OpenGL Programming Guide: The Official Guide To...

CLICK HERE

This assignment serves as an introduction to the OpenGL graphics library. You will be required to generate a room scene which can be navigated in a walk-through fashion. You will begin the project by implementing the OpenGLanalogs of many of the methods that you have implemented for the last ray-tracerassignment. Emphasis is placed both upon the implemention of OpenGL's basic capabilities (e.g. shading. lighting, transparency, materials properties, etc.) and their use in generating more involved effects (e.g. shadows, reflections.)
An overview of the **code** you will be using can be found here or downloaded here.An overview of the .ray file **syntax** can be found here.A (Windows x64) compiled version of the renderer implementing some of the basic features can be found here.The OpenGL programming Guide is an invaluable resource in assisting with OpenGL implementation. You can find a link to the guide here.Getting StartedYou should use the same code-base as in previous assignments (Assignments.zip), as a starting point.As in the previous assignments, code modification should be relegated to the *.todo.cpp files. After you copy the provided files to your directory, the first thing to do is compile the program. To do this, you will first have to compile the JPEG library and then compile the assignment3 executable. **On a Windows Machine**
Begin by double-clicking on Assignments.sln to open the workspace in Microsoft Visual Studios.

- Compile the Assignment3 executable by right-clicking on "Assignment3" and selecting "Build". (If the JPEG.lib, Image.lib, Ray.lib, and Util.lib libraries have not already been compiled, they will be compiled first.)
- The executable Assignment3.exe is compiled in *Release* mode for the *64-bit* architecture and will be placed in the root directory.

On a Linux Machine

- Type make -f Makefile3 to compile the Assignment3 executable. (If the libImage.a, libRay.a, and libUtil.a libraries have not already been compiled, they will be compiled first.) This assumes that JPEG libraries have already been installed on your machine. (If it hasn't been installed yet, you can install it by typing sudo apt-get install libjpeg-dev.)
- The executable Assignment3 is compiled in *Release* mode and will be placed in the root directory.
- If you are having trouble linking either to the gl libraries or the glu libraries, you should install the associated packages: sudo apt-get install libgl1-mesa-dev libglu1-mesa-dev For subsequent assignments, you should also install the glut package: sudo apt-get install freeglut3-dev

Code ModificationsThe code you will be starting with in this assignment is essentially the same code you used for the previous assignment. However,there are a few small changes:

- For those of you who have not (properly) implemented the Shape::updateBoundingBoxfor the different subclasses of Shape, implementation of these methods is provided for this assignment.(The method is needed to determine the center and radius of the scene.) The changes can be found in the following files:
  - Ray/box.todo.cpp
  - Ray/cone.todo.cpp
  - Ray/cylinder.todo.cpp
  - Ray/shapeList.todo.cpp
  - Ray/sphere.todo.cpp
  - Ray/torus.todo.cpp
  - Ray/triangle.todo.cpp

  Please note that these files are old, and some of the member functions no longer exist within the codebase. So just grab the implementation of the Shape::updateBoundingBox method and ignore the rest.

How the Executable WorksThe executable takes in as a mandatory arguments the input (.ray) .ray file name. Additionally,you can also pass in the dimensions of the viewing window and the complexity of the tesselation for objects like thesphere, the cylinder, and the cone. (Specifically, this specifies the resolution, e.g. the number of angular samples.) It is invoked from the command line with:% Assignment3 --in in.ray --width w --height h --cplx cFeel free to add new arguments to deal with the new functionalities you are implementing. Just make sure they aredocumented. What You Have to DoThe assignment is worth 30 points. The following is a list of features that you may implement. The number in parentheses corresponds to how many points it is worth.

- (1) Implement theRay::Camera::drawOpenGL(in Ray/camera.todo.cpp) to draw the camera.
- (1) Implement theRay::ShapeList::drawOpenGL(in Ray/shapeList.todo.cpp) to draw the scene-graph nodes. For now, ignore the input glslProgram parameter.
- (2) Implement theRay::TriangleList::drawOpenGL(in Ray/shapeList.todo.cpp) andRay::Triangle::drawOpenGL(in Ray/triangle.todo.cpp) method to draw triangles with per-vertex normals. For now, ignore the texture coordinatesand the glslProgram parameter.
- (2) Implement the Ray::Sphere::drawOpenGL(in Ray/sphere.todo.cpp) method to draw a sphere at the appropriate tesselation.For now, you can ignore the input glslProgram parameter.
  You may **not** use the gluSphere function from the GLU library to assist you with this. Instead, you will have to explicitly generate the primitive (triangles/polygons) and render them yourself. You should use the parameter Shape::OpenGLTessellationComplexity to set the tessellation complexity.
- (2) Impelement theRay::Material::drawOpenGL(in Ray/scene.todo.cpp) method to draw the material properties.For now, you can ignore the input glslProgram parameter.
- (3) To draw the light sources, implement:
  - Ray::DirectionalLight::drawOpenGL(in Ray/directionalLight.todo.cpp);
  - Ray::PointLight::drawOpenGL(in Ray/pointLight.todo.cpp); and
  - Ray::SpotLight::drawOpenGL(in Ray/spotLight.todo.cpp)

  The input index parameter specifies which of the OpenGL lights you are using and should be used forspecifying the light parameters in the RayLight::drawOpenGL method:glLightfv(GL_LIGHT0+index, ...);
  glEnable(GL_LIGHT0+index);For now, you can ignore the input glslProgram parameter.
- (2) Modify the implementation ofRay::AffineShape::drawOpenGL(in Ray/shapeList.todo.cpp) to take into account the local transformation returned by the call:Ray::AffineShape::getMatrix. You can do this by pushingthe appropriate matrix onto the stack prior to rendering and then popping it off after you are done.For now, you can ignore the input glslProgram parameter.
- (3) Implement triangle texture mapping. To do this you will have to:
  - Modify the implementation ofRay::Triangle::drawOpenGL(in Ray/triangle.todo.cpp) to specify the texture coordinates prior to specifying the vertex positions.
  - Modify the implementation ofRay::Material::drawOpenGL(in Ray/scene.todo.cpp) method to enable and bind the texture if it is present.
  - Modify the implementation ofRay::Texture::initOpenGL(in Ray/scene.todo.cpp) method to generate the texture handle.
- (1) Implement theRay::Box::drawOpenGL(in Ray/box.todo.cpp) method to draw a box.For now, you can ignore the input glslProgram parameter.
- (1) Implement theRay::Cylinder::drawOpenGL(in Ray/cylinder.todo.cpp) method to draw a cylinder with bottom and top capsat the appropriate tesselation. For now, you can ignore the input glslProgram parameter.
  You may **not** use the gluCylinder and gluDisk functions from the GLU library to assist you with this.Instead, you will have to explicitly generate the primitives (triangles/polygons) and render them yourself. You should use the parameter Shape::OpenGLTessellationComplexity to set the

tessellation complexity.

- (1) Implement theRay::Cone::drawOpenGL(in Ray/cone.todo.cpp) method to draw a cone capped off at the bottomat the appropriate tesselation. For now, you can ignore the input glslProgram parameter.
  You may **not** use the gluCylinder and gluDisk functions from the GLU library to assist you with this.Instead, you will have to explicitly generate the primitives (triangles/polygons) and render them yourself. You should use the parameter Shape::OpenGLTessellationComplexity to set the tessellation complexity.
- (1) Implement theRay::Torus::drawOpenGL(in Ray/torus.todo.cpp) method to draw a torus.at the appropriate tesselation. For now, you can ignore the input glslProgram parameter.
  You may **not** use the glutSolidTorus function from the GLUT library to assist you with this. Instead, you will have to explicitly generate the primitive (triangles/polygons) and render them yourself. You should use the parameter Shape::OpenGLTessellationComplexity to set the tessellation complexity.
- (1)Implement mouse control to facilitate translating the camera about the scene.To do this you will have to modify:
  - Ray::Camera::moveForward(in Ray/camera.todo.cpp) to implement a translation by a distance of dist along the forward direction.
  - Ray::Camera::moveRight(in Ray/camera.todo.cpp) to implement a translation by a distance of dist along the right direction.
  - Ray::Camera::moveUp(in Ray/camera.todo.cpp) to implement a translation by a distance of dist along the up direction.
- (2)Implement mouse control to facilitate rotating around the scene scene. You should implement a crystal-ball system inwhich dragging the left mouse button rotates the viewer around the model, rotating either about the up-direction orthe right-direction. To do this you will have to modify:
  - Ray::Camera::rotateUp(in Ray/camera.todo.cpp) to implement a rotation of angledegrees, around the up-axis, about the point center.
  - Ray::Camera::rotateRight(in Ray/camera.todo.cpp) to implement a rotation of angledegrees, around the right-axis, about the point center.
- (2) Implement full scene anti-aliasing using the accumulation buffer.Hint: See the OpenGL Programming Guide for information about jittering.
  (You may need to let GLUT know that you are planning to use an accumulation buffer as well. This can be done by modifying the flags passed in to glutInitDsiplayMode in window.cpp.)
- Generate a .ray file describing a room scene with:
  - (1) Four walls, a floor and ceiling.
  - (3) A table, several chairs, etc. You may choose more interesting furnishings.
  - (1) At least one transparent surface, perhaps the table top.
  - (1) At least three texture mapped surfaces, each with a different texture.
  - (1) At least three point or spot light sources.
  - (2) A Luxo Jr. style lamp with keyboard and/or mouse controls for manipulating the joints of the lamp interactively while the spot light representing the bulb moves accordingly. Hint: see the robotic arm example in the OpenGL Programming Guide.
  - (2) A mirror. Hint: Reflect the world about the mirror and render it again through a stencil buffer.
  - (2) Shadows on at least one surface (such as the floor or table). Hint: See the OpenGL Programming Guide for the transformation which renders objects onto a plane.
  - (2) An object that responds to user mouse clicks (such as a light switch which turns on/off a light when clicked on by the user).
- Support efficient rendering using vertex buffer objects. For this you will need to generate (bind, and set) the buffer objects in Ray::Shape::initOpenGL and draw the buffer objects in

Ray::Shape::drawOpenGL.
    - (1) Implement vertex buffer objects for Ray::TriangleList.
      The implementation of Ray::TriangleList::initOpenGL has already been provided for you, with the vertex buffer object stored in Ray::TriangleList::_vertexBufferID and the element buffer object stored in Ray::TriangleList::_elementBufferID. The vertex buffer stores information about positions, normals, and texture coordinates. The number of vertices is stored in Ray::TriangleList::_vNum and the number of triangles/elements is stored in Ray::TriangleList::_tNum.
    - (1) Implement vertex buffer objects for Ray::Sphere.
    - (1) Implement vertex buffer objects for Ray::Cylinder.
    - (1) Implement vertex buffer objects for Ray::Cone.
    - (1) Implement vertex buffer objects for Ray::Torus.
- OpenGL Shading Language (See the specification for more details.)
    - (2) Write vertex and fragment shaders that implement the traditional fixed pipeline with Phong Shading using point and spot lights. (An implementation using directional lights has already been provided.)
    - (2) Adapt your Phong shader to support bump mapping.
- (?) Impress us with something we hadn't considered...

The assignment will be graded out of **30 points**. In addition to implementing these features, there are several other ways to get more points:

- (1) Submitting one or more images for the art contest.
- (1) Submitting one or more .ray files for the art contest.
- (1) Submitting one or more shader files for the art contest. You may consult online resources (make sure to do reference) but the shaders must be your own creation.
- (2) Winning the regular art contest,
- (2) Winning the .ray file art contest,
- (2) Winning the shider file art contest.

It is possible to get more than 30 points. However, after 30 points, each point is divided by 2, and after32 points, each point is divided by 4. If your raw score is 29, your final score will be 29. If the raw scoreis 33, you'll get 31.25. For a raw score of 36, you'll get 32.

OpenGL Programming Guide: The Official Guide To...

21f597057a