,ITP20002-01 Discrete Mathematics Team 9
21300198 Jinsu Kim 21400646 Chae-eon Lim 21500344 Chansol Suh
21600234 Jeongsup Moon 21700525 Sua Lee
source code : https://github.com/PASTANERD/DM_/tree/master/PA02

Program Assignment 2 Report

Fractal1

Fractal1.1. Description of Recursion

The logic of this code can be described in 3 simple steps. First, find two coordinates that will evenly divide a line drawn between two given coordinates. Second, find another coordinate that will make equilateral triangle shape with two coordinates that are found from the line. Then recursively give 4 pairs of two adjacent coordinates as input of the second step. Third, connect all the coordinates if it reaches to the basis step.

To elaborate the recursive steps described above, a function called drawSnowFlake() is one that used recursively, while decreasing user-input depth by one, until it becomes zero. In the first part of this code, drawSnowFlake() is called three times; each works on each side of a triangle. This function contains two coordinates of each side of the triangle, one angle that shows the tilt of its side, and depth that shows how many times the recursion has to take place. Then, it produces three more coordinates that will make equilateral triangle in the middle of the side. As there are 5 coordinates, 4 sides can be obtained by linking adjacent pair of coordinates. Each pair will be passed down to four different drawSnowFlake() with different angles. When drawSnowFlake() reaches to the basis step, in which the depth equals to 1, lines are drawn to connect coordinate pairs via connectCoordinates().

Fractal1.2. Description on Parameters

In doDraw(), depth is a user-given number meaning number of recursive steps; if it is 1, the function only takes place once, and if it is 2, the function takes place twice recursively. drawSnowFlake() receives 5 parameters as follows: x1, y1, x2, y2, angle, and depth. 'x's and 'y's are coordinates of ends of a line. Using given coordinates, three pairs of coordinates are generated; Two coordinates are placed between one third and two third away from one coordinates. The last coordinate is depend on two generated coordinates. dist() function calculates the distance between newly generated two coordinates, and the result will be multiplied by sine and cosine to generate x-y coordinate of last one. Angle is the angular tilt of the line connected from (x1,y2) to (x2,y2). The angle used in sine and cosine is addition of the tilt of given and 30, because the degree changes as it goes into further recursion steps. As mentioned before, one drawSnowFlake() calls four more drawSnowFlake() recursively. Among 4 sides generated in drawSnowFlake(), two of them have angles unchanged, and the other two sides are respectively increased and decreased by 60. As recursion takes place, those angles should be recalculated when they are put into recursive functions. connectCoordinates() receives 10 inputs representing 5 coordinates. doClear() erase all lines drawn on screen, when "clear" button is clicked.

Fractal1.3. Images Generated with Different Parameters

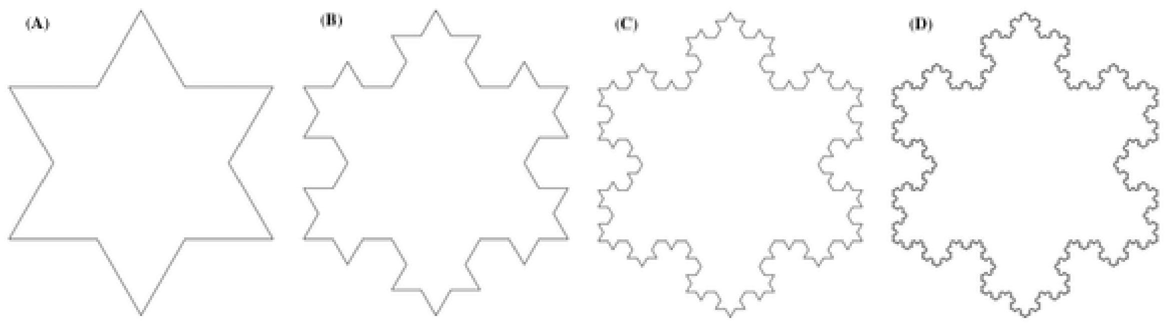

Figure 1. Image generated with different inputs. **(A)** is drawn with input 1, **(B)** is drawn with input 2, **(C)** is drawn with input 3, and **(D)** is drawn with input 5.

Fractal2

Fractal2.1. Description of recursion

The logic of this code is creating two different squares with different angular axis from a top side of a given square. Creating two squares occurs recursively as it pass down two top side coordinates of each newly created squares, to create two squares from each of recently made squares.

doDraw() draws an initial square and then calls drawFractal2() with two coordinates of top side of the square with angle of its tilt, which is zero, and depth, which stands for number of recursion it should take. Then, drawFractal2() creates 5 coordinates that represent part of coordinates of squares. After obtaining all coordinates required for drawing two squares, put four pairs of coordinates into drawSquare(), which connects adjacent coordinates with lines. Then recursively call two drawFractal2(), each containing two top side coordinates of newly made square, an angular tilt that is decreased by 30 or increased by 60, and depth that is decreased by 1. So each drawFractal2() will generate two drawFractal2() recursively.

Fractal2.2. Description on Parameters

doDraw() function assigns user-given input to variable "depth" to count down the number of recursions remained. drawFractal2() has 6 parameters: x1, y1, x2, y2, angle, and depth. The first four parameters are coordinates of the top side of square. Angle represent the tilt of the square. As in the picture drawn in figure 5, bottom square is not tilted at all, so it has angle of 0. Top left square is tilted to left in 30 degree angle, and is represented as -30 degree regarding to arrangements of coordinates. Top right square is tilted to right, so the angle is added by 60. This angular deviation occurs recursively and relatively to the angle of an ancestor square. Since the tilts of two squares are different, the length of sides of two squares are different, and they are measured by using 30-60-90 triangle property. Also new variables that represent coordinates of new squares are calculated using trigonometry.

Fractal2.3. Images Generated with Different Parameters

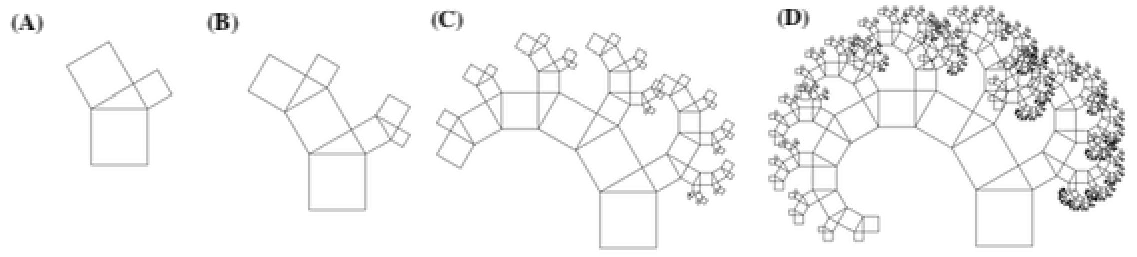

Figure 2. Image generated with different inputs. **(A)** is drawn with input 1, **(B)** is drawn with input 2, **(C)** is drawn with input 5, and **(D)** is drawn with input 9.

Fractal3

We made an interesting and new fractal by ourselves. We decide to call this fractal "octopus" because this fractal look like octopus. The logic of this code can be divided into two parts: Head, Leg. 'Head' part is creating a pentagon recursively which is inside a circle. 'Leg' part is a variation of fractal 'Tree' so that it is represented by a simple line with some attached circles(the number of circles are the same as the value of 'depth'). doDraw() also consists of two main functions. For 'Head', doDraw() draws an initial circle and then calls drawfractal() with 10 coordinates of vertices of initial pentagon. For 'Leg', doDraw() calls drawFractal2() which draws initial lines in different angles with circles attached. drawFractal2() creates 2 coordinates that represents the location of the next line and put two coordinates into drawLine(), which connects coordinates with lines. Now let us describe this fractal divided into two parts in details.

Fractal3.1. Description of recursion

Description of head (pentagon) recursion

The logic of pentagon code begins at function called doDraw(). First, we find five coordinates ((x1, y1).... (x5,y5)) above a circle having a certain radius and drawfractal() is called for drawpentagon() to make a regular pentagon connected from (x1,y1) to (x5,y5) recursively. Drawfractal() have two functions "drawpentagon() and drawfractal()". Drawpentagon() make a regular pentagon, using five coordinate and subsequently drawfractal() is recalled to make slightly rotated regular pentagon in previous pentagon. The method to make slightly rotated regular pentagon will be described specifically in parameter part. In this way, while decreasing user-input depth by one, until it becomes zero, drawfractal() is called recursively. Namely, pentagons that is equal to the number of user-input depth, are drawn internally.

Description of leg (spiral) recursion

The drawFractal2() calculate new point (x2, y2), by rotating 'angle' degree around the starting point (x1, y1) and move it a specified distance multiplied depth by constant value 3.0. Then draw a line through (x1, y1) to (x2, y2), and a circle perpendicular to the center of the line. The radius of circle is the distance distance from (x1, y1) to (x2, y2) multiplied by the constant value 0.04. After this step, call the drawFractcal2(), calculate the new point again, draw an another line, and draw a circle that is tangent to the line. Repeat this process to complete a leg of the rolled shape.

Fractal3.2. Description on Parameters

Description of parameter for drawing head part

Dodraw() function assigns user-given input to variable "depth" to count down the number of recursions remained. Drawfractal() has 11 parameters (x1, y1, x2, y2, x3, y3, x4, y4, x5, y5, depth). First 10 parameters are five coordinates ((x1, y1), (x2, y2),...(x5,y5)) above a circle and the last parameter is depth mentioned previously. After drawpentagon() draw regular pentagon, drawfractal() find 5 new coordinates ((x_1, y_1).... (x_5, y_5)) dividing one side of a pentagon by 1:3 on each pentagon one side to make slightly rotated regular pentagon, using internally dividing point formula. (ex: var x_1 = (3*x1+ x2)/4;, var y_1 =(3*y1+y2)/4;). While drawfractal() is called recursively ,until depth becomes zero, new variables (x_1, y_1, x_2, y_2, x_3, y_3, x_4, y_4, x_5, y_5) that represent these coordinates is assigned to previous variables (x1,y1,....x5,y5) continually and rotated regular pentagon is drawn internally, using new coordinates. Depth_p that is one of parameters of drawpentagon() is equal to depth. Using depth_p and syntax "rgb", whenever new pentagon is drawn, new pentagon is filled with each different color.

Description of parameter for drawing leg part

To make leg part, call drawFractal2() recursively with parameter x1, y1 which is coordinates of starting point to draw, angle to give the degree of rotation, depth given by user and color consisted of rgb values. The parameter depth is decreased by 1 in recursive step and angle becomes angle-20, which means draw new line by rotating 20 degree. And calculated point (x2, y2) become parameter as starting point.

Fractal3.3. Images Generated with Different Parameters

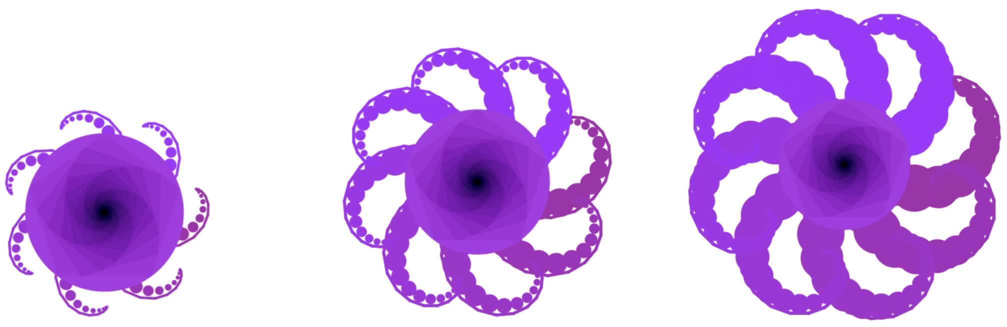

Figure 3. Image generated with different inputs. **(A)** is drawn with input 12, **(B)** is drawn with input 16, and **(C)** is drawn with input 20.

Triomino

Triomino.1. Description of recursion

To construct a recursive algorithm, first, we need basis step. In this case, when n is equal to 1, the 2×2 grid is filled with one of right triominos (Figure 4). Second, there must be recursive step; if $n > 1$, the grid can be divided into four sub-grids which have half-length of its parents.

However, there is one tricky part needed to be proved. When we compose four $2^{n-1} \times 2^{n-1}$ grids into one, $2^n \times 2^n$ grids will be filled with triominos. However, the center of the grid has unfilled 2×2 grid. So we decided to make another approach to divide the grid into five sub-grids, in which four sub-grids are same, and another is $2^{n-1} \times 2^{n-1}$ sub-grid at the center of the grid.

It may look like complex. However, complexity of recursive algorithm is determined by size of the recursive level as well as its own complexity. In this algorithm, there are only arithmetic calculations. So, if the depth is not too big, doing another recursive computation is not that much matter. In other words, making another recursive for center of the grid which have same depth with others adds only constant time complexity. In the reason of that, we dare make function divide center sub-grid even though it draws only a few triominos again which are already drawn.

To simplify recursive steps, there are two cases the function must to follow is like this.

i) If the grid is 2×2
- Function cannot divide the grid.
- Draw triomino

ii) If the grid is $2^n \times 2^n$
- Function can divide the grid
    - Divide grid into four sub-grids.
    - If it is initializing division, divide grid into five sub-grids

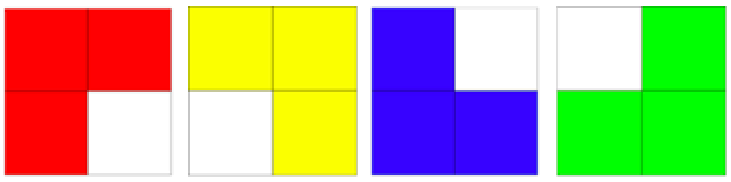
Figure 4. Four different unit triominos

Triomino.2. Description on Parameters

To describe more specifically, we made four different functions which divide grid and fill each of 2×2 grid with each of their way and one which initialize tiling grid with triominos. Former function is findTriomino(). There are four functions for each triominos which is named as findTriominoA(), findTriominoB() and so on. Latter one is initTriomino(). both functions calculate next_depth, half_length, and (x2, y2), (x3, y3) when it is called. next_depth is depth-1 which means dimension level. half_length is half of the grid length. two coordinates means origin coordinates for next dimension. x2 is for right side grids, and y2 is downside grids. so we can divide grid into four sub-grids which have its origin coordinate at (x1, y1), (x2, y1), (x1, y2), (x2, y2) (x1 and y1 is given parameter.) Also, (x3, y3) is for sub-grid at the center of the grid.

initTriomino()	- Initialize tiling triominos and recursively divide grid into five which are left upside, left downside, right upside, right downside, and center.
findTriominoA()	- If depth is 1,it draws the red triomino in the Figure 4. - If depth isn't 1, it recursively calls four triominos for left upside grid, right upside grid, left downside grid, and the center of the grid.
findTriominoB()	- If depth is 1, it draws the yellow triomino in the Figure 4.

	- If depth isn’t 1, it recursively calls four triominos for left upside grid, right upside grid, right downside grid, and the center of the grid.
findTriominoC()	- If depth is 1, it draws the blue triomino in the Figure 4. - If depth isn’t 1, it recursively calls four triominos for left upside grid, left downside grid, right downside grid, and the center of the grid.
findTriominoD()	- If depth is 1, it draws the green triomino in the Figure 4. - If depth isn’t 1, it recursively calls four triominos for right upside grid, left downside grid, right downside grid, and the center of the grid.
doDraw()	- Get input value from user which is determine the depth of the grid.
doClean()	- Make canvas clear

When function ‘initTriomino()’ or ‘findTriomino()’ are called, they get four parameters which are ‘x1’, ‘y1’, ‘length’, ‘depth.’

x1 :	x value for coordinates of origin point of grid which is left upside point
y1 :	y value for coordinates of origin point of grid which is left upside point
length :	the length of the grid, it is given the half of the length
depth :	it is of what 2 to the power, it is given subtracted by 1

By passing half of its length and decreased depth, we can guarantee this algorithm is working recursively

Triomino.3. Images Generated with Different Parameters

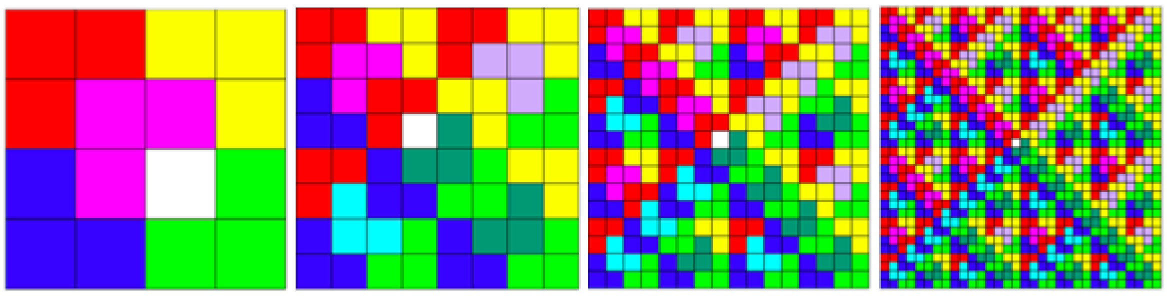

Figure 5. Four tiling triomino solutions with four different inputs.