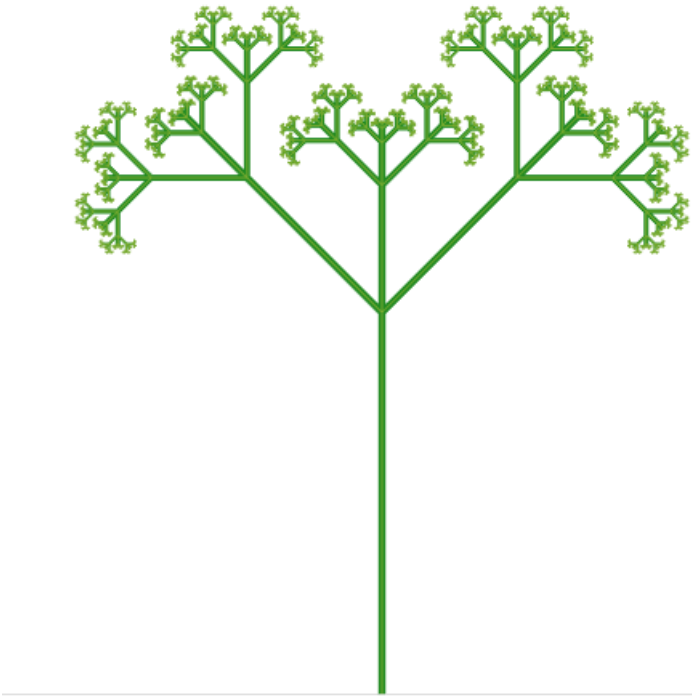
Methods for Generating Fractals



Lecture Notes for High School Students

Dan Alexandru Nedelescu

danedelescu @wpi.edu

Massachusetts Academy of Math and Science
Worcester, Massachussets 01609

- Summer 2018 -

This paper is intended to serve as an introduction to the world of fractals and programming techniques that generate them. These strangely beautiful shapes are entering into our lives more and more often. Whether they are seen in our phone's antennas, used by scientists to study a heartbeat, or even used to model an entire forest from a tree, fractals are becoming essential to understanding fundamental connections between nature and the world of science. My hope is that you will appreciate the elegant simplicity that makes fractals so fascinating.

Sincerely,
Dan Nedelescu

Table of Contents

Section 1: **An Introduction to Fractals**

When someone says draw a shape, you would generally draw something with a finite number of lines. For example, a hexagon is six straight lines and a circle is one curved line.

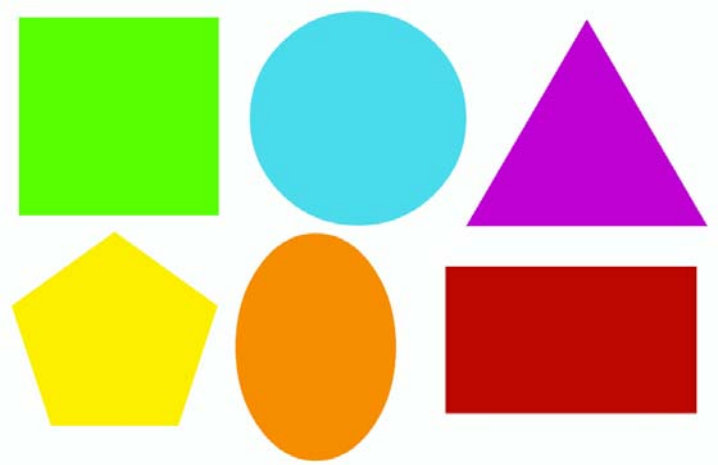


Figure 1.1 Euclidean geometric shapes
(image source: http://free-hd-wall-papers.com/single/shapes-1.html)

Fractals, however, do not fall under this definition. They cannot be defined using typical Euclidean geometry and often seem to not make any logical sense.

Take for instance the Koch Snowflake:

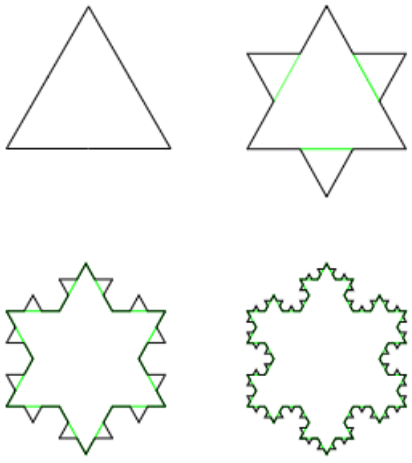


Figure 1.2 Koch Snowflake, first described by Heldge von Koch in 1904
(image from: https://en.wikipedia.org/wiki/Koch_snowflake)

At first glance, this shape seems normal. However, as you keep generating this object, you discover that it contains an *infinitely large perimeter but only a finite area*.

And as you zoom in and zoom out you see that the object looks the same, which is an important property of fractals called *self-similarity*.

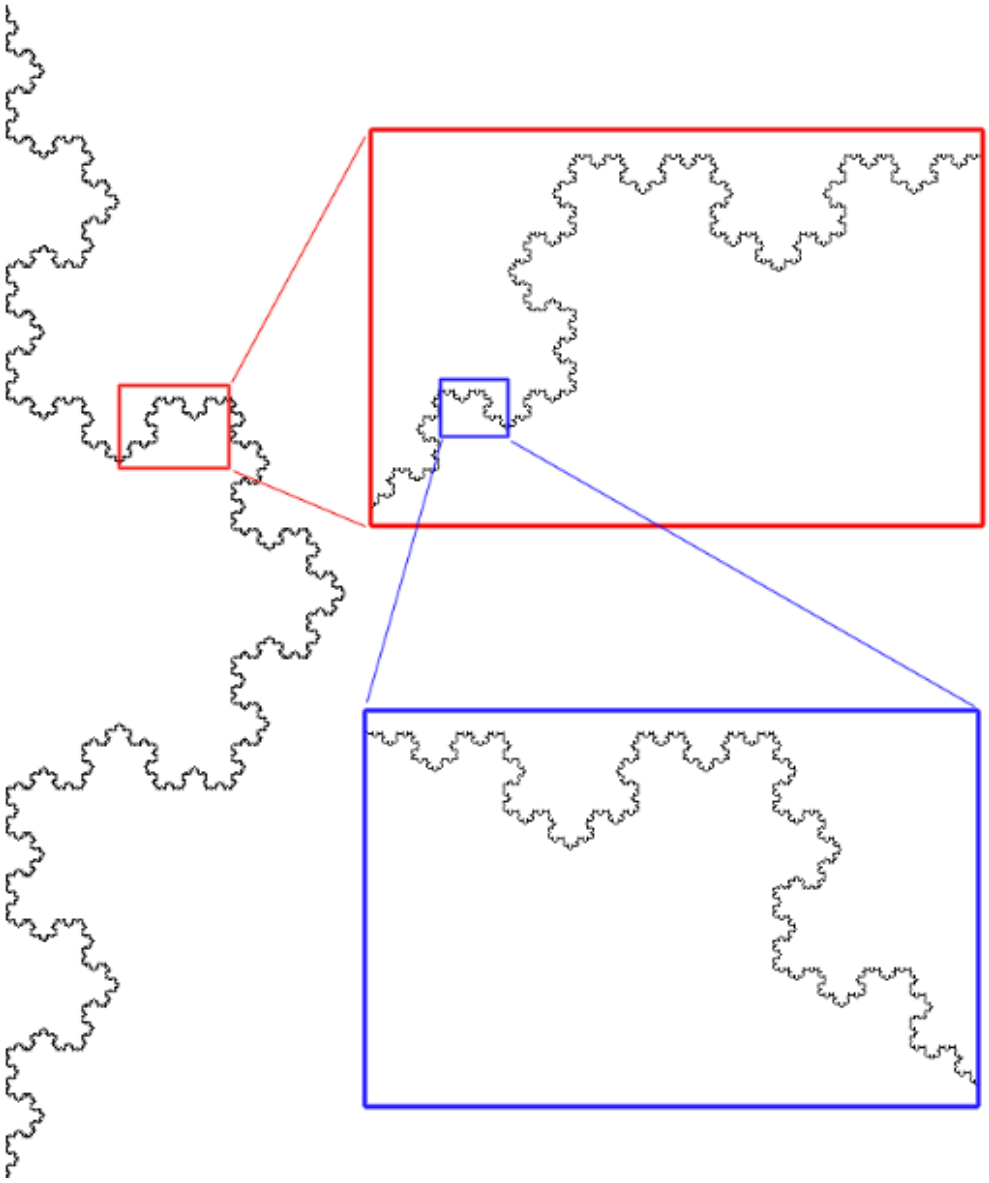


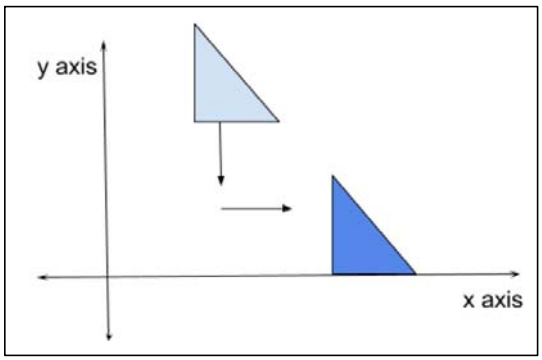
Figure 1.3 Self-similarity in Koch Snowflake
(image source: http://paulbourke.net/fractals/fracdim/)

Taking these properties of fractals forward, let's see how to generate them.

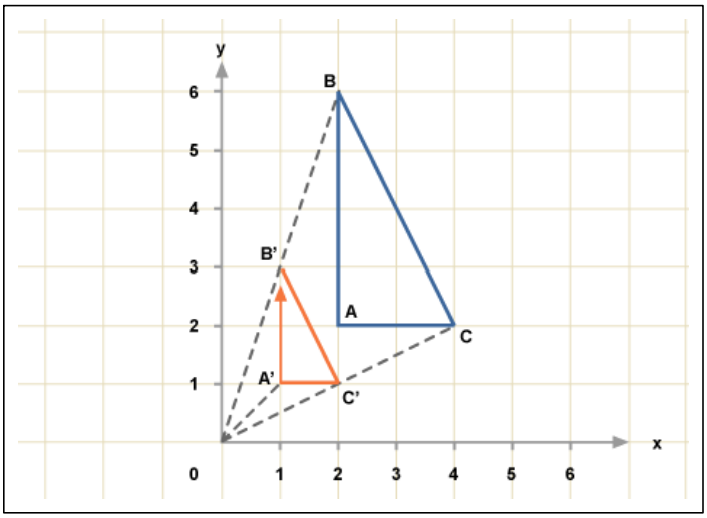
Section 2: **Transformations Used to Generate Fractals**

To generate fractals we need 4 different transformations: translation, scaling, reflection and rotation.

Translation



Scaling



Reflection

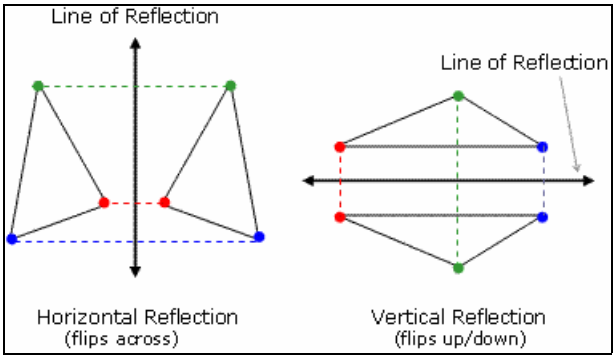
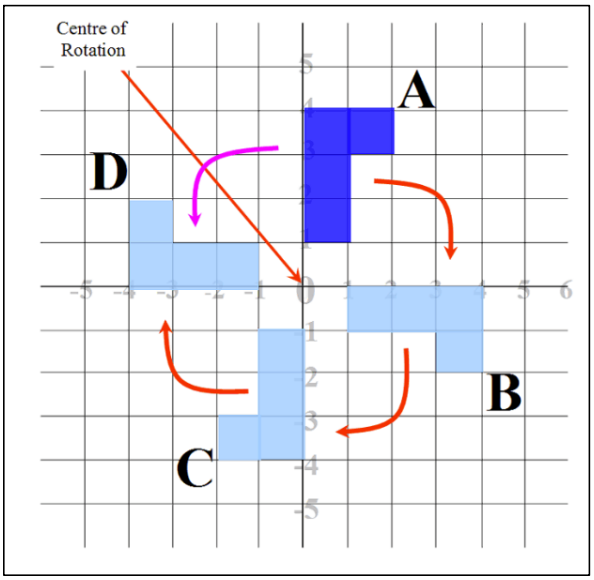


Image sources: https://www.emathematics.net/transformations.php?def=enlargements
http://mathstutor.net.au/maths11/specialist-maths-3/455-2/

Rotation

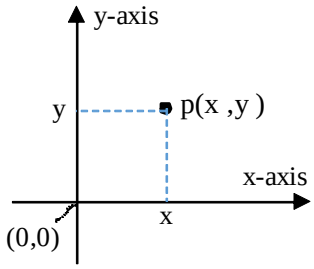


All of these actions can be expressed using matrices.

Let's start with a point p(x , y) in a Cartesian coordinate system.

Translation is fairly easy:

$$\begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix} \rightarrow \begin{pmatrix} \Delta x + x \\ \Delta y + y \end{pmatrix}$$



So is *scaling* if it only has one scaling factor, s:

$$s \cdot \begin{pmatrix} x \\ y \end{pmatrix} \rightarrow \begin{pmatrix} s \cdot x \\ s \cdot y \end{pmatrix}$$

If you want to add different scaling factors, s_h and s_v, for horizontal and vertical scaling respectively, it gets a little bit more complicated, with the matrix multiplication looking like this:

$$\begin{pmatrix} S_h & 0 \\ 0 & S_v \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} \rightarrow \begin{pmatrix} S_h \cdot x \\ S_v \cdot y \end{pmatrix}$$

Reflection across the x-axis works basically by modifying an identity matrix so that y takes negative values and vice versa.

Reflection across the x-axis

$$\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} \rightarrow \begin{pmatrix} x \\ -y \end{pmatrix}$$

Reflection across the y-axis:

$$\begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} \rightarrow \begin{pmatrix} -x \\ y \end{pmatrix}$$

Image source: http://www.11plusforparents.co.uk/Maths/shape17.html

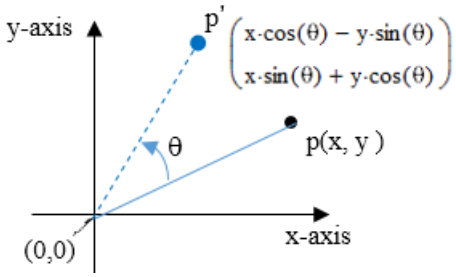
Subsection 2.1: **Rotation**

Rotation is by far the most complicated.

In order to rotate a shape we first need an angle. Let's call this angle θ.

To rotate the point about the origin we must multiply its coordinates with what is called a rotation matrix which looks like this:

$$R(\theta) := \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix}$$



And the transformation is:

$$\begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} \rightarrow \begin{pmatrix} x\cdot\cos(\theta) - y\cdot\sin(\theta) \\ x\cdot\sin(\theta) + y\cdot\cos(\theta) \end{pmatrix}$$

By using matrices, we can combine these operations into one big operation. The combined matrix is known as an *affine transformation*.

Section 3: **Affine and Linear Transformations**

Before we reveal the combined matrix, let's take a moment and understand what an affine operation is.

In mathematics a function, f(x), is defined as being linear if

$$f(\alpha\cdot x_1 + \beta\cdot x_2) = \alpha\cdot f(x_1) + \beta\cdot f(x_2).$$

In other words, any linear function can be expressed as a straight line passing through the origin. A function is also linear if

$f(x,y)$ can be expressed as $f(x,y) = ax + by$.

Affine functions are basically linear functions with an extra twist added on: they contain a translation.

This means that when graphed on plane they are still straight lines, but they do not necessarily need to pass through the origin. Also, in this case $f(x,y)$ can be expressed as $f(x,y) = ax + by + c$.

Section 4: **The Combined Transformation Matrix**

Now let's reveal the combined matrix.

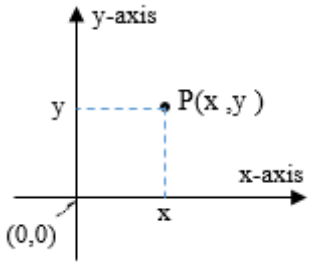
Start with your original point p(x,y) and apply a rotation, a scaling, and a translation.

Let's define each part.

$\begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix}$ Rotation Matrix

s Scaling Factor

$\begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix}$ Translation



From here, if we consider first rotation followed by scaling followed by translation, we can combine all the operations in one equation as follows:

$$s\cdot\begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix}\cdot\begin{pmatrix} x \\ y \end{pmatrix}+\begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix}\rightarrow\begin{pmatrix} \Delta x + s\cdot x\cdot\cos(\theta) - s\cdot y\cdot\sin(\theta) \\ \Delta y + s\cdot x\cdot\sin(\theta) + s\cdot y\cdot\cos(\theta) \end{pmatrix}$$

Furthermore, we can simplify this equation if we include translation in the matrix multiplication by appending a dummy row as shown below:

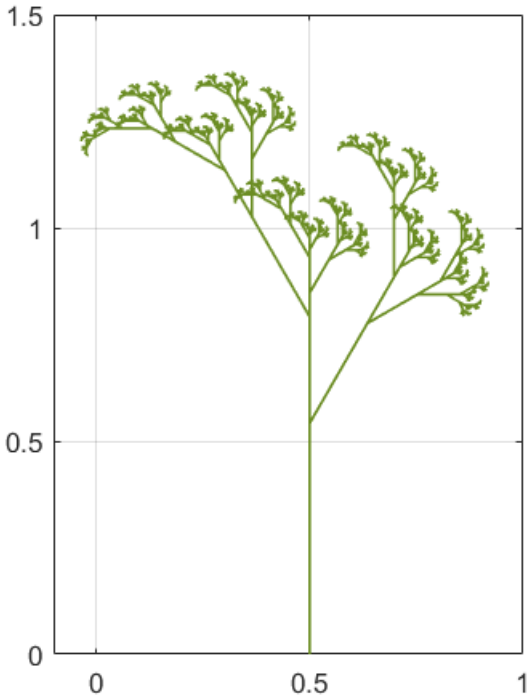
$$\begin{pmatrix} s & 0 & \Delta x \\ 0 & s & \Delta y \\ 0 & 0 & 1 \end{pmatrix}\cdot\begin{pmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix}\cdot\begin{pmatrix} x \\ y \\ 1 \end{pmatrix}\rightarrow\begin{pmatrix} \Delta x + s\cdot x\cdot\cos(\theta) - s\cdot y\cdot\sin(\theta) \\ \Delta y + s\cdot x\cdot\sin(\theta) + s\cdot y\cdot\cos(\theta) \\ 1 \end{pmatrix}$$

$T = \begin{pmatrix} s & 0 & \Delta x \\ 0 & s & \Delta y \\ 0 & 0 & 1 \end{pmatrix}\cdot\begin{pmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix}$ is the combined transformation matrix and represents a rotation, followed by a scaling, followed by a translation (not the other way around).

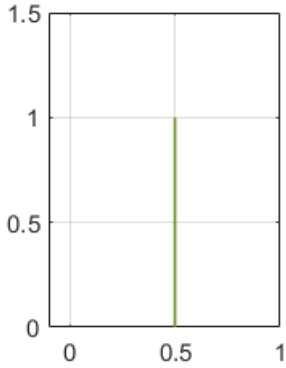
The order we apply these operations matters as they are not commutative!

Section 5: Example 1 - **Fractal Tree**

In order to understand how these transformations can be used to generate fractals, let's look at the following example which uses only four transformations to build a deceptively complex fractal tree.



We start with a vertical straight line measuring 1 unit at a distance x_0=0.5 units from the origin.



Next, let's add the vertices of this geometry in a matrix that looks like $P_0 = \begin{pmatrix} x_1 & x_2 \\ y_1 & y_2 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} 0.5 & 0.5 \\ 0 & 1 \\ 1 & 1 \end{pmatrix}$.

(x_1, y_1) and (x_2, y_2) are the coordinates of the line endpoints and [1 1] is a dummy row we introduced in the previous section.

Now, let's go to the code for the transformations.

First, we denote the branch angle as theta (θ).

Since most of the programming languages require the angle in radians, we define theta as $\theta := \left(\frac{\pi}{180}\right)$ *(value in degrees). In this example the angle is 30degress, therefore $\theta := \left(\frac{\pi}{180}\right) \cdot 30$.

Then we define two matrices R_1 and R_2. These are the rotation matrices and the only difference between them is that one rotates counter-clockwise while the other rotates clockwise.

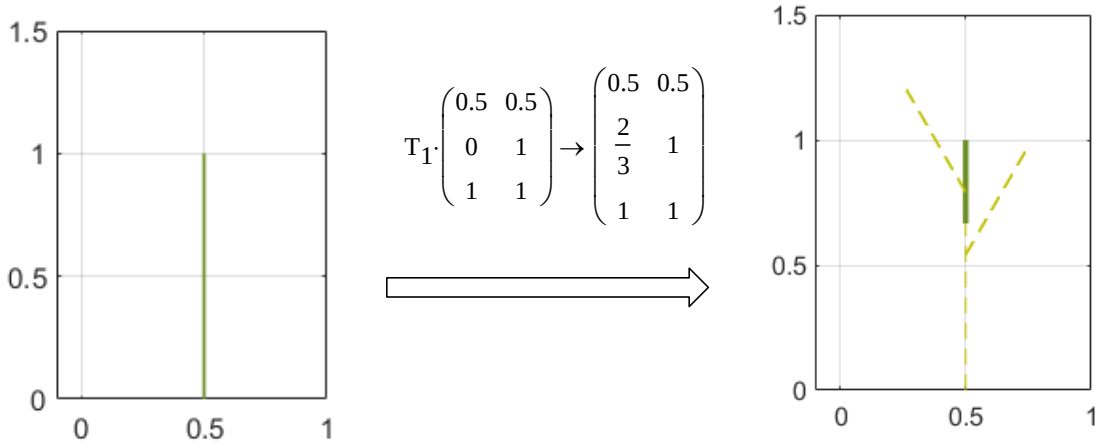
$$R_1 := \begin{pmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix} \qquad R_2 := \begin{pmatrix} \cos(-\theta) & -\sin(-\theta) & 0 \\ \sin(-\theta) & \cos(-\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Now we can proceed with the transformations.

- The first transformation T1 looks like this:

$$T_1 := \begin{pmatrix} \frac{1}{3} & 0 & \frac{1}{3} \\ 0 & \frac{1}{3} & \frac{2}{3} \\ 0 & 0 & 1 \end{pmatrix}$$

This matrix simply takes the original line, scales it down by a factor of 1/3 ($s_1 = \frac{1}{3}$), and then positions it so it aligns with the top 1/3 of the original line as shown in the next figure. If you took a picture after this transformation, it would seem as though nothing happened, but in reality an extra line was added on top of the original line.

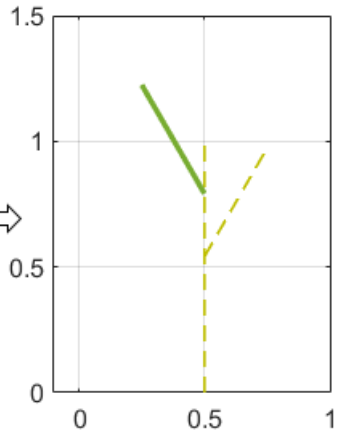


- The next two transformations will create the branches:

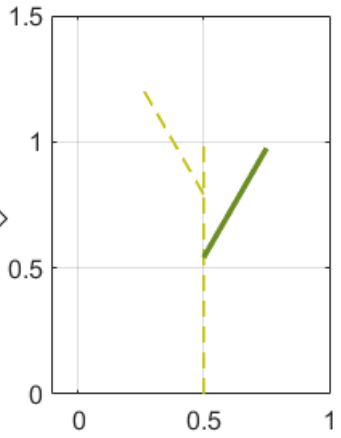
$$T_2 := \begin{bmatrix} \frac{1}{2} & 0 & 0.5\cdot\left(1-\frac{1}{2}\cos(\theta)\right) \\ 0 & \frac{1}{2} & \frac{2}{3} \\ 0 & 0 & 1 \end{bmatrix}\cdot R_1 \qquad T_3 := \begin{bmatrix} \frac{1}{2} & 0 & 0.5\cdot\left(1-\frac{1}{2}\cos(\theta)\right) \\ 0 & \frac{1}{2} & \frac{2}{3} \\ 0 & 0 & 1 \end{bmatrix}\cdot R_2$$

T2 rotates the line around the origin by θ radians counter-clockwise, scales it by half $(s_2 = \frac{1}{2})$ and then translates it by $\Delta x := 0.5\cdot\left(1-\frac{1}{2}\cos(\theta)\right)$ and $\Delta y := \frac{2}{3}$. T3 does the same thing in the opposite direction (clockwise), as it can be seen in the next figures.

$$T_2\cdot\begin{pmatrix} 0.5 & 0.5 \\ 0 & 1 \\ 1 & 1 \end{pmatrix} \Rightarrow$$



$$T_3\cdot\begin{pmatrix} 0.5 & 0.5 \\ 0 & 1 \\ 1 & 1 \end{pmatrix} \Rightarrow$$

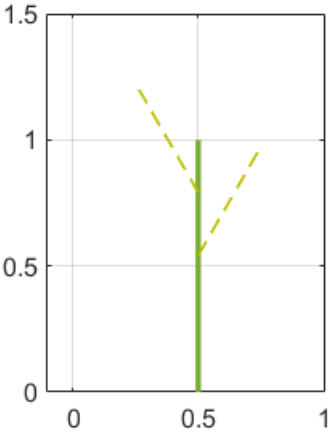


You may have noticed that the translation in the x-direction $(x_0\cdot(1-s_2\cdot\cos(\theta)))$ seems a little bit strange. The reason for this is because we need to compensate for the offset caused by the rotation around the origin and ensure each line (branch) is connected to the tree.

- The final transformation is simply an identity matrix needed to keep the original line.

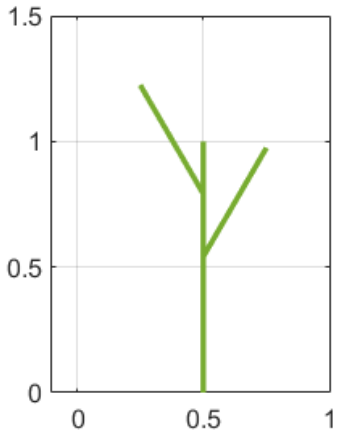
$$T_4 := \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$T_4\cdot\begin{pmatrix} 0.5 & 0.5 \\ 0 & 1 \\ 1 & 1 \end{pmatrix} \rightarrow \begin{pmatrix} 0.5 & 0.5 \\ 0 & 1 \\ 1 & 1 \end{pmatrix} \Rightarrow$$



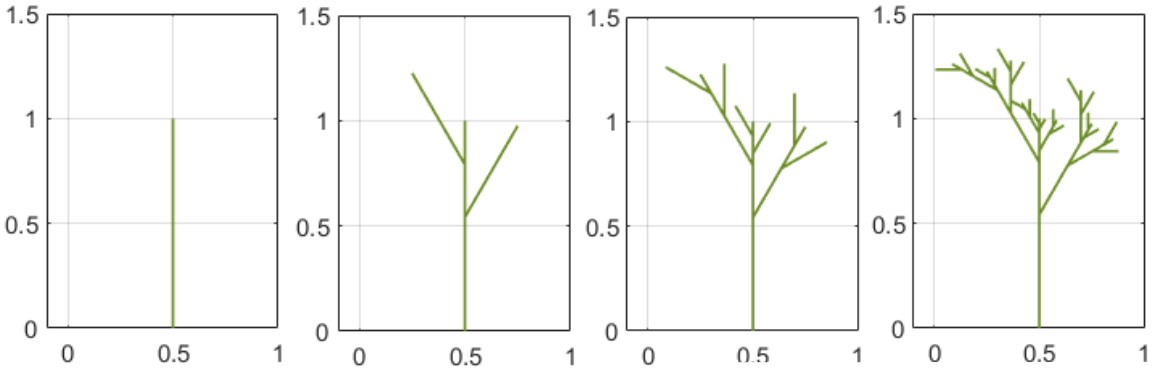
After applying these transformations one time, we end up with a figure which looks like this:

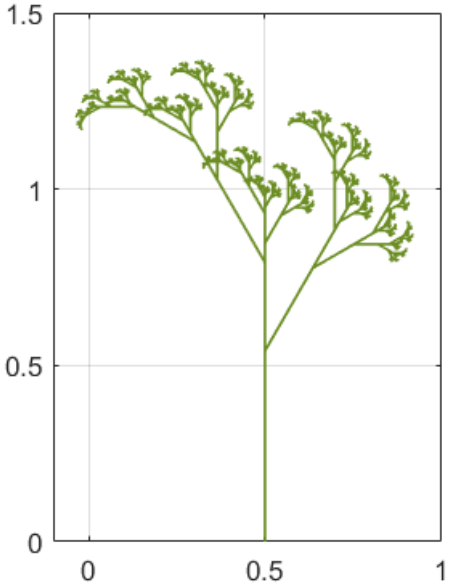
T_1, T_2, T_3, T_4 ⟹



After each iteration we repeat these transformations to the new parts and get a more detailed tree. These iterations compile and leave us with a final tree-looking figure.

original configuration iteration 1 iteration 2 iteration 3





This style of generating fractals is called MRCM, which stands for Multiple Reduction Copy Machine. This is because it acts very much like a copier, taking an original image, scaling and rotating it, and arranging the copies into a new shape. It does the same thing to each of the new copies it makes generating a complex fractal after a sufficient amount of iterations.

And if you would like to create you own tree or an entire forest, you can find the MATLAB code in the Appendix II.

Section 6: Example 2 - **The Koch Snowflake**

The best thing about the code we just made is that we can use it for more than one fractal.
By simply changing the transformations, we can generate beautiful shapes like the Koch Snowflake shown below.

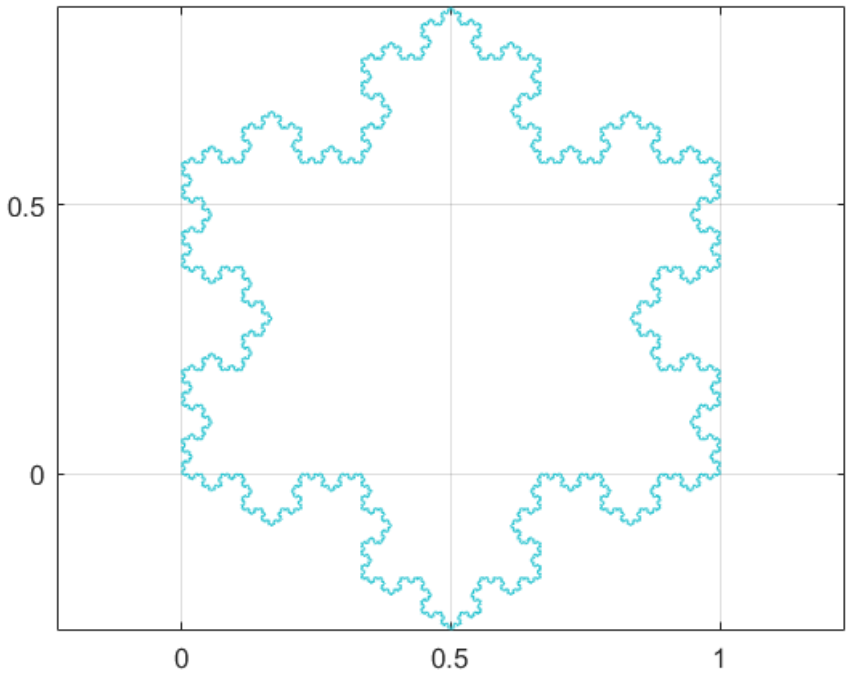


Figure 6.1 Koch Snowflake

To generate the Koch Snowflake we must first generate the Koch curve, as three of these curves create the snowflake.

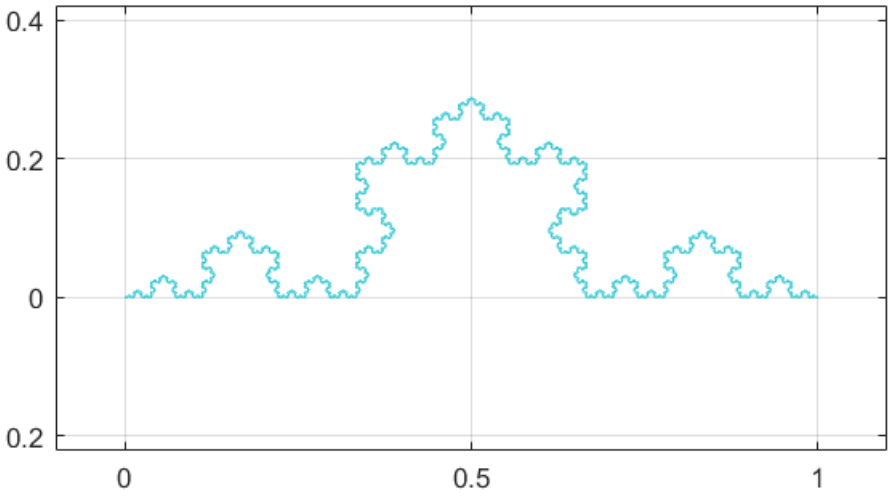
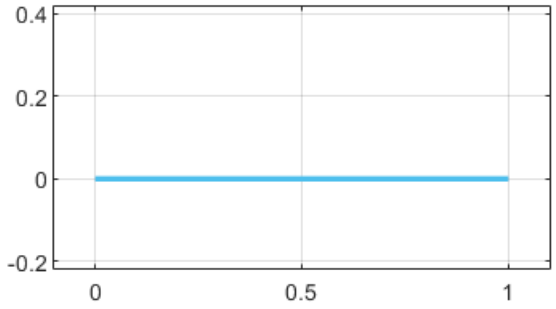


Figure 6.2 Koch Curve

The Koch Curve geometry is obtained with four transformations applied to an horizontal line measuring 1 unit. The matrix of coordinates for this line is $P_0 = \begin{pmatrix} x_1 & x_2 \\ y_1 & y_2 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 0 & 0 \\ 1 & 1 \end{pmatrix}$.

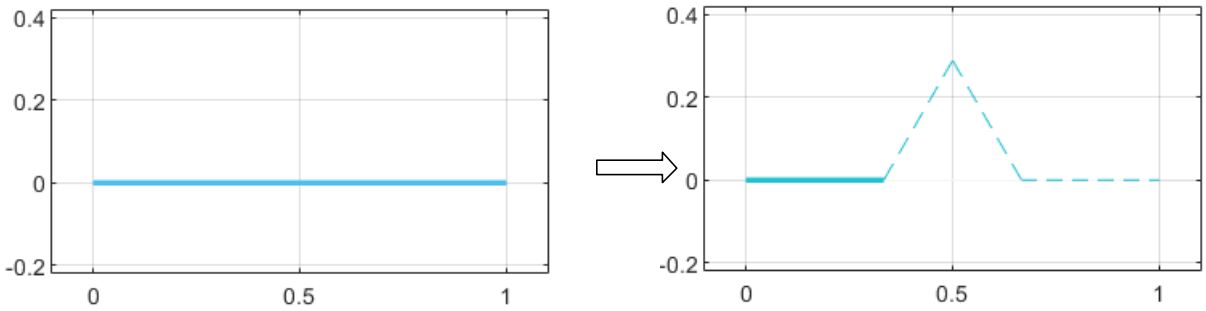


First we define two rotation matrices R_1 and R_2. R_1 rotates the line 60 degrees counterclockwise whereas R2 rotates the line 60 degrees clockwise.

$$R_1 := \begin{pmatrix} \cos\left(\frac{\pi}{3}\right) & -\sin\left(\frac{\pi}{3}\right) & 0 \\ \sin\left(\frac{\pi}{3}\right) & \cos\left(\frac{\pi}{3}\right) & 0 \\ 0 & 0 & 1 \end{pmatrix} \qquad R_2 := \begin{pmatrix} \cos\left(\frac{-\pi}{3}\right) & -\sin\left(\frac{-\pi}{3}\right) & 0 \\ \sin\left(\frac{-\pi}{3}\right) & \cos\left(\frac{-\pi}{3}\right) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

- Our first transformation, $T_1 := \begin{pmatrix} \frac{1}{3} & 0 & 0 \\ 0 & \frac{1}{3} & 0 \\ 0 & 0 & 1 \end{pmatrix}$, scales down the original line by a factor of 1/3 and the equation looks like this:

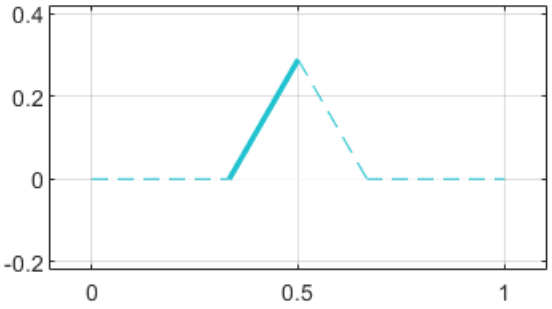
$$T_1 \cdot \begin{pmatrix} 0 & 1 \\ 0 & 0 \\ 1 & 1 \end{pmatrix} \rightarrow \begin{pmatrix} 0 & \frac{1}{3} \\ 0 & 0 \\ 1 & 1 \end{pmatrix}$$



- The second transformation, $T_2 := \begin{pmatrix} \frac{1}{3} & 0 & \frac{1}{3} \\ 0 & \frac{1}{3} & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot R_1$, contains three operations:

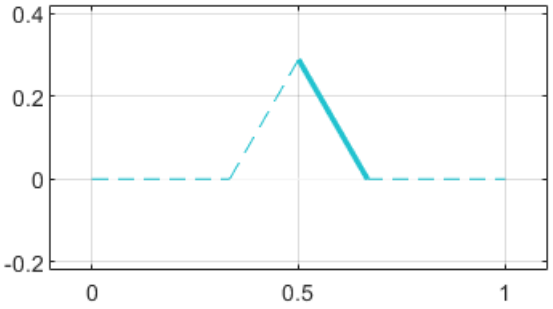
1. Rotate the original line 60 degrees counterclockwise;
2. Scale down the rotated line by a factor of 1/3;
3. Take the scaled line and move it right by a distance of 1/3 of the original line.

$$\begin{pmatrix} \frac{1}{3} & 0 & \frac{1}{3} \\ 0 & \frac{1}{3} & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot R_1 \cdot \begin{pmatrix} 0 & 1 \\ 0 & 0 \\ 1 & 1 \end{pmatrix} \rightarrow \begin{pmatrix} \frac{1}{3} & \frac{1}{2} \\ 0 & \frac{\sqrt{3}}{6} \\ 1 & 1 \end{pmatrix} \Longrightarrow$$



- The third transformation, $T_3 := \begin{pmatrix} \frac{1}{3} & 0 & \frac{1}{2} \\ 0 & \frac{1}{3} & \frac{\sqrt{3}}{6} \\ 0 & 0 & 1 \end{pmatrix} \cdot R_2$, is similar in concept to the second, but uses different translations.

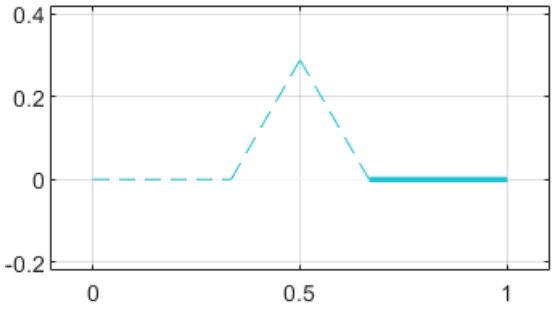
$$\begin{pmatrix} \frac{1}{3} & 0 & \frac{1}{2} \\ 0 & \frac{1}{3} & \frac{\sqrt{3}}{6} \\ 0 & 0 & 1 \end{pmatrix} \cdot R_2 \cdot \begin{pmatrix} 0 & 1 \\ 0 & 0 \\ 1 & 1 \end{pmatrix} \rightarrow \begin{pmatrix} \frac{1}{2} & \frac{2}{3} \\ \frac{\sqrt{3}}{6} & 0 \\ 1 & 1 \end{pmatrix} \Longrightarrow$$



Although the translations may seem odd, they can be derived with a bit of geometry. Otherwise, the only other difference between the second and third rotation is the way in which they rotate, as the third transformation rotates clockwise, not counterclockwise.

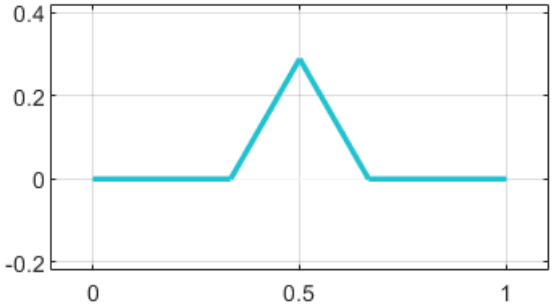
- The fourth and final transformation is:

$$T_4 := \begin{pmatrix} \frac{1}{3} & 0 & \frac{2}{3} \\ 0 & \frac{1}{3} & 0 \\ 0 & 0 & 1 \end{pmatrix} \qquad T_4 \cdot \begin{pmatrix} 0 & 1 \\ 0 & 0 \\ 1 & 1 \end{pmatrix}$$

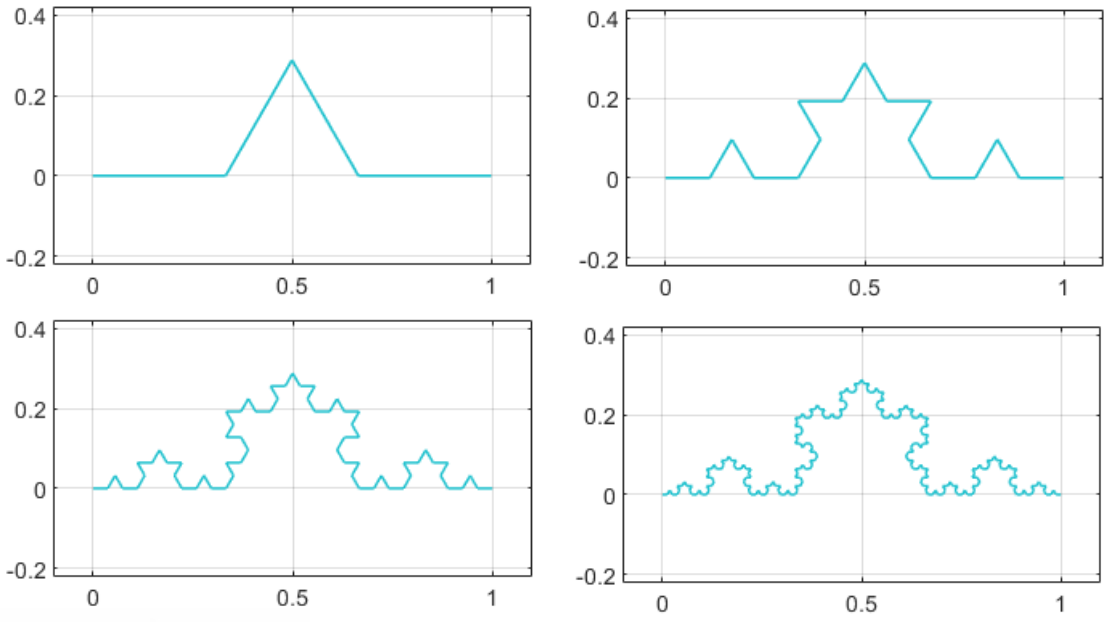


As we can see this transformation is very similar to T_1, but it moves the scaled line right by a distance of 2/3 of the original line.

After one iteration the transformed line looks like this:



These iterations are endlessly repeated on an increasingly smaller scale to produce the final Koch curve as show the figures below.

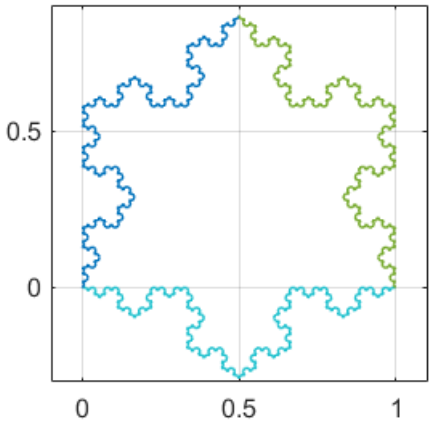


Subsection 6.1: **The Birth of a Snowflake**

Although the finished Koch curve brings us closer to, it does not generate the desired Koch snowflake. In order to create the snowflake, we need to put together three Koch Curves.

For our purposes, let's assemble them as shown below and denote the corresponding matrices of coordinates $\begin{pmatrix} x_1 & x_2 & x_3 & \dots \\ y_1 & y_2 & y_3 & \dots \\ 1 & 1 & 1 & \dots \end{pmatrix}$ as:

np_1 - the bottom curve (light blue)
np_2 - the left curve (dark blue)
np_3 - the right curve (green)
np - the original Koch curve as generated in the previous section.

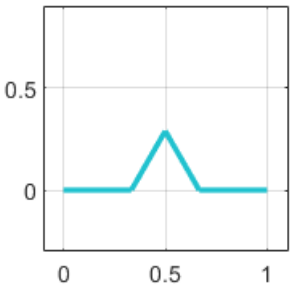


Since the rotation matrix will be used multiple times with different angles, let's define it as a function of theta(θ). This will also help us simplify the code.

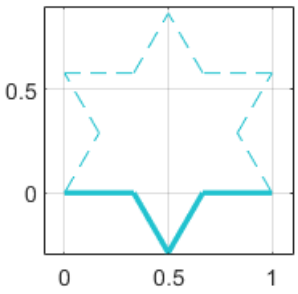
$$R(\theta) := \begin{pmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

- First, let's see how we can generate the bottom piece.

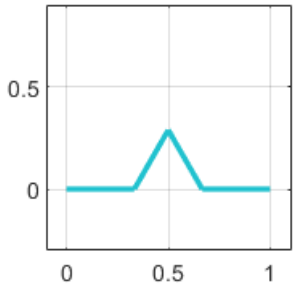
This part is actually relatively simple because all we need to do is reflect the original Koch curve across the x-axis.



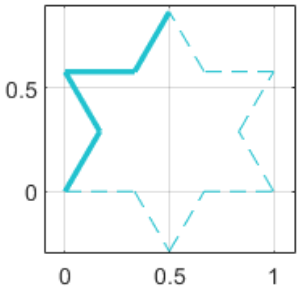
$$np_1 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot np$$



- Then, let's generate the left piece of the snowflake which can be obtained by rotating the original curve by 60 degrees counterclockwise.

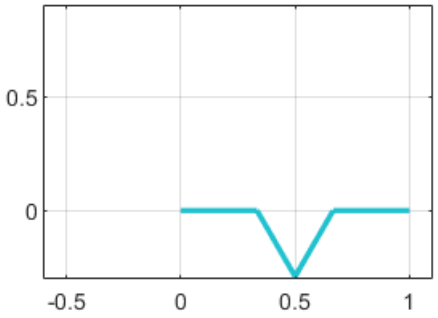


$$np_2 = R(\theta_1)\cdot np = R\left(\frac{\pi}{3}\right)\cdot np$$

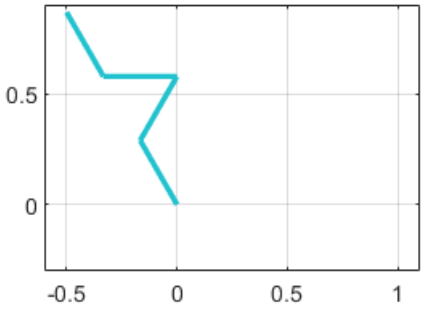


- Finally, let's look at the transformations which generate the right piece of the snowflake.

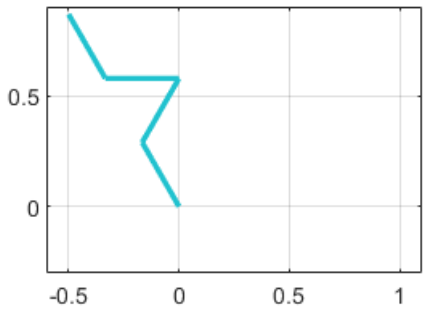
The first transformation takes the bottom portion of the snowflake (np_1) and rotates it 120 degrees counterclockwise as shown below.



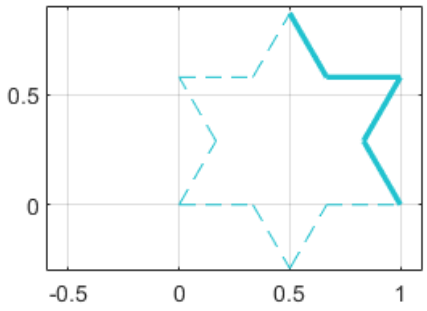
$$np_{3a} = R(\theta_2)\cdot np_1$$



However, this rotated piece is too far left. Therefore, in order to move it right we use our second transformation which is a horizontal translation by $\Delta x = 1$. The matrix equation looks like this:

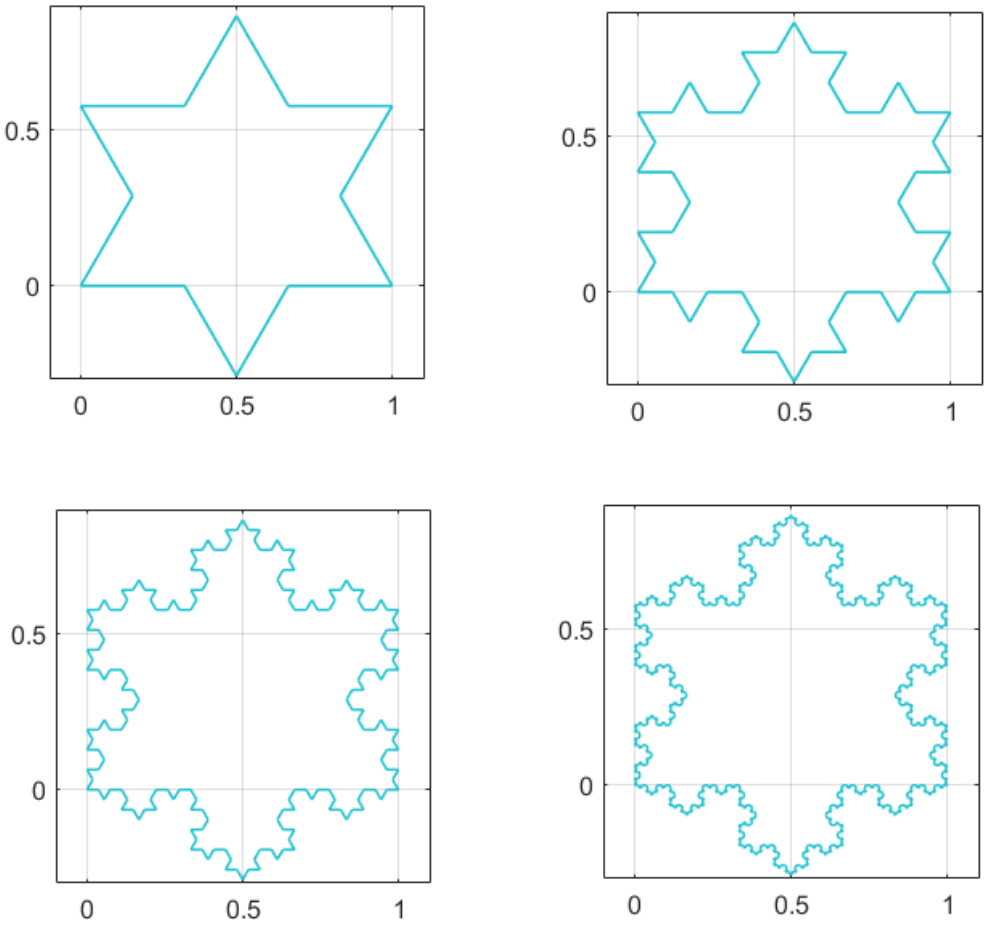


$$np_3 = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}\cdot np_{3a}$$



In MATLAB, this operation can also be expressed as `np3(1,:) = np3a(1,:) + 1`. This instruction takes all the values in the first row of np3a and adds 1, shifting the x-coordinates one unit to the right.

Combining all these equations together, we get a complete Koch snowflake. The first four iterations look like this:



And now, with the MATLAB code in the Appendix III, let it snow!

Acknowledgements

Special thanks to Dr. Adriana Hera for teaching me MATLAB and revising this handout to improve the clarity of some sections.

I would also like to acknowledge the Academic and Research Computing group with whom I worked. I am really appreciative for all the knowledge and skills they shared, and for the horizons they opened up for me.

Finally, I would like to thank Worcester Polytechnic Institute for providing me the opportunity to contribute to their wonderful summer program.

References

Grob, M., & Heidelberg, W. (2005, February 1). MRCM. Retrieved June 2018, from http://m2matlabdb.ma.tum.de/download.jsp?MC_ID=5&SC_ID=13&MP_ID=324.

Flake, G. W. (2011). The computational beauty of nature: computer explorations of fractals, chaos, complex systems, and adaptation. Cambridge, MA: The MIT Press.

Mandelbrot Benoît B. (2004). The Fractal Geometry of Nature. New York: Freeman & Co.

Matrices | Precalculus | Math. (n.d.). Retrieved from https://www.khanacademy.org/math/precalculus/x9e81a4f98389efdf:matrices.

Moler, C. (2011). Experiments with Matlab. Mathworks.

Strang, G. (1993). Introduction to linear algebra. Wellesley, MA: Wellesley-Cambridge Pr.

(2011, August 24). Retrieved October 28, 2019, from https://www.pbs.org/wgbh/nova/physics/hunting-hidden-dimension.html.

Appendix I: **Fractals in Nature and Applications**

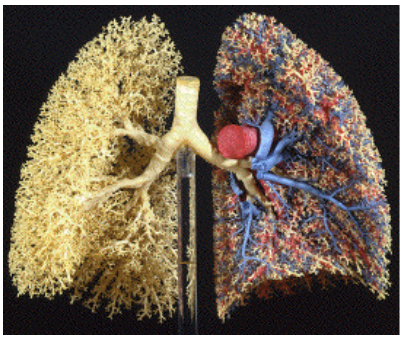
"Clouds are not spheres, mountains are not cones, coastlines are not circles, and bark is not smooth, nor does lightning travel in a straight line."

- *Benoît Mandelbrot*

- Human Anatomy
- Plants
- Clouds
- Coastlines
- Landscapes
- Fractal Antennas (cell phone)
- Data Compression
- Fractal Art
- Special Effects (Star Trek)
- Fractal materials
- Stock market

Human Anatomy

- Where are fractals found in the body
  - The lungs, the kidneys, the circulatory system, and the brain are all fractal systems
  - Other things such as eye movement and your heartbeat are also based on fractals
  - The spread of many diseases such as AIDS or cancer can be modeled by fractals.
- Advantages to having a fractal system
  - Fractals have been chosen through natural selection as the best way to organize a body
  - This is due to the fact that they maximize efficiency
  - For example, the lungs have a volume equivalent to a few tennis balls but have a surface area equivalent to that of a tennis court.



Fractals in Plants

- Fractals can also be seen in nature in the way plants are shaped.
- A classic example is the fern, whose individual leaves are reminiscent of the entire fern.
- Most trees and leaves follow a fractal branching pattern.
- One of the primary examples of fractals in plants is Romanesco Broccoli.

Fractals and Coastlines

- Coastlines are the quintessential example of roughness in nature.
- They are continually rough and cannot be described through simple Euclidean geometry very well.
    - For example if you look at a map of a coastline at a scale of 1in to 100,000 ft, the coastline looks very smooth and continuous.
    - However, if you visit any portion of coastline you know that this is untrue as the coastline is quite random and unpredictable.
    - If you take a smaller scale you tend to notice more details.
    - As your scale gets smaller, those original details you noticed start to look more and more like your original coastline.
    - This property of roughness at all levels causes coastlines to be self-similar

- This is why fractals, not lines and curves, are best suited to describe coastlines.

Fractal Antennas

- All contain self-similar design
- Generally are space filling curves
    - Examples
        - Peano curve
        - Hilbert Curve
- More perimeter can be fit in a smaller space meaning the antenna can pick up a wider range of frequencies
- Benefits
    - Compact
    - Conformal
    - Broadband

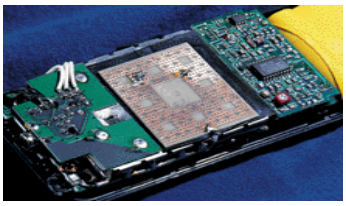


Image sources:

Fractal antenna: http://large.stanford.edu/courses/2012/ph250/ferguson1/
http://www.antenna-theory.com/antennas/fractal.php
Fractal Lungs: https://fractalfoundation.org/OFC/OFC-1-2.html
Fractal Fern: https://en.wikipedia.org/wiki/Barnsley_fern
Broccoflower: https://www.fourmilab.ch/images/Romanesco/
Fractal Coastline: https://www.flickr.com/photos/buggs_moran/4516938146

Appendix II: **MATLAB Code for a Fractal Tree**

Generating fractals using the MRCM method

MRCM method (Multiple Reduction Copy Machine)

```
% references: http://m2matlabdb.ma.tum.de/files.jsp?MC_ID=5&SC_ID=13
%             http://m2matlabdb.ma.tum.de/download.jsp?MC_ID=5&SC_ID=13&MP_ID=324
%              Lecture notes "Methods for Generating Fractals" by Alex Nedelescu, 2018
% by Alex Nedelescu (danedelescu@wpi.edu), summer 2018
```

Original Configuration

```
clear, clc, close all;
N=5; % number of iteration
pts =[0.5, 0; 0.5, 1];% coordintaes of the original configuration:
%    [x1 y1; x2 y2], a straight vertical line

% Define the tree color: [R, G, B]/noColors
darkGreen = [85,107, 47]/256; forestGreen = [34, 139, 34]/256; oliveGreen =[107, 142,35]/256;

% Figure properties
my_fig=figure(); hold on; axis equal; axis([-0.1,1, 0,1.5]);
xticks([0, 0.5, 1]); yticks([0, 0.5, 1, 1.5]);
daspect([1 1 1]); box on; grid on;
% set(gcf, 'units', 'normalized'); set(gcf, 'Position', [0, 0.1, 0.1, 0.3]);

H = plot(pts(:,1),pts(:,2),'Color',[0 1 0]); % original configuration
```

Define the transformations

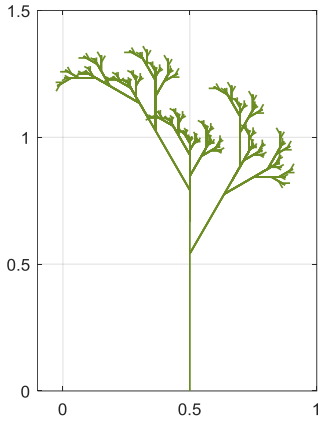
```
theta=(pi/180)*30; % branch angle
R1=[cos(theta) -sin(theta) 0;sin(theta) cos(theta) 0;0 0 1];
R2=[cos(-theta) -sin(-theta) 0;sin(-theta) cos(-theta) 0;0 0 1];
x0=pts(1,1); % =0.5
dx=x0*(1-1/2*cos(theta));

T{1} = [1/3 0 1/3; 0 1/3 2/3; 0 0 1];
T{2} = [1/2 0 dx; 0 1/2 2/3; 0 0 1]*R1;
T{3} = [1/2 0 dx; 0 1/2 2/3; 0 0 1]*R2;
T{4} = [1 0 0;0 1 0;0 0 1];
```

MRCM code

```
mrcm_Tree(pts,N,T,H,oliveGreen);
```

```
function mrcm_Tree(pts,N,T,handle, colorTree)
% function for MRCM (Multiple Reduction Copy Machine)
% pauses between iterations
% reference:http://m2matlabdb.ma.tum.de/files.jsp?MC_ID=5&SC_ID=13
% http://m2matlabdb.ma.tum.de/download.jsp?MC_ID=5&SC_ID=13&MP_ID=324
% code by Alex Nedelescu (danedelescu@wpi.edu), summer 2018

np = [pts'; ones(1,size(pts,1))];   % add the extra row [x;y;1]

set(handle,'xdata',np(1,:),'ydata',np(2,:));

for i = 1:N
    xp = [];
    pause
    for j = 1:length(T) % length(T) = number of transformations
        p = T{j}*np;   % apply transformation
        % to plot each transformation: uncomment next line
        % plot(p(1,:), p(2,:),'LineWidth',1, 'Color', colorTree);

       xp = [xp [NaN NaN 1]', p];  % add image to other copies
    end
    np = xp;
    % plot the entire image after each iteration
    set(handle,'xdata',np(1,:),'ydata',np(2,:), 'Color', colorTree);
end
```

Published with MATLAB® R2018a



Appendix III: **MATLAB Code for a Snowflake**

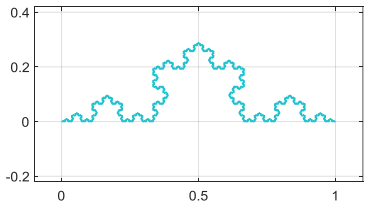
Generating fractals using the MRCM method

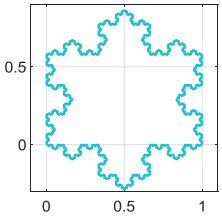
MRCM method (Multiple Reduction Copy Machine)
This code plots both the Koch curve (type =1) and the snowflake (type =2) .

```
% references: http://m2matlabdb.ma.tum.de/files.jsp?MC_ID=5&SC_ID=13
%             http://m2matlabdb.ma.tum.de/download.jsp?MC_ID=5&SC_ID=13&MP_ID=324
%                Lecture notes "Methods for Generating Fractals" by Alex Nedelescu, 2018
% by Alex Nedelescu (danedelescu@wpi.edu), summer 2018
```

Original Configuration

```
clear, clc, close all;
N=5; % number of iteration
pts = [0,0; 1,0]; % Original configuration
%    [x1 y1; x2 y2], a straight vertical line
type = 2; % 1 - Koch curve', 2 - snowflake

% Define the color: [R, G, B]/noColors
blueSnow = [32,195,208]/256;

% figure properties
my_fig=figure('Color',[1 1 1]);; hold on; axis equal;
daspect([1 1 1]); box on; grid on
if type==1
    axis([-0.1,1.1, -0.22, 0.42]);
    xticks([0, 0.5, 1]); yticks([-0.2, 0, 0.2, 0.4])
    set(gcf, 'units', 'normalized'); set(gcf, 'Position', [0, 0.1, 0.2, 0.2]);
else
   axis([-0.1,1.1, -0.3, 0.9]);
    xticks([0, 0.5, 1]); yticks([-0.5, 0, 0.5])
    set(gcf, 'units', 'normalized'); set(gcf, 'Position', [0, 0.1, 0.2, 0.2]);

end
H = plot(pts(:,1),pts(:,2),'Color',[0 0 0]); % original configuration
```





Define the transformations

```
R1=[cos(pi/3) -sin(pi/3) 0;sin(pi/3) cos(pi/3) 0;0 0 1]; %60 degree rot
R2=[cos(-pi/3) -sin(-pi/3) 0;sin(-pi/3) cos(-pi/3) 0;0 0 1]; %-60 degree rot

T{1} = [1/3 0 0; 0 1/3 0; 0 0 1];
T{2} = [1/3 0 1/3; 0 1/3 0; 0 0 1]*R1;
T{3} = [1/3 0 1/2; 0 1/3 sqrt(3)/6; 0 0 1]*R2;
T{4} = [1/3 0 2/3; 0 1/3 0; 0 0 1];
```

MRCM code

```
mrcm_Snow(pts,N,T,H,blueSnow,type);
```

```
function mrcm_Snow(pts,N,T,handle, snowColor,type)
% function for MRCM (Multiple Reduction Copy Machine)
% pauses between iterations
% reference:http://m2matlabdb.ma.tum.de/files.jsp?MC_ID=5&SC_ID=13
% http://m2matlabdb.ma.tum.de/download.jsp?MC_ID=5&SC_ID=13&MP_ID=324
% code by Alex Nedelescu (danedelescu@wpi.edu), summer 2018

np = [pts'; ones(1,size(pts,1))];   % [x;y;1], add the extra row

set(handle,'xdata',np(1,:),'ydata',np(2,:));
R=@(theta) ([cos(theta) -sin(theta) 0;sin(theta) cos(theta) 0;0 0 1]);
theta1=pi/3;
theta2=2*pi/3;

for i = 1:N
    xp = [];
    pause
    for j = 1:length(T) % length(T) = number of transformations
        p = T{j}*np;   % apply transformation
        % to plot each transformation: uncomment next line
        % plot(p(1,:), p(2,:),'LineWidth',1, 'Color', snowColor);
        xp = [xp [NaN NaN 1]', p];  % add image to other copies
    end
    np = xp;
    np1=[1 0 0;0 -1 0;0 0 1]*np;
    np2=R(theta1)*np;
    np3=R(theta2)*np1;
    np3(1,:)=np3(1,:)+1;
    if type == 1; npc=np; else npc=[np1, np2, np3];
    end
    % plot the entire image after each iteration
    set(handle,'xdata',npc(1,:),'ydata',npc(2,:), 'Color', snowColor, 'LineWidth',1);
    % to plot each Koch curve in the snowflake uncomment the next lines
%     plot (np1(1,:),np1(2,:), 'Color', snowColor, 'LineWidth',1);
%     plot (np2(1,:),np2(2,:), 'Color', snowColor, 'LineWidth',1);
%     plot (np3(1,:),np3(2,:), 'Color', snowColor, 'LineWidth',1);
end
```