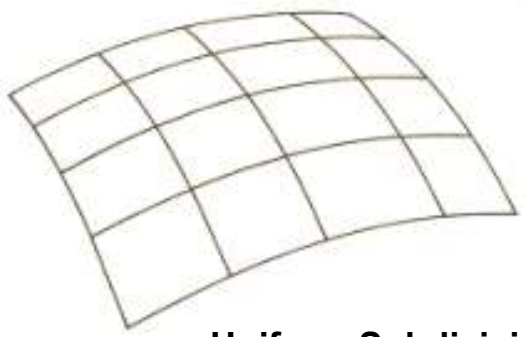


Watt Figure 6.32

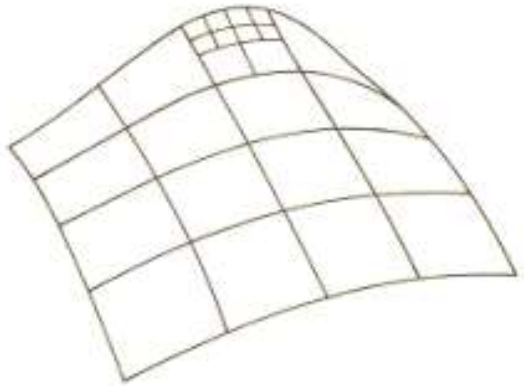
Drawing Bezier Surfaces

- **Better approach is to use adaptive subdivision:**

```
DrawSurface(surface)
{
     if Flat (surface, epsilon) {
       DrawQuadrilateral(surface);
     }
     Else
     {
          SubdivideSurface(surface, ...);
          DrawSurface(surfaceLL);
          DrawSurface(surfaceLR);
          DrawSurface(surfaceRL);
          DrawSurface(surfaceRR);}
}
```



Uniform Subdivision



Adaptive Subdivision

Watt Figure 6.32

Bézier Curves in OpenGL

- **Use *evaluators***
  - **glMap** defines the set of control points
  - **glMapGrid** defines how finely to evaluate the surface
  - **glEvalCoord/glEvalMesh** cause the mesh to be drawn

Defining the Control Points

- **glMap2 [df](target, u1, u2, ustride, uorder, v1, v2, vstride, vorder, points)**
  - **target** specifies what OpenGL command will be executed when this mesh is evaluated, and what's in the control mesh. For drawing, usually use **GL_MAP2_VERTEX3**
  - **u1,u2,v1,v2** define a mapping from values passed to **glEvalCoord** to (0,1), the domain of the Bézier functions
  - **ustride, vstride** indicate how the data is packed in the array
  - **uorder, vorder** define the dimensions of the point array
  - **points** is the actual data

Defining the Control Points

- **glMap2d(GL_MAP2_VERTEX_3, 0, 1, 3, 4, 0, 1, 12, 4, &ctrlpoints1[0][0][0]);**

Defining the Mesh Parameters

- **glMapGrid** specifies how the mesh will be evaluated based on the control points

- **glMapGrid2[df] (un, u1, u2, vn, v1, v2)**

  - **un, vn** define the number of partitions at which to evaluate the surface

  - **u1, u2, v1, v2** define the range of the grid variables

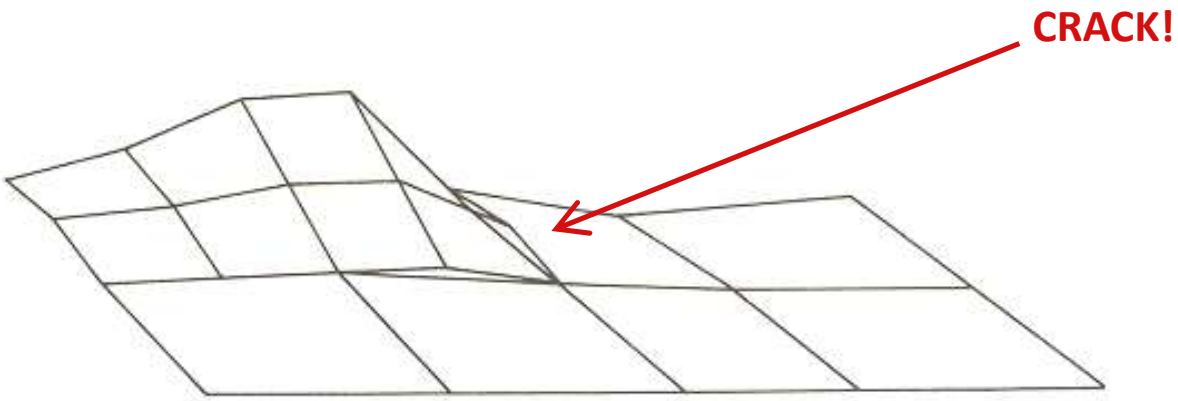
- **glMapGrid2d (20, 0.0, 1.0, 20.0, 0.0, 1.0);**

Drawing the Mesh

- **We can draw the whole mesh at once with:**
- **glEvalMesh**
- **glEvalMesh2 (mode, i1, i2, j1, j2)**
  - **Mode** specifies points, lines or polygons
  - **i1, i2, j1, j2** define the range over which the to evaluate the mesh
- **glEvalMesh2 (GL_LINE, 0, 20, 0, 20);**

Drawing Bezier Surfaces

- One problem with adaptive subdivision is avoiding cracks at boundaries between patches at different subdivision levels



- Avoid these cracks by adding extra vertices and triangulating quadrilaterals whose neighbours are subdivided to a finer level.

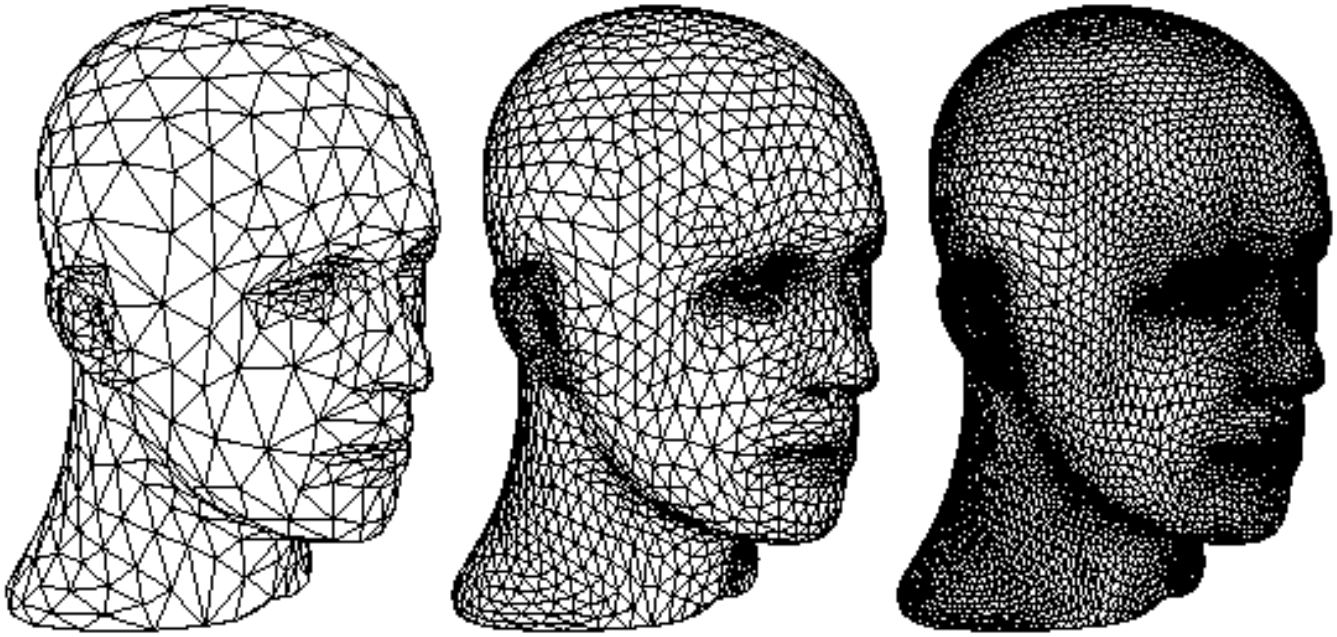
Watt Figure 6.33

Parametric Surfaces

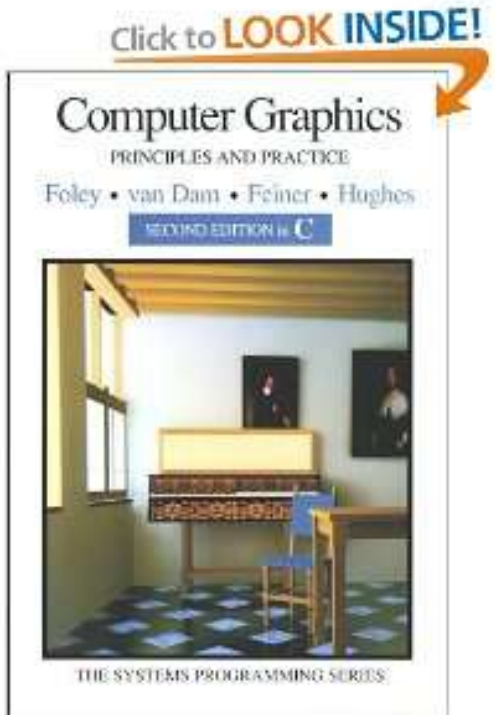
- **Advantages:**
  - Easy to enumerate points on surface
  - Possible to describe complex shapes

- **Disadvantages:**
  - Control mesh must be quadrilaterals
  - Continuity constraints difficult to maintain
  - Hard to find intersections

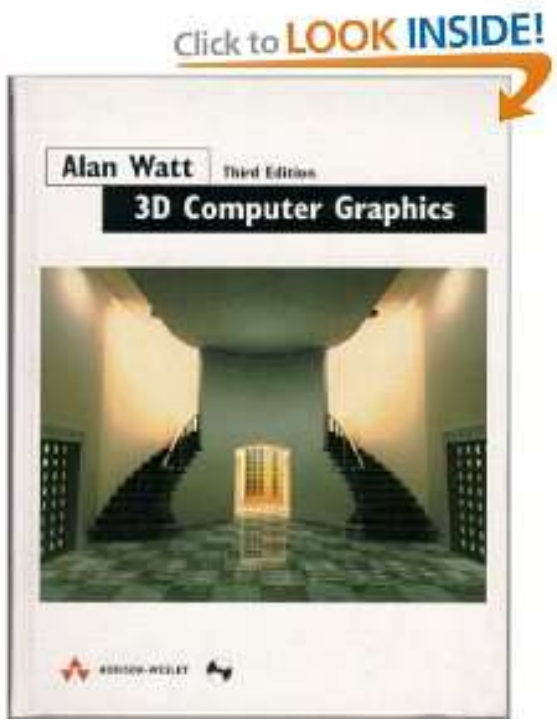
Subdivision Surfaces



Many of the images are from:



Foley, van Dam
Computer Graphics: Principles and Practice (now FvDFH)



Alan Watt
3D Computer Graphics

End