CS52 Assignment 2: Modeling

Out: Oct 12. Due: Oct 26.

Introduction

In your second assignment, you will learn how to setup a transformation hierarchy and tessellate complex objects into triangles. Interactive drawing will then be done through OpenGL in a class included in the framework code. Just like raytracing is a fundamental algorithm for high-quality rendering, transformation hierarchies and tessellated objects are the fundamental entities used in interactive rendering, together with the rendering code included in libraries such as OpenGL/DirectX.

You are to perform this assignment using Java. To ease your development, we are providing a set of classes to load a scene, perform basic mathematical calculations and provide a user interface together with interactive drawing code. You will also need the *jogl* library, that you can download from jogl.dev.java.net for your system (we are providing the Windows version).

The final result of this assignment is an interactive program with support for the following features.

- Tranformations
  - Translation
  - Rotation
  - Scaling
  - Hierarchies
- Tesselation
  - Spheres
  - Meshes
  - Surface of Revolution

To ease your debugging, we are providing the compiled version of the solution code, which can be used to compare your results with ours. Do not try to decompile the code.

Requirements

1. Implement tessellation functions for each surface type. The abstract function declaration is in `Surface.tessellate`. For each Surface class, this function should define a

tessellated mesh of either quads or triangles stored in the `Surface.tesselatedMesh`. Each of these meshes should contain vertex positions and normals. The tessellation level can be changed in the GUI drop-down menu.

Start the tessellation code with just vertex positions (just set the normal array to null). The object will look faceted, but it is a good starting point. You can then fill in the normals after. Also remember that the vertex order in a triangle or quad is important (if oriented wrong, the faces will disappear - you can use the culling option in the GUI to check this).

2. Implement the `Sphere.tesselate` function that creates a polygonal mesh approximating the given sphere. You can pick either quads or triangles for this. The sphere should be tessellated uniformly along ϕ and θ as in

$$\mathbf{P}(\phi, \theta) = \mathbf{O} + R(\sin\theta\cos\phi, \cos\theta, \sin\theta\sin\phi)$$

with $2^{(t_s+2)}$ subdivision along each ϕ and θ, where t_s is the `tessellationLevel` parameter.

3. Implement the `Mesh.tesselate` that will refine a given mesh. This should be implemented as recursively subdividing triangles and quads with the same topology given in the subdivision surface slides. Geometrically, simply define the new vertex and normals as the average of the values of the generating vertices in the parent mesh. For example, since the new vertices in the triangle will be at the midpoint of each edge, their position and normal is the average of the vertices at the endpoints of the edge. The `tesselationLevel` parameter indicates how many times the subdivision process is applied.

4. Implement the `RevolvedSurface.tesselate` method that creates a polygonal mesh approximating a surface of revolution based on a spline profile. For the purpose of this exercise, we will assume that the spline lines in the xy plane and that the surface of revolution axis is the y axis (note that all computation for the spline will be performed with 3d vectors anyway). You should uniformly sample the spline profile and the circle and define the new vertex position and normals as

$$\mathbf{P}(\phi, t) = (\mathbf{s}_x(t)\cos\phi, \mathbf{s}_y(t), \mathbf{s}_x(t)\sin\phi)$$

$$\mathbf{N}(\phi, t) = -(\mathbf{t}_x(t)\cos\phi, \mathbf{t}_y(t), \mathbf{t}_x(t)\sin\phi) \times (\sin\phi, 0, -\cos\phi)$$

where $\mathbf{s}(t), \mathbf{t}(t)$ are the position and tangent on the spline for the parameter t. You can pick either quads or triangles for this and you should use $2^{(t_s+2)}$ subdivision along each parameter. (Hint: this has the same topology as your parametric sphere).

5. Implement transformation hierarchies. We provide an implementation of them using the hardware-accelerated OpenGL library. You are to provide a software-only implementation by first computing the appropriate transformation matrix and then applying this to each vertex position and normal of the tessellated mesh. This is done by implementing the functions that override `HierarchyNode.flatten` for transform and surface nodes. Your result should be stored in `Surface.flattenedTesselatedMesh`.

   You can switch between OpenGL and software transforms using the "flattening" drop-down list in the GUI. This function is called every time the camera is changed.

6. The code in `Transform.flatten` implements combination of transformations in the hierarchy. The definition of the transform at a node is

$$M = M_p \cdot M_l = M_p \cdot T_{\mathbf{t}} \cdot S_{\mathbf{s}} \cdot R^x_{\theta_x} \cdot R^y_{\theta_y} \cdot R^z_{\theta_z}$$

   where M_p is the parent transform and M_l is the local one. You should implement this by first filling in the functions in the `Mat4` class.

7. Implement the `Surface.flatten` method. This method applies the given transform to vertex and position of the tessellated mesh and stores the results in the flattened mesh. For simplicity, you can assume that the scaling transform is uniform (Hint: transforming normals becomes much easier).

8. You are *not* to change the code for the Triangle class. This is there so you can get your code to draw something when it starts up. Note that it will not properly tessellate (otherwise you could just paste its code in the mesh class).

Framework Overview

We suggest you use our framework to create your program. We have removed from the code all the function implementations you will need to provide, but we have left the function declarations which can aid you in planning your solution.

Following is a brief description of the classes provided in the framework. We have taken various steps to simply the framework code as much as possible. In particular, you will notice that most of the variables are declared as public and all classes are in the same package. Please note that this is not good programming practice! We have done this to help making the code more readable by reducing the length of the classes.

Main. This is the main entry point of your code. It will load the scene and launch the GUI.

MainFrame, GLRenderPanel. These are the classes that implements the GUI of the program. You should play a bit with the various options for drawing that can help you debug your code and get intuition about the geometry and topology of the objects. Also you can use left and right mouse button on the viewer to rotate and scale the scene interactively.

FileFormat. This class implements methods need to load the scene as in the previous programming assignment.

Scene. The scene is a container for the scene hierarchy, lights and a view camera. It contains the method `Scene.tesselate` and `Scene.flatten` that applies these methods to the root of the hierarchy.

Camera. This is a representation of the scene viewer.

HierarchyNode. The class is the superclass to each node in the scene hierarchy. It exposes two functions: `HierarchyNode.tesselate` and `HierarchyNode.flatten`. The tesselate function is responsible for turning geometry into collections of triangles or quads as described above. The flatten function is responsible for flattening the hierarchy by applying the transform to each vertex position and normal.

There are two types of node in the hierarchy. **Transform** nodes represents transformations and can have children. **Surface** nodes represents geometry and cannot have children.

Surface, Triangle, Sphere, RevolvedSurface, Mesh. These classes represent the geometry types available in the scene. In particular the `Mesh` is stored as indexed face lists where the vertex indexes are flattened. The `Mesh.quads` variables indicate whether the mesh stores triangles or quads lists. Various methods are provide to create faces and get face indexes if you would like to use them.

Material, Lambertian, Phong; Light, PointLight. These classes represent the materials and lights types for the scene, as in the previous assignments. You are not to change these classes.

Vec3, Mat4, Color These are classes for the basic mathematical types used in the framework. `Vec3` represents three dimensional points, vectors and normals. `Color` represents an (R,G,B) color. `Mat4` represents 4x4 matrices.

Submission

Please send your code and compiled class files to fabio@cs.dartmouth.edu. We will run your code by calling the main method provided. Also add images by screen-capturing the program at tessellation level 3.

Extra credit

Choose *some* of these items as extra credit.

1. Implement extrusion of a spline along the y axis with uniform subdivision.
2. Implement extrusion of a spline along another spline (not necessarily linear).
3. Implement Loop subdivision surfaces. You can just handle the case of meshes with no boundaries. The subdivision topology of the Mesh class will work for this one, but you probably want to use some better adjacency data structure. This is hard, but the most interesting one of the whole set. We would be really impressed if you can get this done! Do not worry about implementing a very efficient solution, just one that works.