Modeling

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

Modeling 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
**Primitive API**

```
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);
    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);
    glVertex3f(-1.0,-1.0,0.0);
glEnd();
```
### Topology

$$\# \text{f} - \# \text{e} + \# \text{v} = 2$$

![Topology Diagram]

### Points/Polygons

```c
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();
}

cube() {
    // Note consistent ccw orientation!
    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] = {
{0,1,2,1},
{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]);
}

Representations

Polygons
+ Simple
- Redundant information

Points/Polygons
+ Share vertices (compress/consistency)

Additional topological information
+ Constant time access to neighbors
  More advanced algorithms such as
  surface normal calculation, subdivision ...
- Additional storage for topology
- More complicated 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)
```

Triangle Adjacency

```c
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];
}

Recursive Subdivision

Bezier curves
Subdivision surfaces
Fractals
Beziers 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

Right side

\[ R_0 = P_0^3 \]
\[ R_1 = P_1^2 \]
\[ R_2 = P_2^1 \]
\[ R_3 = P_3 \]
Subdivision Surfaces

Triangle Mesh
Triangle Mesh – Topological Subdivide

Triangle Mesh – Topological Subdivide
Triangle Mesh – Subdivide

Loop Algorithm - Edge
**Loop Algorithm - Vert**

\[
\begin{array}{ccc}
\frac{1}{16} & \frac{1}{16} & \frac{1}{16} \\
\frac{1}{16} & \frac{10}{16} & \frac{1}{16} \\
\frac{1}{16} & \frac{1}{16} & \frac{1}{16}
\end{array}
\]

**Semi-Regular Meshes**

Extraordinary Points
Loop Subdivision – Extraordinary Vertex

\[ \beta = \frac{1}{k} \left[ \frac{5}{8} - \left( \frac{3}{8} + \frac{1}{4} \cos \left( \frac{2\pi}{k} \right) \right)^2 \right] \]

\( k \) neighbors …

Catmull-Clark Subdivision

*Figure 1. Standard bicubic B-spline patch on a rectangular control-point mesh*
Face Subdivision – Insert Vertex

Edge Subdivision – Insert Vertex
Update Vertex using Computed Vertices

Computed face vertex

Computed edge vertex

Edge vertex
\[ \beta = \frac{4}{k} \]

Face vertex
\[ \gamma = -\frac{1}{k} \]

Vert vertex
\[ 1 - \beta - \gamma \]

Catmull-Clark Subdivision

(a)  

(b)  

(c)  

(d)
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
   Use formula on previous slide

Fractal Subdivision

\[ \Delta x \]
Fractal Subdivision

$\Delta x$

Fractal Subdivision

$\Delta y = \text{random()} \cdot \Delta x$

$\Delta x$
Fractal Subdivision: Height Field

Road to Point Reyes
Things to Remember

Common representations
- Dense polygon mesh data structures
  - Polygon
  - Points/Polygon (MeshArray)

Subdivision algorithms
- Subdivision surfaces
  - Loop subdivision algorithm
  - Catmull-Clark subdivision algorithm
- Fractal subdivision algorithm

Operations
- Computing normals from polygonal mesh