HOMEWORK #2

THE ART CRITIC

This assignment may be done in teams of two. **Due October 8, 2009 by 11:59PM**. The turnin name is cs433_hw2. This homework is a continuation of HW1, with major additions.

Figure 1. Scene concept: perusing a wall of art.

For this assignment, you will simulate the view of an art critic as she takes a virtual tour along the wall of an art gallery. Imagine that beautiful works of art line this wall. Call them $pic_1, pic_2 \ldots pic_5$, and include them with your submission. Your program should simulate the art critic's view as she walks slowly along the wall, examining each picture in turn as shown in Fig.1. In particular, your scene construction must consider the following:

- The critic's walk is along a straight line parallel to the wall. The critic is looking both at the wall and forward, such that her view axis forms a 45° angle with the wall's surface.
- The pictures should be large enough that, as the critic walks along the wall, one picture should fill much of her view as she gets close to it. As the critic

moves along the wall, there will also be times when two pictures are indeed seen, as pic_i moves out of view and pic_{i+1} comes into view.

- The wall should be infinitely long, so that when the critic reaches the last picture pic_n, the series restarts with pic_1, then pic_2, etc. Although the wall has both top and bottom edges, you can ignore the existence of the ceiling and floor.
- Use perspective projection to render the critic's view. As the view axis forms a 45° angle with the wall, perspective foreshortening should be observed in your scene for both the wall and pictures.
- The construction of your scene should *resemble* Fig.1. Nonetheless you are free to make individual decisions about the critic's field of view and walking speed, as well as the size of the pictures, their placement on the wall, etc. Excluding the walking motion, you may assume that all other parameters of the scene remain constant.

Your program should render the scene using the following draw modes:

(1) (60 pts) **Basic Mode:** If the **'b'** key is pressed, only pixel-level GL operations are allowed. You should only plot colors pixel-by-pixel using glVertex and glColor, as in HW1. To implement "Basic" mode, you should consider only what is seen by each pixel in the window space. Imagine a line (*i.e.* ray) extending from the center of projection through a pixel on the projection plane into the view volume (Fig.2). Determine the intersection of this line with the wall full of pictures. The color of the pixel is the color of the wall, or a picture on it, at the point of intersection. Do not worry too much about the efficiency of your solution here.

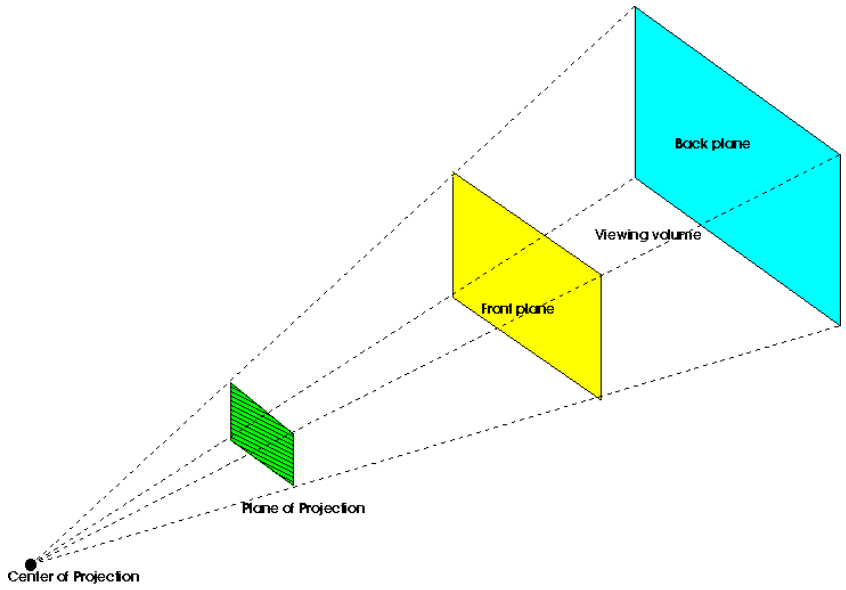

Figure 2. Geometric construction of view volume and projection plane.

(2) (40 pts) **GL Mode:** If the **'g'** key is pressed, your program should draw the scene using the hardware-accelerated GL pipeline. Your scene construction should be similar to Basic mode, but it does not have to be identical. For this mode you are free to use any GL or GLU functions that you find convenient to draw the scene. For instance, you may exploit GL to maintain the view and perspective transformations and to perform polygon filling. You may use textured polygons to represent the gallery pictures. Note that your GL mode should run at interactive frame rates! For specifics on initializing your program for texturing and polygon filling, please consult the GL tutorials and documentation. You may also find the following functions useful:

gluLookAt: defines the eye (camera) pose in world coordinates
glFrustum: defines the sides of a perspective view volume
gluPerspective: builds a view volume based on FOV, aspect ratio
glBindTexture: activates an image ID for texture mapping
glTexCoord2{f,d}: defines texture coordinates for vertices

(3) (25 pts) **For Grads**: Extend the **Basic** mode by anti-aliasing the viewing window using a jittered super-sampling approach. Instead of casting 1 ray per pixel, you should cast k rays. The number of rays k should be selectable from the set $\{1, 4, 9\}$ at runtime using the corresponding numeric keys. Let $k = 1$ by default. Also, jittering should only be used when $k > 1$.

To implement your algorithm, imagine that each of the k rays originates from a unique cell in a regular $\sqrt{k}$ by $\sqrt{k}$ grid overlaying each pixel i on the projection plane. The color for i is then given as the average of the colors returned by its k sub-pixel rays. Further, you should *jitter* each of the k rays around the center of its respective grid cell using a noise function (rand is sufficient). With jittered anti-aliasing, you should see a significant improvement in image quality over the $k = 1$ case, albeit at significant computational cost. If you're curious as to why it works, consider that we're approximating the same stochastic process as used by the human visual system to de-alias what you see (*i.e.* a Poisson process). If interested, see Cook's classic paper [1].

[1] Cook, Robert L, "Stochastic Sampling in Computer Graphics," *ACM Transactions on Graphics (TOG)*, Vol.5(1), 1986, 51-72.