Generative Art with Perlin Noise

CMPE 49G - Project 2

Ramiz Dündar

2016400012

In this project I tried to generate visually appealing images using Perlin Noise. You can find all of the related code here:

https://github.com/ramizdundar/cmpe49g-project2

Note that Processing is required to run this.

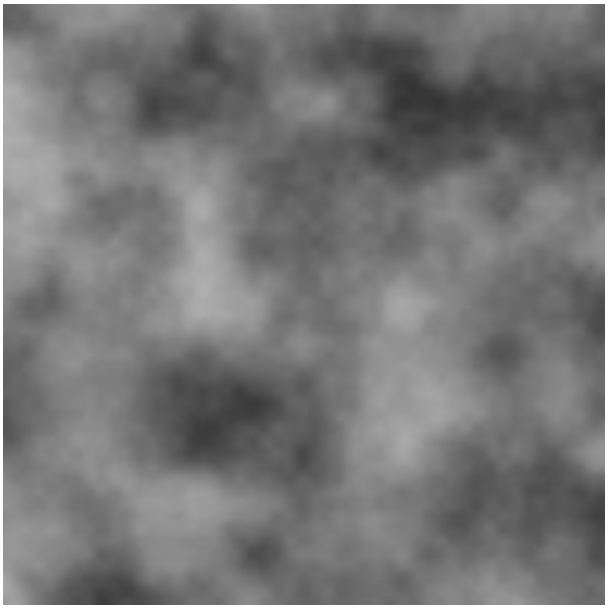
Also since Moodle file size is 2MB images are compressed. For better quality please check github.

I. Perlin Noise

Ken Perlin developed perlin noise in 1983 as a result of his frustration with the "machine-like" look of computer-generated imagery (CGI) at the time. He formally described his findings in a SIGGRAPH paper in 1985 called *An image Synthesizer*. He developed it at Mathematical Applications Group, Inc. (MAGI) for Disney's computer animated sci-fi motion picture *Tron* (1982). In 1997, Perlin was awarded an Academy Award for Technical Achievement for creating the algorithm. In 1997, he won an Academy Award for Technical Achievement from the Academy of Motion Picture Arts and Sciences for this contribution to CGI.

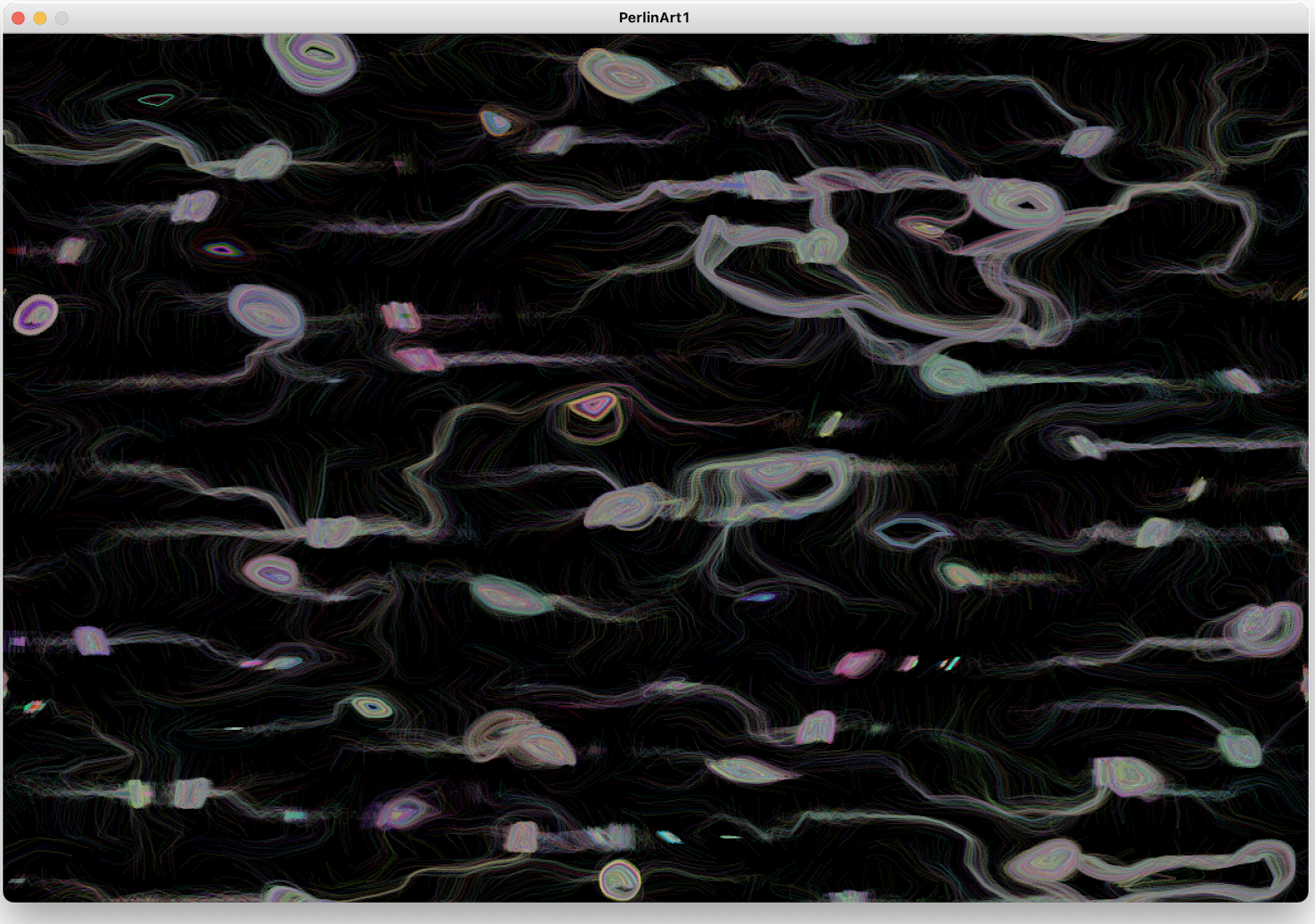
> To Ken Perlin for the development of Perlin Noise, a technique used to produce natural appearing textures on computer generated surfaces for motion picture visual effects. The development of Perlin Noise has allowed computer graphics artists to better represent the complexity of natural phenomena in visual effects for the motion picture industry.

Perlin did not apply for any patents on the algorithm, but in 2001 he was granted a patent for the use of 3D+ implementations of simplex noise for texture synthesis. Simplex noise has the same purpose, but uses a simpler space-filling grid. Simplex noise alleviates some of the problems with Perlin's "classic noise", among them computational complexity and visually-significant directional artifacts.[1]



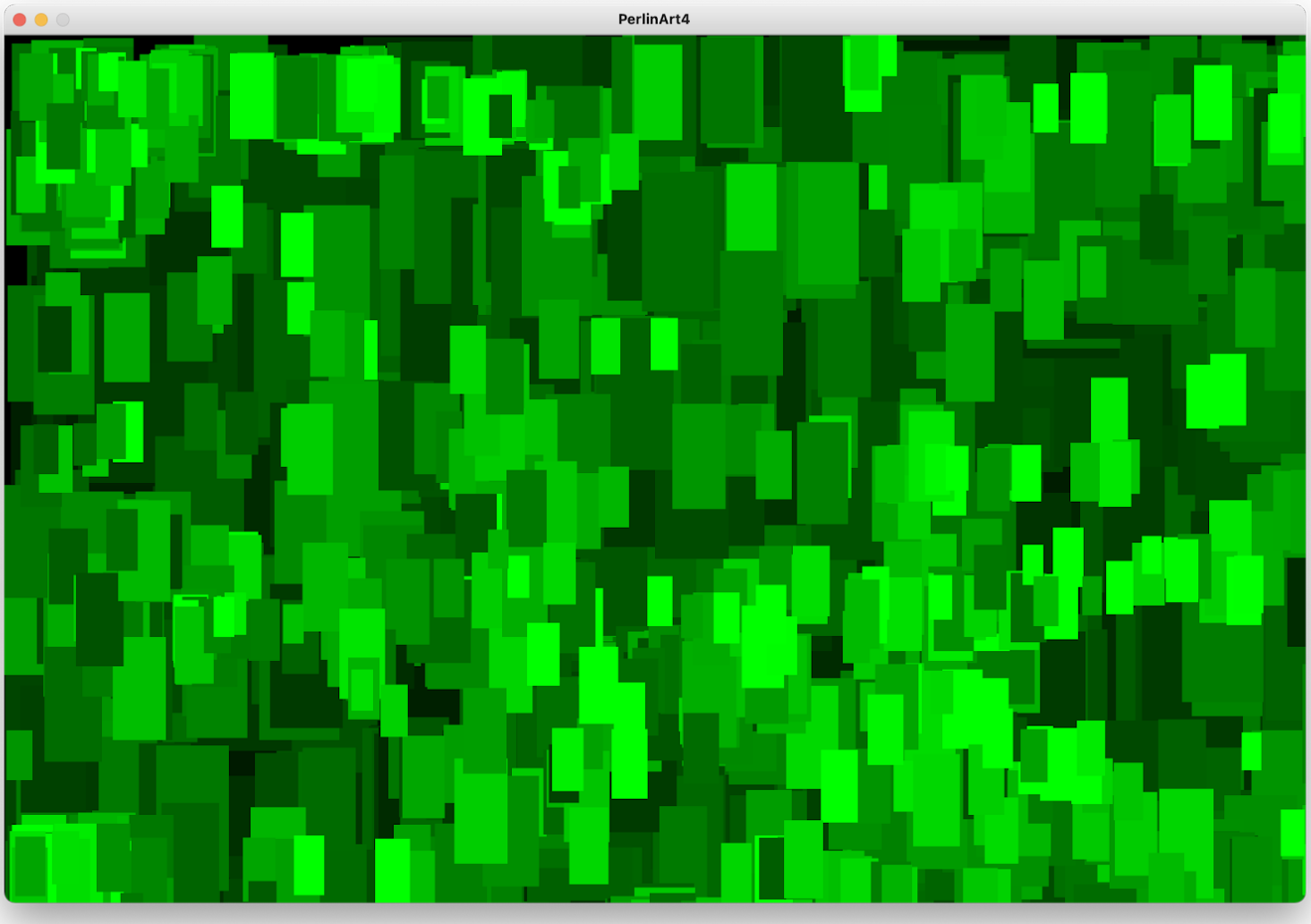
[1] https://en.wikipedia.org/wiki/Perlin_noise

II. 1. Gen: Rainbow Galaxies

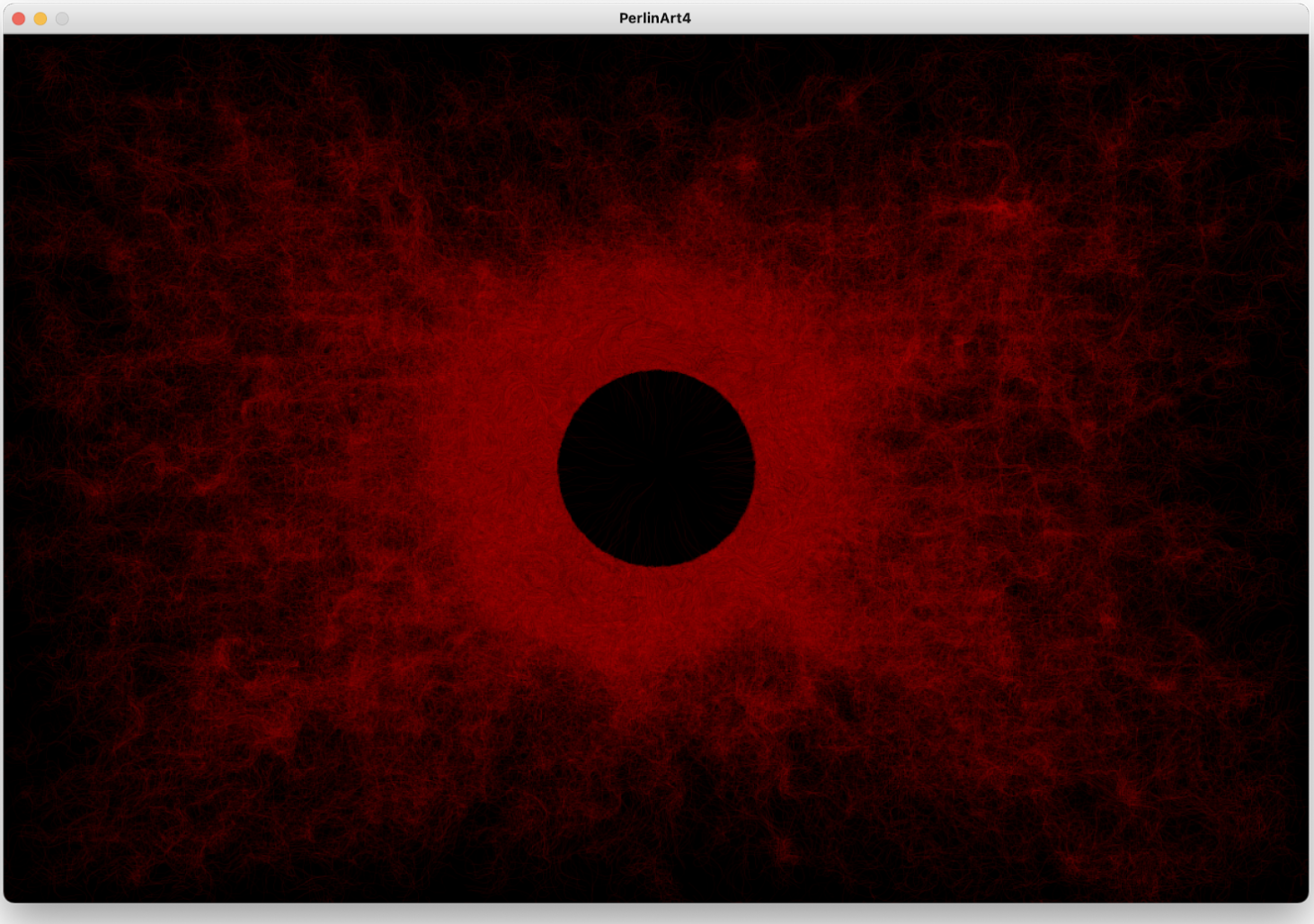


III. 2. Gen: Pink Blink

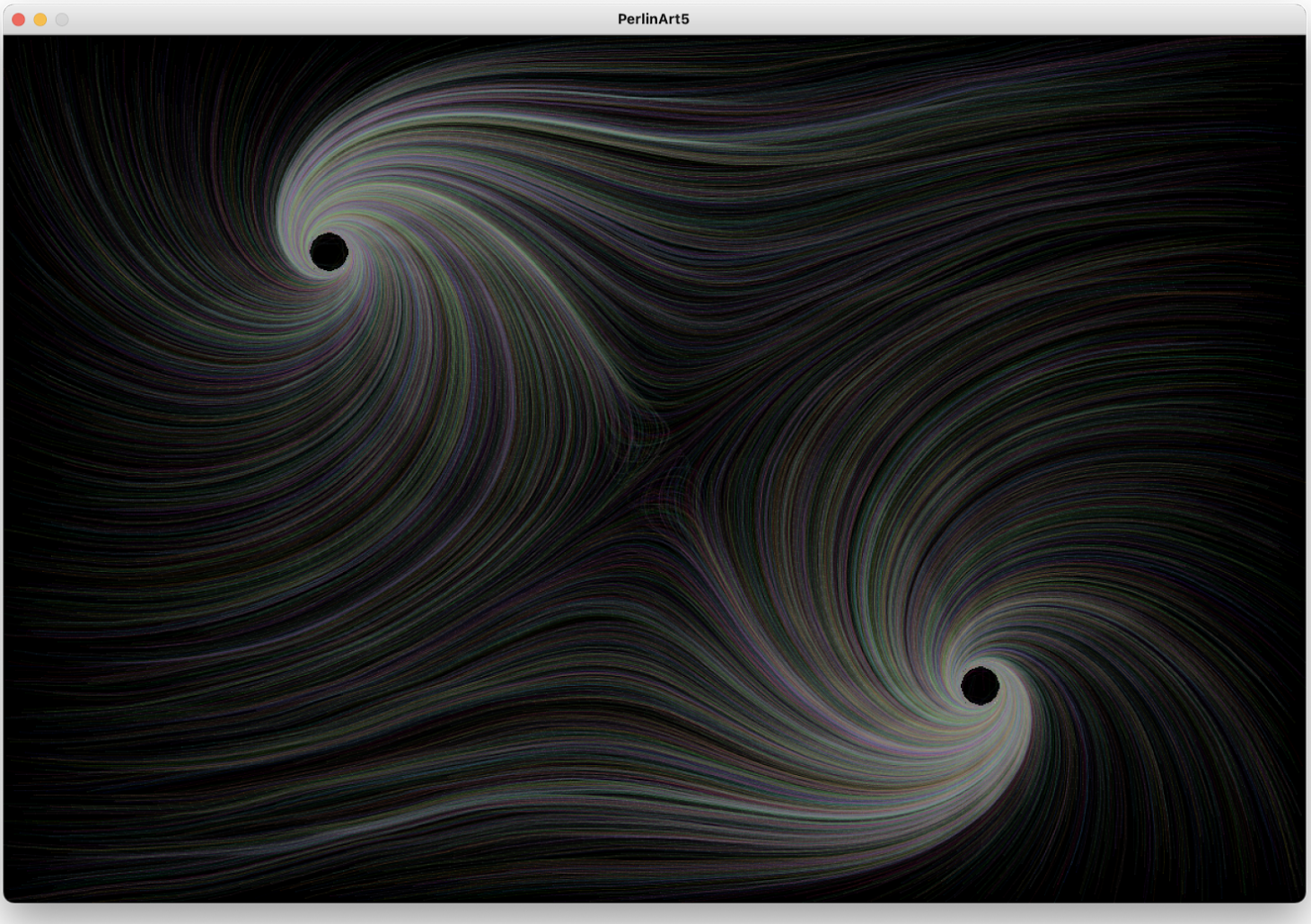
IV. 3. Gen: Greendow



V. 4. Gen: Red Devil



VI. 5. Gen: Eternal Twins



VII. Conclusion

Perlin Noise is an interesting and creative way to produce aesthetic images. However, it's strength lies in creating similar but non-repetitive patterns. Hence, while there may be great pieces to look at, I still prefer hand crafted pieces.

VIII. Appendix

Since I created a separate folder for each piece I will put only the 5th gen's code. All of the generations use codes in the lectures as a base. If you want to explore more of the code I suggest you to visit the github repo:

https://github.com/ramizdundar/cmpe49g-project2

Since I changed variables from each file-and sometimes variables in the functions- to get different results. I can't give a list of global variables that you can change and configure shapes. Means that most of the defined variables in the below code changed and used for getting above images. However, since I saved each picture as a separate sketch, you can get a similar image from the github repo. Also note that images could be different as I experimented with various settings and the above images are the ones I liked the most.

```
FlowField flowfield;
ArrayList<Particle> particles;

boolean isShowingFF = false;
boolean isDrawingTrace = true;

float ratioOfWhiteParticles = 1.0;
float flowFieldTimeStep = 0.007;

void setup() {
  size(1200, 800);

  flowfield = new FlowField(20, flowFieldTimeStep);
  flowfield.updateFF();

  initNParticles(100000);

  background(0);
```

```
}

void draw() {
  if (!isDrawingTrace) background(245);

  flowfield.updateFF();

  if (isShowingFF) flowfield.display();

  for (Particle p : particles) {
    p.applyFieldForce(flowfield);
    p.run();
  }
}

// This is how we init particles
void initNParticles(int n){
  particles = new ArrayList<Particle>();
  for (int i = 0; i < n; i++) {
    float maxSpeed = 10;
    PVector start_point = new PVector(random(width), random(height));
    if (random(1)<ratioOfWhiteParticles) {
      int r = (int) random(255);
      int g = (int) random(255);
      int b = (int) random(255);
      maxSpeed = 10;
      particles.add(new Particle(start_point, maxSpeed, r, g, b));
    }

     else
       particles.add(new Particle(start_point, maxSpeed, 0, 0, 0));
  }
}


public class FlowField {
  PVector[][] vectors;
  int cols, rows;
  float grid_inc = 0.1;
  float noise_time_off = 0;
  float noise_time_inc = 0.002;
  int scl;

  FlowField(int res, float ntime_inc) {
    scl = res;
```

```
    noise_time_inc = ntime_inc;
    cols = floor(width / res) + 1;
    rows = floor(height / res) + 1;
    vectors = new PVector[cols] [rows];
  }

  void updateFF() {
    float xoff = 0;
    for (int y = 0; y < rows; y++) {
      float yoff = 0;
      for (int x = 0; x < cols; x++) {
        float angle = noise(xoff, yoff, noise_time_off) * TWO_PI * 2;
        PVector v = PVector.fromAngle(angle);
        v.setMag(1);
        vectors[x][y] = v;

        xoff += grid_inc;
      }
      yoff += grid_inc;
    }
    noise_time_off += noise_time_inc;
  }


  void display() {
    for (int y = 0; y < rows; y++) {
      for (int x = 0; x < cols; x++) {
        PVector v = vectors[x][y];

        stroke(0, 0, 0, 150);
        strokeWeight(1);
        pushMatrix();
        translate(x * scl, y * scl);
        rotate(v.heading());
        line(0, 0, scl, 0);
        popMatrix();
      }
    }
  }
}


public class Particle {
  PVector pos;
  PVector vel;
```

```
PVector acc;
PVector previousPos;
float maxSpeed;
int r, g, b;
int cnt;

Particle(PVector start, float maxspeed, int pr, int pg, int pb) {
  maxSpeed = maxspeed;
  pos = start;
  vel = new PVector(0, 0);
  acc = new PVector(0, 0);
  previousPos = pos.copy();
  r = pr;
  g = pg;
  b = pb;
  cnt = 0;
}

void run() {
  updatePosition();
  edges();
  show();
}

void updatePosition() {
  pos.add(vel);
  vel.add(acc);
  vel.limit(maxSpeed);
  acc.mult(0);
}

void applyForce(PVector force) {
  acc.add(force);
}

void hole(float x, float y, float R){
  PVector converge2 = new PVector(x, y);
  converge2.setMag(R/(x*x + y*y));
  applyForce(converge2);
  PVector tangential2 = new PVector(-y ,x);
  tangential2.setMag(R/(x*x + y*y));
  applyForce(tangential2);
}

void applyFieldForce(FlowField flowfield) {
```

```
  int xx = floor(pos.x / flowfield.scl);
  int yy = floor(pos.y / flowfield.scl);

  PVector force = flowfield.vectors[xx][yy];
  applyForce(force);
  //PVector diverge = new PVector(pos.x - width/2,  pos.y - height/2);
  //if ((pos.x - width/2) * (pos.x - width/2) + (pos.y - height/2)* (pos.y - height/2) < 10000) {
  //  diverge.setMag(10);
  //  applyForce(diverge);
  //}
  float x = - pos.x + width/2;
  float y = - pos.y + height/2;
  float R = height*height + width*width;
  R /= 1;
  //x -= 300;
  //y -= 200;
  //PVector converge = new PVector(x, y);
  //converge.setMag(R/(x*x + y*y));
  //applyForce(converge);
  //PVector tangential = new PVector(-y ,x);
  //tangential.setMag(R/(x*x + y*y));
  //applyForce(tangential);
  //x += 600;
  //y += 400;
  //PVector converge2 = new PVector(x, y);
  //converge2.setMag(R/(x*x + y*y));
  //applyForce(converge2);
  //PVector tangential2 = new PVector(-y ,x);
  //tangential2.setMag(R/(x*x + y*y));
  //applyForce(tangential2);

  for(int i=0; i<9; i++) {
    float x2 = x + ((i%3)-1)*300;
    float y2 = y + ((i/3)-1)*200;
    hole(x2, y2, R);
  }

  //PVector gravity = new PVector(0,  1);
  //gravity.setMag(0.5);
  ////applyForce(gravity);



}
```

```
  void show() {
    if (isDrawingTrace)
      stroke(r, g, b, 15);
    else
      stroke(r, g, b, 225);
    //strokeWeight(3);
    strokeWeight(1);
    //strokeCap(SQUARE);
    line(pos.x, pos.y, previousPos.x, previousPos.y);
    //point(pos.x, pos.y);
    //if (cnt%60 == 0) {
    //  float w = random(100, 200) * pow(0.9, cnt/60);
    //  rect(pos.x, pos.y, w, 2*w);
    //  float g = noise(pos.x, pos.y, cnt)*100*pow(1.1, cnt/60);
    //  fill(0, g, 0);
    //  stroke(0, g, 0);
    //}
    cnt++;
    updatePreviousPos();
  }
  void updatePreviousPos() {
    this.previousPos.x = pos.x;
    this.previousPos.y = pos.y;
  }

  void edges() {
    if (pos.x > width) {
      pos.x = 0;//10*width-9*pos.x;
      updatePreviousPos();
    }
    if (pos.x < 0) {
      pos.x = width;//-9*pos.x;
      updatePreviousPos();
    }
    if (pos.y > height) {
      pos.y = 0;//10*height-9*pos.y;
      updatePreviousPos();
    }
    if (pos.y < 0) {
      pos.y = height;//-9*pos.y;
      updatePreviousPos();
    }
  }

}
```