**Ma. Elvira A. Oco**

**What is Hilbert Curves?**

A Hilbert curve (also known as a Hilbert space-filling curve) is a [continuous](http://en.wikipedia.org/wiki/Geometric_continuity) [fractal](http://en.wikipedia.org/wiki/Fractal) [space-filling curve](http://en.wikipedia.org/wiki/Space-filling_curve) first described by the German mathematician [David Hilbert](http://en.wikipedia.org/wiki/David_Hilbert) in 1891, as a variant of the space-filling curves discovered by [Giuseppe Peano](http://en.wikipedia.org/wiki/Giuseppe_Peano) in 1890.

Because it is space-filling, its [Hausdorff dimension](http://en.wikipedia.org/wiki/Hausdorff_dimension) is 2(precisely, its image is the unit square, whose dimension is 2 in any definition of dimension; its graph is a compact set homeomorphic to the closed unit interval, with Hausdorff dimension 2).

**What is the purpose of Hilbert Curves?**

Both the true Hilbert curve and its discrete approximations are useful because they give a mapping between 1D and 2D space that fairly well preserves locality. If (x,y) are the coordinates of a point within the unit square, and d is the distance along the curve when it reaches that point, then points that have nearby d values will also have nearby (x,y) values. The converse can't always be true. There will sometimes be points where the (x,y) coordinates are close but their d values are far apart. This is inevitable when mapping from a 2D space to a 1D space. But the Hilbert curve does a good job of keeping those d values close together much of the time. So the mappings in both direction do a fairly good job of maintaining locality. Because of this locality property, the Hilbert curve is widely used in computer science.

For example, the range of [IP addresses](http://en.wikipedia.org/wiki/IP_address) used by computers can be mapped into a picture using the Hilbert curve. Code to generate the image would map from 2D to 1D to find the color of each pixel, and the Hilbert curve is sometimes used because it keeps nearby IP addresses close to each other in the picture. A grayscale photograph can be converted to a dithered black and white image using thresholding, with the leftover amount from each pixel added to the next pixel along the Hilbert curve. Code to do this would map from 1D to 2D, and the Hilbert curve is sometimes used because it does not create the distracting patterns that would be visible to the eye if the order were simply left to right across each row of pixels. Hilbert curves in higher dimensions are an instance of a generalization of [Gray codes](http://en.wikipedia.org/wiki/Gray_code), and are sometimes used for similar purposes, for similar reasons. For multidimensional databases, Hilbert order has been proposed to be used instead of [Z order](http://en.wikipedia.org/wiki/Z-order_%28curve%29) because it has better locality-preserving behavior. For example, Hilbert curves have been used to compress and accelerate [R-tree](http://en.wikipedia.org/wiki/R-tree) indexes .They have also been used to help compress data warehouses.

**Example of use of** **Hilbert Curves?**

Given the variety of applications, it is useful to have algorithms to map in both directions. In many languages, these are better if implemented with iteration rather than recursion. The following [C](http://en.wikipedia.org/wiki/C_%28programming_language%29) code performs the mappings in both directions, using iteration and bit operations rather than recursion. It assumes a square divided into *n* by *n* cells, for *n* a power of 2, with integer coordinates, with (0,0) in the lower left corner, (*n*-1,*n*-1) in the upper right corner, and a distance *d* that starts at 0 in the lower left corner and goes to n^2-1in the lower-right corner.

//convert (x,y) to d

int xy2d (int n, int x, int y) {

int rx, ry, s, d=0;

for (s=n/2; s>0; s/=2) {

rx = (x & s) > 0;

ry = (y & s) > 0;

d += s \* s \* ((3 \* rx) ^ ry);

rot(s, &x, &y, rx, ry);

}

return d;

}

//convert d to (x,y)

void d2xy(int n, int d, int \*x, int \*y) {

int rx, ry, s, t=d;

\*x = \*y = 0;

for (s=1; s<n; s\*=2) {

rx = 1 & (t/2);

ry = 1 & (t ^ rx);

rot(s, x, y, rx, ry);

\*x += s \* rx;

\*y += s \* ry;

t /= 4;

}

}

//rotate/flip a quadrant appropriately

void rot(int n, int \*x, int \*y, int rx, int ry) {

if (ry == 0) {

if (rx == 1) {

\*x = n-1 - \*x;

\*y = n-1 - \*y;

}

//Swap x and y

int t = \*x;

\*x = \*y;

\*y = t;

}

}

The nonterminals are represented by an arrow and the terminals by heavy lines which remain a permanent part of the objects. The axiom AH and the rules RH1 and RH2 should be understood as embedded in the plane.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| AH. | http://eurologo.web.elte.hu/lectures/Image6.gif | RH1. | http://eurologo.web.elte.hu/lectures/Image7.gif | RH2. | http://eurologo.web.elte.hu/lectures/Image8.gif |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| We start with AH | http://eurologo.web.elte.hu/lectures/Image6.gif | by RH1 we obtain | http://eurologo.web.elte.hu/lectures/Image9.gif | and by RH2 we obtain H1 | http://eurologo.web.elte.hu/lectures/Image10.gif | ,but by RH1 |

|  |  |  |  |
| --- | --- | --- | --- |
| http://eurologo.web.elte.hu/lectures/Image11.gif | and by RH2 we obtain H2 | http://eurologo.web.elte.hu/lectures/Image12.gif | etc. |

It is evident that we use the rule RH2 only once - at the last step of a derivation. Let *hi*(*n*) be the procedure to draw the *n*-th Hilbert curve Hn and *ih*(*n*) the procedure to draw Hn in the opposite direction. Then we can easily translate our description into the following logo commands:

TO Hi :n   
IF :n = 0 [ STOP ]  
RT 90 Ih :n - 1 FD :h LT 90 Hi :n - 1 FD :h  
Hi :n - 1 LT 90 FD :h Ih :n - 1 RT 90  
END

TO Ih :n   
IF :n = 0 [ STOP ]  
LT 90 Hi :n - 1 FD :h RT 90 Ih :n - 1 FD :h  
Ih :n - 1 RT 90 FD :h Hi :n - 1 LT 90  
END

Because RT a = LT -a we can, introducing another parameter :d and denoting H 1 º Hi and H -1 º Ih, combine these two commands into a single command:

TO H :d :n   
IF :n = 0 [ STOP ]  
RT :d\*90 H (-:d) :n - 1 FD :h LT :d\*90 H :d :n - 1 FD :h  
H :d :n - 1 LT :d\*90 FD :h H (-:d) :n - 1 RT :d\*90  
END

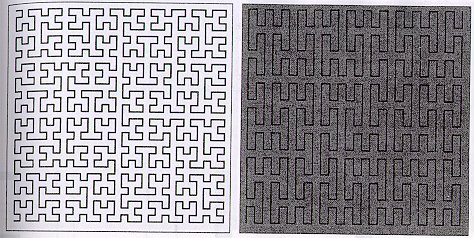


Figure 1 Hilbert curve and Peano curve

or finally, to avoid multiplication

TO Hilb :n :a :h  
IF :n = 0 [ STOP ]  
RT :a Hilb :n - 1 (-:a) :h FD :h LT :a Hilb :n - 1 :a :h   
FD :h  
Hilb :n - 1 :a :h LT :a FD :h Hilb :n - 1 (-:a) :h RT :a  
END

TO Hilbert  
PU SETPOS [-150 -150] PD SETPC [000 000 255]  
Hilb 5 90 10  
END

In a similar way we can develop drawing procedures also for the following curves. In the examples MicroSoft Windows Logo [11] is used. The PostScript pictures of space-filling curves, presented in figures, were produced by Logo2PS [1].

The Hilbert Curve can be expressed by a [rewrite system](http://en.wikipedia.org/wiki/Rewriting) ([L-system](http://en.wikipedia.org/wiki/L-system)).

**Alphabet** : A, B

**Constants** : F + -

**Axiom** : A

**Production rules**:

A → - B F + A F A + F B -

B → + A F - B F B - F A +

Here, *F* means "draw forward", *-* means "turn left 90°", and *+* means "turn right 90°" (see [turtle graphics](http://en.wikipedia.org/wiki/Turtle_graphics)).