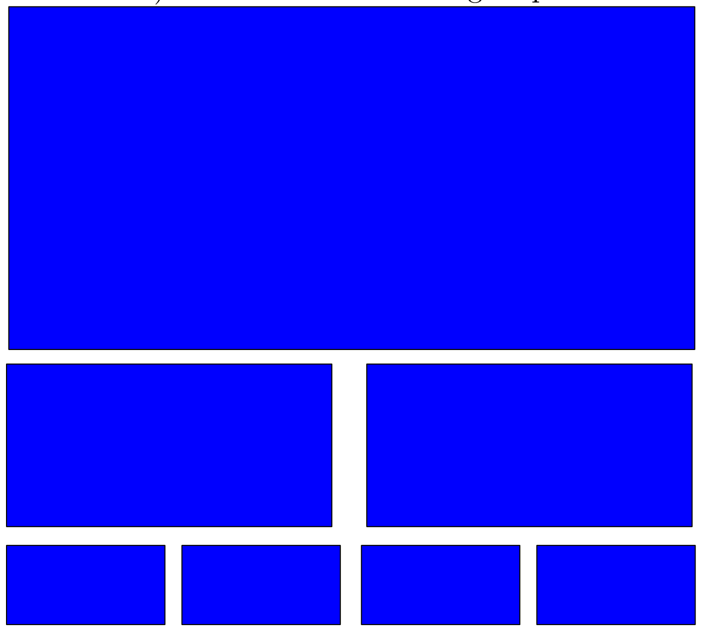
HOMEWORK #4

This assignment may be done in teams of two. *Due Thu 11/13, 2008 by 11:59PM.* The turnin name is cs433_hw4. The objective of this assignment is to articulate a hierarchical model using a scenegraph and to explore a simple shadow-casting algorithm. In this assignment, you'll be creating and rotating a virtual *mobile* like the one in the image below, although less sophisticated. A mobile is a toy that you commonly find dangling from the ceiling above a baby's crib.

(1) A mobile in this homework consists of a set of dangling plates. Each plate is a rectangle whose width/height ratio is 2:1. Each plate has two child plates

hanging from it, i.e. left and right child plates. Each child plate has half the height and width of its parent plate (in the figure some gap appears between the plates for demonstration). The **depth** of the mobile is defined as the number of levels of plates. Assume the depth is initialized to 3, but that the user can increase it. For instance, 3 levels means that from the root plate, there are 2 child plates, each with two child plates themselves (4 descendant plates under the root in total).

- Create a data structure that you can use to instance an arbitrary mobile. Keep in mind that the transformation (pose) of any child plate in the scene is dependent upon the pose of the plate directly above it. In other words, the pose of a child plate is dependent upon the pose of its parent, the parent's pose depends on that of its parent, etc. You should consider a data structure that will allow you to efficiently represent this hierarchical model and its associated hierarchy of transformations (hint: scenegraph). For instance, a simple scenegraph data structure might use an array A of records (structs), where the each plate is stored at a cell A[i], its left child is in A[2i] and its right child in A[2i+1].
- In our scene, the mobiles live in a cube [0..1000,0..1000,0..1000] aligned with the xyz coordinate system. Make sure to normalize the parameters to fit your display window. If you prefer, you may choose to draw the cube with polygons or to outline it in wireframe (i.e. using GL_LINES).
- Create 2 independently movable mobiles, each suspended from a point on the "ceiling" of the cube, one at (500, 1000, -500) (that is, height z =1000, distance z =500 from the plane z=0) and the other at (500, 1000, -1500). The width of the largest plate in both mobiles is 1001.



(2) The user should be able to interactively manipulate the pose (specifically, rotation) of the plates in each mobile, as well as to add or remove levels from the mobile.
   - The user can select a plate, and always exactly one plate is selected. The selected plate should be highlighted in some way (for instance, you might simply change the color of the plate to make it brighter or outline it with GL_LINES). Use the arrow keys to select a different plate. Clicking the left and right arrows will move in the same row of plates, clicking the up arrow will move to the parent plate (if exists) and clicking the down arrow will move to one of the children.
   - The 'F' and 'b' keys toggle between front and back mobiles. The front one is the mobile closet to the projection plane.
   - The 'r' and 'R' keys rotate the selected plate clockwise and counter-clockwise. Keep in mind that the poses of all plates hanging beneath it (the "sub-mobile") are relative. This implies that their positions and orientations in space must change as their parent rotates, as do their sub-mobiles.
   - The '+' key will create another row of leaf-plates, hanging on the current smallest plates. Hitting '-' will kill the lowest row.
   - The 'G' key will start a random, continued slow rotation of the selected plate (and correspondingly its sub-mobile as well).

(3) Consider that the scene has 3 light sources, and implement a simple shadow-casting algorithm. The scene is dense enough that some of the plates in each mobile should block (occlude) the light hitting the plates in the other mobile. In other words, some of the plates will be *in shadow* with respect to some lights sources.
   - Place the 3 colored *spotlights* inside the cube at the corners: a red light at (0,1000, 0), blue at (1000,1000, 0), and green at (0,0,0). These spotlights will affect the color and shadowing of the plates in the 2 mobiles.
   - Use the two-pass Z-buffer shadow-casting algorithm discussed in class to find, for every light source, which plates are visible to it. No other solution will be accepted. If you follow this algorithm, your solution should *not* need to perform any explicit (and very slow) ray tracing.
   - To simplify things, let's assume that each plate is either completely seen by each light, or completely in shadow to it. So all points on the same plate will ultimately have the same color. We say that a plate is **seen** by the blue spotlight if there is any point in this plate *not occluded* from the blue spotlight. Similary it might be seen by the green and/or red spotlights. The color of this plate is the *combined* colors of the spotlights that see it. So for example, a plate is white only if it is seen by all three spotlights.
   - You can ignore shadow-casting with respect to the walls of the cube; the shadow algorithm need only to consider the mobiles. Further, you need not define or implement a local illumination model.

(4) In this homework, we will again use the hardware rendering capabilities of OpenGL, including the Z-buffer, drawing of filled, colored polygons, and transformations in 3D homogeneous coordinates. A few hints:

- OpenGL's matrix stack operations will help you manipulate the mobile's hierarchy of transforms.
- Use orthograhic projection for viewing the scene, with the projection plane at z=0.
- In the shadow casting algorithm, consider each spotlight as a viewpoint, and setup the light's view parameters just as you would for rendering the scene from that location. Each light should be able to "see" toward both mobiles in the scene.
- No hardware GL lighting or texture mapping is needed.
- For the shadow-casting algorithm, you'll need to read values out of select GL hardware buffers. You can read back the contents of a GL buffer by using the function *glReadPixels()*. The format parameter *GL_RGB* will allow you to read from the framebuffer, whereas *GL_DEPTH_COMPONENT* accesses the Z-buffer. You may also find the function *glReadBuffer()* useful. For more information on these functions, you may consult the *Red Book* or the OpenGL spec online at *http://opengl.org/documentation*. The buffer read-back capability should be the extent of the "new" GL functionality required for the assignment.