CS 91R: The Computational Image

Assignment 3: Recursive 2D Geometry

DUE February 19th at 11:59 PM

In this lab, we will explore 2D geometry and learn how to use matrix multiplication to perform scaling, rotation and translation. Also we'll learn about the wondrous world of recursive functional drawing ala M.C. Escher and Peter Henderson.

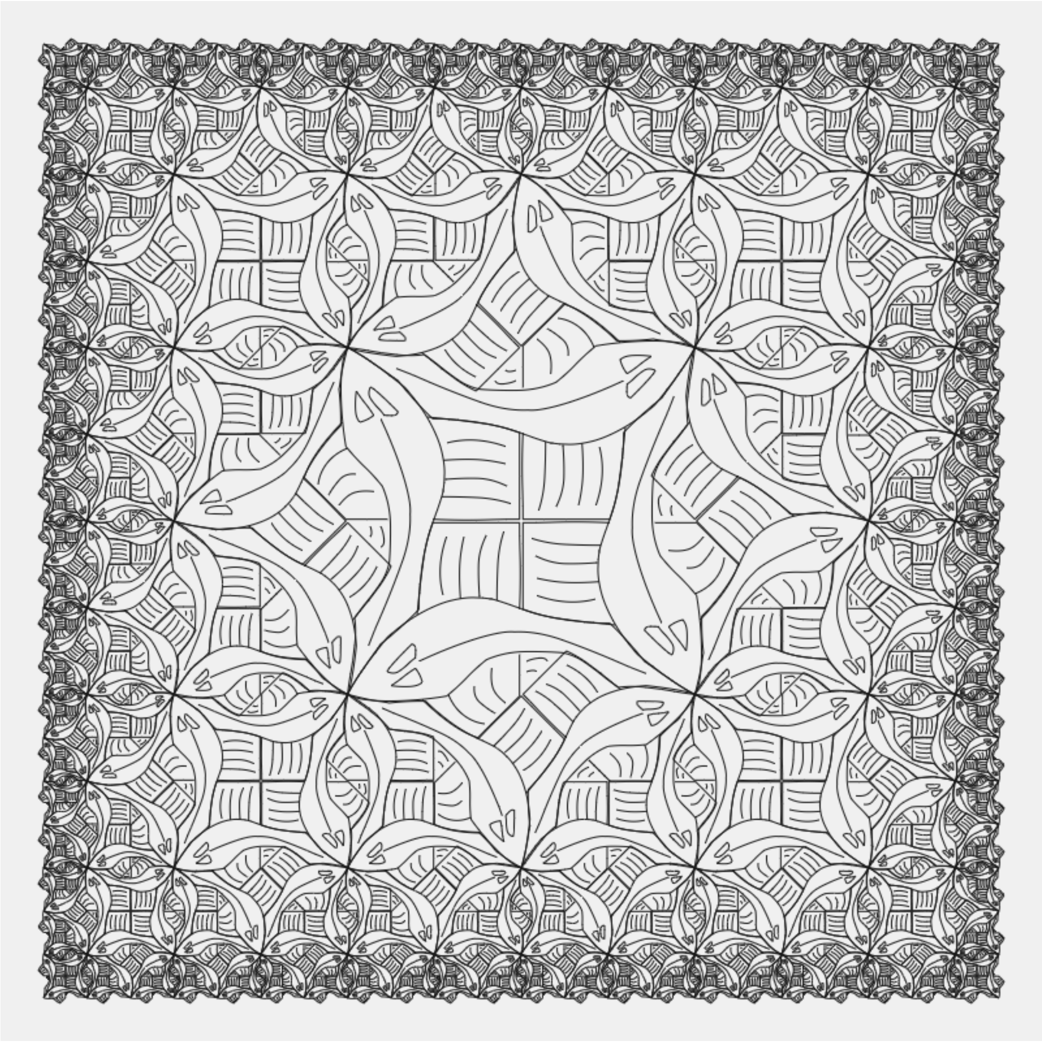


Figure 1: `squarelimit(3)`

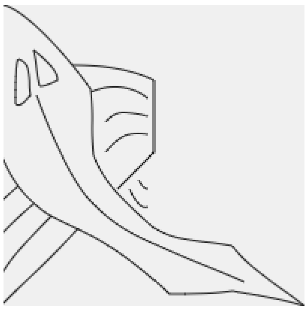
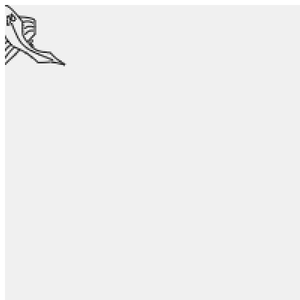
I've included the `fish.csv` file that includes the curve needed for our basic fish. I've also created functions that load in the data and display it (`drawCurve`). Your job is to implement the stub methods (marked `TODO`) to make the final Escher artwork a reality.

1 Tasks

1. `vscale(v: p5.Vector, sx: float, sy: float): p5.Vector`

   - for now just return a new `p5.Vector` with the `x` and `y` components scaled (i.e., multiplied) by the `sx` and `sy` parameters.

   NOTE: `drawCurve` will not seem to work until you complete this step (the curves are too tiny!)

For example, it can be used to transform all the coordinates in the curves.

```
function applyT(curves, t) {
  let newImg = [];
  for (let line of curves) {
    let newline = line.map(t);
    newImg.push(newline);
  }
  return newImg;
}

function shrink(curves) {
  return applyT(curves, (v) => vscale(v, 0.2, 0.2));
}

drawCurve(shrink(fish));
```

Pressing `p` should test this function.

2. `transform(v: p5.Vector, matrix: list): p5.Vector`

   - This function should take a *inhomogenous* vector `v`: (x, y), turn it into an augmented vector $(x, y, 1)$ and multiply it by the provided `matrix`. You should return the result again as an *inhomogenous* vector $(x/w, y/w)$. I suggest storing the `matrix` as a list of `p5.Vectors` allowing you to use the `dot()` method, but it is up to you. More info here on geometric transformations as matrices.

3. Revise `vscale(v: p5.Vector, sx: float, sy:float): p5.Vector`

   - This should now use your `transform` method.

$$\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v.x \\ v.y \\ 1 \end{bmatrix}$$

4. `vtranslate (v: p5.Vector, x:float, y:float): p5.Vector`

   - This should use the your `transform` method. Pressing `k` should test this function.

$$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v.x \\ v.y \\ 1 \end{bmatrix}$$

5. `vrotate (v: p5.Vector, a:degrees): p5.Vector`

   - This should use your `transform` method.

$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v.x \\ v.y \\ 1 \end{bmatrix}$$

- Or adding in the translation bits:

$$\begin{bmatrix} \cos\theta & -\sin\theta & t_x \\ \sin\theta & \cos\theta & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v.x \\ v.y \\ 1 \end{bmatrix}$$

- And finally adding in the translation, rotation, and scale:

$$\begin{bmatrix} s_x\cos\theta & -s_y\sin\theta & t_x s_x\cos\theta\ -t_y s_y\sin\theta \\ s_x\sin\theta & s_y\cos\theta & t_x s_x\sin\theta + t_y s_y\cos\theta \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v.x \\ v.y \\ 1 \end{bmatrix}$$

6. `flip(curve: [[4 p5.Vectors] ...]): [[4 p5.Vectors] ...]`. Pressing `f` should test this function.

Figure 2: `flip(fish)`

7. `rot(curve: [[4 p5.Vectors] ...]): [[4 p5.Vectors] ...]` rotates curves by -90 degees. Pressing `r` should test this function.

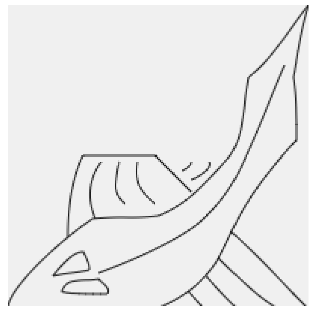


Figure 3: `rot(fish)`

8. `rot45(curve: [[4 p5.Vectors] ...]): [[4 p5.Vectors] ...]` rotates curves by -45 degrees and scales the image. Pressing `s` should test this function.

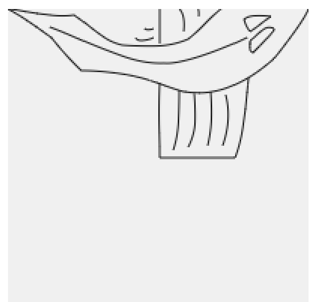


Figure 4: `flip(rot45(fish))`

9. `above(top, bottom, top_proportion, bottom_prop): [[4 p5.Vectors] ...]`. Pressing `a` should test this function.

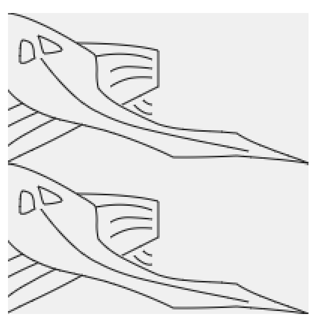


Figure 5: `above(fish, fish)`

10. `beside(left, right, left_proportion, right_prop): [[4 p5.Vectors] ...]`. Pressing `b` or `d` should test this function.

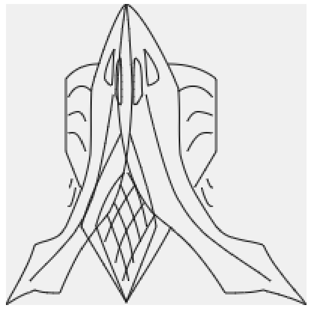
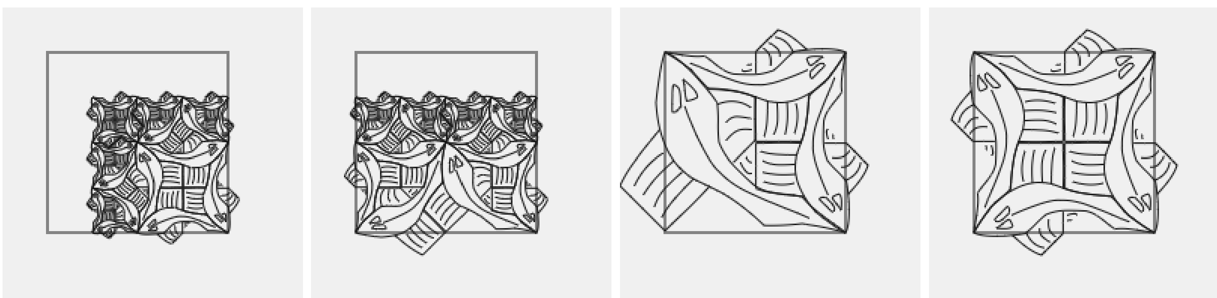


Figure 6: `beside(flip(fish), fish, 2/5, 3/5)`

11. `corner` and `side` are already complete, but here are images to help in debugging the other functions. The images are zoomed out a little (changed `sc` in `drawCurve`) to better see what's going on.

```
t= over(fish, over(smallfish, rot(rot(rot(smallfish)))))
u = over(over(over(smallfish, rot(smallfish)), rot(rot(smallfish))), rot(rot(rot(smallfish))))
```



2 Challenge Problem

- Build some other graphical pattern using this method. More info here.

3 Learning Objectives

- use the `p5.Vector` library
- use homogeneous coordinates
- perform matrix multiplication for 2D geometric transformations
- implement matrix multiplication
- procedurally generate graphics recursively

4 Deliverables

1. Commit the javascript `sketch.js` to the repo. Your sketch should use `key` to toggle between the different ways of viewing the graphics.

2. Write a small reflection (as a markdown document named `reflection.md`) about what you were able to accomplish in this lab. Don't forget the collaboration statement!