

# Aplicaciones algorítmicas y estadísticas de space-filling curves

Proyecto final: Fractales MATE1301

Por: Luis Alejandro Rubiano, [la.rubiano@uniandes.edu.co](mailto:la.rubiano@uniandes.edu.co)

Profesor: Alexander Getmanenko, [a.getmanenko@uniandes.edu.co](mailto:a.getmanenko@uniandes.edu.co)

Departamento de Matemáticas, Universidad de Los Andes, Bogotá, Colombia

## Introducción e historia de las space-filling curves:

Parafraseando lo que dice Sagan en su libro Space-Filling Curves: En 1878, George Cantor demostró que una superficie finito dimensional (variedad suave o que localmente se parece lo suficiente a un espacio vectorial) sin importar sus dimensiones, tiene la misma cardinalidad que una recta en el espacio. Desde entonces las matemáticas no han vuelto a ser iguales después de este descubrimiento que se rebela contra la intuición.

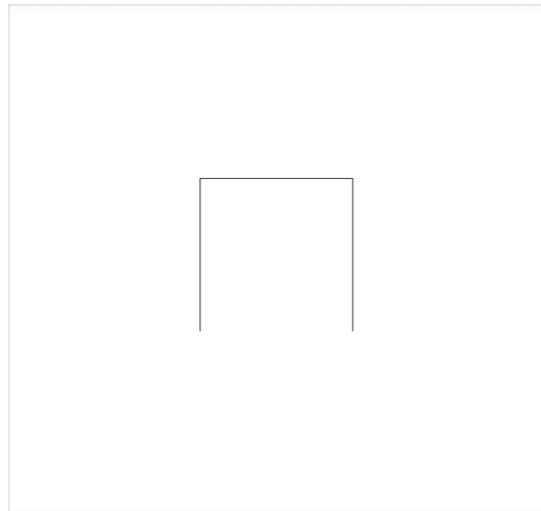
Este hallazgo implica que por ejemplo, el intervalo  $[0,1]$  puede ser mapeado biyectivamente en el cuadrado  $[0,1]^2$ . Una pregunta que surgió a continuación es: ¿pueden estas funciones ser continuas?. En 1879 Eugen Netto probó que estas funciones son necesariamente discontinuas.

Ahora, sin la condición de biyectividad, ¿Es posible es posible un mapeo sobreyectivo y continuo, de  $[0,1]$  a  $[0,1]^2$ ? La respuesta es que si, en 1890 Giuseppe Peano dio con la construcción de una de estas “curvas”. Años después, otros matemáticos crearon otras de estas curvas: David Hilbert en 1891 o Wacław Sierpiński en 1912. Ahora estas son las llamadas “space-filling curves” o curvas que llenan el espacio, un nombre muy autodescriptivo.

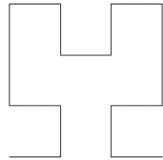
## Curva de Hilbert

La curva de Hilbert es una “space-filling curve” que se genera de manera recursiva. A continuación algunas imágenes, hechas por mi para la tarea 3 del curso, de las primeras aproximaciones a la curva de Hilbert.

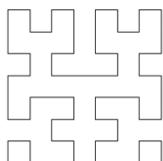
$n = 1$



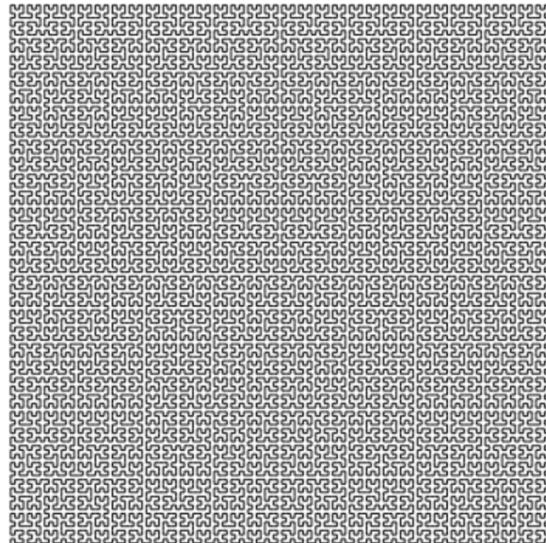
$n = 2$



$n = 3$



$n = 7$  (con ayuda de WolframAlpha)



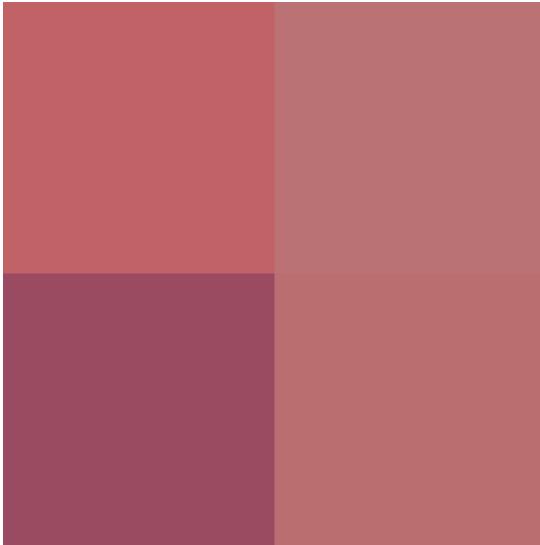
Todas las anteriores no son realmente curvas de Hilbert, sino que son aproximaciones a la curva de Hilbert. La verdadera curva de Hilbert ocurre con  $n = \infty$ , la cual es una curva que pasa por todos los puntos del cuadrado, a pesar de ser infinitamente delgada (algo completamente contra intuitivo).

La construcción de la curva de Hilbert se da al tomar la curva para  $n = 1$ , y para cada siguiente  $n$ , crear 4 copias de la curva, rotando 90 grados a la derecha la copia de abajo a la derecha, y 90 grados a la izquierda la copia de abajo a la izquierda, uniendo las 4 copias de manera que la curva sea continua, y que siempre inicie y acabe en el mismo punto.

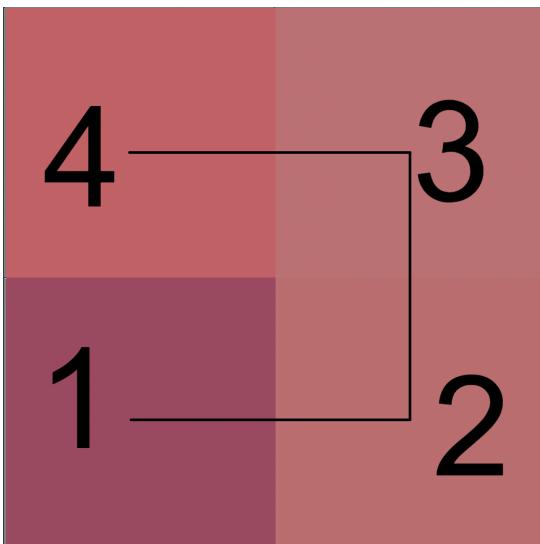
### **Aplicaciones de la curva de Hilbert: Mapeo de imágenes a un intervalo en N.**

(Tomando una metáfora usada por el matemático Grant Sanderson en su canal de YouTube 3Blue1Brown)

Supongamos que quieres hacer un software para ayudar a personas ciegas a ver imágenes a través de sus oídos. Entonces para iniciar a probar iniciamos con una imagen de resolución 2x2:



Podemos descomponerla siguiendo un patrón cualquiera que pase por toda la imagen como el siguiente:

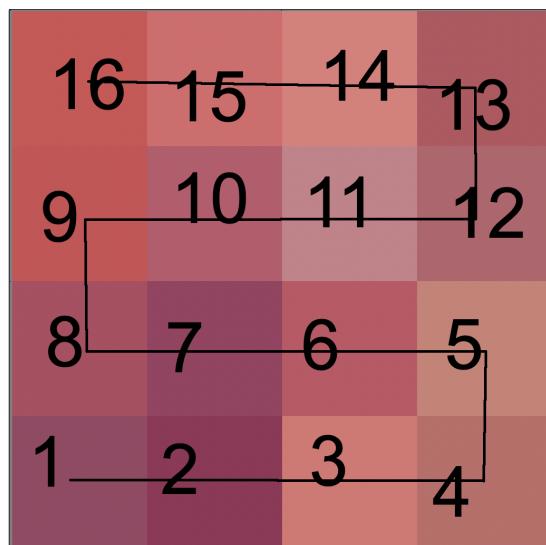
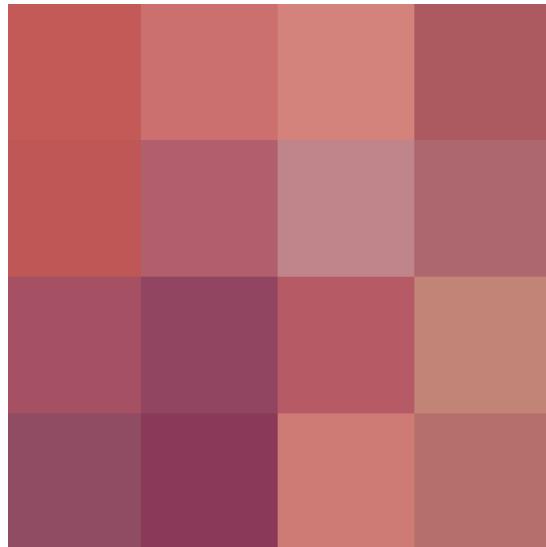


Podemos ponerle números a cada color, que luego se pueden traducir en sonido, estos quedarían algo así:

1	145	78	97
2	177	114	113
3	177	117	118
4	180	103	104

Supongamos ahora que la persona ciega ya se acostumbró a ver la imagen de esta forma, pero ahora queremos aumentar la

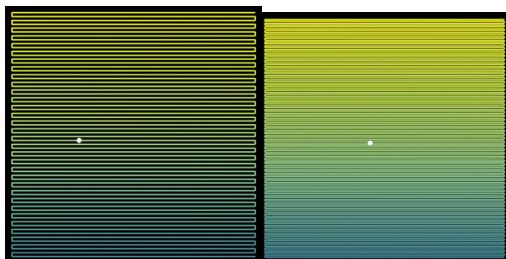
resolución de la imagen, ahora la imagen con una resolución de 4x4 quedaría:



Con

1	134	79	99
2	128	63	87
3	194	126	120
4	172	115	110
5	187	134	123
6	171	96	103
7	136	73	96
8	154	86	99
9	180	93	91
10	166	98	111
11	183	134	139
12	163	106	111
13	161	94	98
14	201	134	128
15	192	115	114
16	184	96	91

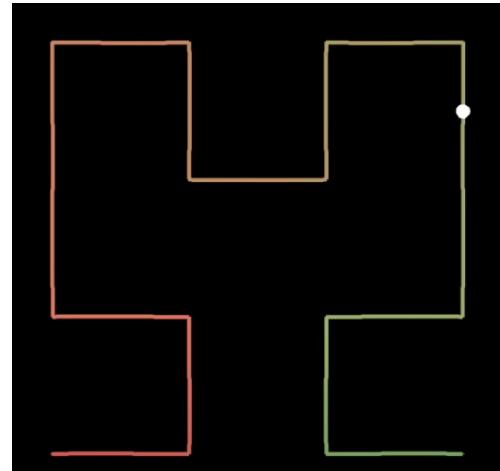
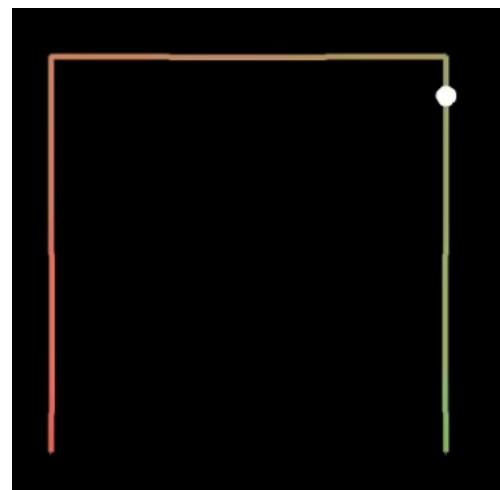
Ahora, las dos listas no guardan una relación entre si, lo que va a ser muy duro para la persona ciega, porque ya se había acostumbrado a interpretar el color y la posición en imágenes de 2x2. Esto se puede ver con algunas imágenes hechas por Grant Sanderson en su video “Hilbert's Curve: Is infinite math useful?”

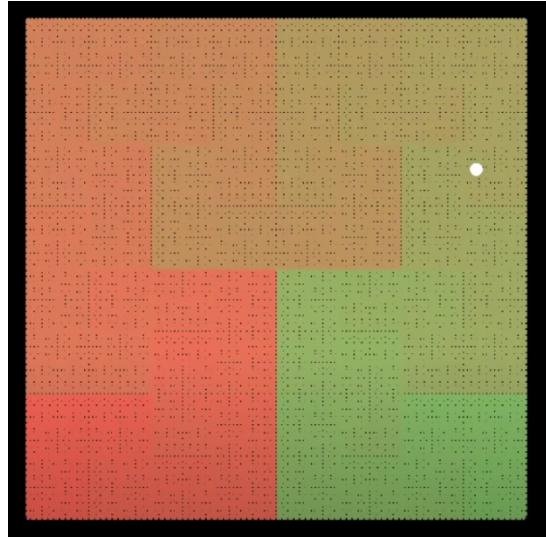
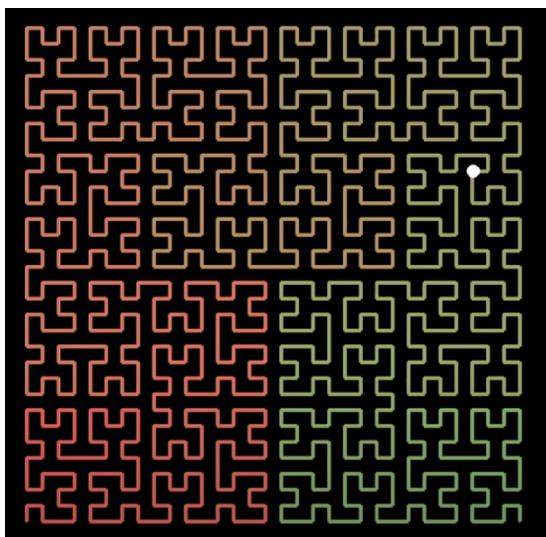
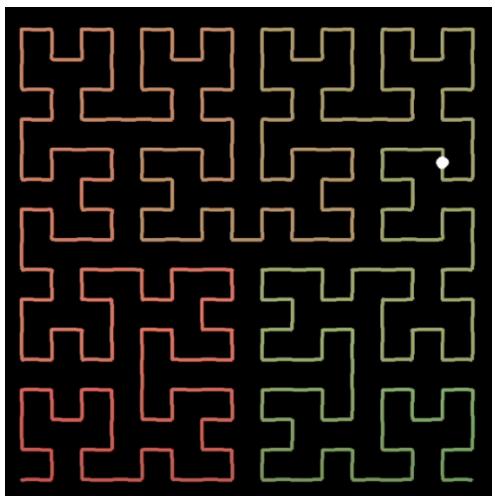
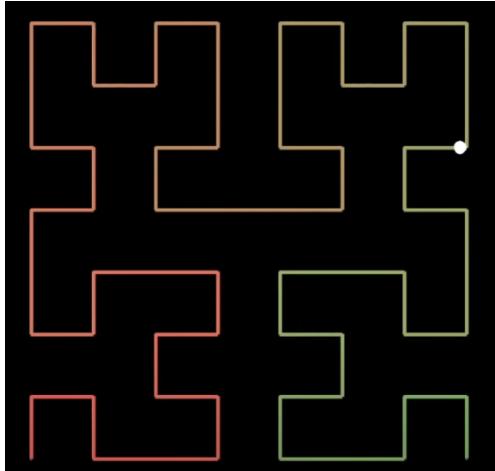


Como se puede ver, si tomamos un punto ubicado en una parte fija de nuestro “sonido” ej: el elemento 7/16 en una imagen de 4x4, al aumentar la resolución de la imagen, va a ser impredecible la posición de este elemento en la imagen.

¿Queda algo que podamos hacer? La respuesta es si: Space-filling curves!

Si en vez de enumerar los cuadrados como lo estábamos antes, lo hacemos de acuerdo al recorrido de la n-esima curva de Hilbert, podemos saber donde estos sonidos caerán, ya que van a converger a algo, ya que la curva de Hilbert está bien definida. Es decir siempre se va a poder determinar  $H(x) = (x,y)$ , que es lo mismo que decir que cuando  $n$  tiende a infinito en nuestra n-esima curva de Hilbert, su valor asociado va a converger a algo en  $R^2$ . Esto se ilustra a continuación:





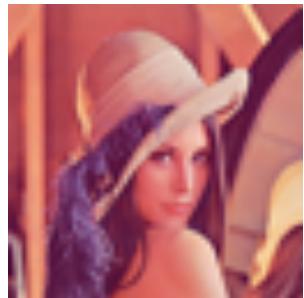
Ahora, si bien la curva de Hilbert real no puede ser sobreyectiva a causa de lo demostrado en 1879 por Eugen Netto, nuestra n-esima curva de Hilbert si lo puede ser porque es un mapeo de un intervalo finito en  $N$  a uno en  $N^2$ .

Ahora sabiendo esto, guardaremos como líneas de texto, diferentes resoluciones de una misma imagen, de manera que nuestras líneas de texto se vean parecidas entre si.

Lo primero que hice fue conseguir la imagen a usar en diferentes resoluciones: use la imagen Lenna Image, la imagen estándar para algoritmos de procesamiento de imágenes. Tomé los tamaños '2x2', '4x4', '8x8', '16x16', '32x32', '64x64', '128x128', '256x256', '512x512', '1024x1024', '2048x2048' y '4096x4096'. Estos tamaños fueron elegidos porque coinciden con el número de vértices en nuestras n-curvas de Hilbert para n's entre 1 y 10, más concretamente, el número de pixeles puede ser expresado como

$2^n * 2^n = 4^n$ . Como la imagen de Lenna solo puede encontrarse en dimensiones 512x512, usé una inteligencia artificial para subir la resolución a los tamaños mayores sin perder la calidad:

<https://letsenhance.io>, y para las dimensiones más pequeñas, bajé la resolución usando la librería Pillow de Python.

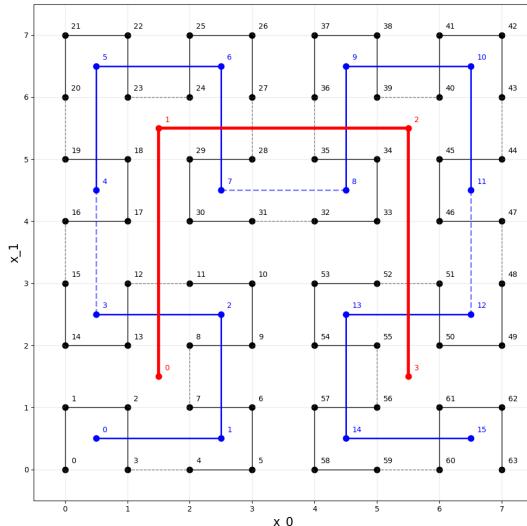


Lenna 64x64



Lenna 4096x4096

Una vez con esto, conseguí una librería de Python que me daba explícitamente los pixeles de una imagen que correspondían a los vértices de su  $n$ -ésima curva de Hilbert. La librería se llama Hilbert Curve y se puede acceder en el siguiente link: <https://github.com/galtay/hilbertcurve>. Esta librería permite hacer cosas de este estilo para dimensiones mayores a 2, pero en mi caso solo trabajé con la dimensión 2. A continuación algunos ejemplos de la librería:



```
>>> from hilbertcurve.hilbertcurve import HilbertCurve
>>> p=1; n=2
>>> hilbert_curve = HilbertCurve(p, n)
>>> distances = list(range(4))
>>> points = hilbert_curve.points_from_distances(distances)
>>> for point, dist in zip(points, distances):
>>>     print(f'point(h={dist}) = {point}')

point(h=0) = [0, 0]
point(h=1) = [0, 1]
point(h=2) = [1, 1]
point(h=3) = [1, 0]
```

Aquí  $n$  es la dimensión y  $p$  la  $p$ -ésima curva de Hilbert.

Una vez hecho esto, solo tocaba hacer listas de los colores (en hexadecimal) de los pixeles, de longitud 1048576 (1024x1024), para que se pudieran comparar entre si. Labor que conseguí al trabajar en el algoritmo y el uso de diferentes librerías de Python como colormap, pillow, math, pandas y numpy. La decodificación de las listas, otra vez en imágenes funcionaba de manera muy parecida.

Inicialmente quería que las listas fueran de longitud 16777216 (4096x4096) para mejores resultados, pero no conseguí ningún visualizador de CSV's gratuito que me permitiera hacerlo.

El resultado del código se ve a continuación:

```

1 ##### REQUIREMENTS: pip install hilbertcurve - colormap - pillow - math - pandas - numpy
2
3 D = 2 #Dimensión curva de hilbert
4 from hilbertcurve.hilbertcurve import HilbertCurve
5 from colormap import rgb2hex
6 from PIL import Image
7 from PIL import ImageColor
8 import math
9 import pandas as pd
10 import numpy as np
11
12 Lenna2 = Image.open("Lenna2.png") #1
13 Lenna4 = Image.open("Lenna4.png") #2
14 Lenna8 = Image.open("Lenna8.png") #3
15 Lenna16 = Image.open("Lenna16.png") #4
16 Lenna32 = Image.open("Lenna32.png") #5
17 Lenna64 = Image.open("Lenna64.png") #6
18 Lenna128 = Image.open("Lenna128.png") #7
19 Lenna256 = Image.open("Lenna256.png") #8
20 Lenna512 = Image.open("Lenna512.png") #9
21 Lenna1024 = Image.open("Lenna1024.jpg") #10
22 #####Lenna2048 = Image.open("Lenna2048.jpg") #11
23 #####Lenna4096 = Image.open("Lenna4096.jpg") #12
24 Imagenes = {2:Lenna2, 4:Lenna4, 8:Lenna8, 16:Lenna16, 32:Lenna32, 64:Lenna64, 128:Lenna128,
25 | | | 256:Lenna256, 512:Lenna512, 1024:Lenna1024}
26
27 print("\n"*5)
28 tamanos = [f'{2**n}x{2**n}' for n in range(1,10+1)]
29
30 def print_menu():
31     print("Presione 1 para seleccionar el tamaño de la imagen")
32     print("Presione 2 para decodificar un texto en su respectiva imagen")
33
34 def main():
35     print_menu()
36     Input = int(input(""))
37     if Input==1:
38         print("Seleccione uno entre los siguientes: ")
39         print(tamanos)
40         tamano = input("")
41         tamano = int(tamano.split("x")[0])
42         n = int(math.log(tamano,2)) #Aproximación enesima de la curva de Hilbert
43         Imagen = Imagenes[tamano]
44         Imagen = Imagen.convert("RGB")
45
46         hilbert_curve = HilbertCurve(n, D)
47         distances = list(range(2**n * 2**n))
48         points = hilbert_curve.points_from_distances(distances)
49         listaDeColores = []
50         for point, dist in zip(points, distances):
51             rgb = Imagen.getpixel(tuple(point))
52             hexa = rgb2hex(*rgb)
53             listaDeColores.append((dist,point,hexa))
54         listaFinal = []
55         for i in listaDeColores:
56             for _ in range(int(4**10/(2**n * 2**n))):
57                 listaFinal.append(i[2])

```

```

61     textfile = open("imagenes.csv", "a")
62     textfile.write(str(tamano)+"x"+str(tamano)+",")
63     for element in listaFinal:
64         textfile.write(element + ",")
65     textfile.write("\n")
66     textfile.close()
67
68 elif Input==2:
69     n = 10
70     hilbert_curve = HilbertCurve(n, D)
71     distances = list(range(2**n * 2**n))
72     points = hilbert_curve.points_from_distances(distances)
73
74     matriz = [[0 for x in range(1024)] for y in range(1024)]
75
76     print("Seleccione uno entre los siguientes: ")
77     print(tamanos)
78     tamano = input("")

79
80     df = pd.read_csv("imagenesT.csv", dtype=str)
81     saved_column = df[tamano]
82     for point, dist, color in zip(points, distances, saved_column):
83         matriz[point[1]][point[0]] = ImageColor.getcolor(color, "RGB")
84
85     array = np.array(matriz, dtype=np.uint8)
86     new_image = Image.fromarray(array)
87     new_image.save("Lenna"+tamano.split("x")[0]+D+".png")
88
89 main()

```

Para analizar los datos puse las listas correspondientes a las imágenes en diferentes resoluciones como columnas, y como eran valores en formato hexadecimal, con ayuda de código en Visual Basic logré asignarle el valor del color de cada celda a su respectiva celda, y de este modo comprar como los colores permanecían parecidos al desplazarse por la lista de cada una de las resoluciones. A continuación algunas imágenes de esto:

	A	B	C	D	E	F	G	H
1	2x2	4x4	8x8	16x16	128x128	256x256	512x512	1024x1024
2	#C06266	#C45A58	#DB7A67	#E4886F	#E1897F	#E2897E	#E2897D	#D8B179
3	#C06266	#C45A58	#DB7A67	#E4886F	#E1897F	#E2897E	#E2897D	#E48C82
4	#C06266	#C45A58	#DB7A67	#E4886F	#E1897F	#E2897E	#E2897D	#DD857B
5	#C06266	#C45A58	#DB7A67	#E4886F	#E1897F	#E2897E	#E2897D	#E68C83
6	#C06266	#C45A58	#DB7A67	#E4886F	#E1897F	#E2897E	#E2897D	#E38881
7	#C06266	#C45A58	#DB7A67	#E4886F	#E1897F	#E2897E	#E2897D	#E0887E
8	#C06266	#C45A58	#DB7A67	#E4886F	#E1897F	#E2897E	#E2897D	#D877D
9	#C06266	#C45A58	#DB7A67	#E4886F	#E1897F	#E2897E	#E2897D	#E18A80
10	#C06266	#C45A58	#DB7A67	#E4886F	#E1897F	#E2897E	#E2897D	#D8897E
11	#C06266	#C45A58	#DB7A67	#E4886F	#E1897F	#E2897E	#E2897D	#D8E87A
12	#C06266	#C45A58	#DB7A67	#E4886F	#E1897F	#E2897E	#E2897D	#DD897E
13	#C06266	#C45A58	#DB7A67	#E4886F	#E1897F	#E2897E	#E2897D	#D78378
14	#C06266	#C45A58	#DB7A67	#E4886F	#E1897F	#E2897E	#E2897D	#DA8379
15	#C06266	#C45A58	#DB7A67	#E4886F	#E1897F	#E2897E	#E2897D	#E58884
16	#C06266	#C45A58	#DB7A67	#E4886F	#E1897F	#E2897E	#E2897D	#E38C82
17	#C06266	#C45A58	#DB7A67	#E4886F	#E1897F	#E2897E	#E2897D	#E28881
18	#C06266	#C45A58	#DB7A67	#E4886F	#E1897F	#DF8981	#DF8985	#E89086
19	#C06266	#C45A58	#DB7A67	#E4886F	#E1897F	#DF8981	#DF8985	#DD857B
20	#C06266	#C45A58	#DB7A67	#E4886F	#E1897F	#DF8981	#DF8985	#E58083
21	#C06266	#C45A58	#DB7A67	#E4886F	#E1897F	#DF8981	#DF8985	#DF857C
22	#C06266	#C45A58	#DB7A67	#E4886F	#E1897F	#DF8981	#DF8880	#DD857B
23	#C06266	#C45A58	#DB7A67	#E4886F	#E1897F	#DF8981	#DF8880	#DF877D
24	#C06266	#C45A58	#DB7A67	#E4886F	#E1897F	#DF8981	#DF8880	#DD867C
25	#C06266	#C45A58	#DB7A67	#E4886F	#E1897F	#DF8981	#DF8880	#DE867C
26	#C06266	#C45A58	#DB7A67	#E4886F	#E1897F	#DF8981	#DF8880	#D78076
27	#C06266	#C45A58	#DB7A67	#E4886F	#E1897F	#DF8981	#DF8880	#DB847A
28	#C06266	#C45A58	#DB7A67	#E4886F	#E1897F	#DF8981	#DF8880	#DD897E
29	#C06266	#C45A58	#DB7A67	#E4886F	#E1897F	#DF8981	#DF8880	#DB847A
30	#C06266	#C45A58	#DB7A67	#E4886F	#E1897F	#DF8981	#DF8985	#DB847A
31	#C06266	#C45A58	#DB7A67	#E4886F	#E1897F	#DF8981	#DF8985	#E48C82
32	#C06266	#C45A58	#DB7A67	#E4886F	#E1897F	#DF8981	#DF8985	#D98278
33	#C06266	#C45A58	#DB7A67	#E4886F	#E1897F	#DF8981	#DF8985	#DD867C
34	#C06266	#C45A58	#DB7A67	#E4886F	#E1897F	#DF8982	#DF8985	#E48D83

	A	B	C	D	E	F	G	H
354906	#9A4A61	#8F4C63	#B6706E	#D68773	#E59677	#ED9F7D	#ECA07D	#E79877
354907	#9A4A61	#8F4C63	#B6706E	#D68773	#E59677	#ED9F7D	#ECA07D	#E89F7B
354908	#9A4A61	#8F4C63	#B6706E	#D68773	#E59677	#ED9F7D	#ECA07D	#E9A9E7A
354909	#9A4A61	#8F4C63	#B6706E	#D68773	#E59677	#ED9F7D	#ECA07D	#E99D79
354910	#9A4A61	#8F4C63	#B6706E	#D68773	#E59677	#ED9F7D	#ECA07D	#E9A9E7A
354911	#9A4A61	#8F4C63	#B6706E	#D68773	#E59677	#ED9F7D	#ECA07D	#E79877
354912	#9A4A61	#8F4C63	#B6706E	#D68773	#E59677	#ED9F7D	#ECA07D	#E99D78
354913	#9A4A61	#8F4C63	#B6706E	#D68773	#E59677	#ED9F7D	#ECA07D	#E89C78
354914	#9A4A61	#8F4C63	#B6706E	#D68773	#E59677	#E2957A	#E59E76	#E59773
354915	#9A4A61	#8F4C63	#B6706E	#D68773	#E59677	#E2957A	#E59E76	#EC9E7A
354916	#9A4A61	#8F4C63	#B6706E	#D68773	#E59677	#E2957A	#E59E76	#ECA07C
354917	#9A4A61	#8F4C63	#B6706E	#D68773	#E59677	#E2957A	#E59E76	#E59975
354918	#9A4A61	#8F4C63	#B6706E	#D68773	#E59677	#E2957A	#E59E76	#E69A76
354919	#9A4A61	#8F4C63	#B6706E	#D68773	#E59677	#E2957A	#E59E76	#E9A9E7A
354920	#9A4A61	#8F4C63	#B6706E	#D68773	#E59677	#E2957A	#E59E76	#E99D79
354921	#9A4A61	#8F4C63	#B6706E	#D68773	#E59677	#E2957A	#E59E76	#E89C78
354922	#9A4A61	#8F4C63	#B6706E	#D68773	#E59677	#E2957A	#DD8D80	#E09474
354923	#9A4A61	#8F4C63	#B6706E	#D68773	#E59677	#E2957A	#DD8D80	#E49878
354924	#9A4A61	#8F4C63	#B6706E	#D68773	#E59677	#E2957A	#DD8D80	#E09377
354925	#9A4A61	#8F4C63	#B6706E	#D68773	#E59677	#E2957A	#DD8D80	#D88E72
354926	#9A4A61	#8F4C63	#B6706E	#D68773	#E59677	#E2957A	#DD8D80	#E09377
354927	#9A4A61	#8F4C63	#B6706E	#D68773	#E59677	#E2957A	#DD8D80	#E49878
354928	#9A4A61	#8F4C63	#B6706E	#D68773	#E59677	#E2957A	#DD8D80	#E69778
354929	#9A4A61	#8F4C63	#B6706E	#D68773	#E59677	#E2957A	#DD8D80	#E49479
354930	#9A4A61	#8F4C63	#B6706E	#D68773	#E59677	#DE8F75	#D59176	#E69779
354931	#9A4A61	#8F4C63	#B6706E	#D68773	#E59677	#DE8F75	#D59176	#E59477
354932	#9A4A61	#8F4C63	#B6706E	#D68773	#E59677	#DE8F75	#D59176	#E89779
354933	#9A4A61	#8F4C63	#B6706E	#D68773	#E59677	#DE8F75	#D59176	#E8997A
354934	#9A4A61	#8F4C63	#B6706E	#D68773	#E59677	#DE8F75	#E99A7D	#E9A9C78
354935	#9A4A61	#8F4C63	#B6706E	#D68773	#E59677	#DE8F75	#E99A7D	#E69872
354936	#9A4A61	#8F4C63	#B6706E	#D68773	#E59677	#DE8F75	#E99A7D	#E99B75
354937	#9A4A61	#8F4C63	#B6706E	#D68773	#E59677	#DE8F75	#E99A7D	#E99B77
354938	#9A4A61	#8F4C63	#B6706E	#D68773	#E59677	#DE8F75	#E3987B	#E79774

	A	B	C	D	E	F	G	H
849681	#BB7275	#AD676E	#93A5B	#F62749	#60173D	#5E153C	#591737	#5D173C
849682	#BB7275	#AD676E	#93A5B	#F62749	#60173D	#5C143B	#5B1138	#5E183D
849683	#BB7275	#AD676E	#93A5B	#F62749	#60173D	#5C143B	#5B1138	#5E163C
849684	#BB7275	#AD676E	#93A5B	#F62749	#60173D	#5C143B	#5B1138	#5E163C
849685	#BB7275	#AD676E	#93A5B	#F62749	#60173D	#5C143B	#5B1138	#5C163B
849686	#BB7275	#AD676E	#93A5B	#F62749	#60173D	#5C143B	#5A1A3E	#5C163B
849687	#BB7275	#AD676E	#93A5B	#F62749	#60173D	#5C143B	#5A1A3E	#5B153A
849688	#BB7275	#AD676E	#93A5B	#F62749	#60173D	#5C143B	#5A1A3E	#611840
849689	#BB7275	#AD676E	#93A5B	#F62749	#60173D	#5C143B	#5A1A3E	#601A3F
849690	#BB7275	#AD676E	#93A5B	#F62749	#60173D	#5C143B	#611338	#641C42
849691	#BB7275	#AD676E	#93A5B	#F62749	#60173D	#5C143B	#611338	#601A3F
849692	#BB7275	#AD676E	#93A5B	#F62749	#60173D	#5C143B	#611338	#5D173C
849693	#BB7275	#AD676E	#93A5B	#F62749	#60173D	#5C143B	#611338	#611840
849694	#BB7275	#AD676E	#93A5B	#F62749	#60173D	#5C143B	#5D133E	#631D42
849695	#BB7275	#AD676E	#93A5B	#F62749	#60173D	#5C143B	#5D133E	#631B41
849696	#BB7275	#AD676E	#93A5B	#F62749	#60173D	#5C143B	#5D133E	#60183E
849697	#BB7275	#AD676E	#93A5B	#F62749	#60173D	#5C143B	#5D133E	#621A40
849698	#BB7275	#AD676E	#93A5B	#F62749	#60173D	#60173D	#5C1237	#5C183D
849699	#BB7275	#AD676E	#93A5B	#F62749	#60173D	#60173D	#5C1237	#561237
849700	#BB7275	#AD676E	#93A5B	#F62749	#60173D	#60173D	#5C1237	#551136
849701	#BB7275	#AD676E	#93A5B	#F62749	#60173D	#60173D	#5C1237	#561237
849702	#BB7275	#AD676E	#93A5B	#F62749	#60173D	#60173D	#5D1338	#520E33
849703	#BB7275	#AD676E	#93A5B	#F62749	#60173D	#60173D	#5D1338	#520E33
849704	#BB7275	#AD676E	#93A5B	#F62749	#60173D	#60173D	#5D1338	#5F1840
849705	#BB7275	#AD676E	#93A5B	#F62749	#60173D	#60173D	#5D1338	#581439
849706	#BB7275	#AD676E	#93A5B	#F62749	#60173D	#60173D	#621842	#5F173C
849707	#BB7275	#AD676E	#93A5B	#F62749	#60173D	#60173D	#621842	#631B40
849708	#BB7275	#AD676E	#93A5B	#F62749	#60173D	#60173D	#621842	#6E2449
849709	#BB7275	#AD676E	#93A5B	#F62749	#60173D	#60173D	#621842	#651B40
849710	#BB7275	#AD676E	#93A5B	#F62749	#60173D	#60173D	#631640	#5E163B
849711	#BB7275	#AD676E	#93A5B	#F62749	#60173D	#60173D	#631640	#5E183C
849712	#BB7275	#AD676E	#93A5B	#F62749	#60173D	#60173D	#631640	#5F193D
849713	#BB7275	#AD676E	#93A5B	#F62749	#60173D	#60173D	#631640	#5D173B
849714	#BB7275	#AD676E	#93A5B	#F62749	#60173D	#63183D	#681939	#5E183C

Para entender los datos, toca pensar que en la columna de 2x2, hay 4 colores distintos, en la columna de 4x4 hay 16 colores distintos, y así sucesivamente.

Al decodificar no hubo perdida de información, lo que confirma que nuestro mapeo fue una biyección.

## Otras aplicaciones de space-filling curves:

- Compresión de imágenes medicas:

Liang, Chen, Huang y Liu, de la Feng Chia University en Taiwan, en su paper Lossless Compression of Medical Images Using Hilbert Space-Filling Curves, explican como lograron comprimir datos de imágenes medicas sin perdida de información, usando métodos muy similares a los míos, con la diferencia de que a la lista de puntos resultante le aplicaron diferentes algoritmos de compresión de datos como los vistos en clase: LZ77 y otros.

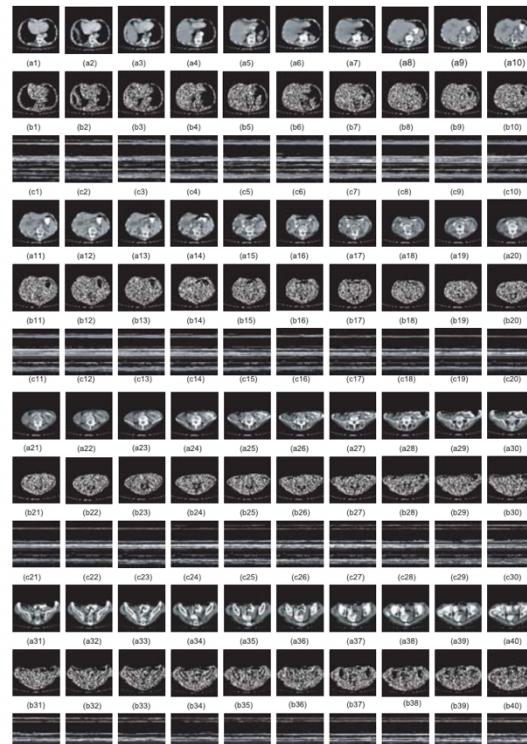


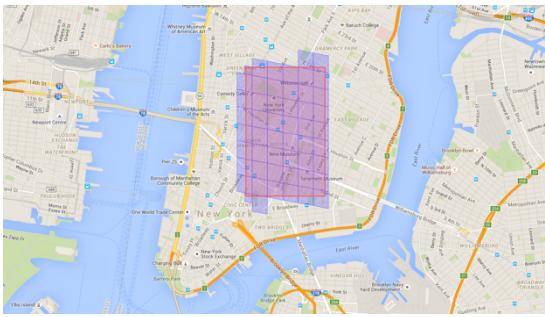
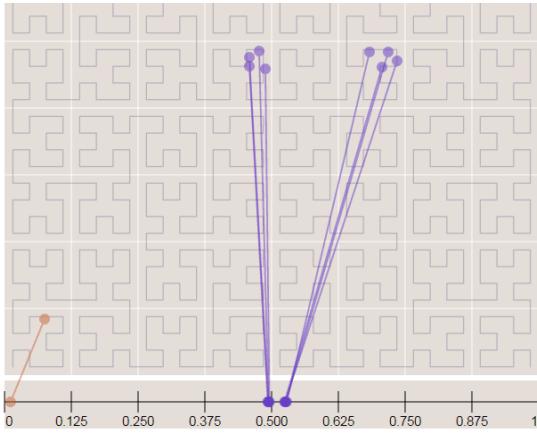
Figure 2: CT medical images

Table 2: Bit-per-pixel of each compression method and pre-processing operation

	None	(1)	(2)	(3)	(4)	(5)	(6)	(7)
None	8.03	3.41	3.16	2.97	3.61	3.30	2.84	5.17
(a)	8.03	3.43	3.09	3.00	3.61	3.31	2.82	6.44
(b)	8.03	3.38	2.56	2.85	3.30	3.01	2.38	5.51
(c)	8.03	3.47	2.65	2.81	3.37	3.12	2.45	6.72
(d)	8.03	3.40	2.49	2.90	3.30	3.03	2.35	6.21

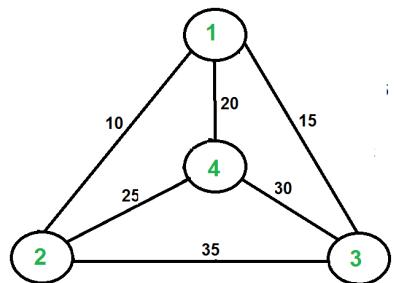
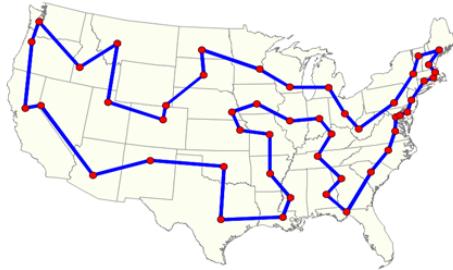
- Curvas de Hilbert en Google Maps y Street View:

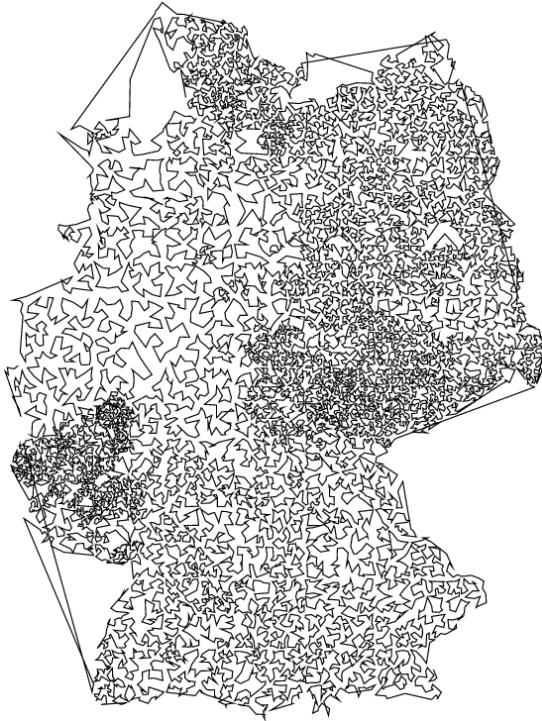
En Google Maps y en Google Street View se usan curvas de Hilbert para introducir la localidad del caché. Esto se puede entender como cuando te mueves un poco mientras miras el mapa, quieras moverte solo un poco en la memoria, que después de todo está ordenada linealmente. Lo anterior aprovecha la continuidad de las curvas de Hilbert, para preservar localidad, es decir que los puntos cercanos al punto A en el mapa, también estén cercanos al punto A en memoria.



- Space-filling curve de Sierpinski para solución al problema del vendedor viajero

El problema del vendedor viajero trata de un vendedor que debe visitar  $n$  ciudades (todas conectadas con todas), pasando solo una vez por cada una de ellas, minimizando el costo de su trayecto (puede ser en dinero/tiempo/etc).





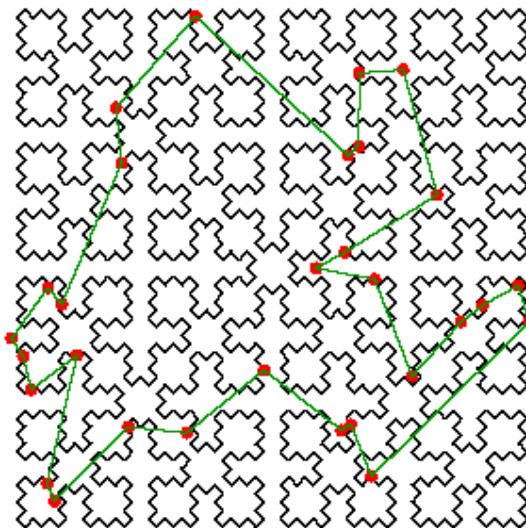
Este es un problema difícil de optimización y de ciencia computacional, en el que el único algoritmo conocido hasta el momento para dar con la solución mínima es el de fuerza bruta, que consiste en revisar todos los posibles casos, que como es  $|S_n|$ , su complejidad es  $O(n!)$ .

Mediante el uso de la Space-filling curve de Sierpinski, se puede construir una solución heurística al problema, que da una solución de aproximadamente 25% más que el óptimo (en una colección aleatoria de ubicaciones) con una complejidad temporal de  $O(n \log(n))$ .

La anterior heurística parte de una noción útil de las space-filling curves, la cual es que tiende a visitar todos los puntos de una región una vez que ha entrado en esa región. Por lo tanto, los puntos que están juntos en el plano tenderán a estar juntos en apariencia a lo largo de la curva.

Esta aplicación ya se ha usado en varios lugares en vida real, algunos ejemplos son:

- Construir un sistema de rutas para una empresa que da comidas a personas enfermas o mayores, en Atlanta, USA
- Dirigir la entrega de sangre de la Cruz Roja a los hospitales en Atlanta, USA
- Optimización de una máquina de dibujo de mapas en la Universidad de Tokio, Japón. Un ejemplo es un dibujo que pasó de demorarse 10 horas a una hora en realizar.



## Referencias:

1. Sagan, H. (1994). Space filling curves. Springer.
2. Grant Sanderson. (2016). Is Finite Math Useful?. 3Blue1Brown – YouTube.  
<https://www.youtube.com/watch?v=3s7h2MHQtxc>
3. Gabriel Altay. (2018). hilbertcurve. Github.  
<https://github.com/galtay/hilbertcurve#visuals>

4. Brian Hayes. (2013). Crinkly Curves. American Scientist Vol 101, p 178-183.  
<https://www.americanscientist.org/sites/americanscientist.org/files/2013416124139665-2013-05Hayes.pdf>
5. Bill Nulty y Paul Goldsman. (2012). Some Combinatorial Applications of Space-filling Curves. Georgia Institute of Technology.  
<https://www2.isye.gatech.edu/~jjb/research/mow/mow.html>
6. Christian S. Perone. (2015). Google's S2, geometry on the sphere, cells and Hilbert curve.  
<https://blog.christianperone.com/2015/08/googles-s2-geometry-on-the-sphere-cells-and-hilbert-curve/>
7. Liang, Chen, Huang y Liu. (N.A.). Lossless Compression of Medical Images Using Hilbert Space-Filling Curves. Feng Chia University, Taiwan.  
<https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.127.6563&rep=rep1&type=pdf>
8. AlphaOpt. (2021). What is the Traveling Salesman Problem?. Alpha-Opt – YouTube.  
<https://www.youtube.com/watch?v=1pmBjIZ20pE&t=97s>