XIAMEN UNIVERSITY

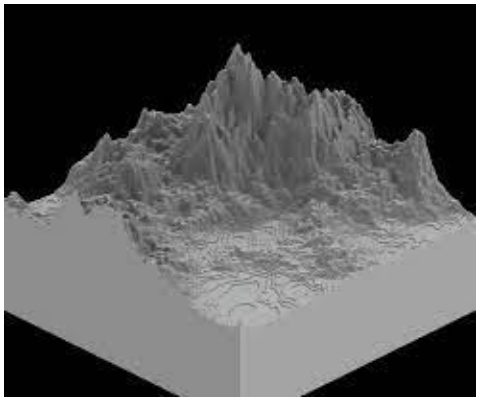
COMPUTER GRAPHICS

第六章 3D模型

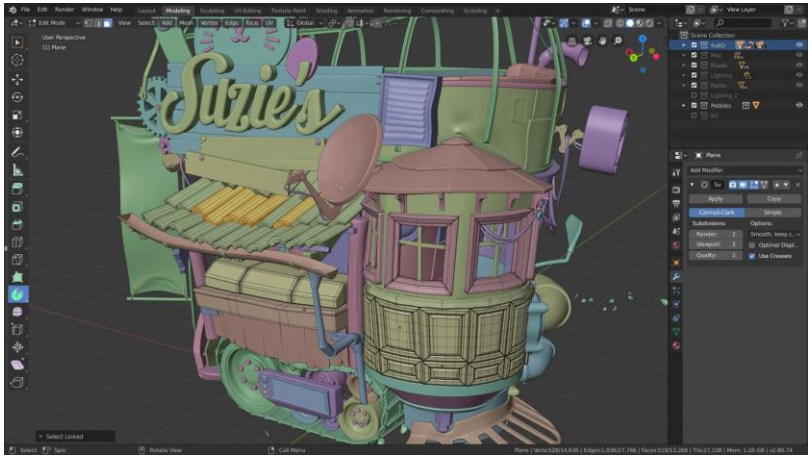
陈中贵
厦门大学信息学院
http://graphics.xmu.edu.cn

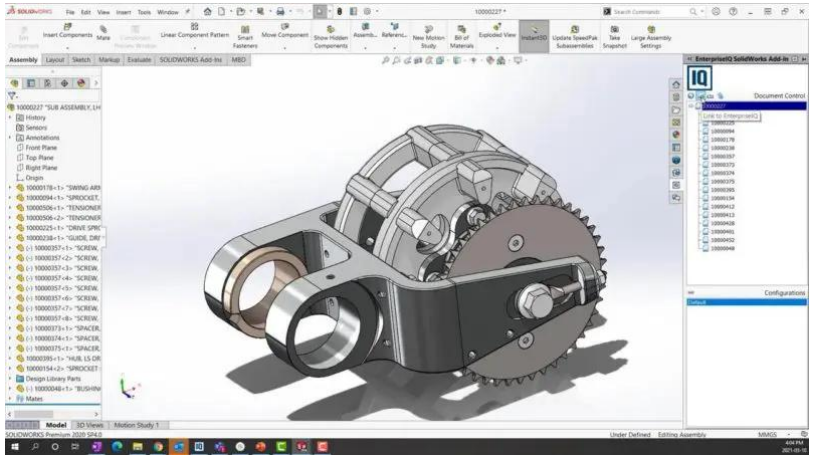
3D 建模

- 通过程序来构建模型



- 加载外部创建的模型





程序构建模型——构建一个球体

- 选择模型逼近精度，表示将球体分成相应份数的圆形部分，如图 6.2 左侧所示，球体被分成了 4 个部分
- 将每个圆形切片的圆周细分为若干个点，如图 6.2 右侧所示
- 将顶点分组为三角形，见图6.3
- 根据纹理图像的性质选择纹理坐标
- 为顶点生成法向量

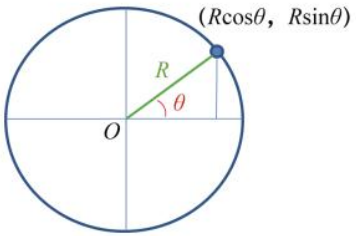


图 6.1　构成圆周的点

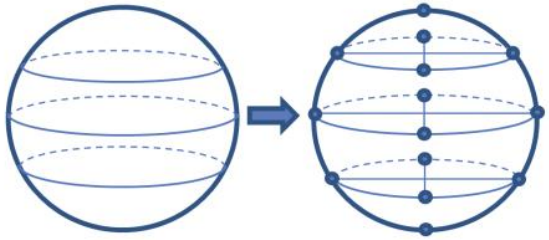


图 6.2　构建圆形顶点

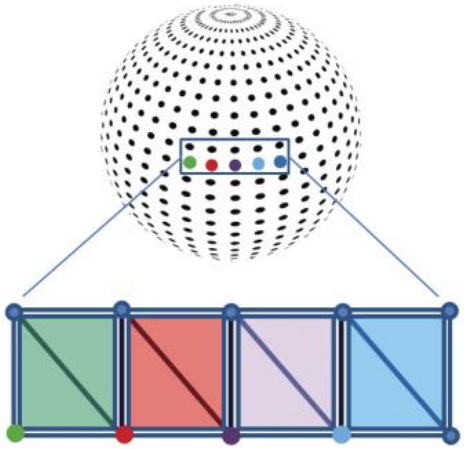
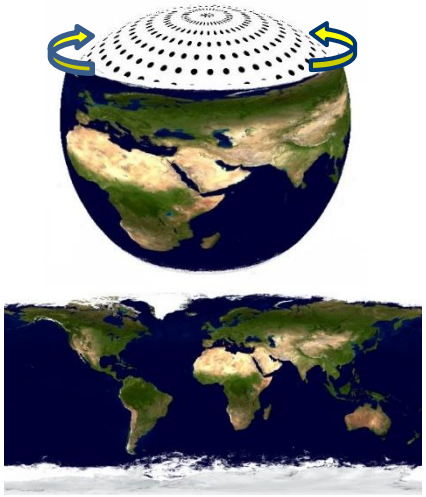


图 6.3　将顶点组合成三角形



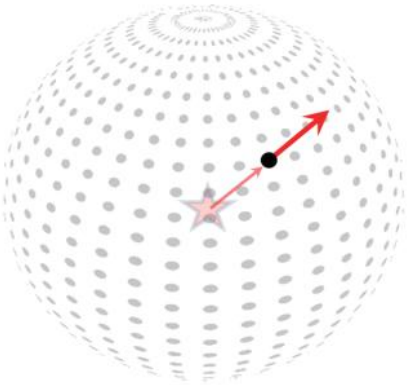


图 6.4　球体顶点法向量

构建三角形顶点和索引

- 使用索引定义三角形，仅存储每个顶点一次，节省内存
- 顶点存储在一维数组中
- 索引数组包含相应三角形的每个点，并将值设为顶点数组 *V* 中的整型引用

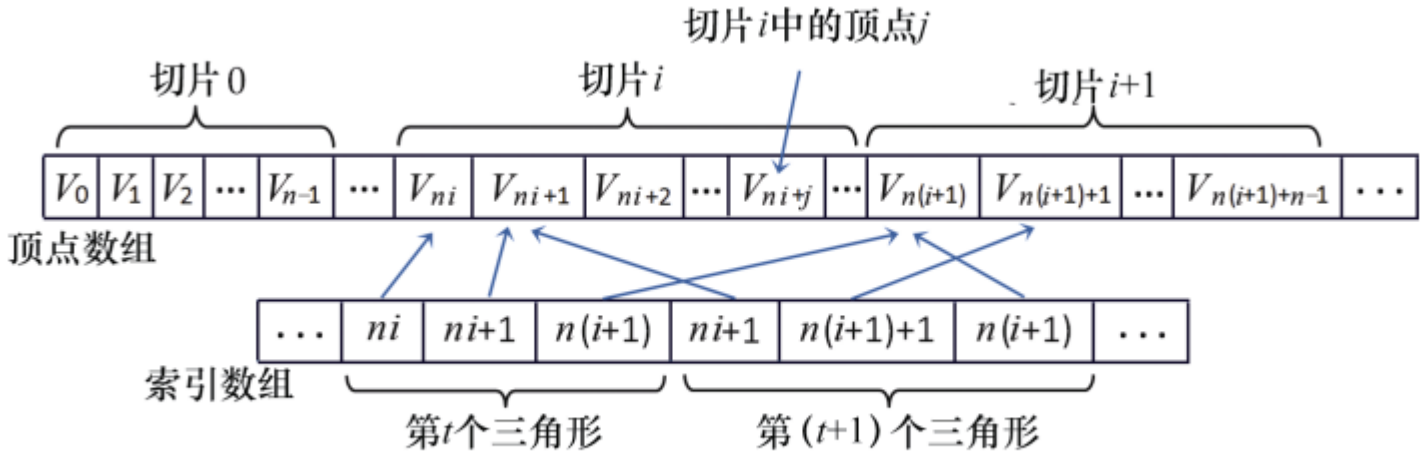


图 6.5　顶点数组和相应的索引数组

利用索引连接三角形

- 球体底部开始，围绕每个水平切片以圆形方式遍历顶点，访问每个顶点时，我们构建两个三角形，在其右上方形成一个方形区域，如图 6.3 所示

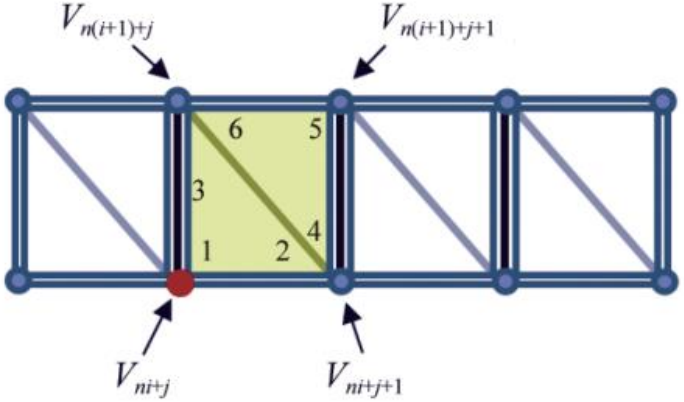


图 6.6　第 i 个切片中的第 j 个顶点的索引序号
（n 为每个切片的顶点数）

```
对于球体中的每个水平切片 i（i 的取值从 0 到球体中的所有切片数）
{    对于切片 i 中的每个顶点 j（j 的取值从 0 到切片中的所有顶点数）
        {计算顶点 j 的指向右边相邻顶点、上方顶点，以及右上方顶点
        的两个三角形的索引
        }
}
```

程序 6.1 程序生成的球体

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

程序 6.1 使用Sphere类

```
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);
    }
```

注意：此处顶点数据冗余，没有真正使用索引

. . . . . *(continued)*

程序 6.1 使用Sphere类

```
    . . .
    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()

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

犹他茶壶

- 1975 年由犹他大学Martin Newell 开发， 使用了各种贝塞尔曲线和曲面
- 在计算机图形学界广泛采用的标准参照物体
- GLUT 包括绘制茶壶的程序

图 6.7　OpenGL GLUT 绘制的茶壶

构建一个环面

- 先定义圆形切片，然后围绕y轴旋转切片以形成环面

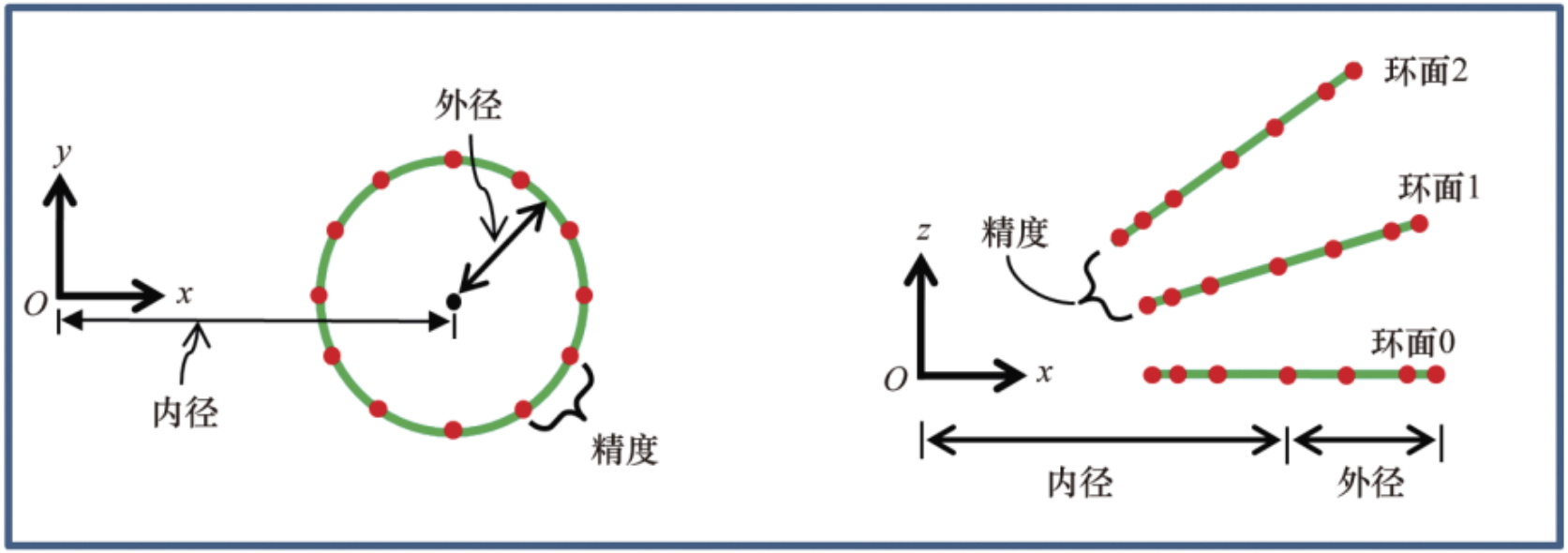
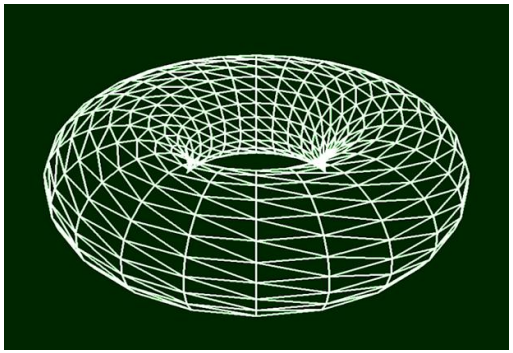


图 6.8　构建一个环面



详细环面构建过程参考：Baker, P., *Paul's Projects*, www.paulsprojects.net

OpenGL 中的索引

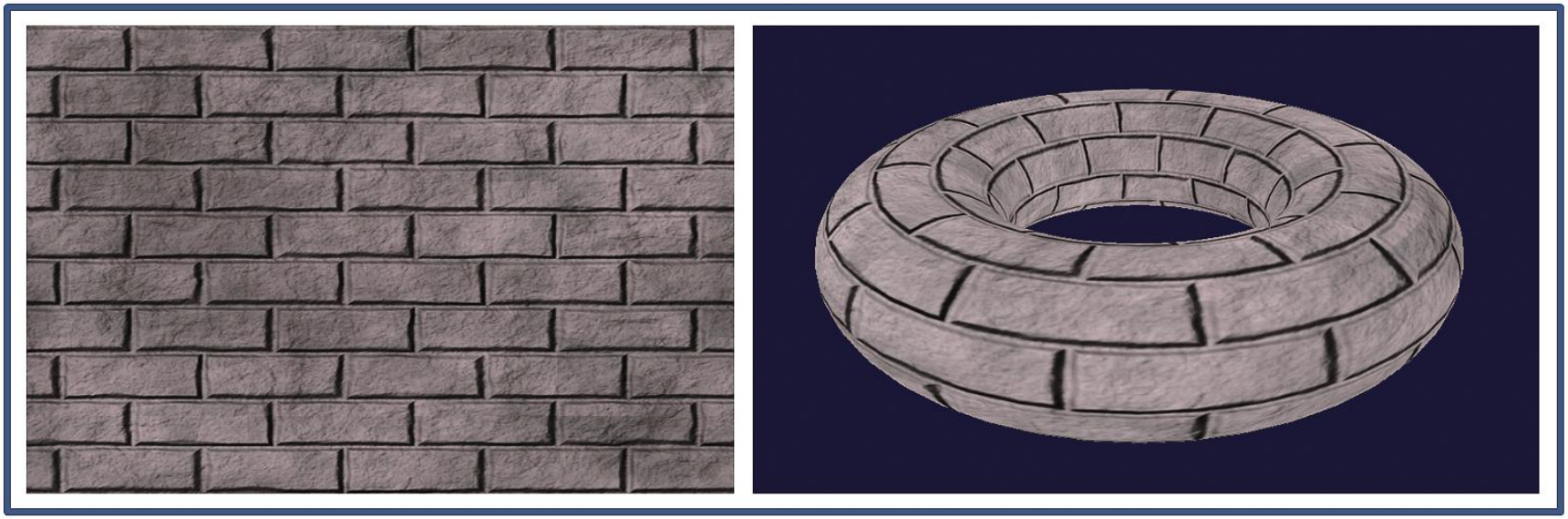
- OpenGL 的索引时，需要将索引本身也加载到 VBO
- 指定 VBO 的类型为 GL_ELEMENT_ARRAY_BUFFER（告诉 OpenGL 这个 VBO 包含索引）

```
std::vector<int> ind = myTorus.getIndices();       // 环面索引的读取函数返回整型向量类型的索引
...
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, vbo[3]);     // vbo[3]是新增的 VBO
glBufferData(GL_ELEMENT_ARRAY_BUFFER, ind.size()*4, &ind[0], GL_STATIC_DRAW);
```

- display()中，我们将 glDrawArrays()调用替换为 glDrawElements()调用（告诉 OpenGL利用索引 VBO 来查找要绘制的顶点）

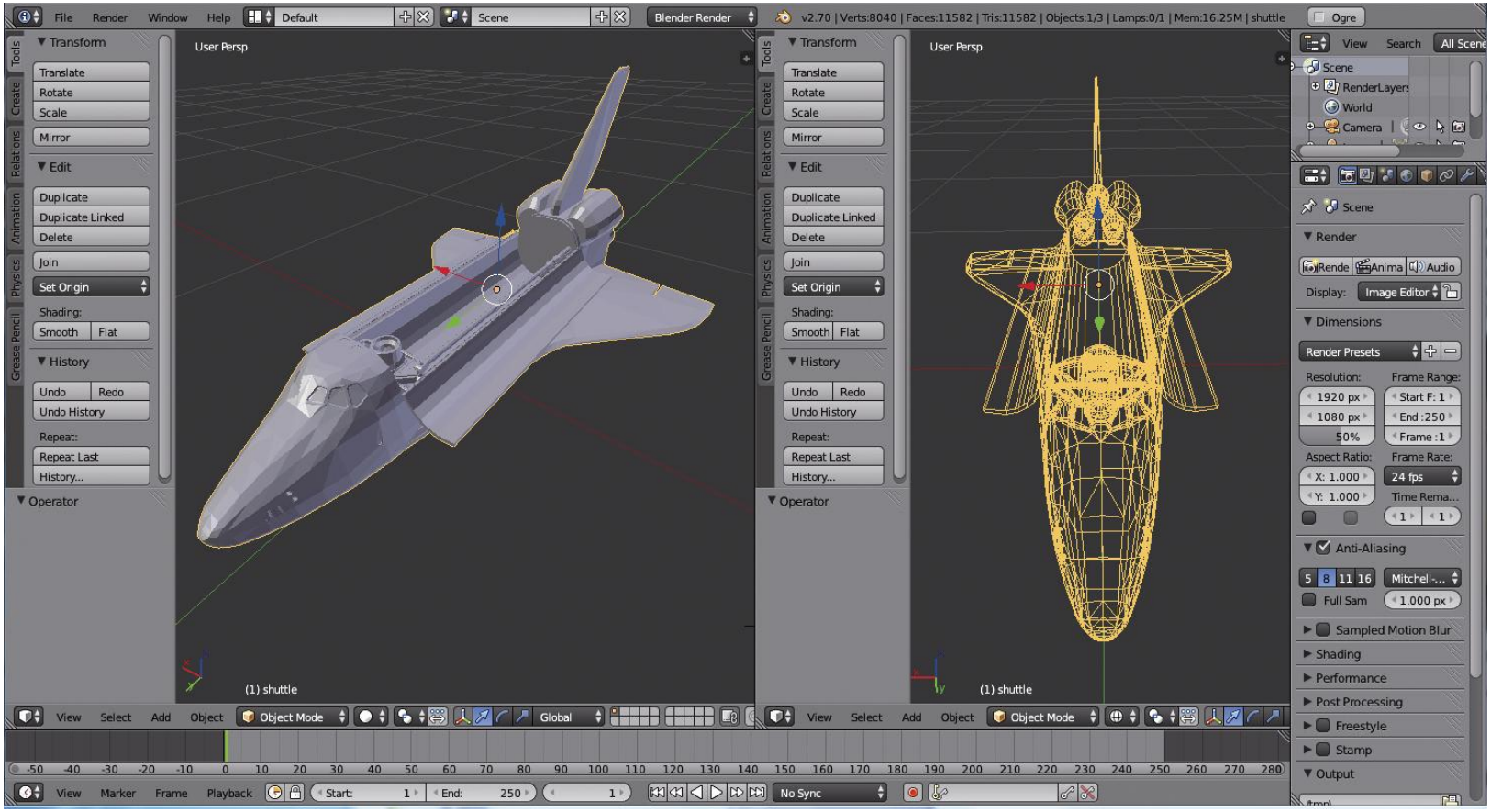
```
numTorusIndices = myTorus.getNumIndices();
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, vbo[3]);
glDrawElements(GL_TRIANGLES, numTorusIndices, GL_UNSIGNED_INT, 0);
```

程序 6.2 程序生成的环面



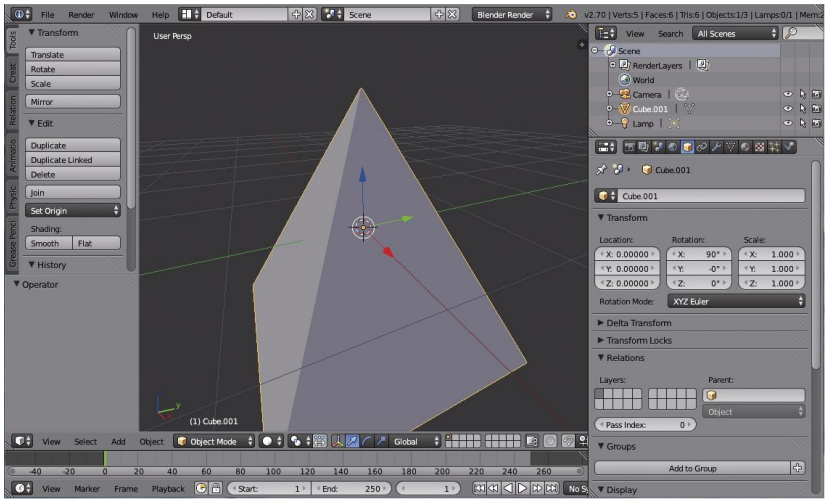
加载外部构建的模型

- 建模软件： Maya、 Blender、 LightWave、 Cinema4D



3D模型文件

- Wavefront（.obj）、 3D Studio Max（.3ds）、斯坦福扫描存储库（.ply）、 Ogre3D（.mesh）等等
- Obj文件格式：每行以字符标签开头，以标明该行的数据类型
  - v：几何数据（顶点位置）
  - vt：纹理坐标
  - vn：顶点法向量。
  - f：面。每个面（三角形）具有3个元素，每个元素具有由“/”分隔的3个值，每个元素的值分别是顶点列表、纹理坐标和法向量的索引



Obj文件格式

vertex locations

texture coordinates

normal vectors

triangle faces

```
# 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
```

Obj文件格式

- OBJ 格式的模型并不要求具有法向量，甚至纹理坐标
  f 2 5 3
- 如果模型具有纹理坐标，但没有法向量，则格式为：
  f 2/7 5/8 3/9
- 如果模型具有法向量，但没有纹理坐标，则格式为：
  f 2//3 5//3 3//3

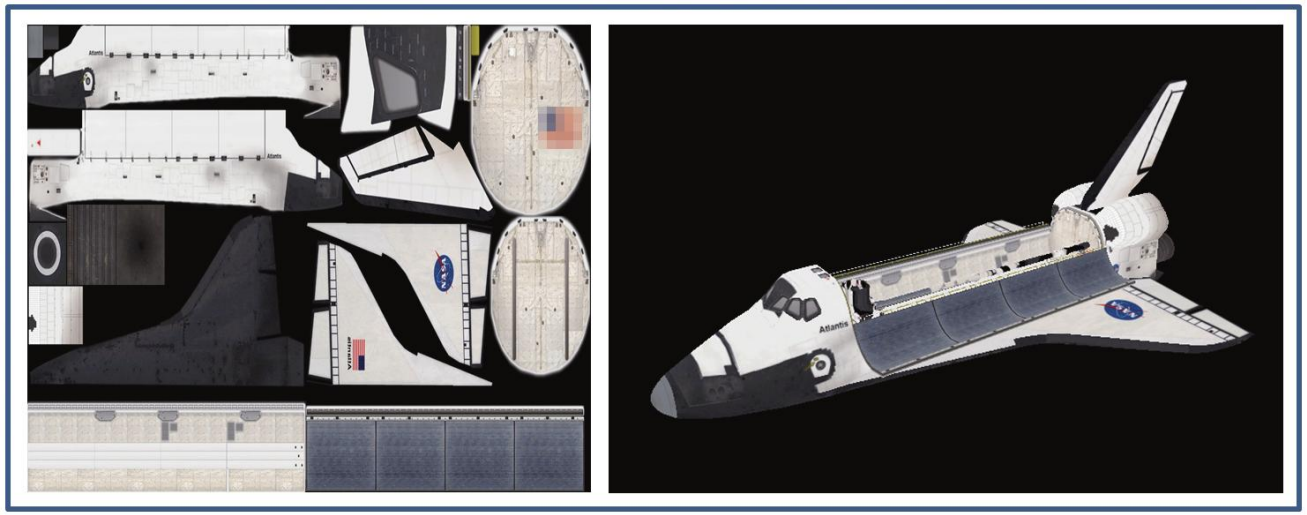


图 6.13 带有纹理的 NASA 航天飞机模型

程序 6.3 简化的OBJ 加载器

```
. . .
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());
```