Department of Informatics
Technical University Munich

Space-Filling Curves
An Introduction

Paper accompanying the presentation held on April 2nd 2005
for the Joint Advanced Student School (JASS)
in St. Petersburg

Levi Valgaerts

1. Introduction

This text is interpreted as a general introduction to the concept of space-filling curves (SFCs). It is mainly a résumé of the presentation I held on the subject for the Joint Advanced Student School 2005. The text covers a short treatment of the most frequently encountered SFCs, followed by some illustrations of their application in scientific computing.

SFCs are encountered in different fields of computer science, especially where it is important to linearize multidimensional data. Examples of multidimensional data are matrices, images, tables and computational grids resulting from the discretization of partial differential equations (PDEs). Data operations like matrix multiplications, load-store operations and updating and partitioning of data sets can be simplified when we choose an efficient way of going through the data. In many applications SFCs present just this optimal manner of mapping multidimensional data onto a one dimensional sequence.

The study of SFCs heavily relies on the knowledge of set theory and topology. The easiest approach of the subject though, is through a geometric treatment which makes the nature of SFCs much faster comprehensible to the reader. A geometric generating process can be applied to all SFCs described in this text. Apart from that I will also mention an analytical treatment which is based on the methods presented in [1] and [2].

2. Space-Filling Curves

2.1. Mathematical Description

In this text we will only deal with two dimensional SFCs. Since a two dimensional curve is defined as a continuous mapping from a closed and bounded line segment into P^2 and since every closed and bounded line segment is homeomorphic to the closed unit-interval I, we can assume the curve to have the domain I. Furthermore we will only consider mappings from I onto the unit-square Ω or a closed triangular region T.

One can easily define a surjective mapping from I onto Ω ([2]) and Cantor showed that I can even be mapped bijectively onto Ω. Netto though showed that such a bijection has to be necessarily discontinuous and can therefore not be called a curve. When we drop the condition of bijectivity it turns out that we can still define surjective mappings from I onto Ω that are continuous. In other words, there exist curves that pass (at least once) through every point of the unit square. Such mappings are called (2D) SFCs. More formally a SFC is a continuous mapping f from I onto P^n where $f(I)$ has a strictly positive Jordan content (area for n=2 or volume for n=3).

2.2. The Hilbert Space-Filling Curve

2.2.1. Geometric Generation

Hilbert was the first to propose a geometric generation principle for the construction of a SFC. The procedure is an exercise in recursive thinking and can be summed up in a few lines:

- We assume that I can be mapped continuously onto the unit-square Ω. If we partition I into four congruent subintervals than it should be possible to partition Ω into four congruent subsquares, such that each subinterval will be mapped continuously onto one of the subsquares. We can repeat this reasoning by again partitioning each subinterval into four congruent subintervals and doing the same for the respective subsquares.
- When repeating this procedure ad infinitum we have to make sure that the subsquares are arranged in such a way that adjacent subsquares correspond to adjacent subintervals. Like this we preserve the overall continuity of the mapping.
- If an interval corresponds to a square, then its subintervals must correspond to the subsquares of that square. This inclusion relationship assures that a mapping of the nth iteration preserves the mapping of the (n-1)th iteration.

Now every t ™ I can be regarded as the limit of a unique sequence of nested closed intervals. With this sequence corresponds a unique sequence of nested closed squares that shrink into a point of Ω, the image $f_h(t)$ of t. $f_h(I)$ is called the Hilbert SFC.

If we connect the midpoints of the subsquares in the nth iteration of the geometric generation procedure in the right order by polygonal lines, we can make the convergence to the Hilbert Curve visible. This is done in fig.1 for the first three iterations and the sixth iteration.

Every point in Ω lies in a sequence of nested closed squares, which corresponds to a sequence of nested closed intervals. Hence the above defined mapping is surjective. If a point in Ω lies on the corner of a square, it may belong to two squares that do not correspond to adjacent intervals and therefore to at least two different sequences of nested closed squares. This means that there are points in Ω that have more than one image in I. f_h can therefore not be injective.

In the nth iteration we have partitioned I into 2^{2n} subintervals, each of length $1/2^{2n}$. The subsquares all have side length of $1/2^{2n}$. If we choose two points t_1 and t_2 in I, such that $|t_1 - t_2| < 1/2^{2n}$, than t_1 and t_2 lie at the worst in two consecutive subintervals. Their images lie at the worst in two consecutive squares and for the distance between the image points it holds that $\|f_h(t_1) - f_h(t_2)\| \le \sqrt{5}/2^n$, where the $\sqrt{5}$ comes from the diagonal of the rectangle formed by the two squares. In the limiting case for $n \to \infty$ this distance tends to 0 and $f_h : I \longrightarrow \Omega$ is therefore continuous.

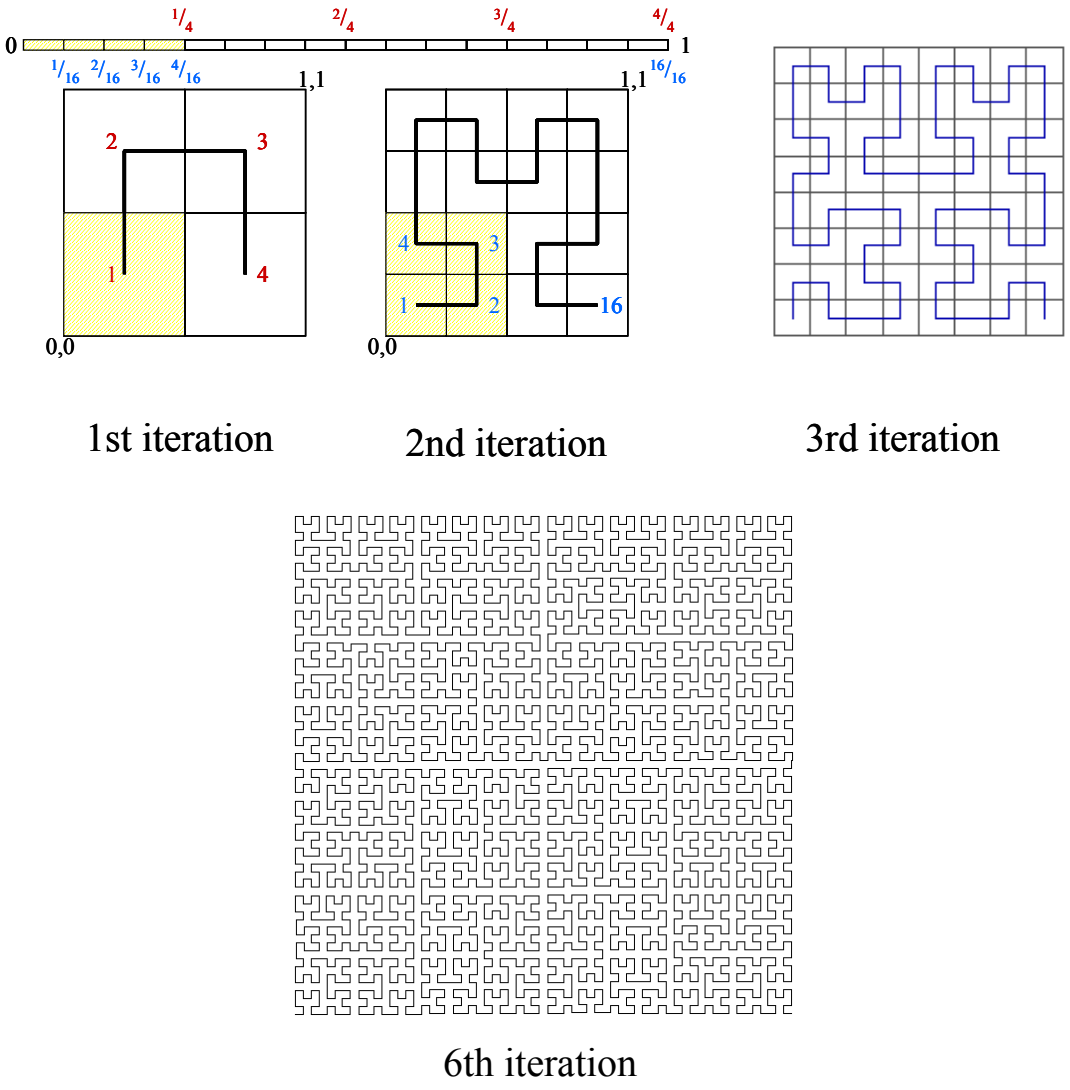

fig.1. Geometric generation of the Hilbert space-filling curve

2.2.2. Arithmetic Definition

An arithmetic description of the Hilbert curve would allow us to calculate the coordinates of the image point of any $t \in I$ using a form of parameter representation. If we keep in mind that the geometric generation was based on a recursive division of the unit interval into four congruent subintervals, we can simplify matters by representing the parameter t in quaternary notation:

$$t = 0_4 q_1 q_2 q_3 \ldots = q_1 / 4 + q_2 / 4^2 + q_3 / 4^3 + \ldots, \quad q_j = 0,1,2 \text{ or } 3$$

Corresponding to the partitions of *I* there was the recursive division of Ω into four subsquares. The former generation principle actually implies that for each step in the iteration the subsquares can be regarded as affine transformations of the original unit-square. To make this clear we take a look at the figures in fig.2. If we apply the recursive generating principle than we discover that if the starting point of the curve is (0,0), then the end point can only be (1,0) or (0,1). In the *0*th iteration we only know that the origin is the starting point of the curve and (1,0) the endpoint, how the curve proceeds throughout Ω is not important. We therefore choose an orientation as shown in the first picture. In the subsequent iterations the orientation of the subsquares have to be such that the exit point of each subsquare coincides with the entry point of the following subsquare, and this while preserving the previous orientation. Fig.2 shows the only possible orientations for the *1*st and *2*nd iteration.

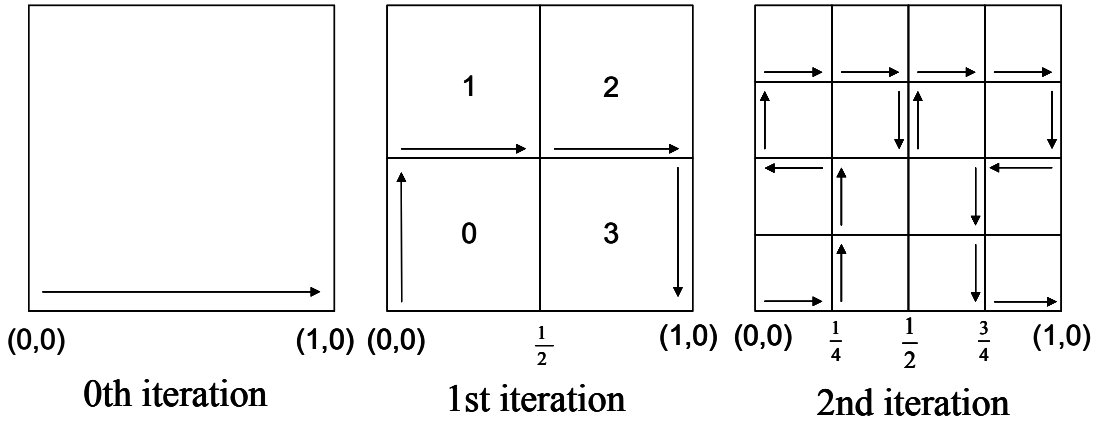

fig.2. The steps in the analytical representation of the Hilbert space-filling curve

To achieve the configuration of the *1*st iteration we have to subject Ω to affine transformations, meaning a combination of rotations, reflections, scaling and translations. We end up with four transformations $\boldsymbol{h}_j$ that map Ω to one of the four subsquares. These transformations can be represented in complex form or in equivalent matrix form:

$$\boldsymbol{h}_0 z = \frac{1}{2}\bar{z}i \qquad : \boldsymbol{h}_0\begin{pmatrix}x_1\\x_2\end{pmatrix} = \frac{1}{2}\begin{bmatrix}0 & 1\\1 & 0\end{bmatrix}\begin{pmatrix}x_1\\x_2\end{pmatrix} + \frac{1}{2}\begin{pmatrix}0\\0\end{pmatrix} = \frac{1}{2}H_0\begin{pmatrix}x_1\\x_2\end{pmatrix} + \frac{1}{2}h_0$$

$$\boldsymbol{h}_1 z = \frac{1}{2}z + \frac{i}{2} \qquad : \boldsymbol{h}_1\begin{pmatrix}x_1\\x_2\end{pmatrix} = \frac{1}{2}\begin{bmatrix}1 & 0\\0 & 1\end{bmatrix}\begin{pmatrix}x_1\\x_2\end{pmatrix} + \frac{1}{2}\begin{pmatrix}0\\1\end{pmatrix} = \frac{1}{2}H_1\begin{pmatrix}x_1\\x_2\end{pmatrix} + \frac{1}{2}h_1$$

$$\boldsymbol{h}_2 z = \frac{1}{2}z + \frac{1}{2} + \frac{i}{2} \qquad : \boldsymbol{h}_2\begin{pmatrix}x_1\\x_2\end{pmatrix} = \frac{1}{2}\begin{bmatrix}1 & 0\\0 & 1\end{bmatrix}\begin{pmatrix}x_1\\x_2\end{pmatrix} + \frac{1}{2}\begin{pmatrix}1\\1\end{pmatrix} = \frac{1}{2}H_2\begin{pmatrix}x_1\\x_2\end{pmatrix} + \frac{1}{2}h_2$$

$$\boldsymbol{h}_3 z = -\frac{1}{2}\bar{z}i + 1 + \frac{i}{2} \qquad : \boldsymbol{h}_3\begin{pmatrix}x_1\\x_2\end{pmatrix} = \frac{1}{2}\begin{bmatrix}0 & -1\\-1 & 0\end{bmatrix}\begin{pmatrix}x_1\\x_2\end{pmatrix} + \frac{1}{2}\begin{pmatrix}2\\1\end{pmatrix} = \frac{1}{2}H_3\begin{pmatrix}x_1\\x_2\end{pmatrix} + \frac{1}{2}h_3$$

where z and (x_1,x_2) are the complex and vector representation of a point of the unit-square. The indices of $\boldsymbol{h}_j$ correspond to the respective subsquare on which Ω is mapped. Applying these four transformations to the configuration of the *1*st iteration yields the *3*rd image and so on.

When we look at the quaternary representation of the parameter t we can conclude that $f_h(t)$ lies in the q_1th subsquare of the first partition, and further in the q_2th subsquare of the second partition within the q_1th subsquare of the first partition and so on. This leads us to the following conclusion :

$$f_h(t) = \lim_{n\to\infty} \boldsymbol{h}_{q_1}\boldsymbol{h}_{q_2}\boldsymbol{h}_{q_3}...\boldsymbol{h}_{q_n}\Omega$$

If we only look at finite quaternaries we get the following simplification :

$$f_h(0_4 q_1 q_2 q_3 ... q_n) = \boldsymbol{h}_{q_1}\boldsymbol{h}_{q_2}\boldsymbol{h}_{q_3}...\boldsymbol{h}_{q_n}\underbrace{\boldsymbol{h}_0\boldsymbol{h}_0\boldsymbol{h}_0...\Omega}$$

$$f_h(0_4 q_1 q_2 q_3 ... q_n) = \boldsymbol{h}_{q_1}\boldsymbol{h}_{q_2}\boldsymbol{h}_{q_3}...\boldsymbol{h}_{q_n}\begin{pmatrix}0\\0\end{pmatrix}$$

because $\boldsymbol{h}_0$ only involves scaling. These formulas can be used to calculate the coordinates of the image of edge points of subintervals of the nth iteration and can easily be implemented into a recursive program ([2]) where n depends on the desired degree of precision. The last formula can be worked out further ([1]) resulting in an expression that only contains the matrix parts of the affine transformations and can be used to find an analytic representation of the component functions of f_h.

2.2.3. Approximating Polygons

The polygonal line that connects the image points of the subinterval edges of I of the nth iteration, is called the nth approximating polygon for the Hilbert curve or the discrete Hilbert curve:

$$p_n : I \to \Omega : p_n(t) = 2^{2n}(t - \frac{k}{2^{2n}}) f_h(\frac{k+1}{2^{2n}}) - 2^{2n}(t - \frac{k+1}{2^{2n}}) f_h(\frac{k}{2^{2n}}),$$
$$for\ k/2^{2n} \le t \le (k+1)/2^{2n},\ k = 0,1,2,3,...2^{2n}-1$$

passes through $f_h(0), f_h(1/2^{2n}), f_h(2/2^{2n}), f_h(3/2^{2n}), ..., f_h((2^{2n}-1)/2^{2n}), f_h(1)$, and $\{p_n\}$ converges uniformly to f_h.

In fig.2 we see the approximating polygons for the first two iterations. We see that these curves pass more than once through certain points. The curves from fig.1 that join the midpoints of the subsequent squares can be regarded as approximating polygons too since they also uniformly converge to f_h.

2.3. The Peano Space-Filling Curve

2.3.1. Geometric Generation

Peano defined a mapping $f_p : I \to \Omega$ using ternaries of the parameter t as follows:

$$f_p(0_3 t_1 t_2 t_3 t_4 \ldots) = \begin{pmatrix} 0_3 t_1 (k^{t_2} t_3)(k^{t_2+t_4} t_5)\ldots \\ 0_3 (k^{t_1} t_2)(k^{t_1+t_3} t_4)\ldots \end{pmatrix} \qquad (2.3.1.)$$

$$with \;\; kt_j = 2 - t_j \;\; (t_j = 0,1,2) \;\; and \;\; k^v \; the \, vth \; iterate \; of \; k$$

and showed it to be surjective and continuous.

To demonstrate that f_p indeed represents a SFC, we can derive from Peano's definition the following:

$$f_p(0_3 00 t_3 t_4 t_5 \ldots) = \begin{pmatrix} 0_3 0\alpha_2\alpha_3\alpha_4\ldots \\ 0_3 0\beta_2\beta_3\beta_4\ldots \end{pmatrix} \text{ and } f_p(0_3 01 t_3 t_4 t_5 \ldots) = \begin{pmatrix} 0_3 0\alpha_2\alpha_3\alpha_4\ldots \\ 0_3 1\beta_2\beta_3\beta_4\ldots \end{pmatrix} \text{ and}\ldots$$

meaning that the subinterval $\left[0, \frac{1}{9}\right]$ will be mapped onto the subsquare $\left[0, \frac{1}{3}\right] \times \left[0, \frac{1}{3}\right]$, the subinterval $\left[\frac{1}{9}, \frac{2}{9}\right]$ will be mapped onto the adjacent subsquare $\left[0, \frac{1}{3}\right] \times \left[\frac{1}{3}, \frac{2}{3}\right]$,…

In general we can apply Hilbert's generation principle as described under 2.2.1. but this time we partition I into 3^{2n} congruent subintervals and map them onto as many subsquares of Ω. Fig.3 shows how this is done for the first three iterations. The polygonal lines give the order in which the subsquares have to be taken.

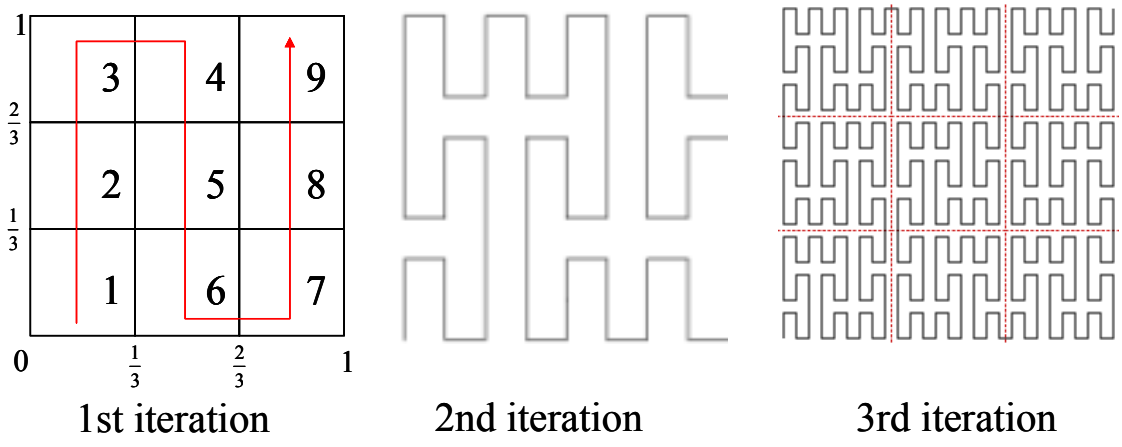

fig.3. Geometric generation of the Peano space-filling curve

2.3.2. Arithmetic Definition

We know that the Peano curve passes through the unit-square diagonally from (0,0) to (1,1). As under 2.2.2. we can define an orientation and affine transformations that map each iterate to one of the 9 subsquares of the next iterate, while preserving the previous orientations. Fig.4 shows the procedure for the first iteration.

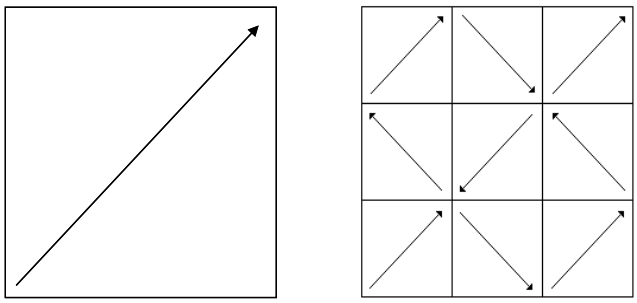

fig.4. The steps in the analytical representation of the Peano space-filling curve

The 9 transformations can be defined in a similar way as with the Hilbert curve: $\boldsymbol{p}_0 z = \frac{1}{3}z, \boldsymbol{p}_1 z = -\frac{1}{3}\bar{z} + \frac{1}{3} + \frac{i}{3}, \ldots$ in complex form, $\boldsymbol{p}_j \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \frac{1}{3} P_j \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \frac{1}{3} p_j, \; j = 0,1,\ldots 8$ in matrix notation. Each $\boldsymbol{p}_j$ maps Ω to the *(j+1)*th subsquare of the first iteration as shown in fig.3.

Making use of the ternary representation of *t* and the equality

$$0_3 t_1 t_2 \ldots t_{2n-2} t_{2n} \ldots = 0_9 (3t_1 + t_2)(3t_3 + t_4)\ldots(3t_{2n-1} + t_{2n})\ldots$$

we can proceed in the same manner is with the Hilbert curve. Realizing that

$$f_p(t) = \lim_{n\to\infty} \boldsymbol{p}_{3t_1+t_2} \boldsymbol{p}_{3t_3+t_4} \ldots \boldsymbol{p}_{3t_{2n-1}+t_{2n}} \Omega$$

and applying to finite ternaries gives us an expression for the calculation of the image points ([2]). By induction it can be proven that this expression leads to formula (2.3.1) for the infinite ternary representation of the parameter *t* ([1]).

2.3.3. Approximating Polygons

The polygonal line that connects the image points (fig.4) or the midpoints of the subsquares (fig.3) in the nth iteration is the nth approximating polygon for the Peano curve. Fig.5 shows that there are also other ways of passing through the subsquares that satisfy the recursive generating principle.

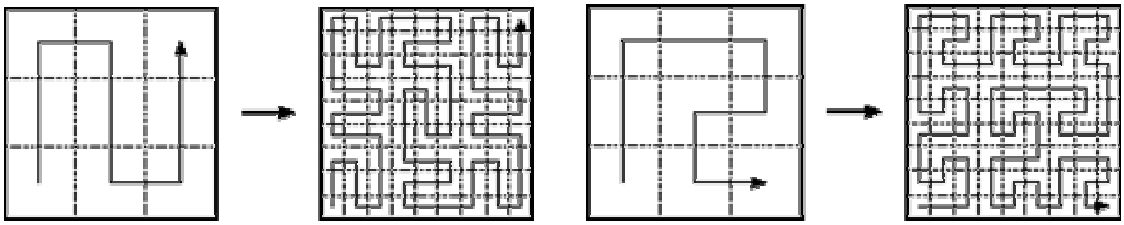

fig.5. Other ways of going through the subsquares for the Peano curve

2.4. The Sierpinski Space-Filling Curve

We can apply the geometric generation procedure to any closed two dimensional region that can be partitioned into mutually congruent (similar) sub regions. We can therefore consider the mapping $f_s : I \to T$ where T is an isosceles right triangle with angular points in (0,0), (1,1) and (2,0), and define it as follows.

We can divide I into 2^n or 2^{2n} subintervals and T into as many subtriangles such that adjacent intervals are mapped to adjacent triangles with an edge in common. Dividing in each step by 4 allows us to derive a simpler arithmetic representation since the transformations will only include rotations over $\pi/2$. When we repeat the partitioning ad infinitum there will correspond with every sequence of nested closed intervals a sequence of nested closed triangles that uniquely converges to a point $f_s(t)$ in T. $f_s(I)$ is called the Sierpinski SFC and fig.6 shows the successive generation steps.

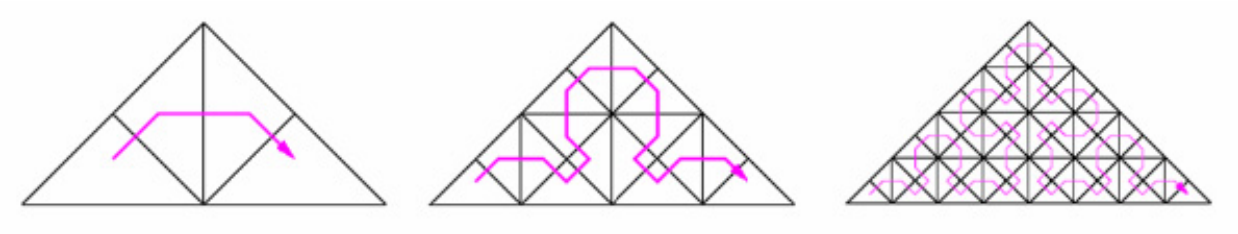

fig.6. Geometric generation of the Sierpinski space-filling curve

Obtaining an analytical representation is analogous to the method presented for the previous two SFCs using quaternaries for t ™ I ([1,2]).

2.5. The Lebesgue Space-Filling Curve

The mapping

$$f(0_3(2t_1)(2t_2)(2t_3)\ldots) = \begin{pmatrix} 0_2 t_1 t_3 t_5 \ldots \\ 0_2 t_2 t_4 t_6 \ldots \end{pmatrix}$$

from the Cantor Set $\Gamma = \{0_3(2t_1)(2t_2)(2t_3)... \mid t_j = 0 \, or \, 1\}$ onto Ω can be shown to be continuous and surjective. In order to obtain a continuous mapping on I we have to extend f into Γ^c continuously. When this is done by linear interpolation between the image points of interval edges that where removed during the construction of Γ, we get a mapping f_l that is called the Lebesque SFC. Fig.7 shows the successive generation steps according to the steps in the recursive construction of Γ.

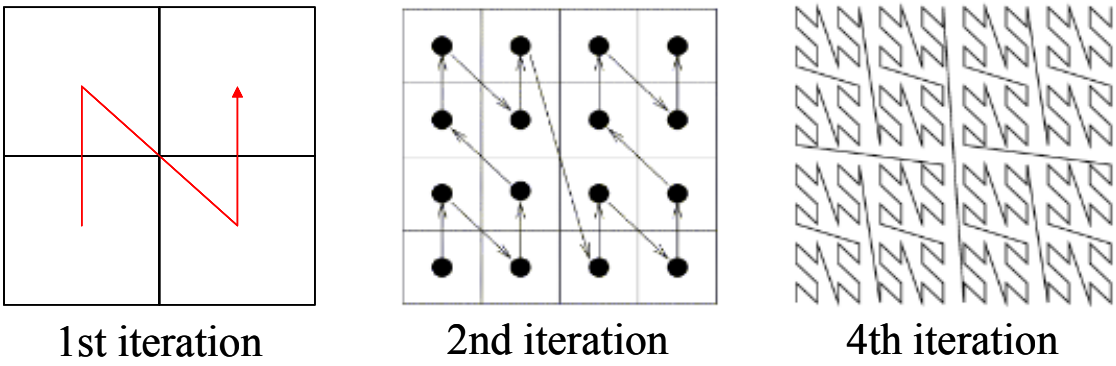

fig.7. Geometric generation of the Lebesgue space-filling curve

The Lebesque SFC is due to its definition almost everywhere differentiable, unlike the SFCs we have seen thus far. On the contrary it lacks the locality property all previous curves possess, meaning that it jumps across the unit-square and that not every restriction of f_l to a subinterval of I is space-filling.

2.6. Representation of Space-Filling Curves Through a Grammar

In the construction of the Hilbert curve we can distinguish four templates which reoccur in every iteration step and can be labeled H, A, B and C (fig.8). These templates are translated in every iteration step into a first iteration of the Hilbert curve (fig.9). These fixed translation schemes can be described by a grammar:

$$
\begin{aligned}
H &\longleftarrow A \uparrow H \rightarrow H \downarrow B \\
A &\longleftarrow H \rightarrow A \uparrow A \leftarrow C \\
B &\longleftarrow C \leftarrow B \downarrow B \rightarrow H \\
C &\longleftarrow B \downarrow C \leftarrow C \uparrow A
\end{aligned}
$$

which is given by the four templates and the transitions between them. The grammar delivers an easy recipe for implementing the order in which the subsquares of the discrete Hilbert curve have to be traversed:

$$H \longleftarrow A \uparrow H \rightarrow H \downarrow B$$
$$\longleftarrow H \rightarrow A \uparrow A \leftarrow C \uparrow A \uparrow H \rightarrow H \downarrow B \rightarrow A \uparrow H \rightarrow H \downarrow B \downarrow C \leftarrow B \downarrow B \rightarrow H$$

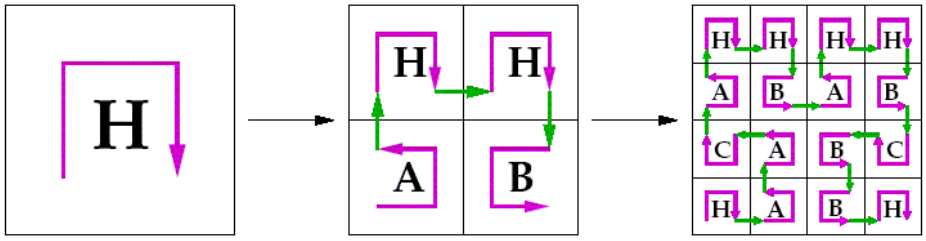

fig.8. Identification of the four templates

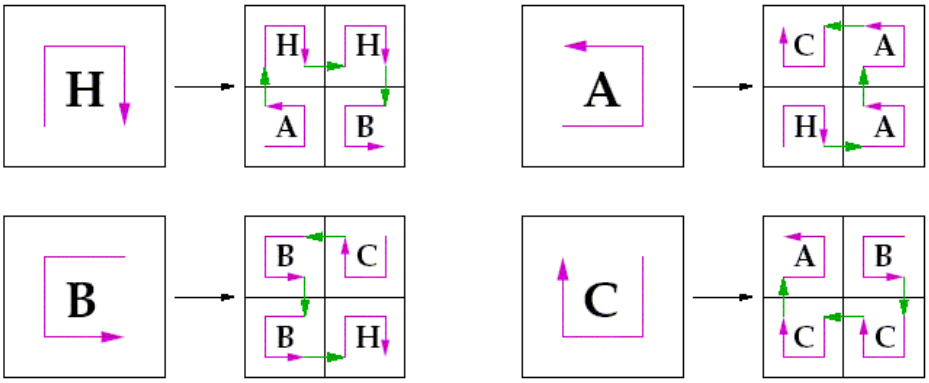

fig.9. Translation scheme of the four templates

3. Applications of Space-Filling Curves

SFCs present a way of mapping multidimensional data onto a one dimensional sequence and thus a way of going through a multidimensional data set in a certain order. Some of the most favorable properties of SFCs in this respect are their locality, their recursive nature and the fact that the linearization is easily computable.

Locality means that a SFC never leaves a region at any level of refinement before traversing all points of that region. Thus neighboring data in a multidimensional space remain neighboring after linearization. This clustering of data is important in exploiting cache memory during computation and for the partitioning of computational grids. The recursive behavior of SFCs allows for example for the linearization of recursive hierarchical data structures.

Storing a computational grid in memory requires a suitable data structure. When we are dealing with adaptively refined grids, adaptive grid refinement algorithms need to be implemented in such a way that the computational complexity remains acceptable ([4]). To ensure that the grid manipulation part does not become too expensive, appropriate representations of the grid must be chosen.

Besides arrays and hash tables, space trees can be used as an implementation of a computational grid. Space trees are data structures in which every node is a new space tree or a leaf corresponding to a grid cell (fig.10). Their recursive nature allows for refinement to any arbitrary level within each cell. Linearizing this data structure can be important to define a traversing order of the cells or to map the data to memory. Since discrete SFCs are defined by a recursion themselves, the order can be deducted directly from the SFC in a top-down depth-first process (in contrary to a breadth first numbering, fig.10). Cells of the first refinement level are visited first according to the first iterate of the SFC. If a cell is further refined, the sub cells are traversed according to the respective iterate of the SFC. Figure 10 shows this for the Peano curve on a grid where the middle cell has been further refined. In the implementation we can make use of the grammar rules as presented earlier.

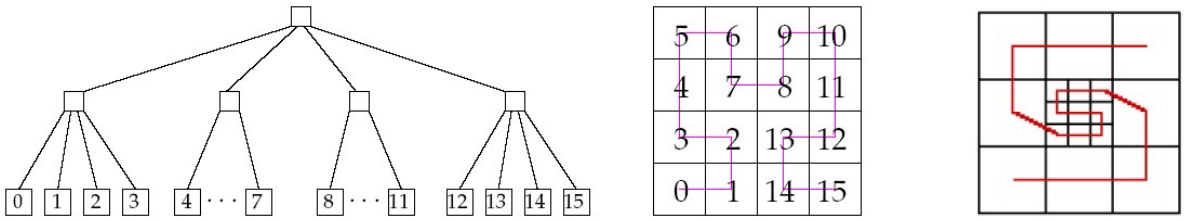

fig.10. Using a space tree and the Hilbert space-filling curve to represent a computational grid (left and middle). Using a Peano curve to traverse the cells in an adaptively refined grid (right)

One of the bottlenecks in high performance computing is the time needed to access data in memory. Jumps in the address space during computation can lead to cache misses and substantial slow down. In designing cache-aware algorithms for the solution of partial differential equations in particular, SFCs can be used to build data structures that allow fast data access and that exploit cache hierarchies in modern computers ([3]).

The result of a discretization scheme (finite differences, finite elements,…) is a system of linear equations that can be solved by using various techniques. The discretization gives rise to 2D stencils that represent the discrete operator. For the evaluation in one grid point, most of the time only direct neighboring points or needed, thus showing the locality of the discrete operator. In the case where the grid is associated with space trees, we can run through the leaves in a sequence described by a SFC. The leaves of the tree correspond to the grid cells and the discrete operator can be

decomposed into parts per cell which accumulate to the whole operator value after one run over all grid cells.

Using SFCs to define the processing order of grid cells allows us to make use of a fixed number of stack data structures and this in such a way that the data needed at a certain point in the sequence of operations are always on top of the stack. Figure 11 shows how grid points are being processed linearly forward and backward in time if we go through the grid in a strictly cell-oriented way using a discrete Peano curve. Therefore stacks are ideal to store the information that corresponds to the grid points. Because all required data always lies on top, data access becomes more cache-efficient.

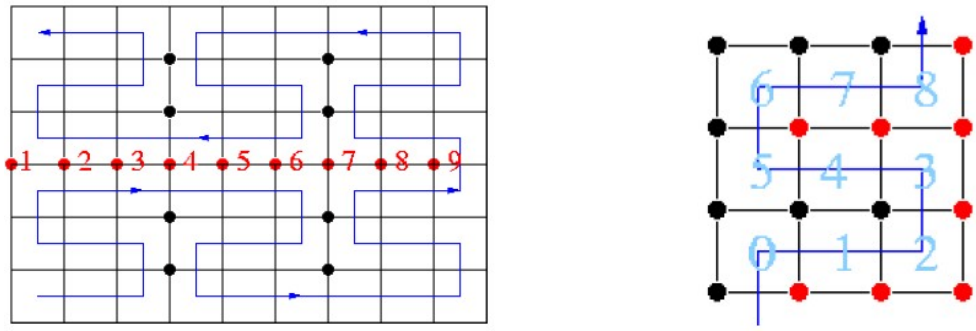

fig.11. Processing order of the grid cells (left) and the use of two stacks (red and black) with the Peano curve (right)

For a 2D regular grid we only need two stacks (fig.10), for adaptively refined grids it can be shown that additional stacks are needed to transfer data over the grid levels. In addition the combination of SFCs and stacks makes the memory behavior deterministic and allows the processing order of the data to be inverted easily. This last property makes it possible to efficiently process the data several times, as required by iterative solvers.

Since most computations in scientific computing will take place on parallel systems, load balancing is necessary ([2, 4]). This means that it is imperative that the same work load is assigned to all processors and none of them becomes idle during computation. Because the work load mainly depends on the amount of data to be processed, a data parallel approach is appropriate. Data partitioning is, in case of PDE solvers, a grid partitioning problem.

Because the local discretization schemes for PDEs use immediately neighboring nodes for the evaluation, we require that the partitions are compact in order to keep data transfer between different processors as small as possible. For the same reason we demand that the edges (separators) of the partitions are as small as possible.

The basic idea in using SFCs for load balancing is to map the elements in space (Ω) to points on an iterate of a discrete SFC. These points can be mapped on I using the inverse mapping of the SFC. The image points on I can now be sorted and grouped together (partitioned), after which each group is mapped to a processor. When I is divided into subintervals of equal workload, we automatically attain a perfect load balance.

SFC can be used for efficient parallelization because they give a simple means to calculate balanced partitions of which the separators can be shown to be almost optimal. The use of Hilbert or Peano curves assures that neighboring points on the unit interval will be mapped to neighboring entities in the multidimensional space, thus satisfying the compactness requirement. Lebesgue-type SFCs do not satisfy the so called Hölder continuity criterion and tend to give disconnected partitions.

References

[1] H. Sagan, Space-Filling Curves, Springer-Verlag, New York, 1994.

[2] M. Bader, Raumfüllende Kurven, Begleitendes Skriptum zum entsprechenden Kapitel der Vorlesung „Algorithmen des Wissenschaftlichen Rechnens“,Technischen Universität München, 2004.

[3] F. Günther, M. Mehl, M. Pögl, C. Zenger, A cache-aware algorithm for PDEs on hierarchical data structures based on space-filling curves, *SIAM Journal of Scientific Computing*, submitted.

[4] G. Zumbusch, Adaptive Parallel Multilevel Methods for Partial Differential Equations, Habilitation, Universität Bonn, 2001.