07. Higher-Order Surfaces

Dr. Hamish Carr

Motivation

- We built Bézier curves in 2-D:
  - Because not everything is straight
  - So curves need a separate representation
  - And we built them using polynomials
- We can do much the same thing in 3-D
  - Higher-order surfaces:
    - Bézier Patches, Subdivision Surfaces, &c.

Bézier Curve

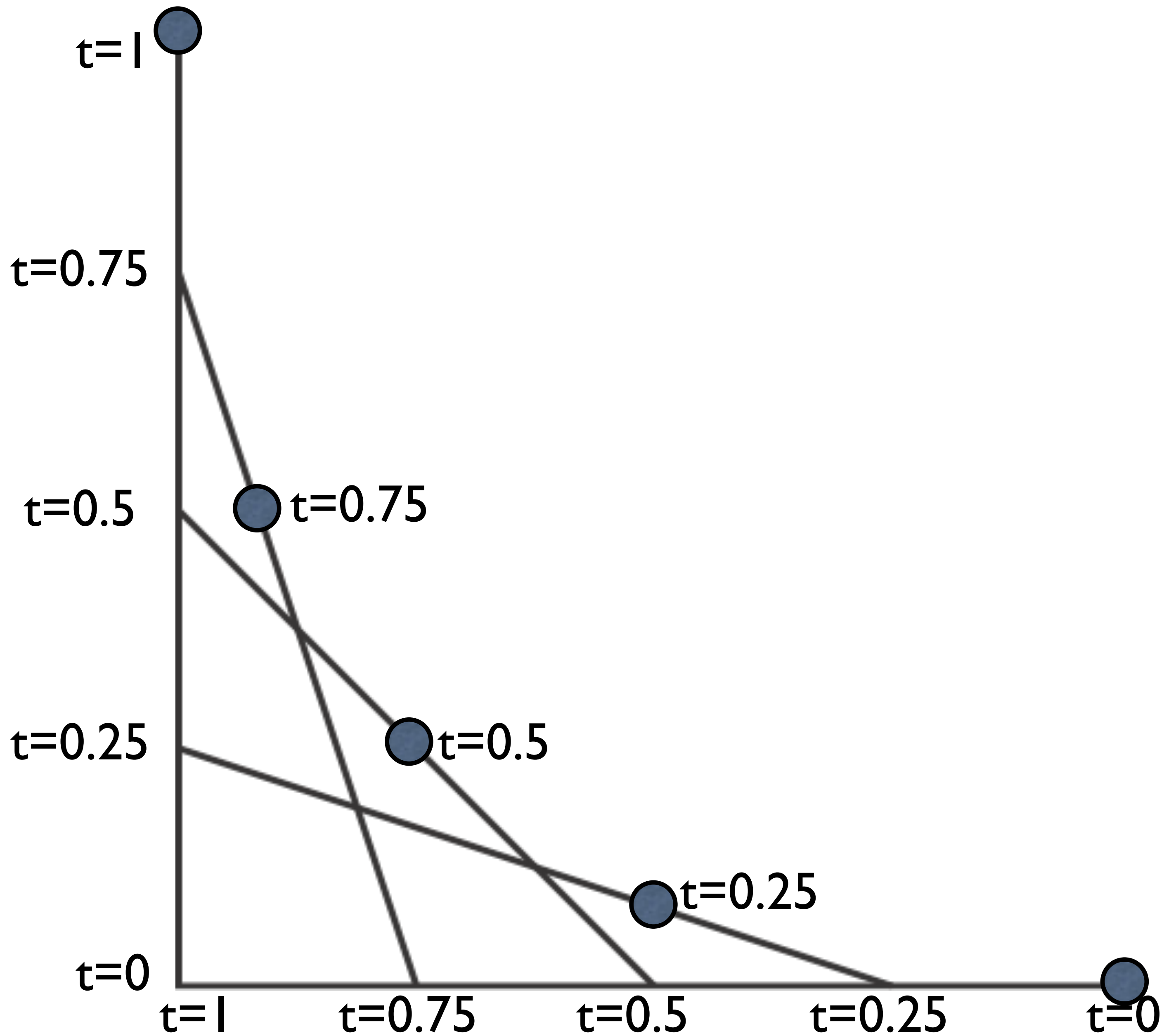
- Repeated linear interpolation
  - based on control polygon
  - actually a polyline
- Recursively defined (de Casteljau)

$$\begin{aligned} p_{i,j} &= (1-t)p_{i,j+1} + tp_{i+1,j} \\ &= \alpha p_{i,j+1} + \beta p_{i+1,j} \end{aligned}$$

- Note barycentric indices

$$\alpha + \beta = 1$$

Construction



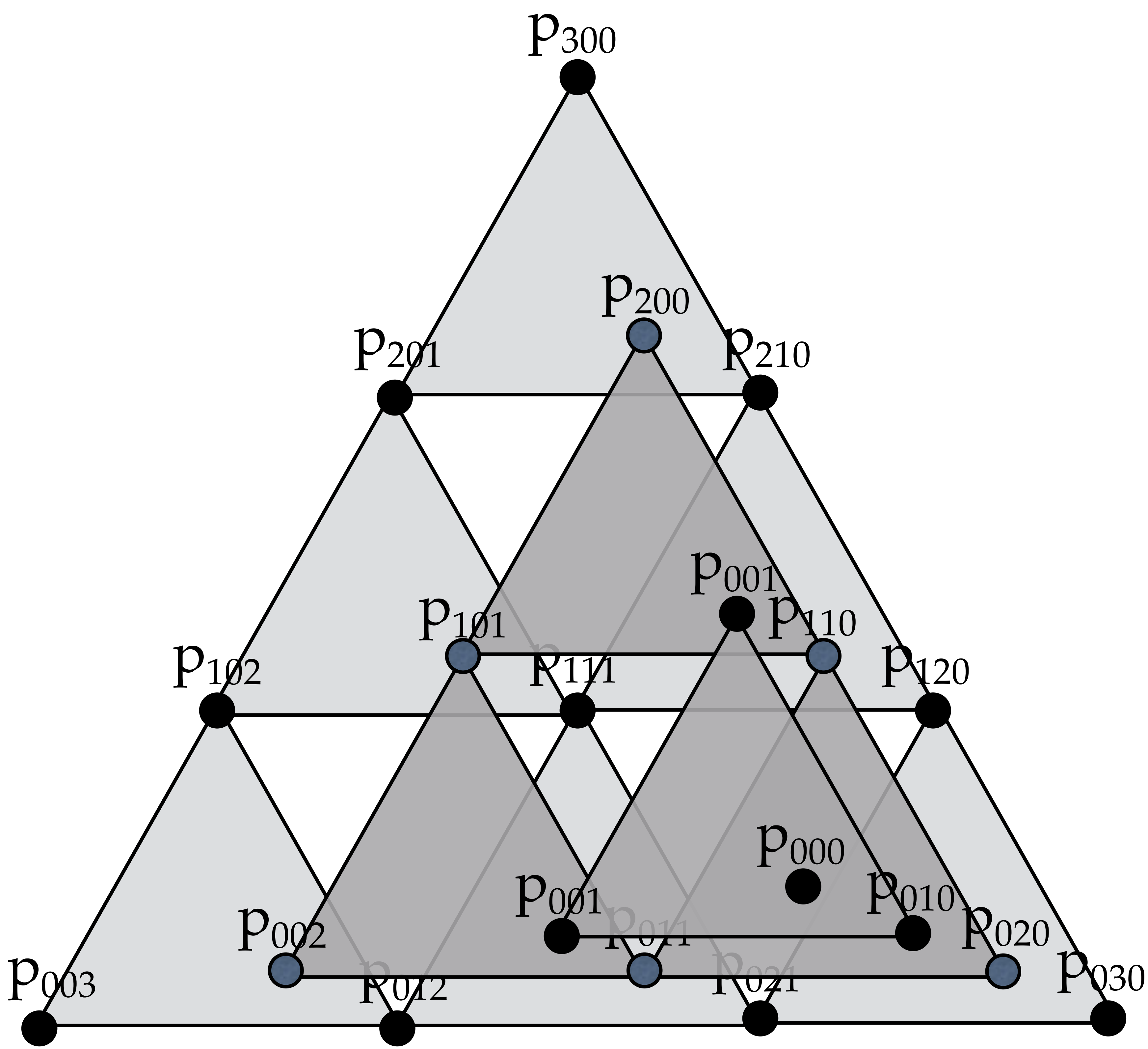
UNIVERSITY OF LEEDS

Bézier Triangle

- Generalisation of triangles
- Repeated linear interpolation
  - based on control net
- Recursively defined (de Casteljau)

$$
\begin{aligned}
p_{i,j,k} &= \alpha p_{i+1,j,k} + \beta p_{i,j+1,k} + \gamma p_{i,j,k+1} \\
\alpha + \beta + \gamma &= 1
\end{aligned}
$$

Construction



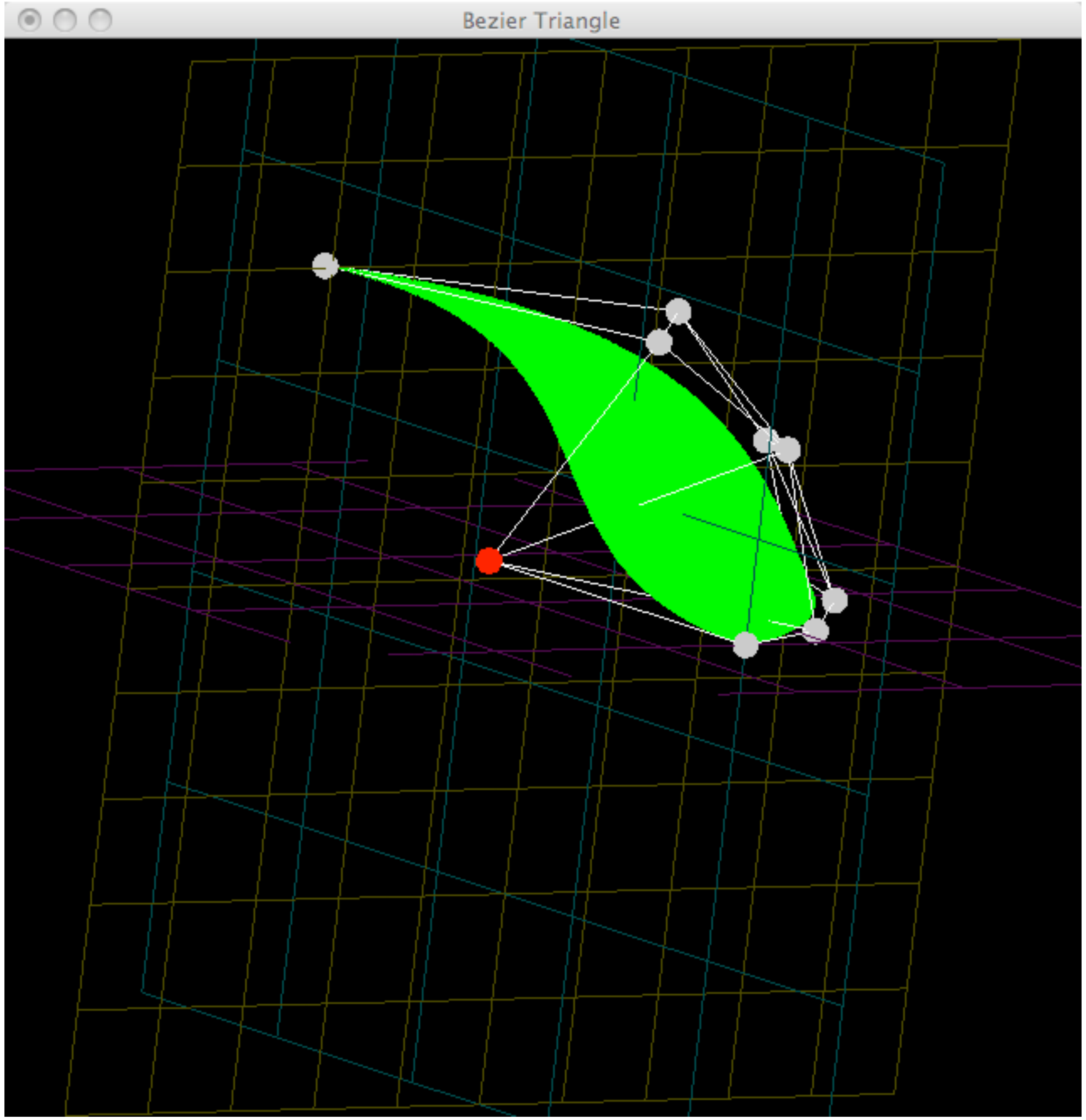
Properties

- Control edges generate Bézier curves
- Bézier patch is cubic polynomial
- Hermite patches also exist
  - & can be converted to/from Bézier
- Bounded by convex hull of control net
- Local control

de Casteljau Patches

```
int N_PTS = 4;
Point bezPoints[N_PTS][N_PTS][N_PTS];

void DrawBezier()
  { // DrawBezier()
  for (float alpha = 0.0f; alpha <= 1.0f; alpha += 0.01f)
    { // parameter loop
    for (float beta = 0.0f; beta <= 1.0f; beta += 0.01f)
      { // parameter loop
      float gamma = 1.0f - alpha - beta;
      for (int diag = N_PTS-1; diag >= 0; diag--)
        { // diagonal loop
        for (int i = 0; i <= diag; i++)
          { // i loop
          for (int j = 0; j <= diag - i; j++)
            { // i loop
            int k = diag - i - j;
            bezPoints[i][j][k] =   alpha*bezPoints[i+1][j  ][k  ]
                                 + beta *bezPoints[i  ][j+1][k  ];
                                 + gamma*bezPoints[i  ][j  ][k+1];
             } // j loop
           } // i loop
        } // diagonal loop
      // draw the point
      DrawPoint(bezPoints[0][0][0]);
      } // parameter loop
    } // parameter loop
  } // DrawBezier()
```

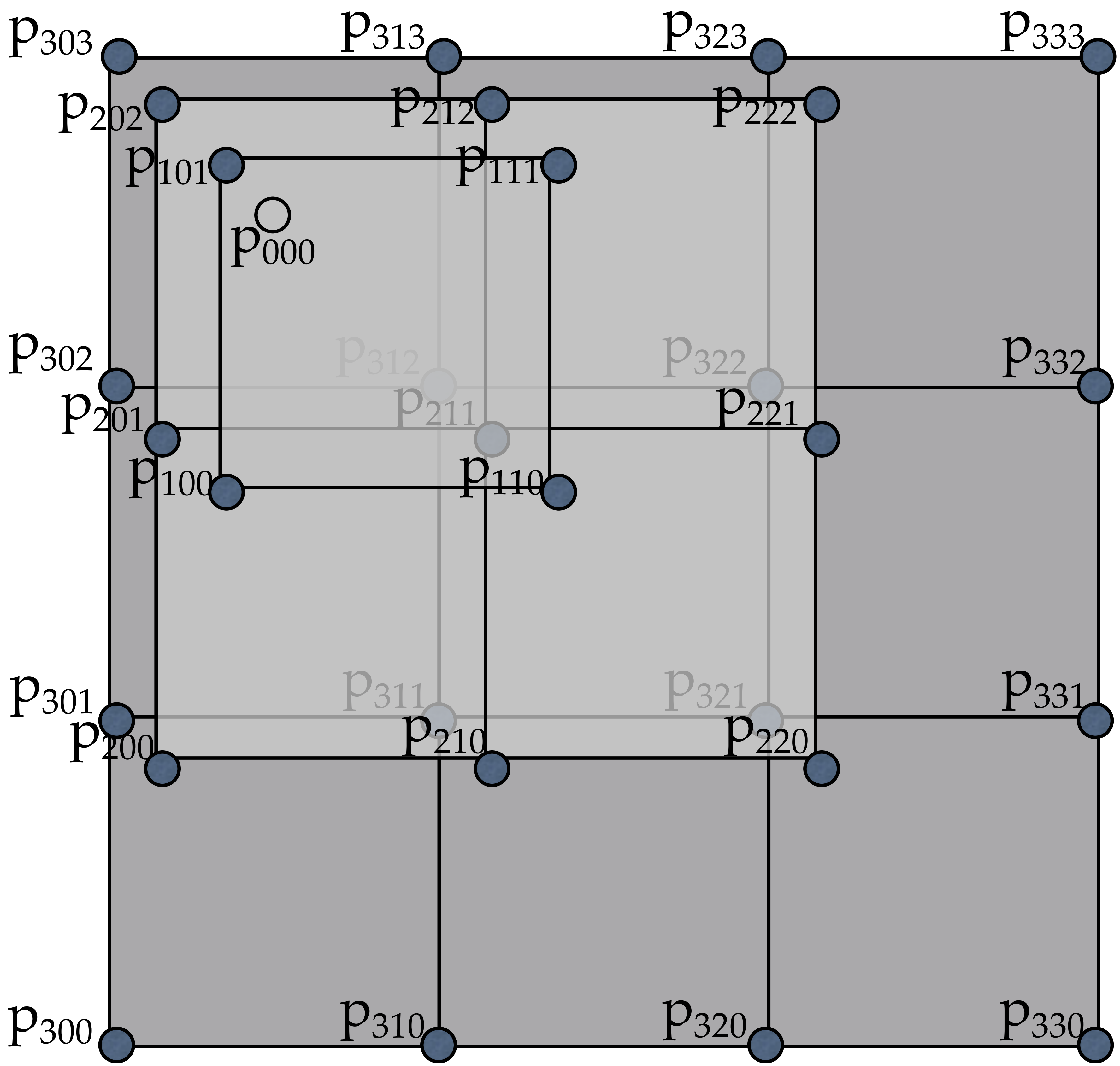
Example



Bézier Tensor Patches

- Triangles are not always best
  - quadrilaterals are often easier
  - easier to construct large surfaces
  - easier to texture
- So how can we do this with squares?
  - Iterate *bilinear* interpolation

Bilinear Béziers



Computation

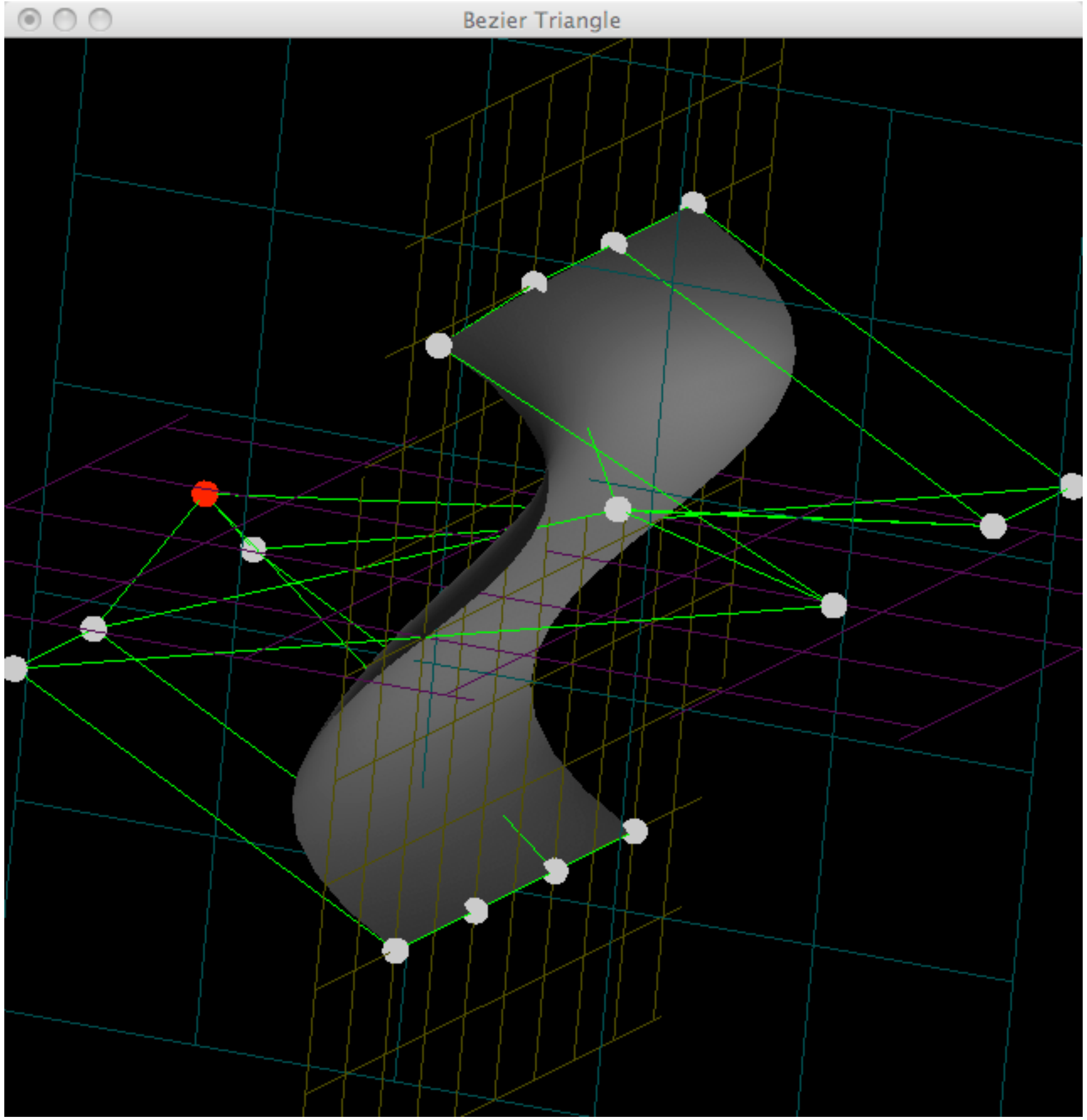
$$p_{i,j,k} = (1-s)(1-t)p_{i+1,j,k} + s(1-t)p_{i+1,j+1,k} + (1-s)tp_{i+1,j,k+1} + s(1-t)p_{i+1,j+1,k+1}$$

$$\begin{aligned} f(s,t) = & \quad (1-s)^3 \big((1-t)^3 p_{300} + 3t(1-t)^2 p_{301} + 3t^2(1-t)p_{302} + t^3 p_{303}\big) \\ & + 3s(1-s)^2 \big((1-t)^3 p_{310} + 3t(1-t)^2 p_{311} + 3t^2(1-t)p_{312} + t^3 p_{313}\big) \\ & + 3s^2(1-s) \big((1-t)^3 p_{320} + 3t(1-t)^2 p_{321} + 3t^2(1-t)p_{322} + t^3 p_{323}\big) \\ & + s^3 \big((1-t)^3 p_{330} + 3t(1-t)^2 p_{331} + 3t^2(1-t)p_{332} + t^3 p_{333}\big) \end{aligned}$$

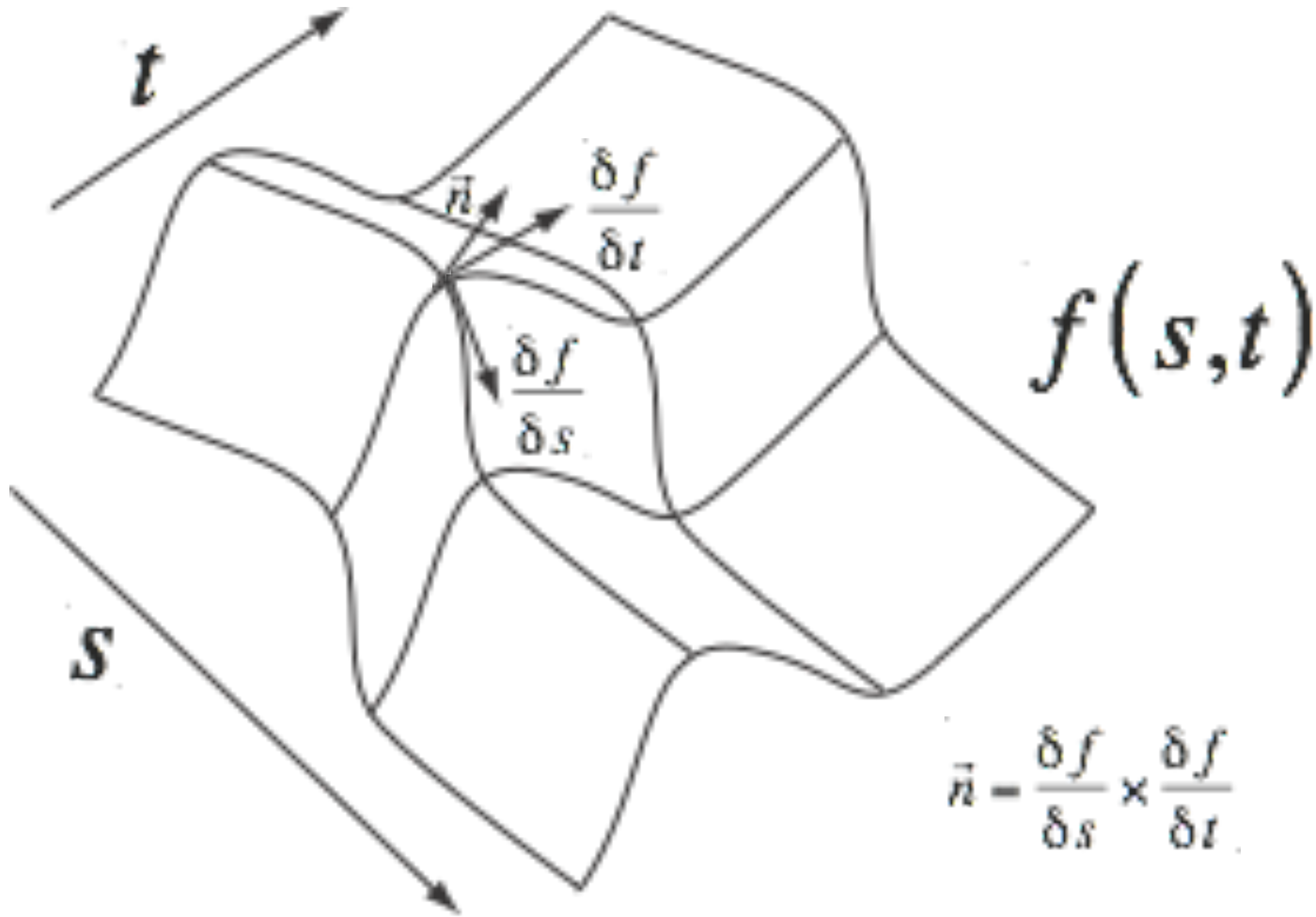
$$\begin{aligned} & \quad (1-t)^3 \big((1-s)^3 p_{300} + 3s(1-s)^2 p_{310} + 3s^2(1-s)p_{320} + s^3 p_{330}\big) \\ & + 3t(1-t)^2 \big((1-s)^3 p_{301} + 3s(1-s)^2 p_{311} + 3s^2(1-s)p_{321} + s^3 p_{331}\big) \\ & + 3t^2(1-t) \big((1-s)^3 p_{302} + 3s(1-s)^2 p_{312} + 3s^2(1-s)p_{322} + s^3 p_{332}\big) \\ & + t^3 \big((1-s)^3 p_{303} + 3s(1-s)^2 p_{313} + 3s^2(1-s)p_{323} + s^3 p_{333}\big) \end{aligned}$$

- The individual terms are Bézier curves
- And we can compute direction vectors
- Then compute normal vectors

Example



Direction Vectors



UNIVERSITY OF LEEDS

Normal Vectors

- Take crossproduct of two vectors
  - tangent to the surface
  - direction vectors
    - partial derivatives
  - from the separated Bézier curves

Normal Vectors

$$
\begin{aligned}
\frac{\partial f}{\partial s}(s,t) \quad = \quad & (-3+6s-3s^2) && \left((1-t)^3 p_{300} + 3t(1-t)^2 p_{301} + 3t^2(1-t)p_{302} + t^3 p_{303}\right) \\
+ \quad & (3-12s+9s^2) && \left((1-t)^3 p_{310} + 3t(1-t)^2 p_{311} + 3t^2(1-t)p_{312} + t^3 p_{313}\right) \\
+ \quad & (6s-9s) && \left((1-t)^3 p_{320} + 3t(1-t)^2 p_{321} + 3t^2(1-t)p_{322} + t^3 p_{323}\right) \\
+ \quad & 3s^2 && \left((1-t)^3 p_{330} + 3t(1-t)^2 p_{331} + 3t^2(1-t)p_{332} + t^3 p_{333}\right)
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial f}{\partial t}(s,t) \quad & (-3+6s-3t^2) && \left((1-s)^3 p_{300} + 3s(1-s)^2 p_{310} + 3s^2(1-s)p_{320} + s^3 p_{330}\right) \\
+ \quad & (3-12t+9t^2) && \left((1-s)^3 p_{301} + 3s(1-s)^2 p_{311} + 3s^2(1-s)p_{321} + s^3 p_{331}\right) \\
+ \quad & (6t-9t) && \left((1-s)^3 p_{302} + 3s(1-s)^2 p_{312} + 3s^2(1-s)p_{322} + s^3 p_{332}\right) \\
+ \quad & 3t^2 && \left((1-s)^3 p_{303} + 3s(1-s)^2 p_{313} + 3s^2(1-s)p_{323} + s^3 p_{333}\right)
\end{aligned}
$$

$$\vec{n} = \frac{\partial f}{\partial s}(s,t) \times \frac{\partial f}{\partial t}(s,t)$$

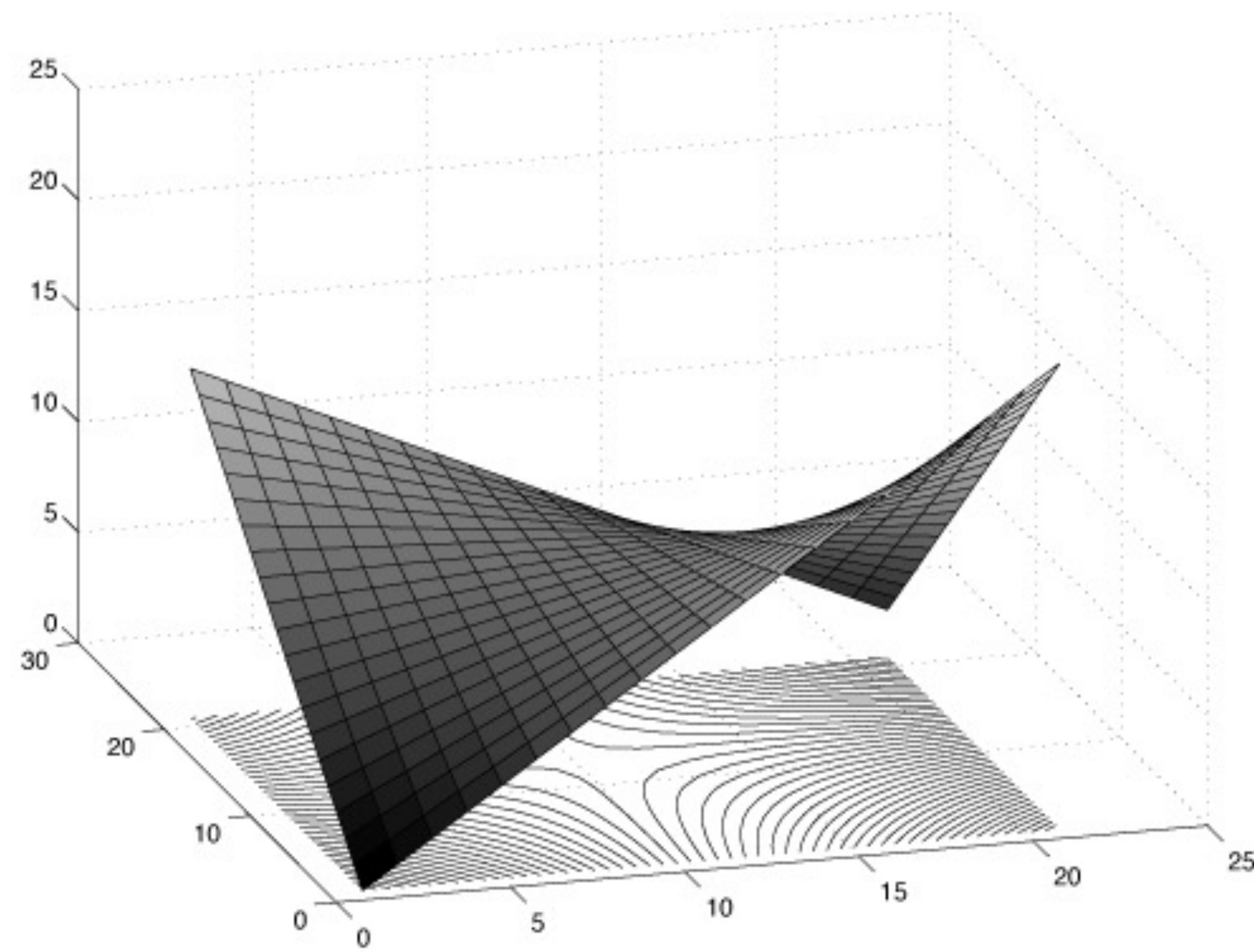
Shortcut

- Bézier patches are very common
  - so they are often built into libraries
  - E.g. OpenGL's Evaluators
- Now the basis of geometry shaders

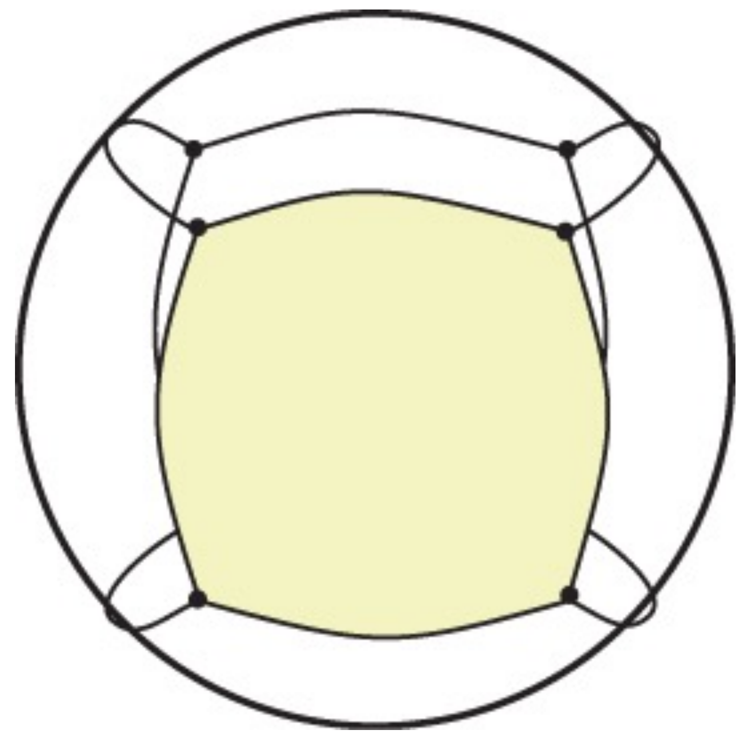
```
GLfloat ctrlpoints[4][4][3];

glMap2f(GL_MAP2_VERTEX_3, 0, 1, 3, 4, 0, 1, 2, 12, 4, &ctrlpoints[0][0][0]);

glEnable(GL_MAP2_VERTEX_3);

glMapGrid2f(20, 0,0, 1,0, 20, 0.0, 1.0);

glEvalCoord2f(i/30.0, j/30.0);
```

Evaluators

- Essentially, a loop through (s,t)
- Computes a set of smaller patches
- Which are rendered as triangles / quads
- I.e. details are stored *algorithmically*



Surface Construction



- Glue these patches together like triangles
- Make sure tangents match at edges
- Back to artistic control

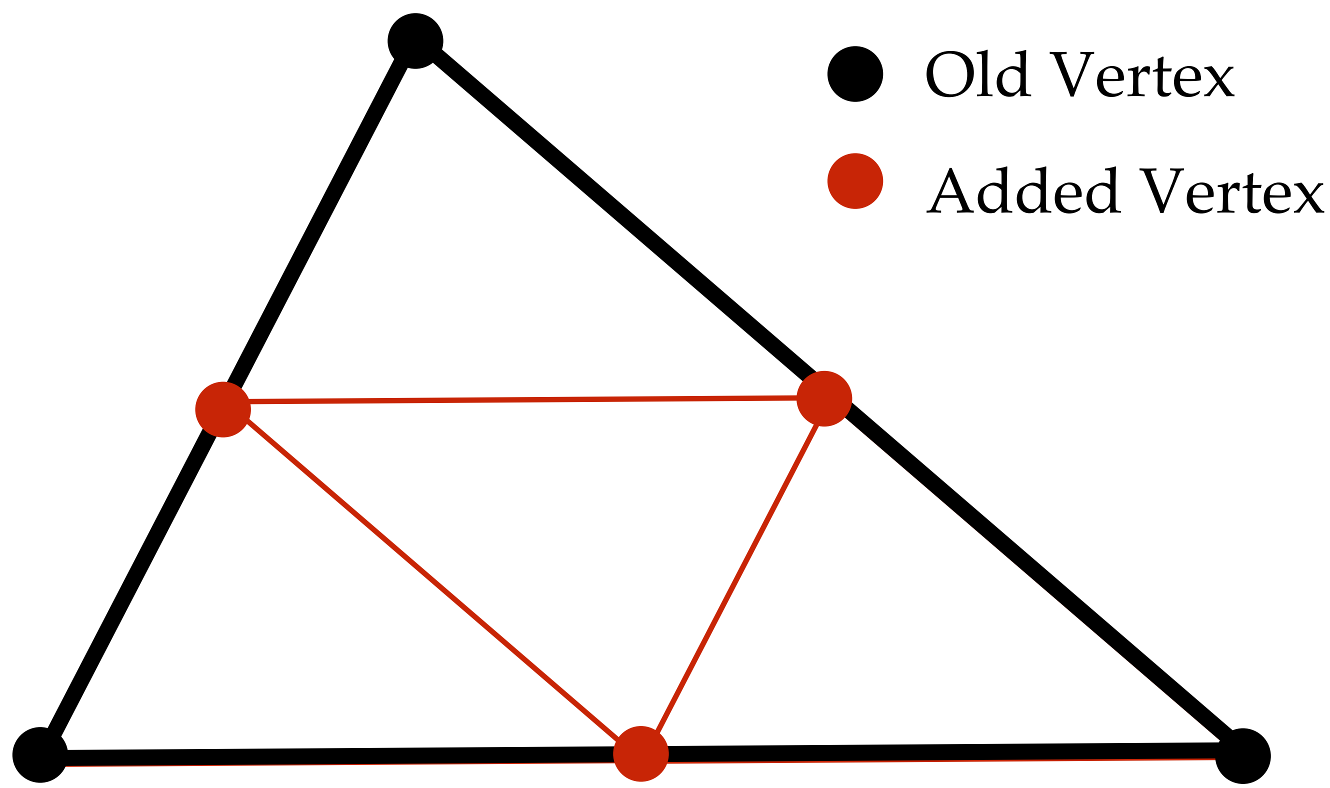
Subdivision Surfaces

- Another algorithm for smooth surfaces
- Start with a rough description
- Compute a finer resolution version
- Surface defined by *limit* of the computation
  - in practice, stop when quads < 1 pixel

Loop Subdivision

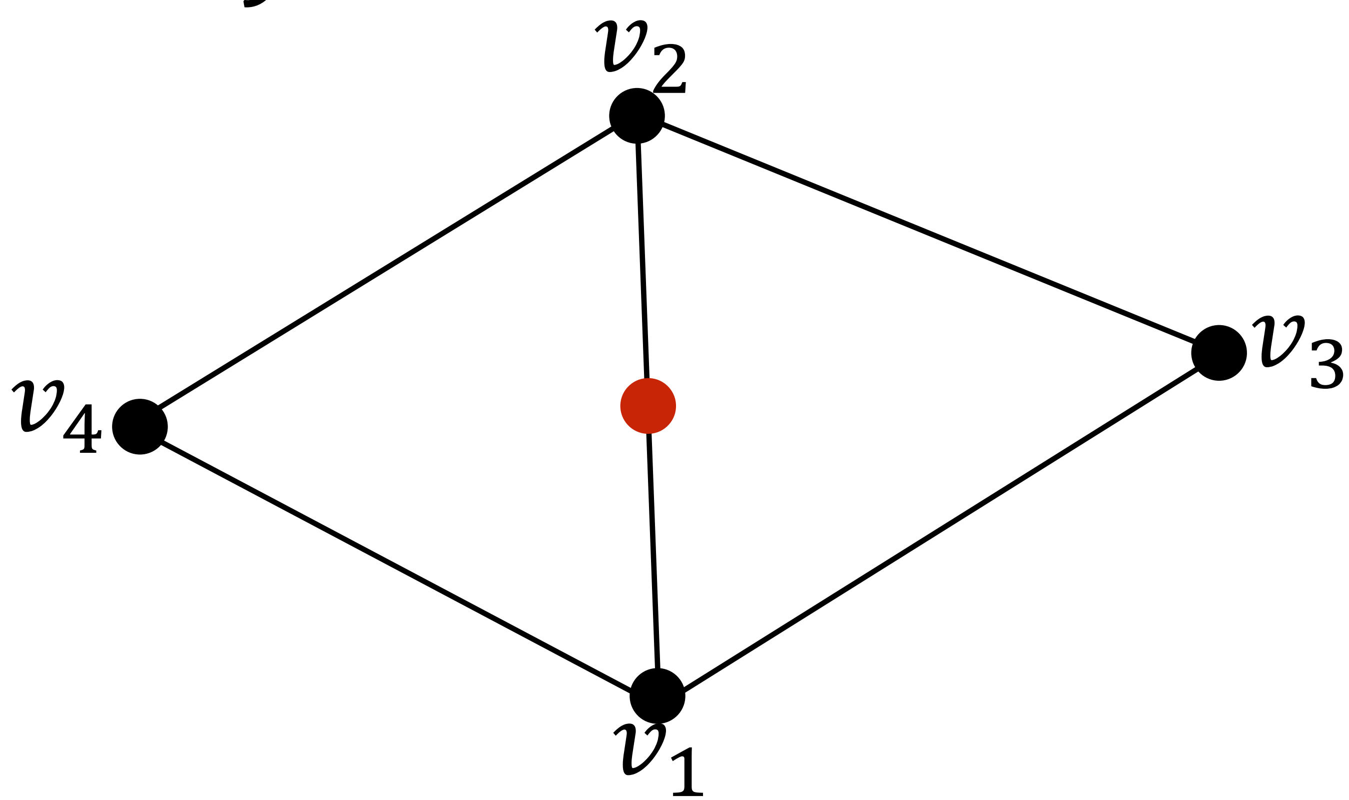
- Uses triangles
- Equivalent to quartic box spline
  - Insert one point along each edge
  - Each triangle becomes 4
- Then choose the geometric positions of:
  - new vertices
  - old vertices

Mesh Topology Changes



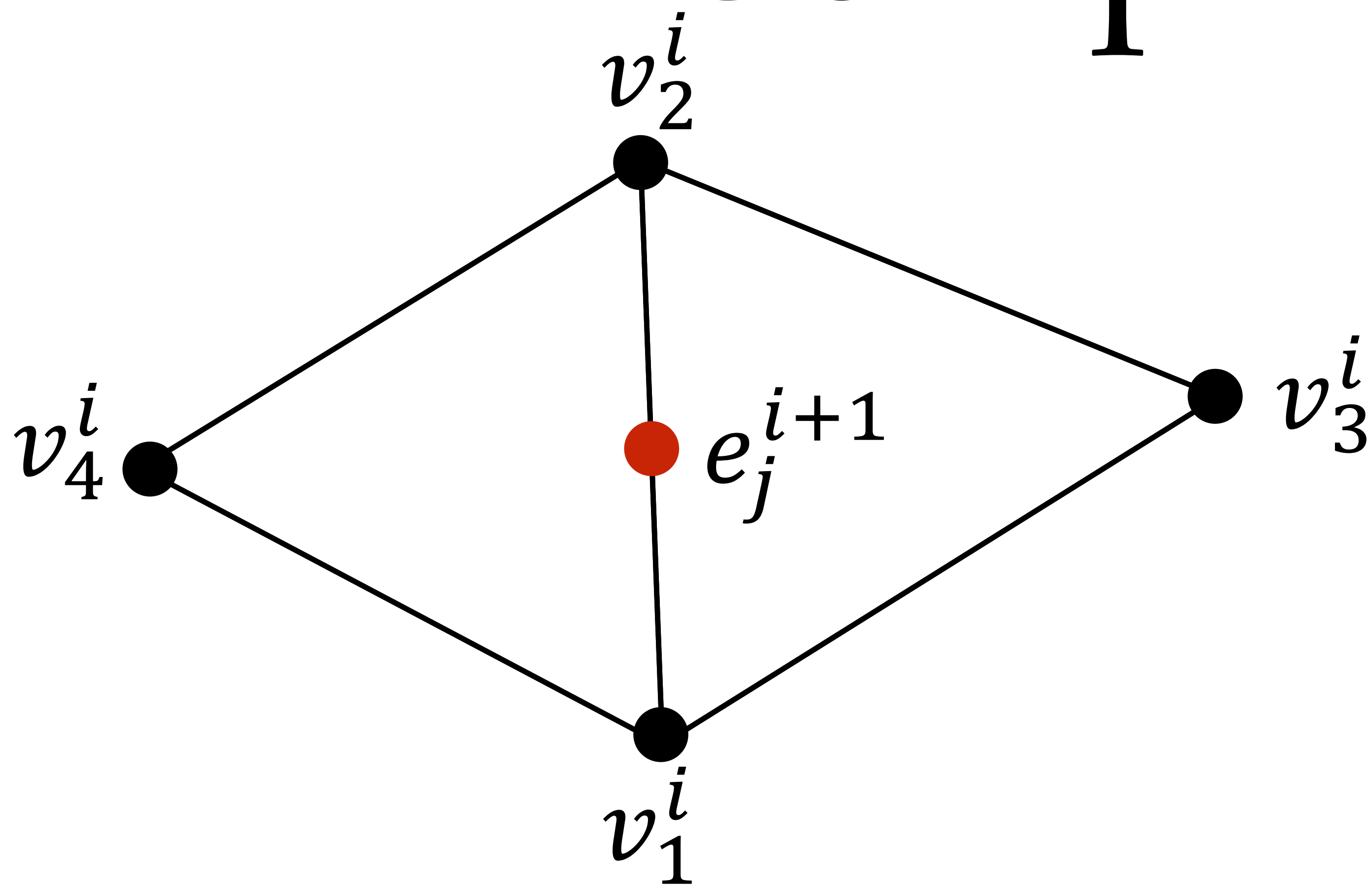
- Add vertex to each edge
- Divide face in 4
- But this gives us 4 coplanar triangles
- Which means the surface doesn't change

Adjacent Faces



- Look at the two faces *adjacent* to the edge
- Their vertices affect how sharp the edge is
- Take weighted average of all four vertices
- Neighbours account for 25% weight

Edge Vertex Computation



- To compute iteration i+1 from iteration i
  - $e_j^{i+1} = \frac{3}{8}\left(v_1^i + v_2^i\right) + \frac{1}{8}\left(v_3^i + v_4^i\right)$

Original Vertices

- If we keep their old position
  - We'll retain all sharp corners
- So we move them inwards as well

$$v_j^{i+1} = (1 - n\alpha)v_j^i + \alpha \sum_{v_k \in N_1(v_j)}^{n} v_k^i$$

  - where $\alpha = \frac{1}{n}\left(\frac{5}{8} - \left(\frac{3}{8} + \frac{1}{4} cos \frac{2\pi}{n}\right)^2\right)$
  - But if n=3, $\alpha = \frac{3}{16}$

Why This Flormula?

- Let's rewrite it:

$$\beta = n\alpha$$

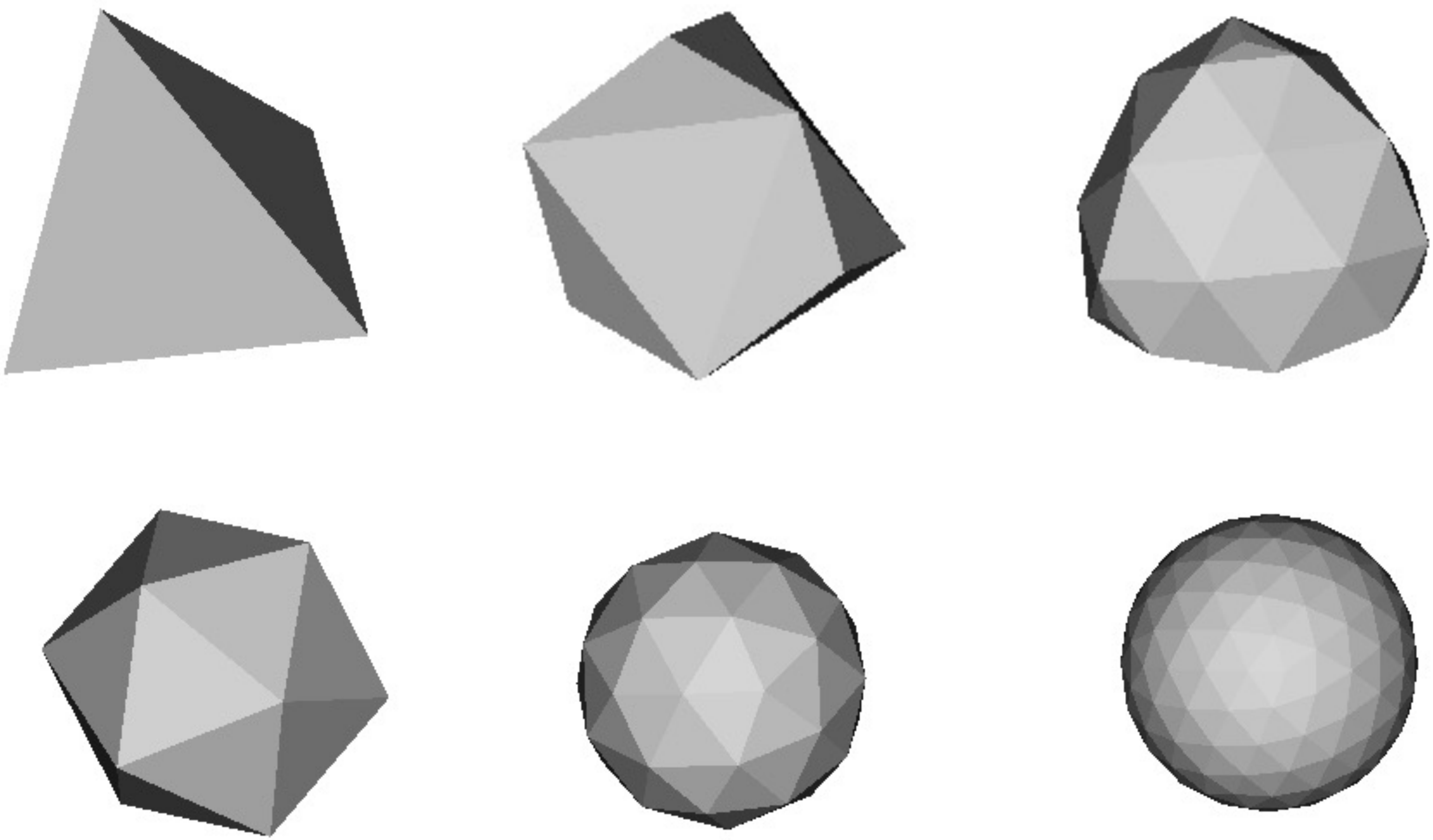
- $$= n\left(\frac{1}{n}\left(\frac{5}{8} - \left(\frac{3}{8} + \frac{1}{4}cos\frac{2\pi}{n}\right)^2\right)\right)$$

$$= \frac{5}{8} - \left(\frac{3}{8} + \frac{1}{4}cos\frac{2\pi}{n}\right)^2$$

  - $v_j^{i+1} = (1 - \beta)v_j^i + \beta\frac{\sum_{j=1}^{n} v_j^i}{n}$

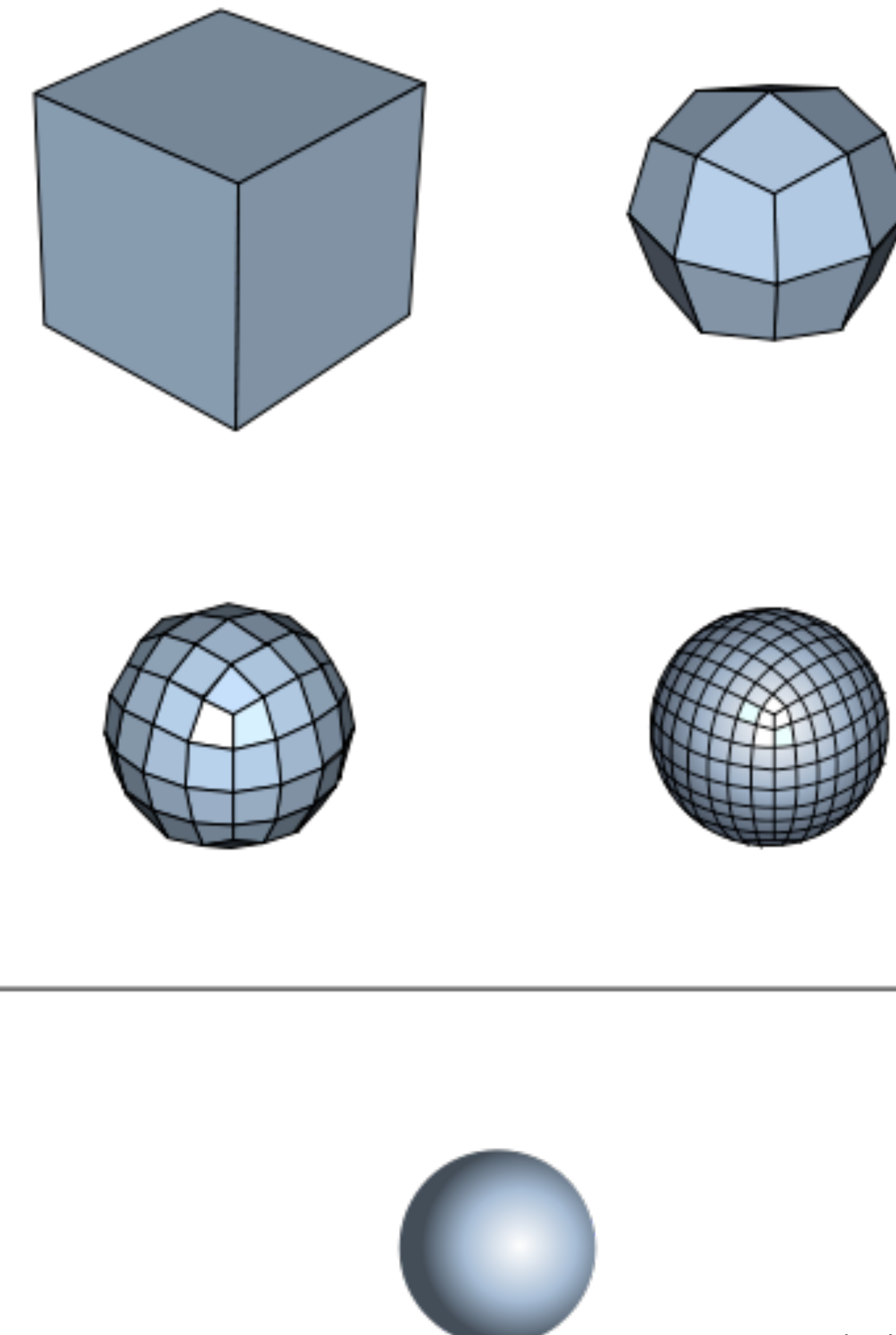
- Lerp between vertex and centroid of nbrs

Results



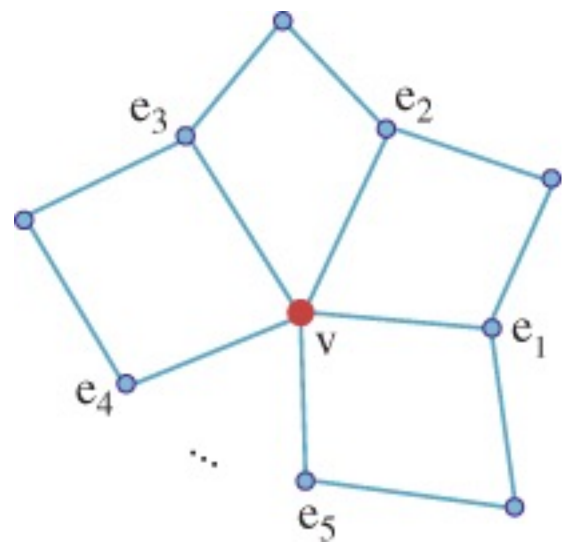
Catmull-Clark Subdivision

- Start with polyhedron
- Add *face* & *edge* points
  - fp = centroid of face
  - ep = midpoint of edge
- Move *original* points in
  - to “average” of fp/ep
- Repeat until done

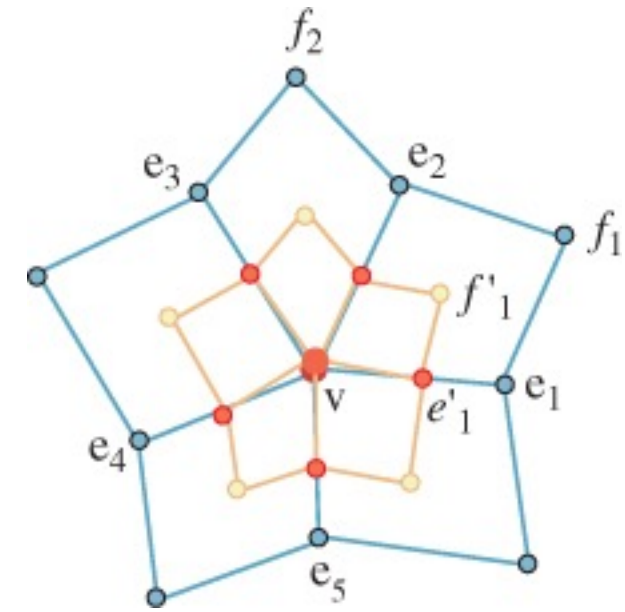


(From wikipedia)

Subdivision Construction



Before



After

1. Face vertex: $f_1' = \frac{v+e_1+e_2+f_1}{4}$

2. Edge vertex: $e_1' = \frac{v+e_2+f_1'+f_2^1}{4}$

3. Move vertex: $v' = \frac{n-2}{n} v + \frac{1}{n^2} \sum_i e_i + \frac{1}{n^2} \sum_i f_i'$

4. Connect as shown & make new faces

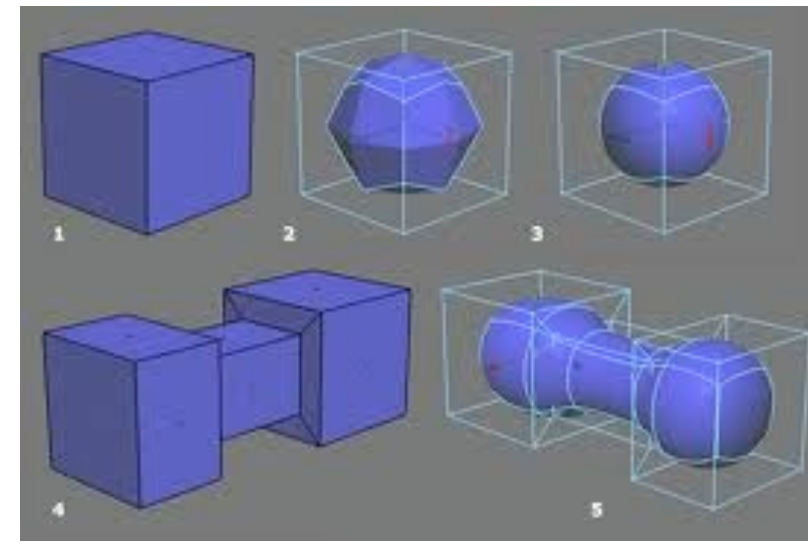
UNIVERSITY OF LEEDS

Subdivision Properties

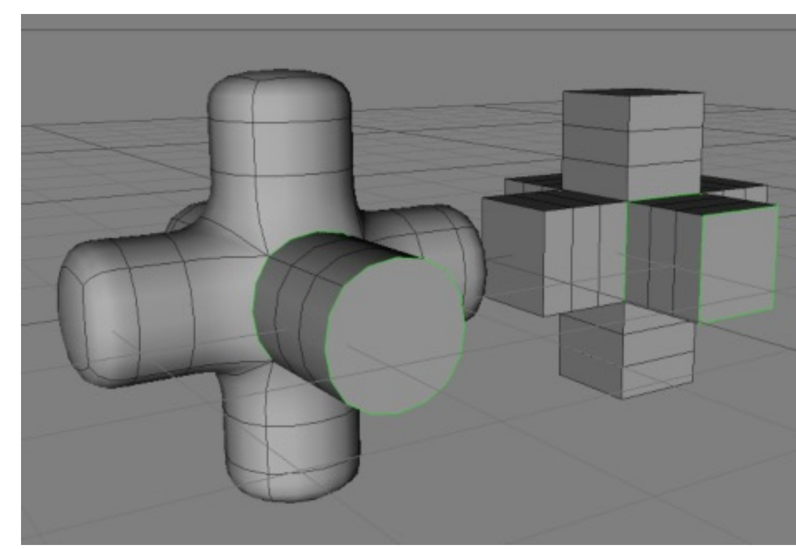
- After one subdivision, it’s a quad mesh
- New vertices are degree 4
- Quads are divided into 4 smaller quads
- Limit surface is a cubic B-spline patch
  - At least near vertices of degree 4
  - Other vertices are *extraordinary*
- We can edit mesh at each level

General Subdivisions

- *Many* ways of doing this
- All basically the same
  - Start with coarse version
  - Refine with extra points
  - Adjust point locations
  - End with *limit* surface



blendernewbies.com



madsie.com