Homework 2: Basic OpenGL

Introduction to Computer Graphics and Imaging (Summer 2012), Stanford University
Due Monday, July 9, 11:59pm

In Lecture 3, we discussed the homogeneous coordinate system appearing often in computer graphics. In the first part of this assignment, we will see how homogeneous coordinates simplify the process of computing camera transformations.

First, a quick warm-up:

Problem 1 (10 points)**.** *(a) Give two different homogeneous expressions for the point* $(x, y, z) = (3, -2, 7)$*. (b) Convert* $(5, -10, 15; -5)$ *to standard coordinates. (c) Explain what the homogeneous point* $(1, 2, 3; 0)$ *represents. (d) Give a homogeneous transformation matrix translating* $(4, -2, 5)$ *to the origin.*

Now, let's continue with the task at hand: finding a matrix converting points in $\mathbb{R}^3$ (really projective space $\mathbb{P}^3$, meaning they are represented with *four* numbers as we saw in Problem 1) to so-called normalized device coordinates. Our first job is to take the camera from an arbitrary location and orientation in space to a convenient *canonical* position and orientation.

Problem 2 (20 points)**.** *Suppose that the camera is located at* eye position $\vec{e} = (e_x, e_y, e_z)$*. Furthermore, assume the camera has* gaze direction $\vec{g} = (g_x, g_y, g_z)$ *and* up vector $\vec{t} = (t_x, t_y, t_z)$*; assume* $\|\vec{g}\| = \|\vec{t}\| = 1$*. Find an expression for the matrix* M_{cam} *translating* $\vec{e}$ *to the origin and rotating so that* $\vec{g}$ *points down the* $-z$ *axis and* $\vec{t}$ *points up the* $+y$ *axis; your expression can contain intermediate variables, matrix inverses, cross products, variables you labeled earlier, or anything else convenient.*

Comprehensive hint: *This is a tricky problem, so let's break it down. First, remember that points in* $\mathbb{R}^3$ *are become sets of four numbers* $(x, y, z; 1)$ *in projective space* $\mathbb{P}^3$ *and that* directions *can be expressed as* $(x, y, z; 0)$*. A reasonable set of steps to get to the solution is to do the following:*

1. *Write a vector representing the product* $M_{cam} \cdot (e_x, e_y, e_z; 1)$*; note that your answer isn't unique since it could be multiplied by a constant (that is, the points* $(x, y, z; w)$ *and* $(ax, ay, az; aw)$ *are the same in projective space), but this degree of freedom actually can be ignored for this problem.*
2. *Write a vector representing the product* $M_{cam} \cdot (t_x, t_y, t_z; 0)$*.*
3. *Write a vector representing the product* $M_{cam} \cdot (g_x, g_y, g_z; 0)$*.*
4. *Define* $\vec{u} = \vec{t} \times \vec{g}$*. Write a vector representing the product* $M_{cam} \cdot (u_x, u_y, u_z; 0)$*. Drawing a picture may help here – remember the right-hand rule!*
5. *Combining your answers to the last few steps, write a matrix equation of the form* $M_{cam}U = V$*, for* $U, V \in \mathbb{R}^{4\times4}$ *constructed from the vectors we've computed and defined above.*
6. *Use a matrix inverse to get an expression for* M_{cam}*. You don't need to simplify here, although it turns out that the simplified expression is fairly nice.*

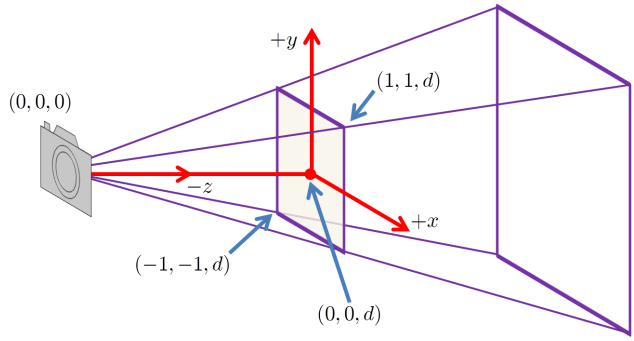


Figure 1: For problem 3.

We now can assume that we have applied M_{cam}, so our camera is located at the origin $(0,0,0)$ and is pointed down the $-z$ axis. Next, we'll implement a *perspective matrix* that converts these coordinates to image coordinates. We're not going to worry about depth computation here; we will discuss an interesting nonlinearity involved in depth computation during Lecture 4.

Problem 3 (20 points)**.** *Write a* 3×4 *matrix* M_{persp} *that takes a point* $(x,y,z;w)$ *in canonical camera coordinates above and yields a point* $(x',y';w')$ *on the image plane* $[-1,1] \times [-1,1]$*. In particular, the square marked in Figure 1 centered on the* $-z$ *axis at distance* d *from the origin should be mapped to* $[-1,1] \times [-1,1]$*, as should lines from the origin through this square.*

Hint: *First work out the projection formula in inhomogeneous coordinates without using matrices, and then figure out how to homogenize and convert to matrix form.*

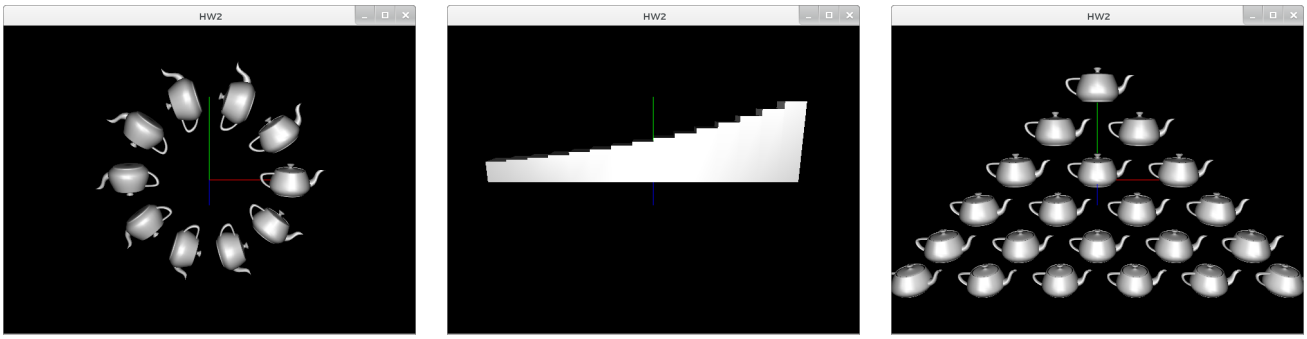
Finally, we can combine these matrices to get the transformation directly from world coordinates to the image plane.

Problem 4 (10 points)**.** *Write such a transformation matrix* P *in terms of* M_{persp} *and* M_{cam}*.*

Now that you understand the *theory* of transformations, we'll spend the rest of this assignment making use of OpenGL's built-in capabilities to compose and construct transformations. We have two objectives here: for you to be exposed to basic OpenGL, and for you to get more practice formulating linear transformations.

In this assignment, we will make use of the GLUT ("GL Utility Toolkit") library to make it easier to write OpenGL code. GLUT is a simple platform-independent way to set up a window and make callbacks for keyboards, mice, timers, and so on. We have provided the GLUT code you need in the starter code, but you should take a look to make sure you understand what it's giving you. We also have set up a simple camera pointing at the origin, where the left mouse button rotates about the y axis and the right mouse button zooms. You can use the keyboard to switch between displays for each of the problems.

Problem 5 (40 points)**.** *Fill in* `problem1`*,* `problem2`*, and* `problem3` *in the skeleton code provided on the course website to reproduce the images shown in Figure 2. We're not worried about* 100% *pixel-perfect reproductions of the images, but do try your best to get close. You should produce these images using the* `glutSolidTeapot` *and* `glutSolidCube` *functions (do not add your own geometry!), combined with OpenGL's transformation mechanisms, including* `glPushMatrix`*,* `glPopMatrix`*,* `glTranslatef`*, and so*



(a) `problem1` (b) `problem2` (c) `problem3`

Figure 2: For problem 5.

on. You may need to look ahead to Lecture 5 if you begin programming this assignment early. To receive full credit, you should use and compose transformations in a way to get the scene cleanly.

Finally, fill in `problem4` *to render an interesting scene of your choosing. Your scene at the least should:*

1. *Make use of OpenGL's transformation mechanisms in a nontrivial way, with at least one instance of nested applications of* `glPushMatrix`*.*
2. *Render at least one triangle by feeding in its coordinates directly (OpenGL immediate mode is OK here, even though it's deprecated)*
3. *Entertain the instructor and/or course assistant*

The best submissions will be shown in class.