CMSC427 Fall 2017
Lab 2 – Transformations (part A)

Due by midnight Thursday, Sept. 28th
Online as combined PDF plus a PDE file

Objectives of lab:

• Get a feel for the operation and composition of 3D transformations

Requirements:

This is a lightweight lab intended to encourage you to work on transformations without strict right or wrong answers, and to get an intuitive sense of these operations. For background on 2D transformations in Processing see: https://processing.org/tutorials/transform2d/

For background on 3D transformations see the tutorial below. It includes material on textures and lighting which are not relevant yet, but will be material we cover soon.
https://processing.org/tutorials/p3d/

The Processing reference pages on the relevant methods is also useful, under Transform and Camera sections (towards the bottom of the page): https://processing.org/reference/

Part I: Understanding modeling transformations

Starting with this code, play with the various transformations to see the results. Check the visual results, and look at the matrices that result. Try translate, rotateX/rotateY/rotateZ, scale, shearX/shearY; try ordering the transforms with rotate then translate, or translate then rotate.

```
size(400,400,P3D);
translate(width/2,height/2,0);
// Add in transformations here
box(100);
printMatrix();
```

For instance, try translating +100 in X and see what the matrix now looks like. Try X, Y and Z rotations and look at the matrix. You should try to understand the orientation of the coordinate system, the orientations of the rotations and shears, and how sequencing effects transforms. Also, you should be able to predict the components of the matrix from a single

For purposes of the lab simply report on some of your observations. There are no strict requirements beyond one:

Write a sketch that place three boxes (and sphere if you'd like) in a scene. Like three boxes in a row, but each with a different orientation. Either use pushMatrix()/popMatrix() to set distinct transformations for each box (push, transform, box, pop; push, transform, box; etc), or plan your transforms to accumulate.

Part II: Understanding camera transforms

The camera transformation takes points in World coordinates into Camera coordinates. The standard approach to determining the transformation, represented by the Processing routine **`camera`**, takes the points At and LookAt, and the vector Up, and produces a static homogenous matrix to place the camera is in the World coordinates. Objects are rendered as seen from the camera position.

For this lab, use the following code to experiment with the camera method. Read the reference page https://processing.org/reference/camera_.html

The camera method takes two points Eye and Center, and a vector Up. The camera is located at the point Eye, looking at Center, with the sense of up given by Up. The two points are often called At and LookAt (so Eye==At, and Center==LookAt).

Try modifying the values in camera to see the result. Try placing the camera out a fixed distance on the X axis, or the Y axis, or out on a diagonal. Note that the practical origin of the World coordinate system is at (width/2, height/2, 0), and the camera parameters here are the default position of the camera. Changing the up vector to (1,0,0) may not change the view since the box is symmetric – scale the box by (2,1) so it's non-uniform, then you'll see a change from effectively rotating the camera.

```
void setup() {
  size(640, 360, P3D);
  stroke(255);
  noFill();
}

void draw() {
  background(0);
  camera(width/2, height/2, (height/2) / tan(PI/6), // At
         width/2, height/2, 0,                      // LookAt
          0, 1, 0);                                 // Up
  translate(width/2, height/2, -100);
  box(200);
}
```

For the purposes of this lab, simply report on the effect of changing each parameter. You can use printCamera() to see the camera matrix.

Foreshadowing – not for this lab, but for the future. The reason to understand Camera transformations is to be able to create a Camera transformation for specific applications, such as a camera following a curve, following a character in a video game, or controlled by a mouse. This tutorial explains these steps. https://learnopengl.com/#!Getting-started/Camera

Part III: Understanding projection transform

The Projection transformation sets the focal length of the camera, determining how wide a view is rendering. This is set with the first FOV (field of view) parameter. The second parameter, aspect ratio, sets the ratio of the vertical to horizontal FOV. The last two set the front and back clipping planes, so objects closer that the front, and farther than the back, are not seen.

Play with this program so you under the basic effects of the four parameters. We'll review the math in class. For the purposes of this lab simply report on changing each parameter. You can use printProjection() to see the matrix.

```
void setup() {
  size(200, 200, P3D);
  noFill();
}

void draw() {
  background(255);
  float fov = PI/(map(mouseX,0,width,2,5));
  float cameraZ = (height/2.0) / tan(fov/2.0);
  perspective(fov, float(width)/float(height),
            mouseY*2, cameraZ*10.0);
  translate(100, 100, -100);
  rotateX(-PI/6);
  rotateY(PI/3);
  box(100);
}
```

Submission

To submit you should create a Word or other document that includes the following:

A. A header with CMSC427 fall 2017 Lab 2 and your name.
B. A short narrative on your experiments for the three parts. One page overall will be enough.
C. A copy of the PDE source code of the three boxes copied and pasted (hint – "copy as HTML" brings along the color coding").
D. A picture of your three boxes copied and pasted into the document.
E. Save as PDF and submit, along with a separate copy of your PDE file from part I.

When you report on your experiments, it is fine to include questions – I did this, got this result, and am not sure why.