Lab 4: 3D Rendering

3D Computer Graphics

Introduction

Import the archive file `3DCG_Lab4.jar` into Eclipse by selecting

```
File > Import... > General > Existing Projects into Workspace
            > Next > Select archive file > Finish
```

If you have set up JOGL correctly in Lab 1, you can simply add the user library jogl-2.0 to this project as follows: right-mouse click on the project's name in the Package Explorer window and select

```
Build Path > Add Libraries ... > User Library > jogl-2.0.
```

Exercise 1

a) Copy your `Face` and `Mesh` class from Lab 3 to the package `geomobj/mesh` in the current project.

b) Copy your `Point`, `Vector` and `Quaternion` class from Lab 3 class to the package `util` in the current project.

c) Copy your `UserEventMediator` class from Lab 3 to the package `ui` in the current project.

d) Copy your `Camera` class from Lab 3 to the package `renderer` in the current project.

The project of this Lab contains one graphical application with `app1.cfg` as configuration file.

e) Open `app1.cfg` in the `apps.app1` package. The first line of this configuration file mentions the name of the file (`simpleScene.sdl`) which contains information about the scene to be rendered.

f) Open the file `simpleScene.sdl`. It describes a very simple scene with a white background and a simple square.

The aim of the following exercises is to implement our own renderer, which makes uses of the ray tracing technique. If you do this successfully, you will be able to render this simple scene.

Exercise 2

In this exercise, we add support for the view volume of a camera.

Remember from Lab 3 that the configuration file of a graphical application contains key/value pairs which are stored in a `java.util.Properties` object (`prop`). This `Properties` object is passed on to all classes which require the configuration settings (for example the camera).

a) Look again at `app1.cfg`. The last three lines specify the world window. These data should be stored in the `Camera` class.

b) Add three public floats to the `Camera` class: `distance`, `width` and `height`.

c) Initialize these instance variables correctly in the constructor of the `Camera` class by reading the appropriate values from the `Properties` object.

Exercise 3

a) Study the implementation of the `Ray` class in the `renderer` package.

b) Add the correct implementation of the `getPoint` method (of the `Ray` class) which returns the 3D coordinates of the point on the ray corresponding to a given floating point value t.

c) Study the implementation of the `HitInfo` class in the `renderer` package. Make sure you know the meaning of its instance variables.

d) The `Intersection` class stores information about the hits between a ray and a 3D object. Study the implementation of this class (`renderer` package). You should understand the meaning of all its methods before you proceed.

Note that the `Intersection` class stores information about ALL hits between a ray and a 3D object.

Our renderer will only be able to visualize 3D objects if they implement the `GeomObj` interface.

e) Open the `GeomObj` interface in the `geomobj` package.

This interface contains one method: `intersection`. This method returns an `Intersection` object containing all the hits between the 3D object implementing this method and the given ray. This implementation should satisfy some conditions as mentioned in the documentation of the `intersection` method.

e) Carefully read the documentation of the `intersection` method.

At present, only the `Square` class implements this `intersection` method.

Exercise 4

The `RayTraceRenderer` class in the `renderer` package is responsible for rendering our scene with the ray tracing technique. Its method `render` is called every time the canvas needs to be repainted. You do not have to understand the current implementation. It is sufficient to know that it clears the image and resets OpenGL.

```
GL2 gl = drawable.getGL().getGL2();
gl.glClear(GL.GL_COLOR_BUFFER_BIT);

GLU glu = new GLU();
gl.glMatrixMode(GL2.GL_PROJECTION);
gl.glLoadIdentity();
glu.gluOrtho2D(0, nCols, 0, nRows);

gl.glMatrixMode(GL2.GL_MODELVIEW);
gl.glLoadIdentity();

gl.glDisable(GL2.GL_LIGHTING);
```

a) Add code to the implementation of the `render` method at the indicated place. Use the pseudo code mentioned in the slides of this Lab as a guide.

b) Note that this pseudo code mentions "openGL commands which set the colour of pixel (r,c) to col". You can use the following lines of code to achieve this:

```
gl.glColor3f(col.r, col.g, col.b);
gl.glRecti(c, r, c+1, r+1);
```

The code added to the `render` method contains a call to the `shade` method of the `RayTracer` class. We will implement this method in the next exercise.

Exercise 5

a) Open the `RayTracer` class in the `renderer.raytracer` package.

b) The `getBestIntersection` method of the `RayTracer` class will loop over all objects in the scene and call their `intersection` method to get information about the hitpoints between the given ray and the different objects in the scene. The `getBestIntersection` method should return the `Intersection` object with the lowest bestHitTime. If none of the objects in the scene intersect with the given ray, an `Intersection` object with an empty arraylist should be returned. Provide an implementation for the `getBestIntersection` method.

c) The `shade` method of the `RayTracer` class will return the colour of the hitpoint closest to the camera. Implement this method as follows: first, it should call the `getBestIntersection` method of the `RayTracer` class to get the `Intersection` object with the lowest bestHitTime. If this best intersection object has no hits, the background colour should be returned. Else, return a red colour.

   Of course, the latter means all shapes will be completely red on the screen. This is a shortcut to allow you to check your implementation as quickly as possible. We will come back to this method and provide a better implementation later on.

Exercise 6

a) Run `App1.java` and check whether your implementation of a raytracer works. You should get an image of a red square on a white background.

b) Use the "f", "b" and arrow keys to check whether you can still animate the camera.

Exercise 7

In this last exercise, we add support for rendering a generic sphere with our raytracer.

a) Create a class `Sphere` in the `geomobj` package. This class should extend the abstract `Shape` class. Provide a correct implementation of the `intersection` method in the `Sphere` class. Use the information provided in the slides of this Lab.

b) Add a third line with 1 word "sphere" to the `simpleScene.sdl` file.

c) The `SceneFactory` class in the `scene` package reads an sdl file and uses the data in this file to create an object of the `Scene` class. Study the implementation of the `SceneFactory` class.

d) Adapt the `createShape` method of the `SceneFactory` class so that it recognizes the "sphere" token which we added to the `simpleScene.sdl` file.

e) Run `App1.java` again. Do you get the expected result?

To be continued ...