HOMEWORK #4

A BUMPY MOBILE

This assignment may be done in teams of two. **Due April 3 2012 by 11:59PM**. In this assignment we will explore scenegraphs, shading, linear interpolation and dynamic bump maps.

Figure 1. A baby's crib mobile.

A common trick for adding textural complexity to polygons is based on changing the normals to the surface at each pixel, according to a predetermined mapping function. Bump maps create the illusion that a surface is indeed more complex than its underlying geometric representation would allow. In this homework, you will create a spinning mobile consisting of bump mapped plates.

Your display window will be divided into 2 viewports. You should use the *Tab* key to toggle the focus of keyboard input between them. In the upper viewport, you will display a dangling mobile (Fig.1) consisting of an arbitrary number of spinning plates. In the lower viewport, you will display a 1D polyline which determines the normal displacements for the bump map. The contents of each of these viewports is outlined further below.

(1) (40 pts) **A dangling mobile:** A mobile is a toy that you commonly find above a baby's crib. For this homework, you will construct a mobile consisting of a series of dangling plates arranged in a balanced binary tree. Youe mobile should be drawn in the *upper* viewport. In particular, your mobile should conform to the following properties:

- 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 slightly less than half the height and width of its parent plate.
- The depth of the mobile is defined as the number of levels of plates. Assume that the depth is initialized to 2, but that the user can increase or decrease it (discussed below). For instance, a depth of 3 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). You should cap the maximum depth to 4 and minimum depth to 1.
- Each plate in the mobile displays a picture on its surface, and this picture can be seen on both sides of the plate. You should use GL's texture mapping capability to render the picture on each plate.
- The viewer's location doesn't move. Take care that the full mobile is still in view at max depth.
- There is one omni-directional point light illuminating the scene, and this light is at the viewer's location.
- You can ignore the connections (the "strings") between the plates.

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].

(2) (20 pts) **Mobile manipulation:** 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, as follows:

- 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 '+' key will create another row of leaf-plates, hanging on the current smallest plates, up to max depth. Hitting '-' will kill the lowest row.
- The 'r' and 'R' keys set the selected plate to rotate clockwise and counterclockwise in incrementally larger steps about the UP axis. 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 'C' key starts the selected plate rotating at some random, constant angular speed about the UP axis (correspondingly, its sub-mobile should rotate in unison as well). While rotating, the user can select other plates and change their combined rotation using the functions given here.
- The 'P' should act like a pause button toggle the rotation animation on and off for the entire mobile.

(3) (40 pts) **Dynamic bump map:** Use a bump mapping technique to modulate the shading of the pictures on your mobile plates. The bump map will be defined using a function $F[u, v]$, discussed below, that determines the displacement of the normal at each pixel $[u, v]$ in the picture. Normal orientation will consequently effect lighting intensity. The 2D function $F[u, v]$ should be drawn in the *lower* viewport and can be changed dynamically by the user. To define the bump map, you should do the following:
- In the following explanation we would assume that the image width is 80 pixels. Of course, this number needs to be normalized to the real size of your image.
- Let $y = f(x)$ be a piecewise linear function, i.e. a polyline, that is the concatenation of 4 line segments. This function is defined by by 9 equally-spaced control points $P_0, \ldots P_8$, where the x-coordinate of P_i is $i \cdot 10$ (for $i = 0, \ldots 8$), and its y-coordinate is initially 0, can be be controled by the user. Render this polyline in the lower viewport using GL_LINES. Then $f(x)$ is a linear interpolates between P_{i-1} and P_i, where i is chozen such that $10(i-1) \leq x < 10 \cdot i$. For example, if $x = 27$ then $f(x)$ interpolates between P_3 and P_4.
- Allow the user to manipulate height (y-coordinate) of the control points. The left and right arrows should move between control points along the polyline. The up and down keys should move P_i up and down by changing its y value accordingly to make it more positive or more negative, respectively.
- The bump mapping idea is then to specify the normal to the plate as follows: We define the function $F(u, v) = f(u)$, and the normal $\vec{N}(u, v)$ is approximated as the normal to the plane passing through the points

$$(u, v, F(u, v)), \quad (u+1, v, F(u+1, v)), \quad (u+1, v+1, F(u+1, v+1)).$$

You will need a formula here to help you compute the normal from this data, and don't forget that the length of the normal vector is always 1. You can disregard discontinuities at the control points by just taking the average for adjacent segments.

- Hint - you might want to use the following fact: The vectors $(-b, a)$ and $(b, -a)$ are both orthogonal to the vector (a, b).
- Initialize the y value of each control point to 0. Hitting the 'ESC' button resets all points P_i to this original value.
- Use the polyline $f(x)$ to define the bump map, which then modulates the pixel intensity of the pictures on the plates. In other words, perturb the normals along the surface of the plates using $F[u, v]$ and then use the perturbed normals in applying the simple diffuse illumination model discussed in class. (Hint: consider the angle between the displaced normal and the light vector.)
- Fix the orientation of the normals relative to the plate. The normals are perturbed according to the bump map and should only change when the bump map function $F[u, v]$ changes. However, remember that plates are themselves rotating in space, so the angle between the light vector and normal is constantly changing. Your program should account for this and correctly change the pixel intensities as the normals rotate in space with the plate.