3D Models

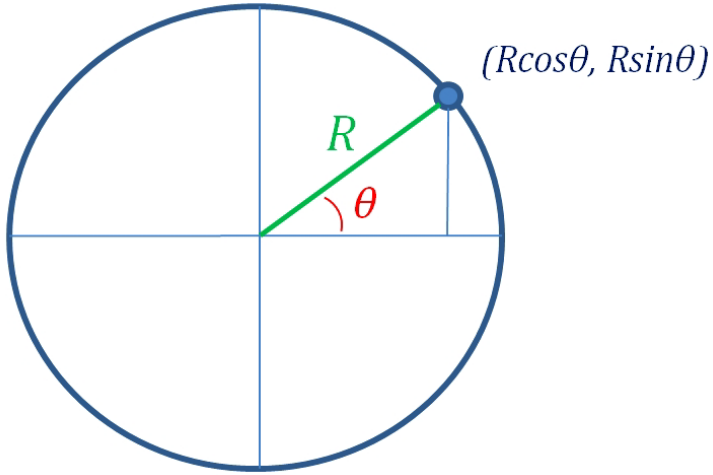
Prof. George Wolberg
Dept. of Computer Science
City College of New York

3D Models

- Build models procedurally
  - Use mathematical definitions
    - Spheres, cones, pyramids
- Load models produced externally
  - Use 3D modeling tools such as Maya or Blender

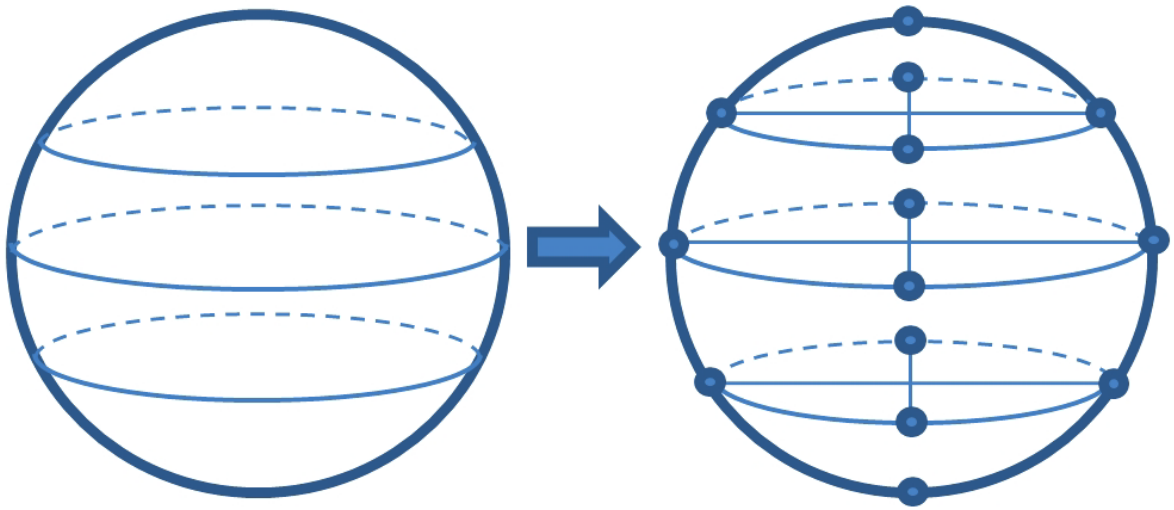
Procedural Modeling

Example: use geometry of circle to algorithmically build a sphere model



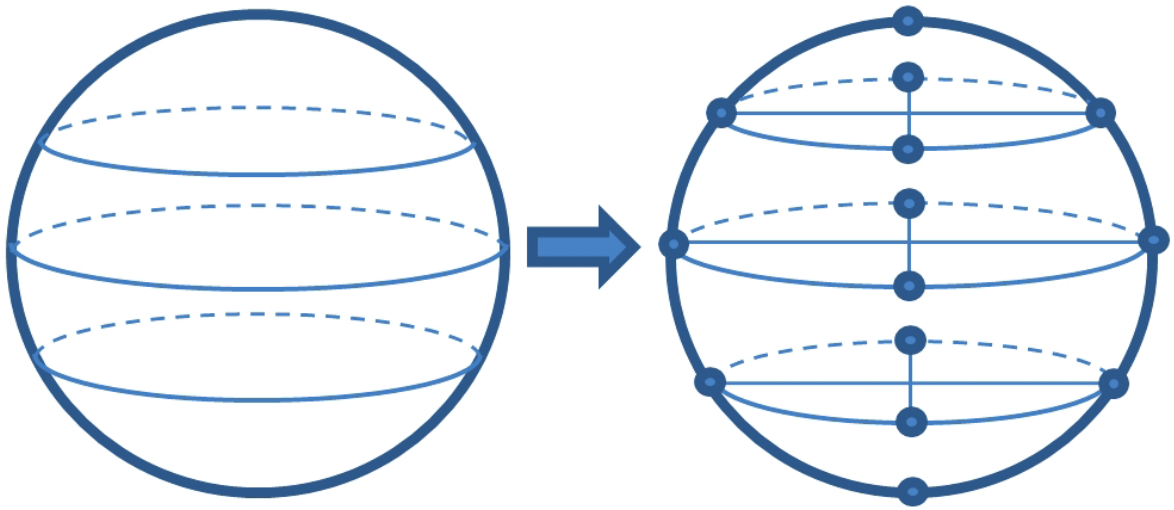
Strategy to Draw Sphere (1)

Select a *precision* representing a number of circular regions the sphere is divided into (e.g. 4)



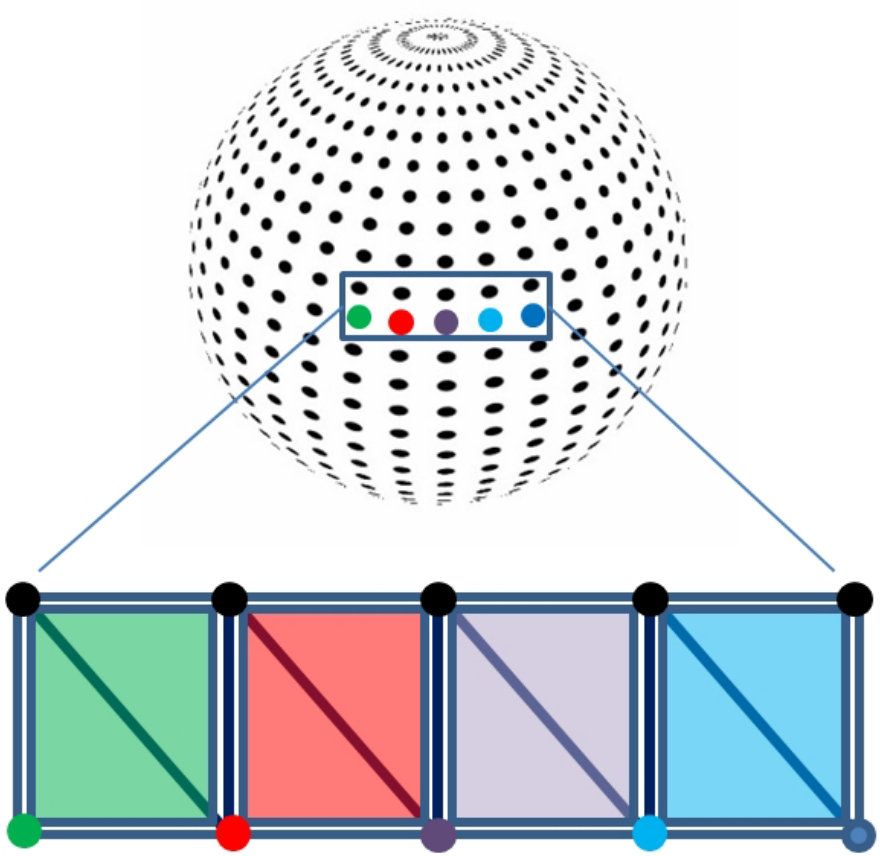
Strategy to Draw Sphere (2)

Subdivide the circumference of each circular slice into some number of points. More points and slices produce more accurate and smoother model. In our example, each slice has same number of points.



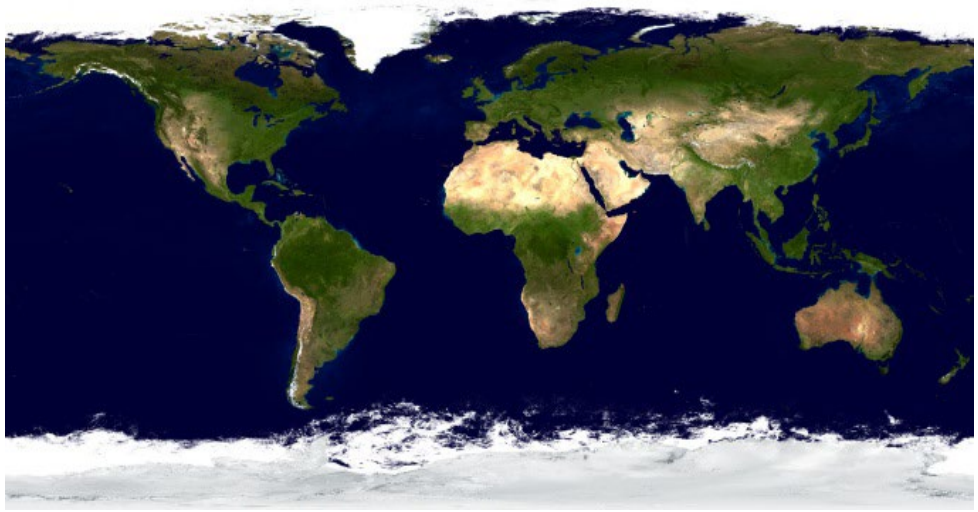
Strategy to Draw Sphere (3)

Group the vertices into triangles and build two triangles at each step.

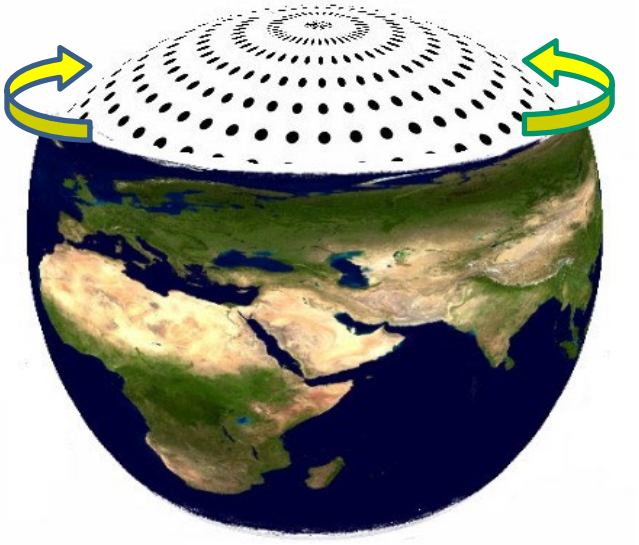


Strategy to Draw Sphere (4)

Select texture coordinates to wrap image around sphere.



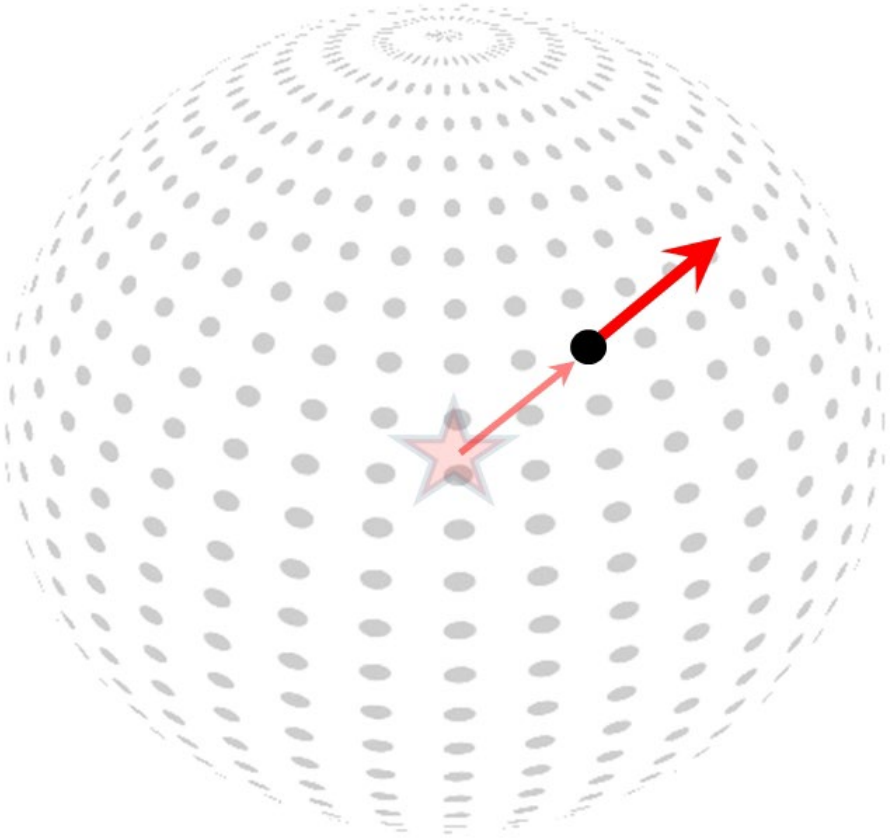
Texture image



Spherical texture coordinates assigned to each vertex

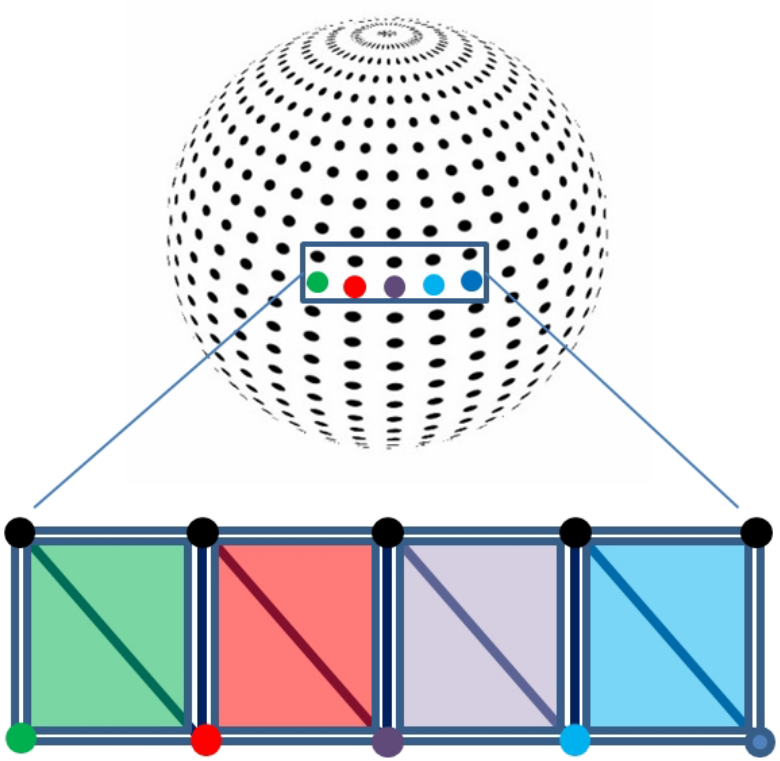
Computing Normal Vectors

The vector pointing from the sphere center to the vertex is the normal vector. Useful for lighting.



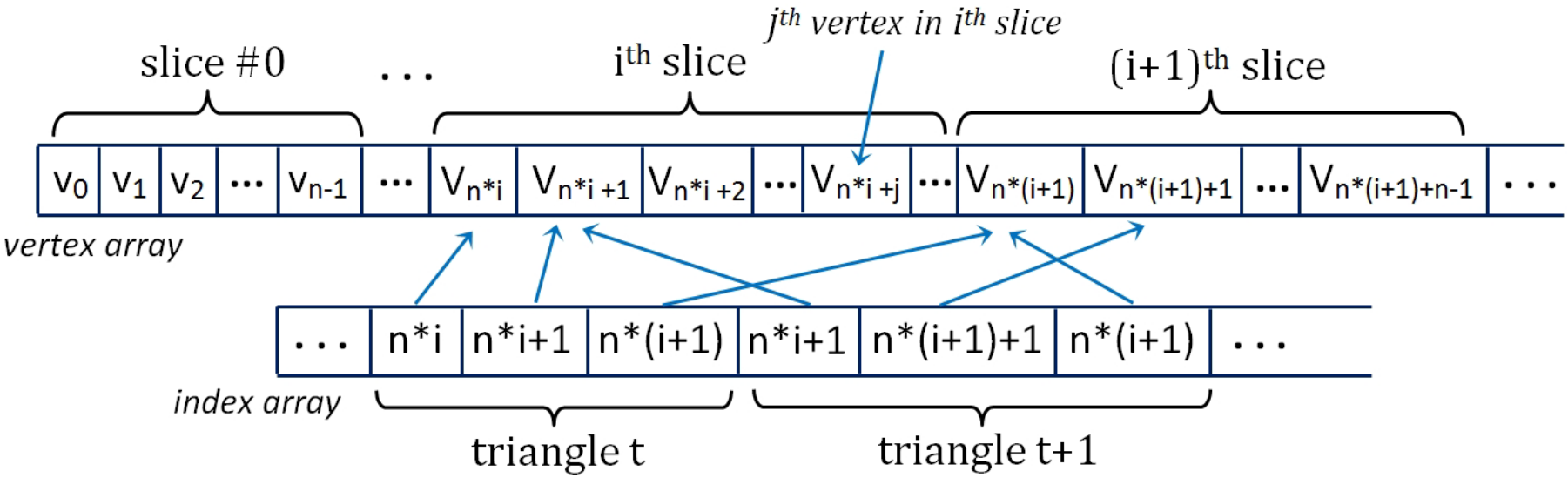
Vertex Indices

Each vertex appears in multiple triangles. To reduce excessive storage, store each vertex once and use indices to reference them.

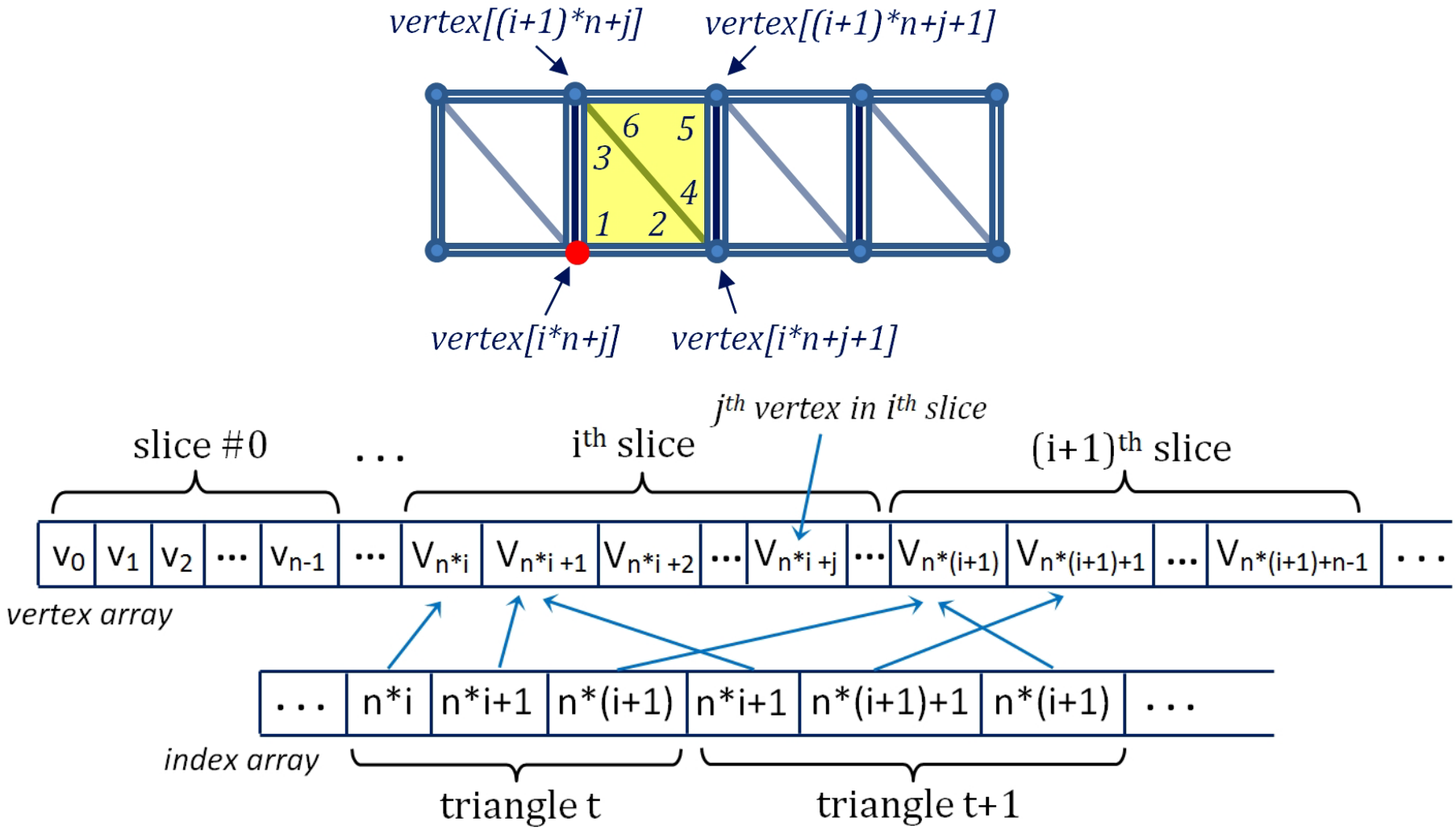


Vertex and Index Arrays (1)

Store vertices in a 1D array, starting with vertices in the bottommost horizontal slice. Assume each slice contains *n* vertices. Each triangle is defined as three indices indexing into vertex array.

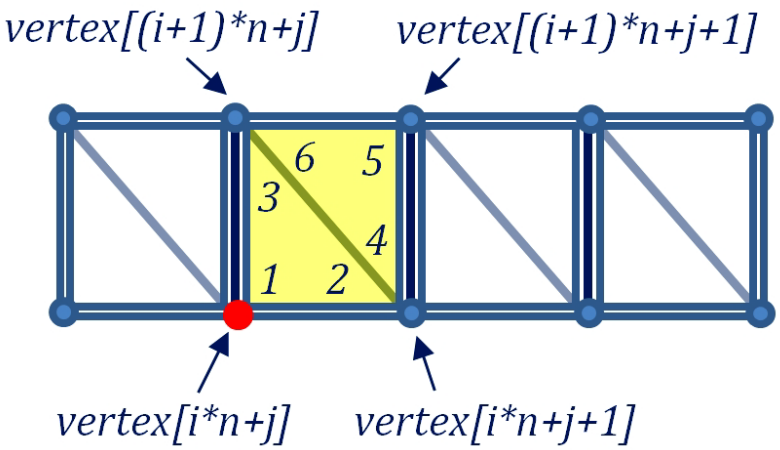


Vertex and Index Arrays (2)



Vertex and Index Arrays (3)

for each horizontal slice **i** ***(i ranges from 0 through all slices in the sphere)*** **{**
    for each vertex **j** ***in slice*** **i** ***(j ranges from 0 through all vertices in the slice)*** **{**
        build two triangles out of neighboring vertices above and right of vertex **j**
    }
}



vertices generated for the j^{th} vertex in the i^{th} slice
(*n = number of vertices per slice*)

Sphere::init()

```
void Sphere::init(int prec) {
    numVertices = (prec+1) * (prec+1);
    numIndices = prec * prec * 6;
    // generate space in the vertex, texCoords, normals, and index arrays
    for (int i=0; i<numVertices; i++) { vertices.push_back(glm::vec3()); }
    for (int i=0; i<numVertices; i++) { texCoords.push_back(glm::vec2()); }
    for (int i=0; i<numVertices; i++) { normals.push_back(glm::vec3()); }
    for (int i=0; i<numVertices; i++) { indices.push_back(0); }

    for (int i=0; I <= prec; i++) {  // calculate triangle vertices
        for (int j=0; j <= prec; j++) {
            float y =  (float) cos(toRadians(180-i*180/prec));
            float x = -(float) cos(toRadians(j*360/prec)) * (float) abs(cos(asin(y)));
            float z = (float) sin(toRadians(j*360/prec)) * (float) abs(cos(asin(y)));
            vertices[i*(prec+1)+j] = glm::vec3(x,y,z);
            texCoords[i*(prec+1)+j] = glm::vec2(((float)j/prec, (float)i/prec));
            normals[i*(prec+1)+j] = glm::vec3(x,y,z);
        }
    }
```

Sphere::init() cont'd

```
for(int i=0; i<prec; i++) {      // calculate triangle indices
    for(int j=0; j<prec; j++) {
        indices[6*(i*prec+j)+0] = i*(prec+1)+j;
        indices[6*(i*prec+j)+1] = i*(prec+1)+j+1;
        indices[6*(i*prec+j)+2] = (i+1)*(prec+1)+j;
        indices[6*(i*prec+j)+3] = i*(prec+1)+j+1;
        indices[6*(i*prec+j)+4] = (i+1)*(prec+1)+j+1;
        indices[6*(i*prec+j)+5] = (i+1)*(prec+1)+j;
    }
}
```

Using the Sphere class

```
mySphere = new Sphere(24);
. . .
void setupVertices(void) {
    std::vector<int> ind = mySphere.getIndices();
    std::vector<glm::vec3> vert = mySphere.getVertices();
    std::vector<glm::vec2> tex = mySphere.getTexCoords();
    std::vector<glm::vec3> norm = mySphere.getNormals();

    std::vector<float> pvalues;      // vertex positions
    std::vector<float> tvalues;      // texture coordinates
    std::vector<float> nvalues;      // normal vectors

    int numIndices = mySphere.getNumIndices();
    for (int i = 0; i < numIndices; i++)  {
        pvalues.push_back((vert[ind[i]]).x);
        pvalues.push_back((vert[ind[i]]).y);
        pvalues.push_back((vert[ind[i]]).z);

        tvalues.push_back((tex[ind[i]]).s);
        tvalues.push_back((tex[ind[i]]).t);

        nvalues.push_back((norm[ind[i]]).x);
        nvalues.push_back((norm[ind[i]]).y);
        nvalues.push_back((norm[ind[i]]).z);
    }
```

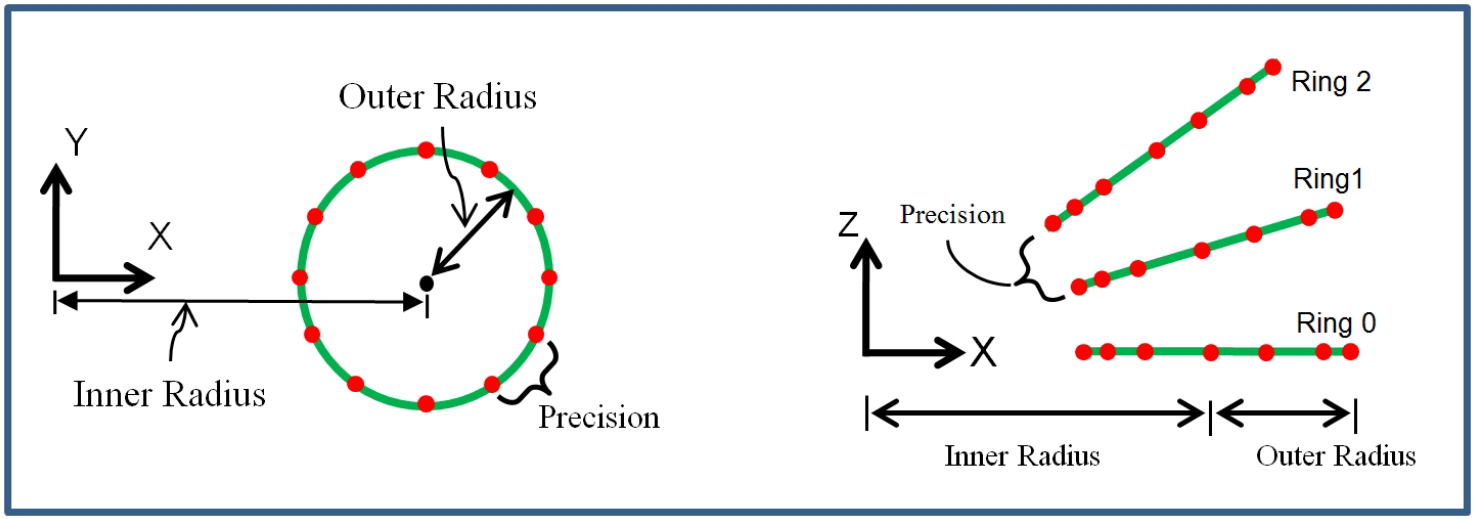
Using the Sphere class (cont'd)

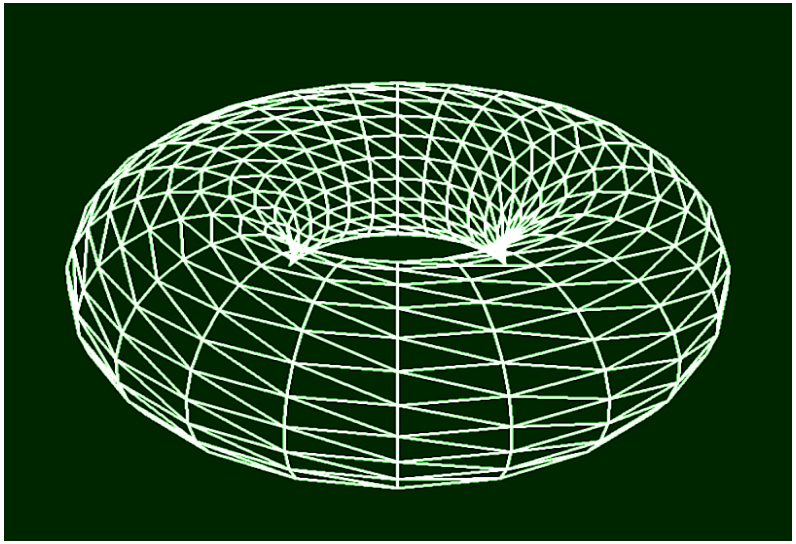
```
    glGenVertexArrays(1, vao);
    glBindVertexArray(vao[0]);
    glGenBuffers(3, vbo);
    // put the vertices into buffer #0
    glBindBuffer(GL_ARRAY_BUFFER, vbo[0]);
    glBufferData(GL_ARRAY_BUFFER, pvalues.size()*4, &pvalues[0], GL_STATIC_DRAW);
    // put the texture coordinates into buffer #1
    glBindBuffer(GL_ARRAY_BUFFER, vbo[1]);
    glBufferData(GL_ARRAY_BUFFER, tvalues.size()*4, &tvalues[0], GL_STATIC_DRAW);
    // put the normal coordinates into buffer #2
    glBindBuffer(GL_ARRAY_BUFFER, vbo[2]);
    glBufferData(GL_ARRAY_BUFFER, nvalues.size()*4, &nvalues[0], GL_STATIC_DRAW);
}
```

in display()

```
. . .
glDrawArrays(GL_TRIANGLES, 0, mySphere.getNumIndices);
. . .
```

Building a Torus





OpenGL Support of Indexing

store the vertices in a standard vbo:

```
glBindBuffer(GL_ARRAY_BUFFER, vbo[0]);
glBufferData(GL_ARRAY_BUFFER, pvalues.size()*4, &pvalues[0], GL_STATIC_DRAW);
```

store the indices in a special vbo:

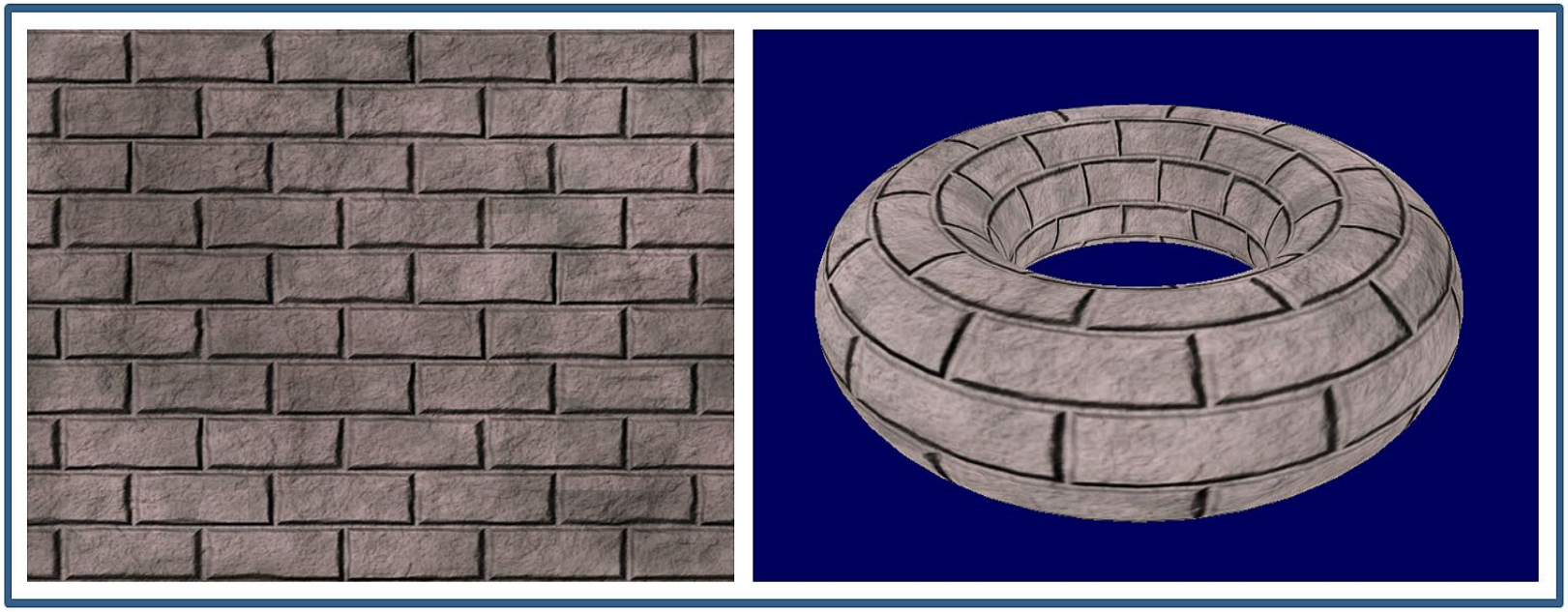
```
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, vbo[3]);
glBufferData(GL_ELEMENT_ARRAY_BUFFER, ind.size()*4, &ind[0], iBuf, GL_STATIC_DRAW);
```

in display(), replace glDrawArrays with glDrawElements:

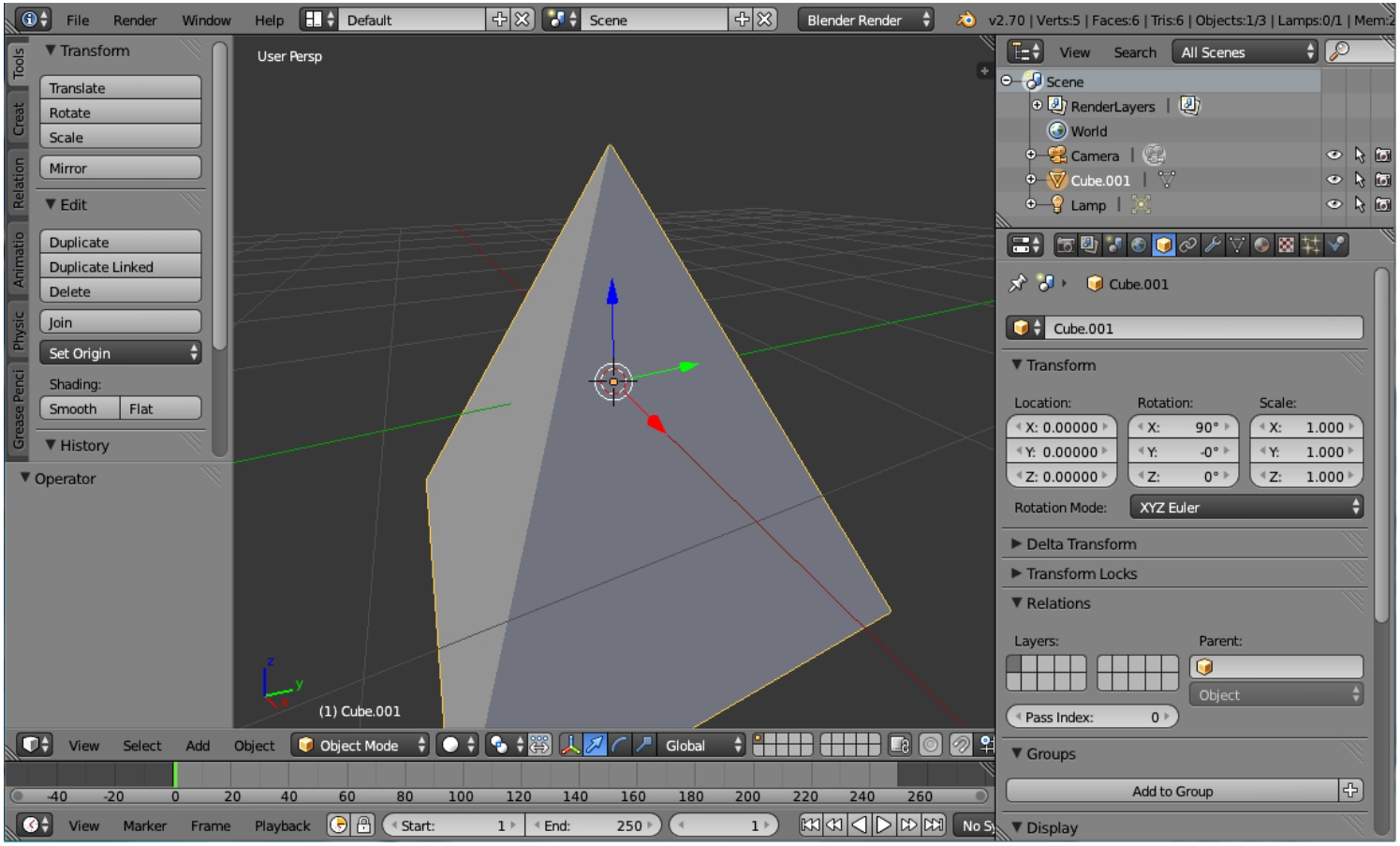
```
int numTorusIndices = myTorus.getNumIndices();
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, vbo[3]);
glDrawElements(GL_TRIANGLES, numTorusIndices, GL_UNSIGNED_INT, 0);
```

OpenGL uses this vbo (internally) to index the vertex attributes.
It isn't necessary to declare anything for it in the shaders.

Texturing Example



Externally Produced Models



Pyramid created in Blender, exported as an OBJ file

OBJ Format

```
# Blender v2.70 (sub 0) OBJ File: ''
# www.blender.org
o Pyramid
v 1.000000 -1.000000 -1.000000
v 1.000000 -1.000000 1.000000
v -1.000000 -1.000000 1.000000
v -1.000000 -1.000000 -1.000000
v 0.000000 1.000000 0.000000
vt 0.515829 0.258220
vt 0.515829 0.750612
...
vn 0.000000 -1.000000 0.000000
vn 0.894427 0.447214 0.000000
...
s off
f 2/1/1 3/2/1 4/3/1
f 1/4/2 5/5/2 2/6/2
f 2/7/3 5/8/3 3/9/3
f 3/9/4 5/10/4 4/11/4
f 5/12/5 1/13/5 4/14/5
f 1/15/1 2/1/1 4/3/1
```

vertex locations

texture coordinates

normal vectors

triangle faces (v/t/n indices for each vertex)

OBJ Face Format

```
f 2 5 3              Only vertex indices available
f 2/7 5/8 3/9        Vertex and texture coordinates only
f 2//3 5//3 3//3     Vertex and normal only (no texture coords)
```

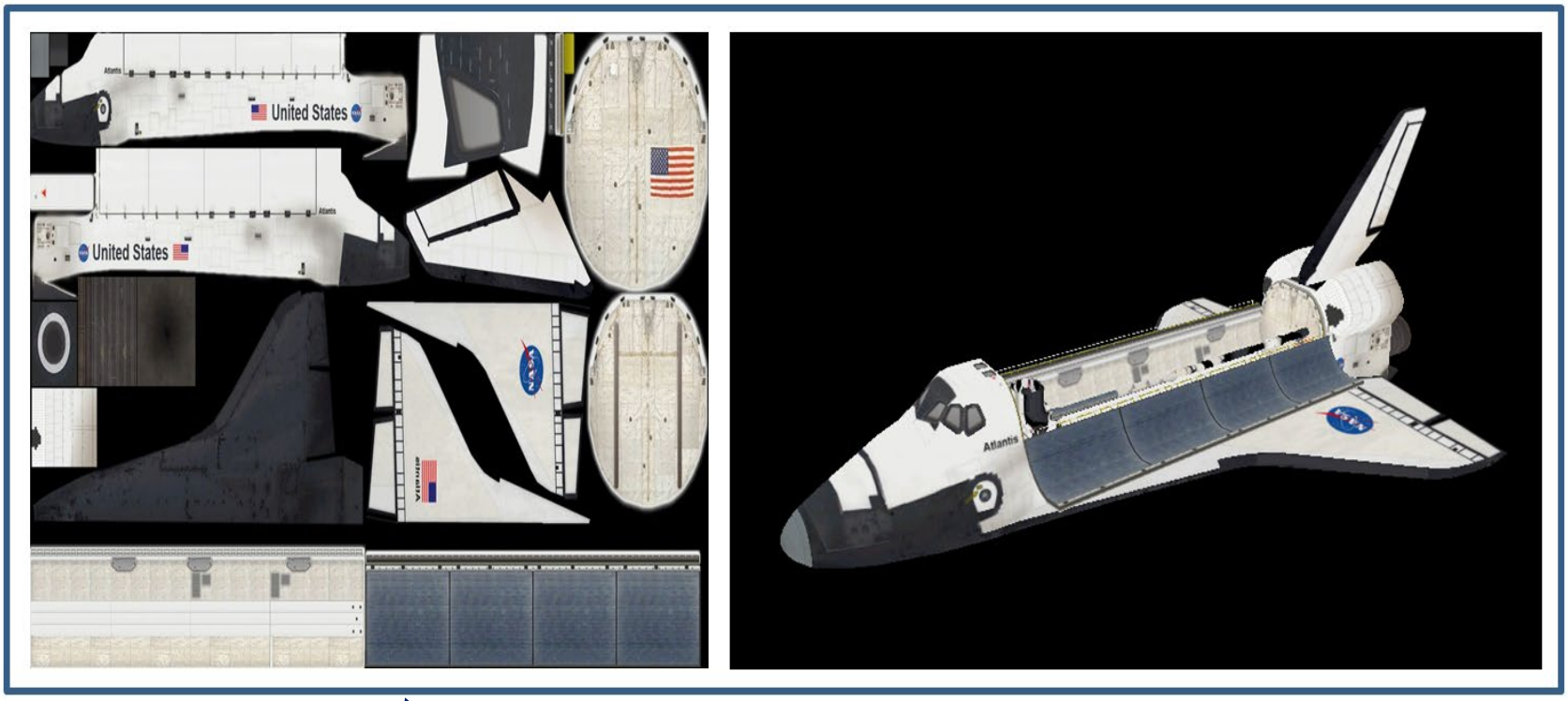
Using the Simple OBJ Model Importer

```
. . .
ImportedModel myModel("shuttle.obj");          //  in top-level declarations
    (note:  ImportedModel class, and model importer code shown in textbook)
. . .
void setupVertices(void)  {
    std::vector<glm::vec3> vert = myModel.getVertices();
    std::vector<glm::vec2> tex = myModel.getTextureCoords();
    std::vector<glm::vec3> norm = myModel.getNormals();

    int numObjVertices = myModel.getNumVertices();

    std::vector<float> pvalues;     // vertex positions
    std::vector<float> tvalues;     // texture coordinates
    std::vector<float> nvalues;     // normal vectors

    for (int i=0; i<numObjVertices; i++)  {
        //  loading pvalues, tvalues, and nvalues same as previous example
        . . .
    }
    //  loading three VBOs with vertices, texture coordinates, and normals same as before
}
```

in display():

```
. . .
glDrawArrays(GL_TRIANGLES, 0, myModel.getNumVertices());
```

Loading NASA Shuttle OBJ Model



Example of UV-mapping – where texture coordinates in the model are carefully mapped to particular regions of the texture image