HOMEWORK #3

This assignment may be done in teams of two. *Due October 25, 2008 by 11:59PM.* This assignment is a continuation of hw1 and hw2, but with more reliance on the 3D rendering capabilities of OpenGL, including the Z-buffer, transformations with homogeneous coordinates in 3D, and manipulation of polygons rather than pixels. The main purpose of this assignment is to practice **shape simplification,** specifically its relationship to QuadTrees (QT). Please read the following description carefully!

(1) The scene contains 3 rotating plates, designated as p_1, p_2 and p_3. On each side of a plate, there is one of 6 images, denoted as pic1.jpg...pic6.jpg. So for instance plate p_1 has on one of its sides pic1.jpg, with pic2.jpg on the other side, and so on for plates p_2 and p_3. The image files should be read from the directory containing the executable. Keep the image naming consistent with the conventions given above.

(2) You may use an orthographic (orthogonal) projection transformation.

(3) Construct a "virtual" viewing window frame, F, which is located at your projection plane (i.e. at the viewplane). Draw F centered in your window, such that F is slightly smaller than the window size. F is simply the outline of a square on the screen; it is composed of 4 lines (you may construct it using GL_LINES). Because F is "anchored" to the viewplane, the location of F should not appear to move or rotate. However, the size of F can be changed via key presses as discussed below. The intent here is to use F as a virtual drawing boundary, and to ignore any regions of the plates *entirely* outside of F, as discussed below.

(4) Rescale your images to a size of exactly 512×512 pixels. You may resize the images in a pre-process step, using your favorite image editing program.

(5) Let *R(x,y)*, *G(x,y)* , and *B(x,y)* denote the RGB value of a given pixel, and let *I(x,y)*, the **intensity,** be defined as *I(x,y)=R(x,y)+G(x,y) + B(x,y).*

(6) **Simplification using Quadtrees:** Recursively subdivide each image into regions using a QT, and then use the QT to render those image regions *inside* the frame F. Let the scalar T denote the sub-division threshold, which is initialized to 2^4 (the user can change it - details below). The process of subdividing an image using a QT is as follows:

   (a) As in any QT, every node v is associated with a region *R(v)* of the picture.

   (b) The region *R(root)* is the whole picture. That is, it contains all pixles $[0..511, 0..511]$

   (c) Every non-leaf node v has 4 children, marked *NW(v)*, *NE(v)*, *SW(v)* and *SE(v)*. For example, $R(NW(v))$ is the region $[0..255, 255..511]$.

   (d) You need to split the region of v iff

$$MaxIntensity(v) - MinIntensity(v) > T$$

Here T is the threshold mentioned above, and MaxIntensity (resp. MinIntensity) is the maximum (minimum) intensity of any pixels in *R(v)*. For the sake of performance, you may **limit your sub-division depth to 7 levels**, including the root. For instance, each 512×512 image has a minimum region size of 8×8 pixels.

(e) For every node v, you should store at least the following fields:
  (i) pointers each of v's 4 children (for a leaf node, these simply point to NULL)
  (ii) The fields MaxIntensity(v), MinIntensity(v), containing the maximum and minimum intensities among all the pixels in *R(v)*
  (iii) The field *AvgR(v)*, *AvgG(v)*, *AvgB(v)*, containing the average Red, Green and Blue values among all the pixels in *R(v)*

(f) When your program runs, construct six quadtrees – one per plate side x 3 plates.

(g) To change the value of T, the use can press either 't', which decreases T by half, or 'T', which doubles T. In constructing the QT, you may need to split nodes, but there is never a need to merge them.

(h) **Render each plate in the scene as follows:**
  (i) For every node v of the QT, if the projection of *R(v)* is *fully outside* the frame F, ignore it – do not render that region. (This is a simple incarnation of region *culling*.)
  (ii) Else (*R(v)* is fully or partially inside the frame F) do
    (A) If v is not a leaf and *MaxIntensity(v)-MinIntensity(v)*$>T$, recurse to each of v's four children.
    (B) Else if (v is a leaf) or *MaxIntensity(v)-MinIntensity(v)*$>T$, construct a polygon covering *R(v)*
    (C) Draw *R(v)* into the window (you may render it using GL_QUADS). Assign a 3-color to the polygon equal to (*AvgR(v)*, *AvgG(v)*, *AvgB(v)*) for *R(v)*.
  (iii) **Hint**. This rendering algorithm is designed to throw away regions outside of the virtual window F, and to render the largest regions inside F that satisfy the given threshold constraint T. This corresponds to drawing the largest, and consequently fewest, QT regions (aka polygons) that satisfy T. Note that if T=1024, then only one polygon per image should be displayed, and if T=1, then it is likely that the lowest level in your QT will be displayed.
  (iv) **Hint**. For efficiency's sake, you may choose to construct your QTs to maximum sub-division depth at program start-up, rather than rebuilding or modifying them each time the user changes T. Make sure that your rendering algorithm is always drawing the *largest* regions satisfying T, rather than blindly recursing to the bottom of the QT each time.

(7) **Further Instructions:**

(a) The plates should rotate as in hw1 and hw2. They should also change their positions in space, for example via random motion or via keyboard commands (please document in your readme). Parts of all plates should be displayed at all time.

(b) Occlusions between polygons can be resolved using the OpenGL z-buffer.

(c) When the user presses the 'S' key, the frame F should increase its edge-length (both horizontally and vertically) by 20%. The pictures should be re-rendered after this operation. The center of the frame is not changed. Hitting the 's' key has the same effect, but the edge-length should be decreased by 20%.

(d) Add a "Freeze" *toggle* for each individual plate: pressing the '1' key should hold plate 1 stationary in space, using its current transformation at the time of the key press. The plate should stay fixed at that position and orientation in space until '1' is pressed again to unfreeze it. At the instant of the second key press, the previous motion (i.e. translation and rotation) should be resumed. Implement the freeze/unfreeze toggle for plates 2 and 3 as well, using keys '2' and '3', respectively.

(e) **Yes, you may now use 3D OpenGL!** You may now use all of OpenGL's basic 3D display and manipulation capabilities. Feel free to mix and match with your existing code base, as you desire. We do recommend that you setup the orthogonal projection transform using *glOrtho()*. The function *glOrtho()* works in a similar way to *gluOrtho2D()*, but requires that you to specify a viewing *volume* rather than a viewing window. You may also use OpenGL's built-in modelview matrix to transform and rotate, as well as OpenGL's facilities for drawing polygons and lines. An updated example program, paintdemo2, will be posted online which demonstrates these OpenGL capabilities.

(f) As usual, submit all code and metadata files, as well as a readme describing extra content, known bugs, keyboard mappings, and anything else of relevance to submission. The turnin name is cs433_hw3.