CMSC 317: The Computational Image
Assignment 5: Geometry in 2D

Part One: Homogeneous Coordinates

Create a processing sketch that allows the user to drop points interactively using the mouse. You should compute the lines (not just the line segments) connecting those points and display them. For this sketch you should use homogeneous coordinates to represent both the points and lines. Remember the point $p = [x, y]^\mathsf{T}$ is represented by the homogeneous coordinates $p = [x, y, 1]^\mathsf{T}$ and $l = [a, b, c]^\mathsf{T}$ represents the line $0 = ax + by + c$. You can find the line passing through two points by taking their cross product: $p_1 \times p_2$ or `p1.cross(p2)` using Processing's `PVector` class. And the intersection point of two lines can be found using the cross product: $l_1 \times l_2$ or `l1.cross(l2)`. The following Processing function draws a line represented in homogeneous coordinates (why are there two cases?):

```
void plotLine(PVector v) {
  if (abs(v.x) < abs(v.y)) {
    line(0, v.z/-v.y, width, (v.x*width + v.z)/-v.y);
  }
  else {
    line(v.z/-v.x, 0, (v.y * height + v.z)/-v.x, height);
  }
}
```

Part Two: 2D Transformations

As part of a (possibly separate) Processing sketch, implement the following 2D transformations. Each transformation should take a `PVector` v and return the transformed point $\mathbf{M}v$. You can use the Apache Commons Math library[1] or the JAMA[2] matrix library for the matrix operations by adding the `jama-102.jar` file to your sketch.

Translation
$$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

Euclidean Transform
$$\begin{bmatrix} \cos\theta & -\sin\theta & t_x \\ \sin\theta & \cos\theta & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

Similarity Transform (or Scaled Euclidean Transform)
$$\begin{bmatrix} s\cos\theta & -s\sin\theta & t_x \\ s\sin\theta & s\cos\theta & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

Use these transform functions to map the pixels in your sketch in an interesting way. For example, consider modifying Shiffman's `Pointillism` sketch to render the webcam's pixels in a rotated and scaled manner.

[1] http://commons.apache.org/proper/commons-math/
[2] http://math.nist.gov/javanumerics/jama/