Assignment 3: Robot Squad -Due Fri. Mar 2nd 12:59 p.m.

In this project you will need to implement a scene graph to render an army of Android-inspired robots and make them march along a curve. As always, a video demo will be posted on piazza. You can reuse the code from the assignment 2 for this project.

1. Sky Box

Start with code that uses your trackball code, and modify it to control the camera instead. (If you didn't get that to work, keyboard controls will suffice.)

Create a sky box for your scene with the robots. A sky box is a large, square box which is drawn around your entire scene. The inside walls of the box have pictures of a sky and a horizon. Sky boxes are typically cubic, which means that they consist of six square textures for the six sides of a cube. Here is a great tutorial for sky boxes in modern OpenGL.

Here is is a nice collection of textures for sky boxes, and here is an even bigger one.

Draw a cubic sky box and make it extremely big. For instance, by giving it coordinates like -1000 and +1000.

Make sure single-sided rendering (triangle culling) is enabled with these lines somewhere in your code to ensure that you will never see the outside of the box (this assumes that your sky box is defined with the triangles facing inward):

```
glEnable(GL_CULL_FACE);
glCullFace(GL_BACK);
```

Use the following settings for your texture after your first `glBindTexture(GL_TEXTURE_CUBE_MAP, id)` for correct lighting and filtering settings:

```
// Make sure no bytes are padded:
glPixelStorei(GL_UNPACK_ALIGNMENT, 1); // Deprecated in modern OpenGL - do not use!

// Select GL_MODULATE to mix texture with polygon color for shading:
glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE); // Deprecated in modern OpenGL - do not use!

// Use bilinear interpolation:
glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_MAG_FILTER, GL_LINEAR);

// Use clamp to edge to hide skybox edges:
glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
```

To familiarize yourself with texture mapping in OpenGL, we provide sample code, which loads a PPM file and uses it as a texture for a quad. If you decide to use one of the above referenced sky box images, you will have to convert them from JPEG to PPM format. The free image processing tool IrfanView for Windows will do this for you. Alternatively, you can use a third party library such as SOIL to natively load JPEG images.

2. Scene Graph Engine

To connect the parts of the robot (head, torso, limbs, eyes, antennae), we need to first implement a simple scene graph structure for our rendering engine. This scene graph should consist of at least three nodes: `Node`, `Transform` and `Geometry`. You are free to add more scene graph node types as you see fit. The OBJ file of the robot can be found in the next part.

- Class `Node` should be abstract and serve as the common base class. It should implement the following class methods:
    - an abstract draw method: `virtual void draw(Matrix4 C)=0`
    - an abstract `virtual void update()=0` method to separate bounding sphere updates from rendering
- `Transform` should be derived from `Node` and have the following features:
    - store a 4x4 transformation matrix M
    - store a list of pointers to child nodes (`std::list<Node*>`)
    - provide class methods to add and remove child nodes (`addChild()`, `removeChild()`) from the list
    - its draw method needs to traverse the list of children and call each child node's draw function
    - when `draw(C)` is called, multiply matrix M with matrix C.
- `Geometry` should be derived from `Node` and have the following features:
    - set the modelview matrix to the current C matrix
    - an initialization method to load a 3D model (OBJ file) whose filename is passed to it (`init(string filename)`. Your OBJ loader from project 2 should work.
    - have a class method which draws the 3D model associated with this node.

3. Walking Android Robot

Now that we have the scene graph classes, it is time to put them to work, and build a robot with them.

Here are some robot parts: head, body, limb, eye, antenna. You will find the OBJ files in this ZIP file.

Build your own robot using the addChild methods. Use at least 3 different types of parts for your robot (e.g., body, head and limb). In total, your robot needs to consist of at least 4 parts, 3 of which need to be moving independently from one another and they need to be connected to the 4th part.

This is an example of a valid robot with two antennas, two eyeballs, one head, one torso and 4 limbs (2 legs and 2 arms):

Once you've created your scene graph, you need to get your rendering engine ready to recursively traverse the scene graph for rendering by creating a root node of type Group and calling its draw() function with the identity matrix as its parameter.

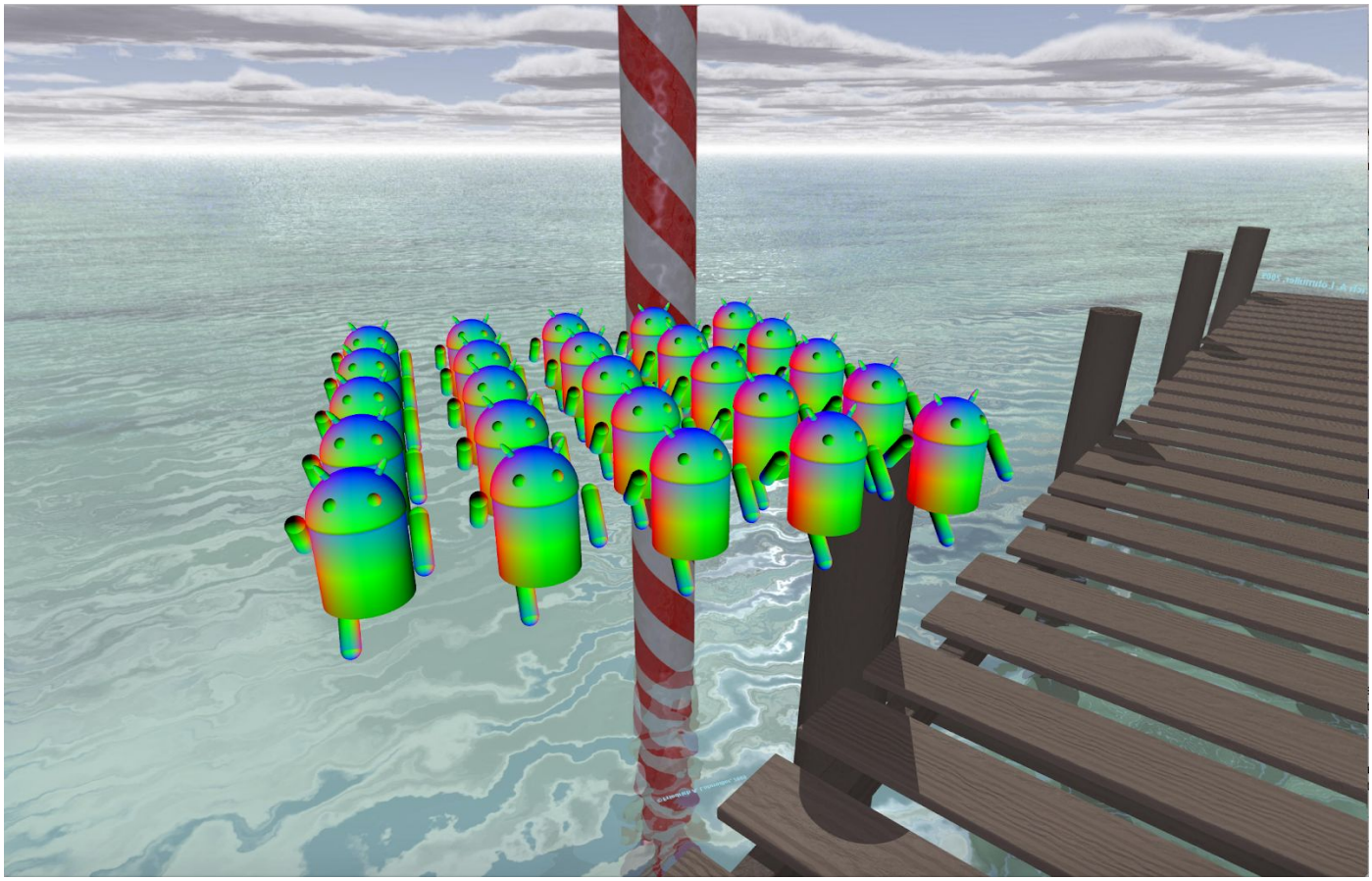
Animate the robot to make it look like it is walking, by changing the matrices in the Transform nodes.

4. Robot Squad

Test your implementation by constructing a scene which consists of a squad of robots, at least 25. The robots can all be identical clones.

- Distribute the robots on a 2D grid (i.e., place them on a plane with uniform spacing). For 25 robots, use a 5x5 grid.
- Enable the animation for all the robots so that they look like they are walking.
- Enable your rotation and scale routines (keyboard or mouse) to allow rotating the grid of 3D objects and zoom in or out.

This image illustrates the grid layout of the robots:



5. March along the curve

Bezier Curves

- Create 5 connected cubic Bezier curves (i.e., 4 control points per curve).
- Draw the Bezier curves by approximating them with at least 150 straight OpenGL segments each.
- You do not need to draw the control points.

March along the Curve

- The Robot Squad should move along the curve you just created.
- The center robot in the squad should be always on the curve during the movement.
- The motion should be smooth

*imagine the small object in the graph is your robot squad

6. Submission

Please zip all your files except obj files and submit it on Gradescope by 12:59 p.m. on March 2nd.

Grade Session:	March 2nd 1:00 - 3:00 p.m.
Late Submission:	March 9th 11:59 p.m.
Late Grade Session:	March 9th 1:00 - 3:00 p.m.