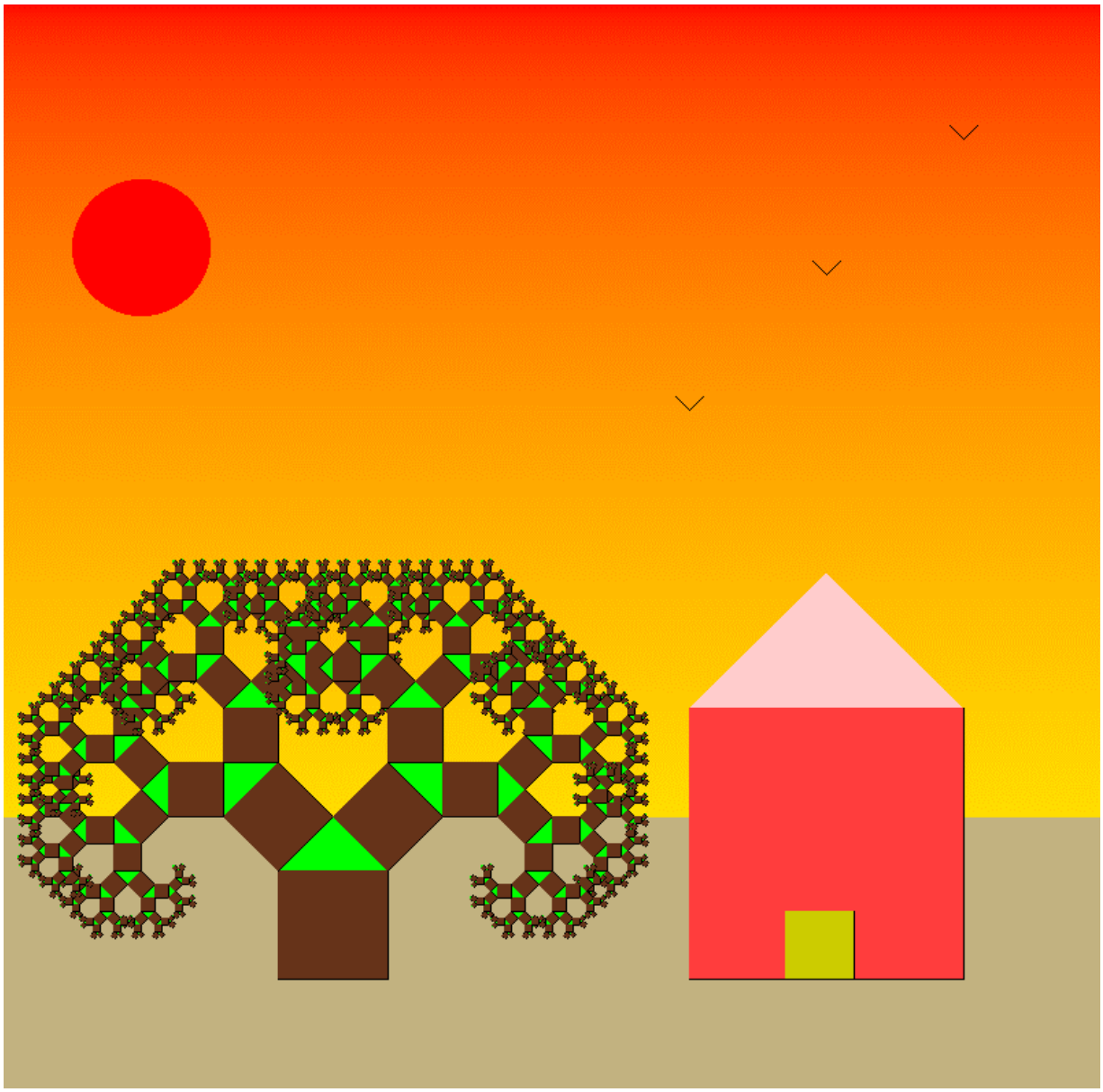
EXPLORING FRACTALS

Code by: Mounika Javvaji

Anoop Reddy Yeddula

1) THE PYTHAGORAS TREE FRACTAL



Fractal Description: The Pythagoras tree is a plane fractal constructed from squares. Invented by the Dutch mathematics teacher Albert E. Bosman in 1942

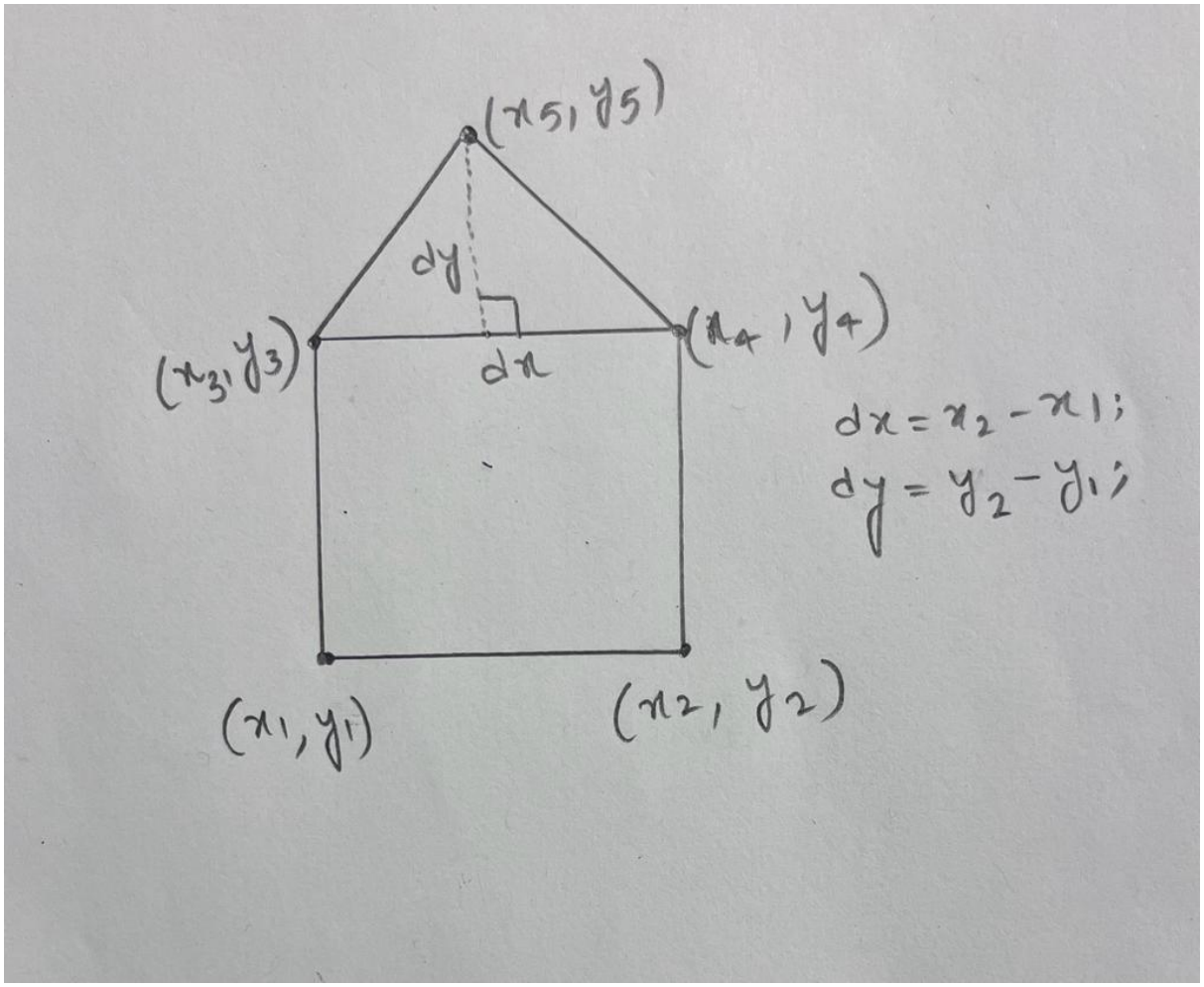
Citation: https://en.wikipedia.org/wiki/Pythagoras_tree_(fractal)

DESIGN PARADIGM AND MATHEMATICAL DESCRIPTION:

we have created the Pythagoras tree recursively. We know that this tree is constructed from squares. To draw the square first we need to give the two points (x1,y1) and (x2,y2)and find the other two points (x3,y3) and (x4,y4) to create a square.

To find (x3,y3) and (x4,y4):

As one of the square's vertices, we have designated the supplied point. (x1, y1) and (x2, y2), which are perpendicular to (x3, y3) and (x4, y4), both generate the new points (x3, y3) and (x4, y4), which together form a right-angle triangle and a square. Point (x5, y5) is the vertex of this triangle's right angle. Recursively calling two children, the functions drawtree (x3, y3, x5, y5, depth) and drawtree (x4, y4, x5, y5, depth) are used to represent the additional children. The depth parameter of the drawtree function regulates the number of recursive levels.



For (x3,y3) and (x4,y4) the equations are:

```
dx = x2 - x1;
dy = y2 - y1;
x3 = x2 - dy;
y3 = y2 + dx;
x4 = x1 - dy;
y4 = y1 + dx;
```

For equilateral triangle the equations are:

```
x5 = x4 + (dx - dy) / 2;
y5 = y4 + (dx + dy) / 2;
```

ARTISTIC DESCRIPTION:

For artistic enhancement we used linear blending. In linear blending we specify two endpoints and then calculated a series of intermediate values along a straight line connecting the endpoints and with the distance 0.0001. The interpolation between the two endpoints is denoted by “i”, which ranges from 0 to 1. the values of “i” is between 0 and 1 and each time i increment by 0.0001. The output is a linear combination of the two endpoints.

```
double r1 = sr + i * (er - sr);
double g = sg + i * (eg - sg);
double b = sb + i * (eb - sb);
```

for the endpoint y value is same, to calculated the new “y” value:

```
double ny = l[1] - ((pow(i,2)) * (f[0] - l[0]));
G_line(l[0], ny, f[0], ny);
```

And I have added sun, house and birds.

House: for house we use the same equations for square and triangle from Pythagoras tree and little door.

Birds: with the two G_line’s we created birds.

Sun: To create a sun we calculated the radius “r” and pass the radius as parameter to G_circle.

```
dx1 = d[0] - c[0]; //d,c are center point.
dy1 = d[1] - c[1];
r = sqrt((dx1 * dx1) + (dy1 * dy1));
```

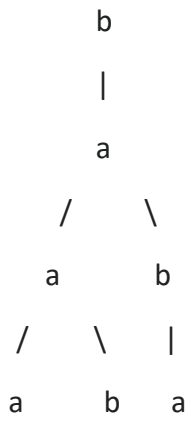
2) L SYSTEMS

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

Citation: https://en.wikipedia.org/wiki/L-system

Design Paradigm & Mathematical Description:

In the fractal we use formal grammar. In here we use symbols to build strings. For instance, let just say b=a and a=ab.



with the above we know how grammar works. For each iteration we define it as production rules and starting “b” as axiom.

Now also use different symbols like +, - [and]. In our program we use A and F character and with following axiom and production rules.

Axiom: A

Production Rule 1: A-> `F−[[A]+A]+F[+FA]−A`

Production Rule 2: F-> 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]+F[+FA]-A

Iteration 3: FF-[[F-[[A]+A]+F[+FA]-A]+ F-[[A]+A]+F[+FA]-A]+FF[FF F-[[A]+A]+F[+FA]-A]- F-[[A]+A]+F[+FA]

for the above production rules we have created a struct function for the rules and after each production rules the string will be stored in character u[100] and then we will copy each rule into new character v[100] and then string interpreter will interpret the rules.

Below two functions for u and v:

```
       void execute_rule(char c) {
int i;
for (i = 0; i < rule_num; i++) {
    if (prd[i].var == c) {
```

```
            strcat(v, prd[i].rule);
            return;
        }
    }
    strncat(v, &c, 1);
}

void apply_rules() {
    int i, len = strlen(u);
    v[0] = '\0';
    for (i = 0; i < len; i++) {
        execute_rule(u[i]);
    }
    strcpy(u, v);
}
```

Having said that each character has function as describe below:

F= move forward

dx = x + flen * cos(current_angle*M_PI/180);
dy = y + flen * sin(current_angle*M_PI/180);

here current_angle is 0. Before anything the turtle Initial angle is zero and each time turtle will change it direction with 22.5 degrees. And flen is how much the turtle should move.

+ and - :

If it's a - or +, the angle of the line being draw changes clockwise or counterclockwise by a certain degree amount. In our program, that certain amount is 22.5◦. which updates the current_angle.

```
else if(direction=='+')
       current_angle+=angle;

   else if(direction=='-')
          current_angle -= angle;
```

[and] :

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.

```
else if(direction== '['){

       sc++;
       stackx[sc] = x;
       stacky[sc] = y;
       stackangle[sc] = current_angle;
       }
   else if(direction== ']')
```

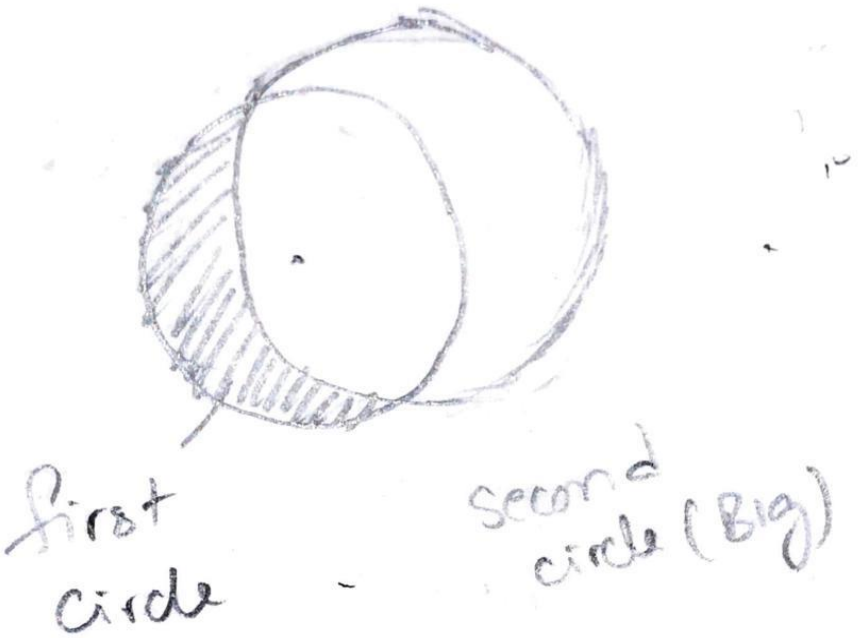
```
{
  x = stackx[sc];
  y = stacky[sc];
  current_angle = stackangle[sc];
  sc--;
 }
```

Artistic Description:

Our initial idea was to create just with tree and a circle type moon but after thinking about the moon we got an idea to make a crescent type moon along with the night effect.

For the crescent we use the below formula:

First I created two circles overlapping like shown below:



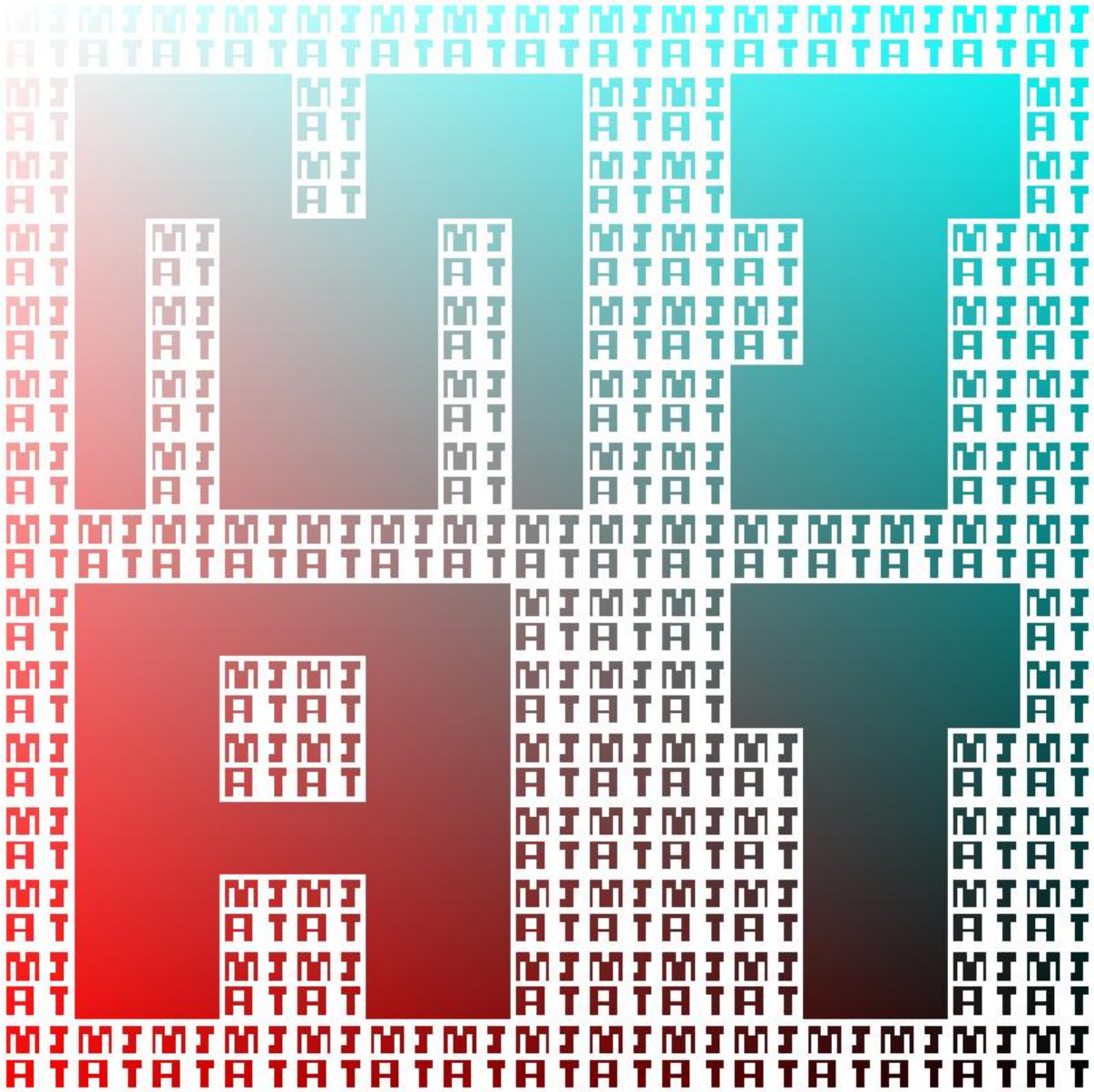
The circles has different formulas:

First circle: `swidth*0.8, sheight*0.8, sheight*0.1`

Second circle: `swidth*0.81, sheight*0.81, sheight*0.09`

I was going to linearly blend the background between two similar tones of blue to give night effect. Lastly, I added a simple black border frame the piece.

3) IFS



Fractal Description

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

Citation: https://en.wikipedia.org/wiki/Iterated_function_system

Design Paradigm & Mathematical Description:

Iterated Function Systems are the method of fractals which results in self-similar fractals.

We use the three main functions scale, translate and rotation.

Scale: scale function is use to small the line or rectangle and when we scale the line it will start on the origin.

```
void scale(double sfx, double sfy) {

                x *= sfx;
                y *= sfy;
        }
```

Translate: This function is used for change the place from origin to where ever the user wants to.

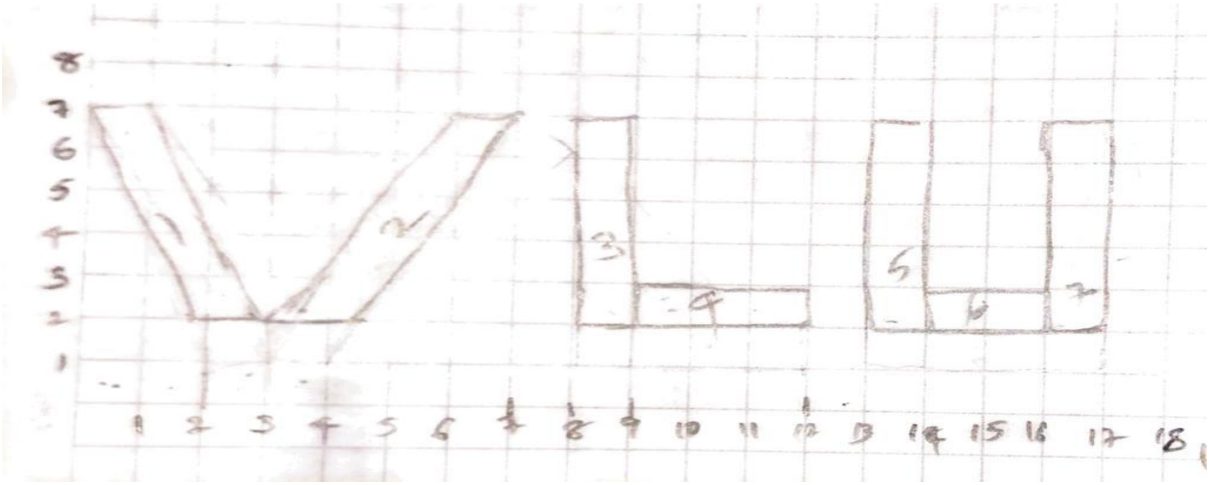
```
void translate(double dx, double dy) {

              x = x + dx;
              y = y + dy;
            }
```

Rotate: Rotate function is used to change the angle of the line.

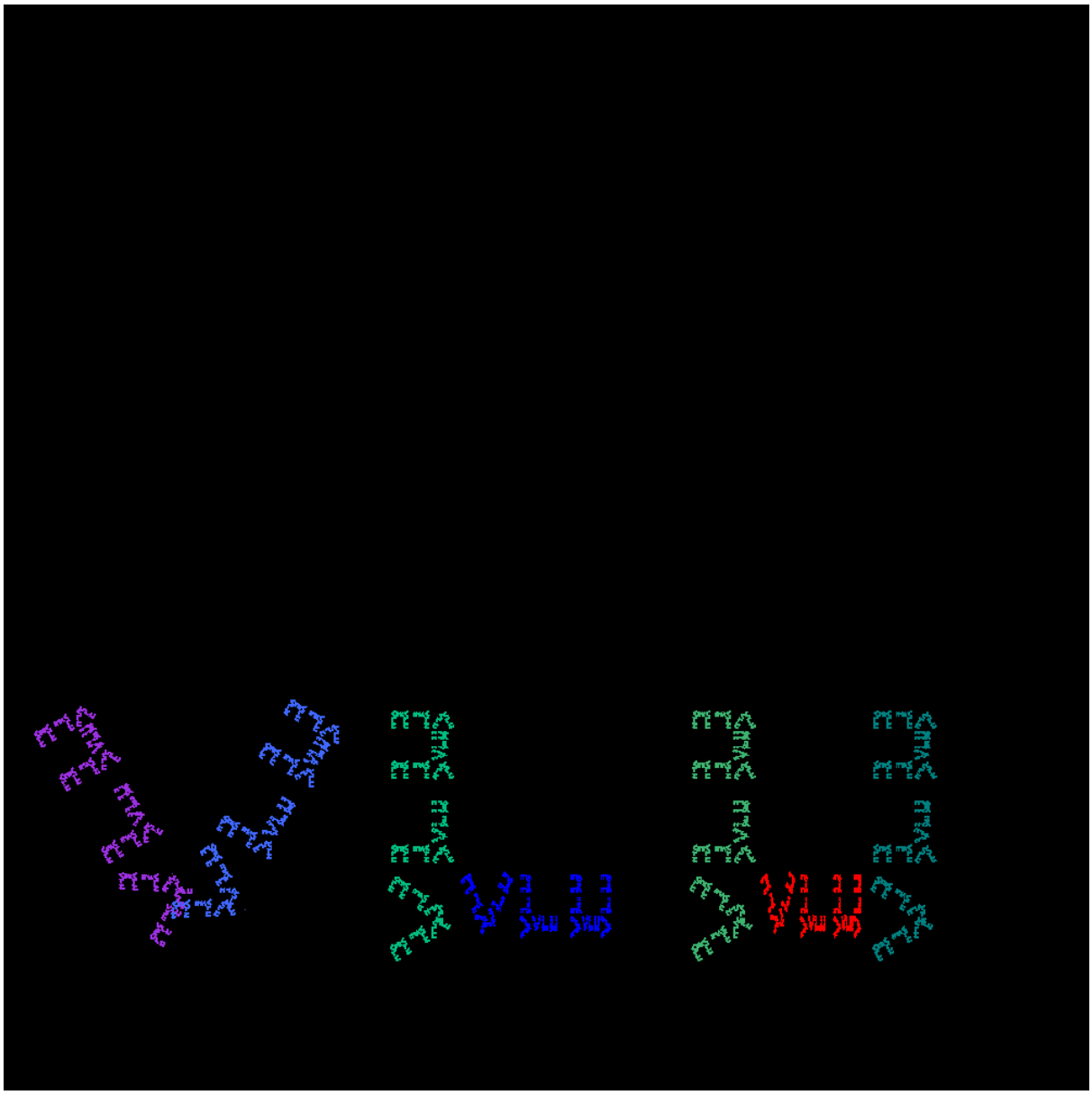
```
void rotate(double d) {
double r, a;
double t = d * (M_PI / 180);
double temp;
double c, s;
c = cos(t);
s = sin(t);
temp = (x * c) - (y * s);
y = (y * c) + (x * s);
x = temp;
}
```

We use different handful of rules to create to call the above functions. R is variable which has random values with the r value only we execute the rules.



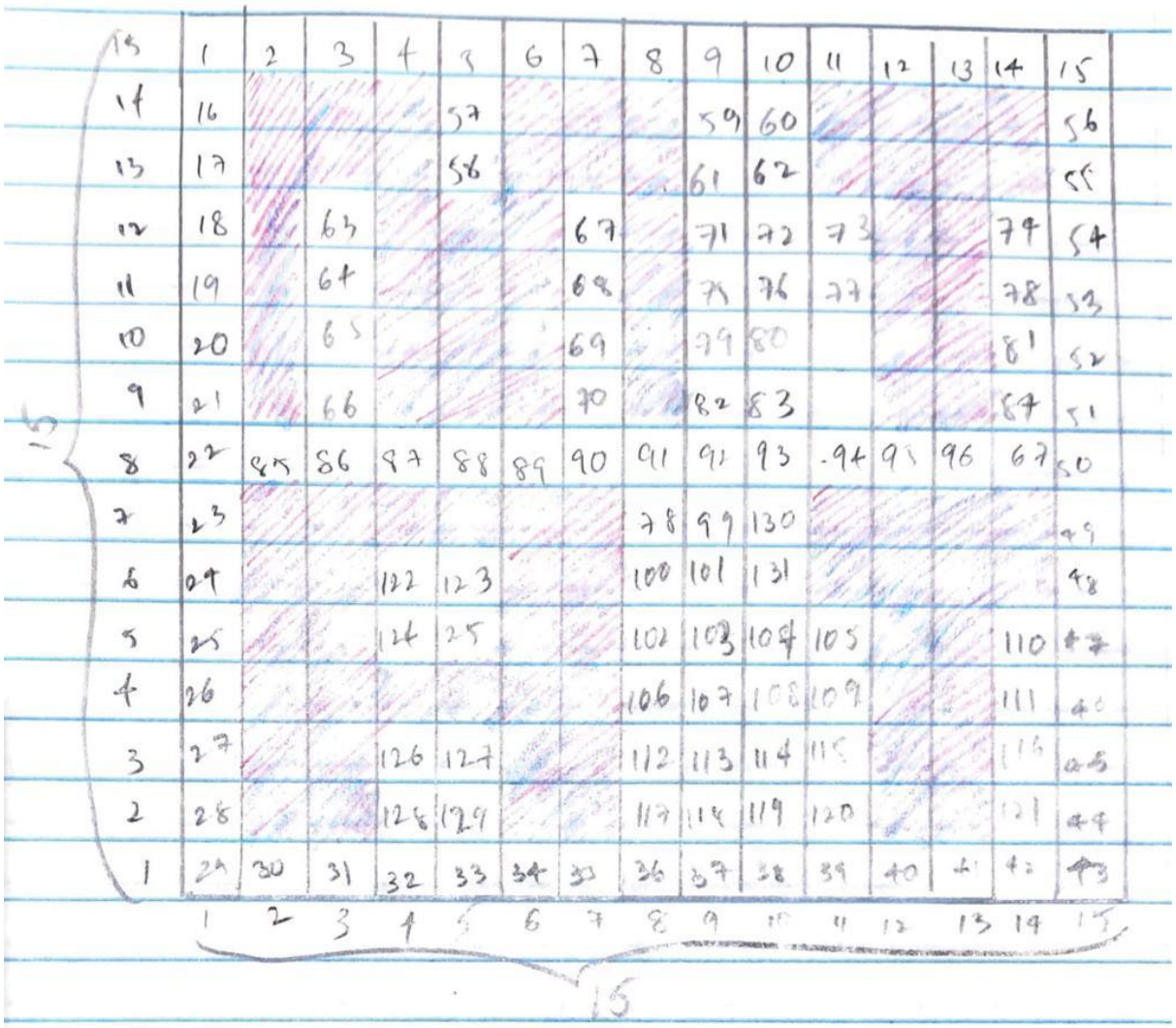
Loot at the above photo we have created 7 rules and each rule has it own function for example, v-1 rule we first scale the big rectangle to the origin and then rotate 60 degrees and at last we translate from the origin. We also add colour to each rule because it would be easy to understand where went mistake.

The output for the about diagram is:



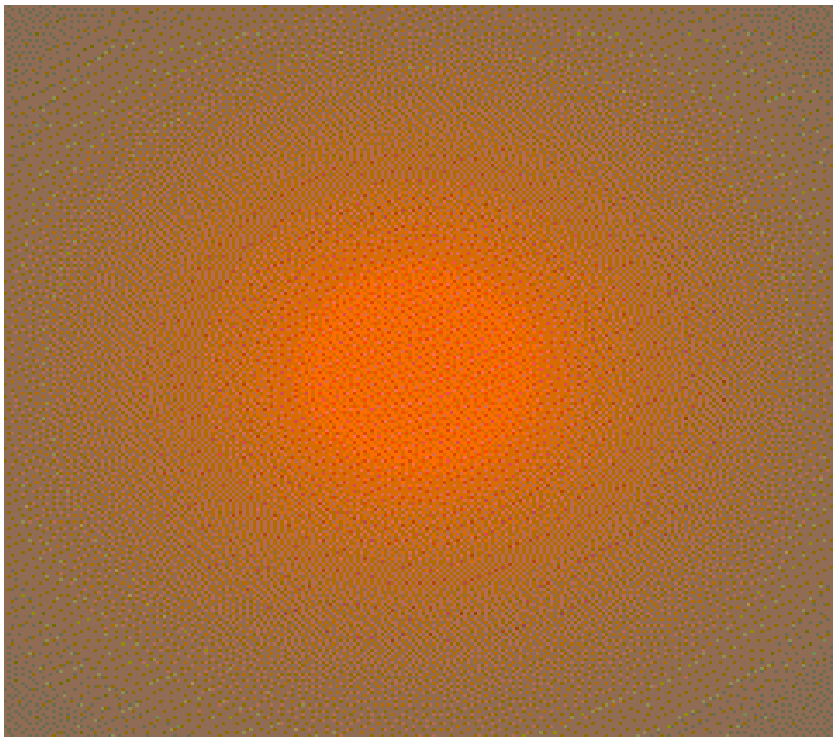
Artistic Description:

Our original idea to keep the above output and one we are searching about ifs in depth and found one image online. We got an idea of creating with our own rules and first we thought it was difficult to make but when we started doing it was easy because we created a new function for colours. We use the colour blend before creating rules.



4) Complex numbers:

Mandelbrot set movie:



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.

Design Paradigm & Mathematical Description:

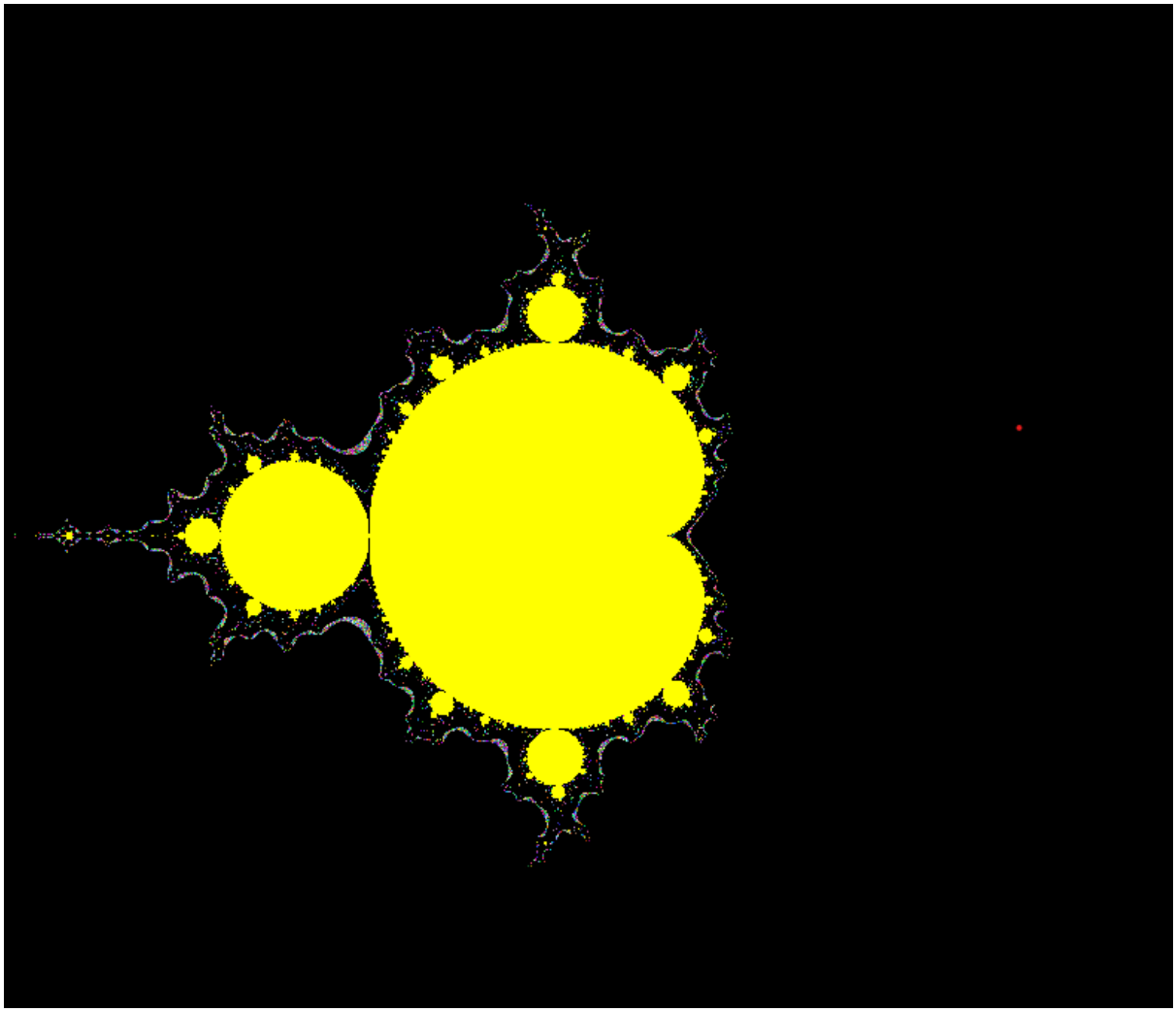
Design Paradigm:
The design paradigm for generating the Mandelbrot set involves iterating over each point in a two-dimensional grid within the complex plane and determining whether that point is within or outside the set. This is achieved by performing iterations of a mathematical formula on each point and checking the behavior of the resulting sequence.

Mathematical Description: To mathematically describe the generation of the Mandelbrot set, we define the function for a given point in the complex plane as follows:
$Z_{n+1} = Z_n^2 + C$
where Z_n is a complex number representing the sequence at iteration n, Z_{n+1} is the value of the sequence at iteration n + 1, and C is the constant complex number corresponding to the point in the complex plane.

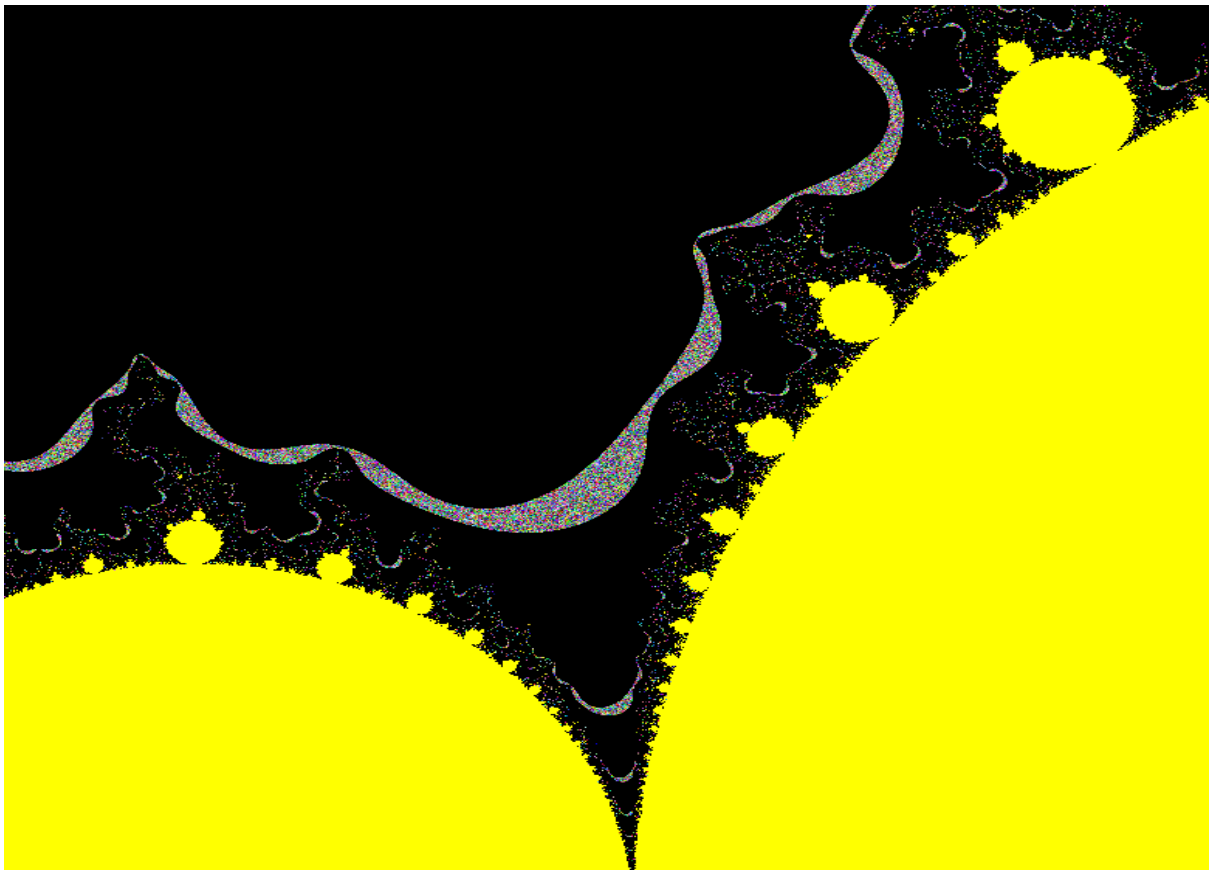
```
cx = 2 * ((x - (ssize / 2.0)) / (ssize / 2.0));
cy = 2 * ((y - (ssize / 2.0)) / (ssize / 2.0));
c = cx + cy * I;
z = 0;
```

The program uses a for loop nested within another for loop to loop over each point of the graphics window in order to create an image. We can translate any point on the x, y coordinate graph to its equivalent position on the complex number plane. Ssize =800. The set of real numbers is represented by the "x" coordinate in the complex number plane, whereas the set of imaginary numbers is represented by the "y" coordinate. The initial value of Z_o is set to zero, and we iterate the above equation until either the magnitude of Z exceeds a predefined (indicating divergence to infinity) or a maximum number of iterations is reached. For my program, I iterated for 200 repetitions.

Artistic Description:

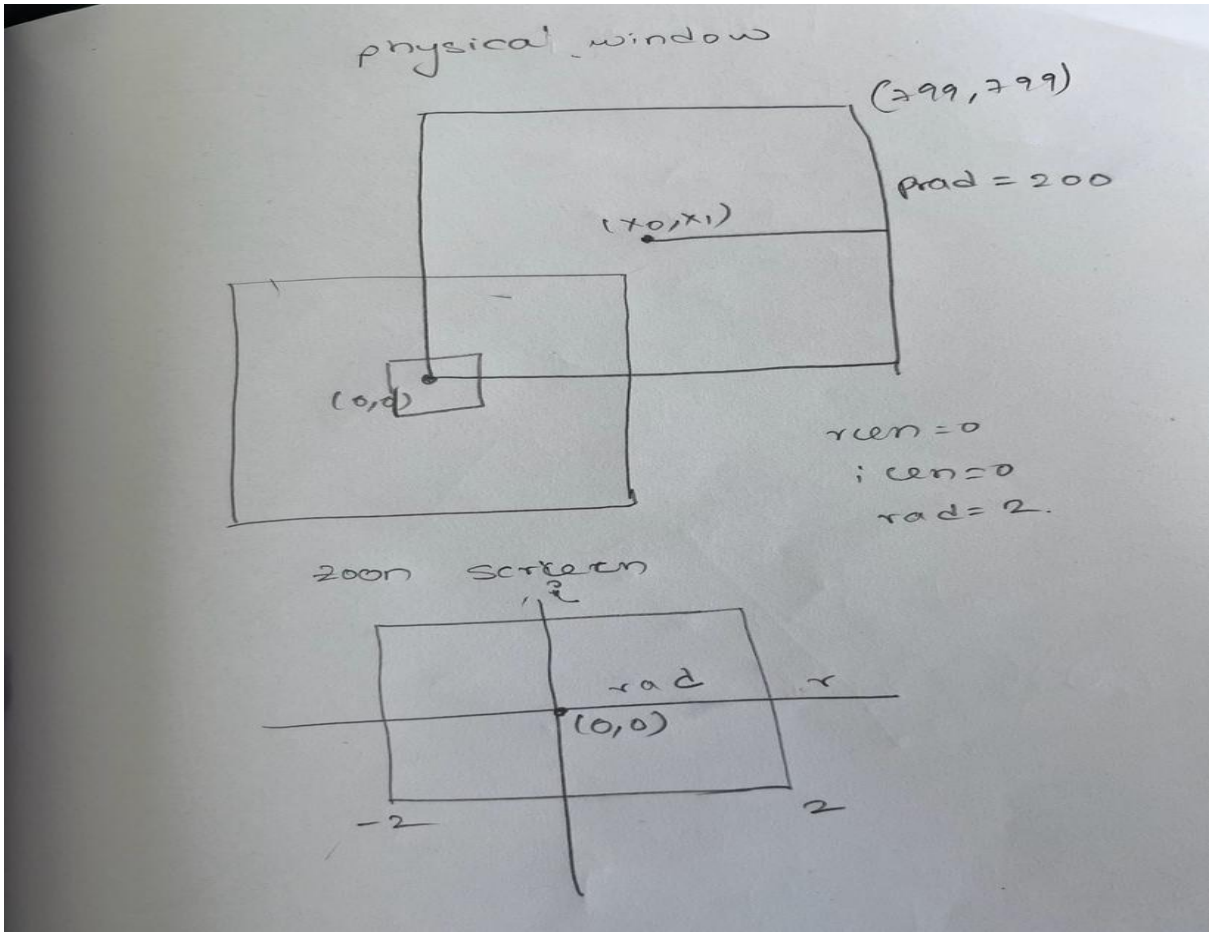


FIRST CLICK:



SECOND CLICK:

My original idea was to create a Mandelbrot movie after David example about the click-to zoom function we have created a Mandelbrot zoom program as well. And also for divergence we used drand48() so that for each number which is diverging will have different colour and converging will have different value.



When ever the user clicks two points on the window where they wanted to zoom the distance will radius for the new window.

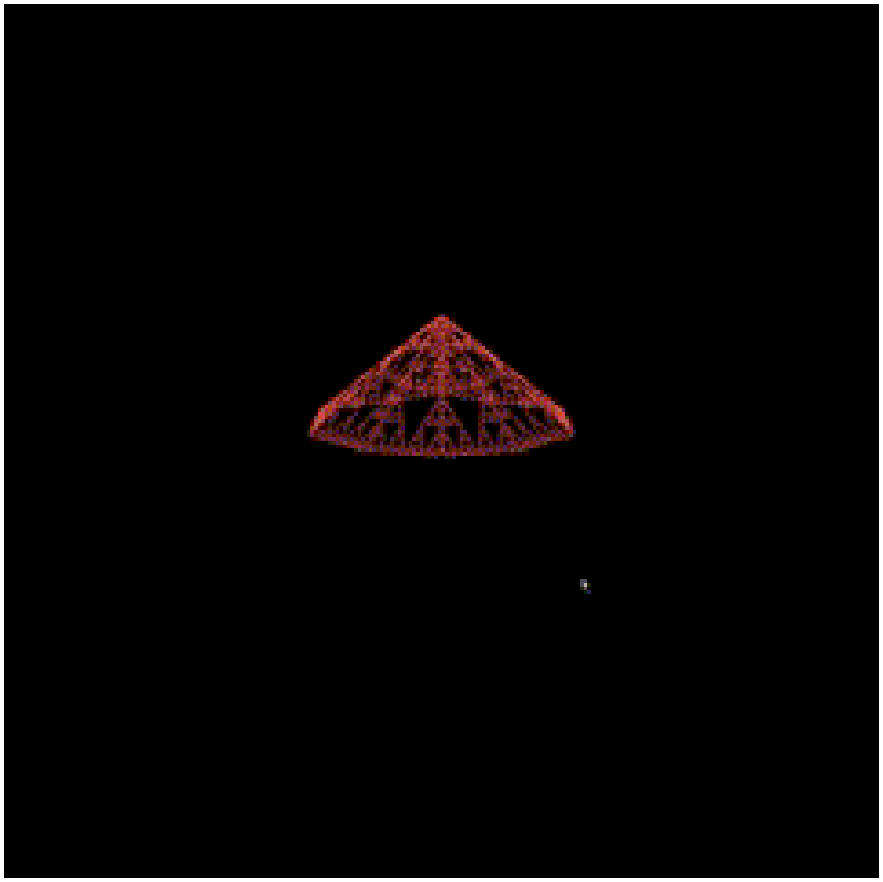
```
double prad;
  prad=wsize/2.0;

  tx = (y[0] - prad) * rad/prad ;
  ty = (y[1] - prad) * rad/prad ;

  rcen = (x[0] - prad) * rad/prad ;
  icen = (x[1] - prad) * rad/prad ;
  rad1 =  sqrt(pow(icen - ty, 2) + pow(rcen - tx, 2));
```

here x0,y0 are the first click and tx , ty are the second click.

5) WIREFRAME:



FRACTAL DESCRIPTION:

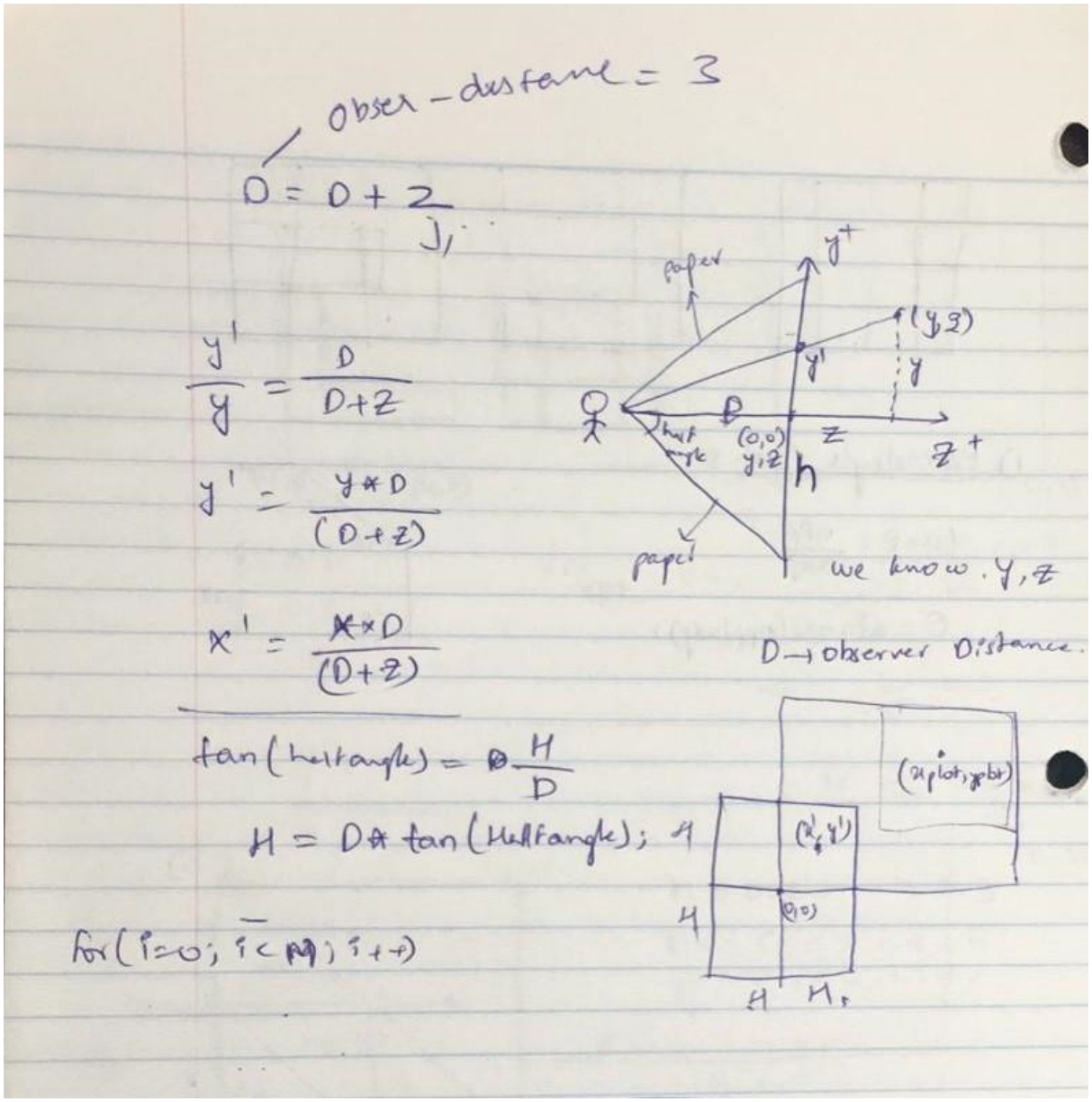
Fractal is represented as a collection of interconnected lines or wireframe structures

Design Paradigm & Mathematical Description:

The wireframe is used to display the image in 3D. For the 3D image we need 3 coordinates x,y,z. where z is the distance plane from observer from the observer the x, y plane created and let just say the observer is looking at a point on the screen (y,z). we need to change the distance from observer to z-plane. As we know the observer distance D =3 and y,z points also.

Observer distance (D)=D+Z

Know we need change the observer point to physical window. The new x' and y' will be calculated with the x,y and d. Along with angle.



Artistic Description:

For the artistic description we have created sierpinksi triangle in 3D. we made the triangle rotate 360degrees over the plane.

As we know how to create sierpinski triangle we created the triangle recursively however in 3D we created a new function build_sierpinski_cone:

```
double n,k ;
  double a,x[100],z[100],yv ;

  N = 0 ;
  n = 8 ;
  for (k = 0; k <= n ; k++) {
    a = k * 2*M_PI/n ;
    x[k] = cos(a) ;
    z[k] = sin(a) ;
  }

  yv = 0 ;
  double yh = 1 ;
  for (k = 0; k < n ; k++) {
    save_line(0,yh,0,  x[k],yv,z[k]) ;
    save_line(x[k],yv,z[k],    x[k+1],yv,z[k+1]) ;
  }

  for (k = 0; k < n ; k++) {
    sierpinski(0,yh,0,  x[k],yv,z[k],  x[k+1],yv,z[k+1],  4) ;
  }
```

The above function builds the Sierpinski _cone by calling the save_line and sierpinski functions. It initializes variables n and k and an array x and z to store the x and z coordinates of the octagon vertices. It sets N (presumably a global variable) to 0 and assigns the value 8 to n. It enters a loop from 0 to n, calculating the x and z coordinates of each vertex of the octagon using the formula x[k] = cos(a) and z[k] = sin(a), where a is the angle based on the current iteration. It then sets yv to 0 and yh to 1.

At last we wanted to fill the sierpinski triangle with the colors drand48() recursively however we mistakenly created call the drabd48 ouside the function in test_sierpinski_rotate(). Which created more beautiful image than we excepted.

CODE:

1) THE PYTHAGORAS TREE FRACTAL

```
#include "FPToolkit.c"

void drawTree(int x1, int y1, int x2, int y2, int level) {
  if (level == 0) {
      return;
  }
  double dx = x2 - x1;
  double dy = y2 - y1;

  double x3 = x2 - dy;
  double y3 = y2 + dx;

  double x4 = x1 - dy;
  double y4 = y1 + dx;

  double x5 = x4 + (dx - dy) / 2;
  double y5 = y4 + (dx + dy) / 2;
  G_rgb(0, 1, 0);
  G_fill_triangle(x3, y3, x4, y4, x5, y5);

  G_rgb(0, 0, 0);
  G_line(x1, y1, x2, y2);
  G_line(x2, y2, x3, y3);
  G_line(x3, y3, x4, y4);
  G_line(x4, y4, x1, y1);
  double a[4], b[4];
  G_rgb(0.4, 0.2, 0.1); // brown
  a[0] = x1;
  b[0] = y1;
  a[1] = x2;
  b[1] = y2;
  a[2] = x3;
  b[2] = y3;
  a[3] = x4;
  b[3] = y4;
  double numab = 4;
  G_fill_polygon(a, b, numab);

  drawTree(x4, y4, x5, y5, level - 1);
  drawTree(x5, y5, x3, y3, level - 1);
  }
int main() {
  double swidth, sheight;
  double center_x, center_y, size, x3, y3, x4, y4, i, r1, x, y;
  double lowleftx, lowlefty, width, height;
  double lowleftx1, lowlefty1, width1, height1;
```

```
swidth = 800;
sheight = 800;
G_init_graphics(swidth, sheight);
double l[2], f[2]; ////LINEAR BLENDING
double sr = 255 / 255.0;
double sg = 0 / 255.0;
double sb = 0 / 255.0;

double er = 255 / 255.0;
double eg = 255 / 255.0;
double eb = 0 / 255.0;
l[0] = 0;
l[1] = 800;
f[0] = swidth;
f[1] = 800;
for (i = 0; i < 1; i += 0.0001) {

  double r1 = sr + i * (er - sr);
  double g = sg + i * (eg - sg);
  double b = sb + i * (eb - sb);

  G_rgb(r1, g, b);
  double ny = l[1] - ((pow(i,2)) * (f[0] - l[0]));

  G_line(l[0], ny, f[0], ny);
}

double er2 = 194 / 255.0; ////SUN
double eg2 = 178 / 255.0;
double eb2 = 128 / 255.0;
G_rgb(er2, eg2, eb2);
lowleftx1 = 0;
lowlefty1 = 0;
width1 = 800;
height = 200;
G_rectangle(lowleftx, lowlefty, width, height);
G_fill_rectangle(lowleftx1, lowlefty1, width1, height);

double d[2], c[2], r, dx1, dy1;
d[0] = 100;
d[1] = 620;
c[0] = 150;
c[1] = 620;
dx1 = d[0] - c[0];
dy1 = d[1] - c[1];
r = sqrt((dx1 * dx1) + (dy1 * dy1));
G_rgb(1, 0, 0);
G_circle(d[0], d[1], r);
G_fill_circle(d[0], d[1], r);

double u[2];//HOUSE
double v[2];
u[0] = 500;
u[1] = 80;
v[0] = 700;
```

```
v[1] = 80;
double dx3;
double dy3;
double x10 = u[0];
double y10 = u[1];
double x11 = v[0];
double y11 = v[1];
dx3 = x11 - x10;
dy3 = y11 - y10;

double x12 = x11 - dy3;
double y12 = y11 + dx3;

double x13 = x10 - dy3;
double y13 = y10 + dx3;

double x14 = x13 + (dx3 - dy3) / 2;
double y14 = y13 + (dx3 + dy3) / 2;
G_rgb(1, 0.8, 0.8);
G_fill_triangle(x13, y13, x12, y12, x14, y14);

G_rgb(0, 0, 0);
G_line(x10, y10, x11, y11);
G_line(x11, y11, x12, y12);
G_line(x12, y12, x13, y13);
G_line(x13, y13, x10, y10);
double a1[4], b1[4];
G_rgb(2.10, 0.24, 0.24); // brown
a1[0] = x10;
b1[0] = y10;
a1[1] = x11;
b1[1] = y11;
a1[2] = x12;
b1[2] = y12;
a1[3] = x13;
b1[3] = y13;
double numab1 = 4;
G_fill_polygon(a1, b1, numab1);

double u1[2];
double v1[2];
u1[0] = 570;
u1[1] = 80;
v1[0] = 620;
v1[1] = 80;
double dx4;
double dy4;
double x101 = u1[0];
double y101 = u1[1];
double x111 = v1[0];
double y111 = v1[1];
dx4 = x111 - x101;
dy4 = y111 - y101;

int x121 = x111 - dy4;
int y121 = y111 + dx4;
```

```
int x131 = x101 - dy4;
int y131 = y101 + dx4;
G_rgb(0, 0, 0);
G_line(x101, y101, x111, y111);
G_line(x111, y111, x121, y121);
G_line(x121, y121, x131, y131);
G_line(x131, y131, x101, y101);
double a2[4], b2[4];
G_rgb(0.8, 0.8, 0); // brown
a2[0] = x101;
b2[0] = y101;
a2[1] = x111;
b2[1] = y111;
a2[2] = x121;
b2[2] = y121;
a2[3] = x131;
b2[3] = y131;
double numab2 = 4;
G_fill_polygon(a2, b2, numab2);

double w[2], e[2], j[2]; //BRIDS
w[0] = 600;
w[1] = 600;
e[0] = 590;
e[1] = 610;
j[0] = 610;
j[1] = 610;
G_rgb(0, 0, 0);
G_line(w[0], w[1], e[0], e[1]);
G_line(w[0], w[1], j[0], j[1]);
double w1[2], e1[2], j1[2];
w1[0] = 700;
w1[1] = 700;
e1[0] = 690;
e1[1] = 710;
j1[0] = 710;
j1[1] = 710;
G_rgb(0, 0, 0);
G_line(w1[0], w1[1], e1[0], e1[1]);
G_line(w1[0], w1[1], j1[0], j1[1]);
double w2[2], e2[2], j2[2];
w2[0] = 500;
w2[1] = 500;
e2[0] = 490;
e2[1] = 510;
j2[0] = 510;
j2[1] = 510;
G_rgb(0, 0, 0);
G_line(w2[0], w2[1], e2[0], e2[1]);
G_line(w2[0], w2[1], j2[0], j2[1]);

int x1 = 200;
int y1 = 80;
int x2 = 280;
int y2 = 80;
```

```
  drawTree(x1, y1, x2, y2, 10);

  int key;
  key = G_wait_key();
  G_save_to_bmp_file("demo52.bmp");
  return 0;
}
```

2) L SYSTEMS

```
#include "FPToolkit.c"


struct production {
    char var;
    char axiom[1000];
    char rule[100];
};

struct production prd[10];
int rule_num = 0;
char u[100000000], v[10000000];
double stackx[100000], stacky[100000], stackangle[1000000];
int sc = -1;
double angle, gangle, flen=4;
double x = 400, y = 100;

void execute_rule(char c) {
    int i;
    for (i = 0; i < rule_num; i++) {
        if (prd[i].var == c) {
            strcat(v, prd[i].rule);
            return;
        }
    }
    strncat(v, &c, 1);
}

void apply_rules() {
    int i, len = strlen(u);
    v[0] = '\0';
    for (i = 0; i < len; i++) {
        execute_rule(u[i]);
    }
    strcpy(u, v);
}
void draw_plant(int depth) {
    int i, len = strlen(u);
    char direction;
    double dx,dy,current_angle=gangle;
    if (depth> 0) {
```

```
        apply_rules();
        draw_plant(depth - 1);
        return;
        }
    for (i = 0; i < len; i++) {
    direction = u[i];

    for (i = 0; i < len; i++) {
                direction=u[i];
                if(direction=='F')
                {
                dx = x + flen * cos(current_angle*M_PI/180);
                dy = y + flen * sin(current_angle*M_PI/180);
                G_rgb(0,0,0);
                G_line(x, y,dx,dy);


                x = dx;
                y = dy;
                }
                else if(direction=='+')
                    current_angle+=angle;

                else if(direction=='-')
                    current_angle -= angle;

                else if(direction== '['){

                   sc++;
                   stackx[sc] = x;
                   stacky[sc] = y;
                   stackangle[sc] = current_angle;
                   }
                else if(direction== ']')
                 {
                   x = stackx[sc];
                   y = stacky[sc];
                   current_angle = stackangle[sc];
                   sc--;
                  }

    }

    }
}

void plant() {
    gangle = 90;
    angle = 22.5;

    strcpy(prd[rule_num].axiom, "A");
    rule_num++;

    prd[rule_num].var = 'A';
    strcpy(prd[rule_num].rule, "F-[[A]+A]+F[+FA]-A");
    rule_num++;
```

```
    prd[rule_num].var = 'F';
    strcpy(prd[rule_num].rule, "FF");
    rule_num++;

    strcpy(u, prd[0].axiom);
}

void border(){
  G_rgb(0,0,0);
  for(int i = 0; i <= 20; i++){
    G_line(0,i,800,i);
    G_line(i,0,i,800);
  }
  for(int i = 780; i <= 800; i++){
    G_line(0,i,800,i);
    G_line(i,0,i,800);
  }
}


int main() {
  double swidth, sheight,i;
  swidth = 800;
  sheight = 800;
  int depth;
  printf("enter the depth");
  scanf("%d",&depth);
  G_init_graphics(swidth, sheight);
  double l[2], f[2];
  double sr = 4 / 255.0;
  double sg = 6 / 255.0;
  double sb = 34 / 255.0;

  double er = 38 / 255.0;
  double eg = 35 / 255.0;
  double eb = 107 / 255.0;
  l[0] = 0;
  l[1] = 800;
  f[0] = swidth;
  f[1] = 800;
  for (i = 0; i < 1; i += 0.0001) {

    double r = sr + i * (er - sr);
    double g = sg + i * (eg - sg);
    double b = sb + i * (eb - sb);
    double ny = l[1] - (i * (f[0] - l[0]));
    G_rgb(r, g, b);
    G_line(l[0], ny, f[0], ny);
  }
  double skyColorTop0 = 22.0 / 255.0;
  double skyColorTop1 = 15.0 / 255.0;
  double skyColorTop2 = 48.0 / 255.0;
  G_rgb(245.0/255.0, 252.0/255.0, 193.0/255.0);
  G_fill_circle(swidth*0.8, sheight*0.8, sheight*0.1);
  G_rgb(0.6*skyColorTop0, 0.6*skyColorTop1, 0.6*skyColorTop2);
```

```
  G_fill_circle(swidth*0.81, sheight*0.81, sheight*0.09);
  plant();
  border();
  draw_plant(depth);

  int key;
  key = G_wait_key();
  G_save_to_bmp_file("plant143.bmp");

  return 0;
}
```

3) IFS

```
#include "FPToolkit.c"
double x = 0;
double y = 0;
double width = 800, height = 800;

void scale(double sfx, double sfy) {
  x *= sfx;
  y *= sfy;
}
void translate(double dx, double dy) {

  x = x + dx;
  y = y + dy;
}
void rotate(double d) {
  double r, a;
  double t = d * (M_PI / 180);
  double temp;
  double c, s;
  c = cos(t);
  s = sin(t);
  temp = (x * c) - (y * s);
  y = (y * c) + (x * s);
  x = temp;
}

void final(double r) {
  if (r < 1.0 / 131.0) {
    scale(1.0 / 15.0, 1.0 / 15.0);
    translate(0 / 15.0, 14.0 / 15.0);
    G_rgb(1, 1, 1);
    G_point(x * width, y * height);
  } else if (r < 2.0 / 131.0) {
    scale(1.0 / 15.0, 1.0 / 15.0);
    translate(1.0 / 15.0, 14.0 / 15.0);
    G_rgb(1, 1, 1);
    G_point(x * width, y * height);
  } else if (r < 3.0 / 131.0) {
```

```
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(2.0 / 15.0, 14.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 4.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(3.0 / 15.0, 14.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 5.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(4.0 / 15.0, 14.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 6.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(5.0 / 15.0, 14.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 7.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(6.0 / 15.0, 14.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 8.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(7.0 / 15.0, 14.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 9.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(8.0 / 15.0, 14.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 10.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(9.0 / 15.0, 14.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 11.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(10.0 / 15.0, 14.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 12.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(11.0 / 15.0, 14.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 13.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(12.0 / 15.0, 14.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 14.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
```

```
  translate(13.0 / 15.0, 14.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 15.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(14.0 / 15.0, 14.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 16.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(0 / 15.0, 13.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 17.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(0 / 15.0, 12.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 18.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(0 / 15.0, 11.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 19.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(0 / 15.0, 10.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 20.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(0 / 15.0, 9.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 21.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(0 / 15.0, 8.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 22.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(0 / 15.0, 7.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 23.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(0 / 15.0, 6.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 24.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(0 / 15.0, 5.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 25.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(0 / 15.0, 4.0 / 15.0);
```

```
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 26.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(0 / 15.0, 3.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 27.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(0 / 15.0, 2.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 28.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(0 / 15.0, 1.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 29.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(0 / 15.0, 0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 30.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(1.0 / 15.0, 0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 31.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(2.0 / 15.0, 0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 32.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(3.0 / 15.0, 0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 31.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(2.0 / 15.0, 0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 32.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(3.0 / 15.0, 0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 33.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(4.0 / 15.0, 0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 34.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(5.0 / 15.0, 0 / 15.0);
  G_rgb(1, 1, 1);
```

```
  G_point(x * width, y * height);
} else if (r < 35.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(6.0 / 15.0, 0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 36.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(7.0 / 15.0, 0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 37.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(8.0 / 15.0, 0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 38.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(9.0 / 15.0, 0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 39.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(10.0 / 15.0, 0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 40.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(11.0 / 15.0, 0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 41.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(12.0 / 15.0, 0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 42.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(13.0 / 15.0, 0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 43.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(14.0 / 15.0, 0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 44.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(14.0 / 15.0, 1.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 45.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(14.0 / 15.0, 2.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
```

```
} else if (r < 46.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(14.0 / 15.0, 3.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 47.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(14.0 / 15.0, 4.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 48.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(14.0 / 15.0, 5.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 49.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(14.0 / 15.0, 6.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 50.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(14.0 / 15.0, 7.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 51.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(14.0 / 15.0, 8.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 52.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(14.0 / 15.0, 9.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 53.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(14.0 / 15.0, 10.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 54.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(14.0 / 15.0, 11.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 55.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(14.0 / 15.0, 12.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 56.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(14.0 / 15.0, 13.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 57.0 / 131.0) {
```

```
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(4.0 / 15.0, 13.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 58.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(4.0 / 15.0, 12.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 59.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(8.0 / 15.0, 13.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 60.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(9.0 / 15.0, 13.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 61.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(8.0 / 15.0, 12.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 62.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(9.0 / 15.0, 12.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 63.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(2.0 / 15.0, 11.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 64.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(2.0 / 15.0, 10.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 65.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(2.0 / 15.0, 9.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 66.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(2.0 / 15.0, 8.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 67.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(6.0 / 15.0, 11.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 68.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
```

```
  translate(6.0 / 15.0, 10.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 69.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(6.0 / 15.0, 9.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 70.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(6.0 / 15.0, 8.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 71.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(8.0 / 15.0, 11.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 72.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(9.0 / 15.0, 11.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 73.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(10.0 / 15.0, 11.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 74.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(13.0 / 15.0, 11.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 75.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(8.0 / 15.0, 10.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 76.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(9.0 / 15.0, 10.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 77.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(10.0 / 15.0, 10.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 78.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(13.0 / 15.0, 10.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 79.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(8.0 / 15.0, 9.0 / 15.0);
```

```
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 80.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(9.0 / 15.0, 9.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 81.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(13.0 / 15.0, 9.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 82.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(8.0 / 15.0, 8.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 83.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(9.0 / 15.0, 8.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 84.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(13.0 / 15.0, 8.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 85.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(1.0 / 15.0, 7.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 86.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(2.0 / 15.0, 7.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 87.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(3.0 / 15.0, 7.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 88.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(4.0 / 15.0, 7.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 89.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(5.0 / 15.0, 7.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 90.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(6.0 / 15.0, 7.0 / 15.0);
  G_rgb(1, 1, 1);
```

```
  G_point(x * width, y * height);
} else if (r < 91.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(7.0 / 15.0, 7.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 92.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(8.0 / 15.0, 7.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 93.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(9.0 / 15.0, 7.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 94.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(10.0 / 15.0, 7.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 95.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(11.0 / 15.0, 7.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 96.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(12.0 / 15.0, 7.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 97.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(13.0 / 15.0, 7.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 98.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(7.0 / 15.0, 6.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 99.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(8.0 / 15.0, 6.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 100.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(7.0 / 15.0, 5.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 101.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(8.0 / 15.0, 5.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
```

```
} else if (r < 102.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(7.0 / 15.0, 4.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 103.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(8.0 / 15.0, 4.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 104.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(9.0 / 15.0, 4.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 105.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(10.0 / 15.0, 4.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 106.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(7.0 / 15.0, 3.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 107.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(8.0 / 15.0, 3.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 108.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(9.0 / 15.0, 3.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 109.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(10.0 / 15.0, 3.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 110.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(13.0 / 15.0, 4.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 111.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(13.0 / 15.0, 3.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 112.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(7.0 / 15.0, 2.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 113.0 / 131.0) {
```

```
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(8.0 / 15.0, 2.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 114.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(9.0 / 15.0, 2.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 115.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(10.0 / 15.0, 2.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 116.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(13.0 / 15.0, 2.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 117.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(7.0 / 15.0, 1.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 118.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(8.0 / 15.0, 1.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 119.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(9.0 / 15.0, 1.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 120.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(10.0 / 15.0, 1.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 121.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(13.0 / 15.0, 1.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 122.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(3.0 / 15.0, 5.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 123.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
  translate(4.0 / 15.0, 5.0 / 15.0);
  G_rgb(1, 1, 1);
  G_point(x * width, y * height);
} else if (r < 124.0 / 131.0) {
  scale(1.0 / 15.0, 1.0 / 15.0);
```

```
    translate(3.0 / 15.0, 4.0 / 15.0);
    G_rgb(1, 1, 1);
    G_point(x * width, y * height);
  } else if (r < 125.0 / 131.0) {
    scale(1.0 / 15.0, 1.0 / 15.0);
    translate(4.0 / 15.0, 4.0 / 15.0);
    G_rgb(1, 1, 1);
    G_point(x * width, y * height);
  } else if (r < 126.0 / 131.0) {
    scale(1.0 / 15.0, 1.0 / 15.0);
    translate(3.0 / 15.0, 2.0 / 15.0);
    G_rgb(1, 1, 1);
    G_point(x * width, y * height);
  } else if (r < 127.0 / 131.0) {
    scale(1.0 / 15.0, 1.0 / 15.0);
    translate(4.0 / 15.0, 2.0 / 15.0);
    G_rgb(1, 1, 1);
    G_point(x * width, y * height);
  } else if (r < 128.0 / 131.0) {
    scale(1.0 / 15.0, 1.0 / 15.0);
    translate(3.0 / 15.0, 1.0 / 15.0);
    G_rgb(1, 1, 1);
    G_point(x * width, y * height);
  } else if (r < 129.0 / 131.0) {
    scale(1.0 / 15.0, 1.0 / 15.0);
    translate(4.0 / 15.0, 1.0 / 15.0);
    G_rgb(1, 1, 1);
    G_point(x * width, y * height);
  } else if (r < 130.0 / 131.0) {
    scale(1.0 / 15.0, 1.0 / 15.0);
    translate(9.0 / 15.0, 6.0 / 15.0);
    G_rgb(1, 1, 1);
    G_point(x * width, y * height);
  } else {
    scale(1.0 / 15.0, 1.0 / 15.0);
    translate(9.0 / 15.0, 5.0 / 15.0);
    G_rgb(1, 1, 1);
    G_point(x * width, y * height);
  }
}
int main() {
  int width, height;
  double x, y;
  int i, q;
  width = 800;
  height = 800;
  G_init_graphics(width, height);
  double x0,y0 , x1,y1, dx, dy ;
  double sf, sf2;

  double r123,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 = 800 ;

  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 ;
    r123 = 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 = r123 + sf2*(r4-r3);
      gg = g + sf2*(g4-g3);
      bb = b + sf2*(b4-b3);
      G_rgb(rr, gg, bb) ;
      G_point(j,k) ;
    }
  }
  double r;
  for (i = 0; i < 60000000; ++i) {
    r = drand48();

    final(r);

  }
  int key;
  key = G_wait_key();
  G_save_to_bmp_file("demo14.bmp");
}
```

4) MADELBRAT MOVIE:

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

const int ssize = 800;
char fname[100];

int reps = 500;
double cx, cy;
int xp, yp;
complex c, z;


double colorStart[] = {0.0, 0.5, 1.0};   // Start color (RGB)
```

```
double colorEnd[] = {1.0, 0.5, 0.0};     // End color (RGB)
double colorScale = 0.4;                  // Color scale factor

void mandelbrot(double power) {
  double x,y;
  for (int x = 0; x < ssize; x++) {
    for (int y = 0; y < ssize; y++) {

      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) {
          break;
        }
      }
      double sf = 1.0 * k / reps;
      sf = pow(sf, 0.3);

      // Color interpolation
      double r = colorStart[0] + sf * (colorEnd[0] - colorStart[0]);
      double g = colorStart[1] + sf * (colorEnd[1] - colorStart[1]);
      double b = colorStart[2] + sf * (colorEnd[2] - colorStart[2]);

      G_rgb(r, g, b);
      G_point(x, y);
    }
  }
}

int main() {

  G_init_graphics(ssize, ssize);
  G_rgb(1, 1, 1);
  G_clear();

  int count = 0;
  for (double i = 1.0; i < 10.0; i += 0.1) {
    mandelbrot(i);
    sprintf(fname, "mandelbrot%06d.bmp", count);
    G_save_to_bmp_file(fname);
    count++;
  }
}
```

MADELBRAT ZOOM:

```
#include "FPToolkit.c"
#include "complex.h"

double wsize;
```

```
void mandelbrot(double rcen, double icen, double rad, double cutoff,
                int mlimit) {
  double delta, x, y;
  complex c;
  complex z;
  int reps;

  delta = 2 * rad / wsize;
  y = icen - rad;
  for (int yp = 0; yp < wsize; yp++) {
    x = rcen - rad;
    for (int xp = 0; xp < wsize; xp++) {
      c = x + y * I;
      z = 0;
      for (reps = 0; reps < mlimit; reps++) {
        if (cabs(z) > cutoff) {
          break;
        }
        z = z * z + c;
      }
      if (reps == mlimit) {
        G_rgb(1.0,1.0,0.0);
      } else {

        int colorIndex = reps % 10;
        switch (colorIndex) {
        case 0:

          G_rgb(drand48(), drand48(), drand48());
          break;
        default:
          G_rgb(0.0, 0.0, 0.0); // Black
          break;
        }
      }

      G_point(xp, yp);
      x = x + delta;
    }
    y = y + delta;
  }
}

int main() {
  wsize = 800.0;
  G_init_graphics(wsize, wsize);
  G_rgb(1.0, 1.0, 1.0);
  G_clear();
  double rcen, icen, rad, cutoff, rad1,rad2;
  int mlimit;
  double x[10], y[10];
  rcen = 0.0, icen = 0.0,rad=2.0,cutoff=rad;

  mlimit = 500;
  mandelbrot(rcen, icen, rad,cutoff , mlimit);
  G_wait_click(x);
```

```
  G_fill_circle(x[0], x[1], 2);
  G_wait_click(y);
  G_fill_circle(y[0], y[1], 2);
  double tx, ty ;
  double prad;
  prad=wsize/2.0;

  tx = (y[0] - prad) * rad/prad ;
  ty = (y[1] - prad) * rad/prad ;

  rcen = (x[0] - prad) * rad/prad ;
  icen = (x[1] - prad) * rad/prad ;
  rad1 =  sqrt(pow(icen - ty, 2) + pow(rcen - tx, 2));
  cutoff=rad;
  mandelbrot(rcen, icen, rad1, cutoff, mlimit);
  G_display_image();

  double p[2],q[2];
  G_wait_click(p);
  G_fill_circle(p[0], p[1], 2);
  G_wait_click(q);
  G_fill_circle(q[0], q[1], 2);
  double tx1, ty1,rcen1,icen1,cutoff1 ;
  double prad1;
  prad1=wsize/2.0;

  tx1 = (q[0] - prad1) * rad1/prad1 ;
  ty1 = (q[1] - prad1) * rad1/prad1 ;

  rcen1 = (p[0] - prad1) * rad1/prad1 ;
  icen1 = (p[1] - prad1) * rad1/prad1 ;
  rad2 =  sqrt(pow(icen1 - ty1, 2) + pow(rcen1 - tx1, 2));
  cutoff1=rad2;
  mandelbrot(rcen1, icen1, rad2, cutoff1, mlimit);
  G_display_image();

  int key;
  key = G_wait_key();
  return 0;
}
```

5) WIREFRAME

```
#include "FPToolkit.c"

/*

        Y+
        |
        |     * (y,z)
        |
<-------+--------Z+
   D    |
        |
        |
        |


y'     y
- =  ---
D    D+z

H = D*tan(halfangle) ;

with the x-axis perpendicular to this plane.

*/

#define M 700000
double Wsize = 800  ; // window size ... choose 600 for repl
double X[M],Y[M],Z[M] ;
double Xplot[M],Yplot[M] ;
int N = 0 ;


int translate(double dx, double dy, double dz)
{
  int i ;

  for (i = 0 ; i < N ; i++) {
    X[i] += dx ;
    Y[i] += dy ;
    Z[i] += dz ;
  }
}



int rotate_x(double degrees)
// Y[] and Z[] will change but X[] will be unchanged
{
  double radians,c,s,temp ;
  int i ;

  radians = degrees*M_PI/180 ;
```

```
  c = cos(radians) ;
  s = sin(radians) ;
  for (i = 0 ; i < N ; i++) {
    temp =  c*Y[i] - s*Z[i] ;
    Z[i] =  s*Y[i] + c*Z[i] ;
    Y[i] = temp ;
  }
}



int rotate_y(double degrees)
// X[] and Z[] will change but Y[] will be unchanged
{
  double radians,c,s,temp ;
  int i ;

  radians = degrees*M_PI/180 ;
  c = cos(radians) ;
  s = sin(radians) ;
  for (i = 0 ; i < N ; i++) {
    temp =  c*X[i] + s*Z[i] ;
    Z[i] = -s*X[i] + c*Z[i] ;
    X[i] = temp ;
  }
}



int rotate_z(double degrees)

{
  double radians,c,s,temp ;
  int i ;

  radians = degrees*M_PI/180 ;
  c = cos(radians) ;
  s = sin(radians) ;
  for (i = 0 ; i < N ; i++) {
    temp =  c*X[i] - s*Y[i] ;
    Y[i] =  s*X[i] + c*Y[i] ;
    X[i] = temp ;
  }
}



int project(double observer_distance, double halfangle_degrees)
{
  int i ;
  double D,tan_half,Window_half ;

  D = observer_distance ;
  tan_half= tan( halfangle_degrees*M_PI/180 ) ;
  Window_half = Wsize/2 ;
```

```
  for (i = 0 ; i < N ; i++) {
    Xplot[i] = (X[i]/(D + Z[i]))  * (Window_half /tan_half) +
Window_half ;
    Yplot[i] = (Y[i]/(D + Z[i]))  * (Window_half /tan_half) +
Window_half ;
  }

}


int draw()
{
  int i ;
  for (i = 0 ; i < N ; i=i+2) {
    G_line(Xplot[i],Yplot[i],  Xplot[i+1],Yplot[i+1]) ;
  }

}




int print_object()
{
  int i ;
  for (i = 0 ; i < N ; i=i+2) {
    printf("(%lf, %lf, %lf)   (%lf, %lf, %lf)\n",
         X[i],Y[i],Z[i],  X[i+1],Y[i+1],Z[i+1]) ;
  }

  printf("================================\n") ;

  for (i = 0 ; i < N ; i=i+2) {
    printf("(%lf, %lf)   (%lf, %lf)\n",
         Xplot[i],Yplot[i],  Xplot[i+1],Yplot[i+1]) ;
  }
}



int save_line(double xs, double ys, double zs,
           double xe, double ye, double ze)
{

  if (N+1 >= M) {
    printf("full\n") ;
    return 0 ;
  }

  X[N] = xs ; Y[N] = ys ; Z[N] = zs ; N++ ;
  X[N] = xe ; Y[N] = ye ; Z[N] = ze ; N++ ;

  return 1 ;
}
```

```
int sierpinski(double xa, double ya, double za,
            double xb, double yb, double zb,
            double xc, double yc, double zc,
            int depth)
{
  double xab,yab,zab,  xbc,ybc,zbc, xca,yca,zca ;

  if (depth == 0) {
    save_line(xa,ya,za,  xb,yb,zb) ;
    save_line(xb,yb,zb,  xc,yc,zc) ;
    save_line(xc,yc,zc,  xa,ya,za) ;
    return 1 ;
  }

  xab = (xa + xb)/2 ;
  yab = (ya + yb)/2 ;
  zab = (za + zb)/2 ;

  xbc = (xb + xc)/2 ;
  ybc = (yb + yc)/2 ;
  zbc = (zb + zc)/2 ;

  xca = (xc + xa)/2 ;
  yca = (yc + ya)/2 ;
  zca = (zc + za)/2 ;

  sierpinski(xa,ya,za,  xab,yab,zab,  xca,yca,zca, depth-1) ;
  sierpinski(xb,yb,zb,  xbc,ybc,zbc,  xab,yab,zab, depth-1) ;
  sierpinski(xc,yc,zc,  xca,yca,zca,  xbc,ybc,zbc, depth-1) ;

  return 1 ;
}
```

```
int build_sierpinski_cone()
{
  double n,k ;
  double a,x[100],z[100],yv ;

  N = 0 ;
  n = 8 ;
  for (k = 0; k <= n ; k++) {
    a = k * 2*M_PI/n ;
    x[k] = cos(a) ;
    z[k] = sin(a) ;
  }

  yv = 0 ;
```

```
  double yh = 1 ;
  for (k = 0; k < n ; k++) {
    save_line(0,yh,0,  x[k],yv,z[k]) ;
    save_line(x[k],yv,z[k],    x[k+1],yv,z[k+1]) ;
  }

  for (k = 0; k < n ; k++) {
    sierpinski(0,yh,0,  x[k],yv,z[k],  x[k+1],yv,z[k+1],  4) ;
  }

}






int test_sierpinski_rotate()
{

  G_init_graphics(Wsize,Wsize) ;


  build_sierpinski__cone() ;

  while (1) {
    G_rgb(0,0,0) ;
    G_clear() ;
    G_rgb(drand48(),drand48(),drand48()) ;

    project(3,45) ; draw() ;


    rotate_x(2) ;
    if (G_wait_key() == 'q') { break ; }
  }

}




int main()
{
  test_sierpinski_rotate() ;
}
```