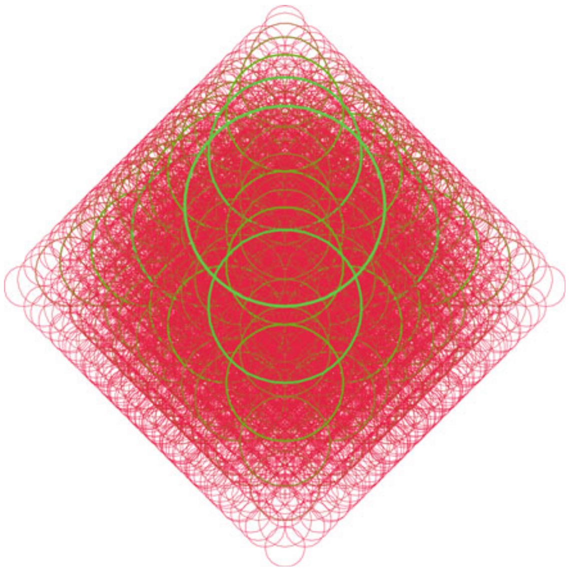
# Chapter 4
# Computer Generated Art

## 4.1 Creating Computer Art

Computer Art is defined as any art that uses a computer. However, in the context of this book we mean it to be art that is generated by a computer or more specifically a computer program. The following example, illustrates how in a very few lines of Python code, using the Turtle graphics library, you can create images that might be considered to be computer art.

The following image is generated by a recursive function that draws a circle at a given x, y location of a specified size. This function recursively calls itself by modifying the parameters so that smaller and smaller circles are drawn at different locations until the size of the circles goes below 20 pixels.



© Springer Nature Switzerland AG 2019
J. Hunt, *Advanced Guide to Python 3 Programming*,
Undergraduate Topics in Computer Science,
https://doi.org/10.1007/978-3-030-25943-3_4

The program used to generate this picture is given below for reference:

```
import turtle

WIDTH = 640
HEIGHT = 360

def setup_window():
    # Set up the window
    turtle.title('Circles in My Mind')
    turtle.setup(WIDTH, HEIGHT, 0, 0)

    turtle.colormode(255)  # Indicates RGB numbers will be in
the range 0 to 255
    turtle.hideturtle()
    # Batch drawing to the screen for faster rendering
    turtle.tracer(2000)

    # Speed up drawing process
    turtle.speed(10)
    turtle.penup()

def draw_circle(x, y, radius, red=50, green=255, blue=10,
width=7):
    """ Draw a circle at a specific x, y location.

    Then draw four smaller circles recursively"""
    colour = (red, green, blue)

    # Recursively drawn smaller circles
    if radius > 50:
        # Calculate colours and line width for smaller circles
        if red < 216:
            red = red + 33
            green = green - 42
            blue = blue + 10
            width -= 1
        else:
            red = 0
            green = 255
        # Calculate the radius for the smaller circles
        new_radius = int(radius / 1.3)
        # Drawn four circles
        draw_circle(int(x + new_radius), y, new_radius, red,
green, blue, width)
        draw_circle(x - new_radius, y, new_radius, red, green,
blue, width)
        draw_circle(x, int(y + new_radius), new_radius, red,
green, blue, width)
        draw_circle(x, int(y - new_radius), new_radius, red,
green, blue, width)

    # Draw the original circle
    turtle.goto(x, y)
    turtle.color(colour)
    turtle.width(width)
    turtle.pendown()
    turtle.circle(radius)
    turtle.penup()

# Run the program
print('Starting')
setup_window()
draw_circle(25, -100, 200)

# Ensure that all the drawing is rendered
turtle.update()
print('Done')
turtle.done()
```

There are a few points to note about this program. It uses recursion to draw the circles with smaller and smaller circles being drawn until the radius of the circles falls below a certain threshold (the termination point).

It also uses the `turtle.tracer()` function to speed up drawing the picture as 2000 changes will be buffered before the screen is updated.

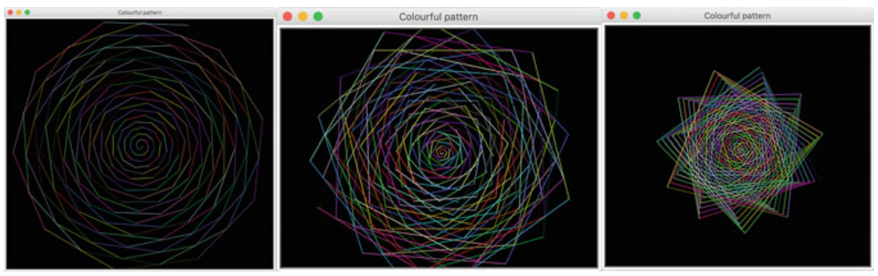
Finally, the colours used for the circles are changed at each level of recession; a very simple approach is used so that the Red, Green and Blue codes are changed resulting in different colour circles. Also a line width is used to reduce the size of the circle outline to add more interest to the image.

## 4.2 A Computer Art Generator

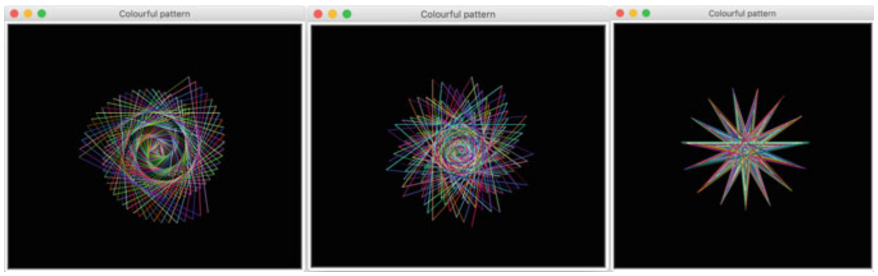
As another example of how you can use Turtle graphics to create computer art, the following program randomly generates RGB colours to use for the lines being drawn which gives the pictures more interest. It also allows the user to input an angle to use when changing the direction in which the line is drawn. As the drawing happens within a loop even this simple change to the angle used to draw the lines can generate very different pictures.

```python
# Lets play with some colours
import turtle
from random import randint

def get_input_angle():
    """ Obtain input from user and convert to an int"""
    message = 'Please provide an angle:'
    value_as_string = input(message)
    while not value_as_string.isnumeric():
        print('The input must be an integer!')
        value_as_string = input(message)
    return int(value_as_string)

def generate_random_colour():
    """Generates an R,G,B values randomly in range
    0 to 255 """
    r = randint(0, 255)
    g = randint(0, 255)
    b = randint(0, 255)
    return r, g, b

print('Set up Screen')
turtle.title('Colourful pattern')
turtle.setup(640, 600)
turtle.hideturtle()
turtle.bgcolor('black')  # Set the background colour of the
screen
turtle.colormode(255)  # Indicates RGB numbers will be in the
range 0 to 255
turtle.speed(10)

angle = get_input_angle()

print('Start the drawing')
for i in range(0, 200):
    turtle.color(generate_random_colour())
    turtle.forward(i)
    turtle.right(angle)

print('Done')
turtle.done()
```

Some sample images generated from this program are given below. The left most picture is generated by inputting an angle of 38 degrees, the picture on the right uses an angle of 68 degrees and the bottom picture an angle of 98 degrees.



The following pictures below use angles of 118, 138 and 168 degrees respectively.



What is interesting about these images is how different each is; even though they use exactly the same program. This illustrates how algorithmic or computer generated art can be as subtle and flexible as any other art form. It also illustrates that even with such a process it is still up to the human to determine which image (if any) is the most aesthetically pleasing.

## 4.3 Fractals in Python

Within the arena of Computer Art fractals are a very well known art form. Factrals are recurring patterns that are calculated either using an iterative approach (such as a for loop) or a recursive approach (when a function calls itself but with modified parameters). One of the really interesting features of fractals is that they exhibit the same pattern (or nearly the same pattern) at successive levels of granularity. That is, if you magnified a fractal image you would find that the same pattern is being repeated at successively smaller and smaller magnifications. This is known as expanding symmetry or unfolding symmetry; if this replication is exactly the same at every scale, then it is called affine self-similar.

Fractals have their roots in the world of mathematics starting in the 17th century, with the term fractal being coined in the 20th century by mathematical Benoit Mandelbrot in 1975. One often cited description that Mandelbrot published to describe geometric fractals is

> a rough or fragmented geometric shape that can be split into parts, each of which is (at least approximately) a reduced-size copy of the whole.

For more information see Mandelbrot, Benoît B. (1983). The fractal geometry of nature. Macmillan. ISBN (978-0-7167-1186-5).

Since the later part of the 20th century fractals have been a commonly used way of creating computer art.

One example of a fractal often used in computer art is the Koch snowflake, while another is the Mandelbrot set. Both of these are used in this chapter as examples to illustrate how Python and the Turtle graphics library can be used to create fractal based art.

### *4.3.1 The Koch Snowflake*

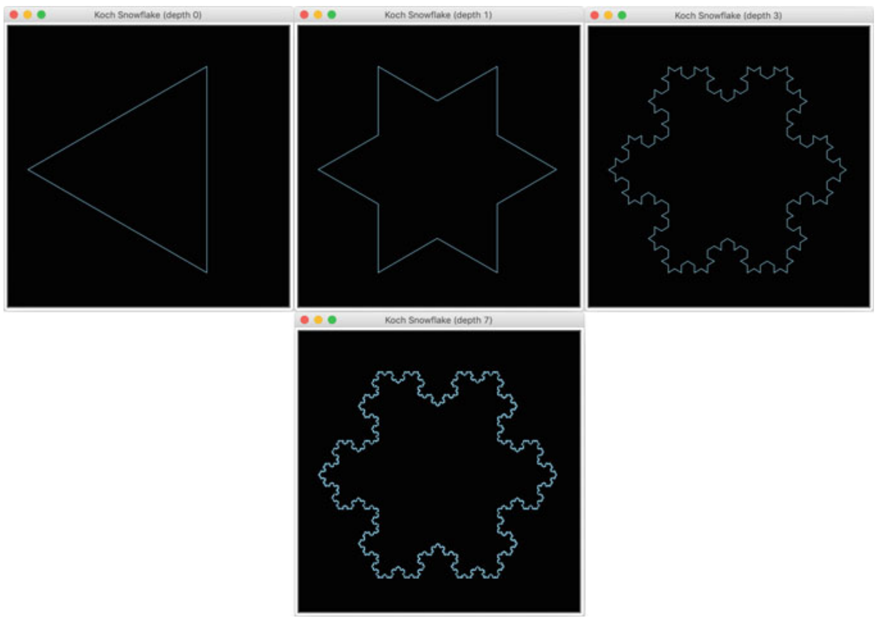
The Koch snowflake is a fractal that begins with equilateral triangle and then replaces the middle third of every line segment with a pair of line segments that form an equilateral bump. This replacement can be performed to any depth generating finer and finer grained (smaller and smaller) triangles until the overall shape resembles a snow flake.

The following program can be used to generate a Koch snowflake with different levels of recursion. The larger the number of levels of recursion the more times each line segment is dissected.

```
import turtle

# Set up Constants
ANGLES = [60, -120, 60, 0]
SIZE_OF_SNOWFLAKE = 300


def get_input_depth():
    """ Obtain input from user and convert to an int"""
    message = 'Please provide the depth (0 or a positive
interger):'
    value_as_string = input(message)
    while not value_as_string.isnumeric():
        print('The input must be an integer!')
        value_as_string = input(message)
    return int(value_as_string)


def setup_screen(title, background='white', screen_size_x=640,
screen_size_y=320, tracer_size=800):
     print('Set up Screen')


     turtle.title(title)
     turtle.setup(screen_size_x, screen_size_y)
     turtle.hideturtle()
     turtle.penup()
     turtle.backward(240)
     # Batch drawing to the screen for faster rendering
     turtle.tracer(tracer_size)
     turtle.bgcolor(background)  # Set the background colour of
  the screen
```

```
def draw_koch(size, depth):
    if depth > 0:
        for angle in ANGLES:
            draw_koch(size / 3, depth - 1)
            turtle.left(angle)
    else:
        turtle.forward(size)

depth = get_input_depth()

setup_screen('Koch Snowflake (depth ' + str(depth) + ')',
             background='black',
             screen_size_x=420, screen_size_y=420)
# Set foreground colours
turtle.color('sky blue')

# Ensure snowflake is centred
turtle.penup()
turtle.setposition(-180,0)
turtle.left(30)
turtle.pendown()

# Draw three sides of snowflake
for _ in range(3):
    draw_koch(SIZE_OF_SNOWFLAKE, depth)
    turtle.right(120)

# Ensure that all the drawing is rendered
turtle.update()
print('Done')
turtle.done()
```

Several different runs of the program are shown below with the depth set at 0, 1, 3 and 7.



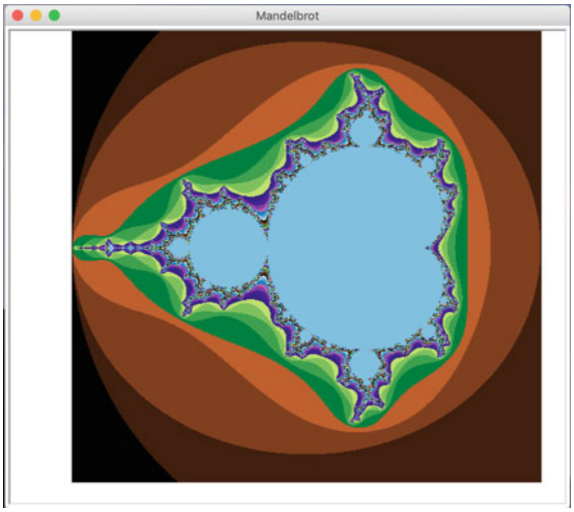
Running the simple `draw_koch()` function with different depths makes it easy to see the way in which each side of a triangle can be dissected into a further triangle like shape. This can be repeated to multiple depths giving a more detailed structured in which the same shape is repeated again and again.

### 4.3.2 *Mandelbrot Set*

Probably one of the most famous fractal images is based on the Mandelbrot set. The Mandelbrot set is the set of complex numbers c for which the function `z * z + c` does not diverge when iterated from z = 0 for which the sequence of functions (func(0), func(func(0)) etc.) remains bounded by an absolute value. The definition of the Mandelbrot set and its name is down to the French mathematician Adrien Douady, who named it as a tribute to the mathematician Benoit Mandelbrot.

Mandelbrot set images may be created by sampling the complex numbers and testing, for each sample point *c*, whether the sequence func(0), func(func(0)) etc. ranges to infinity (in practice this means that a test is made to see if it leaves some predetermined bounded neighbourhood of 0 after a predetermined number of iterations). Treating the real and imaginary parts of *c* as image coordinates on the complex plane, pixels may then be coloured according to how soon the sequence crosses an arbitrarily chosen threshold, with a special color (usually black) used for the values of *c* for which the sequence has not crossed the threshold after the predetermined number of iterations (this is necessary to clearly distinguish the Mandelbrot set image from the image of its complement).

The following image was generated for the Mandelbrot set using Python and Turtle graphics.



The program used to generate this image is given below:

```
for y in range(IMAGE_SIZE_Y):
    zy = y * (MAX_Y - MIN_Y) / (IMAGE_SIZE_Y - 1) + MIN_Y
    for x in range(IMAGE_SIZE_X):
        zx = x * (MAX_X - MIN_X) / (IMAGE_SIZE_Y - 1) + MIN_X
        z = zx + zy * 1j
        c = z
        for i in range(MAX_ITERATIONS):
            if abs(z) > 2.0:
                break
            z = z * z + c
        turtle.color((i % 4 * 64, i % 8 * 32, i % 16 * 16))
        turtle.setposition(x - SCREEN_OFFSET_X,
                           y - SCREEN_OFFSET_Y)
        turtle.pendown()
        turtle.dot(1)
        turtle.penup()
```

## 4.4 Online Resources

The following provide further reading material:

- https://en.wikipedia.org/wiki/Fractal For the Wikipedia page on Fractals.
- https://en.wikipedia.org/wiki/Koch_snowflake The Wikipedia page on the Koch snowflake.
- https://en.wikipedia.org/wiki/Mandelbrot_set Wikipedia page on the Mandelbrot set.

## 4.5 Exercises

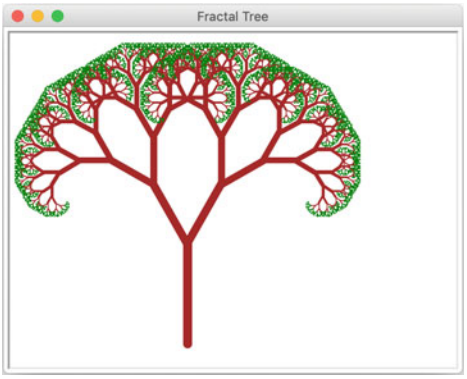
The aim of this exercise is to create a Fractal Tree.

A Fractal Tree is a tree in which the overall structure is replicated at finer and finer levels through the tree until a set of leaf elements are reached.

To draw the fractal tree you will need to:

- Draw the trunk.
- At the end of the trunk, split the trunk in two with the left trunk and the right trunk being 30° left/right of the original trunk. For aesthetic purposes the trunk may become thinner each time it is split. The trunk may be drawn in a particular colour such as brown.
- Continue this until a maximum number of splits have occurred (or the trunk size reduces to a particular minimum). You have now reached the leaves (you may draw the leaves in a different colour e.g. green).

An example of a Fractal Tree is given below:


Fractal Tree