# Exploring Fractals
Portfolio

Jean Choi
Portland State University

Summer 2020

## Contents

## 1 Koch Curve

### 1.1 “Koch Winter”

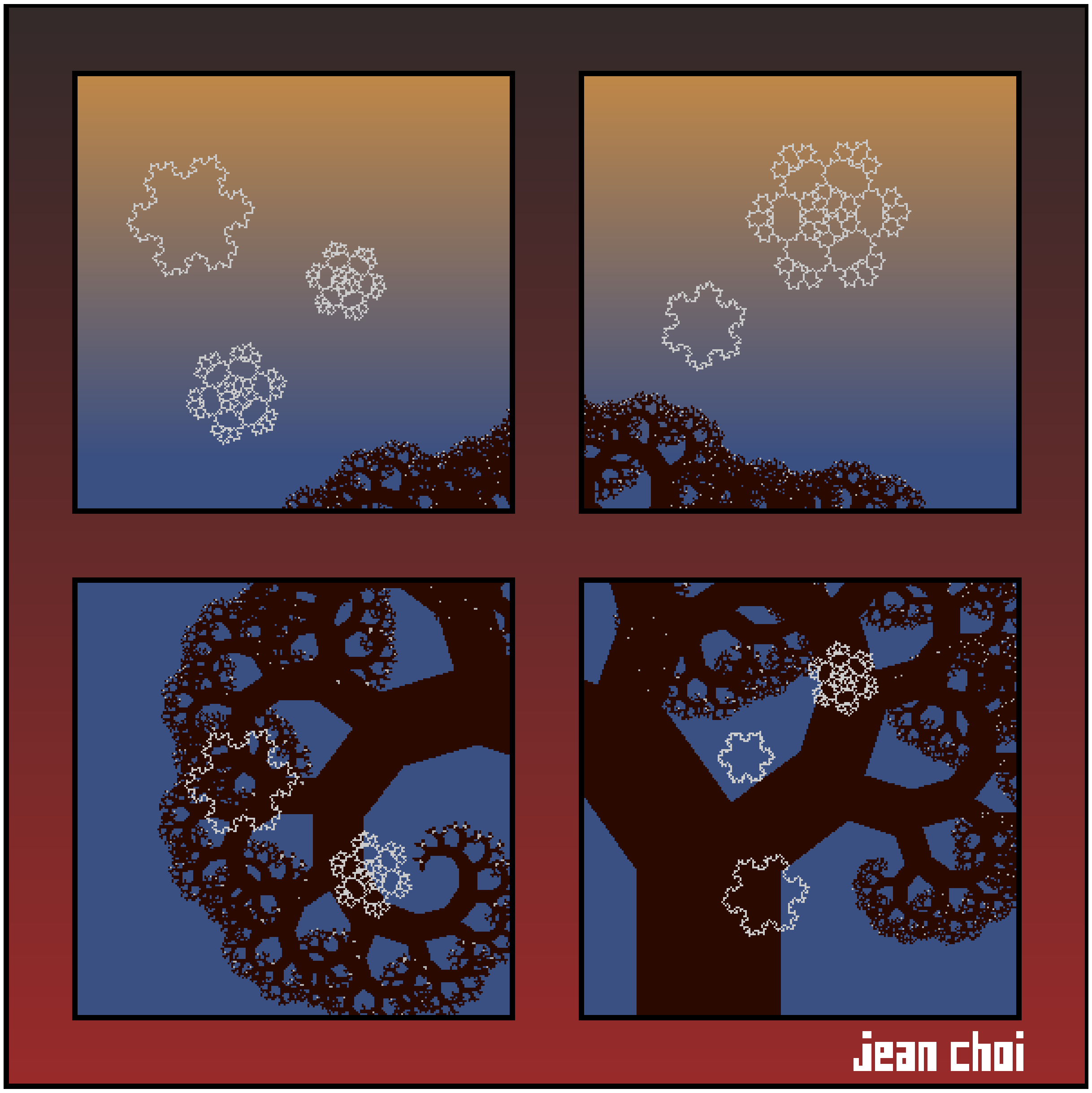

Figure 1: “Koch Winter”

### 1.2 Fractal Description

Pythagoras Tree: Created in 1942 by Dutch mathematics teacher Albert E. Bosman
Koch Curve: Based on the Koch Curve, which appeared in a paper written by Swedish mathematician Helge von Koch in 1904.

### 1.3 Design Paradigm & Mathematical Description

#### 1.3.1 Pythagoras Tree

The Pythagoras Tree is generated by recursion. My recursive function `pytree(double x0, double y0, double x1, double y1, int n, double scale, double height)` has seven parameters, but the last two parameters are not absolutely necessary. The first four parameters create two points, `p0` and `p1`, and the fifth parameter determines the number of recursive calls.

First, the function needs to determine the second set of points, `p2` and `p3` to make the a square. This can be calculated using the initial points passed into the function `(x0, y0)` and `(x1, y1)`. In my program, `height = 1`, so a square is generated.

```
       x2 = x1 - (y1 - y0)
       x3 = x0 - (y1 - y0)
y2 = height * y1 + (x1 - x0)
y3 = height * y0 + (x1 - x0)
```

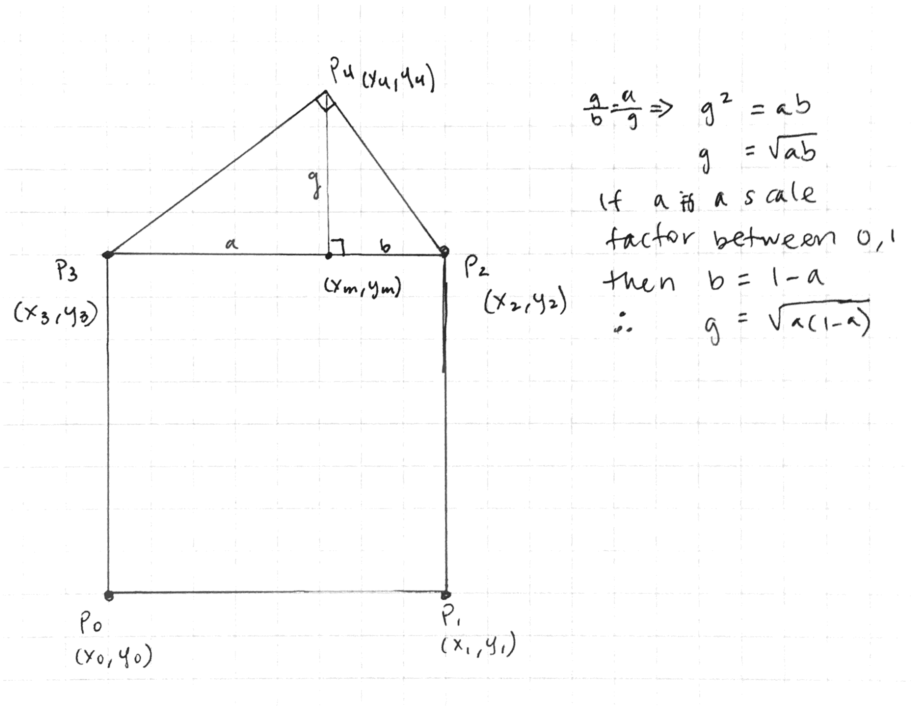

Figure 2: Pythagorean Tree representation on a x,y-coordinate graph

`p2` and `p3` are the hypotenuse of a right triangle, but the function needs to determine `p4`, the final point of the right triangle. Before we can calculate that, we need to pick a point between `p2` and `p3` using a scale factor between 0-1 called `m`. Because points `p3, m, p4` form a similar triangle to `p2, p3, p4` and `p2, m, p4`, we can use those properties to calculate `g`.

```
 g = sqrt(scale*(1-scale))
xm = x3 + scale * (x2 - x3)
ym = y3 + scale * (y2 - y3)
 x4 = xm - g * (y2 - y3)
 y4 = ym + g * (x2 - x3)
```

Finally, the recursive call can be called on `p3, p4` and `p2, p4` to generate the next two branches of the tree.

### 1.3.2 Koch Snowflake

The Koch Curve is generated using recursion and rotational geometry. The recursive function takes two points `p, q` on an x,y-coordinate grid as well as the number `n` recursive calls the function should make. First, the points `r, s` must be calculated. Points `r` and `s` must be equally distributed along the line between `p` and `q`.

```
rx = px + 1.0/3 * dx
ry = py + 1.0/3 * dy
sx = px + 2.0/3 * dx
sy = py + 2.0/3 * dy
```

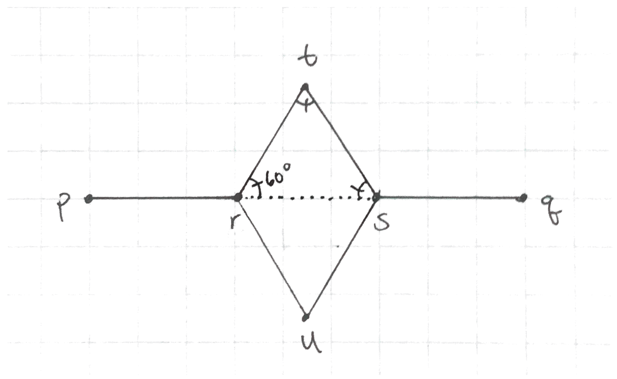

Figure 3: Koch Curve diagram

Point `t` is the third point of an equilateral triangle between points `r, s, t`. This can be calculated using rotational geometry. For my rendition of the Koch curve, I am using an additional point at `u`, which forms the third point of a second equilateral triangle between points `r, s, u`.

```
  double c = cos(60 * M_PI/180.0)
  double s = sin(60 * M_PI/180.0)
 double c1 = cos(-60 * M_PI/180.0)
 double s1 = sin(-60 * M_PI/180.0)
tx = (sx-rx) * c - (sy-ry) * s + rx
ty = (sx-rx) * s + (sy-ry) * c + ry
ux = (sx-rx) * c1 - (sy-ry) * s1 + rx
uy = (sx-rx) * s1 + (sy-ry) * c1 + ry
```

The Koch curve makes four recursive calls, one for each line segment `p-r`, `r-t`, `t-s`, `s-q`. The Koch curve is drawn at the very last recursive call.

In order to make a Koch Snowflake, there must be three initial line segments `p-q, q-o, o-p` that form an equilateral triangle. The Koch recursive function is applied to these three line segments to generate the Koch Snowflake.

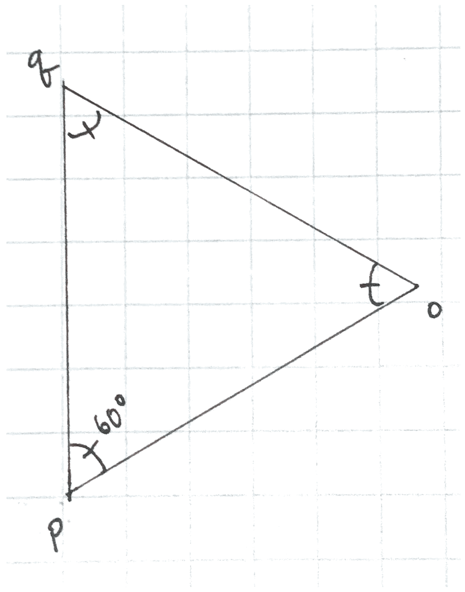

Figure 4: Initial Koch Snowflake diagram

## 1.4 Artistic Description

I wanted to evoke the sense of a winter wonderland in this piece. The perspective is from the inside of a warm house, watching the snow peacefully fall at sunset. It was fortunate that the first two fractals we coded happened to be a barren-looking tree and a snowflake.

**Background:** For the background, I wanted to have the color change between a dark blue and orange to imitate a sunset. Instead of linearly blending for the entire height of the screen, I decided to start the blend from slightly above the midpoint the screen height because I wanted to have the majority of the background to be dark blue in order to contrast with the other warm colors.

**Pythagoras Tree:** I wanted the tree to not be the focal point, but found that a single color made it look very flat and uninteresting. Since I wanted to draw a winter landscape, I thought it would be nice if the tips of the tree branches were white, to imitate snow that has just fallen.

**Koch Snowflake:** Before I discovered that the Koch Anti-Snowflake existed, I had accidentally programmed a piece of code that calculated the initial triangle's point +60°instead of -60°. I thought it would be interesting to combine this with the Koch Snowflake, and the result turned out to be even more realistic to a a real snowflake than the Koch Snowflake. In my drawing, I colored it to be slightly off-white so that they would not be so jarring against the rest of the colors.

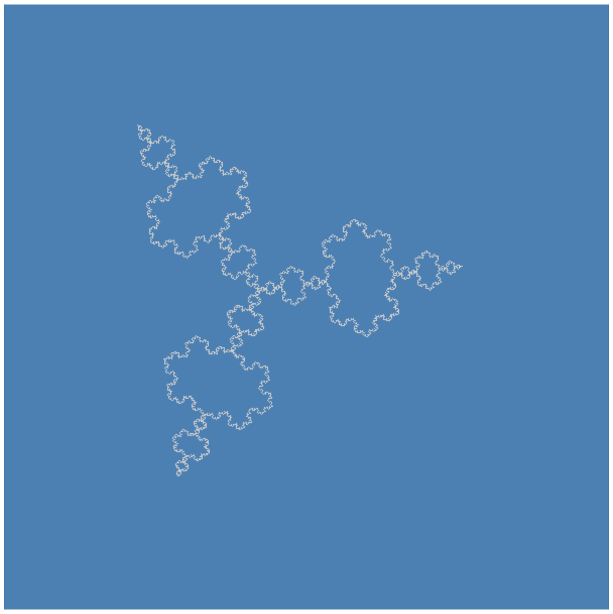
Figure 5: Koch Anti-Snowflake

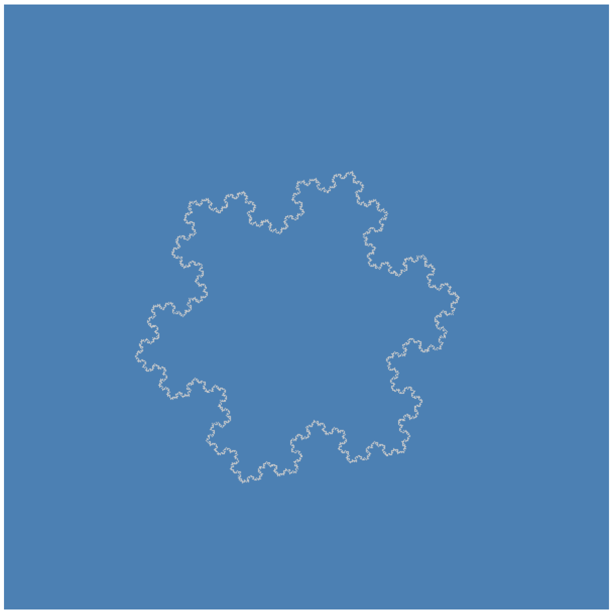
Figure 6: Koch Snowflake

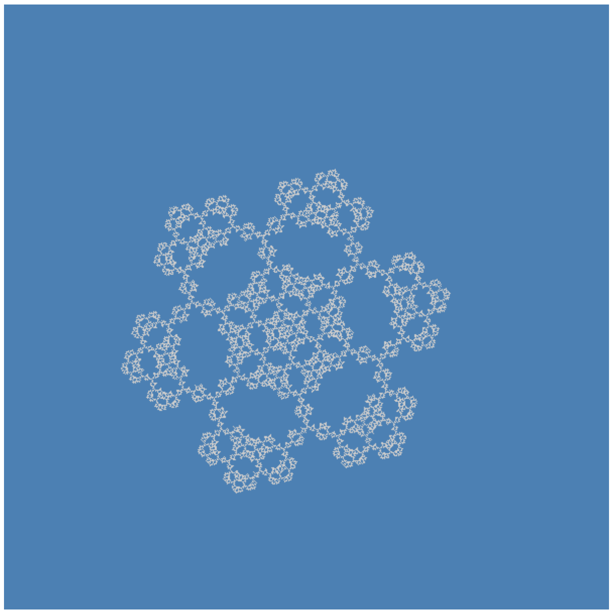

Figure 7: Combined Koch Snowflake

**Window:** I wanted the perspective to be from indoors, so I did linear-blending between a warm red and a cool brown. I hoped that the warmer tones toward the bottom of the piece would remind the viewer of a warm fire behind them. I also added a small 2-pixel border around the window panes to frame the piece better

**Signature:** Just like any art piece, I wanted to make sure that I had my signature on the piece. I thought white would stand out nicely against the wood panel of the window.

## 2 Iterated Function System

### 2.1 "I'd like a Mai Tai, please."

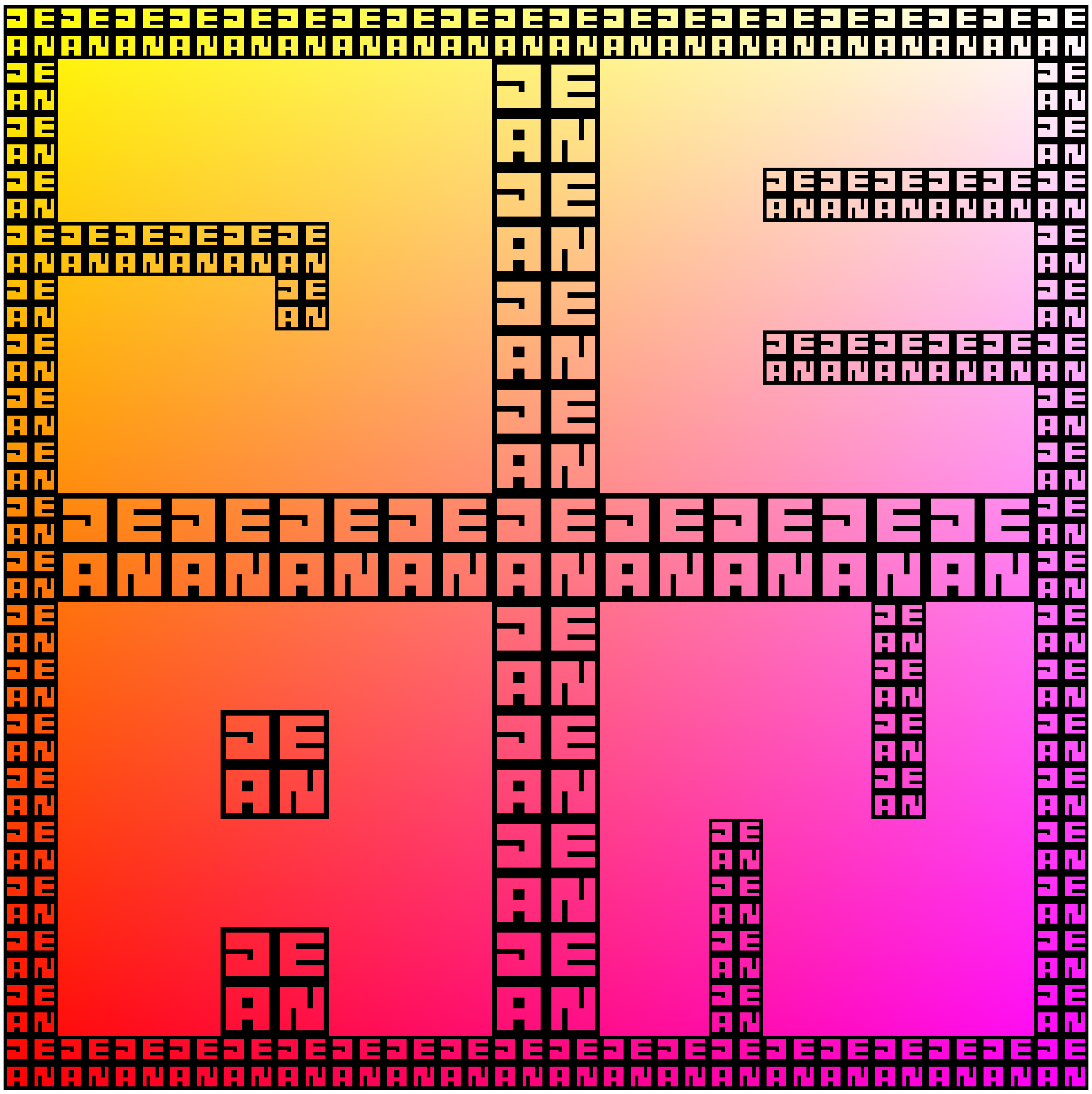

Figure 8: "I'd like a Mai Tai, please."

## 2.2 Fractal Description

Iterated Function Systems: Created in 1981 by John E. Hutchinson.

## 2.3 Design Paradigm & Mathematical Description

Iterated Function Systems map points to a metric space by using certain 'rules' that scale, translate or rotate the start shape into smaller iterations of itself. For this particular piece, I am not making any rotations, only scaling and translating.

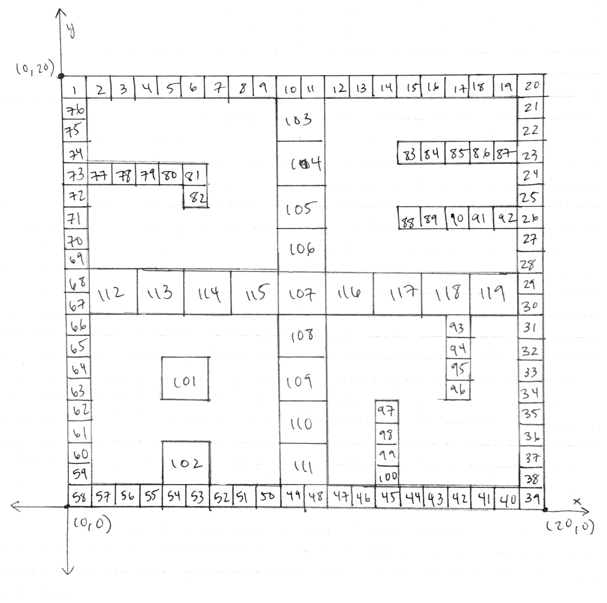

Figure 9: Initial sketch of image

While many fractals generated by iterated function systems have only a handful of rules, I had 119 because I wanted to retain the square geometric motif in my piece. While the `while` loop runs, a point gets mapped to the plane depending on which transformation is made in a particular `switch case`. Each following point uses the previous point as its reference, until the full image is built.

For the image to be at full saturation, there must be many iterations of the `while` loop, in order to have a higher probability for filling in each desired pixel.

## 2.4 Artistic Description

My first iteration of this fractal was using iterated function systems to draw my initials, J.Y.C.

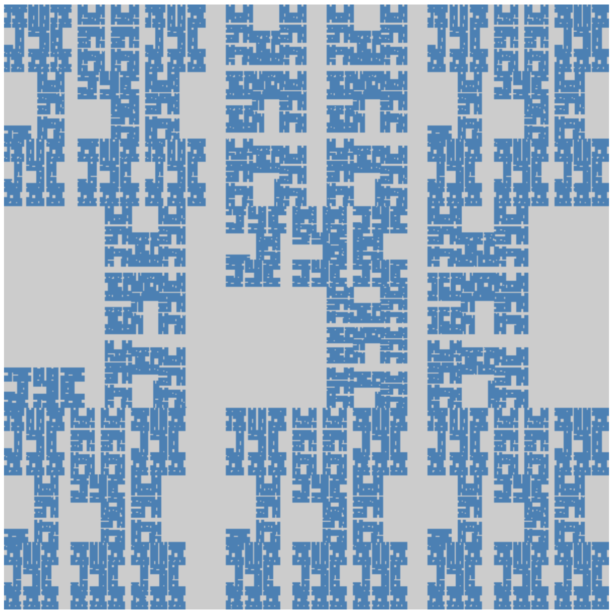

Figure 10: First iteration of initials IFS

While it definitely demonstrates a higher level of 'fractal-ness' than the other, I thought that it looked too complicated, so I instead thought about filling the negative space between letters. When I grabbed my graph paper pad to sketch out some ideas, I noticed that I had long ago labelled my pad with a sticker of my name:

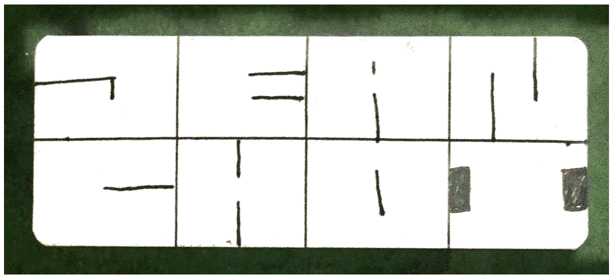

Figure 11: Name inspiration

I knew that I wanted to have a gradient background, but didn't want to only have a linear blend between two colors, so I played around with the colors and eventually came up with the warm palette. The last part of the color palette is black, to contrast with the warm background.

## 3 Lindenmayer System

### 3.1 "Drought"

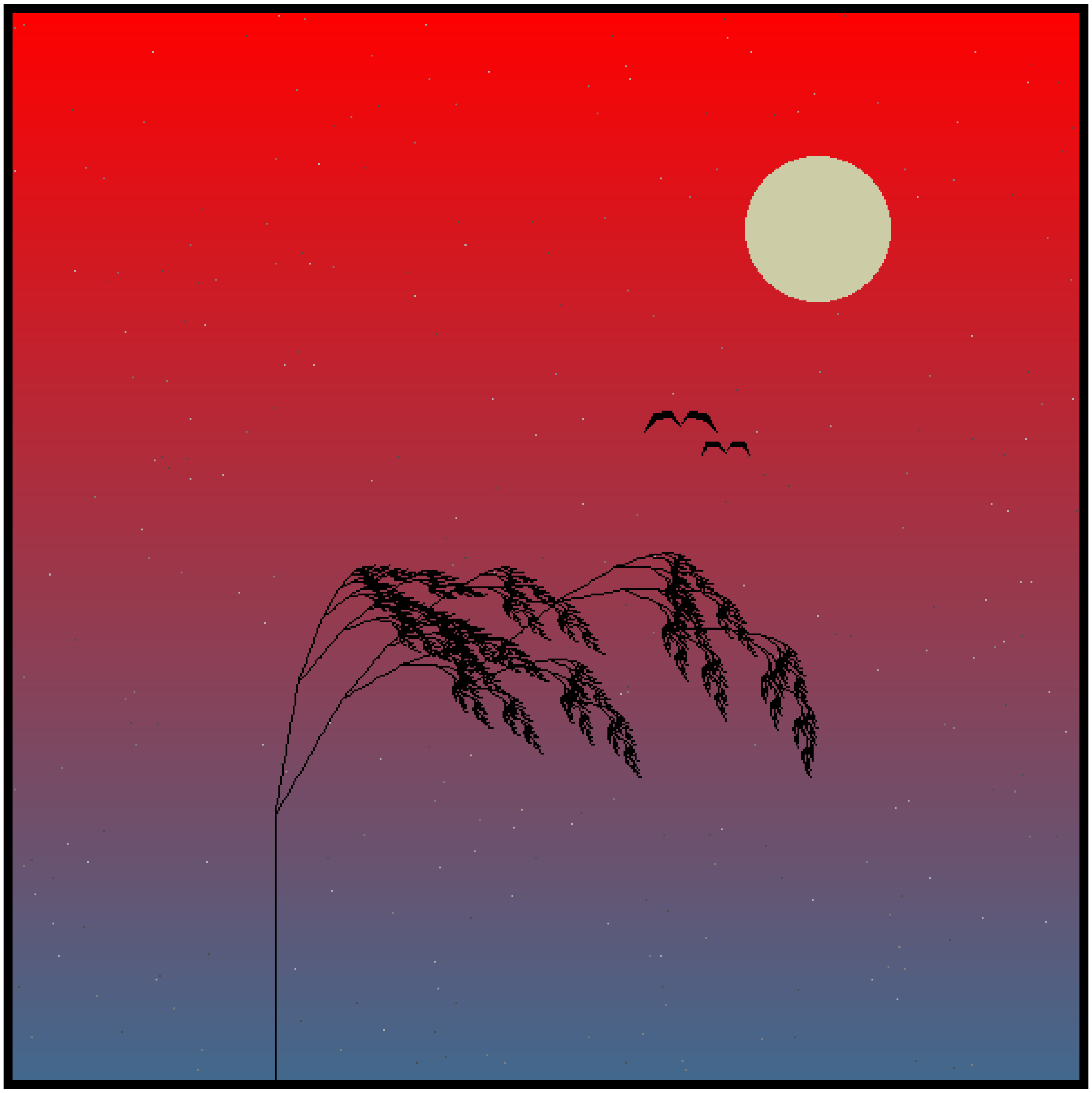

Figure 12: "Drought"

### 3.2 Fractal Description

Lindenmayer Systems: Developed in 1968 by Aristid Lindenmayer, a Hungarian theoretical biologist and botanist at the University of Utrecht.

### 3.3 Design Paradigm & Mathematical Description

Lindenmayer Systems are a type of formal grammar. They use symbols to build strings, by starting with an axiom and using production rules to build a subsequently longer string. The set of symbols I am using are the characters A and F, as well as the symbols `-`, `+`, `[` and `]`. In my program, I have defined the following axiom and production rules:

$$\text{Axiom: } \texttt{A}$$
$$\text{Production Rule 1: } \texttt{A} \rightarrow \texttt{F} - [\texttt{A}] - -\texttt{A} + +[+ + \texttt{A}]$$
$$\text{Production Rule 2: } \texttt{F} \rightarrow \texttt{FF}$$

A string is built by starting with the axiom. Every time the program reads the string, if it comes across a character where it matches with the predecessor of a production rule, it gets replaced by the successor of that rule. So, for my program, calling `buildString()` four times will result in the following strings:

Iteration 1: `A`
Iteration 2: `F-[A]--A++[++A]`
Iteration 3: `FF-[F-[A]--A++[++A]]--F-[A]--A++[++A]++[++F-[A]--A++[++A]]`
Iteration 4: `FFF-[FF-[F-[A]--A++[++A]]--F-[A]--A++[++A]++[++F-[A]--A++[++A]]]--FF-[F-[A]--A++[++A]]--F-[A]--A++[++A]++[++F-[A]--A++[++A]]++[++FF-[F-[A]--A++[++A]]--F-[A]--A++[++A]++[++F-[A]--A++[++A]]]`

After the program builds the string, it must interpret the string `stringInterpreter()` and execute specific commands to draw lines, almost like an Etch-A-Sketch. Before anything is drawn on the graphics window, the starting point and starting angle must be determined. In my implementation, my starting angle is 90°, and my starting point is (150,0). As the string is being parsed, line drawn if an alphabet character is encountered. If it's a `-` or `+`, the angle of the line being draw changes clockwise or counterclockwise by a certain degree amount. In my program, that certain amount is 10 °. Lastly, if the program parses the string and encounters a `[` or `]`, it pushes or pops the current location as well as the current angle on the stack. This allows the program to keep track of the location of branch forks.

In my implementation, I am using seven iterations to build the fractal image. I am also using the function `autoplacer()` to scale my image to 40% of the screen width. This function performs a similar function to `stringInterpreter()`, but instead of drawing the fractal, it keeps track of the minimum and maximum `x` and `y` values for a fractal that is drawn at (0,0) and with a length of 1. This function calculates a new ”bounding box” for the new fractal. Because we started with a length of 1, the scale factor between the old and new ”bounding boxes” is the scale factor itself.

### 3.4 Artistic Description

My original idea was to repeat the fractal many times to imitate a yellow wheat field, against a dusky background. I was going to linearly blend the background between two similar tones of blue, but I couldn't really tell that the linear blend was happening because I had chosen too similar shades of blue. I decided to randomly change one of the colors to a red, just to test that my function was working and decided the result was much better than my original plan.

I wanted to evoke the sense of a field in late summer, the kind of day where the heat remains until the very end of the day. The birds are flying back to their nests and the pollen is starting to settle. For the pollen, I randomly generated points in various shades of gray because I wanted it to be a subtle effect.

The birds and sun were generated statically, by drawing polygons in specific locations. I decided to make everything other than the background single colors because I didn't want anything to compete with the background. Lastly, I added a simple black border frame the piece.

# 4 Complex Numbers

## 4.1 "Multibrot Set Movie"

See attached movie.

## 4.2 Fractal Description

Mandelbrot Set: Fractals derived from complex numbers. Its definition is credited to Adrien Douady who named it in tribute to the mathematician Benoit Mandelbrot.

## 4.3 Design Paradigm & Mathematical Description

A Mandelbrot Set fractal is generated by plotting the set of complex numbers such that the equation $z_{n+1} = {z_n}^2 + c$ does not diverge for all $n$, starting at $z_0 = 0$.

In order to generate an image, the program iterates through each point of the graphics window, using a `for` loop nested inside another `for` loop. For each point on the `x,y` coordinate graph, we can translate it to its equivalent point on the complex number plane. For my program, `ssize = 600`. In the complex number plane, the "`x`" coordinate is the set of real numbers, while the "`y`" coordinate is the set of imaginary numbers.

```
cx = 2*((x-(ssize/2.0))/(ssize/2.0))
cy = 2*((y-(ssize/2.0))/(ssize/2.0))
```

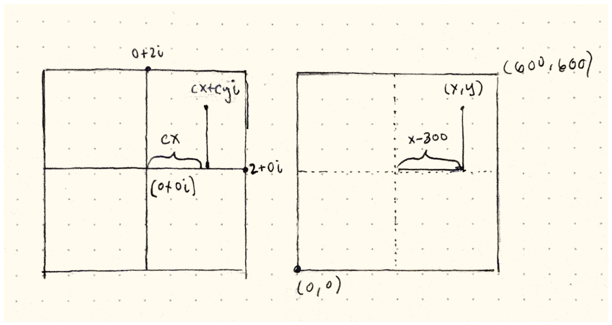

Figure 13: Comparison of point in complex number plane to `x,y` coordinate graph

After the point get translated to its equivalent point on the complex number plane, we iterate through the equation $z_{n+1} = {z_n}^2 + c$ for a particular number of repetitions, starting at $z_0 = 0$. For my program, I iterated for 200 repetitions. At any point during the repetitions, if $|z| \geq 100$, it means that $z$ has diverged to $\infty$ or $-\infty$. Finally, a point gets drawn on the graphics window, but the color depends on where the program exits out of the loop of 200 repetitions.

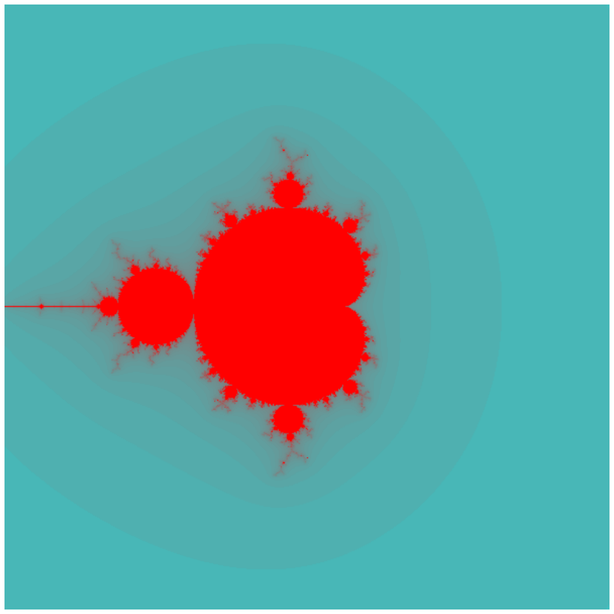

Figure 14: Mandelbrot Set for $z_{n+1} = {z_n}^2 + c$

## 4.4 Artistic Description

I was interested to see what would happen when there were different polynomial values in the equation $z_{n+1} = {z_n}^2 + c$. My program runs using a polynomial value from 1.0 to 10.0, increasing by 0.1 between each frame. I had no real expectations on what it would look like, but it was interesting to see how the fractal “grows” out of the negative real number axis.

I chose a dark teal and a bright salmon as my colors because I wanted to be able to see the linear blending contrast. It is difficult to make art using the Mandelbrot Set because it requires you to use each pixel of the graphics window. With the other fractals in this portfolio, I was able to draw lines or points on top of a background. However, if you zoom into the Mandelbrot Set, it could be useful in making patterned backgrounds.

## 5 Free Style

### 5.1 “Beach”

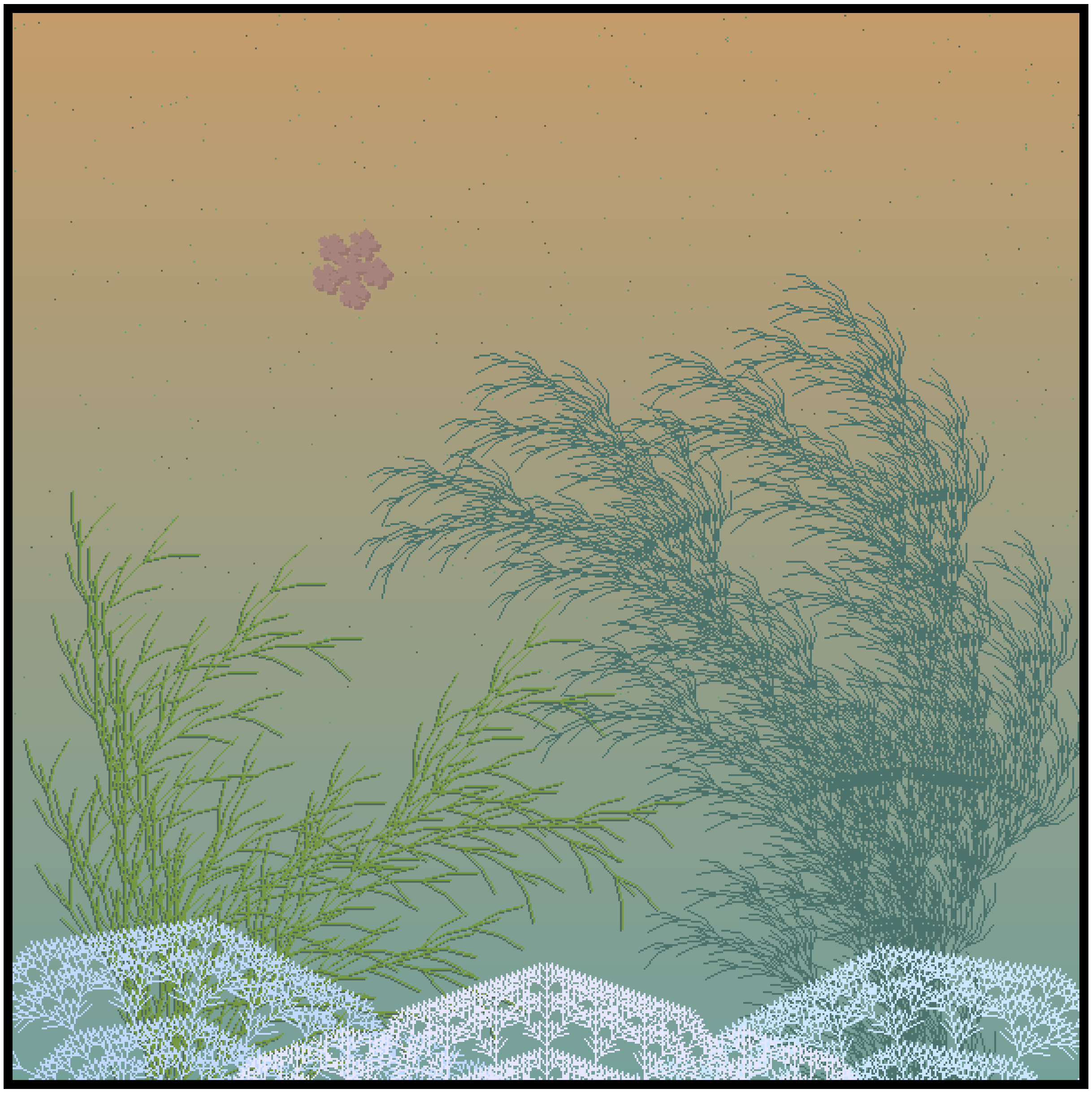

Figure 15: “Beach”

### 5.2 Fractal Description

Lindenmayer Systems: Developed in 1968 by Aristid Lindenmayer, a Hungarian theoretical biologist and botanist at the University of Utrecht.

### 5.3 Design Paradigm & Mathematical Description

The fractals in this piece are all generated by Lindenmayer Systems. Because we have already discussed how L-systems work in Section 3, I will just list the axioms and production rules of each of the fractals in this section. The seaweed variations are original creations, but the "starfish" and "waves" are existing fractals that I found here.

**Left seaweed:**

Axiom: `F`
Production Rule 1: $\texttt{F} \rightarrow \texttt{F}[--\texttt{F}[-\texttt{F}+]][+\texttt{F}[-\texttt{F}]]\texttt{F}$
Iterations: 5
$\Delta$ angle: 17°

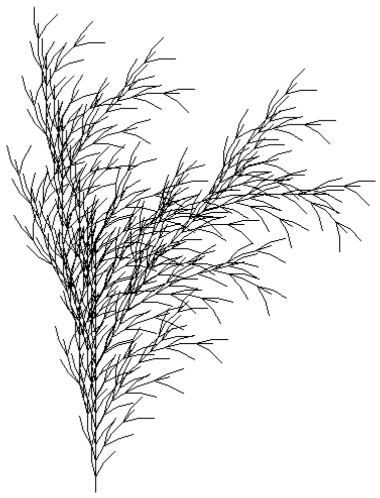

Figure 16: Isolated left seaweed

**Right seaweed:**

Axiom: `F`
Production Rule 1: $\texttt{F} \rightarrow \texttt{F}[+\texttt{F}[+\texttt{F}-]][-\texttt{F}[++\texttt{F}]]\texttt{F}$
Iterations: 4
$\Delta$ angle: 15°

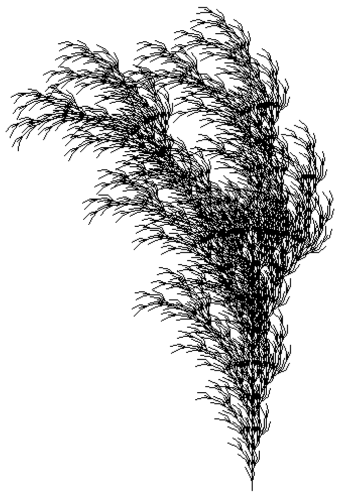

Figure 17: Isolated right seaweed

**“Starfish”:**

Axiom: `F-F-F-F-F`
Production Rule 1: $\mathtt{F} \rightarrow \mathtt{F} - \mathtt{F} - \mathtt{F} + +\mathtt{F} + \mathtt{F} - \mathtt{F}$
Iterations: 5
$\Delta$ angle: 72°

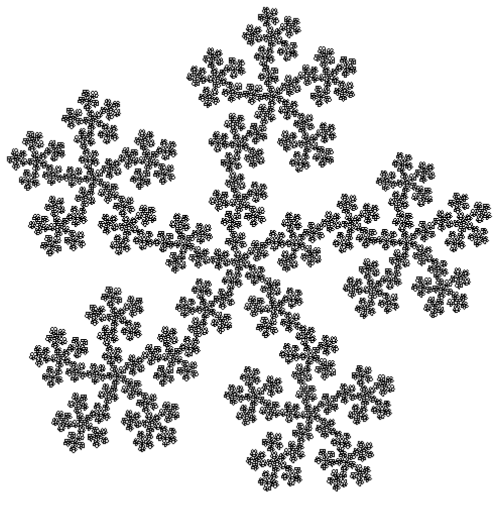

Figure 18: Isolated “starfish” (McWorter’s Pentadendrite)

**"Wave":**

Axiom: `F`
Production Rule 1: `F` $\rightarrow$ `F[+FF][−FF]F[−F][+F]F`
Iterations: 4
$\Delta$ angle: 36°

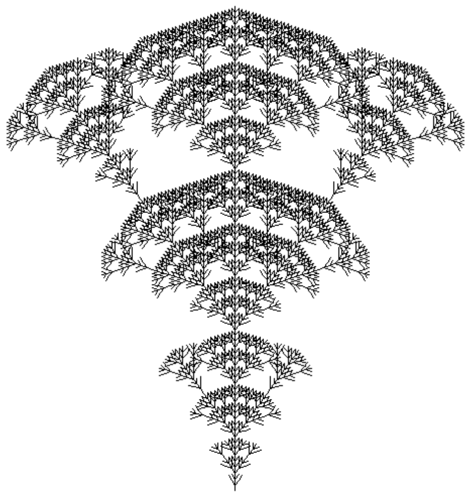

Figure 19: Isolated "wave" (common fractal tree)

### 5.4 Artistic Description

I wanted to imitate the view you would see if you were on a beach, looking down at where the ocean hits the sand. I tried to mimic different seaweed plants in shades of green. There's also a starfish that got washed up on the shore. I came across a fractal which I thought perfectly mimicked the lace-y, foamy appearance of ocean waves, so I added that to the bottom of the panel.

In order to add dimension, I added shadows to the starfish and the seaweed on the left. To do this, I simply drew the fractal in a darker color first, and then a second fractal in a lighter color, slightly askew from the shadow fractal. I did not add a shadow the the seaweed on the right because its leaves were already quite dense and adding a second layer would have clouded its defined leaves. To add more dimension to the waves, I colored each of them in various shades of light blue.

The background is a linear blend between a beige and a sea-foam green. I also added some texture to the sand by randomly drawing points in various shades of beige. Finally, I added a black border to frame the piece.

## 6 Code

### 6.1 kochWinter.c

```
/*
* Jean Choi
* 7/13/2020
* CS410P: Exploring Fractals, taught by Dr. David Ely
*
* The goal of this program is to make the Koch Snowflake
* in a winter wonderland from the indoor perspective.
*/

#include "FPToolkit.c"
double c = cos(60 * M_PI/180.0);
double s = sin(60 * M_PI/180.0);
double c1 = cos(-60 * M_PI/180.0);
double s1 = sin(-60 * M_PI/180.0);

void kochSnowflake(int depth, double *p, double *q);
void koch(int n, double px, double py, double qx, double qy);
void pytree(double x0, double y0, double x1, double y1, int n, double scale, double height);
void drawWindow();
void background();
void signature(double x, double y, double len);
void drawJ(double x, double y, double len);
void drawE(double x, double y, double len);
void drawA(double x, double y, double len);
void drawN(double x, double y, double len);
void drawC(double x, double y, double len);
void drawH(double x, double y, double len);
void drawO(double x, double y, double len);
void drawI(double x, double y, double len);

int main()
{
    // repl.it display
    G_choose_repl_display() ;

    // set dimensions
    const int swidth = 600 ;
    const int sheight = 600 ;
    G_init_graphics (swidth,sheight) ;

    // clear the screen in a given color
    G_rgb (0.3, 0.5, 0.7) ;
    G_clear () ;
    background();

    // TREES
    pytree(350, 40, 430, 40, 15, 0.66, 1);

    // SNOWFLAKE
        // user picks two points and enters depth
        double p[2], q[2];
    int depth = 3;
    for (int i = 0; i < 5; i++) {
        G_wait_click(p);
        G_wait_click(q);
        kochSnowflakeNormal(depth, p, q);
    }

    depth = 3;
    for (int i = 0; i < 5; i++) {
        G_wait_click(p);
        G_wait_click(q);
        kochSnowflake(depth, p, q);
    }

    // draw window
    drawWindow();

    // draw signature
    signature(470,10,3);

    //================================================

    int key ;
    key =  G_wait_key() ;
    // save file
    G_save_to_bmp_file("kochWinter.bmp") ;
}

void kochSnowflake(int depth, double *p, double *q){
    double o[2];

    // calculate third point of triangle, o[2]
    o[0] = (q[0]-p[0]) * c1 - (q[1]-p[1]) * s1 + p[0] ;
    o[1] = (q[0]-p[0]) * s1 + (q[1]-p[1]) * c1 + p[1] ;

    // call koch recursive function on all three lines
    koch(depth, p[0], p[1], q[0], q[1]);
    koch(depth, q[0], q[1], o[0], o[1]);
    koch(depth, o[0], o[1], p[0], p[1]);
}

// recursive function to draw a koch curve above and below two given points
void koch(int n, double px, double py, double qx, double qy) {
    G_rgb (0.8, 0.8, 0.8) ;
    double rx, ry, sx, sy, tx, ty, ux, uy;
    if (n == 0) {
        return;
    } else {
        double dx = qx-px;        double dy = qy-py;
        rx = px + 1.0/3 * dx ;
        ry = py + 1.0/3 * dy ;
        sx = px + 2.0/3 * dx ;
        sy = py + 2.0/3 * dy ;
        tx = (sx-rx) * c - (sy-ry) * s + rx ;
        ty = (sx-rx) * s + (sy-ry) * c + ry ;
        ux = (sx-rx) * c1 - (sy-ry) * s1 + rx ; // for extra spicy snowflake
        uy = (sx-rx) * s1 + (sy-ry) * c1 + ry ; // for extra spicy snowflake

        koch(n-1, px, py, rx, ry) ;
        koch(n-1, rx, ry, tx, ty) ;
        koch(n-1, rx, ry, ux, uy) ; // for extra spicy snowflake
        koch(n-1, tx, ty, sx, sy) ;
        koch(n-1, ux, uy, sx, sy) ; // for extra spicy snowflake
        koch(n-1, sx, sy, qx, qy) ;

        G_rgb(0.8,0.8,0.8) ;
        if(n==1){
            G_line(px,py,rx,ry) ;
            G_line(rx,ry,tx,ty) ;
            G_line(tx,ty,sx,sy) ;
            G_line(sx,sy,qx,qy) ;
        }
    }
}

// recursive function that draws a pythagoras tree
void pytree(double x0, double y0, double x1, double y1, int n, double scale,
double height){

    // determine second set of points (x2, y2), (x3, y3)
    double x2, y2, x3, y3, x4, y4, xm, ym, g;
    x2 = x1 - (y1 - y0);        y2 = height * y1 + (x1 - x0);
    x3 = x0 - (y1 - y0);        y3 = height * y0 + (x1 - x0);

    // determine 3rd point (x4, y4) to make third point of right triangle,
    // using (x2, y2), (x3, y3)
    g = sqrt(scale*(1-scale)) ;
    xm = x3 + scale * (x2 - x3) ;
    ym = y3 + scale * (y2 - y3) ;

    x4 = xm - g * (y2 - y3) ;
    y4 = ym + g * (x2 - x3) ;

    double boxX[4], boxY[4];
    boxX[0] = x0;     boxY[0] = y0;
    boxX[1] = x1;     boxY[1] = y1;
    boxX[2] = x2;     boxY[2] = y2;
    boxX[3] = x3;     boxY[3] = y3;
    double triangleX[3], triangleY[3];
    triangleX[0] = x3;    triangleY[0] = y3;
    triangleX[1] = x2;    triangleY[1] = y2;
    triangleX[2] = x4;    triangleY[2] = y4;

    // draw box
    G_rgb (0.163, 0.038, 0) ; //brown
    G_fill_polygon(boxX, boxY, 4);
    // draw triangle
    if(n == 1)
        G_rgb(1,1,1) ;
    G_fill_polygon(triangleX, triangleY, 3);

    if(n <= 0){
        return;
    }
    else {
        pytree(x3, y3, x4, y4, n-1, scale, height);
        pytree(x4, y4, x2, y2, n-1, scale, height);
    };
}

// draws a window that changes colors
// based on dr. ely's color-fading.c
void drawWindow(){
    double x0,y0 , x1,y1, dx, dy ;
    double sf ;

    double r,g,b , sr,sg,sb , er,eg,eb ;
    y0 = 0 ;
    y1 = 600 ;

    dy = y1-y0 ;

    sr = 0.6 ;   sg = 0.167 ;    sb = 0.167 ;
    er = 0.2 ;   eg = 0.167 ;    eb = 0.167 ;

    for(int k=y0;k<=y1;k++) {

        sf = (k-y0)/dy ;
        //sf = pow(sf,3) ;
        r = sr + sf*(er-sr) ;  g = sg + sf*(eg-sg) ;  b = sb + sf*(eb-sb) ;
        G_rgb(r,g,b) ;
        if ((0<=k && k<=2) || (598<=k && k<=600)){
            G_rgb(0,0,0) ;
            G_line(0,k,600,k);
        } else if ((38<=k && k<=40) || (280<=k && k<=282) ||
        (318<=k && k<=320) ||  (560<=k && k<=562)) {
            G_line(0,k,600,k);
            G_rgb(0,0,0);
            G_line(0,k,2,k);
            G_line(38,k,282,k);
            G_line(318,k,562,k);
            G_line(598,k,600,k);
        } else if ((2 < k && k < 38) || (282 < k && k < 318) || (562 < k && k < 598)){
            G_line(0,k,600,k);
            G_rgb(0,0,0);
            G_line(0,k,2,k);
            G_line(598,k,600,k);
        } else {
            G_line(0,k,38,k) ;
            G_line(280,k,320,k) ;
            G_line(560,k,600,k) ;
            G_rgb(0,0,0);
            G_line(0,k,2,k);
            G_line(38,k,40,k);
            G_line(280,k,282,k);
            G_line(318,k,320,k);
            G_line(560,k,562,k);
            G_line(598,k,600,k);
        }
    }
}

// draws the background
// based off of dr. ely's color-fading.c
void background(){
    double x0,y0 , x1,y1, dx, dy ;
    double sf ;

    double r,g,b , sr,sg,sb , er,eg,eb ;
    y0 = 350 ;
    y1 = 600 ;

    dy = y1-y0 ;

    sr = 0.234 ;  sg = 0.314 ;  sb = 0.510 ;
    er = 0.849 ;  eg = 0.573;  eb = 0.238 ;

    for(int k=0;k<=y0;k++){
        G_rgb(sr,sg,sb) ;
        G_line(0,k,600,k) ;
    }

    for(int k=y0;k<=y1;k++) {
        sf = (k-y0)/dy ;
        //sf = pow(sf,3) ;
        r = sr + sf*(er-sr) ;  g = sg + sf*(eg-sg) ;  b = sb + sf*(eb-sb) ;
        G_rgb(r,g,b) ;
        G_line(0,k,600,k) ;
    }
}

// draws my signature
// (x,y) is the location of the lower left point of the 'J'
// len determines the size of each letter
void signature(double x, double y, double len) {
    double dx = len*4;
    G_rgb(1,1,1);
    drawJ(x,y,len);
    drawE(x+dx,y,len);
    drawA(x+dx*2,y,len);
    drawN(x+dx*3,y,len);
    drawC(x+dx*4.5,y,len);
    drawH(x+dx*5.5,y,len);
    drawO(x+dx*6.5,y,len);
    drawI(x+dx*7.5,y,len);
```

```
}

void drawJ(double x, double y, double len) {
    double dy = len * 7;
    double dx = len * 3;
    for(double i = y; i <= dy+y; i++){
        if(y <= i && i <= y+(0.142*dy)){
          G_line(x, i, x+dx, i);
        } else if(y+(0.142*dy) < i && i <= y+(0.286*dy)) {
          G_line(x, i, x+(0.33*dx), i);
          G_line(x+(0.66*dx), i, x+dx, i);
        } else if ((y+(0.286*dy) < i && i <= y+(0.714*dy)) ||
          (y+(0.857*dy) < i && i <= y+dy)){
          G_line(x+(0.66*dx), i, x+dx, i);
        } else {}
    }
}

void drawE(double x, double y, double len) {
    double dy = len * 5;
    double dx = len * 3;
    for(double i = y; i <= dy+y; i++){
        if((y <= i && i <= y+(0.2*dy)) ||
        (y+(0.4*dy) < i && i <= y+(0.6*dy)) ||
        (y+(0.8*dy) < i && i <= y+dy)){
            G_line(x, i, x+dx, i);
        } else if(y+(0.2*dy) < i && i <= y+(0.4*dy)) {
            G_line(x, i, x+(0.33*dx), i);
        } else {
            G_line(x, i, x+(0.33*dx), i);
            G_line(x+(0.66*dx), i, x+dx, i);
        }
    }
}

void drawA(double x, double y, double len) {
    double dy = len * 5;
    double dx = len * 3;
    for(double i = y; i <= dy+y; i++){
        if((y <= i && i <= y+(0.2*dy)) ||
        (y+(0.4*dy) < i && i <= y+(0.6*dy)) ||
        (y+(0.8*dy) < i && i <= y+dy)){
            G_line(x, i, x+dx, i);
        } else if(y+(0.2*dy) < i && i <= y+(0.4*dy)) {
            G_line(x, i, x+(0.33*dx), i);
            G_line(x+(0.66*dx), i, x+dx, i);
        } else {
            G_line(x+(0.66*dx), i, x+dx, i);
        }
    }
}

void drawN(double x, double y, double len) {
    double dy = len * 5;
    double dx = len * 3;
    for(double i = y; i <= dy+y; i++){
        if((y+(0.8*dy) < i && i <= y+dy)) {
            G_line(x, i, x+dx, i);
        } else {
            G_line(x, i, x+(0.33*dx), i);
            G_line(x+(0.66*dx), i, x+dx, i);
        }
    }
}

void drawC(double x, double y, double len) {
    double dy = len * 5;
    double dx = len * 3;
    for(double i = y; i <= dy+y; i++){
        if((y <= i && i <= y+(0.2*dy)) ||
            (y+(0.8*dy) < i && i <= y+dy)){
            G_line(x, i, x+dx, i);
        } else {
            G_line(x, i, x+(0.33*dx), i);
        }
    }
}

void drawH(double x, double y, double len) {
    double dy = len * 7;
    double dx = len * 3;
    for(double i = y; i <= dy+y; i++){
        if(y <= i && i <= y+(0.571*dy)){
            G_line(x, i, x+(0.33*dx), i);
            G_line(x+(0.66*dx), i, x+dx, i);
        } else if (y+(0.571*dy) < i && i <= y+(0.714*dy)){
            G_line(x, i, x+dx, i);
        } else {
            G_line(x, i, x+(0.33*dx), i);
        }
    }
}

void drawO(double x, double y, double len) {
    double dy = len * 5;
    double dx = len * 3;
    for(double i = y; i <= dy+y; i++){
        if((y <= i && i <= y+(0.2*dy)) ||
        (y+(0.8*dy) < i && i <= y+dy)){
            G_line(x, i, x+dx, i);
        } else {
            G_line(x, i, x+(0.33*dx), i);
            G_line(x+(0.66*dx), i, x+dx, i);
        }
    }
}

void drawI(double x, double y, double len) {
    double dy = len * 7;
    double dx = len * 1;
    for(double i = y; i <= dy+y; i++){
        if((y <= i && i <= y+(0.714*dy)) ||
        (y+(0.857*dy) < i && i <= y+dy)){
            G_line(x, i, x+dx, i);
        }
    }
}
```

## 6.2 jean.c

```
/*
* Jean Choi
* 7/23/2020
* CS410P: Exploring Fractals, taught by Dr. David Ely
*
* The goal of this program is to create my name, JEAN,
* using Iterated Function Systems
*/

#include "FPToolkit.c"

double x[1] = {0} ;
double y[1] = {0} ;
int n = 1 ;


void scale (double xscale, double yscale);
void translate (double dx, double dy);
void rotate (double theta) ;
void background() ;


int main()
{
    // repl.it display
    G_choose_repl_display() ;

    // set dimensions
    const double n = 119.0; // number of blocks
    const double s = 20.0;
    const double swidth = 600 ;
    const double sheight = 600 ;
    G_init_graphics (swidth,sheight) ;

    // clear the screen in a given color
    G_rgb (0.8, 0.8, 0.8) ; // light gray
    G_clear () ;

    background();

    int j = 0;
    double r;
    double theta = M_PI/180.0;
    while (j < 100000000) {
        int r = (rand() % 120)+ 1; //random integers

        Gi_rgb(13, 68, 104);

        (1 <= r && r <= 100) ? scale(1/s, 1/s) : scale(2/s, 2/s);

        switch(r) {
            case 1 : translate(0/s, 19/s); break; //top line start
            case 2 : translate(1/s, 19/s); break;
            case 3 : translate(2/s, 19/s); break;
            case 4 : translate(3/s, 19/s); break;
            case 5 : translate(4/s, 19/s); break;
            case 6 : translate(5/s, 19/s); break;
            case 7 : translate(6/s, 19/s); break;
            case 8 : translate(7/s, 19/s); break;
            case 9 : translate(8/s, 19/s); break;
            case 10 : translate(9/s, 19/s); break;
            case 11 : translate(10/s, 19/s); break;
            case 12 : translate(11/s, 19/s); break;
            case 13 : translate(12/s, 19/s); break;
            case 14 : translate(13/s, 19/s); break;
            case 15 : translate(14/s, 19/s); break;
            case 16 : translate(15/s, 19/s); break;
            case 17 : translate(16/s, 19/s); break;
            case 18 : translate(17/s, 19/s); break;
            case 19 : translate(18/s, 19/s); break;
            case 20 : translate(19/s, 19/s); break;
            case 21 : translate(19/s, 18/s); break; // going down right side
            case 22 : translate(19/s, 17/s); break;
            case 23 : translate(19/s, 16/s); break;
            case 24 : translate(19/s, 15/s); break;
            case 25 : translate(19/s, 14/s); break;
            case 26 : translate(19/s, 13/s); break;
            case 27 : translate(19/s, 12/s); break;
            case 28 : translate(19/s, 11/s); break;
            case 29 : translate(19/s, 10/s); break;
            case 30 : translate(19/s, 9/s); break;
            case 31 : translate(19/s, 8/s); break;
            case 32 : translate(19/s, 7/s); break;
            case 33 : translate(19/s, 6/s); break;
            case 34 : translate(19/s, 5/s); break;
            case 35 : translate(19/s, 4/s); break;
            case 36 : translate(19/s, 3/s); break;
            case 37 : translate(19/s, 2/s); break;
            case 38 : translate(19/s, 1/s); break;
            case 39 : translate(19/s, 0/s); break;
            case 40 : translate(18/s, 0/s); break; // going backwards on bottom row
            case 41 : translate(17/s, 0/s); break;
            case 42 : translate(16/s, 0/s); break;
            case 43 : translate(15/s, 0/s); break;
            case 44 : translate(14/s, 0/s); break;
            case 45 : translate(13/s, 0/s); break;
            case 46 : translate(12/s, 0/s); break;
            case 47 : translate(11/s, 0/s); break;
            case 48 : translate(10/s, 0/s); break;
            case 49 : translate(9/s, 0/s); break;
            case 50 : translate(8/s, 0/s); break;
            case 51 : translate(7/s, 0/s); break;
            case 52 : translate(6/s, 0/s); break;
            case 53 : translate(5/s, 0/s); break;
            case 54 : translate(4/s, 0/s); break;
            case 55 : translate(3/s, 0/s); break;
```

```
            case 56 : translate(2/s, 0/s); break;
            case 57 : translate(1/s, 0/s); break;
            case 58 : translate(0/s, 0/s); break;
            case 59 : translate(0/s, 1/s); break;  // going back up on left side
            case 60 : translate(0/s, 2/s); break;
            case 61 : translate(0/s, 3/s); break;
            case 62 : translate(0/s, 4/s); break;
            case 63 : translate(0/s, 5/s); break;
            case 64 : translate(0/s, 6/s); break;
            case 65 : translate(0/s, 7/s); break;
            case 66 : translate(0/s, 8/s); break;
            case 67 : translate(0/s, 9/s); break;
            case 68 : translate(0/s, 10/s); break;
            case 69 : translate(0/s, 11/s); break;
            case 70 : translate(0/s, 12/s); break;
            case 71 : translate(0/s, 13/s); break;
            case 72 : translate(0/s, 14/s); break;
            case 73 : translate(0/s, 15/s); break;
            case 74 : translate(0/s, 16/s); break;
            case 75 : translate(0/s, 17/s); break;
            case 76 : translate(0/s, 18/s); break;
            case 77 : translate(1/s, 15/s); break; // J negative space
            case 78 : translate(2/s, 15/s); break;
            case 79 : translate(3/s, 15/s); break;
            case 80 : translate(4/s, 15/s); break;
            case 81 : translate(5/s, 15/s); break;
            case 82 : translate(5/s, 14/s); break;
            case 83 : translate(14/s, 16/s); break; // E negative space top
            case 84 : translate(15/s, 16/s); break;
            case 85 : translate(16/s, 16/s); break;
            case 86 : translate(17/s, 16/s); break;
            case 87 : translate(18/s, 16/s); break;
            case 88 : translate(14/s, 13/s); break; // E negative space bottom
            case 89 : translate(15/s, 13/s); break;
            case 90 : translate(16/s, 13/s); break;
            case 91 : translate(17/s, 13/s); break;
            case 92 : translate(18/s, 13/s); break;
            case 93 : translate(16/s, 8/s); break; // N negative space top
            case 94 : translate(16/s, 7/s); break;
            case 95 : translate(16/s, 6/s); break;
            case 96 : translate(16/s, 5/s); break;
            case 97 : translate(13/s, 4/s); break; // N negative space bottom
            case 98 : translate(13/s, 3/s); break;
            case 99 : translate(13/s, 2/s); break;
            case 100 : translate(13/s, 1/s); break;
            case 101 : translate(4/s, 5/s); break; // A negative space
            case 102 : translate(4/s, 1/s); break;
            case 103 : translate(9/s, 17/s); break; // vert space between letters
            case 104 : translate(9/s, 15/s); break;
            case 105 : translate(9/s, 13/s); break;
            case 106 : translate(9/s, 11/s); break;
            case 107 : translate(9/s, 9/s); break;
            case 108 : translate(9/s, 7/s); break;
            case 109 : translate(9/s, 5/s); break;
            case 110 : translate(9/s, 3/s); break;
            case 111 : translate(9/s, 1/s); break;
            case 112 : translate(1/s, 9/s); break; // horizontal space between letters
            case 113 : translate(3/s, 9/s); break;
            case 114 : translate(5/s, 9/s); break;
            case 115 : translate(7/s, 9/s); break;
            case 116 : translate(11/s, 9/s); break;
            case 117 : translate(13/s, 9/s); break;
            case 118 : translate(15/s, 9/s); break;
            default : translate(17/s, 9/s);
        }

        G_point(swidth*x[0], sheight*y[0]);
        j++;
    }


        //================================================

        int key ;
        key =  G_wait_key() ;

        // save file
        G_save_to_bmp_file("jean.bmp") ;
}

void background() {
    double x0,y0 , x1,y1, dx, dy ;
    double sf, sf2
    double r,g,b;
    double rr,gg,bb;
    double r1,g1,b1 ;
    double r2,g2,b2 ;
    double r3,g3,b3 ;
    double r4,g4,b4 ;
    y0 = 0 ;
    y1 = 600 ;

    dy = y1-y0
    r1 = 1.0 ;  g1 = 0.0 ;  b1 = 0.0 ;
    r2 = 1.0 ;  g2 = 1.0 ;  b2 = 0.0 ;
    r3 = 0.0 ;  g3 = 1.0 ;  b3 = 0.0 ;
    r4 = 0.0 ;  g4 = 1.0 ;  b4 = 1.0 ;

    for(int k=y0;k<=y1;k++) {
        sf = (k-y0)/dy ;
        r = r1 + sf*(r2-r1) ;
        g = g1 + sf*(g2-g1)  ;
        b = b1 + sf*(b2-b1) ;
        for(int j=y0; j<=y1; j++){
            sf2 = (j-y0)/dy;
            rr = r + sf2*(r4-r3);
            gg = g + sf2*(g4-g3);
            bb = b + sf2*(b4-b3);
            G_rgb(rr, gg, bb) ;
            G_point(j,k) ;
        }
    }

}

void translate (double dx, double dy) {
    int i = 0 ;
    while (i < n) {
        x[i] = x[i] + dx ;
        y[i] = y[i] + dy ;
        i++ ;
    }
}

void scale (double xscale, double yscale) {
        for (int i = 0; i < n; i++) {
                x[i] = xscale * x[i] ;
                y[i] = yscale * y[i] ;
        }
}

void rotate (double theta) {
        double temp ;
        double radians = theta*M_PI/180.0 ;
        for (int i = 0; i < n; i++) {
                temp = x[i]*cos(radians) - y[i]*sin(radians) ;
                y[i] = x[i]*sin(radians) + y[i]*cos(radians) ;
                x[i] = temp ;
        }
}
```

## 6.3 wheat.c

```
/*
* Jean Choi
* 7/16/2020
* CS410P: Exploring Fractals, taught by Dr. David Ely
*/

#include "FPToolkit.c"

double swidth = 600;
double sheight = 600;

// for string builder
char v[1000000];
char u[1000000];
char temp[1000000];
typedef struct Production {
    char var ;
    char axiom[100] ;
    char rule[100] ;
} Production ;
const int rule_num = 0;

// for string interpreter
double cangle = 90;
double angle = 10;
double sx = 50;
double sy = 50;
double length = 5;

// for autoplacer
double minx, miny, maxx, maxy ;
double apx, apy, aplength;

// for stack
double x[1000000], y[1000000], z[1000000];
int top = -1;

int isempty() {
  return (top<=-1) ? 1: 0;
}

int isfull() {
  return (top>=1000000) ? 1 : 0;
}

void stackinfo(){
printf("top %d: %lf %lf %lf\n", top, x[top], y[top], z[top]);
}

double popx() {
  return !isempty() ? x[top] : printf("Stack X is empty.\n");
}
```

```
double popy() {
  return !isempty() ? y[top] : printf("Stack Y is empty.\n");
}

double popz() {
  return !isempty() ? z[top] : printf("Stack Z is empty.\n");
}

void push(double xd, double yd, double zd) {
    if(!isfull()) {
        top++;
        x[top] = xd;
        y[top] = yd;
        z[top] = zd;
    } else {
        printf("Could not insert data, Stack is full.\n");
    }
}


// builds a string based on a given axiom and rules
void buildString(int depth){
    struct Production prd[rule_num];
    u[0] = '\0';

    // AXIOM
    int rule_num = 0 ;
    strcpy(prd[rule_num].axiom,"A") ;
    rule_num++ ;
    // RULE 1
    prd[rule_num].var = 'A' ;
    strcpy(prd[rule_num].rule,"F-[A]--A++[++A]") ;
    rule_num++ ;
    // RULE 2
    prd[rule_num].var = 'F' ;
    strcpy(prd[rule_num].rule, "FF") ;
    rule_num++ ;

    // rule replace n times
    strcpy (u, prd[0].axiom);
    for(int j = 0; j < depth; j++) {
        strcpy(v,u);
        memset(&u[0], 0, sizeof(u));
        for(int i = 0; i < strlen(v); i++) {
            if(v[i] == prd[1].var) {
                strcat(u, prd[1].rule);
            } else if( v[i] == prd[2].var) {
                strcat(u, prd[2].rule);
            } else {
                temp[0] = v[i];
                temp[1] = '\0' ;
                strcat(u, temp);
            }
        }
    }
}

// resize and center image
void autoplacer(double sf) {
    //by the end of this function, sx, sy, length should change
    //set minx, miny, maxx, maxy to initial point
    minx = 0 ; maxx = 0 ; miny = 0 ; maxy = 0 ;
    apx = 0; apy = 0; aplength = 1;
    double nx, ny;

    for(int i = 0; i < strlen(u); i++) {
        if(u[i] == '+') {
            cangle += angle;
        } else if (u[i] == '-') {
            cangle -= angle;
        } else if (u[i] == 'F' || u[i] == 'A') {
            nx = aplength * cos(cangle*M_PI/180.0) + apx;
            ny = aplength * sin(cangle*M_PI/180.0) + apy;
            if(nx < minx)
                minx = nx;
            if(nx > maxx)
                maxx = nx;
            if(ny < miny)
                miny = ny;
            if(ny > maxy)
                maxy = ny;
            apx = nx; apy = ny;
        } else if (u[i] == '[') { // push on stack
            push(apx,apy,cangle);
        } else if (u[i] == ']') { // pop off stack
            apx = popx();
            apy = popy();
            cangle = popz();
            top--;
        } else {}
    }

    //calculate new bounding box
    double dx = maxx-minx;
    double dy = maxy-miny;
    double dnx, dny;
    if (dx > dy) {
        dnx = sf * swidth;
        dny = dnx * dy / dx;
    } else {
        dny = sf * sheight;
        dnx = dny * dx / dy;
    }
    // printf("dnx %lf\n", dnx);
    // printf("dny %lf\n", dny);

    // calculate new length
    double xmultiplier = dnx/dx;
    length = xmultiplier;

    // find delta minx to 0, and delta miny to 0
    double dminx = 0-minx;
    double dminy = 0-miny;

    // calculate new sx, sy
    sx = (swidth-dnx)/2 + dminx * xmultiplier;
    sy = (sheight-dny)/2 + dminy * xmultiplier;
}

// decodes the string into a graphic
void stringInterpreter(double sx, double sy, double sf){
    double nx, ny;

    autoplacer(sf);

    double x = sx; double y = sy;
    cangle = 90;

    for(int i = 0; i < strlen(u); i++) {
        if(u[i] == '+') {
            cangle += angle;
        }
        if (u[i] == '-') {
            cangle -= angle;
        }
        if (u[i] == 'F' || u[i] == 'A') {
            nx = length * cos(cangle*M_PI/180.0) + x;
            ny = length * sin(cangle*M_PI/180.0) + y ;
            G_rgb(0,0,0) ;
            G_line(x, y, nx, ny);
            x = nx; y = ny;
        }
        if (u[i] == '[') { // push on stack
            push(x,y,cangle);
        }
        if (u[i] == ']') { // pop off stack
            x = popx();
            y = popy();
            cangle = popz();
            top--;
        }
    }
}

void background(){
    double x0,y0 , x1,y1, dx, dy ;
    double sf ;

    double r,g,b;
    double r1,g1,b1 ;
    double r2,g2,b2 ;
    y0 = 0 ;
    y1 = 600 ;

    dy = y1-y0 ;

    r1 = 0.254 ;  g1 = 0.412 ;  b1 = 0.557 ;
    // r2 = 0.286 ;  g2 = 0.490 ;  b2 = 0.667 ;
    r2 = 1.0; g2 = 0.0 ; b2 = 0.0;

    for(int k=y0;k<=y1;k++) {
        sf = (k-y0)/dy ;
        r = r1 + sf*(r2-r1) ;  g = g1 + sf*(b2-g1) ;  b = b1 + sf*(b2-b1) ;
        G_rgb(r,g,b) ;
        G_line(0,k,600,k) ;
    }
}

void bird1(int sx, int sy){
    G_rgb (0.0, 0.0, 0.0) ; // black
    double bx[6], by[6];
    bx[0] = sx ;        by[0] = sy ;
    bx[1] = bx[0]-5 ;   by[1] = by[0]+5 ;
    bx[2] = bx[1]-6 ;   by[2] = by[1]-1 ;
    bx[3] = bx[2]-4 ;   by[3] = by[2]-6 ;
    bx[4] = bx[3]+3 ;   by[4] = by[3]+9 ;
    bx[5] = bx[4]+7 ;   by[5] = by[4]+1 ;
    G_fill_polygon (bx,by,6) ;
    bx[0] = sx-2 ;        by[0] = sy ;
    bx[1] = bx[0]+5 ;   by[1] = by[0]+5 ;
    bx[2] = bx[1]+6 ;   by[2] = by[1]-1 ;
    bx[3] = bx[2]+4 ;   by[3] = by[2]-6 ;
    bx[4] = bx[3]-3 ;   by[4] = by[3]+9 ;
    bx[5] = bx[4]-7 ;   by[5] = by[4]+1 ;
    G_fill_polygon (bx,by,6) ;
}

void bird2(int sx, int sy){
    G_rgb (0.0, 0.0, 0.0) ; // black
    double bx[6], by[6];
    bx[0] = sx ;        by[0] = sy ;
    bx[1] = bx[0]-6 ;   by[1] = by[0]+6 ;
    bx[2] = bx[1]-8 ;   by[2] = by[1]-2 ;
    bx[3] = bx[2]-8 ;   by[3] = by[2]-7 ;
    bx[4] = bx[3]+6 ;   by[4] = by[3]+11 ;
    bx[5] = bx[4]+10 ;   by[5] = by[4]+2 ;
    G_fill_polygon (bx,by,6) ;
    bx[0] = sx-2 ;        by[0] = sy ;
    bx[1] = bx[0]+6 ;   by[1] = by[0]+6 ;
    bx[2] = bx[1]+8 ;   by[2] = by[1]-2 ;
    bx[3] = bx[2]+8 ;   by[3] = by[2]-7 ;
    bx[4] = bx[3]-6 ;   by[4] = by[3]+11 ;
    bx[5] = bx[4]-10 ;   by[5] = by[4]+2 ;
    G_fill_polygon (bx,by,6) ;
}
```

```
void border(){
    G_rgb(0,0,0);
    for(int i = 0; i <= 4; i++){
        G_line(0,i,600,i);
        G_line(i,0,i,600);
    }
    for(int i = 595; i <= 600; i++){
        G_line(0,i,600,i);
        G_line(i,0,i,600);
    }
}

int main()
{
    G_choose_repl_display() ;
    G_init_graphics(swidth, sheight) ;

    G_rgb(1,1,1) ;
    G_clear() ;

    //background color
    background();

    //haze
    for(int i = 0; i <= 300; i++) {
        if (i%3 == 0){
            G_rgb(0.7,0.7,0.7);
        } else if (i%3 == 1) {
            G_rgb(0.4,0.5,0.5);
        } else {
            G_rgb(0.3,0.3,0.3) ;
        }
        int rx = rand() % 599;
        int ry = rand() % 599;
        G_point(rx,ry);
    }

    //sun
    G_rgb(0.8,0.8,0.65);
    G_fill_circle(450,475,40);

    //birds
    bird1(400, 350);
    bird2(375, 365);

    //plant
    cangle = 90;
    buildString(7);
    stringInterpreter(150,0,0.5);

    //border
    border();
    //===============================================

    // int key ;
    // key =  G_wait_key() ;


    // save file
    G_save_to_bmp_file("wheathaze.bmp") ;
}
```

### 6.4 mandelbrot_movie.c

```
/*
* Jean Choi
* 8/6/2020
* CS410P: Exploring Fractals, taught by Dr. David Ely
*/

#include "FPToolkit.c"
#include <stdio.h>
#include <math.h>
#include <complex.h>

const int ssize = 600;
char fname[200];

int reps = 200;
double cx,cy ;
int xp, yp;
complex c, z;

double sr, sg, sb, er, eg, eb ;
double r, g, b ;

void mandelbrot(double power) {
    sr = 0.0 ;  sg = 0.3 ;  sb = 0.3 ;
    er = 1.0 ;  eg = 0.3 ;  eb = 0.3 ;

    // iterate through each pixel of window
    for (int x = 0; x < ssize; x++){
        for (int y = 0; y < ssize; y++) {

            // map to coordinating complex number
            cx = 2*((x-(ssize/2.0))/(ssize/2.0)) ;
            cy = 2*((y-(ssize/2.0))/(ssize/2.0)) ;
            c = cx + cy*I ;
            z = 0;
            int k ;
            for (k = 0 ; k < reps ; k++) {
                z = cpow(z, power) + c ;
                if (cabs(z) > 100) { // diverged to inf or -inf
                    break ;
                }
            }
            double sf = 1.0*k/reps;
            sf = pow(sf,0.3) ;
            r = sr + sf*(er-sr) ;
            g = sg + sf*(eg-sg) ;
            b = sb + sf*(eb-sb) ;
            G_rgb(r,g,b) ;
            G_point(x,y);
        }
    }
}
```

```
int main()
{
    // repl.it display
        G_choose_repl_display() ;

        // set dimensions
        G_init_graphics (ssize,ssize) ;
    G_rgb(1,1,1);
    G_clear();

    int count = 0;
    for(double i = 1.0; i < 10.0; i=i+0.1){
        mandelbrot(i);
        sprintf(fname, "mandelbrot_movie%04d.bmp", count) ;
        G_save_to_bmp_file(fname) ;
        count++ ;
    }
}
```

### 6.5 beach.c

```
/*
* Jean Choi
* 8/11/2020
* CS410P: Exploring Fractals, taught by Dr. David Ely
*/

#include "FPToolkit.c"

double swidth = 600;
double sheight = 600;

// for string builder
char v[1000000];
char u[1000000];
char temp[1000000];
typedef struct Production {
    char var ;
    char axiom[100] ;
    char rule[100] ;
} Production ;
const int rule_num = 0;

// for string interpreter
double cangle = 0;
double angle = 0;
double sx = 50;
double sy = 50;
double length = 5;

// for autoplacer
double minx, miny, maxx, maxy ;
double apx, apy, aplength;

// for stack
double x[1000000], y[1000000], z[1000000];
int top = -1;

int isempty() {
    return (top<=-1) ? 1: 0;
}

int isfull() {
    return (top>=1000000) ? 1 : 0;
}

void stackinfo(){
    printf("top %d: %lf %lf %lf\n", top, x[top], y[top], z[top]);
}

double popx() {
    return !isempty() ? x[top] : printf("Stack X is empty.\n");
}
```

```
double popy() {
    return !isempty() ? y[top] : printf("Stack Y is empty.\n");
}

double popz() {
    return !isempty() ? z[top] : printf("Stack Z is empty.\n");
}

void push(double xd, double yd, double zd) {
   if(!isfull()) {
        top++;
        x[top] = xd;
        y[top] = yd;
        z[top] = zd;
    } else {
        printf("Could not insert data, Stack is full.\n");
    }
}

void buildSeaweed2(int depth){
    struct Production prd[rule_num];
    u[0] = '\0';

    // AXIOM
    int rule_num = 0 ;
    strcpy(prd[rule_num].axiom,"F") ;
    rule_num++ ;
    // RULE 1
    prd[rule_num].var = 'F' ;
    strcpy(prd[rule_num].rule,"F[+F[+F-]][-F[++F]]F") ;
    rule_num++ ;

    // rule replace n times
    strcpy (u, prd[0].axiom);
    for(int j = 0; j < depth; j++) {
        strcpy(v,u);
        memset(&u[0], 0, sizeof(u));
        for(int i = 0; i < strlen(v); i++) {
            if(v[i] == prd[1].var) {
                strcat(u, prd[1].rule);
            } else {
                temp[0] = v[i];
                temp[1] = '\0' ;
                strcat(u, temp);
            }
        }
    }
}

void buildSeaweed(int depth){
    struct Production prd[rule_num];
    u[0] = '\0';
    // AXIOM
    int rule_num = 0 ;
    strcpy(prd[rule_num].axiom,"F") ;
    rule_num++ ;
    // RULE 1
    prd[rule_num].var = 'F' ;
    strcpy(prd[rule_num].rule,"F[--F[-F+]][+F[-F]]F") ;
    rule_num++ ;

    // rule replace n times
    strcpy (u, prd[0].axiom);
    for(int j = 0; j < depth; j++) {
        strcpy(v,u);
        memset(&u[0], 0, sizeof(u));
        for(int i = 0; i < strlen(v); i++) {
            if(v[i] == prd[1].var) {
                strcat(u, prd[1].rule);
            } else {
                temp[0] = v[i];
                temp[1] = '\0' ;
                strcat(u, temp);
            }
        }
    }
}

void buildStarfish(int depth){
    struct Production prd[rule_num];
    u[0] = '\0';

    // AXIOM
    int rule_num = 0 ;
    strcpy(prd[rule_num].axiom,"F-F-F-F-F") ;
        rule_num++ ;
    // RULE 1
    prd[rule_num].var = 'F' ;
    strcpy(prd[rule_num].rule,"F-F-F++F+F-F") ;
    rule_num++ ;

    // rule replace n times
    strcpy (u, prd[0].axiom);
    for(int j = 0; j < depth; j++) {
        strcpy(v,u);
        memset(&u[0], 0, sizeof(u));
        for(int i = 0; i < strlen(v); i++) {
            if(v[i] == prd[1].var) {
                strcat(u, prd[1].rule);
            } else {
                temp[0] = v[i];
                temp[1] = '\0' ;
                strcat(u, temp);
            }
        }
    }
```

```
}

void buildTree(int depth){
    struct Production prd[rule_num];
    u[0] = '\0';

    // AXIOM
    int rule_num = 0 ;
    strcpy(prd[rule_num].axiom,"F") ;
    rule_num++ ;
    // RULE 1
    prd[rule_num].var = 'F' ;
    strcpy(prd[rule_num].rule,"F[+FF][-FF]F[-F][+F]F") ;
    rule_num++ ;

    // rule replace n times
    strcpy (u, prd[0].axiom);
    for(int j = 0; j < depth; j++) {
        strcpy(v,u);
        memset(&u[0], 0, sizeof(u));
        for(int i = 0; i < strlen(v); i++) {
            if(v[i] == prd[1].var) {
                strcat(u, prd[1].rule);
            } else {
                temp[0] = v[i];
                temp[1] = '\0' ;
                strcat(u, temp);
            }
        }
    }
}

// resize and center image
void autoplacer(double deg, double sf) {
    //by the end of this function, sx, sy, length should change
    //set minx, miny, maxx, maxy to initial point
    minx = 0 ; maxx = 0 ; miny = 0 ; maxy = 0 ;
    apx = 0; apy = 0; aplength = 1;
    double nx, ny;

    cangle = deg;

    for(int i = 0; i < strlen(u); i++) {
        if(u[i] == '+') {
            cangle += angle;
        } else if (u[i] == '-') {
        cangle -= angle;
        } else if (u[i] == 'F' || u[i] == 'A' || u[i] == 'B') {
        nx = aplength * cos(cangle*M_PI/180.0) + apx;
        ny = aplength * sin(cangle*M_PI/180.0) + apy;
            if(nx < minx)
                minx = nx;
            if(nx > maxx)
                maxx = nx;
            if(ny < miny)
                miny = ny;
            if(ny > maxy)
                maxy = ny;
            apx = nx; apy = ny;
        } else if (u[i] == '[') { // push on stack
            push(apx,apy,cangle);
        } else if (u[i] == ']') { // pop off stack
            apx = popx();
            apy = popy();
            cangle = popz();
            top--;
        } else {}
    }

    //calculate new bounding box
    double dx = maxx-minx;
    double dy = maxy-miny;
    double dnx, dny;
    if (dx > dy) {
        dnx = sf * swidth;
        dny = dnx * dy / dx;
    } else {
        dny = sf * sheight;
        dnx = dny * dx / dy;
    }

    // calculate new length
    double xmultiplier = dnx/dx;
    length = xmultiplier;

    // find delta minx to 0, and delta miny to 0
    double dminx = 0-minx;
    double dminy = 0-miny;

    // calculate new sx, sy
    sx = (swidth-dnx)/2 + dminx * xmultiplier;
    sy = (sheight-dny)/2 + dminy * xmultiplier;
}


void border(){
    G_rgb(0,0,0);
    for(int i = 0; i <= 4; i++){
        G_line(0,i,600,i);
        G_line(i,0,i,600);
    }
    for(int i = 595; i <= 600; i++){
        G_line(0,i,600,i);
        G_line(i,0,i,600);
    }
}

// decodes the string into a graphic
```

```
void stringInterpreter(double sx, double sy, double deg, double sf){
    double nx, ny;

    autoplacer(deg, sf);

    double x = sx; double y = sy;

    // G_rgb(1,0,0);
    // G_fill_circle(x, y, 2);
    cangle = deg;

    for(int i = 0; i < strlen(u); i++) {
        if(u[i] == '+') {
            cangle += angle;
        }
        if (u[i] == '-') {
            cangle -= angle;
        }
        if (u[i] == 'F' || u[i] == 'A' || u[i] == 'B') {
            nx = length * cos(cangle*M_PI/180.0) + x;
            ny = length * sin(cangle*M_PI/180.0) + y ;
            G_line(x, y, nx, ny);
            x = nx; y = ny;
        }
        if (u[i] == '[') { // push on stack
            push(x,y,cangle);
        }
        if (u[i] == ']') { // pop off stack
            x = popx();
            y = popy();
            cangle = popz();
            top--;
        }
    }
}

void background(){
    double x0,y0 , x1,y1, dx, dy ;
    double sf ;
    double r,g,b , sr,sg,sb , er,eg,eb ;
    y0 = 0 ;
    y1 = 600 ;

    dy = y1-y0 ;
    sr = 0.456 ;  sg = 0.627 ;  sb = 0.606 ;
    er = 0.770 ;  eg = 0.611 ;  eb = 0.418 ;
    for(int k=0;k<=y0;k++){
        G_rgb(sr,sg,sb) ;
        G_line(0,k,600,k) ;
    }

    for(int k=y0;k<=y1;k++) {
        sf = (k-y0)/dy ;
        //sf = pow(sf,3) ;
        r = sr + sf*(er-sr) ;  g = sg + sf*(eg-sg) ;  b = sb + sf*(eb-sb) ;
        G_rgb(r,g,b) ;
        G_line(0,k,600,k) ;
    }
}

int main()
{
    G_choose_repl_display() ;
    G_init_graphics(swidth, sheight) ;

    G_rgb(1,1,1) ;
    G_clear() ;
    background();

    // sand texture
    for(int i = 0; i <= 800; i++) {
        if (i%3 == 0){
            G_rgb(0.4,0.640,0.473);
        } else if (i%3 == 1) {
            G_rgb(0.415,0.555,0.410);
        } else {
            G_rgb(0.369,0.389,0.3) ;
        }
        int rx = rand() % 599;
        int ry = rand() % 599;
        if(ry >= 200){
            G_point(rx,ry);
        }
    }

    //seaweed right
    angle = 17;
    G_rgb(0.300, 0.450, 0.420) ;
    buildSeaweed2(5);
    stringInterpreter(500, -150, 90, 1);

    //seaweed left shadow
    angle = 15;
    G_rgb(0.314, 0.417, 0.360) ;
    buildSeaweed(4);
    stringInterpreter(99, -151, 90, .8);

    //seaweed left
    angle = 15;
    G_rgb(0.456, 0.598, 0.259) ;
    buildSeaweed(4);
    stringInterpreter(100, -150, 90, .8);

    //starfish shadow
    angle = 72;
    G_rgb(0.60, 0.464, 0.435) ;
    buildStarfish(5);
    stringInterpreter(202, 473, 0, .07);
    //starfish
    angle = 72;
    G_rgb(0.65, 0.514, 0.485) ;
    buildStarfish(5);
    stringInterpreter(200, 475, 0, .07);

    //wave pt1
    angle = 36 ;
    G_rgb(0.757,0.845,1.0);
    buildTree(4);
    stringInterpreter(30, -385, 80, .8);

    //wave pt2
    angle = 36 ;
    G_rgb(0.8,0.9,1.0);
    buildTree(4);
    stringInterpreter(570, -400, 100, .8);

     //wave pt3
    angle = 36 ;
    G_rgb(.9,.9,1);
    buildTree(4);
    stringInterpreter(300, -350, 90, .7);

    //border
    border();
    // //===============================================

    int key ;
    key =  G_wait_key() ;

    // save file
        G_save_to_bmp_file("beach.bmp") ;
}
```