An Efficient Algorithm to Generate Grids Using a Modified Transformation Method

Abstract

Basic building block in a structure or a design can be an open square formed by connected lines. A mapping between points on a line segment and points in a square is displayed by various Space Filling Curves(SFCs), which show the relations between those points in one dimensional and two dimensional domains. Thereafter, a recursive building block which is a smaller version of the original open square can fill out the whole two dimensional plane, a square.

In this project, novel building blocks were suggested using an alternative geometrical analysis and combinatorial algorithm. MATLAB and JavaScript programming were developed for the project. This research aims constructing artistic patterns or designs while mathematical and computational iteration fill out a two dimensional domain. Using the presented procedures in the Space Filling Curves analysis, a new alignment was introduced so that one quadrant's end of the curve lines up with the next quadrant's beginning of the curve in an efficient way.

The objective of this research project is developing an alternative method that can be efficiently used in grid generation or Space Filling Curve analysis, which can be applied to solar cell engineering(grid generation), the image processing, numerical finite element methods, and combinatorial optimization. Compared to the building blocks by the existing SFC method, such as Peano and Hilbert curve, present research illustrated histogram analysis and graphical data showing that filling an open square using the present method can be efficient in CPU operations and computer running time. Using the presented simplified complex numbers and mathematical notations, less operations such as less matrix multiplications were observed.

Introduction

Many combinatorial mathematicians have studied the patterns of SFCs(Space Filling Curves. Also many algorithms about those patterns have been developed. Peano, Cantor, and Hilbert proposed a geometric generation principle for the construction of the Space Filling Curves [4]. Many other space-filling curves have been discovered since 1900. These and related ideas, together with a detailed bibliography, may be found in the book by H. Sagan [1].

The relationship between points on a line segment and points in a square is displayed by the Hilbert curve, which shows the mapping between those points. A space-filling curve in n-dimensions is a continuous, surjective mapping from one domain to another domain. Basic building block of the curves is an open square formed by connected lines. A complex process made by the mathematicians need to be analyzed to simplify the procedure recursively converting each points on 1 dimensional domain, $I = \{t| \ 0 \leq t \leq 1\}$ which denotes the unit interval, to coordinate values on 2 dimensional unit square, $Q = \{ (x, y)| \ 0 \leq x \leq 1, 0 \leq y \leq 1\}$. The values of the mapping points on a line segment have to be between zero and one, and can be decimals or fractions, written in *m/n* form [2-3].

Defining a function $f : I \rightarrow Q$, a recursive smaller version of the original open square on 2 dimensional domain can fill out the whole square. The relationship between points on a line segment and points in a square is displayed by the SFCs, which shows the mapping between those points. Based on the studies of the SFCs, a new efficient alignment or algorithm can be formed so that one quadrant's end of the curve lines up with the next quadrant's beginning of the curve? If this can come true, we can reduce operational counts that cost calculating the complex mathematical and computational calculations on matrix manipulations.

Based on the building blocks using the existing SFC method, such as Peano and Hilbert curves, filling the open square can be possible by connecting lines [5]. Using a new algorithm studied in this paper, we aim or hypothesize that complex mathematical and computational artistic patterns can be created. Using the presented SFC(Space Filling Curve) procedure recursively, a smaller version of the original open square on 2 dimensional domain can finally fill out the whole square in an efficient manner.

The objective of this research project is developing a new algorithm that can be efficiently used in Space Filling Curve, which can be applied to the image processing (compression), numerical finite element methods, and combinatorial optimization.

Materials and Methods

Hilbert Mapping

The relationship between points on a line segment and points in a square is displayed by the Hilbert curve, which shows the mapping between those points.

We define a sequence of functions $f_n : [0,1] \to [0,1]^2$ that converges to some surjective function $f : [0,1] \to [0,1]^2$. A sequence $f_n : [0,1] \to [0,1]^2$ is called a Hilbert sequence if it converges to some surjective function $f : [0,1] \to [0,1]^2$. The n^{th} term, f_n, is called the n^{th} Hilbert curve and the convergent function f is called the Hilbert curve.

Let us define the first Hilbert curve, f_1 (and the rest of the sequence) as a connection of certain points. We first divide the square $[0,1]^2$ into four equal squares, namely $\left[0,\frac{1}{2}\right]^2$, $\left[0,\frac{1}{2}\right] \times \left[\frac{1}{2},\ 1\right]$, $\left[\frac{1}{2},1\right] \times \left[0,\frac{1}{2}\right]$, $\left[\frac{1}{2},\ 1\right]^2$, which correspond to domains h_0, h_1, h_2, and h_3 respectively.

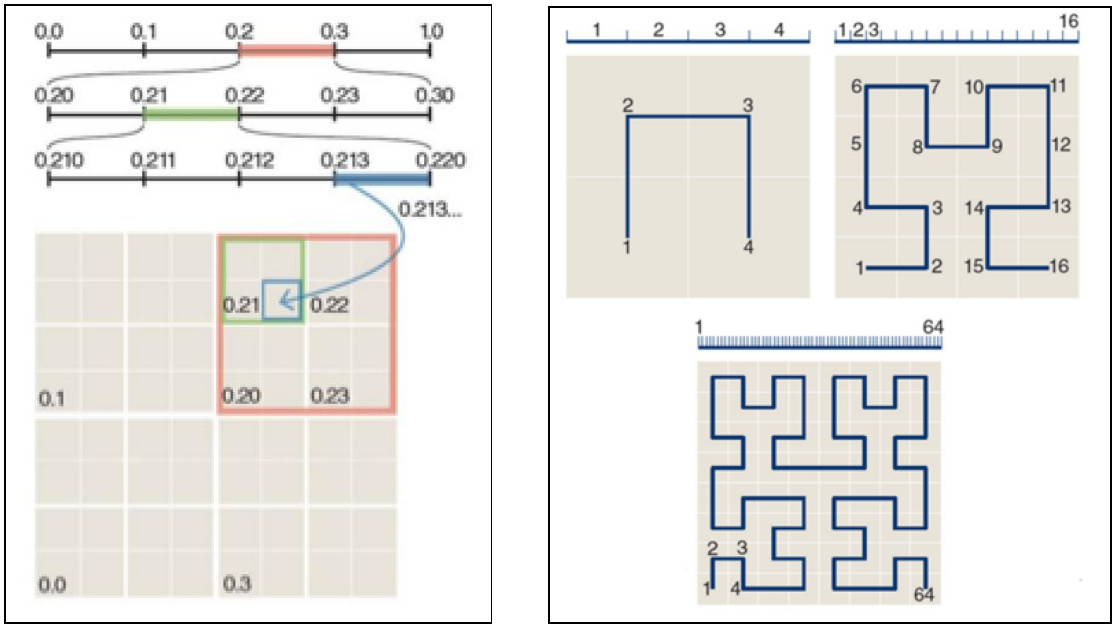


Figure 1-1. Recursive iterations to construct Hilbert curve

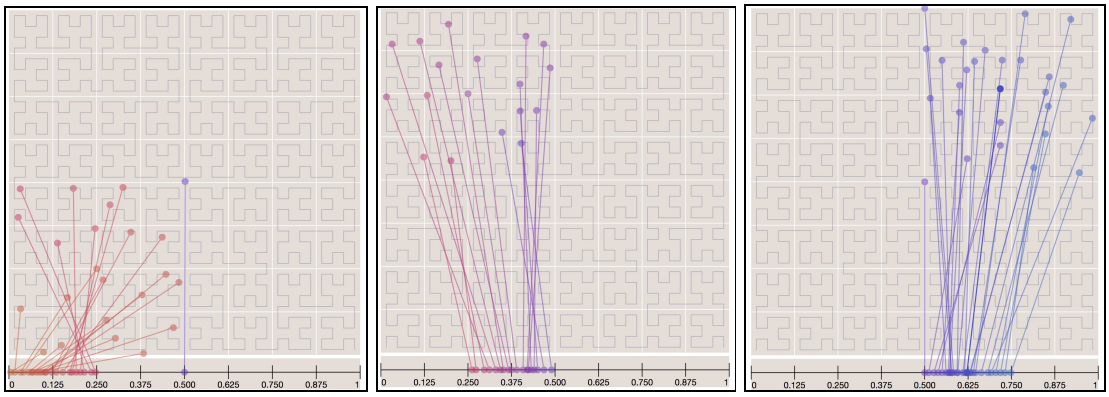


Figure 1-2. Confirmation: surjective function $f : [0, 1] \rightarrow [0, 1]^2$ (used Javascript)

Also we decimals on the line segment will be mapped into the square's related point; Any values inside the square will be mapped into the segment's related point, with the definitions of h_0, h_1, h_2, and h_3 given in the following equations (1)-(4) :

$$h_0\left(\begin{matrix}x\\y\end{matrix}\right) = \frac{1}{2}\left[\begin{matrix}0 & 1\\1 & 0\end{matrix}\right]\left(\begin{matrix}x\\y\end{matrix}\right) + \frac{1}{2}\left(\begin{matrix}0\\0\end{matrix}\right) = \frac{1}{2}H_0\left(\begin{matrix}x\\y\end{matrix}\right) + \frac{1}{2}h_0 \qquad (1)$$

$$h_1\left(\begin{matrix}x\\y\end{matrix}\right) = \frac{1}{2}\left[\begin{matrix}1 & 0\\0 & 1\end{matrix}\right]\left(\begin{matrix}x\\y\end{matrix}\right) + \frac{1}{2}\left(\begin{matrix}0\\1\end{matrix}\right) = \frac{1}{2}H_1\left(\begin{matrix}x_1\\x_2\end{matrix}\right) + \frac{1}{2}h_1 \qquad (2)$$

$$h_2\left(\begin{matrix}x\\y\end{matrix}\right) = \frac{1}{2}\left[\begin{matrix}1 & 0\\0 & 1\end{matrix}\right]\left(\begin{matrix}x\\y\end{matrix}\right) + \frac{1}{2}\left(\begin{matrix}1\\1\end{matrix}\right) = \frac{1}{2}H_2\left(\begin{matrix}x_1\\x_2\end{matrix}\right) + \frac{1}{2}h_2 \qquad (3)$$

$$h_3\left(\begin{matrix}x\\y\end{matrix}\right) = \frac{1}{2}\left[\begin{matrix}0 & -1\\-1 & 0\end{matrix}\right]\left(\begin{matrix}x\\y\end{matrix}\right) + \frac{1}{2}\left(\begin{matrix}2\\1\end{matrix}\right) = \frac{1}{2}H_3\left(\begin{matrix}x_1\\x_2\end{matrix}\right) + \frac{1}{2}h_3 \qquad (4)$$

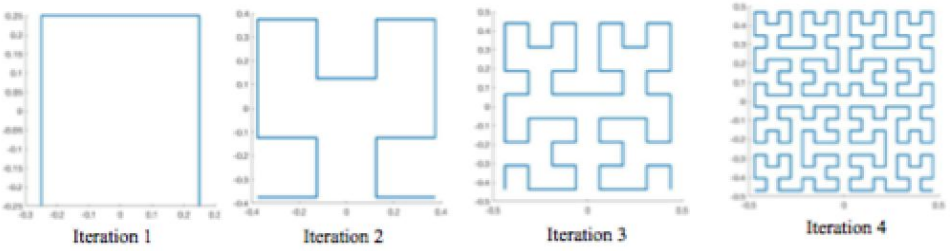


Figure 1-3. Recursive iterations to construct Hilbert curve (re-drawn using MATLAB)

With the complex variables, we may write equations (1)–(4) in the compact form:

$$h_0 z = \tfrac{1}{2}\bar{z}i$$

$$h_1 z = \tfrac{1}{2}z + \tfrac{i}{2}$$

$$h_2 z = \tfrac{1}{2}z + \tfrac{1}{2} + \tfrac{i}{2}$$

$$h_3 z = -\tfrac{1}{2}\bar{z}i + 1 + \tfrac{i}{2}$$

Or, we may write the equations in the further compact form using subscripts:

$$h_j z \triangleq \tfrac{1}{2}H_j + \tfrac{1}{2}h_j, \quad j = 0,\ 1,\ 2,\ 3$$

where $H_0 z = \bar{z}i,\ H_1 z = z,\ H_2 z = z,\ H_3 z = -\bar{z}i,\ h_0 = 0,\ h_1 = i,\ h_2 = 1 + i,$ and $h_3 = 2 + i$.

The emphasis of this research is on the simpler and faster computation of space-filling curve orders. MATLAB and JavaScript programming were developed for the project.

In order to prepare a modified algorithm, to understand how the existing recursive function operates is necessary. We will begin by looking at the inputs of the function.

Modifying the Inputs

Looking at the following MATLAB code(developed for this project) shown in listing 1, we see the Hilbert procedure calls itself in a recursive way. On each loop, it passes modified data of its original input values to the next step.

Simplified Algorithm Using Self Loop Function

First, we define [x,y] as order(n) that shows the vector coordinates[x1, x2, x3, ...] and [y1, y2, y3, ...] of the points on the n-th order of the Hilbert curve. To plot output, for example to see the 5-th order curve, users need to type "line [x,y]=order(5);line(x,y)" on the command line to find the [xi, yi] and connected lines.

(Listing 1)

```
function [x,y] = hilbert(n)
if n<=0                          %if n=0, hilbert(0) (0,0)   (1)
  x=0;
  y=0;
else                             % n=1, 2, etc
```

```
 [a,b]=hilbert(n-1);                % if n=1, [a,b]=hilbert(0),
                                    [a,b]=[0,0] from (1)
x=0.5*[-.5+b -.5+a .5+a  .5-b];    %x=0.5*[-.5+0 -.5+0 .5+0  .5-0];
y=0.5*[-.5+a  .5+b .5+b -.5-a];    %y=0.5*[-.5+0  .5+0 .5+0 -.5-0];
end
```

(Listing 2)

```
For n=1
else                               % n=1, 2, etc
 [a,b]=hilbert(n-1);               % if n=1, [a,b]=hilbert(0),
                                   [a,b]=[0,0] from (1)
x=0.5*[-.5+b -.5+a .5+a  .5-b];  %x=0.5*[-.5+0 -.5+0 .5+0  .5-0];
y=0.5*[-.5+a  .5+b .5+b -.5-a];  %y=0.5*[-.5+0  .5+0 .5+0 -.5-0];
end
```

(Listing 3)

```
For n=2
else                               % n=1, 2, etc
 [a,b]=hilbert(n-1);               % if n=2, [a,b]=hilbert(1),
x=0.5*[-.5+b -.5+a .5+a  .5-b];
y=0.5*[-.5+a  .5+b .5+b -.5-a];
end
```

(Output)

```
x=0.5*[-.5+0  -.5+0   .5+0    .5-0] = [-0.25   -0.25    0.25    0.25] =[a]
y=0.5*[-.5+0   .5+0    5+0   -.5-0]=  [-0.25    0.25    0.25   -0.25] =[b]
```

We can make use of these vector elements to move to the next step. For example

(Output)

```
  x=0.5*[-.5+b -.5+a .5+a .5-b];
   =0.5*[-.5+(-0.25)   -.5+(-0.25)   .5+(-0.25)   .5-(-0.25)
         -.5+(0.25)    -.5+(-0.25)   .5+(-0.25)   .5-(0.25)
         -.5+(0.25)    -.5+(0.25)    .5+(0.25)    .5-(0.25)
```

-.5+(-0.25) -.5+(0.25) .5+(0.25) .5-(-0.25)]

= [-0.375 -0.125 -0.125 -0.375

-0.375 -0.375 -0.125 -0.125

0.125 0.125 0.375 0.375

0.375 0.125 0.125 0.375]

y=0.5*[-.5+a .5+b .5+b -.5-a] can be found in the same way and the results are shown below.

Table 1(a). Data for the basic building block of the Hilbert curve (point 1-8)

Coord.	1	2	3	4	5	6	7	8
x	-0.375	-0.125	-0.125	-0.375	-0.375	-0.375	-0.125	-0.125
y	-0.375	-0.375	-0.125	-0.125	0.125	0.375	0.375	0.125

Table 1(b). Data for the basic building block of the Hilbert curve (points 9-16)

Coord.	9	10	11	12	13	14	15	16
x	0.125	0.125	0.375	0.375	0.375	0.125	0.125	0.375
y	0.125	0.375	0.375	0.125	-0.125	-0.125	-0.375	-0.375

Table 1(a) and (b) show the data for the basic building block of the Hilbert curve, which is an open square formed by connected lines. A complex pattern is made by the presented procedure recursively converting each line to a smaller version of the original open square.

The lines of each of the small squares are then converted to even smaller squares, and so on. After second iterations, we can have four smaller separate squares, and the vertices of each square are connected to form the next pattern formed by a single continuous line.

Open Square- First Iteration - Hilbert(1)

Table 2 shows the data for the basic building block of the Hilbert curve(1), which is an open square formed by connecting 3 lines.

Table 2. Hilbert(1)

Coord.	1	2	3	4
x	-0.25	-0.25	0.25	0.25
y	-0.25	0.25	0.25	-0.25

First, we define [x,y] as order(n) that shows the vector coordinates[x1, x2, x3, x4] and [y1, y2, y3, y4] of the points on the n1st order of the Hilbert curve.

xo=a, yo=b

x=0.5*[-.5+yo -.5+xo .5+xo .5-yo];

y=0.5*[-.5+xo .5+yo .5+yo -.5-xo];

for n=1

x=0.5*[-.5+0 -.5+0 .5+0 .5-0] = [-0.25 -0.25 0.25 0.25]

y=0.5*[-.5+0 .5+0 5+0 -.5-0]= [-0.25 -0.25 0.25 -0.25]

To plot output, for example to see the n-th order curve, user need to type "line [x,y]=order(n);line(x,y)" on the command line to find the [xi, yi] and connected lines.

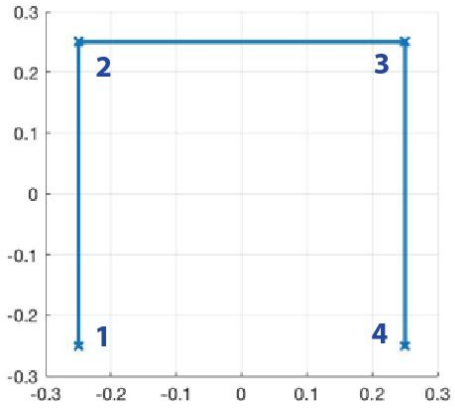


Figure 2. First Iteration - Hilbert(1)

We can make use of these vector elements to move to the next step, n=2. For example y=0.5*[-.5+a .5+b .5+b -.5-a] can be found in the same way and the results are shown in the table 1.

SFC after 2 iterations

After 2 iterations, end points are connected to each other. In Figure 3 below, 16 coordinates of x and y are shown.

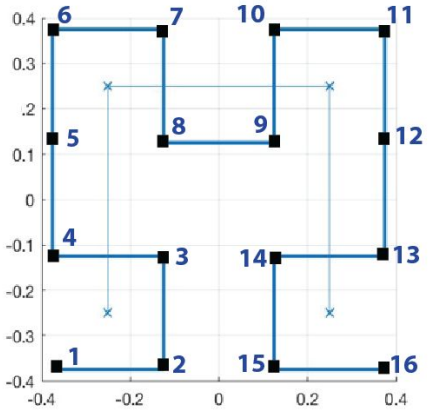


Figure 3. After 2 iterations

Coordinates of x and y of the 16 points after 2 iterations

In the Figure 4 below, coordinate values of x and y of the 16 points after 2 iterations are compared to each other.

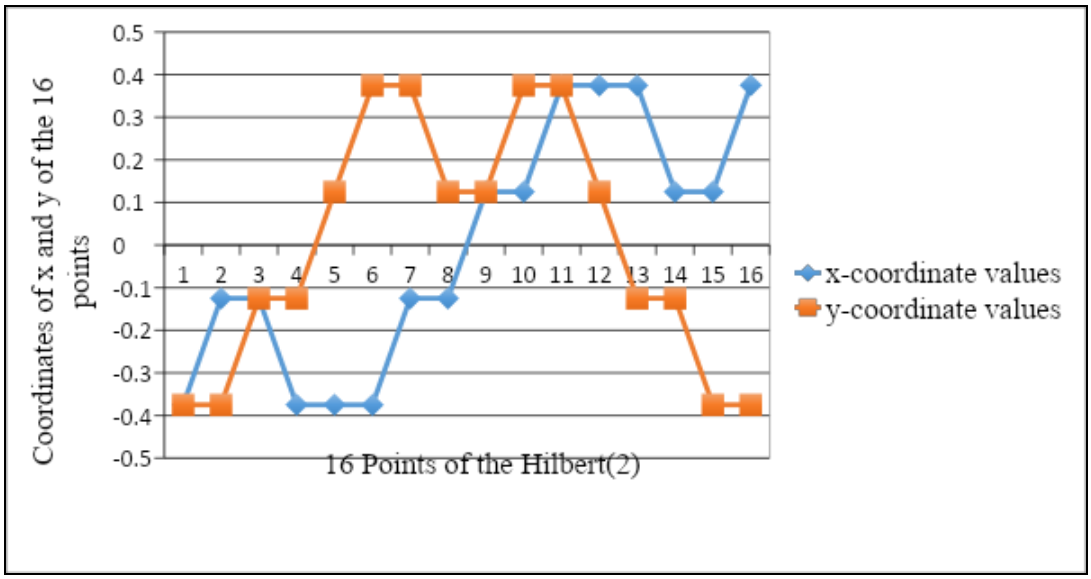
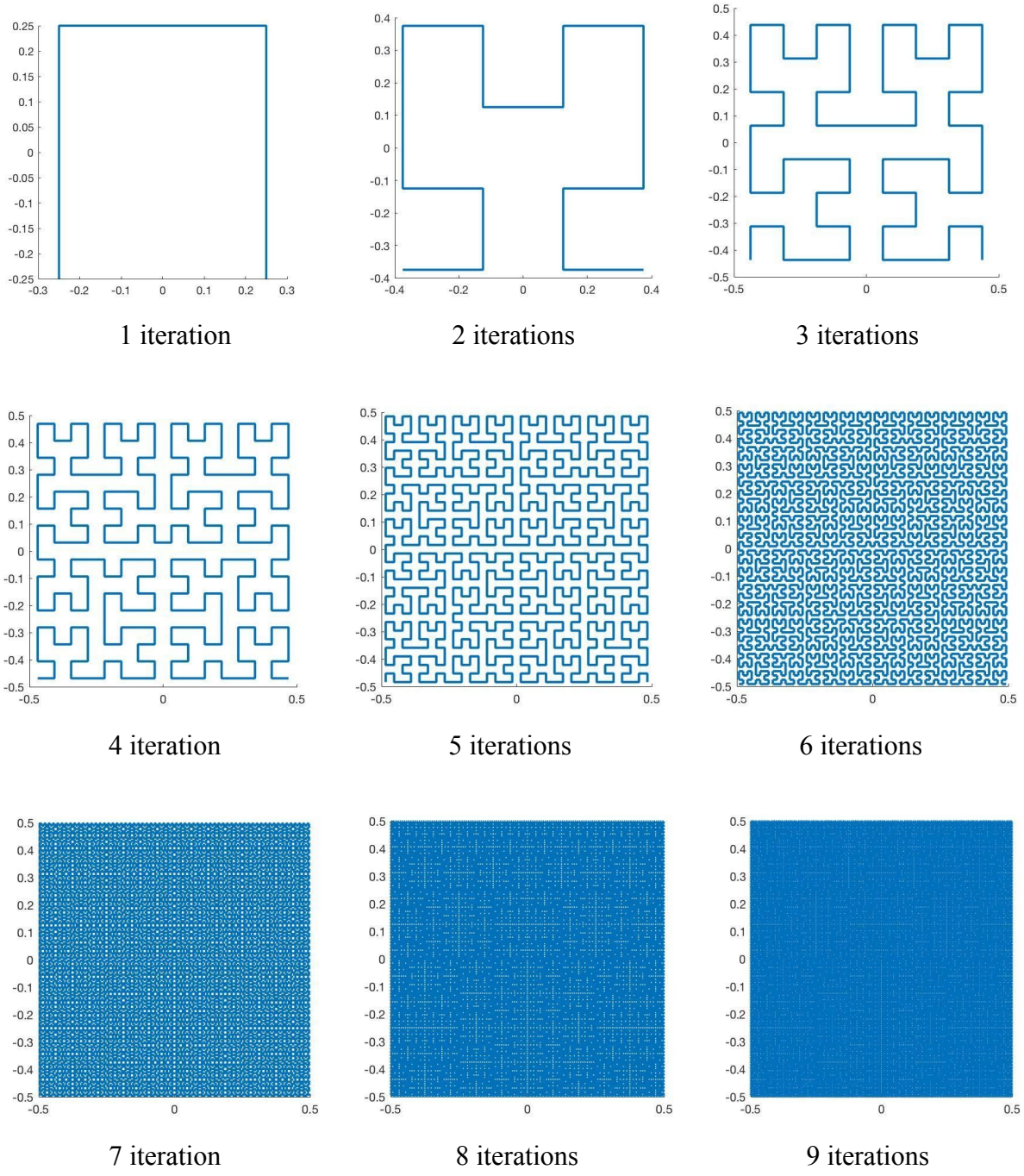


Figure 4. Comparison of coordinates of x and y of the 16 points after 2 iterations

Patterns: 1-9 iterations

The Figure 5 below shows the patterns with the basic building block of the Hilbert curve. All of the images of the first order to higher order are created using the presented procedure, recursively converting each line to a smaller version of the original open square.



1 iteration 2 iterations 3 iterations

4 iteration 5 iterations 6 iterations

7 iteration 8 iterations 9 iterations

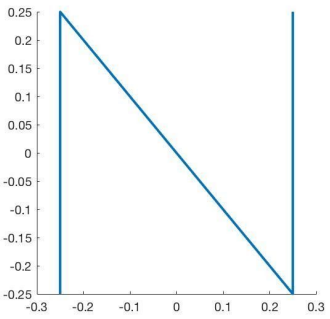
Figure 5. First nine stages in the generation of Hilbert's space-filling curve

After the first stage, the lines of each of the small squares are converted to even smaller squares, and so on. After second iterations, we can have four smaller separate squares, and the vertices of each square are connected to form the next pattern, which is formed by a single continuous line.

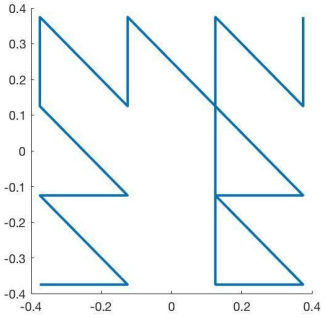
Data and Results

Case 1. N-shaped Geometry

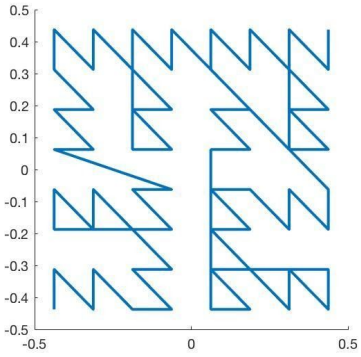
Figure 6 below shows the patterns with the basic building block of the presented N-shaped geometry formed by 3 connected lines as a starter.



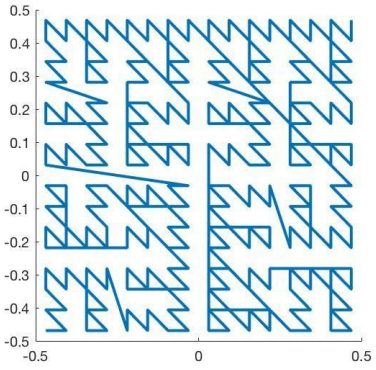
1 iteration



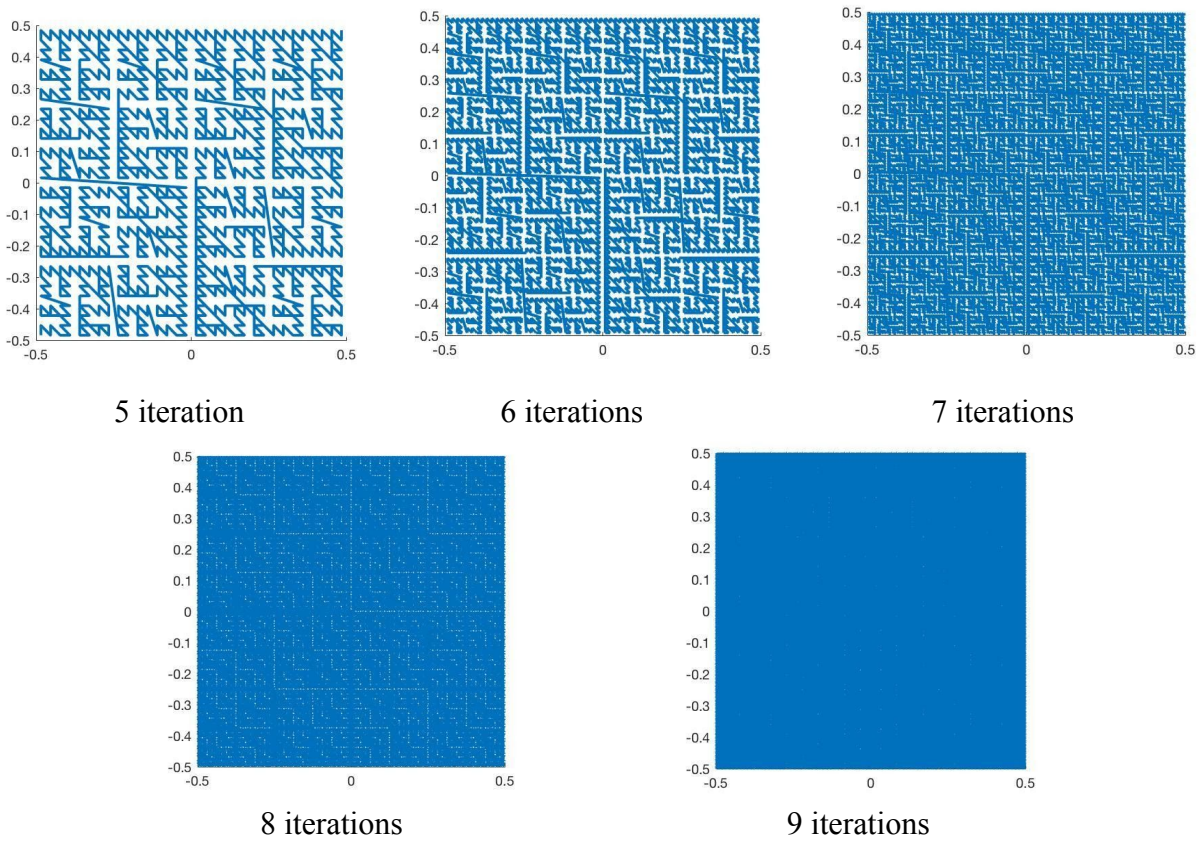
2 iterations



3 iterations



4 iterations



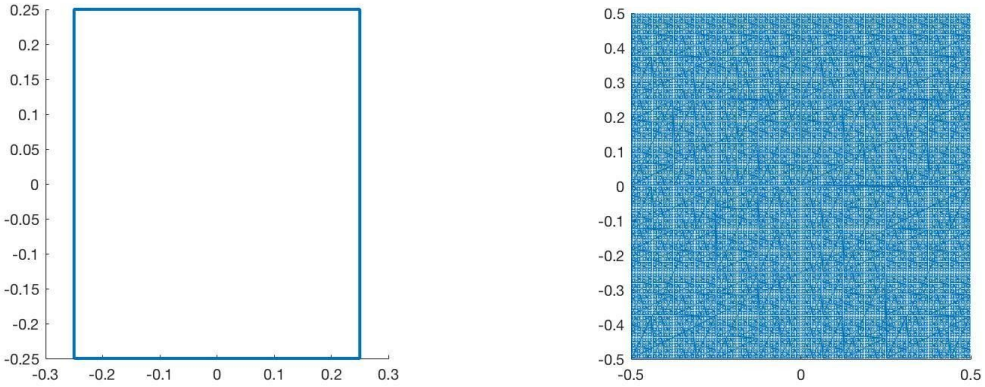
5 iteration 6 iterations 7 iterations

8 iterations 9 iterations

Figure 6. First nine stages in the generation of space-filling curve for the N-shaped geometry

Case 2. HFC (Hilbert's space-Filling Curve) Analogue: Closed Square (Rectangle)

Figure 7 below shows the patterns with the basic building block of the presented closed square (rectangle) formed by 4 connected lines as a starter.



1 iteration 7 iterations - Hilbert-2 (7)

Figure 7. HFC (Hilbert's space-Filling Curve) analogue with the closed square-rectangle

To find the level of SFC, an accurate distribution of black and white color is found by the Histogram. The Histogram for the Hilbert-2 (7), the closed square (rectangle) with the 7 iterations, is shown as follows:

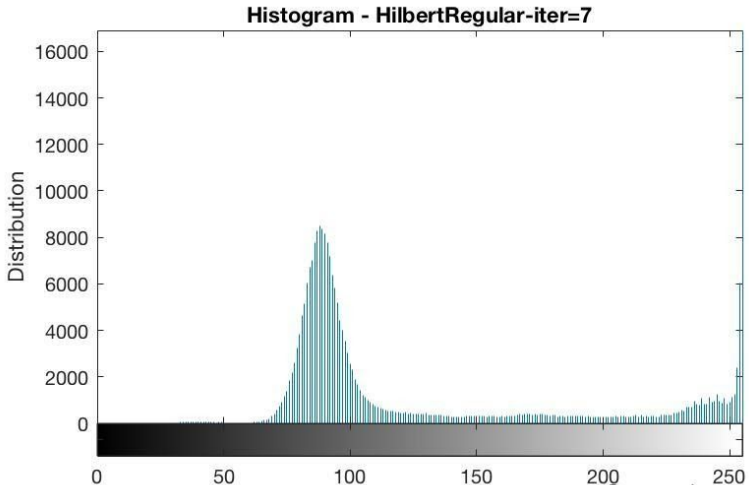


Figure 8. Histogram for the Hilbert-2 (7) with the closed square-rectangle

This stage Hilbert-2 (7) shows a level with mostly dark gray (the mean value is about 90). As the iteration increases, the mean value will decrease. The horizontal axis of the graph represents the tonal variations.

Case 3. HFC (Hilbert's space-Filling Curve) Analogue: Self-Intersecting Curve

Regular space-filling curves such as the Hilbert curves never self-intersect. The curves will completely cover a rectangular region of the plane if the recursion is infinite. The way they fill the region, however, is entirely decided by their elemental shape and the recursion rule [7].

In this section, a descriptive and empirical experiment to find new curves that do not self-intersect were suggested. Consider the following example of generating a curve, which is similar to Figure 3. But this time, we adjust the widths, which are not even causing a self-intersecting curve (thus it is not SFC).

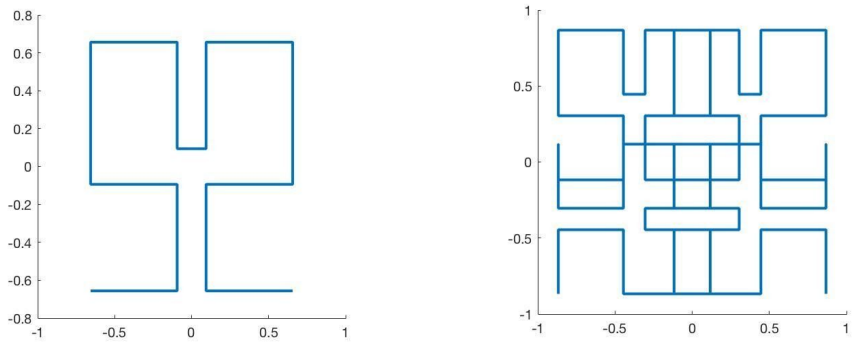


Figure 9. HFC (Hilbert's space-Filling Curve) Analogue: Self-Intersecting Curve

Modifying the Inputs (HFC Analogue)

Case 4A. Curve With Self-Intersecting and Not Filling the Space

x=.5*[-.6+yo -.4+xo .4+xo .6-yo];

y=.5*[-.6+xo .4+yo .4+yo -.6-xo];

Table 3. Hilbert(1)

Coord.	1	2	3	4
x	-0.3	-0.2	0.2	0.3
y	-0.3	0.2	0.2	-0.3

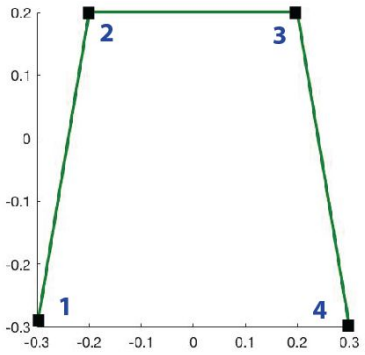


Figure 10. After nine iteration (Self-intersecting/Not filling space)

Case 4B. Curve With Self-Intersect and No Filling the Space

Figure 11 shows a second example of self-intersecting (not filling space).

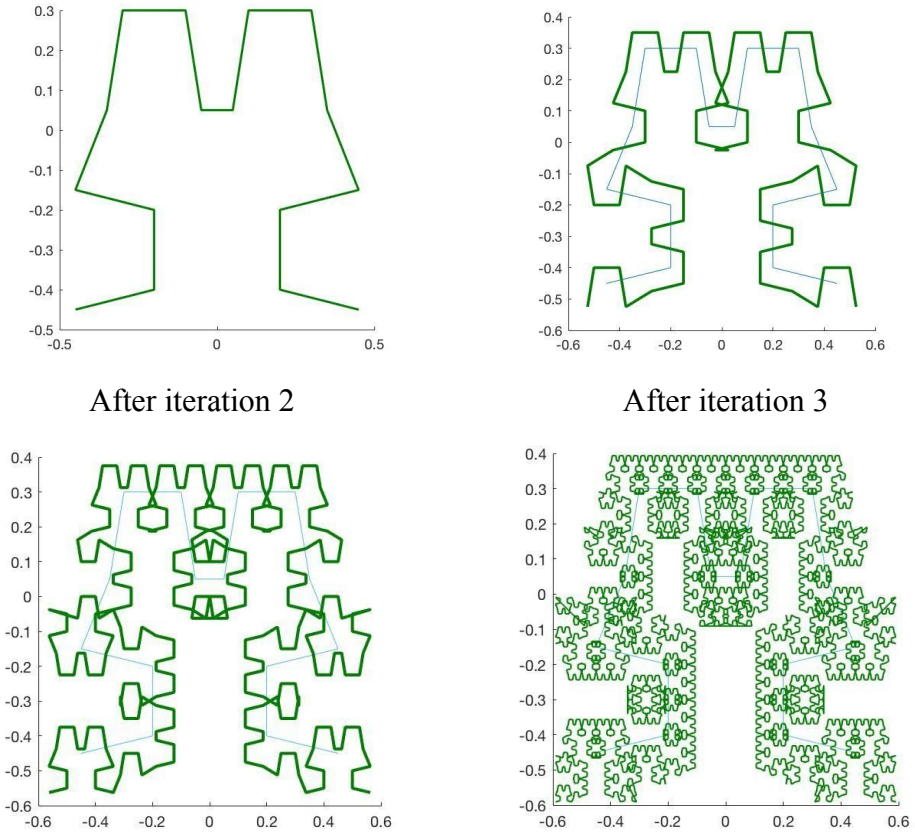


Figure 11. The 4 stages in the generation of HFC analogue (Self-intersecting)

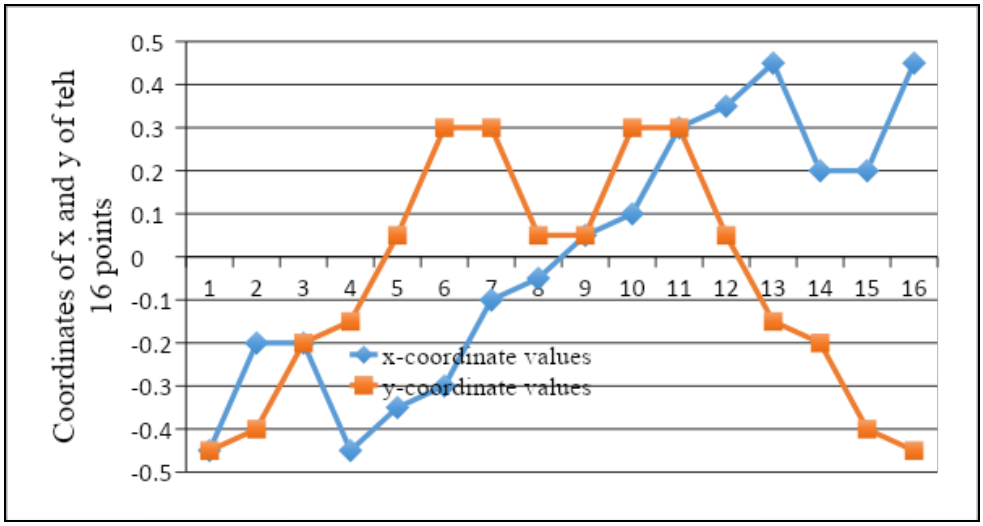


Figure 12. Coordinates of x and y of the 16 points after iteration 1 (Case 4B)

Table 3(a). Data for the basic building block of the Hilbert curve (point 1-8) (Case 4B)

Coord.	1	2	3	4	5	6	7	8
x	-0.45	-0.2	-0.2	-0.45	-0.35	-0.3	-0.1	-0.05
y	-0.45	-0.4	-0.2	-0.15	0.05	0.3	0.3	0.05

Table 3(b). Data for the basic building block of the Hilbert curve (point 9-16) (Case 4B)

Coord.	9	10	11	12	13	14	15	16
x	0.05	0.1	0.3	0.35	0.45	0.2	0.2	0.45
y	0.05	0.3	0.3	0.05	-0.15	-0.2	-0.4	-0.45

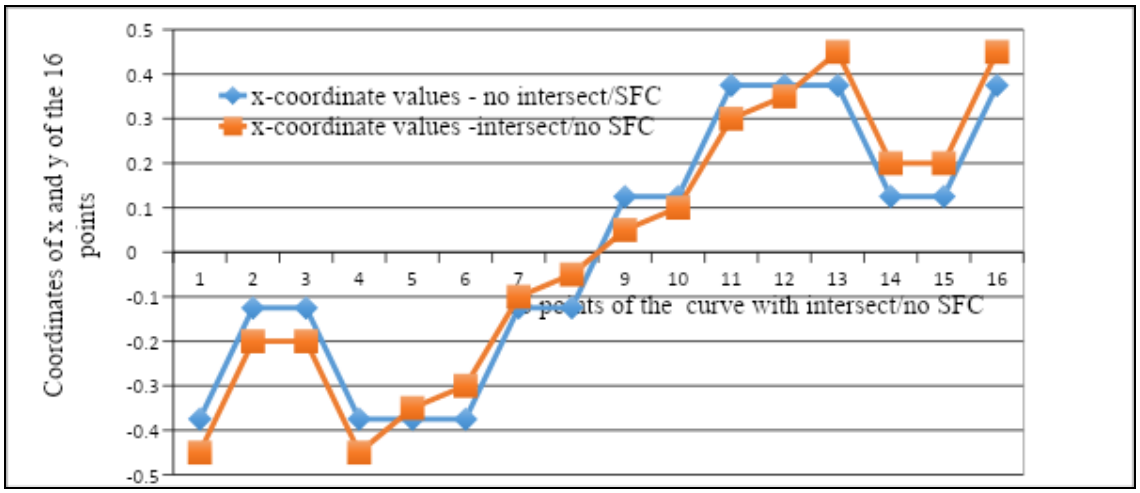


Figure 13. Comparisons of the x- coordinate values: No Intersect/ SFC and Intersect/ No SFC

Case 5. Curve With No Intersect And No Filling the Space HFC (Hilbert's space-Filling Curve) analogue

The following code calculates the y coordinate values. The new height becomes the half of the original height since the multiplier 0.5 changed to half (0.25) in the current code.

x=.5*[-.5+yo -.5+xo .5+xo .5-yo];

y=.25*[-.5+xo .5+yo .5+yo -.5-xo];

Figures 14 below shows the development of a modified curve with no self-intersect/No SFC. This time, we change the height and width from the HFC, causing not self-intersecting curve and not forming SFC.

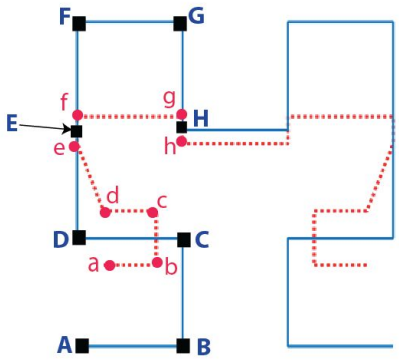
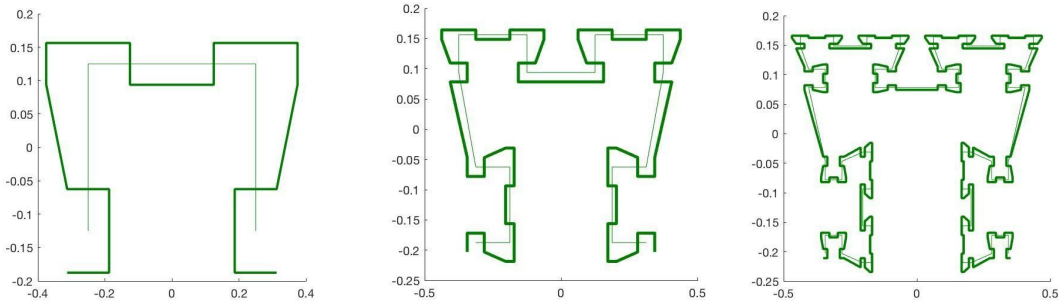


Figure 14. Curve with no self-intersect/No SFC



After iteration 2 After iteration 3 After iteration 4

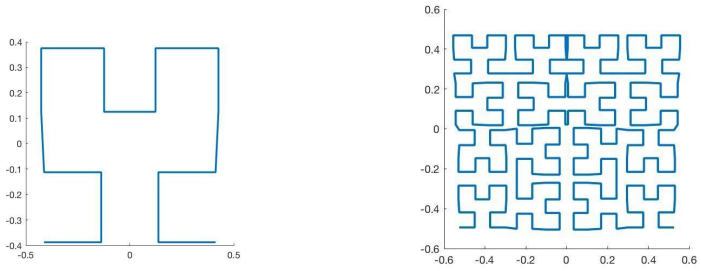
Figure 15. Stages in the generation of the curve with no self-intersect/No SFC

Case 6. Curve With Slight Intersect And Incomplete Filling the Space

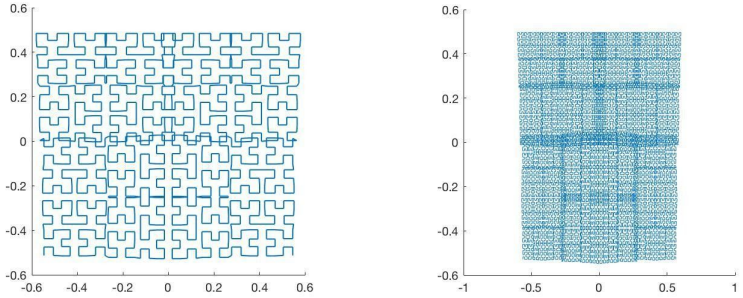
Now, the new width is adjusted 0.55 times the original width and the two vertical lines (4-6 and 11=13 in Figure 3) are not straight. The following code calculates the coordinate values.

x=.55*[-.5+yo -.5+xo .5+xo .5-yo];

y=.5*[-.5+xo .5+yo .5+yo -.5-xo];



Iteration 2 Iteration 3



Iteration 4 Iteration 5

Figure 16. Stages in the generation of the curve with no slight self-intersect/incomplete SFC

Discussions and Conclusions

Based on the building blocks using the existing SFC methods, filling the open square can be possible by connecting lines. Using an alternative algorithm studied in this paper, this project aimed that a 2D domain fully filled with pixels can be created using mathematical and computational calculations. MATLAB and JavaScript programming were developed for the project. Grids developed using the presented method can be applied to develop a digital image processing, numerical finite element methods, combinatorial optimization and solar cell engineering(grid generation).

Compared to the building blocks created by the existing SFC methods, such as Peano and Hilbert curves, present research showed that filling the 2D domain was available in different manners and with simpler operations with less computational calculations. Using simplified mathematical notations, less operational works such as less matrix multiplications were observed in this research.

1. We checked if our proposed curves, HFC analogues, would completely cover a rectangular region of the plane if the recursion increases. We used the control target as a regular space-filling curve, the Hilbert curve that never self-intersects.
2. Modified curves forming self-intersect, SFC, no self-intersect, and no SFC are all developed and tested in this research.

3. Using the presented SFC (Space Filling Curve) procedure recursively, new geometrical starters different from the smaller version of the original open square used in HFC were proposed.
4. The objective of this research project was to develop an alternative algorithm that can be efficiently used in SFC. One of the emphasis of this research was on the fast computation of space-filling curve orders. Using an algorithm studied in this paper, operational counts that cost while calculating the complex mathematical and computational calculations were reduced.
5. These results can be applied to the image compression and combinatorial optimization in an efficient manner.

References

1. Space-Filling Curves (Springer-Verlag), Sagan, Hans (1994), ISBN-13: 978-0387942650
2. Cannon, James W. "Group invariant Peano curves", Geometry & Topology
3. Mandelbrot, B. B. "Ch. 7: Harnessing the Peano Monster Curves" The Fractal Geometry of Nature, W. H. Freeman
4. Peano, G. (1890), "Sur une courbe, qui remplit toute une aire plane", Mathematische Annalen (in French), 36 (1): 157–160
5. marcin-chwedczuk.github.io/iterative-algorithm-for-drawing-hilbert-curve
6. www4.ncsu.edu/~njrose/pdfFiles/HilbertCurve.pdf
7. Benoit B. Mandelbrot: Fractals: Form, Chance, and Dimension, Free- man, ISBN 0716704730, 1977