Intro to Generative Art
Problem Set 2

Harvard University
Taught by Sudhan Chitgopkar

This problem set is optional, but *highly encouraged*. You may work in groups and/or solicit outside help including but not limited to Generative AI tools. All outside sources/code used *must* be properly credited in the code documentation submitted.

1 Pool Party!

Understanding how forces, `PVectors`, and collision detection is a critical part of many natural and mathematical simulations. To build upon the toy bouncy ball model we developed during lecture, *develop an interactive game of pool*. You're welcome to modify your in-game physics as you see fit to make the code accessible and the game enjoyable. Our bouncy ball visualization starter code is located in the GitHub repository here.

2 How Do You Take Your Coffee?

During lecture, we visualized our first pattern in nature — the way light hits the inside of a coffee cup. As an exercise in converting this visualization into art, we considered an variety of modifications including changing the `multVal` (which modified light ray bounce direction).

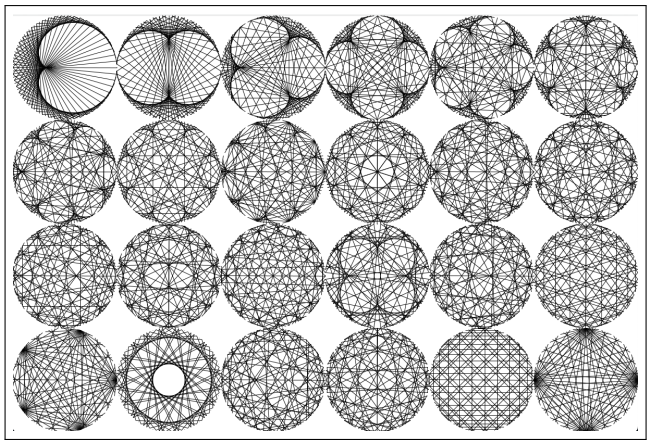

Figure 1: Results of incrementing `multVal`

In the same spirit, *modify a structural aspect of math behind the visualization to create your own piece of art from our coffee cup model*. While you're welcome to modify artistic elements like color or stroke weight, the focus of this exercise is to engage with the modular arithmetic driving the model to help you familiarize yourself with parameters that can be commonly changed beyond this visualization.

To get started, recall the code we wrote for our initial coffee cup visualization:

```
float numPoints = 200, r = 400;
int multVal = 2;

void setup () {
  frameRate(2); //slows down animation speed
  fullScreen();

  stroke(255);
} //setup

void draw() {
  background(0);
  translate(width/2, height/2);

  for (int i = 0; i < numPoints; i++) {
    line(r * cos(i/numPoints * TWO_PI), r * sin(i/numPoints * TWO_PI),
        r * cos(multVal * i/numPoints % numPoints * TWO_PI),
        r * sin(multVal * i/numPoints % numPoints * TWO_PI));
  } //for

  multVal++; //comment out to stop animation
} //draw
```