Computer Graphics

Prof. Feng Liu

Fall 2021

http://www.cs.pdx.edu/~fliu/courses/cs447/

11/22/2021

Last time

- Polygon Mesh and Modeling

Today

- Final Exam
  - December 06 (Monday) 17:30-19:00
  - To know list available on our class website
- Modeling Technologies

Modeling Techniques

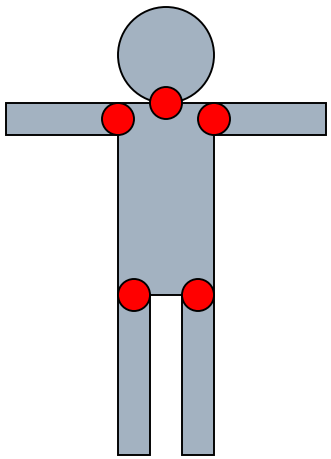
- Obtaining polygonal meshes
- Hierarchical modeling
- Instancing and Parametric Instancing
- Constructive Solid Geometry
- Sweep Objects
- Subdivision

Animating a character

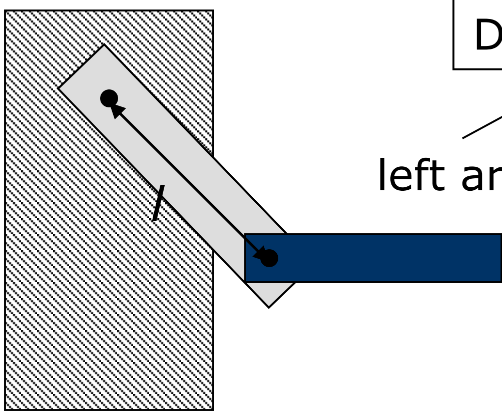
https://www.youtube.com/watch?v=TOMEtLJUyXY

Hierarchical Modeling

- Hierarchical model: Group of meshes related by a tree (or graph) structure
  - Properties of children are derived from their parents
  - Most useful for animating polygonal meshes
- Consider a walking (humanoid, classic) robot:
  - How would you move the robot around?
  - Does the entire robot move in the same way?
  - Does the position of one part of the robot depend on other parts?



Hierarchical Model Example



Move body
Draw body

left arm

Rotate about shoulder
Draw upper arm

Translate *(l,0,0)*
Rotate about origin
of lower arm
Draw lower arm

Important Point:

- Every node has its own local coordinate system.
- This makes specifying transformations much much easier.

Hierarchical Details

- Generally represented as a tree, with transformations and instances at any node
  - Can use a general graph, but resolving inheritance conflicts is a problem
- Rendered by traversing the tree, applying the transformations, and rendering the instances
- Particularly useful for animation
  - Human is a hierarchy of body, head, upper arm, lower arm, etc…
  - Animate by changing the transformations at the nodes
- Other things can be inherited (colors, surface properties)

OpenGL Support

- OpenGL defines **`glPushMatrix()`** and **`glPopMatrix()`**
  - Takes the current matrix and pushes it onto a stack, or pops the matrix off the top of the stack and makes it the current matrix
  - Note: Pushing does not change the current matrix
- Rendering a hierarchy (recursive):

```
RenderNode(tree)
      glPushMatrix()
            Apply node transformation
            Draw node contents
            RenderNode(children)
      glPopMatrix()
```

Instancing

- Sometimes you need many copies of the “same” object
  - Like chairs in a room
- Define one chair, the base or the prototype
- Create many *instances* (copies) of it, and apply a different transformation to each
- Appears in scene description languages (Renderman, Inventor) as “defining” a label for an object
- Advantages?

OpenGL Support

- OpenGL defines *display lists* for encapsulating commands that are executed frequently

```
list_id = glGenLists(1);
glNewList(list_id, GL_COMPILE);
glBegin(GL_TRIANGLES);
       draw some stuff
glEnd();
glEndList();
```

And later

```
glCallList(list_id);
```

More Display Lists

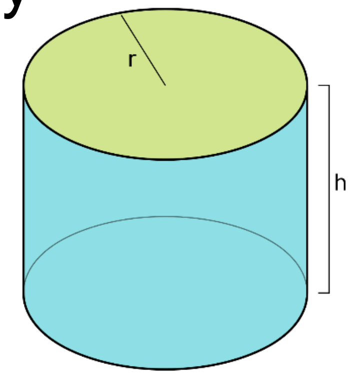
- Almost any command can go in a display list
  - Viewing transformation set-up
  - Lighting set-up
  - Surface property set-up
- But some things can't
  - Causes strange bugs – always check that a command can go in a display list
- The list can be:
  - `GL_COMPILE`: things don't get drawn, just stored
  - `GL_COMPILE_AND_EXECUTE`: things are drawn, and also stored

Display Lists Good/Bad

- You should use display lists when:
  - You do the same thing over and over again
  - The commands are supported
  - Nothing changes about the way you do it
- Advantages:
  - Can't be much slower than the original way
  - Can be much faster
- Disadvantages:
  - Can't use various commands that would offer other speedups
    - For example, can't use glVertexPointer()

Parametric Instancing

- Many things, called primitives, are conveniently described by a label and a few parameters
  - Cylinder: radius, height, does it have end-caps, …
  - Bolts: length, diameter, thread pitch, …
  - Other examples?



- This is a modeling format:
  - Provide software that knows how to draw the object given the parameters, or knows how to produce a polygonal mesh
  - How you manage the model depends on the rendering style
  - Can be an exact representation

https://en.wikipedia.org/wiki/Cylinder#/media/File:Circular_cylinder_rh.svg

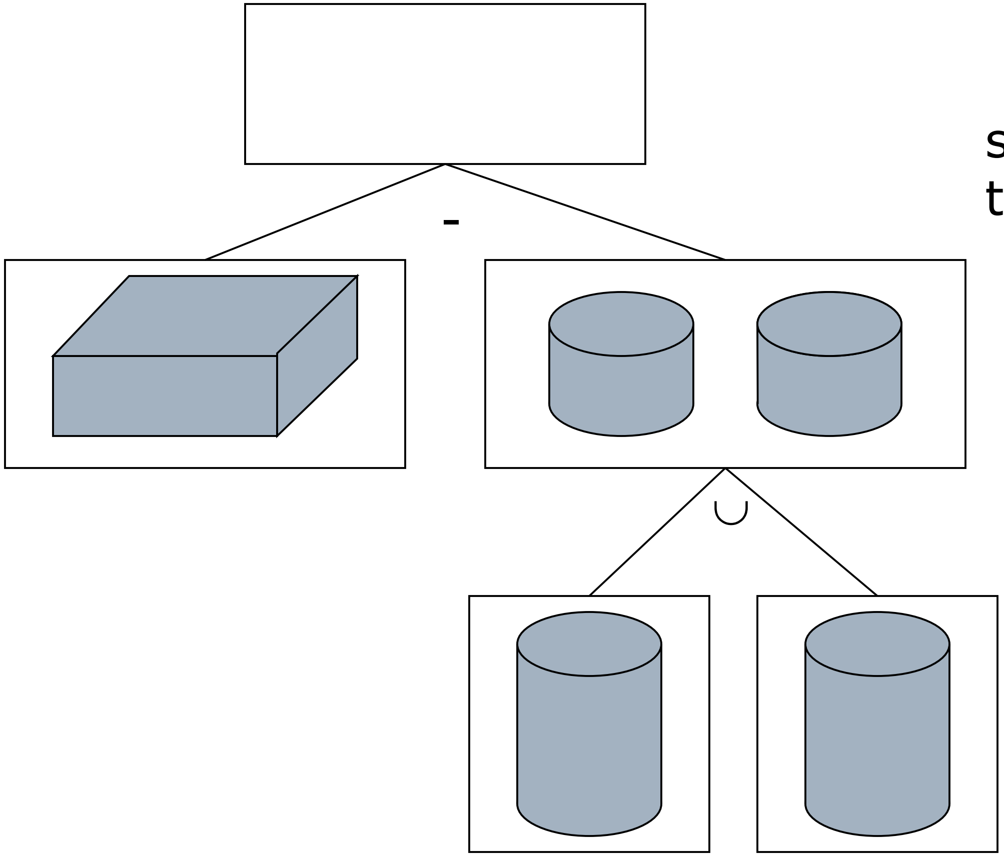
Rendering Instances

- Generally, provide a routine that takes the parameters and produces a polygonal representation
  - Conveniently brings parametric instancing into the rendering pipeline
  - May include texture maps, normal vectors, colors, etc
  - OpenGL utility library (glu) defines routines for cubes, cylinders, disks, and other common shapes
  - Renderman does similar things, so does POVray, …
- The procedure may be dynamic
  - For example, adjust the polygon resolution according to distance from the viewer

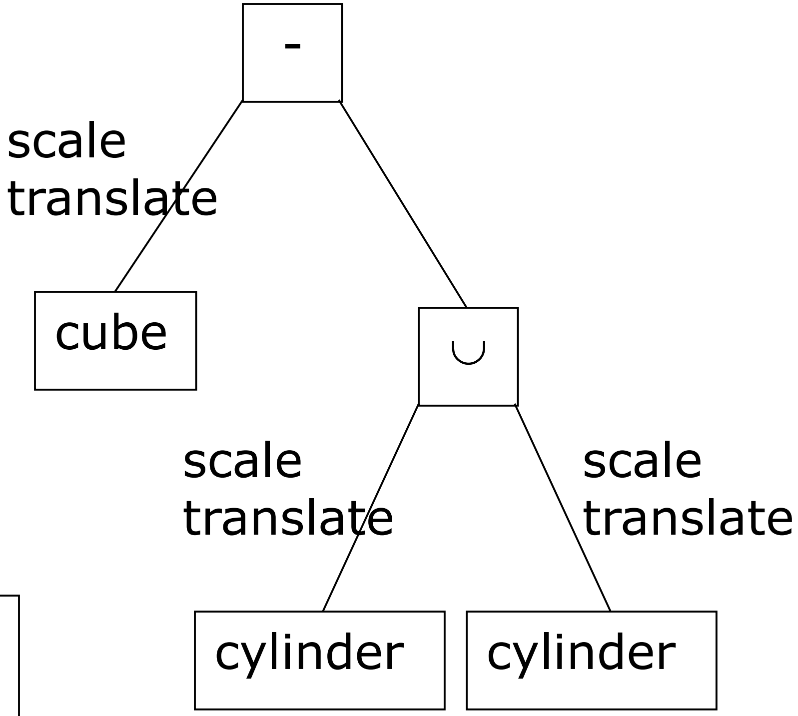
Constructive Solid Geometry (CSG)

- Based on a tree structure, like hierarchical modeling, but now:
  - The internal nodes are set operations: union, intersection or difference (sometimes complement)
  - The edges of the tree have transformations associated with them
  - The leaves contain only geometry
- Allows complex shapes with only a few primitives
  - Common primitives are cylinders, cubes, etc, or quadric surfaces
- Motivated by computer aided design and manufacture
  - *Difference* is like drilling or milling
  - A common format in CAD products

CSG Example



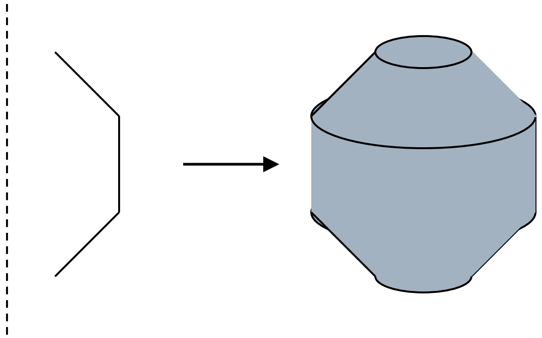
-
∪

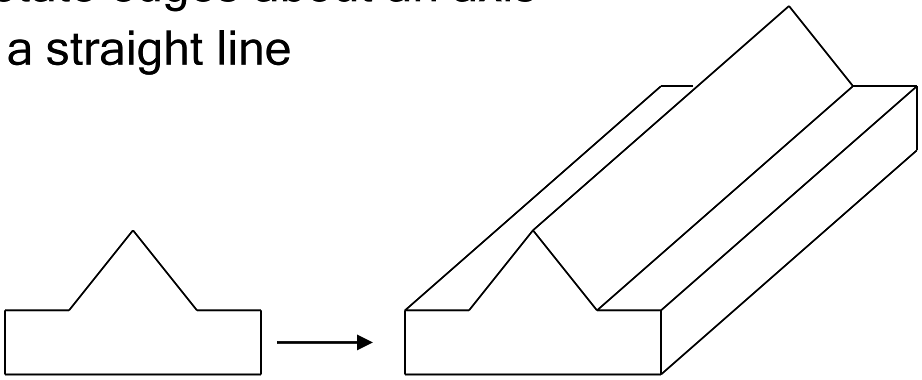


-
scale
translate
cube
∪
scale
translate
scale
translate
cylinder
cylinder

Sweep Objects

- Define a *polygon* by its edges
- Sweep it along a *path*
- The path taken by the edges form a surface - the sweep surface
- Special cases
  - Surface of revolution: Rotate edges about an axis
  - Extrusion: Sweep along a straight line

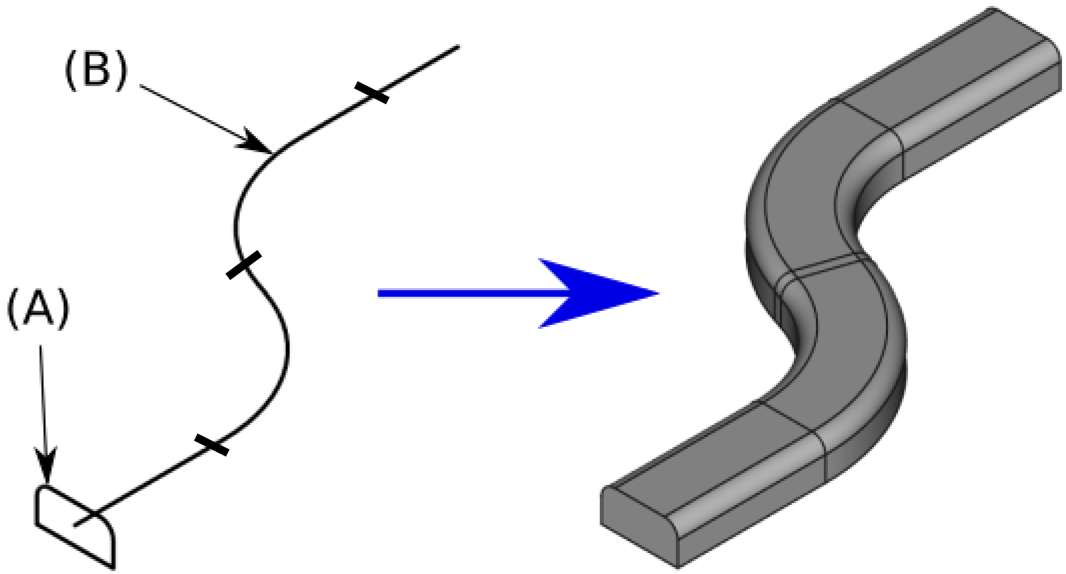




Rendering Sweeps

- Convert to polygons
  - Break path into short segments
  - Create a copy of the sweep polygon at each segment
  - Join the corresponding vertices between the polygons
  - May need things like end-caps on surfaces of revolution and extrusions
- Normals come from sweep polygon and path orientation
- Sweep polygon defines one texture parameter, sweep path defines the other

Rendering Sweeps



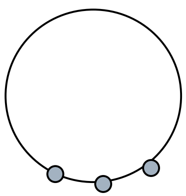
https://wiki.freecadweb.org/File:Part_Sweep_simple.png

A Circular Tube (A torus)

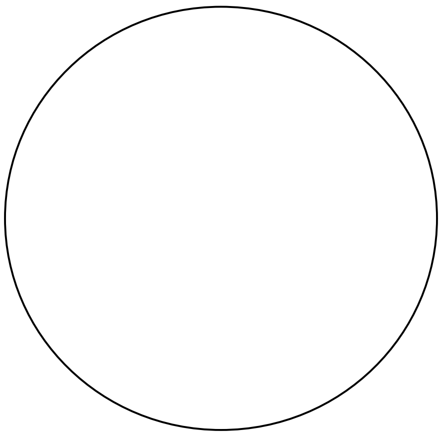
- What do we sweep, along what path?

A Circular Tube (A torus)

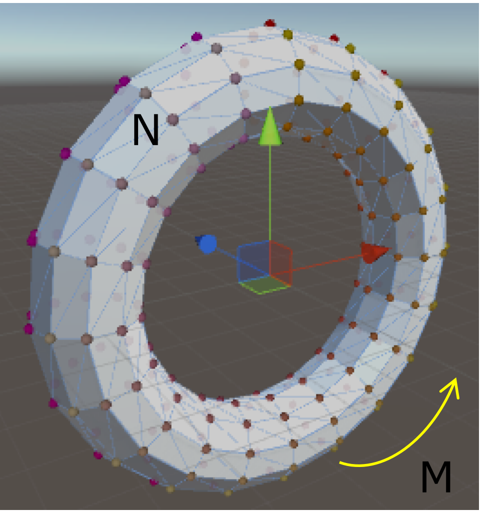
- What do we sweep, along what path?



Geometry



Path



A Circular Tube (A torus)

- What do we sweep, along what path?

```
Vector3 points[2][N];
int             start_i = 0;
int             end_i = 1;
for ( int i = 0 ; i < N ; i++ )
        points[start_i][i] = TorusPoint(M-1,i);
for ( int j = 0 ; j < M ; j++ ) {
        glBegin(GL_TRIANGLE_STRIP);
                for ( int i = 0 ; i < N ; i++ ) {
                        glVertex3fv(points[start_i][i]);
                        points[end_i][i] = TorusPoint(j, i);
                        glVertex3fv(points[end_i][i]);
                }
                glVertex3fv(points[start_i][0]); //close the loop
                glVertex3fv(points[end_i][0]);
        glEnd();
        int     temp = start_i; start_i = end_i; end_i = temp;
}
```

https://catlikecoding.com/unity/tutorials/swirly-pipe/

General Sweeps

- The path maybe any curve
- The polygon that is swept may be transformed as it is moved along the path
  - Scale, rotate with respect to path orientation, …
- One common way to specify is:
  - Give a poly-line (sequence of line segments) as the path
  - Give a poly-line as the shape to sweep
  - Give a transformation to apply at the vertex of each path segment
- Difficult to avoid self-intersection

Smooth versus General

- Polygon meshes are very general, but hard to model with
  - In a production context (film, game), creating a dense, accurate mesh requires lots of work
  - Biggest problem is smoothness
- We desire a way to “smooth out” a polygonal mesh
  - We can model at a coarse level, and automatically fill in the smooth parts
- *Subdivision surfaces* are part of the answer

Subdivision Schemes

- Basic idea: Start with something coarse, and refine it into smaller pieces, smoothing along the way
  - We will see how it can be used for modeling specific objects, and as a modeling scheme in itself
- In this lecture:
  - Subdivision for tessellating a sphere
  - Subdivision for fractal surfaces

Tessellating a Sphere

- Spheres are frequently parameterized in polar coordinates:

$$x = \cos\theta\cos\phi, \quad y = \sin\theta\cos\phi, \quad z = \sin\phi$$

$$0 \le \theta < 2\pi, \quad -\pi/2 \le \phi \le \pi/2$$

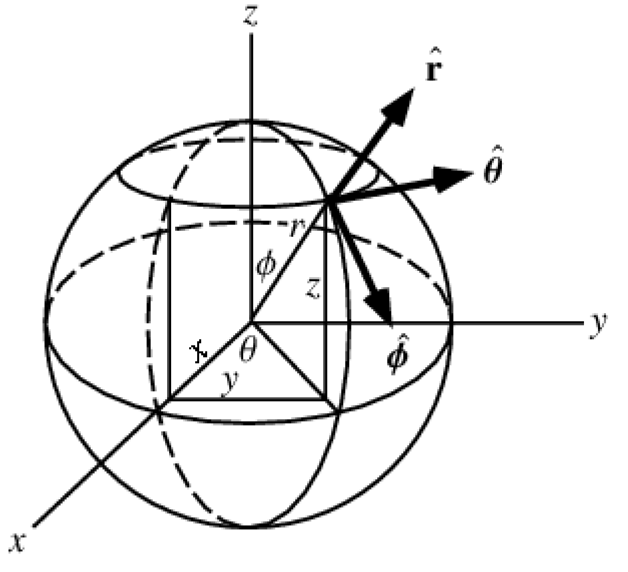


Image source: http://mathworld.wolfram.com/SphericalCoordinates.html

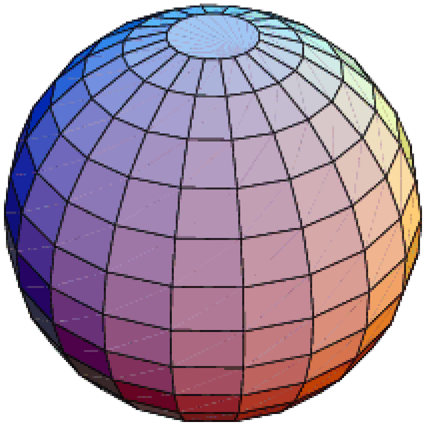
Tessellating a Sphere

- Spheres are frequently parameterized in polar coordinates:

$$x = \cos\theta\cos\phi, \quad y = \sin\theta\cos\phi, \quad z = \sin\phi$$

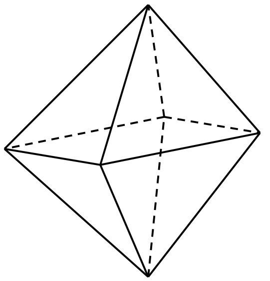
$$0 \le \theta < 2\pi, \quad -\pi/2 \le \phi \le \pi/2$$

- Tessellation: The process of approximating a surface with a polygon mesh
- One option for tessellating a sphere:
  - Step around and up the sphere in constant steps of θ and ϕ
  - Problem: Polygons are of wildly different sizes, and some vertices have very high degree

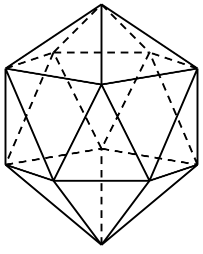


Subdivision Method

- Begin with a coarse approximation to the sphere, that uses only triangles
  - Two good candidates are platonic solids with triangular faces: Octahedron, Isosahedron
  - They have uniformly sized faces and uniform vertex degree
- Repeat the following process:
  - Insert a new vertex in the middle of each edge
  - Push the vertices out to the surface of the sphere
  - Break each triangular face into 4 triangles using the new vertices

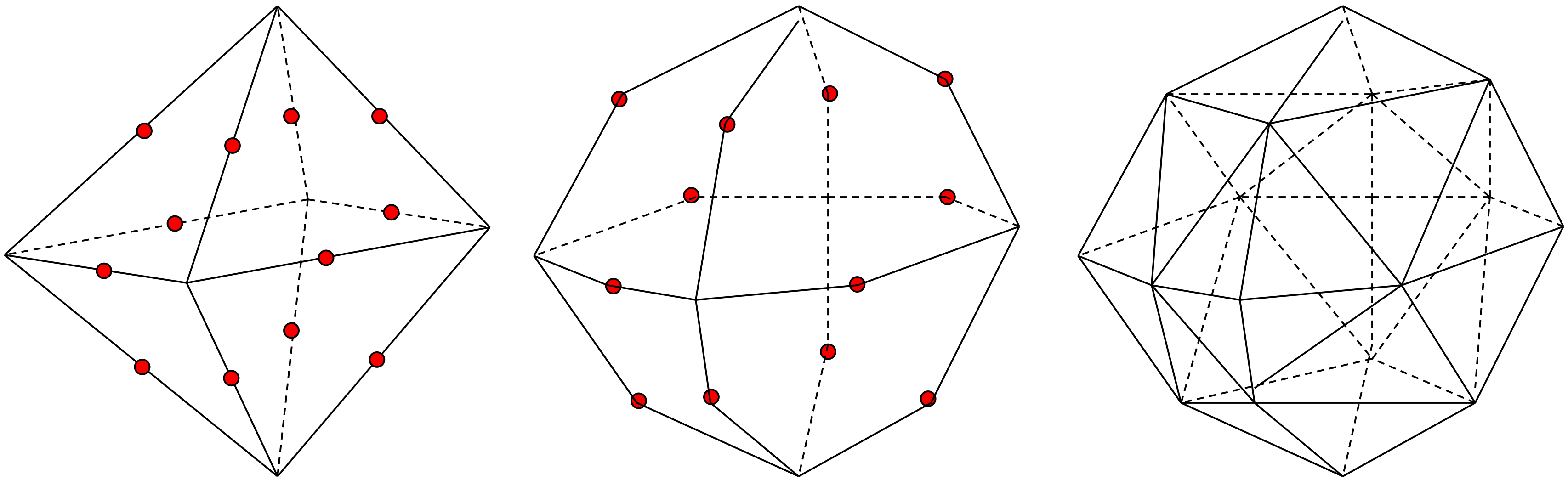


Octahedron

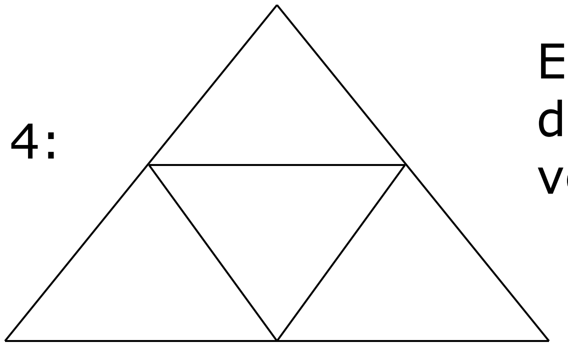


Isosahedron

The First Stage



Each face gets split into 4:



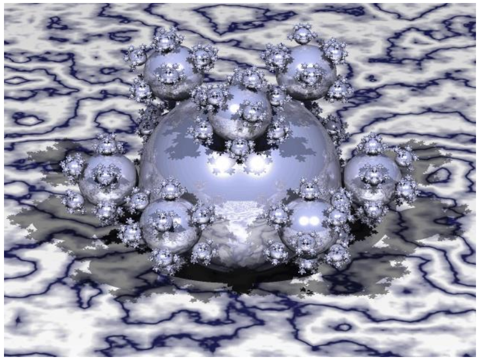
Each new vertex is degree 6, original vertices are degree 4

Sphere Subdivision Advantages

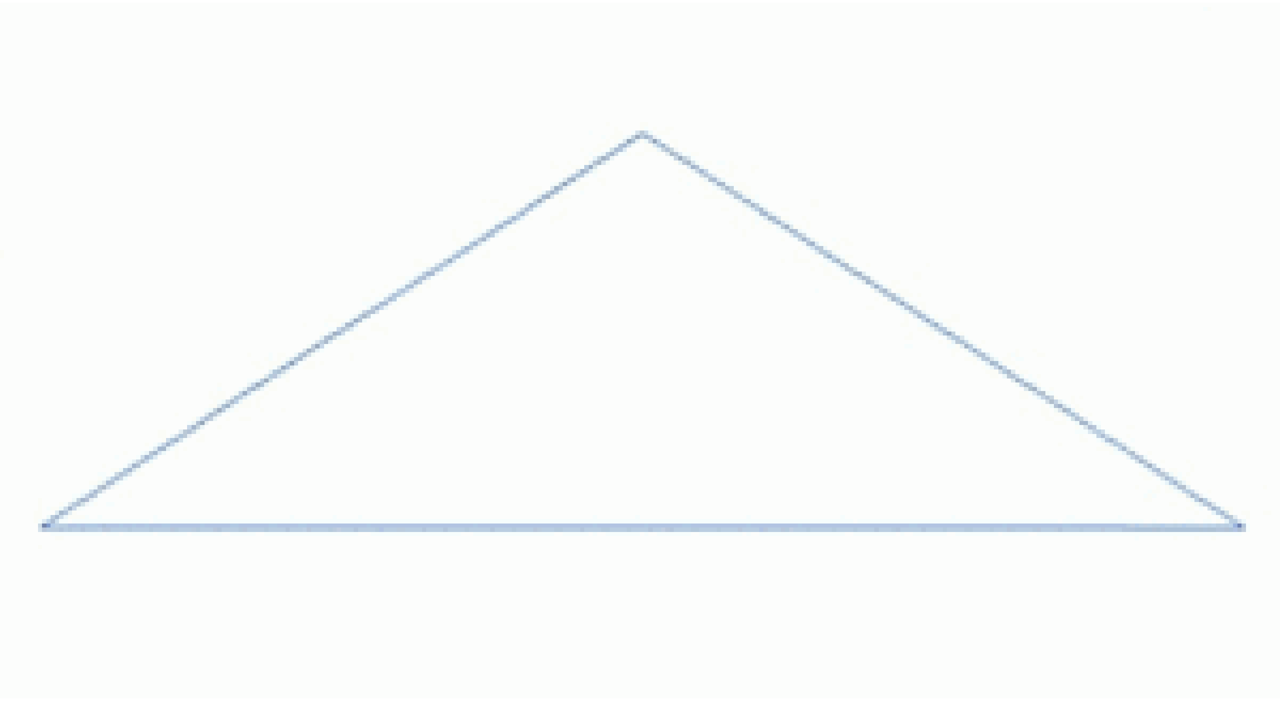
- All the triangles at any given level are the same size
  - Relies on the initial mesh having equal sized faces, and properties of the sphere
- The new vertices all have the same degree
  - Mesh is *regular (or uniform)* in newly generated areas
  - Makes it easier to analyze what happens to the surface
- The location and degree of existing vertices does not change

Fractal Surfaces

- Fractals are objects that show self similarity
  - The word is overloaded – it can also mean other things
- Landscapes and coastlines are considered fractal in nature
  - Mountains have hills on them that have rocks on them and so on
  - Continents have gulfs that have harbors that have bays and so on

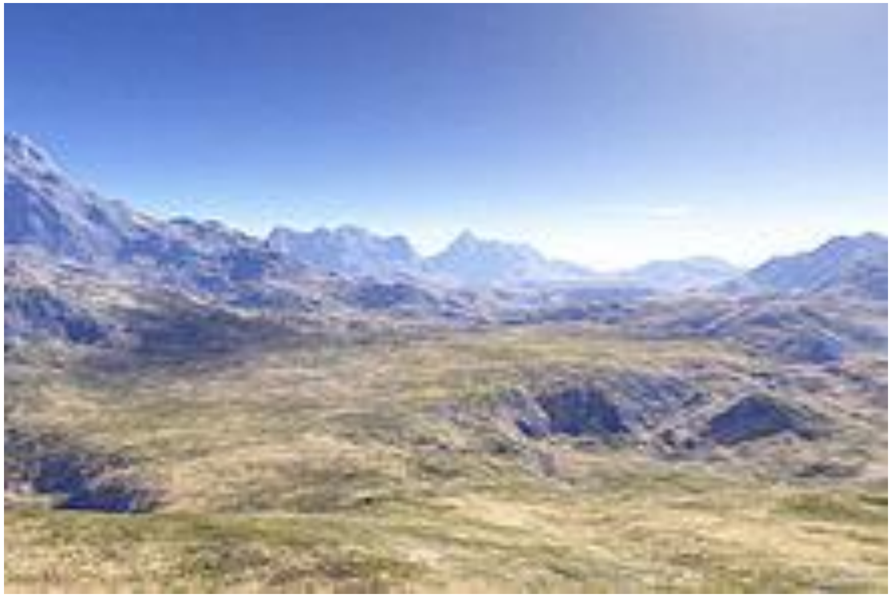


Fractal



https://en.wikipedia.org/wiki/Fractal_landscape

Fractal



https://en.wikipedia.org/wiki/Fractal_landscape

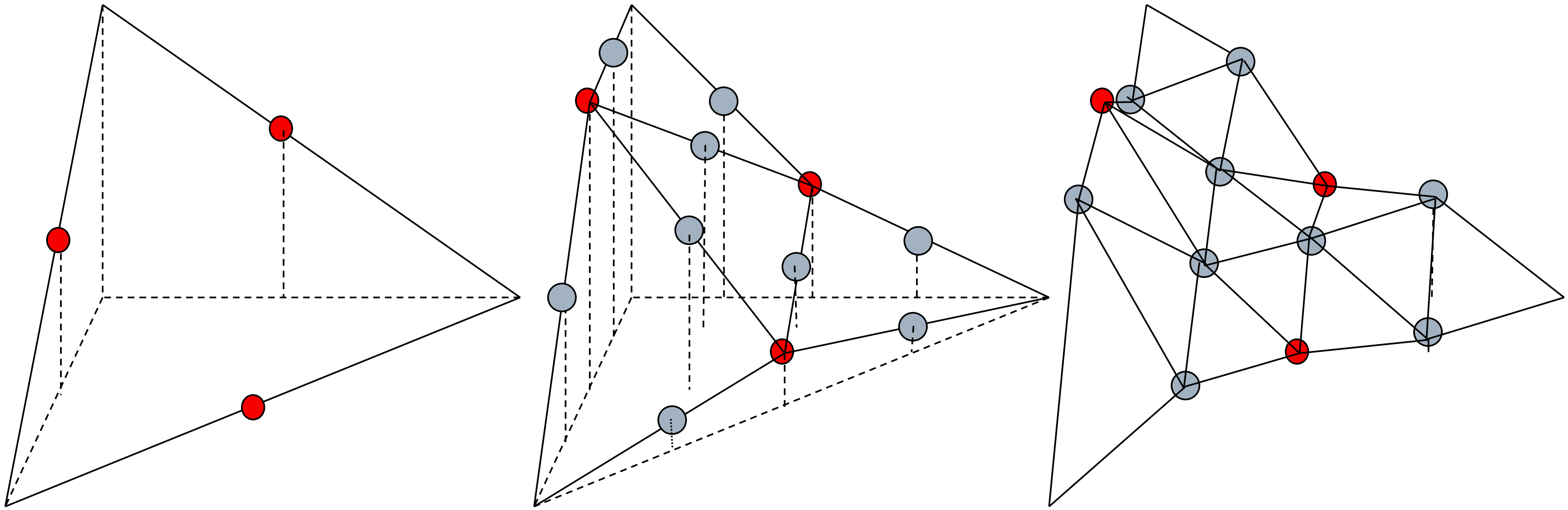
Fractal Surfaces

- Fractals are objects that show self similarity
  - The word is overloaded – it can also mean other things
- Landscapes and coastlines are considered fractal in nature
  - Mountains have hills on them that have rocks on them and so on
  - Continents have gulfs that have harbors that have bays and so on
- Subdivision is the natural way of building fractal surfaces
  - Start with coarse features, subdivide to finer features
  - Different types of fractals come from different subdivision schemes and different parameters to those schemes

Fractal Terrain (1)

- Start with a coarse mesh
  - Vertices on this mesh won’t move, so they can be used to set mountain peaks and valleys
  - Also defines the boundary
  - Mesh must not have dangling edges or vertices
    - Every edge and every vertex must be part of a face
- Also define an “up” direction
- Then repeatedly:
  - Add new vertices at the midpoint of each edge, and randomly push them up or down
  - Split each face into four, as for the sphere

Fractal Terrain Example



A mountainside

Fractal Terrain Details

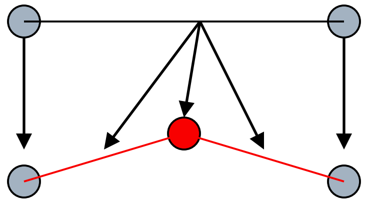
- There are options for choosing where to move the new vertices
  - Uniform random offset
  - Normally distributed offset – small motions more likely
  - Procedural rule – eg *Perlin noise*
- Reducing the offset of new points according to the subdivision level is essential
  - Define a scale, s, and a ratio, k, and at each level: $s_{i+1}=ks_i$
- Colors are frequently chosen based on “altitude”

Fractal Terrain Algorithm

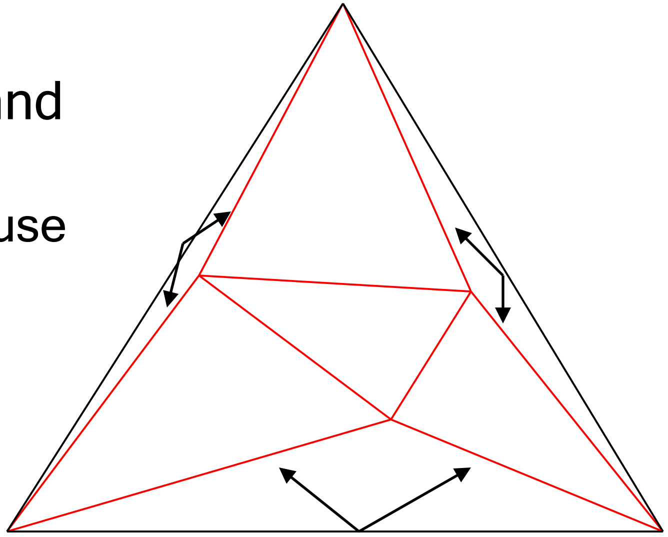
- The hard part is keeping track of all the indices and other data
- Same algorithm works for subdividing sphere

```
Split_One_Level(struct Mesh terrain)
      Copy old vertices
      for all edges
            Create and store new vertex
            Create and store new edges
      for all faces
            Create new edges interior to face
            Create new faces
      Replace old vertices, edges and faces
```

Subdivision Operations



- Split an edge, create a new vertex and two new edges
  - Each edge must be split exactly once
  - Need to know endpoints of edge to create new vertex
- Split a face, creating new edges and new faces based on the old edges and the old and new vertices
  - Require knowledge of which new edges to use
  - Require knowledge of new vertex locations



Data Structure Issues

- We must represent a polygon mesh so that the subdivision operations are easy to perform
- Questions influencing the data structures:
  - What information about faces, edges and vertices must we have, and how do we get at it?
  - Should we store edges explicitly?
  - Should faces know about their edges?

Next Time

- Implicit Surfaces