This assignment may be done in pairs if you prefer.

**IMPORTANT**: All parts of all assignments should be understood for the exams, regardless of whether your partner did it.

**Hint. Read the assignment carefully, and think it through before starting to code.**

If you have no other plan, you may find the sample code on the OpenGl page in the class web tree for matrix manipulation useful.

In this assignment we will improve “parallelepiped -world” which was started in the previous assignment. The same rules regarding input and output and program exit from assignment three applies.

The first step is to incorporate visibility. We will trick OpenGL into doing this for us, without letting it handle the projection. To do this, change your projection system to use the mapping to the standard 3D viewing box (unless you already did it this way). Use OpenGL’s facility for rendering 3D **orthogonal** scenes (glOrtho()), and apply it to the orthogonalized version of your world. Check that you get the same image for the same cube as in the third assignment. You are allowed to tell OpenGL that the world is 3D orthogonal, but not perspective. You are to handle the perspective part by mapping the world into the appropriate 3D box. OpenGL will then map your 3D objects onto the 2D plane by dropping the z-coordinate, after dealing with visibility and clipping. You should implement back-face culling as in assignment three.
Because we now want to have more than one object, you need to request that OpenGL deals with visibility. You can do this by adding the symbol GLUT_DEPTH to the symbols that are or’d in the call to glutInitDisplayMode() as in glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH), and then doing a glEnable(GL_DEPTH_TEST). We also have to “or” in GL_DEPTH_BUFFER in glClear(). (The order of the all the initialization processes does matter, and (I believe) the enable should be done after the window creation). See the large variety of OpenGL references for more information.

The first improvement noticeable to the user is to allow multiple objects. Your program should handle multiple “pp” commands in sequence, each one creating a parallelepiped.

You should use the same color scheme for all parallelepipeds. Each face of a given parallelepiped should be a different color (except black).

On program startup the last box created is the selected object. You should provide some user feedback to identify the selected one (perhaps color reversal).

Give the user a way to add a default cube at a default location through the menu. When a cube is added through the menu, it becomes the selected parallelepiped.

The user should be able to change the selected object with a single click with the left button while the pointer is on the object to be selected. This “picking” function must be implemented by yourselves, and will be used for future assignments.

Now provide the user with the following facilities to modify the selected object.

The x key is to be used to shrink the parallelepiped in the x direction (world coordinates), and X key is to be used to stretch it. Ditto for y and Y and z and Z. Scaling should be relative to the center of gravity of the parallelepiped.

Dragging the mouse with the left button down should translate the parallelepiped in the direction of the mouse drag. Note that the drag defines a line in the camera plane. The b key is used to translate the parallelepiped away from the camera, and the f key is used to translate the parallelepiped towards the camera.

For grad students:

The r key is used to rotate the parallelepiped clockwise around its center using the direction that the camera is pointed (viewing direction) as the axis of rotation. R is used for counter-clockwise.

Dragging the mouse with the middle button down should rotate the parallelepiped around its center using an axis through it which is perpendicular to the drag. Rotation direction should be the natural one.

End grad student part
Try to calibrate key press actions and drags to affect a natural amount of motion. This will require rotations and translations proportional in magnitude to the stroke length.

As usual, when the user enters “q” in the graphics window, the program first writes the appropriate commands to standard output, and then exits. If you start up your program with those commands as input, the screen should look exactly as it did on exit.

**HINT for exams.** It may be helpful to understand why you can write out the state of the world with one command for each parallelepiped, despite all the transforms.

**Extra credit**

If you would like to improve on the program, be sure to explain what you did in the README file, and it will be considered for modest extra credit.

**Deliverables**

You must electronically submit a README containing any relevant information, but at a minimum, your name; an executable (called a4); and a src directory containing source files and a Makefile which can be used to build the executable.

The program must compile and run on one of the graphics machines (gr01, ... , gr08). Put in the README file the machine which you have verified this on.

The turnin name is cs433_hw4.