Random Function Report
Software Design - Fall 2015

Written by

Amanda Sutherland

October 13, 2015

Olin College of Engineering

Contents

List of Figures

1 Project Overview

This mini-project aimed to solidify our understanding of recursion within the context of creating computational art.

2 Implementation

This project, as with the last one, was implemented in one script. This script, the shell of which was provided, is split into functions based on the operations necessary to create and work with random functions. The script is generally split into the following sections:

1. Build the random functions
2. Put these functions into a list so they are in a good format for doing recursion with
3. Evaluate the function, finding its mathematical value
4. Map the functions to colors based on their depth
5. Create an image based on these colors and functions

These points are not the functions themselves, but describe the order of the steps. As it is a single script, I tested it incrementally in the python command line interpreter.

2.1 Outside Material

Pillow is used to create images, and lambda functions are used to represent nested functions. Lambda functions are particularly useful as it makes it possible to entirely bypass the step where the functions are evaluated, as it does that already.

2.2 Running the Code

Running the code is simple - run "random_art.py" in the terminal, first changing the name of the figure in the final function to run the code.

3 Outputs

The script outputs an image! As per the current state of the code, it outputs a single image as figure1.png to the folder the script is in. The figures below are the result of running the code for three different scenarios. I changed the depths allowed by changing the relevant section of code in build_random _function.

1. Figures - Take 1

```
if min_depth > 1:
    i = randint(0,2)
else:
    i = randint(0,6)
return functions[i]
```

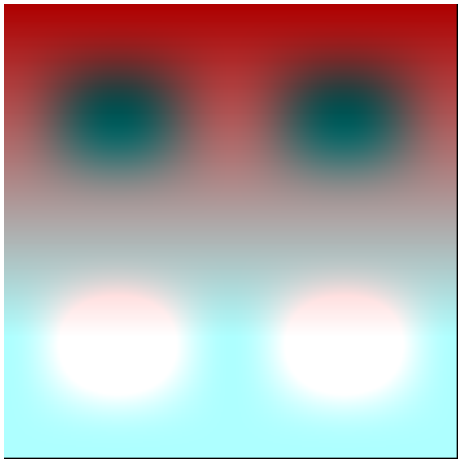

Figure 1: Take 1 - 1

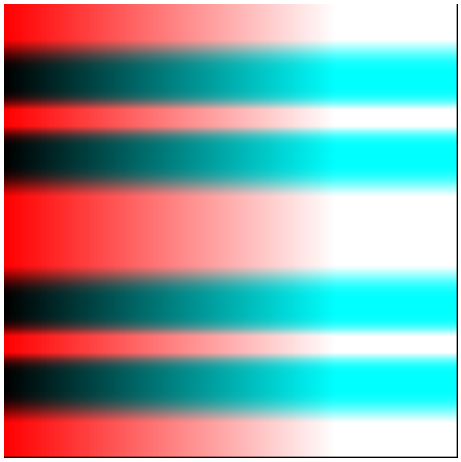

Figure 2: Take 1 - 2

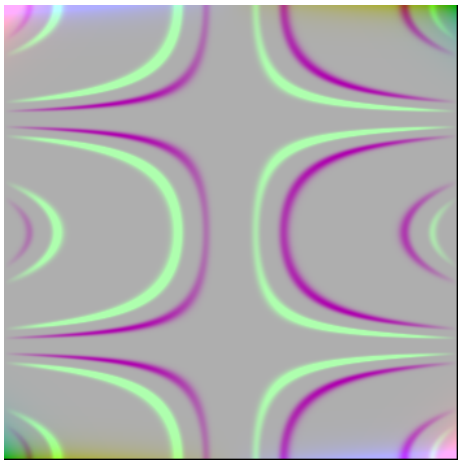

Figure 3: Take 1 - 3

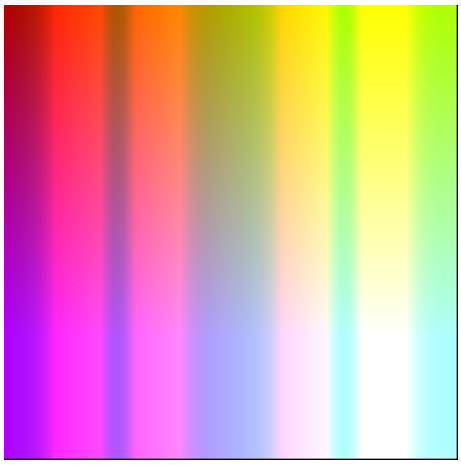

Figure 4: Take 1 - 4

2. Figures - Take 2

```
if min_depth > 1:
    i = randint(0,5)
else:
    i = randint(0,6)
return functions[i]
```

Figure 5: Take 2 - 1

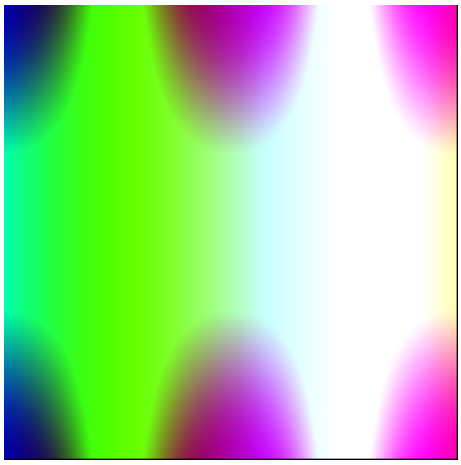

Figure 6: Take 2 - 2

Figure 7: Take 2 - 3

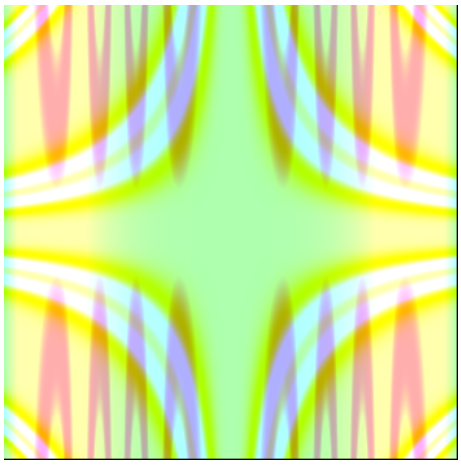

Figure 8: Take 2 - 4

Figure 9: Take 2 - 5

3. Figures - Take 3

```
if min_depth > 1:
    i = randint(1,3)
else:
    i = randint(0,6)
return functions[i]
```

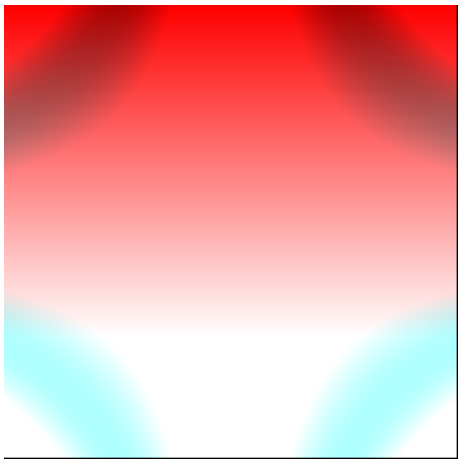

Figure 10: Take 3 - 1

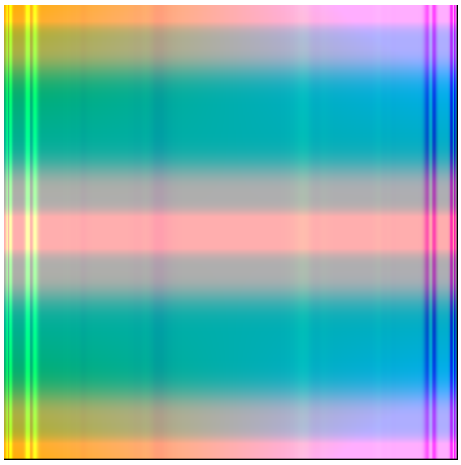

Figure 11: Take 3 - 2

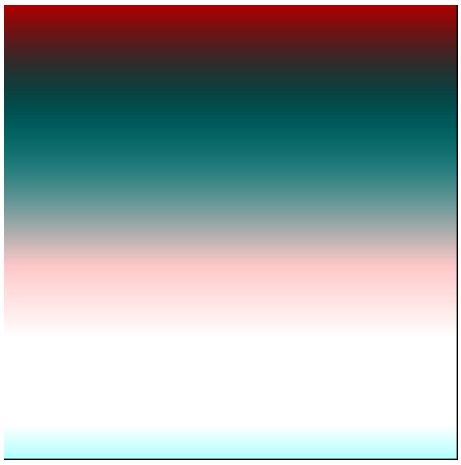

Figure 12: Take 3 - 3