Lab2

By Xiyuan He (xh31) and Xuan Liu (xl94)

File Organization

All the files are divided into two folders, One is images, which contains all the images. The other is files, where you can see txt files stored separately

Additional Shapes

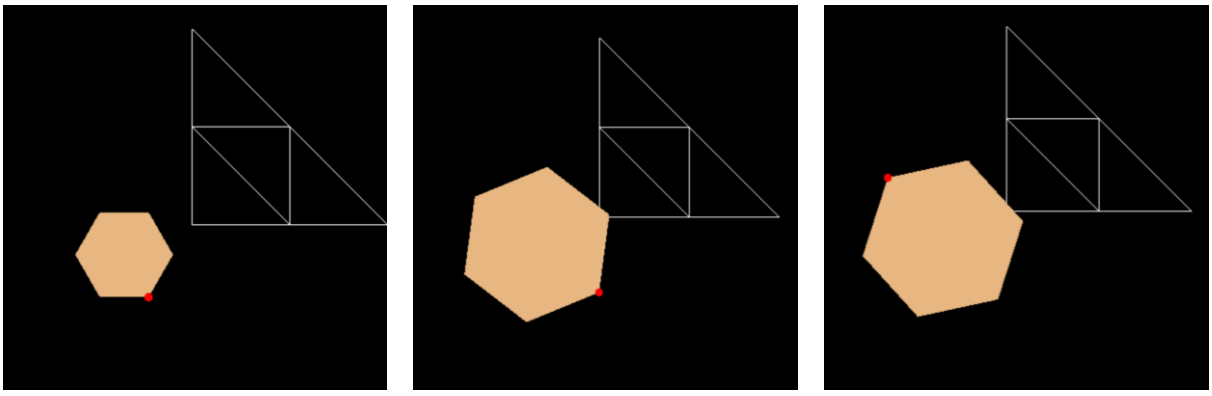
We add hexagon, octagon, circle, and one additional shape Star, the pictures are shown below. For every shape, we define two constructors. One is a given center point and a reasonable length. The center defaults to (0, 0), and the distance from each vertex to the center of the graph is set by the given length. Another constructor is set the required vertices of a given graph. This constructor can allow irregular polygons to appear and can also allow we to draw a new graph by mouse manipulation. We will take the additional shape Star for example.

```
Star2::Star2() :Geom2() {
   init();
   Star2 def(Pt2(0, 0), 50);
   for (int j = 0; j < size(); j++)
       _pts[j] = def._pts[j];
}

Star2::Star2(const Pt2& c, double r) {
   init();

   double degs[12] = { 0, };
   for (int i = 0; i < 12; i++) {
       degs[i] = i * M_PI / 6;
   }
   int flag = -1;
   for (int j = 0; j < 12; j++) {
       double temp_r = 0;
       if (flag == -1) {
           temp_r = r / 1.732;
       }
       else {
           temp_r = r;
       }
       flag *= -1;
       double dx = cos(degs[j]) * temp_r;
       double dy = sin(degs[j]) * temp_r;
       _pts[j] = c + Vec2(dx, dy, 0);
   }
}
```

We create a star with six vertices as a new shape and the definition of the shape is in **TinyGeom.cpp**. In order to draw a suitable star shape, we take the default point as the center and the given length as the side length to determine the position of the twelve points. Among

them, the distance from the six points to the center is the initial side length, and the distance from the other six points to the center is around half of the initial length. The details and the other shapes can be seen below.

In order to maintain the convexity of the shape, we assume the mouse manipulation as the uniform scaling from the centroid of the shape and apply the transformation of dragged vertex to each vertex of the current shape in **GeometryViewer.cpp**. We take **hexagon** as an example and the result can be seen below. We could use this algorithm to maintain the convexities of all the shapes including the **Star** we create.



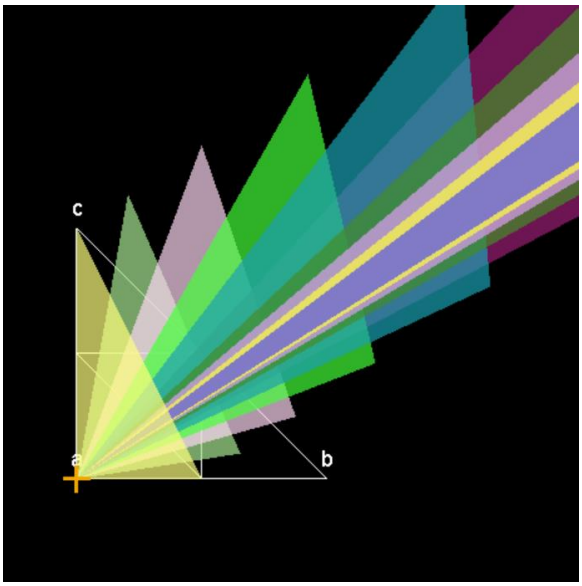
Matrix definition

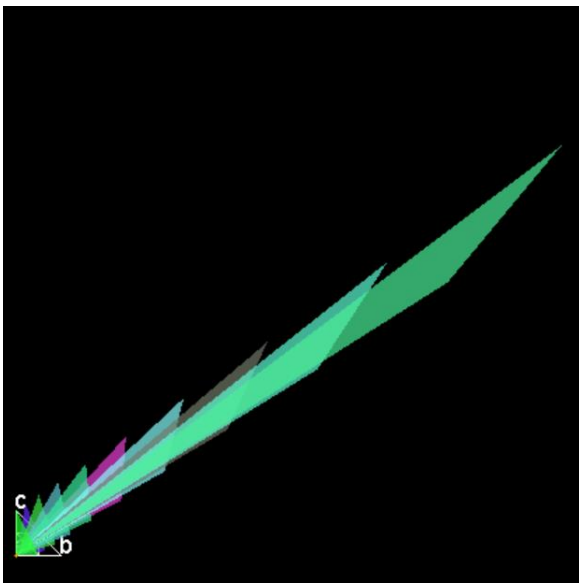
For overloaded matrix addition(operator +) and multiplication(operator *), we implement arbitrary matrix calculations of the same size. For inverse operations(operator !), we assume that all that is needed to be calculated are 3 * 3 matrices, which will make the algorithm more concise.

Simple operators

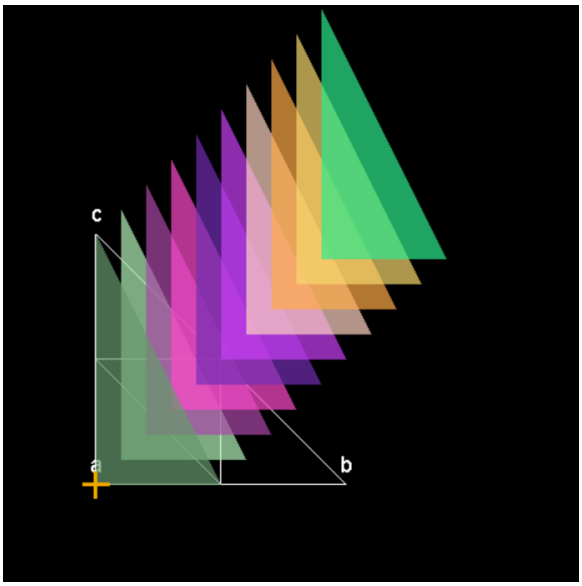
All the data (coordinates) can be seen in SimpleOps.txt, here we show the final result graphs.

Scale :

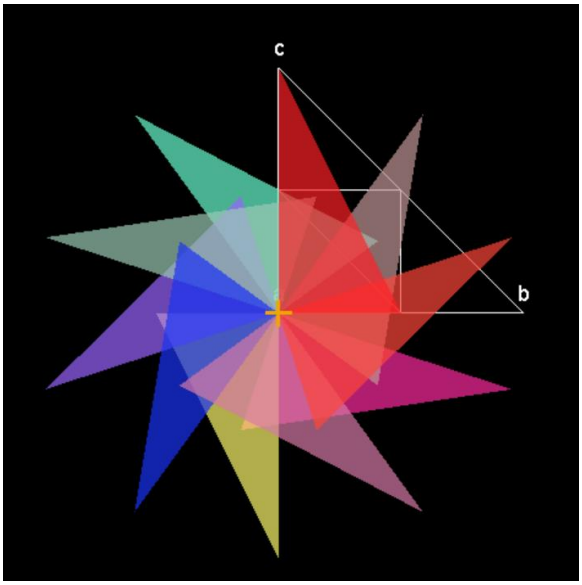




Shift:



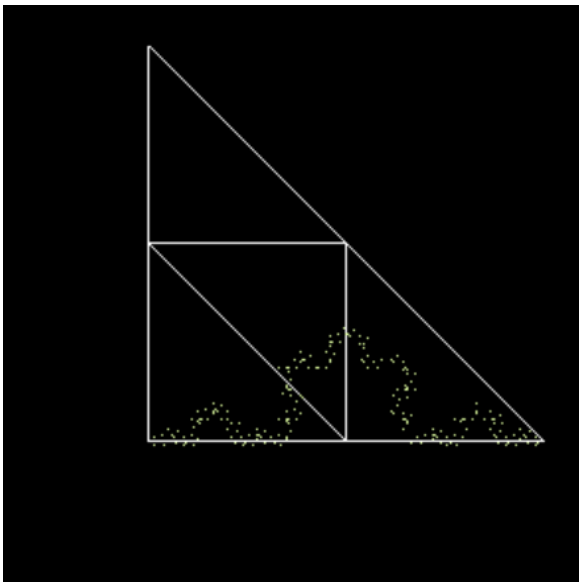
Spin:

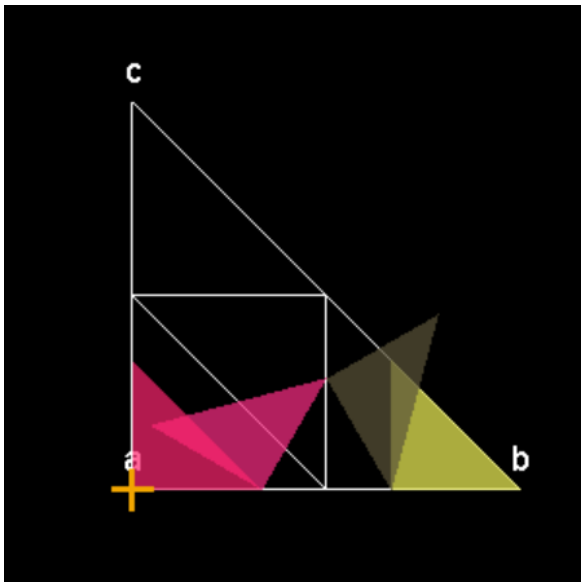


Fractals

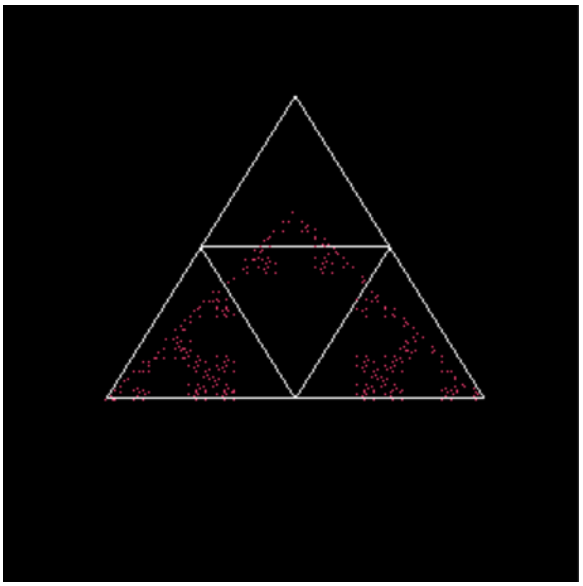
For each fractal we construct, we will give two pictures, one is showing the transformation we apply in **Geometry Viewer**, and the other one is showing the final result in **IFSViewer**. Details can be seen in **BasicFractals.txt** and pictures can be seen in file **images**.

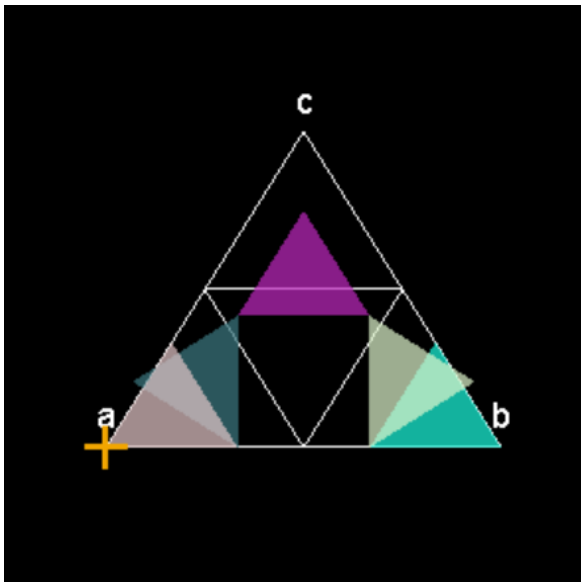
KochSnowflakeSide: Apply about four times to get the fractal



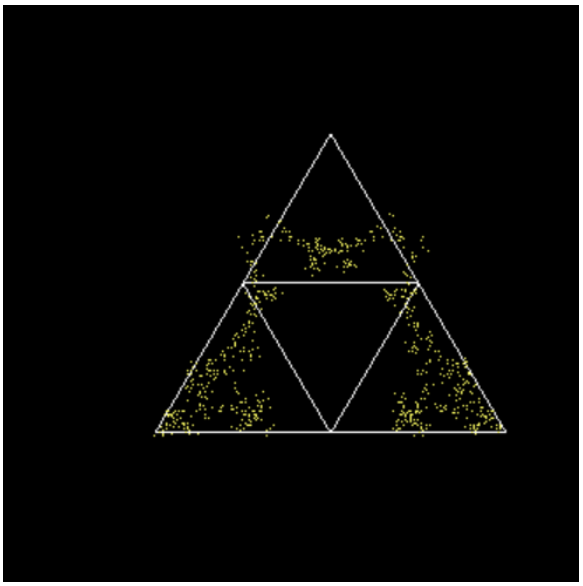


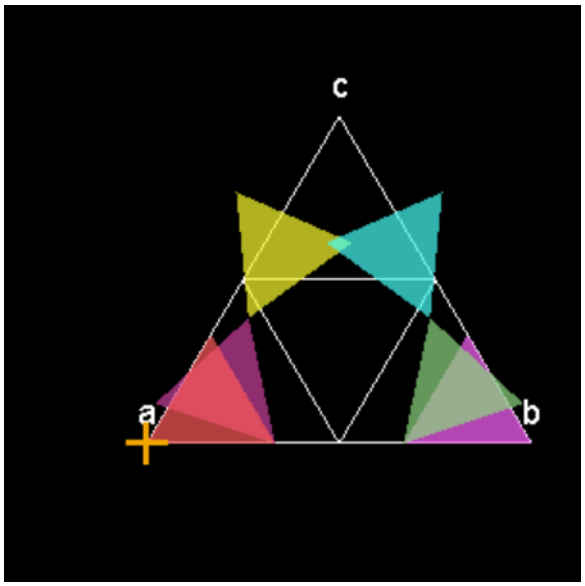
SnowflakeSide4: Apply about four times to get the fractal





SnowflakeSide5: : Apply about four times to get the fractal

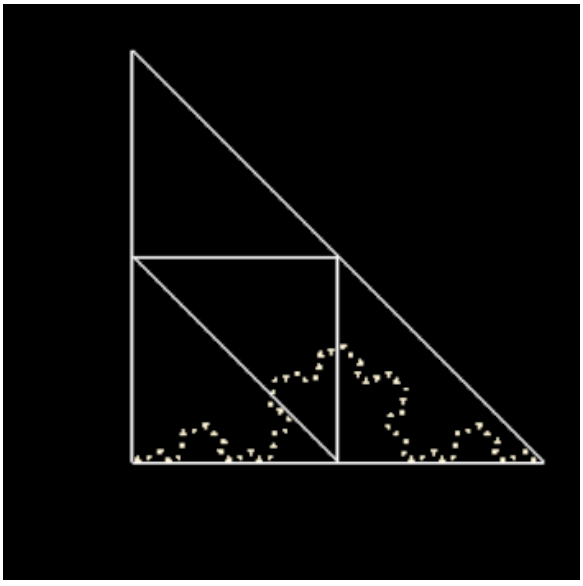


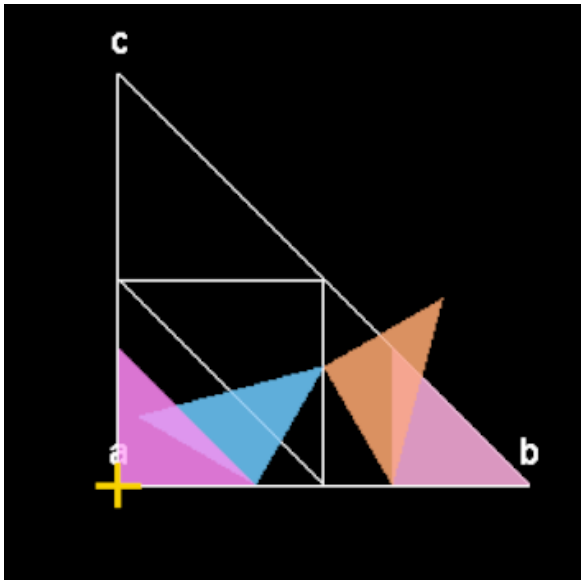


Snowflake: (two steps)

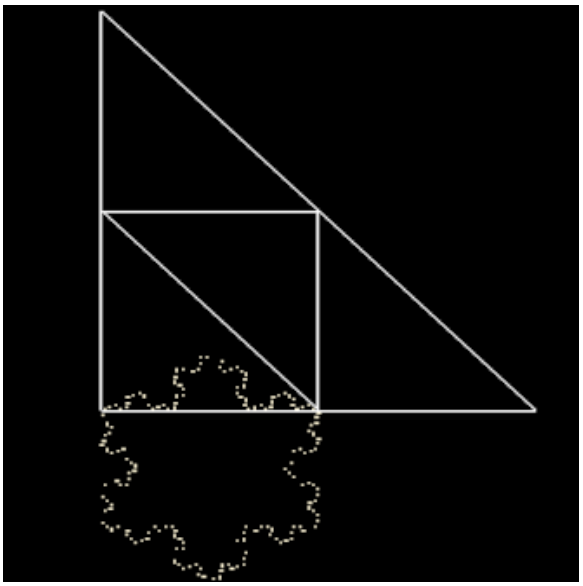
We think it's not possible to draw the Snowflake by a function. So for this one, Firstly, you will need to load the **KochSnowflakeSide.txt** from the folder "files", apply reasonable times. Secondly, you need to load the **Snow_rotate.txt** from the folder "files" and apply only once. Then you will see the corresponding image of Snowflake.

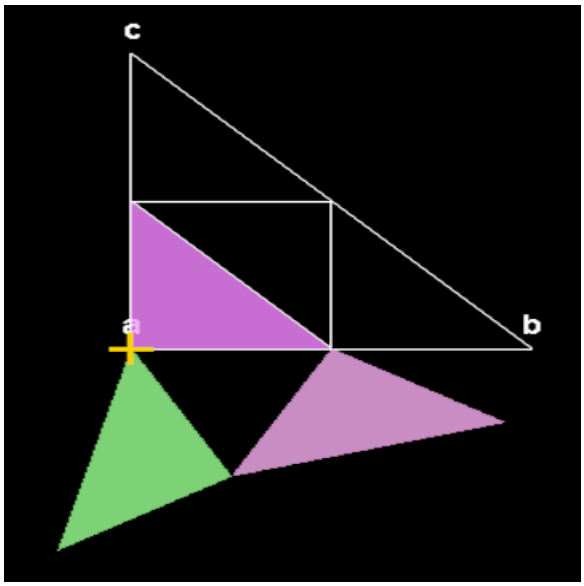
First choose KochSnowflakeSide and apply



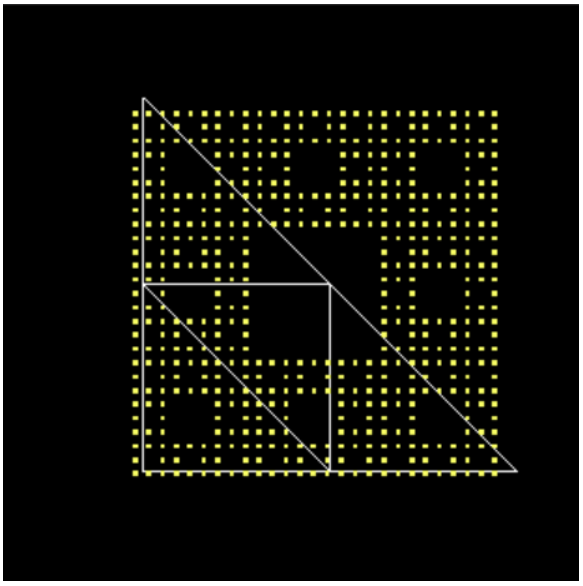


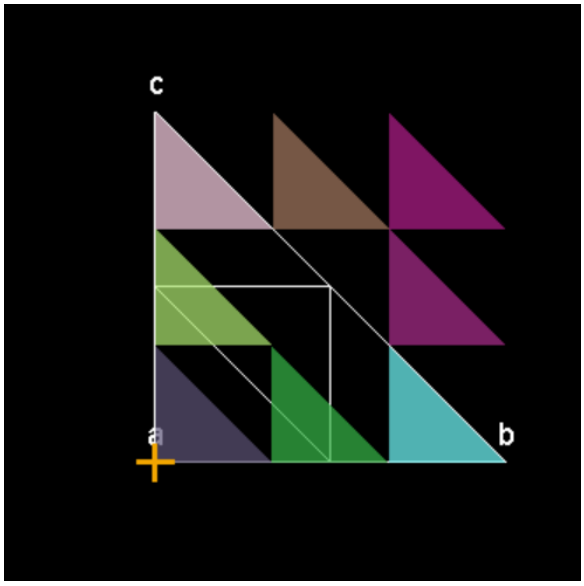
Then choose SnowRotate and apply



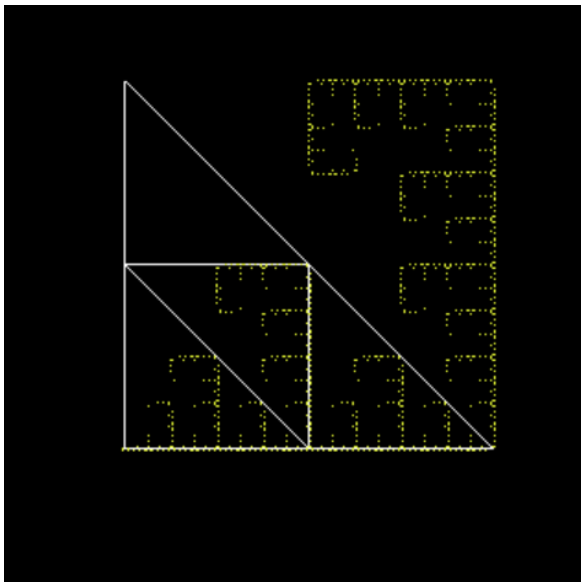


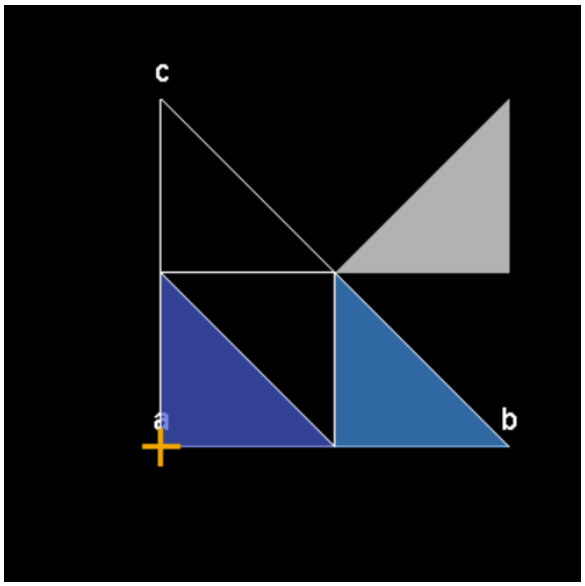
SquareGasket：Apply about three or four times to get the fractal



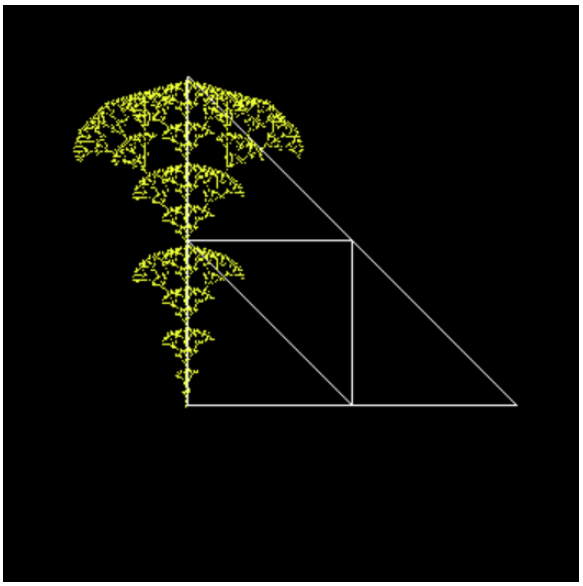


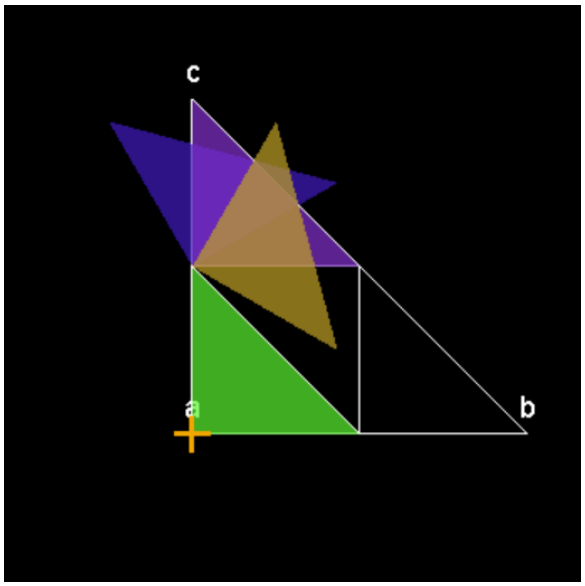
FractalHangman: Apply about six times to get the fractal





FractalBush: Apply about five or six times to get the fractal

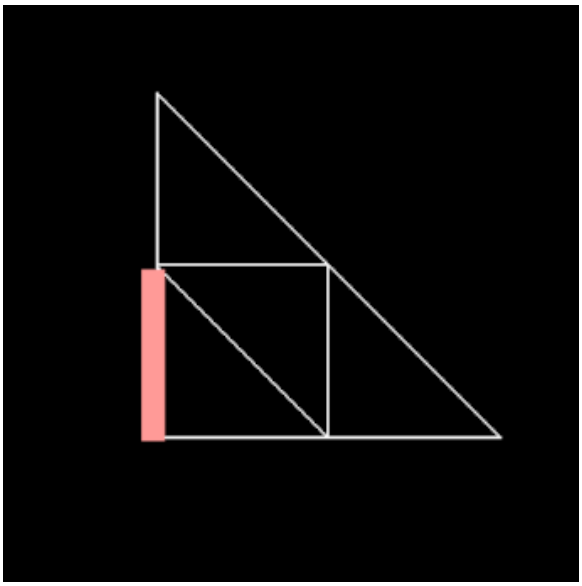


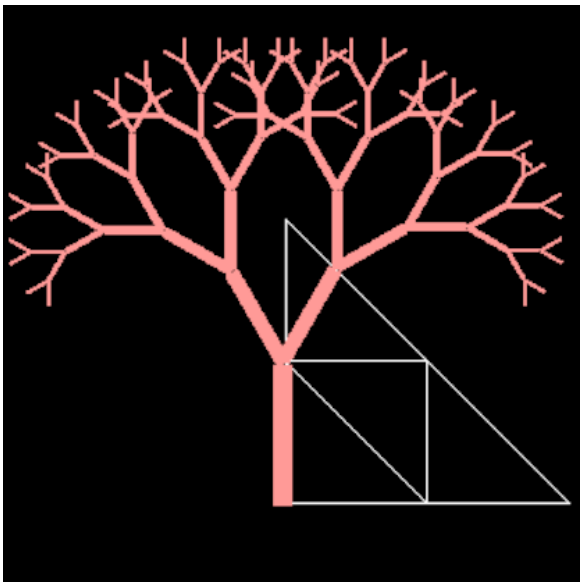


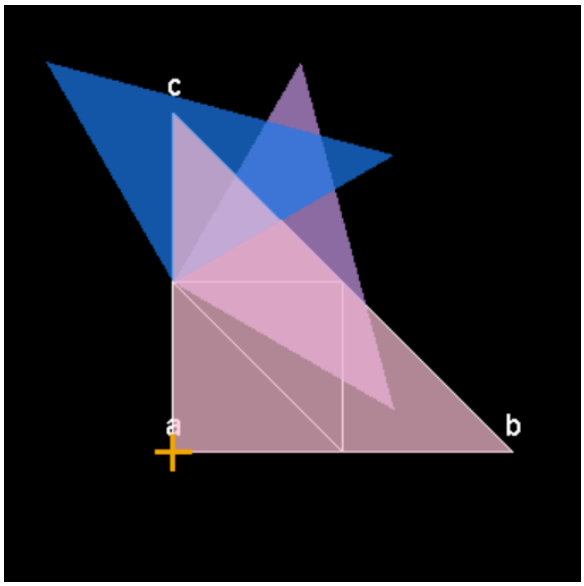
FractalTree: Apply about six or seven times to get the fractal

Because of the coordinates, in order to get the fractal, we need to stretch the rectangle in **IFSViewer** to align with the y axis of the base shape. Otherwise, it will get the wrong shape.

Basic Case:



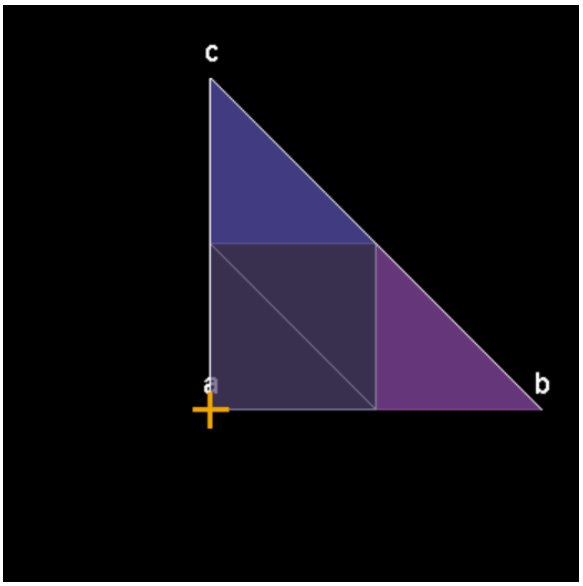




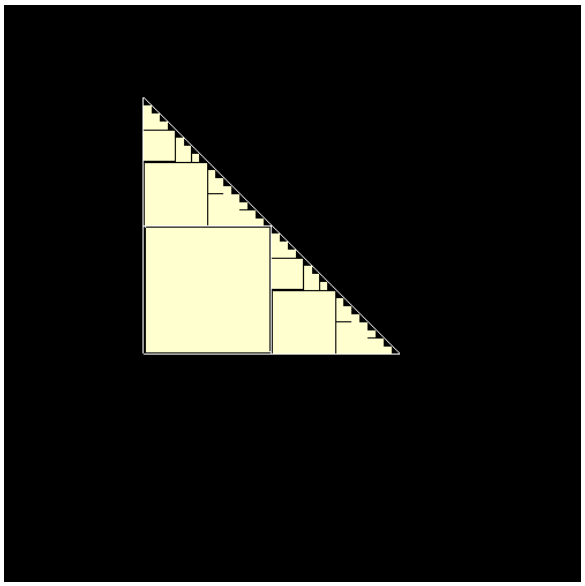
FractalStairs: Apply about six or seven times to get the fractal

Also, same as the **FractalTree**, to get the complete **FractalStairs**, the base shape should align to the y axis and x axis in the **IFSViewer**.

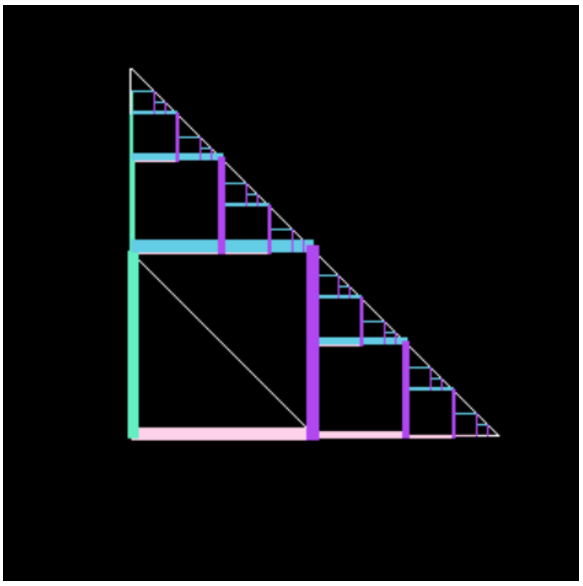
Basic case:

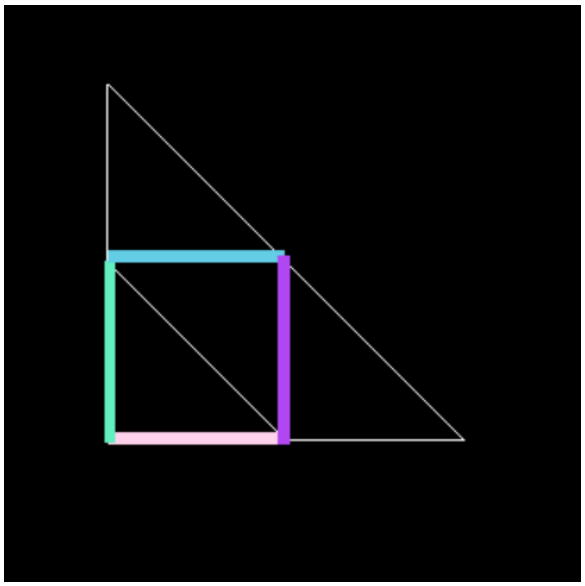


Result:

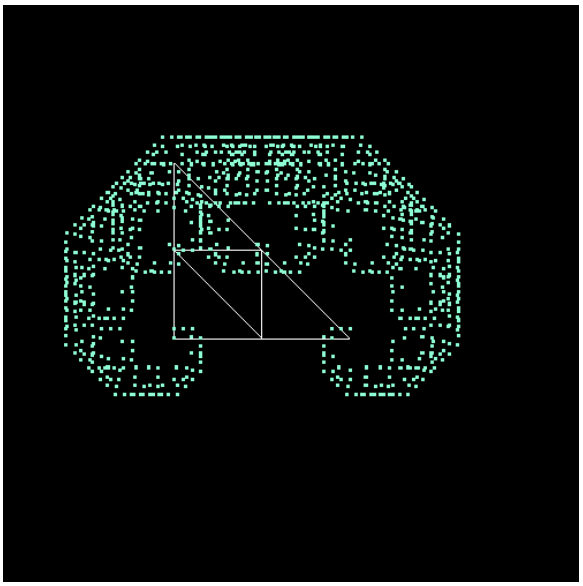


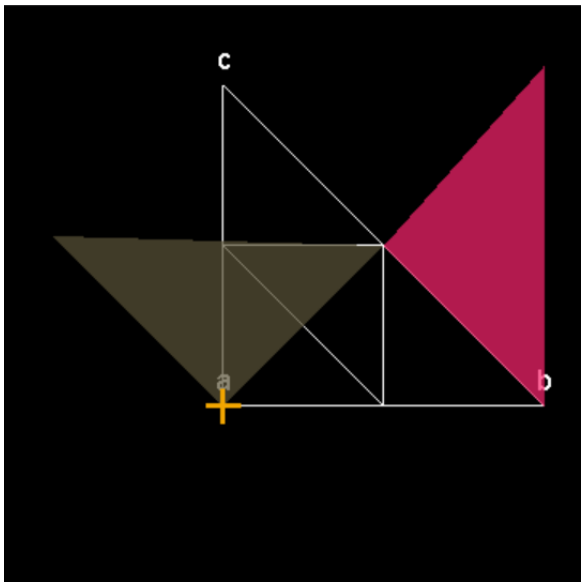
It's better to set the original shape as a square and its size equals to the square that is at the center of the grid triangle. To make the picture more obvious, we use two slender rectangles as the length and width of the square to show the details of the iterated function, so there are two pictures of **FractalStairs** in the file.





FractalC: Apply about eleven times to get the fractal

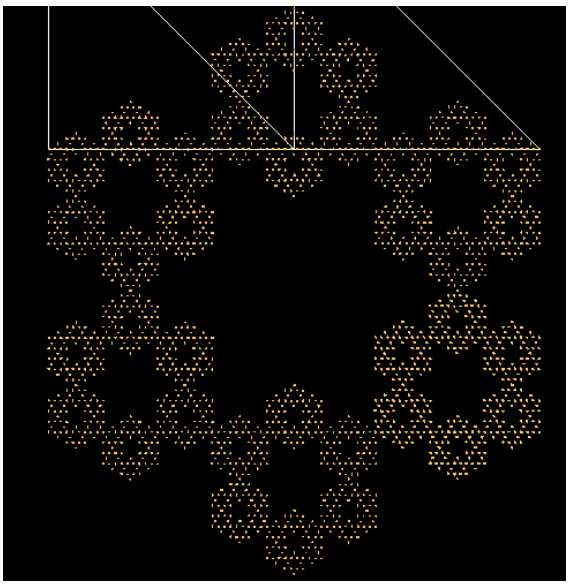


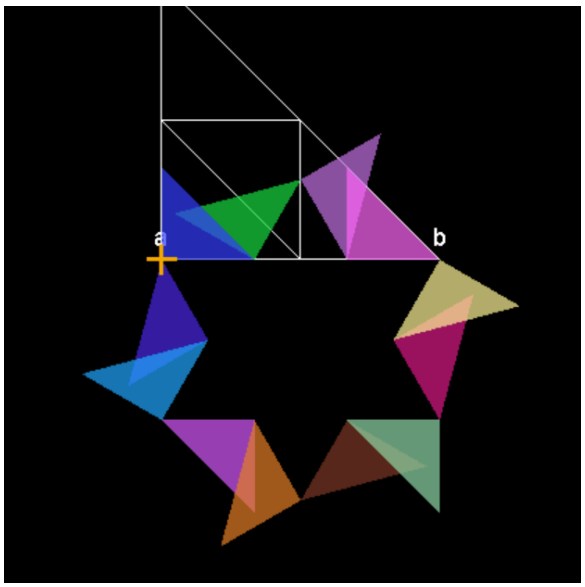


Four self-created fractals with pictures

In this part, we design four fractals and we will also show both the pictures in **Geometry Viewer** and **IFSViewer**. Details can be seen in **Design.txt** and file images. There are two special cases for design3_firework and design2_spiral, we defined the original shapes to make the firework more beautiful. They are also in the "images" folder, with the name of "**firework-origin.jpg**" and "**spiral_origin.jpg**". Also, you could use the shape you like to generate this iterated function.

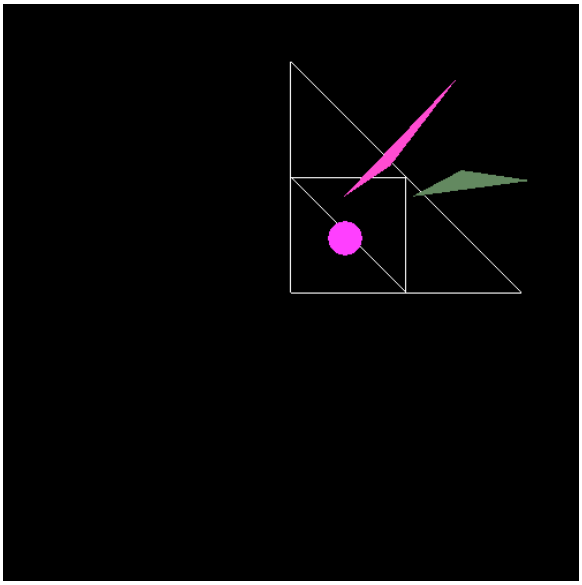
Design_1: (design_1_star_of_david) Apply four times to get the fractal.



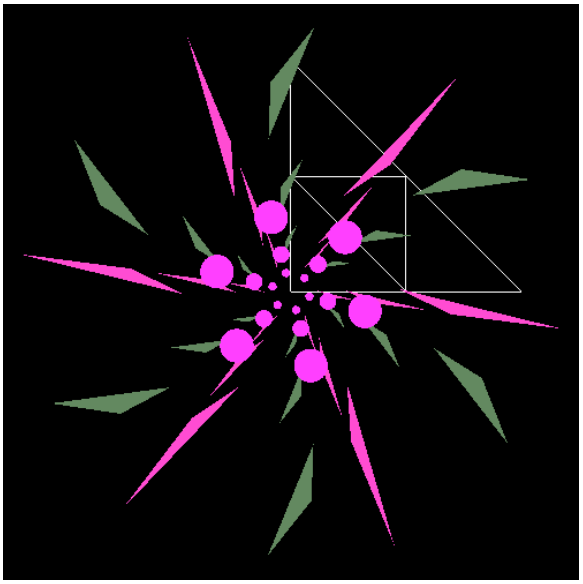


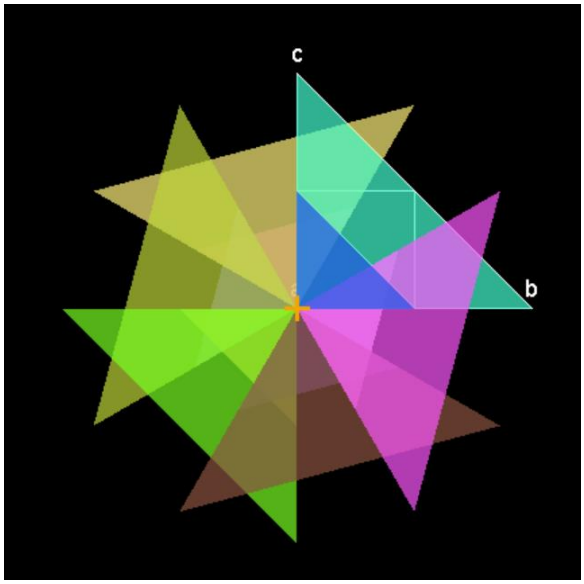
Design_2: design_2_spiral

We set the basic case as:



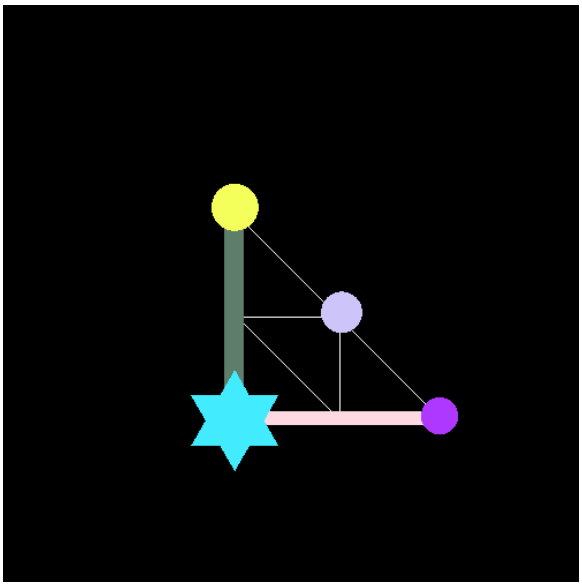
Final result: Apply four or five times to get the fractal



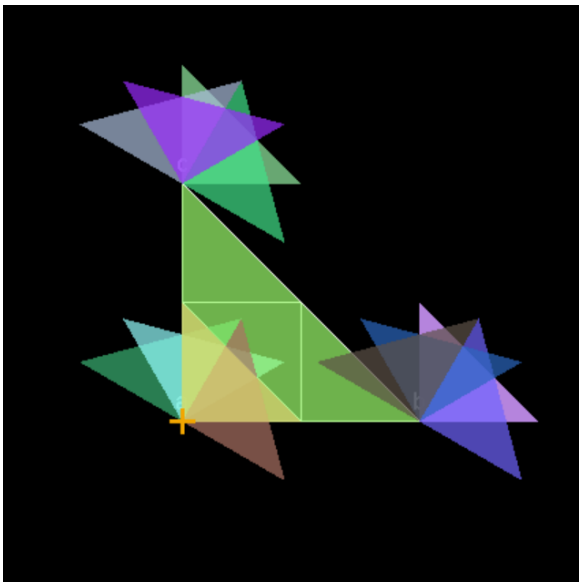


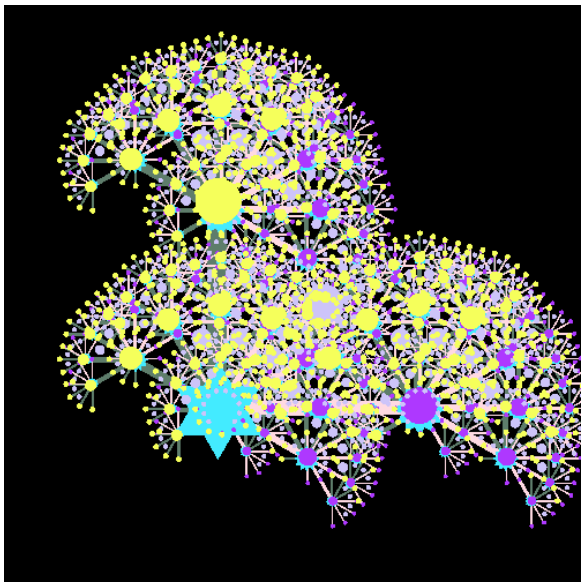
Design_3: design_3_firework

We set the basic case as:



Final result: Apply four or five times to get the fractal





Design_4:(design_4_stars)

The basic case is a star which we create with six vertices and apply four or five times to get the fractal.