Assignment 1 COMP 557 Winter 2015

Posted: Friday, January 23

Due: Monday, February 2 at 23:59

General Information

- This assignment is worth 10 % of your final course grade.
- The T.A.s handling this assignment are Fahim Mannan, Charles Bouchard, and Shayan Rezvankhah. Their office locations and emails are posted at the top right corner of the public web page.
- T.A. office hours will be posted and will be dynamic and depend on demand.
- Use the myCourses discussion boards (Assignments/A1 and Other/PyOpenGL) for clarification questions.
- Do not change any of the given code. Add code only where instructed.
- Submit your code `A1.py` using the myCourses Assignment/A1 folder. Include your name and student ID number in the given location at the top of the file.

  If you have any issues that you wish the TAs to be aware of when grading, then include them as a separate comment with your submission. Otherwise do not leave separate comments (such as "here is my submission...") since it just adds clutter.
- **Late assignments** will be accepted up to only 3 days late and will be penalized by *1 point per day on your final course grade.* If you submit one minute late, this is equivalent to submitting 23 hours and 59 minutes late, etc.

  Note that the assignment has 100 points and so the late penalty is 10 points per day.

Overview

The goal of this assignment is to introduce you to the OpenGL API. In particular, you will practice using the 3D transformations and projections that we saw in lectures 1-6. Your task will be to design a simple 3D scene and draw it from various perspectives.

The posted code contains three files:

- `A1.py` The file contains comments at the beginning that gives you the organization of the code. You need to add code only in places marked by comments.
- `A1DrawHelperFunctions.py` which contains some helper functions for drawing.
- `A1main.py` which is the file that you run.

The given code allows you to interact with the scene with the keyboard and mouse. OpenGL itself does not provide any functions for such interaction so one has to use an extra library called GLUT (The OpenGL Utility Toolkit). GLUT provides windowing and user interaction by letting the user define callback functions which gets called when events occur (e.g. mouse movement, keystroke, etc).

The code defines four viewports: (1) bottom left, (2) bottom right, (3) upper left, (4) upper right. Each viewport defines a camera and we will refer to these cameras by their viewports 1-4. You can manipulate the position and orientation of camera 1 using the mouse and keyboard.

- **LEFT MOUSE:** Moving the mouse forward/backward while holding down the left button down will tilt camera 1 upwards/downwards. Moving the mouse left/right will pan camera 1 left/right.

  Alternatively, camera 1 rotations can be performed using:

- **LEFT/RIGHT/UP/DOWN Arrow Keys:** Holding the up/down arrow keys down will tilt the camera upwards/downwards. The left/right arrow keys will pan the camera left/right.
- **RIGHT MOUSE:** Moving the mouse forward/backward while holding down the right button down will translate camera 1 in the $-z$ and z direction, respectively. Moving the mouse left/right will translate the camera in the x and $-x$ respectively.

  Alternatively, camera 1 translation in world coordinates can be achieved using:

- **WASD keys:** Holding down 'W' translates the camera in the world's $-z$ direction (forward), 'S' translates in world's z direction (backward), 'D' translates in world's x direction (right), 'A' translates in -x direction (left).

Note that the rotation and translation have been implemented in the world coordinate system, not in the camera 1 coordinate system.

Requirements (100 points total)

1. **(30 points)**

   Make a static 3D scene that contains a floor but no walls or ceiling, and several objects such as boxes, cones, teapot, etc. The objects may *not* be floating in the air. They must either be resting on the floor or on top of each other. Add your code in the `drawScene()` function.

   Note that the provided "solution" does not satisfy the above, in two ways. First, the scene contains a rotating object. Second, there is no floor.

   Use all of the OpenGL transformations `glRotatef()`, `glTranslatef()`, and `glScalef()` in a non-trivial way. For example, rotating by 0 degrees or scaling by 1 are both trivial. For full points, your scene should contain multiple objects. Read up on how `glPushMatrix()` and `glPopMatrix()` works and use them. Have some fun here!

2. **(10 points)**

   Camera 2 (viewport 2) is an external static observer who views the same scene under orthographic projection. When the user uses the mouse/keyboard to translate and rotate camera 1, the boundary of camera 1's view volume should be drawn in viewport 2 and should move appropriately. For example, as camera 1 pans left/right, the view volume drawn in viewport 2 should rotate left/right. When camera 1 translates forward/backward, the view volume should translate forward/backward. However, the objects in the scene and the (world) coordinate axes should not move. Note that the world coordinate axes are colored RGB for xyz.

   To move the view volume, you will need to implement `drawMovingViewVolume()`.

3. **(30 points)**

   Viewport 3 is used to visualize the result of the `GL_PROJECTION` transformation that was used to make the image in viewport 1. Your task in viewport 3 is to draw the normalized view volume of camera 1, and the normalized scene. The view volume in particular should be transformed to a $2 \times 2 \times 2$ view volume centered at the origin, and the objects in the scene should appear suitably deformed as well.

   The camera for viewport 3 is an external static observer who views the normalized scene under orthographic projection.

   Note that the starter code for viewport 3 includes a call to `projective()` which defines a projective matrix for you to use, but it does not include instructions for the normalization. You will need to add these. The starter code also includes instructions to draw a *left* handed coordinate system so that the z axis points away from the camera. (Normalized view coordinates are left handed.)

Normalize the scene using `glScalef()` and `glTranslatef()`. Your normalized view volume must be consistent with the drawn left handed coordinate system, in particular, objects that are further away from the camera now have a bigger positive z value. The slightly tricky part of this question (conceptually at least) is that camera 3 itself is right-handed.

Finally, note that when camera 1 is moved using keyboard/mouse, the objects shown in viewport 3 should deform but the normalized view volume (a cube) and the coordinate system shown should *not*.

4. **(30 points)**

   Viewport 4 shows the same scene, now viewed by a moving perspective camera. The position and motion of the camera are defined by `drawMovingCamera()`. The position of this camera initially follows a circle of radius `movingCameraRadius` and is embedded in the world $y =$ `movingCameraHeight` plane. The lookat point of the camera is the center of the view volume. Pressing `'i'/'I'` and `'k'/'K'` will change the height of the camera. This will result in the view of the camera changing in viewport 4 and the change in orientation of the camera should be visible in the other viewports.

   Add this moving camera to your solution of viewport 3 (the camera is already shown in viewports 1 and 2).

   Note that initially, the radius of the circle is so large that this new camera is outside the view volume and hence will not be visible in viewport 1. Once the main camera (viewport 1) moves, this new moving camera may come into view in viewport 1 though.

   Now let's turn to the content of this question, which has two parts:

   The first part is to draw the scene, as seen by the moving camera, using standard OpenGL commands.

   The second part is much closer to modern OpenGL (3.0 and beyond) where `glRotatef`, `gluLookAt()` etc commands are *deprecated* and instead one needs to define all the transformations directly.

   In the second part, you are thus not allowed to use `gluPerspective` and `gluLookat`, nor are you allowed to use `glTranslate` and `glRotate`. Instead, you must implement these transformations by setting the `GL_PROJECTION` and `GL_MODELVIEW` matrices directly, and using `glMultMatrixd`.

   A few important notes:

   `glMultMatrixd` can take a 2D array as input but it expects the matrix to be in column major order, i.e. entry (i,j) is expected to be ith column and jth row. If your matrix indexing is in row major order (as in the lectures), then you will need to pass its transpose to `glMultMatrixd` using `numpy.transpose()`.

The starter code provides the functions for translation (`translate`), scaling (`scale`), rotation about the X axis (`rotateX`) and Y axis (`rotateY`). These functions return a 4×4 numpy matrix. You have to use these functions to construct your `GL_MODELVIEW` matrix.

Instead of `gluPerspective`, you will need to implement and then call the function `perspective`. This function needs to be implemented using `projective(near, far)` and normalization technique given in class. You cannot just look up `gluPerspective` on the web.

Note that you *are* allowed to call the existing draw commands (`drawScene()`, `drawMovingViewVolume()`). We are *not* asking you to re-implement these commands.

The starter code contains a toggle option (key 't') which switches between the solutions for the two parts of the problem.

Good luck and have fun!