Modeling

A Volkswagan Beetle becomes the subject of a 1970 simulation project. Ivan Sutherland(left) and assistants plot coordinates for digitizing the car.

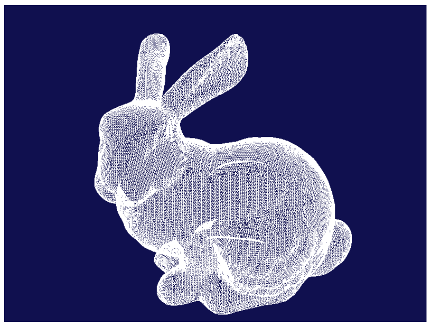
Simulating the Everyday World

Three broad areas:

- **Modeling (Geometric) = Shape**
- **Animation = Motion/Behavior**
- **Rendering = Appearance**

Geometric Modeling

1. How to represent 3d shapes
   - Polygonal meshes

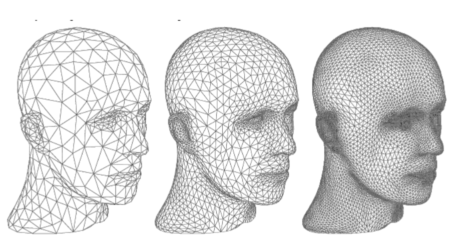


Stanford Bunny
69451 triangles

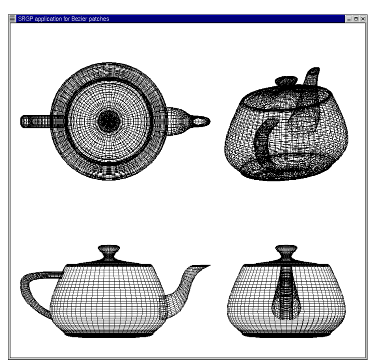
David, Digital Michelangelo Project
28,184,526 vertices, 56,230,343 triangles

Geometric Modeling

1. How to represent 3d shapes
   - Smooth surfaces
     - Bicubic spline surfaces
     - Subdivision surfaces



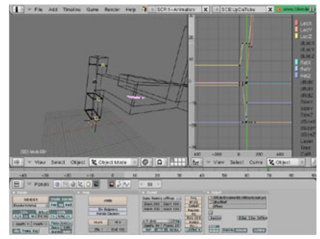
Caltech Head

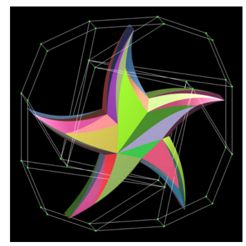


Utah Teapot

Geometric Modeling

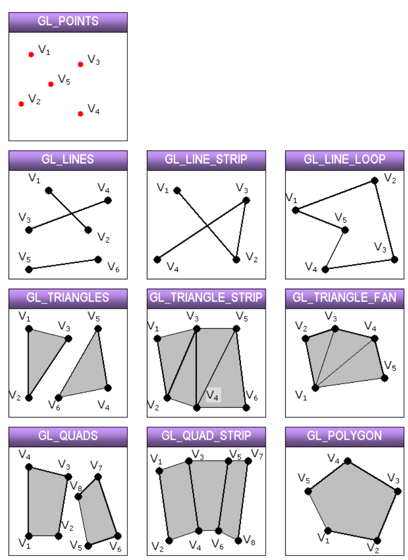
1. How to represent 3D shapes
2. How to create 3D shapes
   1. CAD tools
   2. Scanners
   3. Procedurally
3. How to manipulate 3D shapes
   1. Deform/skin/morph/animate
   2. Smooth/compress
   3. Set operations, …





OpenGL Primitives

```
glBegin( GL_* );
    glVertexfv(v1);
    glVertexfv(v2);
    glVertexfv(v3);
    glVertexfv(v4);
    glVertexfv(v5);
    glVertexfv(v6);
glEnd();
```



Explicit Coordinates

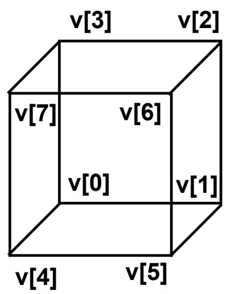
```
glBegin(GL_POLYGON);
    glVertex3f(-1.0,-1.0,0.0);
    glVertex3f(1.0,-1.0,0.0);
    glVertex3f(1.0,1.0,0.0);
    glVertex3f(-1.0,1.0,0.0);
glEnd();
```


Coordinates Stored in Arrays

```
float v1[3] = {-1.0,-1.0,0.0};
float v2[3] = { 1.0,-1.0,0.0};
float v3[3] = { 1.0, 1.0,0.0};
float v4[3] = {-1.0, 1.0,0.0};

glBegin(GL_POLYGON);
    glVertex3fv(v1);
    glVertex3fv(v2);
    glVertex3fv(v3);
    glVertex3fv(v4);
glEnd();
```

Points/Polygons

```
typedef float Point[3];

Point verts[8] = {
 {-1.,-1.,-1.},
 { 1.,-1.,-1.},
 { 1., 1.,-1.},
 {-1., 1.,-1.},
 {-1.,-1., 1.},
 { 1.,-1., 1.},
 { 1., 1., 1.},
 {-1., 1., 1.},
};
```




```
face(int a, int b, int c, int d) {
    glBegin(GL_POLYGON);
      glVertex3fv(verts[a]);
      glVertex3fv(verts[b]);
      glVertex3fv(verts[c]);
      glVertex3fv(verts[d]);
    glEnd();
}
// Note consistent ccw orientation!
cube() {
    face(0,3,2,1);
    face(2,3,7,6);
    face(0,4,7,3);
    face(1,2,6,5);
    face(4,5,6,7);
    face(0,1,5,4);
}
```


Points/Polygons

```
typedef float Point[3];
Point verts[8] = {
 {-1.,-1.,-1.},
 { 1.,-1.,-1.},
 { 1., 1.,-1.},
 {-1., 1.,-1.},
 {-1.,-1., 1.},
 { 1.,-1., 1.},
 { 1., 1., 1.},
 {-1., 1., 1.},
};
int polys[6][4] = {
 {0,3,2,1},
 {2,3,7,6},
 {0,4,7,3},
 {1,2,6,5),
 {4,5,6,7},
 {0,1,5,4}
}
```

```
face(int poly[4]) {
    glBegin(GL_POLYGON);
      glVertex3fv(poly[0]);
      glVertex3fv(poly[1]);
      glVertex3fv(poly[2]);
      glVertex3fv(poly[3]);
    glEnd();
}

cube() {
 for( int i = 0; i < n; i++ )
    face(polys[i]);
}
```

Comparison

Polygons

+ Simple
- Redundant information

Points/Polygons

+ Sharing vertices reduces memory requirements

Additional Topological Information

Requirements:

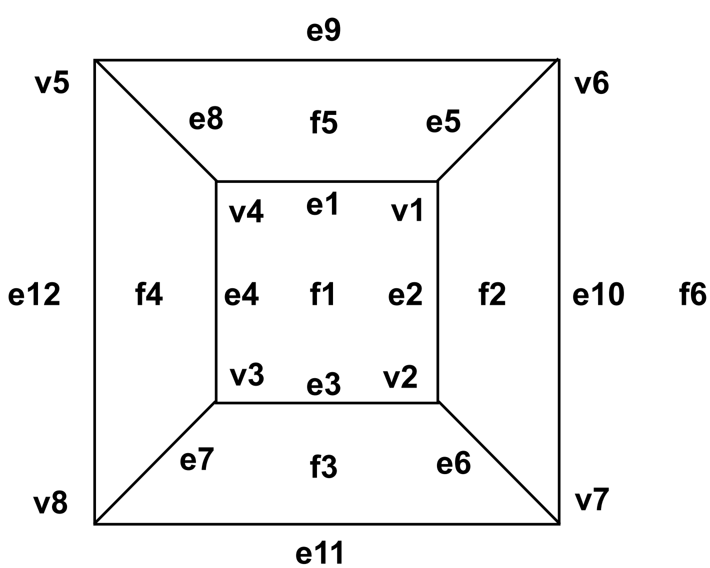
- Constant time access to neighbors
  e.g. surface normal calculation, subdivision
- Editing the geometry
  e.g. adding new vertices, new faces, etc.
- Maintain topological consistency

Topological data structures

Calculating Normals at Vertices

```
for f in mesh.faces():
  N = 0
  for v1, v2, v3 in f.consecutivevertices():
    N += cross(v2-v1,v3-v1)
  f.N = normalize(N)
for v in mesh.verts():
  N = 0
  for f in v.faces():
    N += f.N
  v.N = normalize(N)
```


Topology



Topological Properties

An edge connects exactly two faces

An edge connects exactly two vertices

A face consists of a ring of edges and vertices

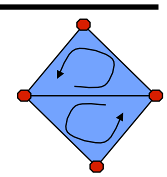
An vertex consists of a ring of edges and faces

Euler's formula #f - #e + #v = 2

(Check for a cube: 6 – 12 + 8 = 2)

Note: These properties define a 2D manifold

Triangle Meshes



Mesh has V vertices, E edges, and T triangles

2E = 3T

- There are 3 edges per triangle
- Each edge is part of 2 triangles

T=2V-4

- T – E + V = 2 => V = 3/2 T – T + 2 = T/2 + 2

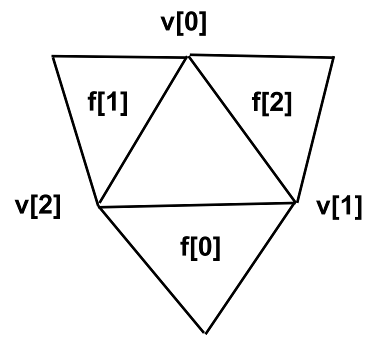
In the limit as the number of triangles increases

- There are twice as many triangles as vertices
- Each vertex has 6 triangles

However, except for an infinite surface, all vertices cannot have exactly 6 triangles

Triangle

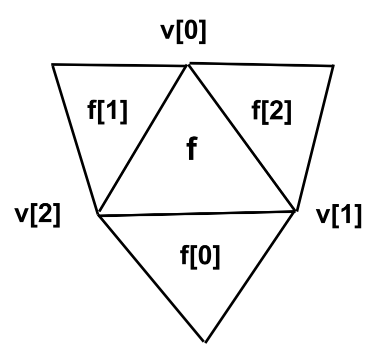
```
struct Vert {
   Point pt;
   Face *f;
}

struct Face {
   Vert *v[3];
   Face *f[3];
}
```



Finding CW Face Around a Vert

Find the next face clockwise around a vertex v given a face f

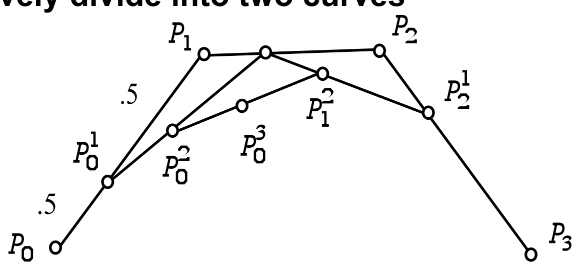
```
Face *fcwvf(Vert *v, Face *f)
{
  if( v == f->v[0] ) return f[1];
  if( v == f->v[1] ) return f[2];
  if( v == f->v[2] ) return f[0];
}
```



Subdivision Surfaces

Bezier Curves – Midpoint Subdivision

Recursively divide into two curves

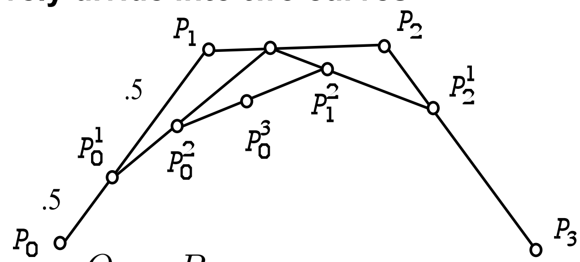


Left side

$$Q_0 = P_0$$
$$Q_1 = P_0^1$$
$$Q_2 = P_0^2$$
$$Q_3 = P_0^3$$

Bezier Curves – Midpoint Subdivision

Recursively divide into two curves



Left side

$$Q_0 = P_0$$

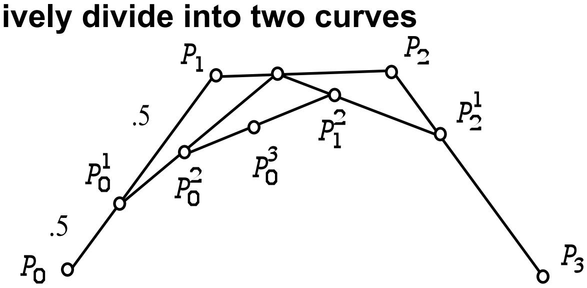
$$Q_1 = P_0^1 = \frac{1}{2}P_0 + \frac{1}{2}P_1$$

$$Q_2 = P_0^2 = \frac{1}{4}P_0 + \frac{1}{2}P_1 + \frac{1}{4}P_2$$

$$Q_3 = P_0^3 = \frac{1}{8}P_0 + \frac{1}{4}P_1 + \frac{1}{4}P_2 + \frac{1}{8}P_3$$

Bezier Curves – Midpoint Subdivision

Recursively divide into two curves



Right side

$$R_0 = P_0^3$$

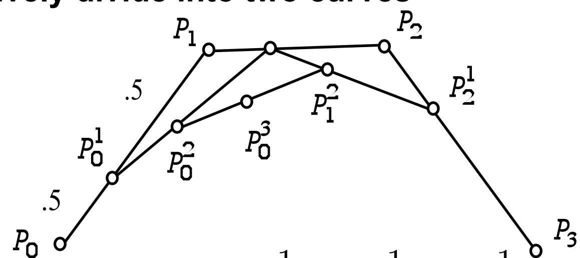
$$R_1 = P_1^2$$

$$R_2 = P_2^1$$

$$R_3 = P_3$$

Bezier Curves – Midpoint Subdivision

Recursively divide into two curves

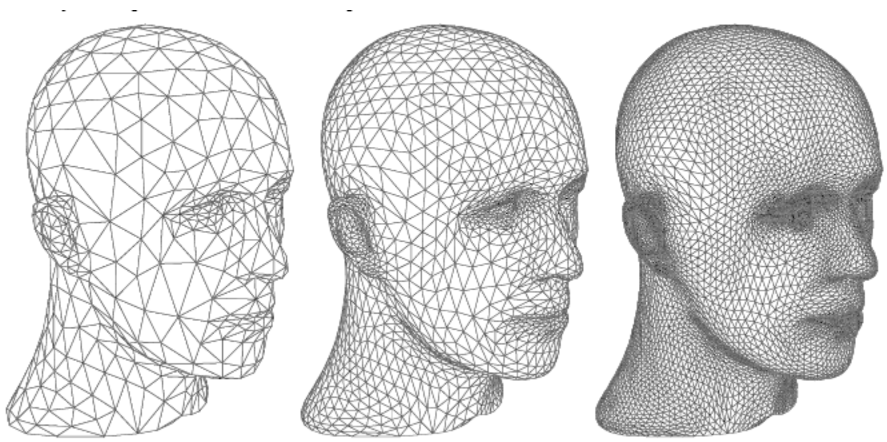


Right side

$$R_0 = P_0^3 = \frac{1}{8}P_0 + \frac{1}{4}P_1 + \frac{1}{4}P_2 + \frac{1}{8}P_3$$
$$R_1 = P_1^2 = \frac{1}{4}P_1 + \frac{1}{2}P_2 + \frac{1}{4}P_3$$
$$R_2 = P_2^1 = \frac{1}{2}P_2 + \frac{1}{2}P_3$$
$$R_3 = P_3$$

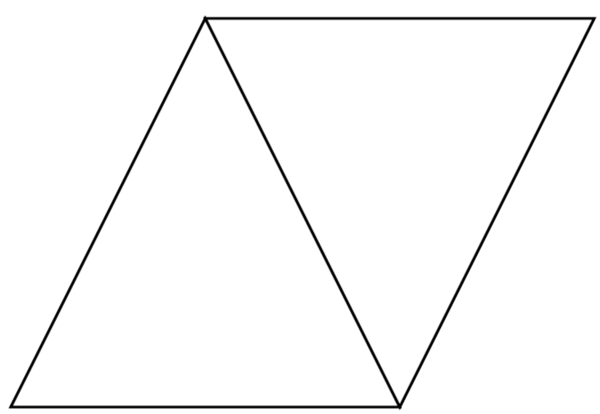
Loop Subdivision

Loop Subdivision Surface



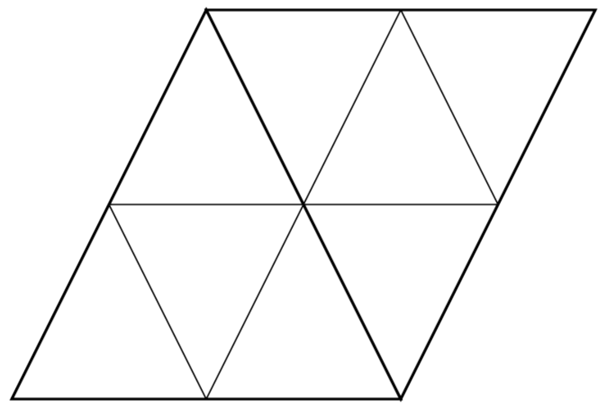
CS148 Lecture 18 Pat Hanrahan, Fall 2009

Triangle Mesh

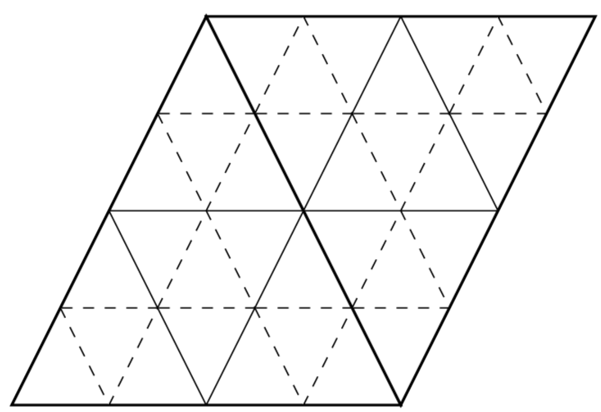


CS148 Lecture 18 Pat Hanrahan, Fall 2009

Subdivide Each Triangle into 4 Triangles



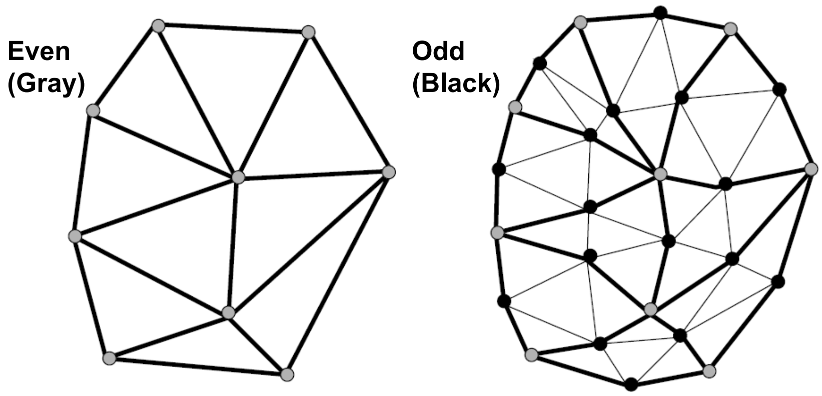
Subdivide Each Triangle into 4 Triangles



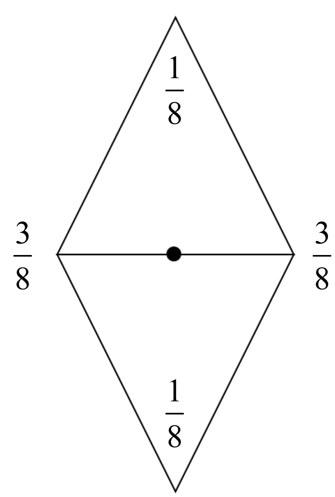
Two Types of Vertices

Even – Vertices that existed in previous mesh

Odd – Vertices created in new mesh

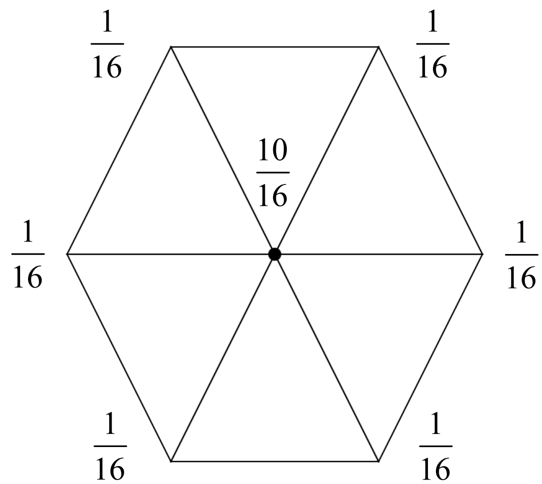


Loop Algorithm – Odd (New) Vertices



Loop Algorithm – Even (Old) Vertices

For degree 6 vertices



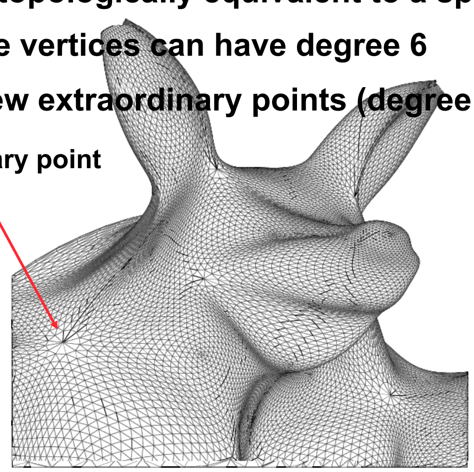
CS148 Lecture 18 Pat Hanrahan, Fall 2009

Semi-Regular Meshes

Most of the mesh has vertices with degree 6

If the mesh is topologically equivalent to a sphere, then not all the vertices can have degree 6

Must have a few extraordinary points (degree != 6)



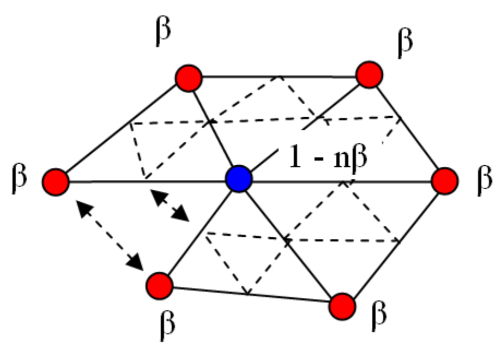
CS148 Lecture 18 Pat Hanrahan, Fall 2009

Weights at an Extraordinary Point

Challenge: find weights that generate a smooth surface

Want the surface normal to be continuous

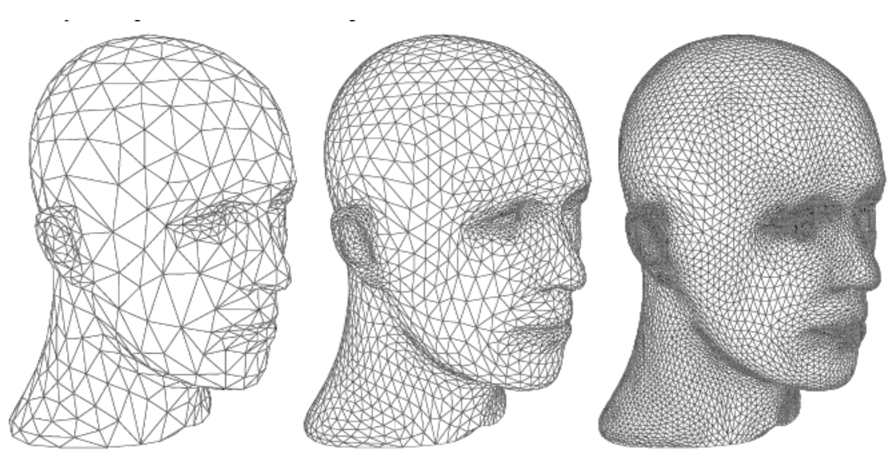
This is a hard math problem!



Warren weights

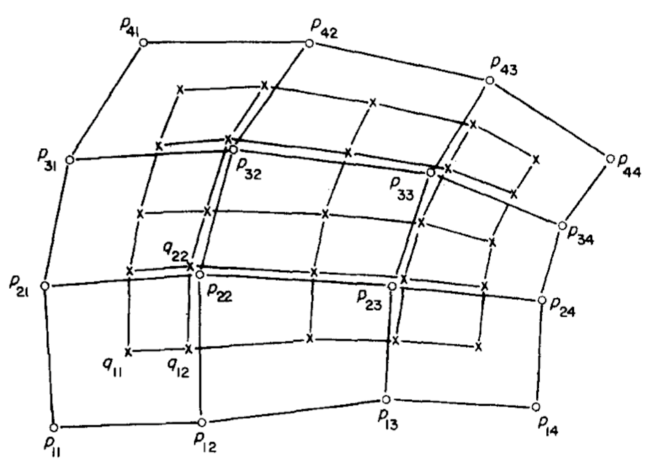
$$\beta = \begin{cases} \frac{3}{8n} & n > 3 \\ \frac{3}{16} & n = 3 \end{cases}$$

Loop Subdivision Surfaces



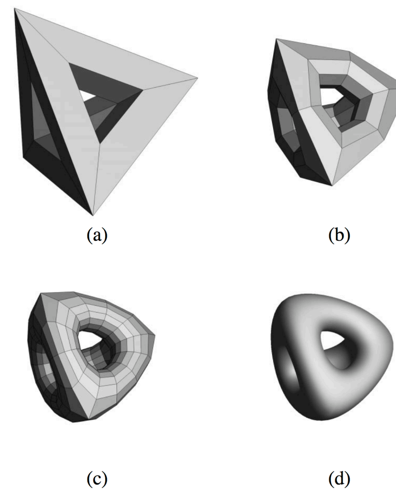
Catmull-Clark Subdivision

Subdividing Regular Quadrilateral Mesh



Catmull-Clark Subdivision

Allow faces with any number of vertices

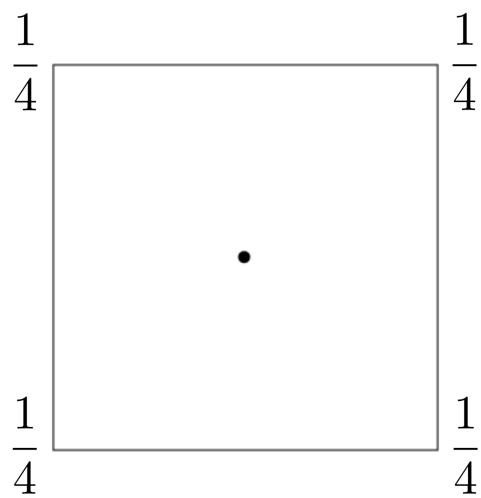


Catmull-Clark Subdivision

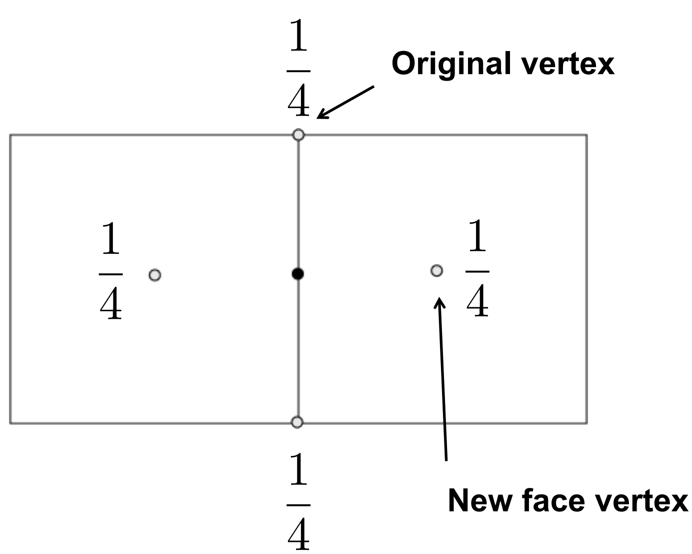
For an arbitrary mesh:

1. **Add a vertex at a face**
   Place at the average of the face vertices
2. **Add a vertex at an edge**
   Place at the average of the edge vertices and the two new face vertices
3. **Adjust original vertices**

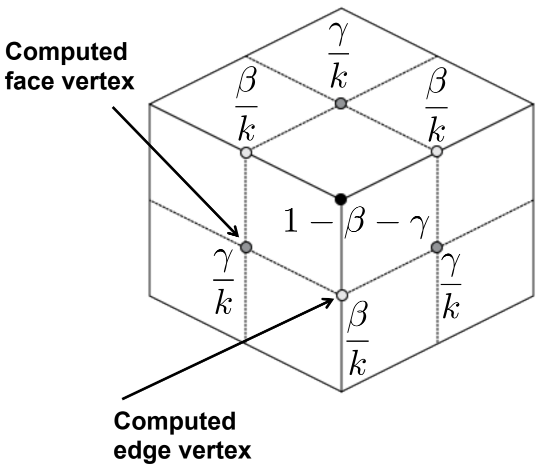
Quad Face Subdivision – Insert Vertex



Edge Subdivision – Insert Vertex



Update Vertex using Computed Vertices



New edge vertex

$$\beta = \frac{4}{k}$$

New face vertex

$$\gamma = -\frac{1}{k}$$

Old vert vertex

$$1 - \beta - \gamma$$

Assignment

Basic loop

- **Read in mesh**
- **Repeat**
  - **Recursively subdivide the mesh**
  - **Calculate new positions of vertices**
- **Compute normals and draw**

Challenges

- **Develop topological data structure that efficiently supports subdivision**

Things to Remember

Dense polygon mesh data structures

- **Polygons**
- **Points/Polygon**

Subdivision surfaces

- **Loop subdivision algorithm**
- **Catmull-Clark subdivision algorithm**