C
Curves That Fill Space

A space-filling curve completely fills up part of space by passing through every point in that part. It does that by changing direction repeatedly. We will only discuss curves that fill up part of the two-dimensional plane, but the concept of a space-filling curve exists for any number of dimensions.

◇ **Exercise C.1:** Show an example of a space-filling curve in one dimension.

Several such curves are known and all are defined recursively. A typical definition starts with a simple curve C_0, shows how to use it to construct another, more complex curve C_1, and defines the final, space-filling curve as the limit of the sequence of curves $C_0, C_1, \ldots$.

C.1 The Hilbert Curve

(This discussion is based on the approach of [Wirth 76].) Perhaps the most familiar of these curves is the Hilbert curve, discovered by the great mathematician David Hilbert in 1891. The Hilbert curve [Hilbert 91] is the limit of a sequence $H_0, H_1, H_2 \ldots$ of curves, some of which are shown in Figure C.1. They are defined by the following:

0. H_0 is a single point.

1. H_1 consists of four copies of (the point) H_0, connected with three straight segments of length h at right angles to each other. Four orientations of this curve, labeled 1, 2, 3, and 4, are shown in Figure C.1a.

2. The next curve, H_2, in the sequence is constructed by connecting four copies of different orientations of H_1 with three straight segments of length $h/2$ (shown in bold in Figure C.1b). Again there are four possible orientations of H_2, and the one shown is #2. It is constructed of orientations 1223 of H_1, connected by segments

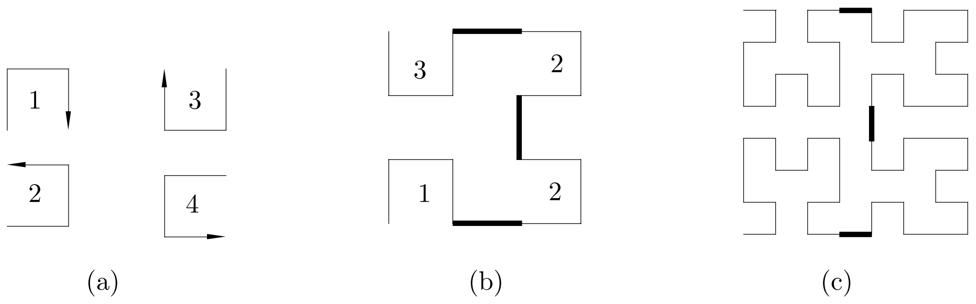

Figure C.1: Hilbert Curves of Orders 1, 2, and 3.

that go to the right, up, and to the left. The construction of the four orientations of H_2 is summarized in Table C.2.

Drawing this curve is thus done recursively. A procedure to draw orientation #4 of H_i is shown in Figure C.3. It makes four recursive calls to draw the four curves of order $i-1$, and draws straight segments between the calls, to connect them (the names A, B, C, and D are used instead of 1, 2, 3, and 4).

Figure C.4 is a complete Pascal program for H_n, compiled by the Metrowerks$^{\text{TM}}$ Macintosh Pascal compiler. Notice how the main program determines the initial values of the starting point (x, y) of the curve, and the segment size h. Variable $h0$ defines the size, in pixels, of the square containing the curve, and should be a power of 2.

Curve H_3 is shown in Figure C.1c. The particular curve shown is orientation 1223 of H_2.

Figures C.5, C.6 and C.7 show the Hilbert curves of orders 4, 5 and 6. It is easy to see how fast these curves become extremely complex.

C.2 The Sierpiński Curve

Another well-known space-filling curve is the Sierpiński curve. Figure C.8 shows curves S_1 and S_2, and Sierpiński has proved [Sierpiński 12] that the limit of the sequence $S_1, S_2, \ldots$ is a curve that passes through every point of the unit square $[0, 1] \times [0, 1]$.

To construct this curve, we need to figure out how S_2 is constructed out of four copies of S_1. The first thing that comes to mind is to follow the construction method used for the Hilbert curve, i.e., to take four copies of S_1, eliminate one edge in each, and connect them. This, unfortunately, does not work, since the Sierpiński curve is very different from the Hilbert curve. It is closed, and it has one orientation only. A better approach is to start with four parts that constitute four orientations of one open curve, and connect them with straight segments. The segments are shown dashed in Figure C.8. Notice how Figure C.8a is constructed of four orientations of a basic, three-part curve connected by four short, dashed segments. Figure C.8b is similarly constructed of four orientations of a complex, 15-part curve, connected by the same short, dashed segments. If we denote the four basic curves A, B, C, and D, then the basic construction rule of the Sierpiński curve is S: A$\searrow$B$\swarrow$C$\nwarrow$D$\nearrow$,

1: 2 ↑	1 →	1 ↓	4
2: 1 →	2 ↑	2 ←	3
3: 4 ↓	3 ←	3 ↑	2
4: 3 ←	4 ↓	4 →	1

Table C.2: The Four Orientations of H_2.

```
PROCEDURE D(i: INTEGER);
BEGIN
IF i>0 THEN BEGIN
  A(i-1); x:=x-h; MoveTo(x,y);
  D(i-1); y:=y-h; MoveTo(x,y);
  D(i-1); x:=x+h; MoveTo(x,y);
  C(i-1);
  END;
END (*A*);
```

Figure C.3: A Recursive Procedure.

```
PROGRAM Hilbert; (* A Hilbert curve *)
USES ScreenIO, Graphics;

CONST LB = 5; Width = 630; Height = 430;
(* LB=left bottom corner of window *)
n=6; (* n is the order of the curve*)
h0=8; (* h0 should be a power of 2 *)

VAR h,x,y,x0,y0: INTEGER;

PROCEDURE B (i: INTEGER); FORWARD;
PROCEDURE C (i: INTEGER); FORWARD;
PROCEDURE D (i: INTEGER); FORWARD;

PROCEDURE A(i: INTEGER);
BEGIN
IF i>0 THEN BEGIN
  D(i-1); x:=x-h; MoveTo(x,y);
  A(i-1); y:=y-h; MoveTo(x,y);
  A(i-1); x:=x+h; MoveTo(x,y);
  B(i-1);
  END;
END (*A*);

PROCEDURE B(i: INTEGER);
BEGIN
IF i>0 THEN BEGIN
  C(i-1); y:=y+h; MoveTo(x,y);
  B(i-1); x:=x+h; MoveTo(x,y);
  B(i-1); y:=y-h; MoveTo(x,y);
  A(i-1);
  END;
END (*B*);

PROCEDURE C(i: INTEGER);
BEGIN
IF i>0 THEN BEGIN
  B(i-1); x:=x+h; MoveTo(x,y);
  C(i-1); y:=y+h; MoveTo(x,y);
  C(i-1); x:=x-h; MoveTo(x,y);
  D(i-1);
  END;
END (*C*);

PROCEDURE D(i: INTEGER);
BEGIN
IF i>0 THEN BEGIN
  A(i-1); y:=y-h; MoveTo(x,y);
  D(i-1); x:=x-h; MoveTo(x,y);
  D(i-1); y:=y+h; MoveTo(x,y);
  C(i-1);
  END;
END (*D*);

BEGIN (* Main *)
  OpenGraphicWindow
   (LB,LB,Width,Height,'Hilbert curve');
  SetMode(paint);
  h:=h0; x0:=h DIV 2; y0:=x0; h:=h DIV 2;
  x0:=x0+(h DIV 2); y0:=y0+(h DIV 2);
  x:=x0+400; y:=y0+350; SetPen(x,y);
  A(n);

  ScBOL;
  ScWriteStr
   ('Hit a key & close window to quit');
  ScFreeze;
END.
```

Figure C.4: Pascal Program for a Hilbert Curve of Order i.

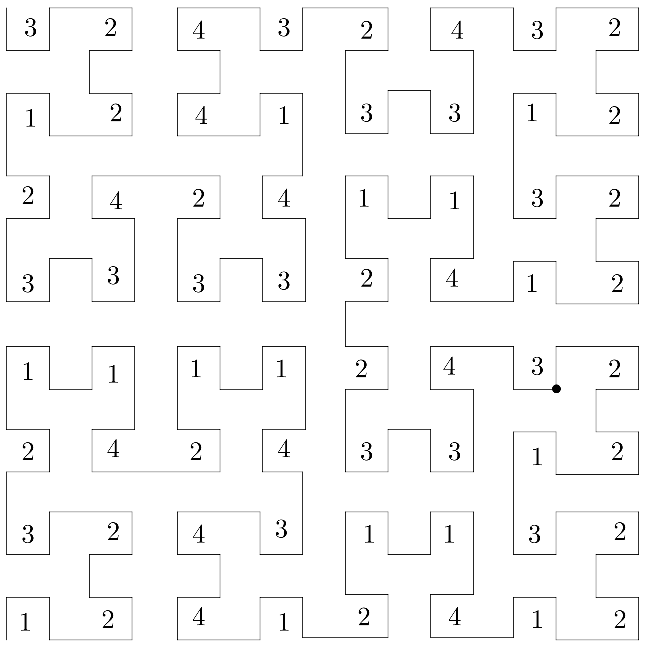

Figure C.5: Hilbert Curve of Order 4.

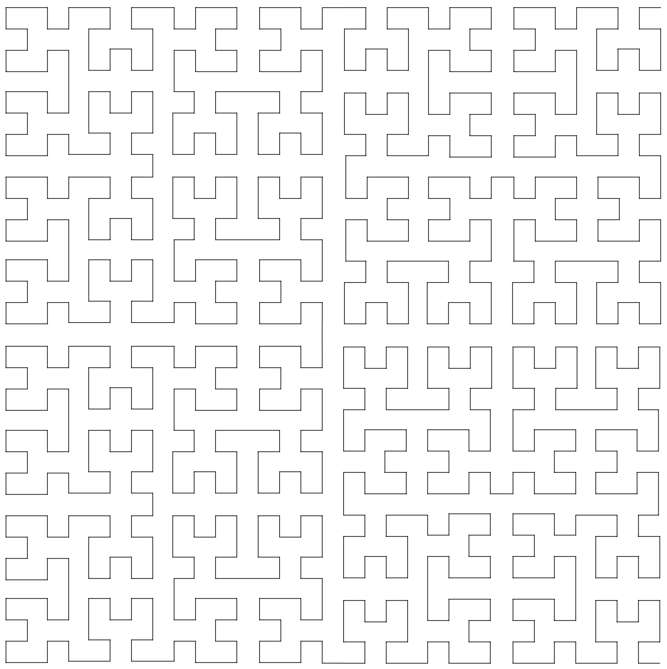

Figure C.6: Hilbert Curve of Order 5.

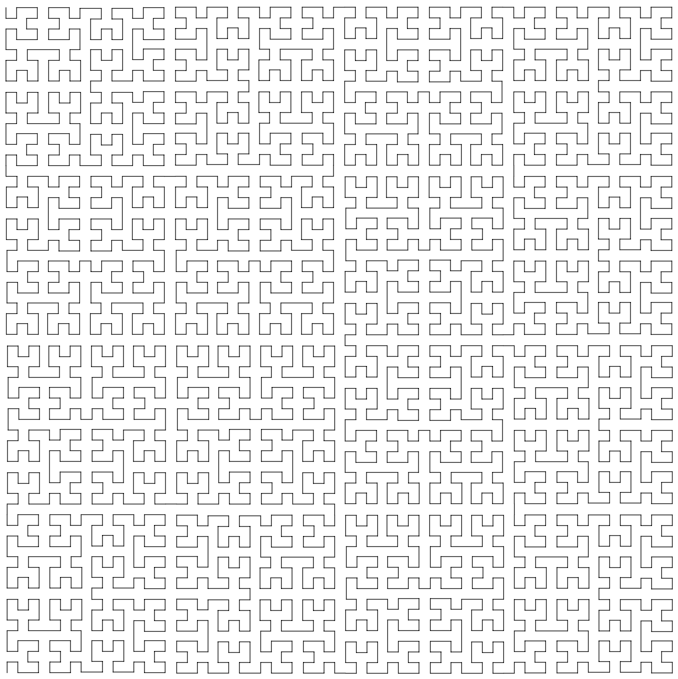
Figure C.7: Hilbert Curve of Order 6.

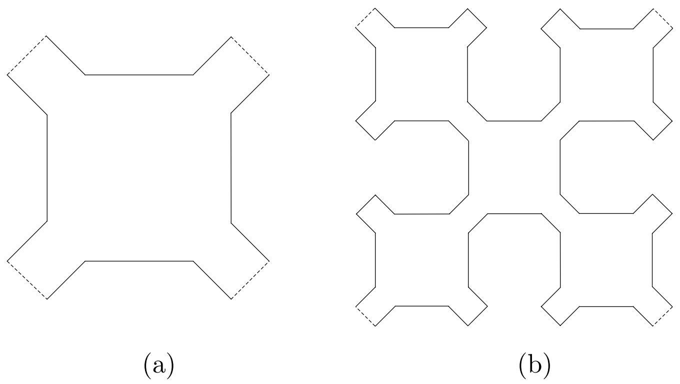
Figure C.8: Sierpiński Curves of Orders 1 and 2.

and the recursion rules are:

$$\begin{array}{ll} \text{A:} & \text{A}\searrow\text{B}\rightarrow\rightarrow\text{D}\nearrow\text{A} \\ \text{B:} & \text{B}\swarrow\text{C}\downarrow\downarrow\text{A}\searrow\text{B} \\ \text{C:} & \text{C}\nwarrow\text{D}\leftarrow\leftarrow\text{B}\swarrow\text{C} \\ \text{D:} & \text{D}\nearrow\text{A}\uparrow\uparrow\text{C}\nwarrow\text{D} \end{array} \tag{C.1}$$

Figure C.9 shows the five Sierpiński curves of orders 1 through 5 superimposed on each other. They were drawn by the Pascal program of Figure C.10.

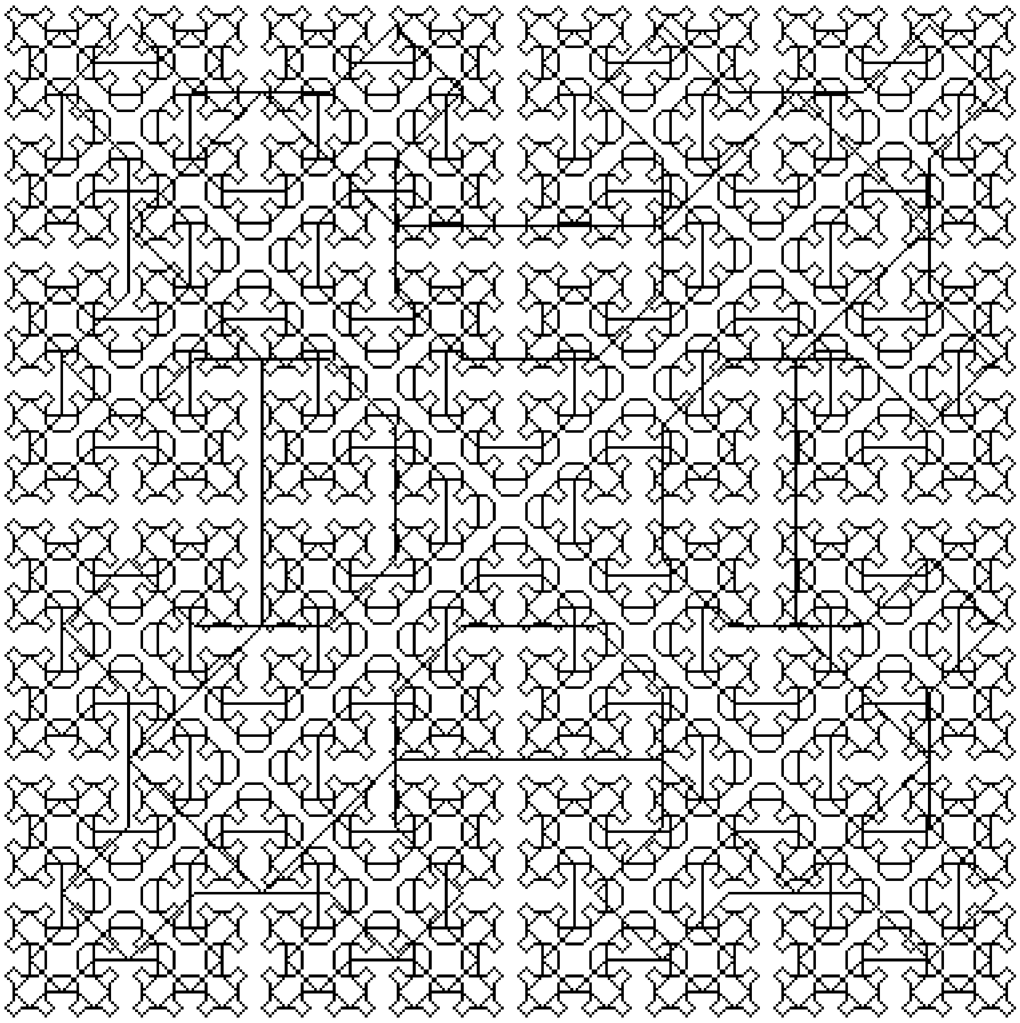

Figure C.9: Sierpiński Curves of Orders 1–5.

```
program Sierpinski;

USES ScreenIO, QuickDraw;

PROCEDURE SierpinskiP(size,number:word);
(* Adapted from Wirth, 2nd ed, p.115 *)

VAR
H, x, y:   INTEGER;

PROCEDURE B (level: word); FORWARD;
PROCEDURE C (level: word); FORWARD;
PROCEDURE D (level: word); FORWARD;

PROCEDURE A (level: word);
BEGIN
IF level > 0 THEN
BEGIN
A(level-1); Line(H,-H);
IF ScTellInput()>0 THEN BEGIN END;
B(level-1); Line(2*H,0);
IF ScTellInput()>0 THEN BEGIN END;
D(level-1); Line(H,H);
IF ScTellInput()>0 THEN BEGIN END;
A(level-1)
END
END {A};

PROCEDURE B (level: word);
BEGIN
IF level > 0 THEN
BEGIN
B(level-1); Line(-H,-H);
IF ScTellInput()>0 THEN BEGIN END;
C(level-1); Line(0,-2*H);
IF ScTellInput()>0 THEN BEGIN END;
A(level-1); Line(H,-H);
IF ScTellInput()>0 THEN BEGIN END;
B(level-1)
END
END {B};

PROCEDURE C (level: word);
BEGIN
IF level > 0 THEN
BEGIN
C(level-1); Line(-H,H);
IF ScTellInput()>0 THEN BEGIN END;
D(level-1); Line(-2*H,0);
IF ScTellInput()>0 THEN BEGIN END;
B(level-1); Line(-H,-H);
IF ScTellInput()>0 THEN BEGIN END;
C(level-1)
END
END {C};

PROCEDURE D (level: word);
BEGIN
IF level > 0 THEN
BEGIN
D(level-1); Line(H,H);
IF ScTellInput()>0 THEN BEGIN END;
A(level-1); Line(0,2*H);
IF ScTellInput()>0 THEN BEGIN END;
C(level-1); Line(-H,H);
IF ScTellInput()>0 THEN BEGIN END;
D(level-1)
END
END {D};

VAR
level: WORD;
BEXIT : BOOLEAN;

BEGIN (* SierpinskiP *)

level := 0;
H := size DIV 4;
x := 2 * H;
y := 3 * H;
    BEXIT := FALSE;
REPEAT
level := level + 1;; x := x - H;;
H := H DIV 2; y := y + H;;
MoveTo (x, y);
A (level); Line (H, -H);
IF ScTellInput()>0 THEN BEXIT:=TRUE;
B (level); Line (-H, -H);
IF ScTellInput()>0 THEN BEXIT:=TRUE;
C (level); Line (-H, H);
IF ScTellInput()>0 THEN BEXIT:=TRUE;
D (level); Line (H, H);
IF ScTellInput()>0 THEN BEXIT:=TRUE;
IF level = number THEN BEXIT:=TRUE;
 UNTIL BEXIT;
END {SierpinskiP};

VAR
ch:  CHAR;
prop:  termProp;

BEGIN
ScOpenWindow( 10, 10, 400, 400 );
ScGetProp (prop);
prop. showCurs := FALSE;
ScSetProp(prop); (*Hide alpha cursor*)
ScClear;
SierpinskiP(400,1);ScBeep (1);ScFreeze;
SierpinskiP(400,2);ScBeep (1);ScFreeze;
SierpinskiP(400,3);ScBeep (1);ScFreeze;
SierpinskiP(400,4);ScBeep (1);ScFreeze;
SierpinskiP(400,5);ScBeep (1);ScFreeze;
ScClose;
END {Sierpinski}.
```

Figure C.10: A Pascal Program for Sierpiński Curves of Orders 1–5.

◇ **Exercise C.2:** Figure I.6 shows three iterations of the Peano space-filling curve, developed in 1890. Use the techniques developed earlier for the Hilbert and Sierpiński curves, to describe how the Peano curve is constructed. (Hint: The curves shown are P_1, P_2, and P_3. The first curve, P_0 in this sequence is not shown.)

C.3 Traversing the Hilbert Curve

Space-filling curves are used in image compression (Section 4.29), which is why it is important to develop methods for a fast traversal of such a curve. Two approaches, both table-driven, are illustrated here for traversing the Hilbert curve.

The first approach [Cole 86] is based on the observation that the Hilbert curve H_i is constructed of four copies of its predecessor H_{i-1} placed at different orientations. A look at Figures C.1, C.5, C.6, and C.7 should convince the reader that H_i consists of 2^{2i} nodes connected with straight segments. The node numbers thus go from 0 to $2^{2i}-1$ and require $2i$ bits each. In order to traverse the curve we need a function that will compute the coordinates (x, y) of a node i from the node number i. The (x, y) coordinates of a node in H_i are i-bit numbers.

A look at Figure C.5 shows how successive nodes are initially located at the bottom left quadrant, and then move to the bottom right quadrant, the top right quadrant, and finally the top left one. This figure shows orientation #2 of the curve, so we can say that this orientation of H_i traverses quadrants 0, 1, 2, and 3, where quadrants are numbered $\left(\begin{smallmatrix}3&2\\0&1\end{smallmatrix}\right)$. It is now clear that the two leftmost bits of a node number determine its quadrant. Similarly, the next pair of bits in the node number determine its subquadrant within the quadrant, but here we run into the added complication that each subquadrant is placed at a different orientation in its quadrant. This approach thus uses Table C.11 to determine the coordinates of a node from its number.

Bit			Next	Bit			Next	Bit			Next	Bit			Next
pair	x	y	table	pair	x	y	table	pair	x	y	table	pair	x	y	table
00	0	0	2	00	0	0	1	00	1	1	4	00	1	1	3
01	1	0	1	01	0	1	2	01	0	1	3	01	1	0	4
10	1	1	1	10	1	1	2	10	0	0	3	10	0	0	4
11	0	1	4	11	1	0	3	11	1	0	2	11	0	1	1
	(1)				(2)				(3)				(4)		

Table C.11: Coordinates of nodes in H_i.

As an example, we compute the xy coordinates of node 109 (the 110th node) of orientation #2 of H_4. The H_4 curve has $2^{2\cdot 4} = 256$ nodes, so node numbers are 8 bits each, and $109 = 01101101_2$. We start with Table C.11(1). The 2 leftmost bits of the node number are 01, and table (1) tells us that the x coordinate start with 1, the y coordinate, with 0, and we should continue with table (1). The next pair of bits is 10, and table (1) tells us that the next bit of x is 1, the next bit of y is 1, and we should stay with table (1). The third pair of bits is 11, so table (1) tells us that the next bit of x is 0, the next bit of y is 1, and we should move to table (4). The last pair of bits is 01, and table (4) tells us to append 1 and 0 to the coordinates of x and y, respectively. The coordinates are thus $x = 1101 = 13$, $y = 0110 = 6$, as can be verified directly from Figure C.5 (the small circle).

It is also possible to transform a pair of coordinates (x, y), each in the range $[0, 2^i - 1]$, to a node number in H_i by means of Table C.12.

xy pair	Int. pair	Next table	xy pair	Int. pair	Next table	xy pair	Int. pair	Next table	xy pair	Int. pair	Next table
00	00	2	00	00	1	00	10	3	00	10	4
01	11	4	01	01	2	01	01	3	01	11	1
10	01	1	10	11	3	10	11	2	10	01	4
11	10	1	11	10	2	11	00	4	11	00	3
	(1)			(2)			(3)			(4)	

Table C.12: Node Numbers in H_i.

◇ **Exercise C.3:** Use Table C.12 to compute the node number of the H_4 node whose coordinates are $(13, 6)$.

The second approach to Hilbert curve traversal uses Table C.2. Orientation #2 of the H_2 curve shown in Figure C.1(b) is traversed in order 1223. The same orientation of the H_3 curve of Figure C.1(c) is traversed in 2114 1223 1223 4332, but Table C.2 tells us that 2114 is the traversal order for orientation #1 of H_2, 1223 is the traversal for orientation #2 of H_2, and 4332 is for orientation #3. The traversal of orientation #2 of H_3 is thus also based on the sequence 1223. Similarly, orientation #2 of H_4 is traversed (Figure C.5) in the order

$$\begin{array}{ll} 1223\ 2114\ 2114\ 3441 & 2114\ 1223\ 1223\ 4332 \\ 2114\ 1223\ 1223\ 4332 & 3441\ 4332\ 4332\ 1223, \end{array}$$

which is reduced to 2114 1223 1223 4332, which in turn is reduced to the same sequence 1223.

The idea is therefore to create the traversal order for orientation #2 of H_i by starting with the sequence 1223 and recursively expanding it $i - 1$ times, using Table C.2.

◇ **Exercise C.4:** (Easy.) Show how to apply this method to traversing orientation #1 of H_i.

A MATLAB function `hilbert.m` to compute the traversal of the curve is available at [Matlab 99]. It has been written by Daniel Leo Lau (`lau@ece.udel.edu`). The call `hilbert(4)` produces the 4×4 matrix

$$\begin{array}{cccc} 5 & 6 & 9 & 10 \\ 4 & 7 & 8 & 11 \\ 3 & 2 & 13 & 12 \\ 0 & 1 & 14 & 15 \end{array}.$$

C.4 Traversing the Peano Curve

The Peano curves P_0, P_1, and P_2 of Figure Ans.62 have 1, 3^2, and 3^4 nodes, respectively. In general, P_n has 3^{2n} nodes, numbered $0, 1, 2, \ldots, 3^{2n} - 1$. This suggests that the Peano curve [Peano 90] is somehow based on the number 3, in contrast to the Hilbert curve, which is based on 2. The coordinates of the nodes vary from $(0, 0)$ to $(n-1, n-1)$. It turns out that there is a correspondence between the node numbers and their coordinates [Cole 85], which uses base-3 reflected Gray codes (Section 4.2.1)

A reflected Gray code [Gray 53] is a permutation of the i-digit integers such that consecutive integers differ by one digit only. Here is one way to derive these codes for binary numbers. Start with $i = 1$. There are only two 1-bit digits, namely, 0 and 1, and they differ by 1 bit only. To get the RGC for $i = 2$ proceed as follows:

1. Copy the sequence $(0, 1)$.
2. Append (on the left or on the right) a 0 bit to the original sequence and a bit of 1 to the copy. The result is $(00, 01)$, $(10, 11)$.
3. Reflect (reverse) the second sequence. The result is $(11, 10)$.
4. Concatenate the two sequences to get $(00, 01, 11, 10)$.

It is easy to see that consecutive numbers differ by one bit only.

◇ **Exercise C.5:** Follow the rules above to get the binary RGC for $i = 3$.

Notice that the first and last numbers in an RGC also differ by one bit. RGCs can be created for any number system using the following notation and rules: Let $a = a_1a_2 \cdots a_m$ be a non-negative, base-n integer (i.e., $0 \le a_i < n$). Define the quantity $p_j = (\sum_{i=1}^{j} a_i) \bmod 2$, and denote the base-$n$ RGC of a by $a' = b_1b_2 \cdots b_m$. The digits b_i of a' can be computed by

$$b_1 = a_1; \quad b_i = \left\{ \begin{array}{lll} a_i & \text{if } p_{i-1} = 0; & \multirow{2}{*}{Odd n} \\ n-1-a_i & \text{if } p_{i-1} = 1. & \\ \hline a_i & \text{if } a_{i-1} \text{ is even;} & \text{Even } n \\ n-1-a_i & \text{if } a_{i-1} \text{ is odd.} & \end{array} \right. \qquad i = 2, 3, \ldots, m.$$

[Note that $(a')' = a$ for both even and odd n.] For example, the RGC of the sequence of base-3 numbers (trits) 000, 001, 002, 010, 011, 012, 020, 021, 022, 100, 101... is 000, 001, 002, 012, 011, 010, 020, 021, 022, 122, 121... .

The connection between Peano curves and RGCs is: Let a be a node in the peano curve P_m. Write a as a ternary (base-3) number with $2m$ trits $a = a_1a_2 \cdots a_{2m}$. Let $a' = b_1b_2 \cdots b_{2m}$ be the RGC equivalent of a. Compute the two numbers $x' = b_2b_4b_6 \cdots b_{2m}$ and $y' = b_1b_3b_5 \cdots b_{2m-1}$. The number x' is the RGC of a number x, and similarly for y'. The two numbers (x, y) are the coordinates of node a in P_m.

Bibliography

Backus, J. W. (1959) "The Syntax and Semantics of the Proposed International Algebraic Language," in *Proceedings of the International Conference on Information processing*, pp. 125–132, UNESCO.

Chomsky, N. (1956) “Three Models for the Description of Language,” *IRE Transactions on Information Theory* **2**(3):113–124.

Cole, A. J. (1985) “A Note on Peano Polygons and Gray Codes,” *International Journal of Computer Mathematics* **18**:3–13.

Cole, A. J. (1986) “Direct Transformations Between Sets of Integers and Hilbert Polygons,” *International Journal of Computer Mathematics* **20**:115–122.

Gray, Frank (1953) “Pulse Code Communication,” United States Patent 2,632,058, March 17.

Hilbert, D. (1891) “Ueber Stetige Abbildung Einer Linie auf ein Flachenstuck,” *Math. Annalen* **38**:459–460.

Lindenmayer, A. (1968) “Mathematical Models for Cellular Interaction in Development,” *Journal of Theoretical Biology* **18**:280–315.

Matlab (1999) is URL `http://www.mathworks.com/`.

Naur, P. et al. (1960) “Report on the Algorithmic Language ALGOL 60,” *Communications of the ACM* **3**(5):299–314, revised in *Communications of the ACM* **6**(1):1–17.

Peano, G. (1890) “Sur Une Courbe Qui Remplit Toute Une Aire Plaine,” *Math. Annalen* **36**:157–160.

Sierpiński, W. (1912) “Sur Une Nouvelle Courbe Qui Remplit Toute Une Aire Plaine,” *Bull. Acad. Sci. Cracovie* Serie A:462–478.

Sagan, Hans (1994) *Space-Filling Curves*, New York, Springer Verlag.

Wirth, N. (1976) *Algorithms + Data Structures = Programs*, Englewood Cliffs, NJ, Prentice-Hall, 2nd Ed.

All letters come and go—L is here to stay.
Grzegorz Rozenberg