Problem Set 3
CMSC 427
Distributed Thursday, March 31, 2005
Due: Thursday, April 14, 2005

Programming

The source code that is the solution to problem set 2 will form the basis for this problem set. You can begin with that code, or your own solution to problem set 2, and modify it.

1. **Rate-controlled animation (20 points).** In our previous code, we used the mouse to move around the scene. The viewing position was updated every time the idle function was called. This isn't a great idea, because it doesn't give us control over the speed of movement. Also, how fast we move can depend on how fast a processor we are running on.
    a. So, your first task will be to update this code so that your motion depends on the number of seconds that have passed since the last update. That is, your motion should be controlled to be a certain number of units per second. You'll probably want to use the C function clock() and the constant CLOCKS_PER_SEC from time.h.
    b. Add keyboard commands to control your speed. Pressing 's' should increase your speed by 10%, and pressing 'S' should slow you down by 10%.
2. **Rotating Cube (30 points).** Next we will add a new, moving object.
    a. Create a cube with a pattern of your choice texture mapped onto each side. The cube should appear in a random spot on the ground plane.
    b. Have the cube move like a die rolling. That is, it moves in a series of rolls. For each roll, the cube will have one edge that is fixed on the ground plane, and it will rotate about that edge. So, at the beginning of the roll, the cube will have one side flat on the ground plane. As it rolls, one edge of the cube stays on the ground plane. At the end of the roll, a new side of the cube is flat on the ground plane. Each of these rolls should take **0.5 seconds**. First get this to work for just a single roll.
    c. Now do this so that the cube rolls continuously. Every time the cube rolls, it rolls in a random direction, but one that is different from the direction of the previous cube position, so that it doesn't go back to the position that it was in immediately before. So for the first roll, there will be four directions that the cube can roll in. After the first roll, there will be three possible directions, since we'll rule out the direction it just came from.
3. **Collision Detection (20 points)** Finally, make sure that as the cube rolls, it does not roll into any of the other rectanguloids in the scene, and that it doesn't roll off the ground plane. You can do this when you decide on a new direction for the cube to roll in. Pick a direction that will not result in a collision with another object, or in leaving the ground plane.
4. **Challenge Problem:** Give one side of the cube a special pattern. This is the target. Now add a test to detect whether the viewer is flying into the cube. When the viewer collides with the rolling cube, if it flies into the target side of the cube, you win. If the viewer flies into a different side of the cube, you lose.

Pencil and Paper Exercises (6 points each)

1. Motion
    a. Suppose we have a globe centered at the origin, with a radius of 1. It is oriented so that the north pole is on the y axis. We want the globe to spin so that the north and south poles are fixed, and so that it spins around once every ten seconds. Give a matrix that will account for the motion that occurs in one second.
    b. Now suppose that the globe is standing still. But there is a line of bugs on the globe. The first bug is at the equator in the position (1,0,0). The other bugs are in a row that goes due south on the globe. The bugs are crawling up the globe as if they were going straight north at a speed that will take the lead bug to the north pole in twenty seconds. Provide a matrix that will update the position of any one of the bugs over a one second period. That is, we should be able to apply this matrix to the position of a bug at time *t*, and obtain its position at time *t+1*.
    c. Now suppose the globe is spinning and the bugs are crawling north at the same time. Provide a matrix that we can apply to a point where one of the bugs started, and that will give the position of that bug at time *t*. This matrix should not depend on where the bug started.
2. Viewpoint. Suppose we want to render a scene from the viewpoint of an observer in an air balloon. The observer is located at position (3,10, 2). He is looking straight down on the world (that is, in the negative *y* direction). There is a river flowing on the ground (which is the y = 0 plane) along the line x = z in the direction (1,0,1). The observer is looking so that the river seems to be flowing up in their field of view (their up vector is aligned with the river). Provide a matrix that will transform points into the world to points in this viewer's coordinate system.
3. Collision. Suppose that we have a 2D square centered at the origin. The sides of the square might have any orientation. The square has a width of 1. There is a wall along the line x = 10. The square is moving in the direction (1,0) at 1 unit per second.
    a. We put a circle around the square as a coarse bound, to test for collisions. How long will it be before this circle collides with the wall? Does this depend on the orientation of the square?
    b. Suppose instead of a circle, we put an axial rectangle around the square (that is, a rectangle whose sides are parallel to the x and y axes). How long will it be before this rectangle collides with the wall? Does this depend on the orientation of the square?
4. Visibility. We are looking at the world with a perspective camera that has a focal point at (0,0,0), and an image plane of z=1. We are looking at two polygons in the world, P1 and P2. We want to render the polygons so that, as in the painter's algorithm, we only render a polygon if it is the most distant, unrendered polygon that affects a set of pixels. That is, there should not be another polygon that is further away and that will intersect the first polygon in the image.

For different choices of P1 and P2, we will ask: Is it safe to render P1 first? If yes, indicate for all of the following tests whether P1 passes or fails them (if no, P1 must fail all tests). Don't just indicate the first test that P1 passes, indicate all of them.

1. It is safe because all of P1's z coordinates are bigger than any of P2's.
2. It is safe because if we project the two polygons into the image, axial rectangles that bound them do not intersect.
3. It is safe because all the vertices of P1 lie behind the plane that P2 lies in.
4. It is safe because all the vertices of P2 lie in front of the plane that P1 lies in.
5. It is safe because if we project the two polygons into the image, they do not intersect.

a. Answer these questions when P1 has vertices (0,0,4), (1,0,2), (1,1,4) and P2 has vertices (1,0,3), (5,0,1), (1,1,3).
b. Answer these questions reversing the polygons. That is, when P2 has vertices (0,0,4), (1,0,2), (1,1,4) and P1 has vertices (1,0,3), (5,0,1), (1,1,3).
c. Answer these questions for the case in which P1 has vertices (0,0,4), (1,0,2), (1,1,4) and P2 has vertices (3,0,3), (7,0,1), (3,1,3).
d. Answer these questions reversing the polygons in (c). That is, for the case in which P2 has vertices (0,0,4), (1,0,2), (1,1,4) and P1 has vertices (3,0,3), (7,0,1), (3,1,3).

5. Color
    a. Can you create an example of two colors in which the colors have the same hue, and the second one has one-fifth the value of the first, but it is five times as saturated? Give an example, or show why this is impossible.
    b. Suppose in a subtractive CMYK color system we mix one part cyan with two parts magenta and one part black. What would be an RGB description of this color? There may be more than one valid answer to this.