HOMEWORK #3

THE ART CRITIC (PART 2)

In this assignment we will construct an arbitrary 3D world to experiment with a simple shadowing algorithm. This assignment may be done in teams of two. **Due October 27, 2009 by 11:59PM**. The turnin name is cs433_hw3.

Figure 1. Shadowing of pictures in a gallery of art.

We revisit the art critic analogy. The critic has just walked into a great gallery, where numerous paintings are hanging on wires from the ceiling. Three colored spotlights illuminate the room. Your objective is to construct this gallery from a scene descriptor file and to apply a simple z-buffer algorithm to determine which pictures are illuminated by each of the lights (Fig.1). Specifically, you should:

(1) (50 pts) **Construct a 3D world from descriptor file:** The gallery should be constructed at program startup from the scene description given in $config.ini$, which defines the lights, viewpoint, and pictures. You should assume that there are up to 3 lights, and that each light is monochromatic on either the Red, Green, or Blue color channel, exclusive. There is exactly one viewpoint and an arbitrary number of pictures (up to small n) scattered about the room. The pictures are always orthogonal to the Z axis.

(a) The format of the scene descriptor file is as follows:

`viewpoint` $pos_x\ pos_y\ pos_z\ look_x\ look_y\ look_z\ fov$

`light` $channel\ pos_x\ pos_y\ pos_z\ look_x\ look_y\ look_z\ fov$
`light` $channel\ pos_x\ pos_y\ pos_z\ look_x\ look_y\ look_z\ fov$
`light` $channel\ pos_x\ pos_y\ pos_z\ look_x\ look_y\ look_z\ fov$

`picture` $file1.jpg\ width\ height\ pos_x\ pos_y\ pos_z$
`picture` $file2.jpg\ width\ height\ pos_x\ pos_y\ pos_z$
$\vdots$
`picture` $filen.jpg\ width\ height\ pos_x\ pos_y\ pos_z$

where

- $pos_{x,y,z}$ are reals defining the position of an object in world space
- $width, height$ are reals defining dimensions of a picture
- $look_{x,y,z}$ are reals defining a view reference point in world space
- fov is a real defining field of view in degrees
- $channel$ is a character (R,G,B) defining the light channel

(b) For your convenience, a # in your scene descriptor file should designate a comment line. You should be able to add as many of these lines as you wish, and have your program ignore them.

(c) Render the scene using perspective projection, with Z-buffering enabled to provide the correct depth occlusion cues.

(d) Allow the viewpoint to be moved interactively. The up and down arrow keys move the viewpoint forward along the line of sight, parallel to the X-Z plane. The left and right keys rotate the viewpoint about its "Up" axis.

(e) As an invariant, let the Y-axis define "Up" in this world, such that the user's walk is constrained in the X-Z plane.

(f) Assume that the pictures are always upright and orthogonal to the Z axis. You may ignore the existence of the ceiling and floor, as well as the wires from which the pictures are suspended. A background is welcomed, but not necessary for full credit.

(g) Mark the location of each spotlight in the 3D world using any convenient method, e.g. via simple geometric models or block pixels. You should use lines, cylinders, or cones, emanating from the light source, to indicate spotlight direction. Set the color of the model equal to the light's emissive color. This will simplify your debugging and our grading!

(2) (50 pts) **Cast shadows using Z-buffer algorithm:** You should determine which pictures are in shadow with respect to each light source and illuminate the scene accordingly. Use the simple Z-buffer algorithm discussed in class to determine visibility; ray casting solutions are *not* acceptable, as they are impractically slow for realtime rendering.

(a) For simplicity, assume that a picture is either completely in shadow or completely illuminated by a spotlight l. In particular, a picture is in shadow for l if and only if *no part* of that picture is visible from l. This will allow each picture to be uniformly lit (or unlit) by each light source.

(b) Illuminate each picture using the appropriately colored incident light. Each spotlight, for which the picture is not in shadow, adds one monochrome color component. For example, if all three lights illuminate the picture, the incident light is white. If the picture is in shadow to green but not for red and blue, the incident light is purple. Thus, simply modulate the color channels of each picture by 0 or 1 on the basis of your shadow test. If the light for a particular channel does not exist in the scene descriptor, the modulation value for that color channel is 0 by default.

(c) Lights should be animated to move parallel to the Z axis, so that the shadows they cast will change dynamically over time. A light's orientation, i.e. direction, will never change. Let the lights start out in a stationary state at the position given in the descriptor file. The user can then choose a light for movement by pressing the letter corresponding to the light's color (i.e. "R", "G", or "B"), followed by the "{" or "}" button to increment its velocity. For the active light, let each "}" key press add some small, positive stepping dV to the velocity, and, conversely, let "{" add a negative stepping $-dV$. Pressing "p" should reset that light's velocity to 0 (a stationary position). Be sure to update the light's graphical representation (see above) as it moves !

(3) **Use the power of OpenGL:** You may use any and all available functions of OpenGL, GLU, and GLUT to complete this assignment. In debugging your shadow algorithm, it's often good to start by rendering to the screen what each light actually *sees*. Texturing in this assignment may be done as in the "GL" mode of HW2. Note that hardware lighting and materials should not be needed for this assignment. You may find the following to be useful:

(a) *gluLookAt*() and *gluPerspective*() can be used to define the view parameters of your spotlights for the shadowing casting algorithm.

(b) The primitive geometry functions of the GLU package, such as *gluSphere*(), may be useful for marking light locations and direction vectors. Alternatively, you may draw lights with *GL_POINTS* and *GL_LINES*.

(c) Read back the contents of an OpenGL frame buffer into an array using the function *glReadPixels*(). The format parameter should be set to *GL_RGB* for the color component, or to *GL_DEPTH_COMPONENT* for the Z-buffer component.

(d) (at most 8 points bonus) It would be nice to have some keys controlling parameters. For example change the speed and direction of motions of the viewpoint/lamps. If you opt to add these options, then hitting the 'h' key should print a help list of the other keys you have defined. Also add some details in a 'readme' file.

(e) (at most 15 points) Rather than treating each picture uniformly as far as shadows from other pictures are concern, apply a quad-tree like subdivision of each picture. Alternatively, and less challenging least subdivide each picture to 5×5 smaller pictures and treat each one individually.