Assignment 5: Texture Mapping and Exploded Views!

Out: 5th November 2017 **Due: 13th November 2017 at 8:59pm**

In this assignment you will extend your program from Assignment 4 by adding two features: texture mapping and creating an illustrative, exploded view of your scene graph using bounding boxes. You will retain all other features that you have implemented in previous assignments.

Look at the provided humanoid-lights-textures.xml and face-hierarchy.xml for examples.

Part 1 : Texture Mapping (40 points)

In this part, you must add support for texture mapping in your scene graph.

Part 1.1: Supporting texture mapping (30 points)

This entails the following:

1. Verify that textures from the XML files are being parsed and stored correctly in your scene graph. Please use the XML parser files attached with this assignment to add support to the required tags.
2. Modify the renderer so that it sends relevant texture data to the shaders. Use the shaders from the example posted on blackboard for lights and texture mapping, and modify your lighting shaders accordingly.
3. Try a simple scene first to verify that texture mapping works.

Part 1.2: Textures in your scene (10 points)

Modify your scene file so that it uses textures. If you do not want to texture a certain object, make it default to a white texture (you must supply a white texture file!). Adding a white texture will show output from only lighting (look at the shader code to see why).

You are required to use at least 2 different texture images, and must texture at least one part of your animating model. You are also required to keep at least one object untextured.

Part 2: Exploding Views (50 points)

An exploded view is often used to illustrate the inner structure of a 3D model. Exploded views are common in illustrating structures of machines, or even in furniture assembly instructions. Here is an example (https://en.wikipedia.org/wiki/Exploded-view_drawing#/media/File:Gear_pump_exploded.png) .

In this part you will write a function that will produce an exploded view scene graph from an original scene graph. Your original and exploded scene graph will co-exist, so that your program can render either one.

The algorithm for creating an exploded view of a scene graph uses axis-aligned bounding boxes. Specifically, you will add an axis-aligned bounding box to every node in your scene graph (in its own coordinate system), and then use them to create an exploded view.

Part 2.1: Bounding boxes (20 points)

Each node will store a bounding box. The sides of this bounding box will always be aligned with the coordinate planes of the coordinate system of that node. You must write a function that updates these bounding boxes during every frame, as they will change with animation.

Calculating the bounding box uses a different algorithm for each type of node:

1. Leaf node: this is the simplest. The PolygonMesh class directly provides functions that return the bounds of its bounding box.
2. Group node: The bounding box of a group node is simply a box that tightly encloses the bounding boxes of all its children. Since no transformations are involved, all these boxes already align with the group's coordinate system planes. This means that computing this bounding box is a simple matter of computing minima and maxima in each dimension.
3. Transform node: The transform node's bounding box is a box that tightly encloses the **transformed** bounding box of its child. To do this, you must transform the vertices of its child's bounding box and then find a box that contains all of them.

What to do:

Write a function that descends down the scene graph and updates all bounding boxes (carefully determine what the order should be). Call this function when drawing the scene graph, so that bounding boxes are updated in each frame.

Part 2.2: Exploded view (20 points)

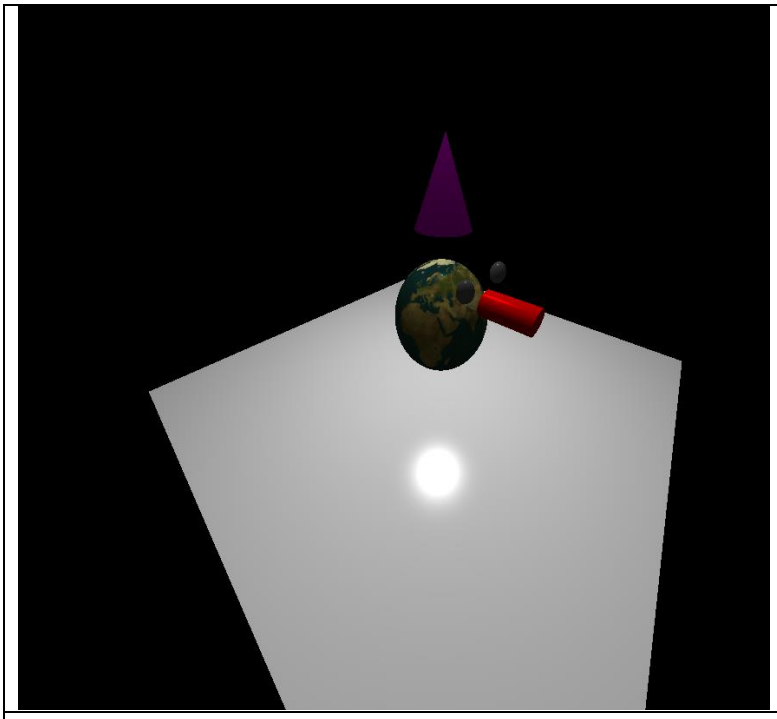
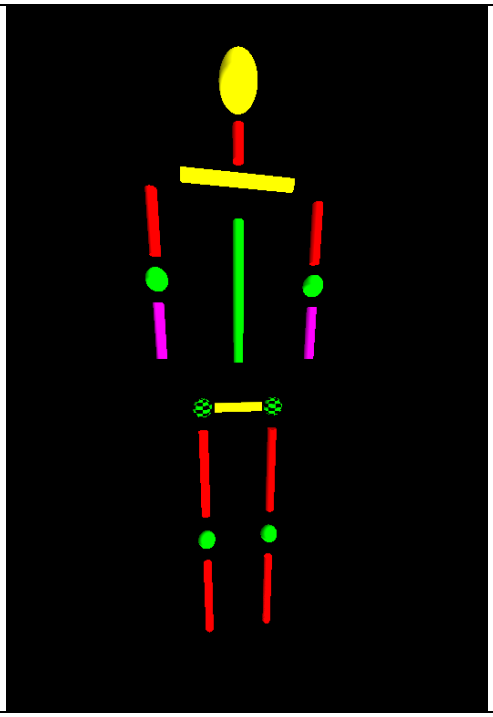
Imagine a group node's bounding box: it contains the bounding boxes of all its children. An exploded view of this node is formed by "pushing" all its children radially outward from the center of its bounding box.

1. Compute the center C_g of the bounding box of the group node 'g'.
2. Let N_g be the node which is the exploded version of the sub-graph 'g'. We wish to compute N_g.
3. For each child 'f':
    a. Compute the center C_f of the bounding box of 'f'.
    b. Determine the vector V = C_g -> C_f.
    c. Determine a suitable distance along V by which 'f' must be pushed. This is a simple translation T_f.
    d. Compute a new transform node N_f with T_f, with a child the exploded version of 'f'.
    e. Add N_f as a child to N_g
4. Copy other data from g to N_g (i.e. lights).
5. Return N_g as the result.

Exploding a transform node is simpler: it is simply a copy of the original transform node, but with the exploded version of its child. Exploding a leaf node is the simplest: it is a copy of the original node.

Making copies of nodes in each case is necessary so that the exploded scene graph is independent of the original scene graph. This will allow you to easily switch between rendering them!

Here are some illustrations of exploded views:

	
Exploded face-hierarchy	Exploded humanoid

What to do:

Write a function in your nodes that descends the scene graph (again, in what order?). Specifically for a node, this function returns an exploded view of the scene graph rooted at this node.

Now write a function in your scene graph class that uses the above function, creates and returns the exploded view scene graph.

In the view, you can now determine which scene graph to render.

Part 2.3: Program setup (10 points)

Set up your program so that pressing the 'E' key shows your scene in exploded view, and pressing 'E' again toggles back to the original view. In the exploded view, the camera may be keyboard-controlled or static. In the exploded view you must choose a suitable camera position so that the scene can be visualized.

Extra credit (5 points)

Make an animation video of your final rendering (in mp4 or avi format). The video should showcase your achievement in this assignment. Look at the description on blackboard and the example code.

Program preparation and README file (10 points)

Prepare your program as specified above (switching between the exploded view and original view)

The README file should summarize what features work in your program. It should also include a link to the extra credit video, if you have done it.

What to submit

Submit the IntelliJ/Qt project folder set up correctly with your scene files you used as a zipped file (make sure you include the README file). Also include a screen shot in jpg or png format that shows your exploded view (for your final scene, or another simpler scene other than the provided screen shots).