1 Fractal Assignment Overview

Fractals are a big topic, this assignment intends to give a brief overview while allowing for exploration. Not all parts of this assignment are required to be understood or completed to get a "perfect" score. There will be a maximum of 100 points, simply collect enough points to win. For more details on how to win, refer to the lists of points in the sections below.

2 Introduction

The term comes from the Latin word "fractus" which means "to break". The field is still relatively new and even among mathematicians there is debate on how to most properly define fractals. There are a few key features of fractals that will be introduced. First we'll talk about the Cantor Set which was transformative in set theory. It is produced by taking a solid line, breaking it into thirds, removing the middle component (as an open interval) and repeating this infinitely. The process is illustrated below.

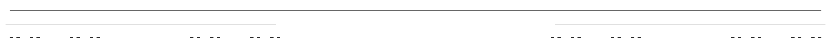

Figure 1: Cantor set: 0, 1, and 5 generations

If the length of the line is originally 1 and at each successive iteration it is $\frac{2}{3}$ of the prior iterations length, then at the n^{th} iteration the length would be $(\frac{2}{3})^n$. It follows that the length is zero at $n = \infty$ and that no point is connected to another. Since the point set is everywhere separated it can be thought of as having the same (topological) dimension of an individual point $D_T = 0$. The topological dimension is 1 higher than the minimum dimension required to separate it. The empty set is defined to have dimension $D_T = -1$. The empty set separates every already separated set. A separated set (ex: the Cantor Set) has the dimension 0 = -1 (separator) + 1 (one higher).

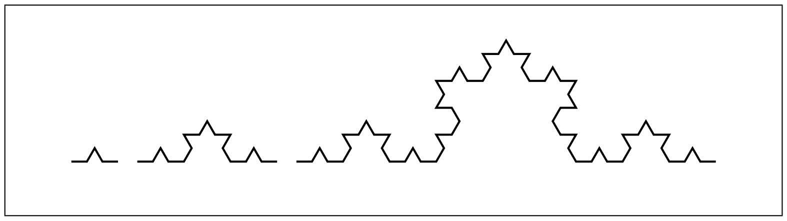

Figure 2: Koch Snowflake: 1, 2, and 3 generations

For simplicity of rendering, this is created not replacing but instead by growing. Notice that these all have a similar base shape and mentally scale them to overlap. The Koch Snowflake is a line made by (recursively) replacing every line segment with a new line segment which is a little bit longer than the prior line segment. To be more precise, a line is split into thirds, the middle third has 2 sides of equilateral triangle with the middle segment missing. This replacement has 4 segments and is $\frac{4}{3}$ longer than the prior segment. Iterating n times gives the new length of the line as $(\frac{4}{3})^n$. With an infinite number of iterations the length grows to infinity and yet it's clearly still a continuous line of $D_T = 1$. As the measurement scale decreases, the length increases, meaning that the measurement scale determines the line's length. This is the same phenomena behind the coastline paradox.

A square has dimension $D = 2$. Scale the length of an edge by $\frac{1}{2}$ and the area (or mass) will change by $(\frac{1}{2})^{D=2} = \frac{1}{4}$. Similarly, we can do this for fractals. For the Koch curve, each segment is scaled by $\frac{1}{3}$ of the length of the next larger segment yet has $\frac{1}{4}$ of the "mass" since it takes 4 segments to make the next larger segment. Now consider how the mass changes as the scale changes: $(\frac{1}{2})^D = (\frac{1}{3})$, $2^D = 3$. This is called the fractal (or Hausdorff) dimension $D_H = \frac{log(N)}{log(\epsilon)}$ where N is the number of sticks required to replace ϵ (the scaling ratio). This is just a number which helps classify how rapidly a particular pattern scales. For the Cantor Set $D_H = \frac{log(2)}{log(3)} = 0.630929...$ and for the Koch Snowflake $D_H = \frac{log(4)}{log(3)} = 1.261859...$ which is consistent with this chart. Check out this video for a better explanation.

Clearly this procedure of recursive replacement of similar (or identical) line segments can produce a variety of infinitely detailed sets. Each line could be replaced by any number of line segments in any arrangement (connected or disconnected). Below is the Peano Curve, a curve which has 9 segments and an original length of 3 of those segments. This leads us to $D_H = \frac{log(9)}{log(3)} = 2$. Somewhat surprisingly, this line ($D_T = 1$) has a fractal dimension of 2 and can fill the plane. This curve lives in a class called space filling curves which can fill any number of dimensions, ex: Hilbert curve (shows a 3D example). Another such curve was in the prior assignment, it is worth quoting Mandelbrot via The Fractal Geometry of Nature: "The Brownian motion's trail is topologically a curve, of dimension 1. However, being practically plane filling, it is fractally of dimension 2."

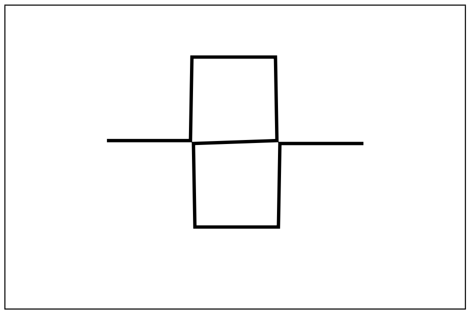

Figure 3: Peano Curve: generator pattern

Let's now take a look at a definition provided on wikipedia. "In mathematics, a fractal is a detailed, recursive, and infinitely self-similar mathematical set whose Hausdorff [or fractal] dimension strictly exceeds its topological dimension." This quote originates in Benoit Mandelbrot's *The Fractal Geometry of Nature.* Replacement is a recursive procedure that leads to increasing detail. Replacing with the same pattern leads to self-similar sets. Self-similarity is an important but not sufficient feature in fractals and can manifest in a variety of interesting ways (see Mandelbrot Set). We previously showed that $D_H > D_T$ for several fractal curves yet this leaves out objects which should perhaps be considered.

Below we'll look at a shape that resembles the "triforce" called the Sierpinski Gasket which is created by recursively replacing the smallest triangles with another (smaller) triforce. It scales by a factor of two making $\epsilon = 2$. It takes 3 of the i^{th} to produce the $(i+1)^{th}$, so $N = 3$. Using these numbers in the prior formula gives $D_H = \frac{log(3)}{log(2)} 1.5849$. At first glance this might appear to be topologically 2D, but it can easily be seen that the area vanishes to zero $(\frac{3}{4})^n$. The Sierpinski Gasket can be separated by just (2 or more) points (dimension 0) meaning it has a topological dimension of 1. This is independent of the primitive used (a filled triangle or lines).

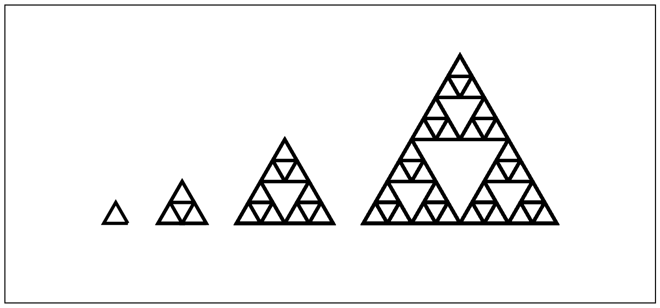

Figure 4: Sierpinski Gasket/Triangle/Sieve: 1 to 4 generations

3 L-Systems

An Lindenmayer system is a rule based string replacement procedure. Consider the rules $A \rightarrow ABA$ and $B \rightarrow BBB$ where A means draw forward and B means move forward. Starting with A, we have a line. Doing the replacement procedure gives us ABA which is a line separated into 3^{rd}s (like the Koch Snowflake without the central piece). One more application of the rule yields $ABABBBABA$ which is one way of producing the Cantor Set. In fact, L-Systems can produce a wide variety of things. Here's an L-System (made with LaTeX) generating a plant:

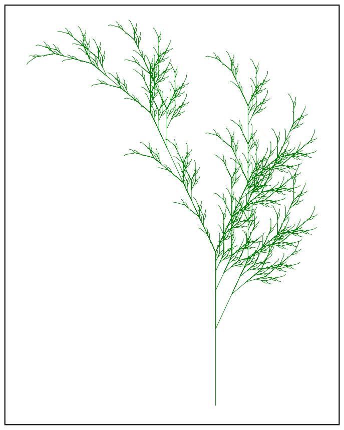

Figure 5: $X \rightarrow F-[[X]+X]+F[+FX]-X$ and $F \rightarrow FF$

Some new notation is introduced: $+, -, [,]$. $+$ and $-$ mean rotate left or right by the angle and [and] mean to save and restore the state respectively. The Koch snowflake can be made by moving forward, rotating (60 degrees), moving forward, rotating twice in the opposite direction, moving forward, rotating, and lastly moving forward. One can turn that pattern into an L-system production rule: $F \rightarrow F + F - -F + F$ and is exactly what's used to create the Figure 2. The rule $F \rightarrow FF$ as in Figure 5 means that the next layer's line length will be half of the current one.

Similar to the Cantor Set is the 2D version called the Sierpinski Carpet which is made up of squares divided into a 3 by 3 grid with the middle square removed. Each step replaces the prior square with a set of 8 surrounding the middle. At first glance the topological dimension would appear to be 2 since that corresponds to the topology of the replacement primitive. The Hausdorff dimension is $D_H = 1.892789 = \frac{log(8)}{log(3)}$ and would violate the prior definition that $D_H > D_T$. If instead, you look at the empty space at each iteration you see that at each step $\frac{8}{9}$ of the remaining area becomes empty. At an infinite number of iterations the area is $0 = (\frac{8}{9})^\infty$ which means the topological dimension is not 2. In fact, since this is the 2D version of the Cantor Set, it can be used as the separating set (most clearly orthogonally through the middle). Since the Cantor Set has $D_T = 0$ the Carpet has $D_T = 0 + 1 = 1$. Note that the boarder stays fully connected.

4 Fractal Trees

In this section you will use processing to generate a fractal tree. Below is a basic setup to get you started which draws a line from the top left to the bottom right. You're allowed to change the API, this boilerplate is merely a suggestion. Fractal 2-branch binary trees follow a simple pattern: draw a trunk, split left/right, repeat. You'll be creating a recursive function which will draw a line segment and then recursively call itself with different starting locations and angles. Before continuing, let's compute the Dimensions. The lines could be separated with a point so its topological dimension is 1 and the Hausdorff dimension is $D_H = 1 = \frac{\log(2)}{\log(2)}$ meaning this somewhat surprisingly (given the name and assignment) doesn't classify as a fractal via the prior definition. It does, however, satisfy the definition when there are 3 segments.

1. (0 points) Make a line rotate in a circle with one point at the center of the canvas.
2. (25 points) Fractal tree 2-branching recursively
3. (15 points) Fractal tree 3-branching into the Sierpinski Gasket
4. (10 points) Mouse over leaves change color
5. (5 points) Animate the tree somehow
6. (5 points) Stroke thickness reduces every layer
7. (10 points) Creative stuff, stylize it (make it look good/interesting)
8. (30 points) Fractal Bezier Tree (derivatives must match use processing's bezier())
9. (20 points) Fractal Tree in 2D with a vanishing point (as $\frac{[x,y]}{z}$)
10. (20 points) Fractal Tree in 3D with a ground plane of cubes

```
void setup(){
  size(600,600);
}
PVector drawtree(PVector start, float angle, float length, int layer){
  return new PVector(0,0);
}
void draw(){
  background(255);
  line(0,0,width,height);
  //drawtree(...);
}
```

4.1 Part 1 - Rotating Line

This is simply a starting point in case the 2-tree sounds too difficult. The width and height environment variables in processing can allow you to define a point o at the center of the display. Another point r away from this central point can orbit o by using the following trigonometric functions $x = cos(\theta f(t))$ and $y = sin(\theta f(t))$ where $f(t)$ is a time varying function (ex: $f(t) = t$).

4.2 Part 2 - Fractal 2-Tree

The algorithm: draw a line, rotate left/right, and call the parent function. Below you can see a diagram of four iterations of an L-system growing a fractal 2-tree. Rather than growing like this from left to right, you'll be starting by drawing the base of the trees from right to left at the appropriate locations and rotations. Each time you call the parent function you'll reduce the length of the line so that the size doesn't explode. You might choose a scaling factor of 2 which allows the maximum distance from branch to the leaf as 2 times the length of the current branch. Remember to have an exit condition in your recursive function (layer > 1 ? drawTree : drawLeaf). You'll likely also want to stop drawing if the length of the segment is smaller than some value (ex: 1 pixel).

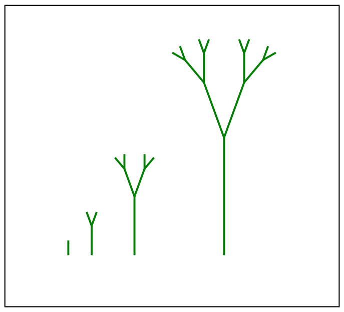

Figure 6: Fractal Tree

4.3 Part 3 - Fractal 3-Tree

Rather than drawing a trunk, we draw 3 lines out at 120 degrees (resembling the Mercedes logo). Proceeding, as before, you'll see that it is a fairly trivial extension from the 2-tree. The fractal 3-Tree (Gasket) below is 8 iterations of the L-system rules $X \rightarrow [FX][-FX][+FX]$ and $F \rightarrow FF$. If you look at the center, you'll see something resembling the cross section of a cube which has points at the edges of the central triangle an in the next iteration's centers. While this cross section appears to come from a Menger Sponge, it is not. This has a scaling of $\frac{1}{2}$ not $\frac{1}{3}$.

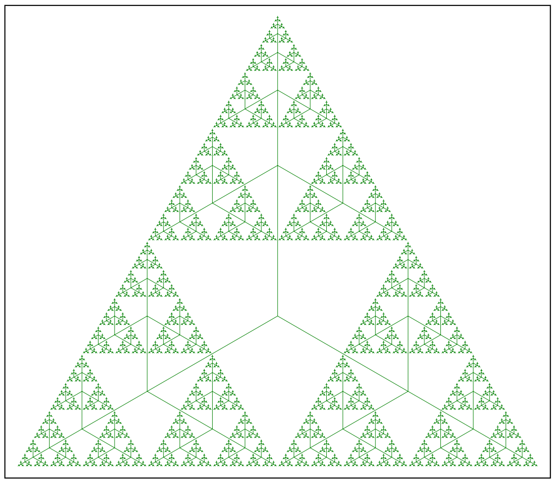

Figure 7: Sierpinski Gasket via a Fractal Tree

The gasket slices the 3D version of the following L-System.

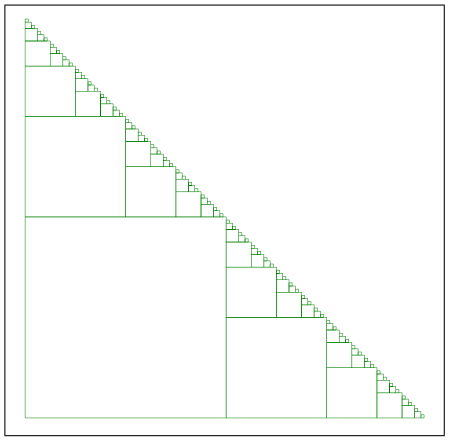

Figure 8:
$X \rightarrow F[X] - F - F[++X] - F$ and
$F \rightarrow FF$

4.4 Parts 4-7 - Fun Features

In order to make the leaves change color upon mouse over, you'll need to calculate the (Euclidean or L^2) distance between the mouse and the leaves. Obtain the mouse position with mouseX and mouseY. You might also want the mouse to perturb the locations of the leaves or change the angle of the branches. Feel free to explore here.

4.5 Part 8 - Bezier Tree

Linear interpolation is most simply connecting two points with a straight line and breaking it up into equal segments. Bezier curves are most simply curves generated from layers of linear interpolation. Consider 3 points (A, B, C). Take a point $p\%$ of the way from A to B and $p\%$ from B to C, connect these by another line and then take the point $p\%$ of the way along that line to BC. De Casteljau's algorithm is a way to efficiently compute bezier curves. Fortunately for you, Processing has a very convenient API for drawing Bezier curves with the function bezier(). The format for the parameters are (starting point, starting direction, ending direction, ending point). Those locations are control points for cubic Bezier curve (with 4 control points).

Your task will be to start drawing at the starting point move towards the first split and curve to the ending point of the next branch. You'll need to adjust where the branch begins and ends. The branch will begin at the midpoint of the original parent branch and will end at the midpoint of the original child branch. Use the original ending point as the direction vector for both the starting and ending direction. This is why the template code above returns a PVector but it may be easier via some other approach. If you get this working, you'll have a result similar to the following. And, if not, don't fret, there are other points to be had in this assignment.

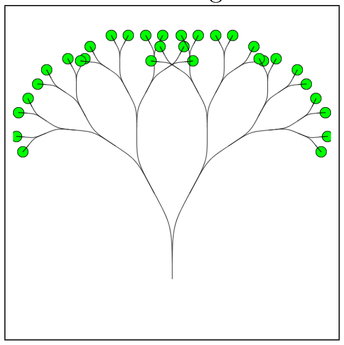

Figure 9: Bezier Tree

4.6 Part 9 - Vanishing Point

In this section you can create a scene that looks somewhat 3D by dividing the x and y coordinates by z. As an object gets further away it appears smaller, this is what the division does. This form of projection is known as weak perspective projection. Create a grid as shown in the example and scale the base of the tree accordingly. You might consider drawing other objects at a variety of depths to further show the shrinking effect.

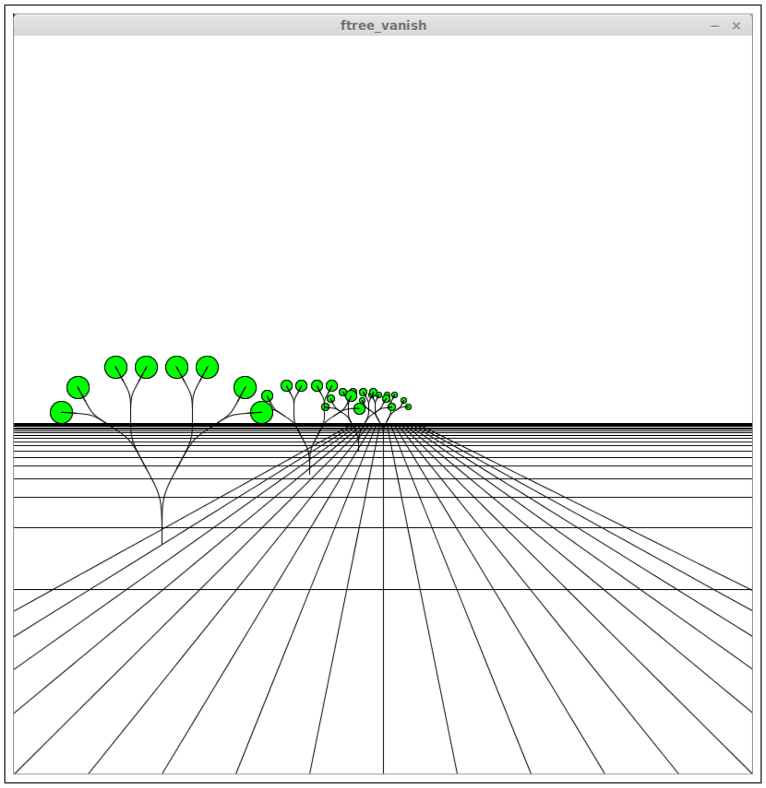

Figure 10: Bezier Tree with a vanishing point

4.7 Part 10 - 3D

In this section you can create a scene that uses P3D, a camera, light, and ambient material. Instead of a fractal 2-tree, you'll create a fractal 4-tree where each branch will be rotated 90 degrees about the up axis. You can decide to render the leaves as spheres or not. I happened to think it looked somewhat better (more wintry) without rendering leaves. What's the Hausdorff dimension of of this 4-tree?

```
//https://processing.org/reference/beginCamera_.html
void setup(){
  size(100, 100, P3D);
  beginCamera();
  camera();
  rotateX(-PI/6);
  endCamera();
  noStroke();
}

int t = 0;
void draw(){
  pointLight(255, 255, 255, 100, 100, 100);
  background(200);
  t += 1;
  translate(50, 50, 0);
  rotateY(PI/3 + 0.01*t);
  ambient(255);
  //sphere(45);
  box(45);
}
```

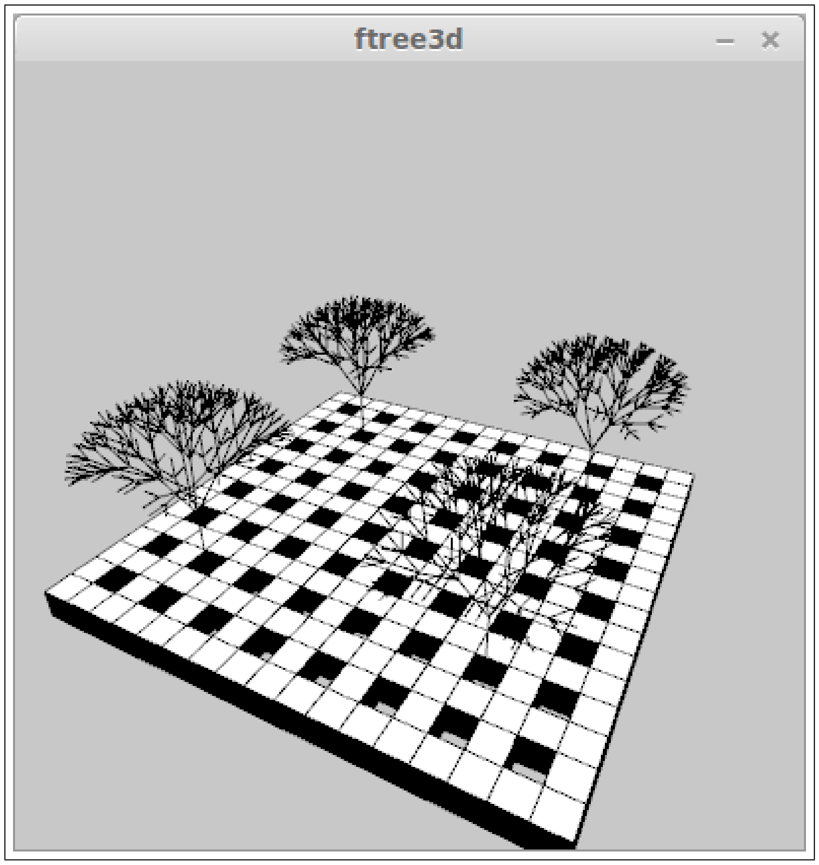

Figure 11: Fractal Tree in 3D with a cube patio

5 Extras

1. (10 points) Write a page about fractals
2. (10 points) Sierpinski Carpet
3. (10 points) Menger Sponge

5.1 Report

If you feel like exploring more of the theory, different types of fractals, and the applications, this part is for you. Write at least a page exploring some area you find interesting.

5.2 Carpet

In this section you'll recursively step through layers and draw 8 squares only at the last layer. The dimensionality of this was discussed previously.

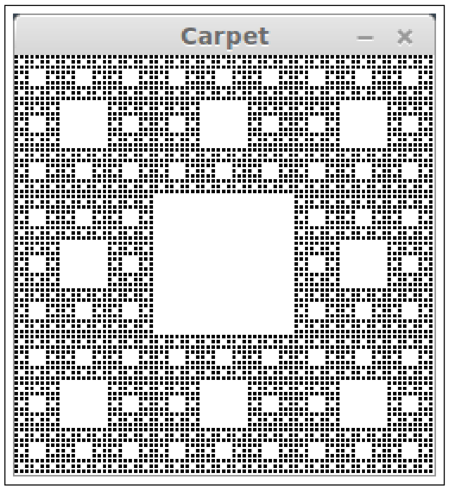

Figure 12: Sierpinski Carpet

5.3 Sponge

In this section you'll recursively step through layers and draw cubes only at the last layer. The dimensionality of this is left as an exercise for the reader.

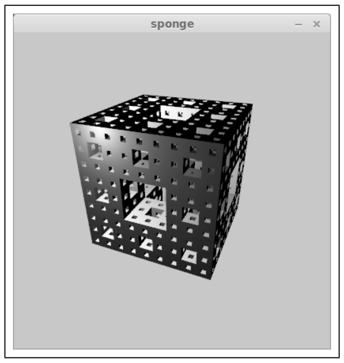

Figure 13: Menger Sponge

6 Sumbissions

6.1 What to include

1. Submit a zipped folder (using your last/family name)
2. A readme.txt file
   (a) Your name
   (b) Which sections you did
   (c) How many points you expect in each section
3. Each part should be a processing (java) sketch in its own folder.
4. Example submission zip structure:
   (a) readme.txt
   (b) 5-1-report.pdf
   (c) fractaltree/fractaltree.pde
   (d) fbeztree/fbeztree.pde
   (e) fvptree/fvptree.pde
   (f) sponge/sponge.pde
5. I should be able to open a sketch and click play.
6. Don't include image or movie files

6.2 Grading

1. There are ∼170 points available, but your points earned will be capped at 100.
2. Points will be awarded for correctness, completion, and elegance.
3. Practice your best software engineering on all assignments.
4. If you think you might get points off, do an extra part (or two).

6.3 Why

1. There are extra parts to accommodate different expertise and interests.
2. The extra points are meant to reduce stress associated with typical grading metrics.

6.4 Warning

Don't copy code from elsewhere! If you find references that help, include them in your readme file (and maybe a comment in your code). Do your work honestly, don't let paranoia detract from learning. Instead, let curiosity guide you.