CS 418: Interactive Computer Graphics

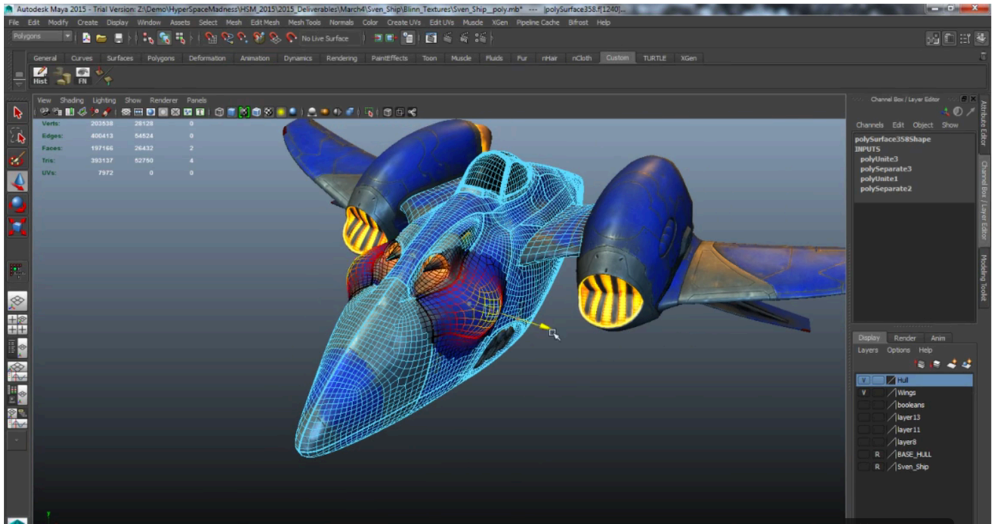
Simple Mesh Generation

Eric Shaffer

Some slides adapted from Angel and Shreiner:

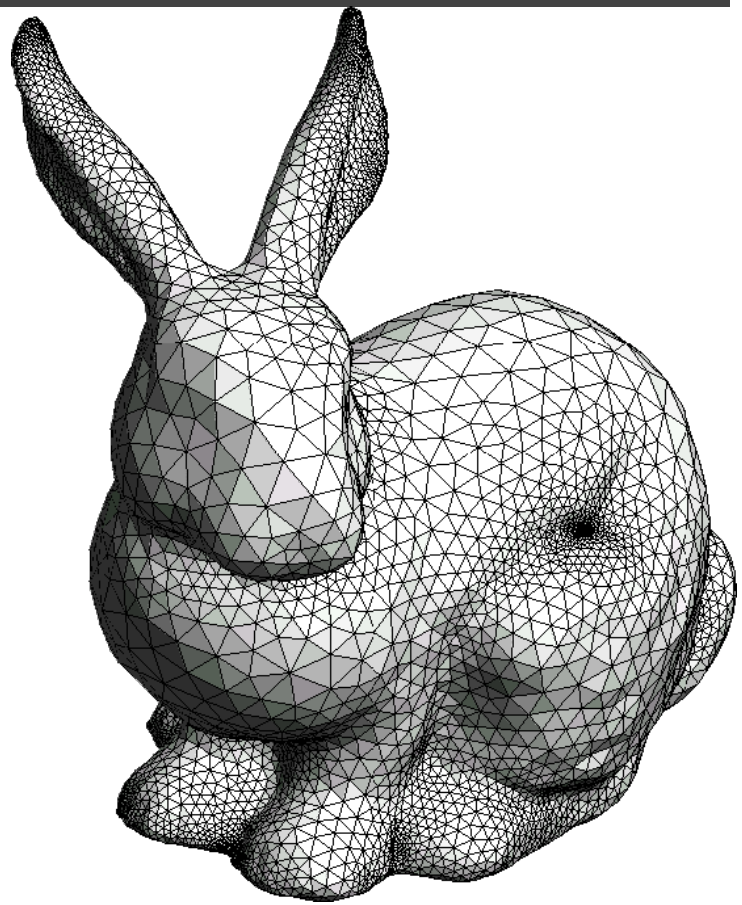
Geometric Modeling

- You may have noticed we have drawn only simple shapes
- Geometric modeling is not easy
- Even with sophisticated computational tools like Maya
  - Still labor intensive



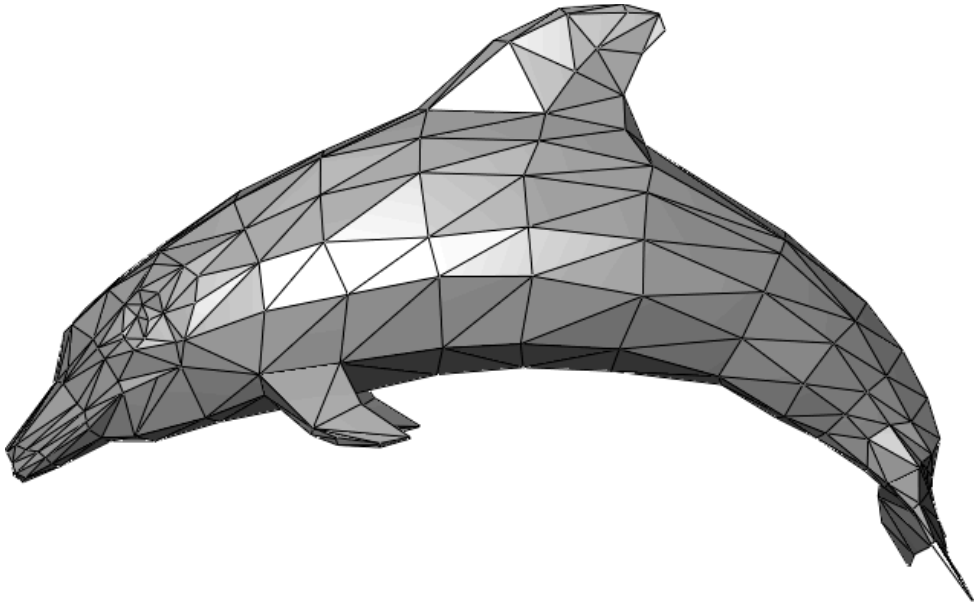
Geometric Modeling

- So how do you get geometric models for this course?
- Well…you can get models in files and implement a file reader
  - Some browsers prevent file reading for security reasons
  - We'll see how to work around that later
- OBJ file format is popular
  - for example stanford_bunny.obj



Data Structures for Geometry

- Most of the geometry we render will be in the form of triangle meshes
- There are lots of ways to store meshes
  - What three we have already seen?
- One other common one is an *indexed face mesh*



The OBJ File Format

- The OBJ file format is a popular storage format for meshes
- Developed by Wavefront Technologies…now part of AutoDesk
- Text files with .obj extension
- # indicates a comment

```
# List of geometric vertices, with (x,y,z[,w])
# coordinates, w is optional and defaults to 1.0.
  v 0.123 0.234 0.345 1.0
  v ...
  ...
# Polygonal face elements
  f 1 2 3
...
```

Indexed Face Mesh

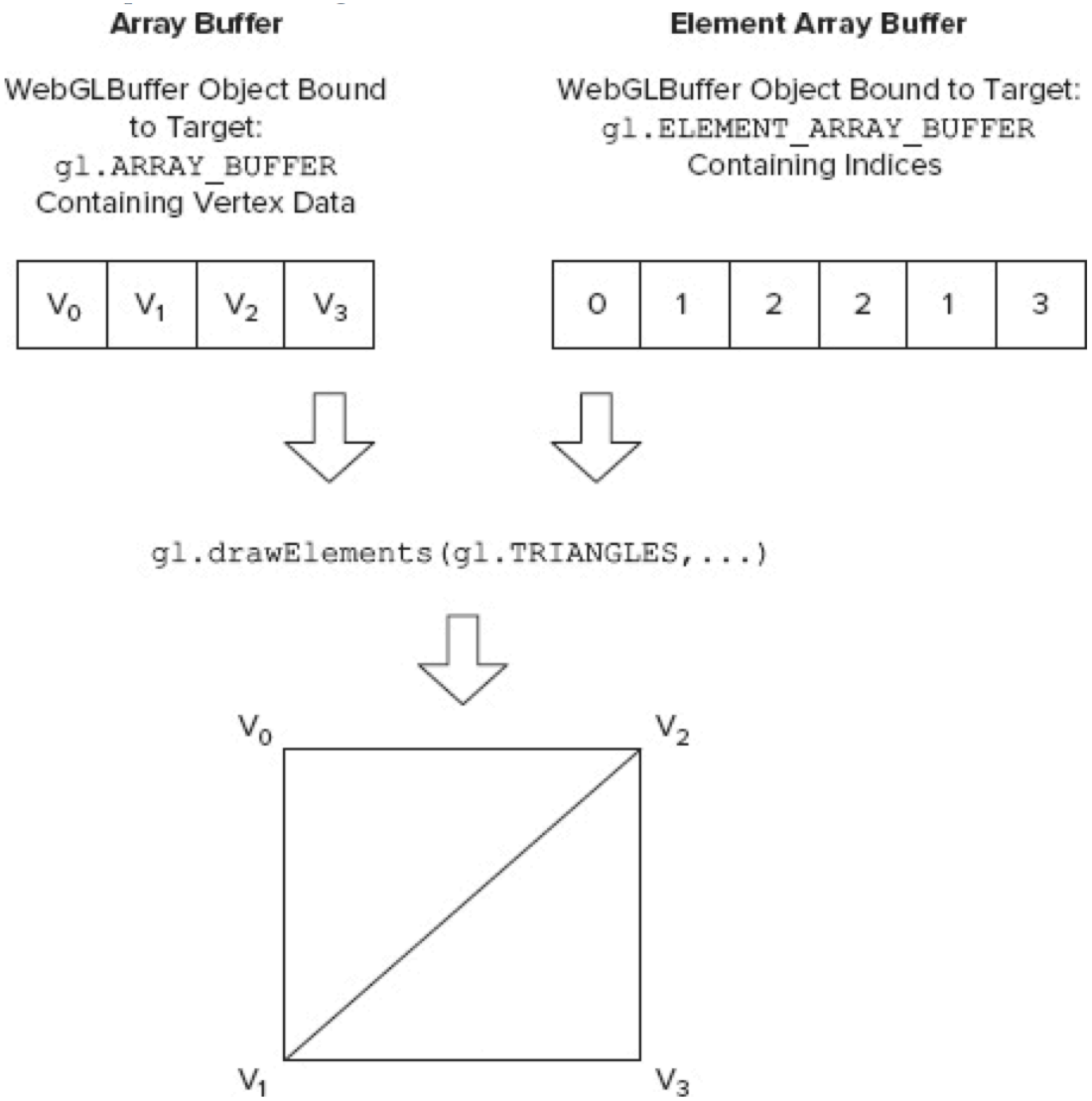
- Can be used for offline storage…a file format
- Or can be used as an internal data structure
- One block of data are the vertices
  - Each vertex is a set of 3 coordinates
  - Often referred to as the geometry of the mesh
- Another block of data is the set of triangles
  - Each triangle is set of 3 integers vertex IDs
  - The vertex IDs are indices into the vertex block
- What are some advantages of this representation?

gl.drawElements() method

- WebGL supports drawing indexed face meshes
- Simply need another buffer for the indexed faces

```
function draw() {
    ...
gl.bindBuffer(gl.ARRAY_BUFFER,meshVertexPositionBuffer);
gl.vertexAttribPointer(
       shaderProgram.vertexPositionAttribute,
       meshVertexPositionBuffer.itemSize, gl.FLOAT,
       false, 0, 0);
gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, meshIndexBuffer);
gl.drawElements(gl.TRIANGLES,
                  meshIndexBuffer.numberOfItems,
                  gl.UNSIGNED_SHORT, 0);
}
```

gl.drawElements() method



Setting up Indexed Drawing

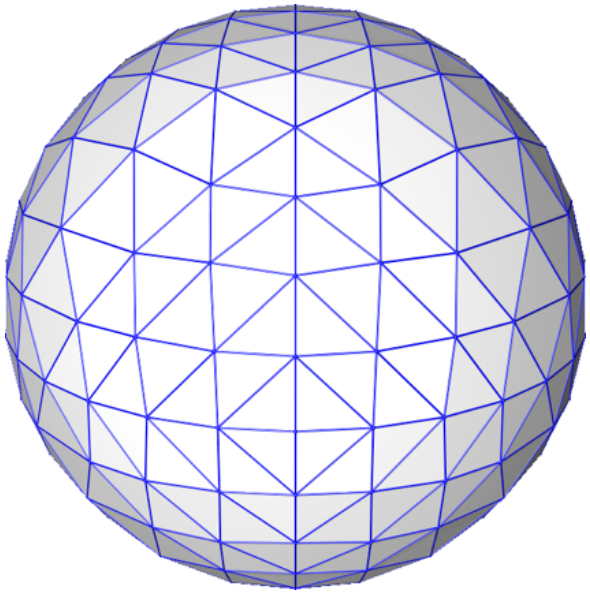
```
function setupBuffers() {

meshVertexPositionBuffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, meshVertexPositionBuffer);
var meshVertexPositions = [     1.0,  5.0,  0.0,  //v0
                                0.0,  5.0,  0.0, //v1
                                  ...    ];
gl.bufferData(gl.ARRAY_BUFFER, new
              Float32Array(meshVertexPositions),
              gl.STATIC_DRAW);
meshVertexPositionBuffer.itemSize = 3;
meshVertexPositionBuffer.numberOfItems = 36;
gl.enableVertexAttribArray(
              shaderProgram.vertexPositionAttribute);
```

Setting up Indexed Drawing

```
meshIndexBuffer = gl.createBuffer();
gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER,
              meshIndexBuffer);
var meshIndex = [     0, 1, 2,
                      2, 1, 3,
                      2, 3, 4,
                                …];
gl.bufferData(gl.ELEMENT_ARRAY_BUFFER,
              new Uint16Array(meshIndex),
              gl.STATIC_DRAW);
meshIndexBuffer.itemSize = 1;
meshIndexBuffer.numberOfItems = 150;
}
```

Geometric Modeling

- You can type the geometry in by hand
  - Hard code it into the .js file
  - Useful for testing
  - Obviously not scalable
- You can procedurally generate geometry
  - Write code to produce a bunch of triangles
  - We'll look at some simple methods today



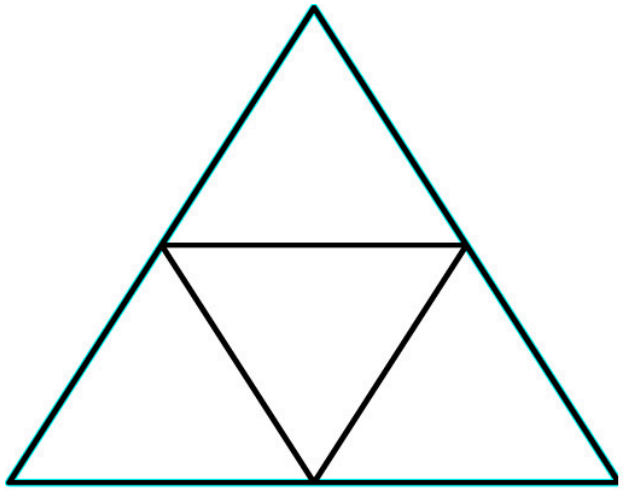
Generating a Tessellated Quad

- Recursion (or iteration) can be used to generate refined geometry
  - Meaning lots of triangles…
- How can we generate a tessellated plane recursively?

```
function planeFromSubdivision(n, minX,maxX,minY,maxY, vertexArray)
{
    var numT=0;
    var va = vec4.fromValues(minX,minY,0,0);
    var vb = vec4.fromValues(maxX,minY,0,0);
    var vc = vec4.fromValues(maxX,maxY,0,0);
    var vd = vec4.fromValues(minX,maxY,0,0);

    numT+=divideTriangle(va,vb,vd,n, vertexArray);
    numT+=divideTriangle(vb,vc,vd,n, vertexArray);
    return numT;

}
```

Subdividing a Triangle

- Find the midpoints of each edge
- Create 4 triangles from the original triangle
  - There are other ways you could subdivide
  - This method generates equilateral triangles if you start out with an equilateral



Subdividing a Triangle

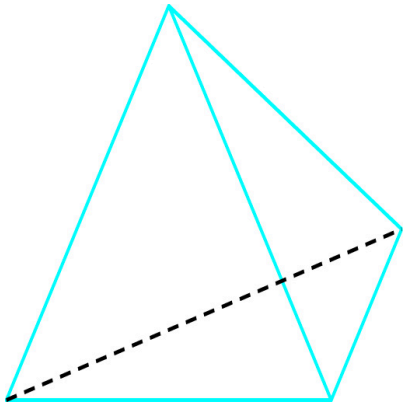
```
function divideTriangle(a,b,c,numSubDivs, vertexArray){
    if (numSubDivs>0){
        var numT=0;
        var ab =  vec4.create(); vec4.lerp(ab,a,b,0.5);
        var ac =  vec4.create(); vec4.lerp(ac,a,c,0.5);
        var bc =  vec4.create(); vec4.lerp(bc,b,c,0.5);

        numT+=divideTriangle(a,ab,ac,numSubDivs-1, vertexArray);
        numT+=divideTriangle(ab,b,bc,numSubDivs-1, vertexArray);
        numT+=divideTriangle(bc,c,ac,numSubDivs-1, vertexArray);
        numT+=divideTriangle(ab,bc,ac,numSubDivs-1, vertexArray);
        return numT;
    }
    else …
```

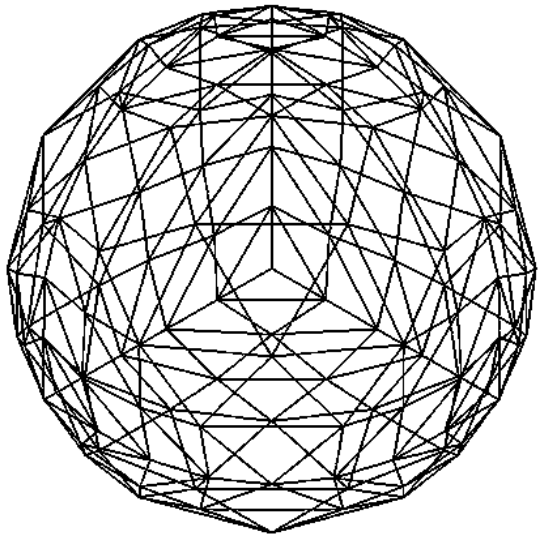
Subdividing a Triangle

```
else
    {
        // Add 3 vertices to the array
        pushVertex(a,vertexArray);
        pushVertex(b,vertexArray);
        pushVertex(c,vertexArray);
        return 1;

    }
}

function pushVertex(v, vArray)
{
 for(i=0;i<3;i++)
 {
     vArray.push(v[i]);
 }
}
```

Generating a Sphere

- Start with a tetrahedron centered on the origin
  - Vertices at a distance of 1 from the origin
- Recursively subdivide the triangular faces
  - Normalize the new vertices that get introduced





Generating a Sphere

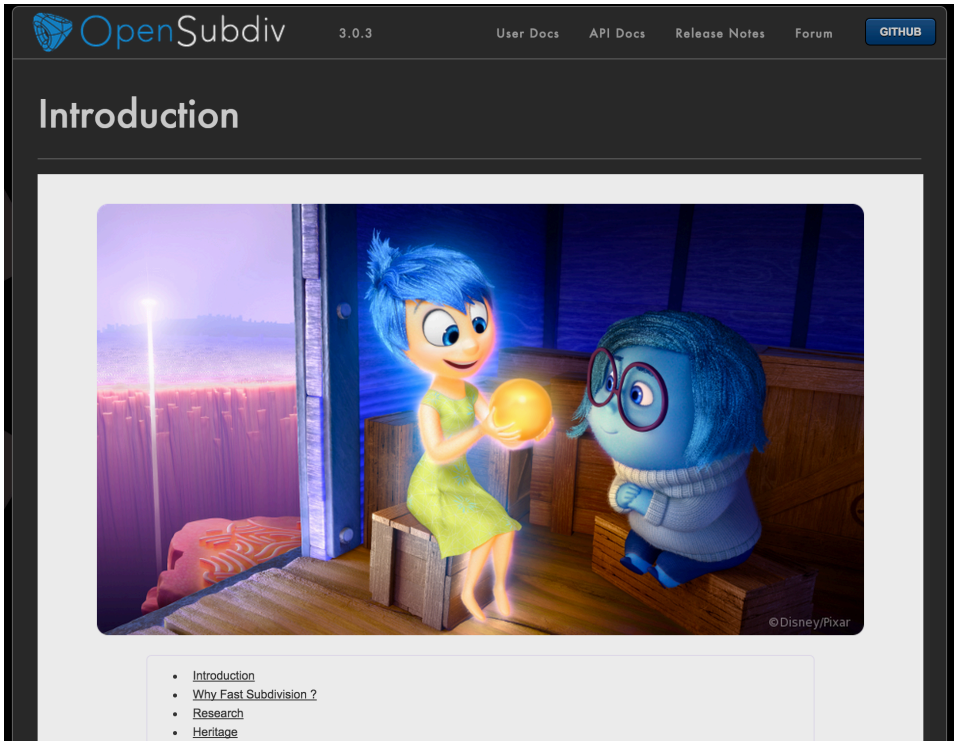
```
  function sphereFromSubdivision(numSubDivs, vertexArray, normalArray)
{
    var numT=0;
    var a = vec4.fromValues(0.0,0.0,-1.0,0);
    var b = vec4.fromValues(0.0,0.942809,0.333333,0);
    var c = vec4.fromValues(-0.816497,-0.471405,0.333333,0);
    var d = vec4.fromValues(0.816497,-0.471405,0.333333,0);

    numT+=sphDivideTriangle(a,b,c,numSubDivs, vertexArray, normalArray);
    numT+=sphDivideTriangle(d,c,b,numSubDivs, vertexArray, normalArray);
    numT+=sphDivideTriangle(a,d,b,numSubDivs, vertexArray, normalArray);
    numT+=sphDivideTriangle(a,c,d,numSubDivs, vertexArray, normalArray);
    return numT;
}
```

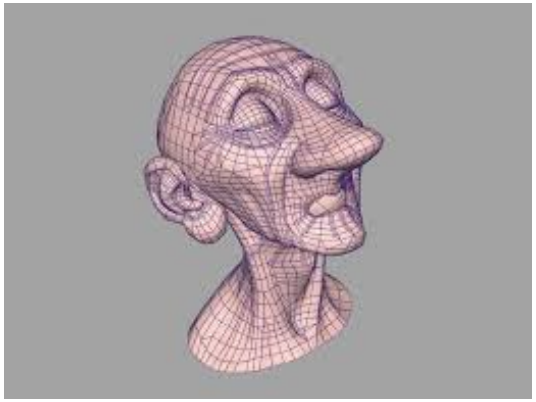
Generating a Sphere

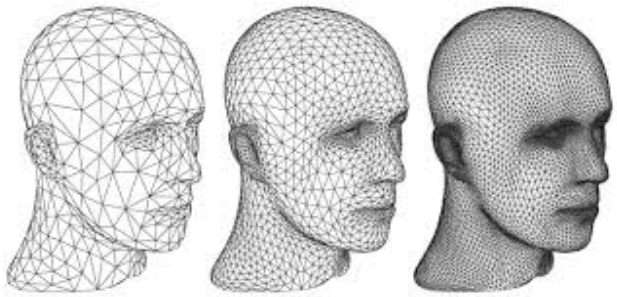
```
 function sphDivideTriangle(a,b,c,numSubDivs, vertexArray,normalArray)
{
    if (numSubDivs>0)
    {
        var numT=0;
        var ab =  vec4.create();vec4.lerp(ab,a,b,0.5);vec4.normalize(ab,ab);
        var ac =  vec4.create();vec4.lerp(ac,a,c,0.5);vec4.normalize(ac,ac);
        var bc =  vec4.create();vec4.lerp(bc,b,c,0.5);vec4.normalize(bc,bc);

        numT+=sphDivideTriangle(a,ab,ac,numSubDivs-1, vertexArray, normalArray);
        numT+=sphDivideTriangle(ab,b,bc,numSubDivs-1, vertexArray, normalArray);
        numT+=sphDivideTriangle(bc,c,ac,numSubDivs-1, vertexArray, normalArray);
        numT+=sphDivideTriangle(ab,bc,ac,numSubDivs-1, vertexArray, normalArray);
        return numT;
    }
    else …
```

Subdivision Surfaces

This may seem like a very simple idea, but subdivision is used in production modeling tools





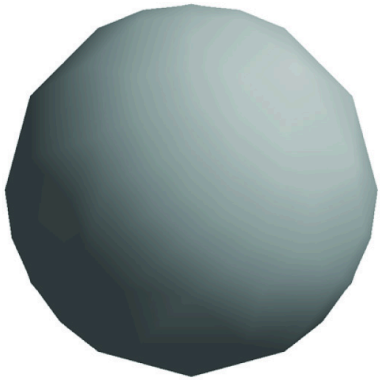


Generating Normals

- For the plane, what were the normals?
- For the sphere, compute the normal from the vertex
  - N = V-(0,0,0) …which is the same as V
- In general, computing normals from a triangle soup isn't easy
  - Easier using an indexed representation

Sphere with Phong Shading

Flat Shading: Each triangle drawn using same normal at each vertex



Using true normals, different normal at each vertex

From original paper on the Phong Model

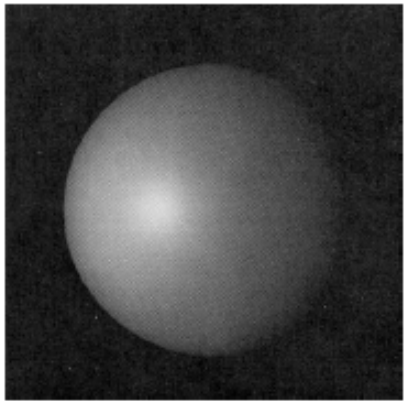


Fig. 10(a). A sphere displayed with the improved shading.

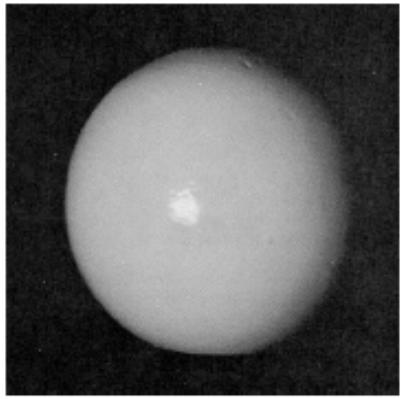


Fig. 10(b). A photograph of a real sphere.

Bui Tuong Phong

- December 14, 1942 – July 1975
- Born in Hanoi
- Earned his PhD in 2 years at the University of Utah (1973)
  - Worked with Professor Ivan Sutherland
  - Dissertation work was the Phong reflectance model
  - Also produced model and realistic image of a VW bug

Graphics and Image Processing
W. Newman Editor

Illumination for Computer Generated Pictures

Bui Tuong Phong
University of Utah

Fig. 9. Improved shading, applied to the example of Figure 2.