Transformations and basic OpenGL drawing

Due date: Tuesday, February 22

Your program should show a moving car with rotating wheels. the wheels should stay in contact with the ground, and should rotate properly (i.e. the lower point of the wheel at any instant in time should not be moving with respect to the ground. Draw a large rectangle to show where the ground plane is.

The body of the car can be very basic (in the simplest case, just a couple of boxes) but it should have four circular wheels with spokes. These are minimal requirements your object should meet. Bonus points will be awarded for interesting-looking cars.

The user controls the car with up and down arrows (speed) and left and right arrows (direction). Pressing space should move the car back to the original position.

What to hand in

Just a working version of the program and the source code.

Restrictions

Do not use `glut` commands for drawing objects. For matrix operations, use only `glLoadMatrix`, `glMultMatrix`, `glPushMatrix()`, and `glPopMatrix()`. Do not use `glTranslate`, `glScale` and `glRotate`. Implement their functionality using the formulas discussed in class. However, we recommend that you use these commands while debugging the scene, and, once everything is working, replace them with your own equivalent functions.

Implementation Suggestions

Trying to calculate positions of all vertices of all objects in the scene by hand is not a very good idea; also, in that case, if you would want to move something, it would be rather difficult.

Typically, one builds scenes like this hierarchically as explained in class.

First write functions that draw boxes and cylinders; (we ask you not to use the `aux` or `glut` library call, but feel free to use it for debugging). Build more complicated objects by using transformations and calls to these functions.

Make as few preliminary calculations on paper as possible; try to structure your program in such a way that all objects and their positions are computed directly from several basic dimensions, without any manual calculations. Avoid having too many specific dimensions in your program: try to parametrize each object by several numbers; e.g. the wheel can be specified by the radius, thickness of the tire, the number of spokes and the radius of spokes.

It is useful to be able to see the coordinate system; for debugging purposes, draw thin long boxes of different colors along the positive directions of the axes.

A note on lighting To see the scene, we need a light source; if we do not put a light somewhere, the polygons of the scene are still visible, but they are *flat-shaded* i.e. all pixels in the image of any polygon have the same color. Pictures like this do not look very interesting, especially if all polygons have the same color (default is white). To make things more interesting, either make boxes of different colors, using `glColor3f()` function calls or use a single light We will discuss lighting in greater detail later; for this assignment, add the following OpenGL calls after you create your GLUT window, but before you call the `glutMainLoop` function; this will add two lights, red and blue, to your scene.

```
    GLfloat diffuse[] = {0.8, 0.8, 0.8, 1.0};
    GLfloat lgt1_diffuse[] =  { 0.05f, 0.05f, 0.6f, 1.0f };
    GLfloat lgt2_diffuse[] =  { 0.6f, 0.05f, 0.05f, 1.0f };
    GLfloat light_pos1[] =   { 5.0f, 5.0f, 0.0f, 1.0f };
    GLfloat light_pos2[] =   { -5.0f, 5.0f, 0.0f, 1.0f };

    glEnable(GL_LIGHTING);
    glEnable(GL_DEPTH_TEST);
    glShadeModel(GL_SMOOTH);
    glEnable(GL_LIGHTING);
    glMaterialfv( GL_FRONT, GL_DIFFUSE, diffuse);

    glLightfv(GL_LIGHT1, GL_POSITION,light_pos1);
    glLightfv(GL_LIGHT1, GL_DIFFUSE, lgt1_diffuse);
    glLightfv(GL_LIGHT2, GL_POSITION,light_pos2);
    glLightfv(GL_LIGHT2, GL_DIFFUSE, lgt2_diffuse);
    glEnable(GL_LIGHT1);
    glEnable(GL_LIGHT2);
```

To make the lighting work, you also need to specify normals for each polygon. Here is an example of the code that defines a unit square in the XY plane with normal pointing along Z:

```
glBegin(GL_POLYGON);
glNormal3f(0.0f, 0.0f, 1.0f);
glVertex3f(0.0f, 0.0f, 0.0f);
glVertex3f(1.0f, 0.0f, 0.0f);
glVertex3f(1.0f, 1.0f, 0.0f);
glVertex3f(0.0f, 1.0f, 0.0f);
glEnd();
```