Appendix A
Solutions to Selected Exercises

A.1 Two Motivating Examples

(no exercises)

A.2 Space-Filling Curves

2.1 The corner points of the approximating polygons reflect an entry/exit point of the curve in each subsquare. As an entry point, the respective points – by construction – will always be located in the first subsquare and correspond to the leftmost of the respective nested intervals. Hence, the entry point is an image of the left boundary of the respective interval. As examples, we obtain the parameter–point pairs \( 0 \rightarrow (0, 0), \frac{1}{4} \rightarrow (0, \frac{1}{2}) \), etc.

2.2 Some further variants to construct a curve similar to Moore’s variant are given in Fig. A.1 (see also [168]).

2.3 The proof of continuity of the Peano curve can be copied almost word by word from the respective proof for the Hilbert curve, as given in Sect. 2.3.5: For two given parameters \( t_1 \) and \( t_2 \), we choose a refinement level \( n \), such that \( |t_1 - t_2| < 9^{-n} \) (owing to the substructuring into nine subsquares in each step); \( t_1 \) and \( t_2 \) are then mapped to points \( p(t_1) \) and \( p(t_2) \) that are lying in two adjacent subsquares of side length \( 3^{-n} \). Their distance \( \|p(t_1) - p(t_2)\| \) thus has to be smaller than \( 3^{-n} \cdot \sqrt{5} \) (and the rest of the proof is straightforward).

2.4 I hope you didn’t take that exercise too seriously...
Fig. A.1 Three variants of the Hilbert-Moore curve. The left-most variant uses the same orientation of the four Hilbert parts as Moore’s curve, but moves the start and end point to the centre of the unit square.

Fig. A.2 Construction and grammar symbols of the Peano-Meander curve.

A.3 Grammar-Based Description of Space-Filling Curves

3.1 The derivation of a grammar for the Peano-Meander curve is illustrated in Fig. A.2. We need the following symbols and production rules:

- Non-terminals: \(\{M, W, L, N\}\), start symbol \(M\)
- Terminals: \(\{\uparrow, \downarrow, \leftarrow, \rightarrow\}\)
- Production rules:

\[
M \arrow N \uparrow N \uparrow M \rightarrow M \rightarrow M \downarrow W \arrow L \downarrow L \rightarrow M \\
W \arrow L \downarrow L \downarrow W \arrow W \arrow W \uparrow M \rightarrow N \uparrow N \arrow W \\
L \arrow W \arrow W \arrow L \downarrow L \rightarrow L \downarrow L \rightarrow N \uparrow M \rightarrow M \rightarrow L \\
N \arrow M \rightarrow M \rightarrow N \uparrow N \uparrow N \arrow L \downarrow W \arrow W \uparrow N
\]

3.2 From Fig. 2.5, it is straightforward to obtain the production rule for the start symbol (here: \(M\)):

\[
M \arrow \tilde{B} \uparrow \tilde{B} \rightarrow \tilde{A} \downarrow \tilde{A}
\]
where the non-terminals $\tilde{A}$ and $\tilde{B}$ correspond to the patterns represented by $A$ and $B$ in the regular Hilbert-curve grammar (as in Fig. 3.1). However, their orientation is exactly inverse. This also reflects in the productions for $\tilde{A}$ and $\tilde{B}$:

$$\tilde{A} \leftarrow \tilde{C} \rightarrow \tilde{A} \uparrow \tilde{A} \leftarrow \tilde{H}$$
$$\tilde{B} \leftarrow \tilde{H} \leftarrow \tilde{B} \downarrow \tilde{B} \rightarrow \tilde{C}$$

(and similar for $\tilde{H}$ and $\tilde{C}$). Thus, the Hilbert-Moore grammar requires five non-terminals, but $M$ is only used as start symbol.

3.3 An algorithm for matrix-vector multiplication using Hilbert traversal is discussed in Sect. 13.2, see in particular Algorithm 13.3.

3.5 If rotation is neglected, all basic patterns of the Hilbert curve come down to the two patterns $\Uparrow \Uparrow \Uparrow \Uparrow$ and $\Downarrow \Downarrow \Downarrow \Downarrow$. For these, two non-terminals, in the following denoted $R$ and $L$, are sufficient, with terminal productions

$$R \leftarrow \Uparrow \Uparrow \Uparrow \Uparrow \Uparrow \Uparrow \Uparrow \Uparrow$$
$$L \leftarrow \Downarrow \Downarrow \Downarrow \Downarrow \Downarrow \Downarrow \Downarrow \Downarrow$$

Thus, we have to include the turns and moves between the patterns into the non-terminal productions:

$$R \leftarrow \Uparrow \Uparrow \Uparrow \Uparrow \Uparrow \Uparrow \Uparrow \Uparrow$$
$$L \leftarrow \Downarrow \Downarrow \Downarrow \Downarrow \Downarrow \Downarrow \Downarrow \Downarrow$$

Compare Fig. 3.6 on page 39 for illustration. From the structure of the productions, it is clear that a turtle that obeys to this grammar might do multiple turns before performing the next forward step.

A.4 Arithmetic Representation of Space-Filling Curves

4.1 With $8 = 20_4$ and $5 = 11_4$, the quaternary representation of the fractions $\frac{1}{8}$, $\frac{1}{4}$, and $\frac{2}{5}$ can be computed in a regular long division, as learned in school for the decimal system:

$$1 : 20_4 = 0.02_4$$
$$1 : 3_4 = 0.11_4$$
$$2 : 11_4 = 0.1\overline{2}_4$$

Thus, we have:

$$0 \div 20 = 0.02$$
$$0 \div 10 = 0.0$$
$$3 \div 10 = 0.3$$
$$3 \div 10 = 0.3$$

Above, the digits are taken in quaternary. To get the digits in decimal, each quartet is converted into decimal:

$$0.02_4 = 0.0000$$
$$0.11_4 = 0.0625$$
$$0.1\overline{2}_4 = 0.6667$$

Thus, the result of the division is:

$$1 : 204 = 0.0000$$
$$1 : 34 = 0.0625$$
$$2 : 114 = 0.6667$$

- Compare Fig. 3.6 on page 39 for illustration. From the structure of the productions, it is clear that a turtle that obeys to this grammar might do multiple turns before performing the next forward step.
Thus, we have \( \frac{1}{8} = 0.02, \frac{1}{3} = 0.11\ldots \), and \( \frac{2}{3} = 0.1\overline{2}\ldots \)

4.2 \( \frac{1}{3} \) has the quaternary representation \( 0.111\ldots \) (see Exercise 4.1). The recursion equation (4.1) for the Hilbert mapping thus reads:

\[
\begin{align*}
h \left( \frac{1}{3} \right) &= h(0.4,111\ldots) = H_1 \circ h(0.4,111\ldots) = H_1 \circ h \left( \frac{1}{3} \right), \\
\end{align*}
\]

which means that \( h \left( \frac{1}{3} \right) \) is a fixpoint of operator \( H_1 \). The fixpoint equation

\[
H_1 \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{2} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 0 \\ \frac{1}{2} \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix},
\]

is solved by \( x = 0 \) and \( y = 1 \), such that \( h \left( \frac{1}{3} \right) = (0,1) \).

For \( \frac{2}{3} \) (or \( 0.4,1.21212\ldots \)), Eq. (A.1) turns into

\[
h \left( \frac{2}{3} \right) = h(0.4,1.21212\ldots) = H_1 \circ H_2 \circ h(0.4,1.21212\ldots) = (H_1 \circ H_2) \circ h \left( \frac{2}{3} \right).
\]

Hence, \( h \left( \frac{2}{3} \right) \) can be computed as the fixpoint of the operator \( H_1 \circ H_2 \) (see Exercise 4.6).

4.4 Analogous to Eq. (4.3) on page 50, we obtain the following arithmetisation for the Hilbert-Moore mapping \( m(t) \),

\[
m(0.4,q_1q_2q_3\ldots) = \lim_{n \to \infty} M_{q_1} \circ H_{q_2} \circ H_{q_3} \circ \ldots \circ H_{q_n} \left( \begin{pmatrix} 0 \\ 0 \end{pmatrix} \right),
\]

where we need to derive a new set of operators \( M_i \) to reflect the fact that, in the first recursion step, we assemble four regular Hilbert curves, but using different orientation. The operators \( M_i \) are:

\[
M_0 := \begin{pmatrix} 0 & -\frac{1}{2} \\ \frac{1}{2} & 0 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} \frac{1}{2} \\ 0 \end{pmatrix}, \quad M_1 := \begin{pmatrix} 0 & -\frac{1}{2} \\ \frac{1}{2} & 0 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} \frac{1}{2} \\ \frac{1}{2} \end{pmatrix},
\]

\[
M_2 := \begin{pmatrix} 0 & \frac{1}{2} \\ -\frac{1}{2} & 0 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} \frac{1}{2} \\ 1 \end{pmatrix}, \quad M_3 := \begin{pmatrix} 0 & \frac{1}{2} \\ -\frac{1}{2} & 0 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} \frac{1}{2} \\ \frac{1}{2} \end{pmatrix}.
\]

4.6 As an example, we compute the operator \( H_{12} = H_1 \circ H_2 \):

\[
H_1 \circ H_2 \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{2} \end{pmatrix} \left[ \begin{pmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{2} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} \frac{1}{2} \\ \frac{1}{2} \end{pmatrix} \right] + \begin{pmatrix} 0 \\ \frac{1}{2} \end{pmatrix}
\]

\[
= \frac{1}{4} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} \frac{1}{2} \\ \frac{1}{2} \end{pmatrix}.
\]
A.5 Approximating Polygons

As there are only four orientations of the basic Hilbert pattern, the matrix part of the operators (corresponding to the rotation/reflection) will only have four different values (up to scaling).

A.5 Approximating Polygons

5.1 The Koch curve can be enclosed by triangular areas in the following way:

![Koch curve illustration]

On refinement level $n$, each triangle has a basis of length $3^{-n}$, a height of $3^{-n} \sqrt{3}/6$, and thus an area of $3^{-2n} \sqrt{3}/12$. Each green triangle is split into four smaller ones from each level to the next. Thus, there are $4^{-n} = 2^{-2n}$ triangles on the $n$-th level, with a total area of $2^{-2n} \cdot 3^{-2n} \sqrt{3}/12 = (\frac{3}{4})^{-2n} \sqrt{3}/12$, which converges to 0 for $n \to \infty$. As the Koch curve is enclosed by the green triangles on all levels, its area has to be even smaller, such that it can only be 0.

5.3 Figure A.3 shows subsequent iterations for two Koch curves, where the “middle third” is replaced by a very narrow isosceles triangle. Cesaro showed, in 1905 [64], that if the acute angle approaches 0, the curve becomes space-filling – compare the approximating polygon of the Sierpinski curve (compare Fig. 6.3).

5.4 The “turtle” grammar is quite simple, as it only requires a single non-terminal:

$$K \leftarrow K \ l \ K \ r \ r \ K \ l \ K$$

($l$ and $r$ are terminal symbols that represent a left or right rotation by 60°).

In contrast, deriving a plotter grammar for the Koch curve is very tedious, as the “baseline” of the curve can occur in all 60°-steps.

5.5 The construction of the grammars should be no problem, however, it is worth to state that a “turtle” grammar with only one non-terminal will not work, as the generator has to be applied in two different orientations (first “turn left” or first “turn right”). Hence, the generator-based approach to construct fractal curves only works, if we allow such variations of the generator.

5.6 For the canonical Peano curve, the respective generator is applied in two symmetric orientations. Figure A.4 shows the second iteration and polygon of a
\[ \alpha = 5^\circ \Rightarrow D \approx 1.785; \]

\[ \alpha = 1^\circ \Rightarrow D \approx 1.951; \]

Fig. A.3  A Koch curve approximating the Sierpinski curve

Fig. A.4  Iteration and approximating polygon of the Peano curve required by Exercise 5.6

curve where the generator is uniformly oriented (the first turn in a subsquare is always to the right).

5.7  Iterations of the resulting curves are plotted in Fig. A.5. The values for \( q \), \( r \), and the resulting fractal dimension \( D \) are given for each curve (following the computation in Sect. 5.3).
A.5 Approximating Polygons

\[ r = 3, \ q = 5 \Rightarrow D \approx 1.465: \]

\[ r = 4, \ q = 8 \Rightarrow D = 1.5: \]

\[ r = 6, \ q = 18 \Rightarrow D \approx 1.613: \]

\[ r = 2\sqrt{2}, \ q = 4 \Rightarrow D = \frac{4}{3}: \]

Fig. A.5 Fractal curves resulting from the generators given in Exercise 5.7
A.6 Sierpinski Curves

6.3 The construction of a turtle-based grammar for the Sierpinski curve is discussed in Sect. 14.2 – see Fig. 14.8, in particular.

6.4 Exercise 6.1 leads to a grammar with eight non-terminals, which correspond to eight congruency classes of subtriangles for such generalised Sierpinski curves. The proof for congruency of the patterns in Sect. 6.2.2 has to be extended to the remaining four congruency classes (but works in exactly the same way).

A.7 Further Space-Filling Curves

7.1 Our standard arithmetisation technique, applied to Morton order, leads to the following equation for the Morton mapping \( m(t) \):

\[
m(0,q_1q_2q_3\ldots) = \lim_{n \to \infty} M_{q_1} \circ M_{q_2} \circ \cdots \circ M_{q_n} \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \quad \text{with} \quad M_i \begin{pmatrix} x \\ y \end{pmatrix} = \frac{1}{2} \left[ \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + b_i \right] = \frac{1}{2} \left[ \begin{pmatrix} x \\ y \end{pmatrix} + b_i \right].
\]

The components of the translation vector \( b_i \) are both either 0 or 1. Applying the same technique as in Sect. 4.6.2, we obtain

\[
m(0,q_1q_2q_3\ldots) = \frac{1}{2} b_{q_1} + \frac{1}{2^2} b_{q_2} + \frac{1}{2^3} b_{q_3} + \ldots,
\]

which corresponds to a binary representation.

7.3 From Fig. A.6, we can derive the following grammar to describe the approximating polygons of the Gosper curve:

\[
G \leftarrow \quad G \uparrow R \uparrow R \uparrow G \uparrow R \uparrow G \uparrow G \uparrow R \\
| \quad \uparrow \uparrow l \uparrow \uparrow l \uparrow r \uparrow r r \uparrow r \uparrow \uparrow \uparrow r \uparrow
\]

\[
R \leftarrow \quad G \uparrow R \uparrow R \uparrow L \uparrow L \uparrow G \uparrow R \uparrow G \uparrow R \\
| \quad \uparrow \uparrow l \uparrow \uparrow \uparrow l \uparrow l \uparrow r r \uparrow r \uparrow \uparrow l \uparrow r \uparrow
A.8 Space-Filling Curves in 3D

8.1 The operators for the approximation of Sagan’s 3D Hilbert curve [233] are:

\[
H_0 \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} \frac{1}{2}x + 0 \\ \frac{1}{2}z + 0 \\ \frac{1}{2}y + 0 \end{pmatrix}, \quad H_1 \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} \frac{1}{2}z + 0 \\ \frac{1}{2}y + \frac{1}{2} \\ \frac{1}{2}x + 0 \end{pmatrix}
\]

\[
H_2 \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} \frac{1}{2}x + \frac{1}{2} \\ \frac{1}{2}y + \frac{1}{2} \\ \frac{1}{2}z + 0 \end{pmatrix}, \quad H_3 \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} \frac{1}{2}y + \frac{1}{2} \\ -\frac{1}{2}x + \frac{1}{2} \\ -\frac{1}{2}y + \frac{1}{2} \end{pmatrix}
\]

\[
H_4 \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} -\frac{1}{2}z + 1 \\ -\frac{1}{2}x + \frac{1}{2} \\ \frac{1}{2}y + \frac{1}{2} \end{pmatrix}, \quad H_5 \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} \frac{1}{2}x + \frac{1}{2} \\ \frac{1}{2}y + \frac{1}{2} \\ \frac{1}{2}z + \frac{1}{2} \end{pmatrix}
\]

\[
H_6 \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} -\frac{1}{2}z + \frac{1}{2} \\ \frac{1}{2}y + \frac{1}{2} \\ -\frac{1}{2}x + 1 \end{pmatrix}, \quad H_7 \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} \frac{1}{2}x + 0 \\ -\frac{1}{2}z + \frac{1}{2} \\ -\frac{1}{2}y + 1 \end{pmatrix}
\]
Fig. A.7 Different scenarios of constructing a conforming triangular grid in restricted quadtree cells. The *nodes* indicate the vertices of the restricted quadtree grid – *nodes on edges* indicate hanging nodes in the restricted quadtree.

Fig. A.8 A restricted quadtree grid and its triangular counterpart (newest vertex bisection)

### A.9 Refinement Trees and Space-Filling Curves

9.1 In a restricted quadtree, grid vertices can either be placed on the corners of the cells or also on the midpoints of a cell edge (if the neighbouring cell is refined). To construct a conforming grid of triangles, we replace each square cell by a set of triangle cells that cover the square cell and use all vertices – as illustrated in Fig. A.7. An example of a small quadtree grid and the corresponding triangular grid, which is compatible with newest vertex bisection, is given in Fig. A.8.

9.3 Figure A.9 shows a simple triangular grid together with the Sierpinski order on the grid cells. The Sierpinski order defines a triangle strip, i.e., a sequence of edge-connected triangle cells. When reading the vertex data A to O, we have to read one additional vertex per grid cell (in Fig. A.9, the vertices are labelled such that the data is read in alphabetical order), while two vertices can always be reused:

- In the optimal case, the two reused vertices are the two predecessors in the node stream: In our example, the first five vertices are read as ABCDE, and correspond
to triangles ABC, BCD, and CDE (i.e., the last three vertices in the stream determine the triangle).

- When reading F, however, the last two vertices were D and E, whereas the next triangle is CEF.
- One option is to swap C and D on the vertex stream. With such a swap command that exchanges the second- and third-latest vertex on the stream, our triangle strip for Fig. A.9 reads (with s as swap command):

  ABCDE s FG s DH s I s J s K s L s FM s KNO.

Note that vertices D, F, and K have to be included twice in the data stream.

- Another option is to replicate all “missing” vertices within the triangle strip and thus introduce additional, duplicate triangles: Hence, after the strip ABCDE, we would need to read C again, which leads to the strip ABCDECF, in which the triangle CDE occurs twice (as CDE and DEC). The entire strip then reads:

  ABCDECFEGDHGIJGKLGFLMKNO.

- To avoid the duplication of triangles (and respective duplicate processing), we can also allow “degenerate” triangles (where two of the vertices are identical) in the strip: Changing ABCDE to ABCDCE introduces such a degenerate triangle CDC, but now has the correct sequence CE at the end to proceed with reading F from the strip to obtain triangle CEF. The entire strip for Fig. A.9 then reads:

  ABCDCEFEGDGHGIJGKGLFLMKMNO.

**A.10 Parallelisation with Space-Filling Curves**

**10.3** Algorithm A.1 is an example on how to determine the process-local partition in a size-encoded quadtree. To keep this prototypical implementation simple, the algorithm just marks the subtrees as being local or remote. Once a subtree is entirely
Algorithm A.1: Mark partitions as local or remote in a size-encoded quadtree

Procedure markPart (currIndex)
  Parameter: currIndex: quadtree nodes that have already been marked (0 on entry)
  Data: sizestream: size encoding of spacetree;
       streamptr: current position
       startPartition, endPartition: interval boundaries of the local partition
  Variable: ref: size (sizestream elements) of the children (as array)

begin
  // read info on all childs from sizestream
  for i = 1, ..., 4 do
    streamptr := streamptr + 1;
    ref[i] := sizestream[streamptr];
  end
  for i = 1, ..., 4 do
    if currIndex > endPartition or currIndex + ref[i] < startPartition then
      // mark partition as remote
      markRemote (sizestream, currIndex);
      // skip partition in bitstream
      streamptr := streamptr + ref[i];
    else if currIndex ≥ startPartition and currIndex + ref[i] ≤ endPartition then
      // mark partition as local
      markLocal (sizestream, currIndex);
      // skip partition in bitstream
      streamptr := streamptr + ref[i];
    else if ref[i] > 0 then
      // recursive call to subtree (contains local and remote notes)
      markPart (currIndex);
    end
  end
  // update variable currIndex
  currIndex = currIndex + numNodes (sizestream, streamptr);
end

inside (or outside) the partition interval, the entire subtree is marked as local (or remote). Function numNodes() returns the number of nodes in a subtree – if all nodes (including inner nodes) of the tree are counted, this information can directly be obtained from the size-encoding; if only the leaf-nodes (i.e., quadtree cells) are counted, we require an additional algorithm to determine this number (and we might want to augment the size-encoding by this data). Function numNodes() is used to update variable currIndex, which holds the number of nodes (leaves only or including inner nodes) that have already been marked during the traversal.

Algorithm A.1 is a sequential algorithm, but can be modified to work in a parallel setting as illustrated in Fig. 10.5. Here, the situation might occur that a subtree that is supposed to be local is not yet stored locally – for example, during the repartitioning of a grid. Hence, Algorithm A.1 needs to be extended by respective communication operations that obtain this part from another process. Similarly, formerly local subtrees (stored as full subtrees) might be declared remote, such that the subtree representation will have to be send to the respective process.
The information required to determine left and right nodes is provided by a turtle grammar, as introduced in Sect. 3.4. See Chap. 14 for an extensive discussion.

### A.11 Locality Properties of Space-Filling Curves

#### 11.2

The following table lists the diameter-to-volume ratios for some simple geometrical objects in 2D and 3D – the last column denotes the constant $c$ in the ratio $d = c \cdot \sqrt[3]{V}$:

<table>
<thead>
<tr>
<th>Object</th>
<th>Typ. length</th>
<th>Diameter $d$</th>
<th>Area/volume $V$</th>
<th>Ratio $d = \sqrt[3]{V}$</th>
<th>$c$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Square</td>
<td>$a$</td>
<td>$a\sqrt{2}$</td>
<td>$a^2$</td>
<td>$d = \sqrt{2} \cdot V^{1/2}$</td>
<td>1.41</td>
</tr>
<tr>
<td>Rectangle (3:1)</td>
<td>$a, 3a$</td>
<td>$a\sqrt{10}$</td>
<td>$3a^2$</td>
<td>$d = \frac{\sqrt{10}}{\sqrt[3]{3}} \cdot V^{1/2}$</td>
<td>1.83</td>
</tr>
<tr>
<td>Circle</td>
<td>$r$</td>
<td>$2r$</td>
<td>$\pi r^2$</td>
<td>$d = \frac{2}{\sqrt[3]{\pi}} \cdot V^{1/2}$</td>
<td>1.13</td>
</tr>
<tr>
<td>Cube</td>
<td>$a$</td>
<td>$a\sqrt[3]{3}$</td>
<td>$a^3$</td>
<td>$d = \sqrt[3]{3} \cdot V^{1/3}$</td>
<td>1.73</td>
</tr>
<tr>
<td>Cuboid (3:1:1)</td>
<td>$a, 3a$</td>
<td>$a\sqrt{11}$</td>
<td>$3a^3$</td>
<td>$d = \frac{\sqrt{11}}{\sqrt[3]{3}} \cdot V^{1/3}$</td>
<td>2.30</td>
</tr>
<tr>
<td>Sphere</td>
<td>$r$</td>
<td>$2r$</td>
<td>$\frac{4}{3}\pi r^3$</td>
<td>$d = \frac{2}{\sqrt[3]{3\pi}} \cdot V^{1/3}$</td>
<td>1.24</td>
</tr>
</tbody>
</table>

### A.12 Sierpinski Curves on Triangular and Tetrahedral Meshes

#### 12.2

The 3D Sierpinski curve, as given in Sect. 8.3, is face-connected, such that a first component of the proof for Hölder continuity is in place: two parameters that are in adjacent intervals will be mapped to adjacent tetrahedral cells. An upper bound for the points’ distance is thus the sum of the largest side lengths of the tetrahedra. In the standard proofs for Hölder continuity, this is put in relation with the size of the corresponding parameter intervals – in the ideal case, we have a ratio of $2^{-n}:8^{-n}$, which for the 3D Hilbert curve means that bisecting the side length in each of the three dimension will finally lead to eight subcubes and eight corresponding subintervals.

For the face-connected 3D Sierpinski curve, this ratio is less favourable: three bissection levels are not sufficient to halve the size of each subtriangle (in the terms of maximal side length) – instead we require at least five bisection steps to guarantee this. Hence, the ratio of tetrahedral side lengths to interval sizes is more
like $2^{-n}; 32^{-n} = (2^5)^{-n}$. Hence, the exponent $\frac{1}{3}$ for Hölder continuity cannot be achieved – though a somewhat smaller exponent is possible.

12.3 We have already answered this question. If we stick to a uniform bisection rule, as in Sect. 8.3, we are always faced with the bottom-most situation in Fig. 12.6. Hence, we only get black tetrahedra.

12.5 Figure A.10 illustrates the first three bisection steps of a tetrahedral cell according to the bisection scheme by Maubach – see Eq. (12.6) on page 192. The starting level and sequence of nodes in the tuple notation for the initial tetrahedron were chosen to exactly match the refinement via the Baensch-Kossaczky scheme, as in Fig. 12.5 on page 185. Hence, the two schemes will produce the same sequence of child cells from identical initial tetrahedra.
A.14 Numerical Simulation on Spacetree Grids Using Space-Filling Curves

A.13 Cache Efficient Algorithms for Matrix Operations

13.2 For the block operation $Q + = QP$, the respective $3 \times 3$ matrix multiplication is

\[
\begin{pmatrix}
a_6 & a_5 & a_0 \\
a_7 & a_4 & a_1 \\
a_8 & a_3 & a_2 \\
\end{pmatrix}
\begin{pmatrix}
b_0 & b_5 & b_6 \\
b_1 & b_4 & b_7 \\
b_2 & b_3 & b_8 \\
\end{pmatrix}
= \begin{pmatrix}
c_6 & c_5 & c_0 \\
c_7 & c_4 & c_1 \\
c_8 & c_3 & c_2 \\
\end{pmatrix}.
\]  \hspace{1cm} (A.2)

The derivation of the optimal execution order is illustrated in Fig. A.11. There, we connect only those operations where indices of successively accessed matrices are either identical or differ by at most 1.

A.14 Numerical Simulation on Spacetree Grids Using Space-Filling Curves

14.2 Of the three indices of the neighbour cells, only the index on the colour edge (i.e., edges that are not between cells with contiguous indices) is difficult to obtain. Indices on crossed edges are easy, as they are the increment and decrement of the current cell index.

The straightforward option to determine crossed-edge indices is to take the Sierpinski traversal of Algorithm 14.2 (to exchange refinement info between cells), and turn it into an algorithm to exchange indices, instead. If you invest an additional integer variable per cell to store the index of the crossed-edge neighbour, you obtain a data structure that allows direct access to all edge-connected cells.

Algorithm A.2 further reduces the storage requirements of this approach: only indices on colour edges shall be stored – for these indices, we adopt the standard...
Algorithm A.2: Sierpinski traversal to propagate refinement information (for element types \( H_o \) and \( H_n \))

**Procedure** \( H_o() \)

**Data:**
- `bitstream`: bitstream representation of spacetree (`streamptr`: current position);
- `green, red`: stacks to neighbour indices;
- `input, output`: streams for indices of colour-edge neighbours

**Variable:**
- `currIndex`: index of the current cell;
- `left/rightIndex, hypoIndex`: indices of the three neighbour cells

```
begin
  // move to next element in bitstream
  streamptr := streamptr + 1;
  if `bitstream[streamptr]` then
    // update local index and determine indices of crossed-edge neighbours
    hypIndex := currIndex;
    currIndex = currIndex + 1;
    rightIndex := currIndex + 1;
    // for colour edge, obtain Index from stack
    leftIndex := `green.pop()`;
    // write own index to colour edge output stream
    output.push(currIndex);
  else
    // recursive call to children
    \( V_o() \);
    \( K_o() \);
  end

end
```

// procedure \( H_n() \) is identical to \( H_o() \) up to the following lines:

**Procedure** \( H_n() \) begin

```
// ...
if `bitstream[streamptr]` then
  // ...
  leftIndex := `input.pop()`;
  // write own index to colour edge output stream
  green.push(currIndex);
else
  // recursive call to children...
end
```

stack&stream approach. Algorithm A.2 implements the \( H_o \)- and \( H_n \)-pattern for this idea, which have the hypotenuse and the right leg as crossed edges, and the left leg as an old/new colour edge. \( H_o \) and \( H_n \) only differ in the accesses to the colour stack, so the procedure for \( H_n \) only shows the two changed statements.

14.5 Ensuring a 2:1 size balance between adjacent elements of a quadtree or octree grid can also be implemented via respective traversals, as in Algorithm 14.2.
However, we now have to synchronise the refinement status of four edges (for quadrees) or six faces (for octrees), respectively. Also, a stack-based scheme to exchange the refinement data will not work (compare Sect. 14.3). An interesting variant is the question whether the 2:1 size balance should also be enforced between elements that are only node- or edge-connected (the latter in 3D).


References


References


References


Index

Access locality function, 206
Peano matrix multiplication, 208
Adaptive grids, 145
Adaptive refinement, 9
Algorithm
vertex labelling, 89
Applications (of SFC), 235–238
Approximating polygon
definition, 67
entry and exit points, 20
generator, 69
Hilbert curve, 19, 68
3D, 110, 111
length, 69–72
Peano curve, 68
quasi-Sierpinski curve, 183
Sierpinski curve, 79
Arithmetic representation, 94
\(\beta\Omega\)-curve, 103–104
Hilbert curve, 47–49
3D, 113
Peano curve, 57–59
Sierpinski curve, 80–81

Bänsch-Kossaczky scheme, 186, 188, 192
Barnes-Hut algorithm, 140
Basic patterns, 31
Hilbert curve, 31
3D, 110
Peano curve
3D, 116
\(\beta\Omega\)-curve, 101–104
arithmeticisation, 103–104
entry and exit points, 101
grammar, 101–102
iterations, 101
locality properties, 107
Bézier curves and surfaces, 1
Bially’s algorithm, 64
Bijective, 11
Bisection
of tetrahedra, 124, 184–186, 191–192
of triangles, 83
Bisection refinement, 141
Bitstream
for modified depth-first traversal, 154
Bitstream encoding, 140, 225
for parallel traversals, 153–155
Blocking, 13, 202
Block layout (for matrices), 213
Boundary-extended spacetrees, 232

Cache
associative, 197
direct-mapped, 198
first-, second-, third-level, 196
ideal cache, 209
L1, L2, L3, 196
\(n\)-associative, 197
prefetching, 198
replacement strategy, 198
Cache-aware, 213
algorithms, 198
Cache lines, 197
Cache memory, 195
hierarchy, 195, 196
and locality, 198–199
Cache-oblivious, 213
algorithms, 199, 233
Canonical tetrahedron, 186, 192
Cantor, G., 16, 17
Cantor Set, 97, 170
Cantor’s mapping, 16–17
bijectivity, 16
Cartesian grid, 7, 8, 144
Cartesian mesh, 2
Characteristic function, 2
Closed curve, 24, 101
Clustering, 13
of data, 12
Colour edge, 224
Colour stacks, 219
rules, 222–225
Column-major, 12, 200
Combinatorial problems, 236
Communication, 144, 145
Communication pattern
master-slave, 152
Compactness, 170
partitions, 145
Computational fluid dynamics, 141, 163
Computer aided design, 1
Computer graphics, 1, 141, 233, 237
Conforming grid, 226
Conforming refinement
spacetree grid, 226–227
Conforming triangular grid, 142
Congruency classes, 188, 192
Connected graph, 172
Connected partitions, 174
Connected SFC, 24, 107, 160, 168, 170, 174
definition, 94
Contiguous, 11
Continuity, 12
of space-filling curves, 24
Curve
definition, 17
parameter representation, 17
Data base applications, 179
Data bases, 237
Data compression, 237
Data structures
arrays, 12
matrices, 12
multidimensional data, 10
Daxpy operation, 197
Delauney triangulation, 238
Depth-first traversal, 3, 133, 134, 137
algorithm, 5
Diffusion approach
for load balancing, 149
Dimension of fractal curves, 71
Discontinuous Galerkin methods, 218
Discrete locality measure, 171
Distributed memory, 153, 155
Dithering, 238
Domain decomposition, 158, 160
dual graph, 172, 193
Dynamically adaptive, 145
Dynamically adaptive grids, 228
Edge-connected, 94, 182
Edge cut, 172
Element-based discretisation, 217–218
Entry and exit points
of partitions, 162
of SFC, 75
Entry edge, 223
Error estimation, 227
Error estimator/indicator, 145
Euclidian distance, 171
Exit edge, 223
Face-connected, 94, 173, 188, 229
Finite Element methods, 163, 181, 188, 218
Finite state machine, 62, 64
Finite Volume methods, 218
First-access order, 220
Fractal curve, 70, 72, 107
Fractal dimension, 72
Generalised Sierpinski curve, 82–88
algorithm, 85
circle-filling, 88
congruency classes of triangles, 85
continuity, 85, 87
definition, 84
grammar, 86
locality, 86
triangle-filling, 85
on triangles with curved edges, 87
Generator, 69
Gosper island, 107
Koch curve, 70
Generic Space-filling Heuristic, 235
Geographical information system, 237
Geometry modelling, 1
surface-oriented, 1
volume-oriented, 2
Geoscience applications, 163
Ghost cells, 158, 159
Hilbert order, 159, 160
refinement-tree grid, 159–160
Global refinement edge
  of a tetrahedron, 186
Gosper, W., 107
Gosper curve, 104–107
grammar, 106
variants, 108
Gosper flowsnake. See Gosper curve
Gosper island, 74, 106, 107
Grammar, 94
  $\beta\Omega$-curve, 101–102
  context-free, 136
derivation rule, 32
generated strings, 33
Gosper curve, 106
Hilbert curve, 31–36
  3D, 114–116
H-index, 99
non-terminal symbols, 31
Peano curve, 37–38
  3D, 119
production rules, 32
quasi-Sierpinski curve, 182
shape grammars, 43
Sierpinski curve, 79–80, 86
table-based implementation, 44
terminal productions, 34
terminal symbols, 31
turtle graphics, 39
Granularity of partitions, 152
Graph-filling curves, 193
Graph partitioning, 172
  algorithms, 173
  connected, 172
d-program, 172
  index-based, 162
  locality measure, 172–177
Halo cells, 159
Hamiltonian Path, 193
Hanging nodes, 157, 181, 226
Hash functions, 162
Hash table, 217
Hausdorff dimension, 72
Heat equation, 7–8, 143, 215
  residual computation, 216
  stationary problem, 7
  system of linear equations, 8, 143, 216
Hilbert, D., 17
Hilbert curve
  approximating polygon, 19, 68
  length, 69
  arithmetisation, 47–49
  basic patterns, 31, 33
  construction, 18
  continuity, 23–24, 67
definition, 21
  fractal dimension, 71
  grammar, 31–33, 133
  adaptive, 135–136
  context-free, 136
  with terminal productions, 34–36
  turtle graphics, 39–42
  higher-dimensional, 126
  iterations, 18
  as limit curve, 19
  mapping (see Hilbert mapping)
surjectivity, 22
  3D (see 3D Hilbert curve)
  traversal algorithm (see Hilbert traversal)
turtle grammar, 222
Hilbert index, 56
  algorithm, 57, 64
  operators, 56
  3D, 113
  uniqueness, 56
Hilbert mapping, 21, 22
  algorithm, 51–52, 64
  vertex labelling, 89
  finite quaternaries, 50–51
  finite state machines, 62
  infinite quaternaries, 50, 53–55
  inverse (see Hilbert index)
  non-recursive implementation, 63
  operators, 49
  3D, 113
  recursion unrolling, 60
  3D, 113
  uniqueness, 52–55
Hilbert-Moore curve, 24
Hilbert order, 6
  adaptive spacetree, 135
  matrix-vector multiplication, 200
  for optimisation, 141
  quadtree, 6, 137
Hilbert traversal, 34–36
  adaptive algorithm, 136
  with bitstream encoding, 137
  adaptive spacetree, 135
  call tree, 134
  recursion unrolling, 36
turtle-based, 42
H-index, 99–101
  grammar, 99
  iterations, 99
  locality properties, 107
Hölder continuity, 167–170, 178
parallelisation, 168–170
partition shape, 169
quasi-Sierpinski curve, 184
surface-to-volume ratio, 173
3D Hilbert curve, 168
3D quasi-Sierpinski curve, 190
3D Sierpinski curve, 184
Hölder continuous
definition, 167
H-order. See also H-index
vs. Sierpinski curve, 100

Ideal cache, 209
Image compression, 140
Image data base, 179
Image processing, 238
Index
based on SFC, 179
computation, 28
for partitioning, 146
Input stream, 219
Inverse mapping, 146
Inversion property, 220, 229
Morton order, 231
Iterations
βΩ-curve, 101
definition, 18
Hilbert curve, 18
H-index, 99
Morton order, 95
Peano curve, 25
Sierpinski curve, 78
Z-curve, 96

Join traversal, 156
algorithm, 158

k\textsuperscript{d}-spacetree. See also Spacetree
definition, 129
Kd-trees, 237
K\textsuperscript{-}Median problem, 236
Knopp, K., 63, 77
Koch curve, 63, 70, 72, 74
construction, 70
length, 70–71
Koch snowflake, 73

Last-access order, 220
Last-in-first-out, 219

Lebesgue curve, 97–98, 138
continuity, 98
definition, 97
vs. Morton order and Z-Curve, 97
Left-right splitting (via SFC), 161
Length of coast lines, 72, 73
Load balancing, 9, 145, 149
diffusion approach, 149
Load distribution, 144
exchange subgrid, 155
subtree-based, 150–153
Locality measures, 177
discrete, 171
graph partitions, 172–177
for index/inverse mapping, 179
for iterations of SFC, 171
partitions, 178
Locality of data, 12
Locality preserving, 94, 146
Locality properties, 185
βΩ-curve, 107
H-index, 107
Local refinement edge
of a tetrahedron, 186
Longest-edge bisection, 192
Loop unrolling, 202
L-systems, 33, 43
LU-decomposition, 212

Mandelbrot, B., 73
Manhattan distance, 171
Mapping
computation, 28
Hilbert curve (see Hilbert mapping)
Peano curve (see Peano mapping)
Sierpinski curve (see Sierpinski mapping)
Master-slave structure, 152
Matrix multiplication
algorithm, 202
blocking and tiling, 212
Peano curves (see Peano matrix multiplication)
as 3D traversal, 202
Matrix operations, 163
Matrix storage, 213
conversion of formats, 213
Matrix-vector multiplication, 199–201
algorithm, 199, 200
Hilbert traversal, 201
cache efficiency, 199–201
Hilbert order, 200
Maximum distance, 171
Memory-bound performance, 196–197
Memory gap, 195
Modified depth-first traversal, 153–155
algorithm, 156
Molecular dynamics, 163, 233
Moore, E. H., 24, 29
Morton order, 94–96, 107, 138, 140, 229–232
construction, 95
inversion property, 231
iterations, 95
non-continuous, 95
for optimisation, 141
projection property, 229
quadtree, 139
Multicore CPUs, 196
Multidimensional arrays, 10
Multidimensional data, 9
algorithms and operations, 10
Multigrid method, 9

N-body problem, 162
Nearest-neighbour problem, 237
Neighbour relations, 12, 13, 146
Nested intervals, 21, 48
Netto, E., 17
Newest vertex bisection, 89, 141
Node-connected, 94, 173, 182, 189
Non-terminal symbols. See Grammar
Non-uniform memory access, 196
N-order, 96
Norm cell scheme, 2, 131
number of cells, 2
n-tuple, 10
NUMA. See Non-uniform memory access
Numerical linear algebra, 13
NURBS, 1

Octree, 3, 4, 107, 129
computer graphics, 140
grid generation, 140
number of grid cells, 132
$2^3$-spacetree, 130
Old/new classification of edges, 224
Output stream, 220

Padding, 210
Palindrome property, 228, 232
Parameterised by volume, 169, 191
Lebesgue/Peano/Sierpinski curve, 170
3D Hilbert curve, 169

Particle-base simulation, 163
Partition boundaries
left and right part, 161
subtree-based partitioning, 152
Partitioning, 9, 10
criteria for efficiency, 144–146
index-based, 146–147
parallel sorting, 147
refinement-tree, 149–150
parallel algorithm, 150
sequentialised refinement trees, 153–156
software, 163
space-filling curves, 146–156, 162, 173
subtree-based, 150–153, 162
traversal-based, 148–149
unstructured grids, 148

Partitions, 144
compact, 145, 170
connected, 173–177, 179
data exchange, 157–162
disconnected, 174
length of boundary, 145
locality measures, 178
Morton order, 175–177
number of unknowns, 144
spacetree, 175
surface-to-volume ratio, 173, 178
Peano, G., 17, 25
Peano curve
approximating polygon, 68
arithmetisation, 57–59
construction, 25
construction by Peano, 122
continuity, 27
dimension recursive, 116–117, 122
fractal dimension, 71
grammar, 37–38
dimension recursive, 116–117
iterations, 25
grids of arbitrary size, 120–122
mapping (see Peano mapping)
Meurthe order, 29
notation, 29
projection property, 232
$5 \times 5$ or $7 \times 7$ refinement, 119
surjectivity, 26
switch-back type, 120
3D (see 3D Peano curve)
traversal algorithm (see Peano traversal)
variants, 29
Meander type (see Peano-Meander curve)
switch-back type, 26
Peano index, 146
Peano iterations
  grids of arbitrary size, 210
Peano mapping
  algorithm, 59
  dimension recursive, 122
  operators, 57–59
  Peano’s original formulation, 122
Peano matrix multiplication, 201–210
  block-recursive scheme, 204–206
  cache efficiency, 206–210
  cache misses, 208–210
  contiguous access, 208
  increment/decrement access, 205, 206
  matrices of arbitrary size, 210
  memory access pattern, 205–206
    locality properties, 206
    parallelisation, 210, 212
    recursive blocking, 208
    recursive implementation, 207
    $3 \times 3$-scheme, 203
Peano-Meander curve, 26
Peano order
  grids of arbitrary size, 120–122
  for matrix elements, 204
  spacetree, 139
Peano scan, 237
Peano traversal, 38, 232
  grids of arbitrary size, 121
Performance
  daxpy operation, 197
  matrix multiplication, 197
  memory-bound, 196–197
Plotter, 235
Production rules. See Grammar
Projection property, 232
  Morton order, 229

Quadtree, 3–7, 107, 129, 137, 140, 213
  bitstream encoding, 138, 140
  boundary cells, 131
  construction, 3, 4
  depth-first traversal, 5
  ghost cells, 159
  Hilbert order, 6, 138, 140
  Morton order, 139
  number of grid cells, 131–132, 140
  restricted quadtree, 141
  sequentialisation by Hilbert curve, 133
  sequential order, 3–7
  $2^3$-spacetree, 130
  traversal, 3
  Z-order, 5

Quasi-Sierpinski curve, 182–184
  algorithm, 183
  approximating polygon, 183
  grammar, 182
  Hölder continuity, 184
  mapping, 183
  3D, 189–191
    Hölder continuity, 190–191
Quasi-Sierpinski order
  on triangular meshes, 184
Quaternary representation, 48
Queue data structures, 230
Queue property
  (violation by) Morton order, 231
  Range queries, 237
  Recursion unrolling, 60–62
  Recursive blocking (for matrix storage), 213
  Recursive SFC, 24, 107, 168, 170
    definition, 93
  Red-black refinement, 186
  Red-green refinement, 181
  Refinement bit, 134
  Refinement cascade, 226
  Refinement tree, 138, 141, 162, 175, 193, 225
    parallel grid partitions, 155–156
    refinement-tree partitioning, 149–150
  Residual computation
    element-based, 218
  Residuals, 216
  Row-major, 12, 199
  R-trees, 237
  Runtime complexity, 196

Sagan, H., 28
Self-avoiding walks, 193
Self-similar, 94
Self-similarity, 48
Separator, 160
Sequentialisation, 9–13
  column-major, 12
  row-major, 12
Sequentialisation by SFC, 27
Sequential order, 3, 146
  family, 11
  Hilbert curve, 6
  locality, 6–7
  requirements, 11–12
Shared memory, 153
Sierpinski, W., 77
Sierpinski curve, 100
  approximating polygon, 79
Index

arithmeticisation, 80–81
construction, 77
definition, 77
generalised curve (see Generalised Sierpinski curve)
graham, 79–80, 100
vs. H-order, 100
iterations, 78
mapping (see Sierpinski mapping)
node-connected (see Quasi-Sierpinski curve)
parallelisation, 162
3D (see 3D Sierpinski curve)
turtle grammar, 223
Sierpinski-Knopp curve, 77
Sierpinski mapping, 63, 81–82
algorithm, 82
non-recursive implementation, 82
operators, 81
Sierpinski order
adaptive triangular grid, 140
red-green refinement, 184
triangle strips, 141
Signal processing, 237
Simple SFC, 107, 109
definition, 94
Smoothed Particle Hydrodynamics, 163
Space-filler, 94, 107
Space-filling curve
connected (see Connected SFC)
definition, 17
recursive (see Recursive SFC)
Spacetree, 175, 211
adaptive, 130
adaptive traversal, 134–135
definition, 129
k^d-spacetree, 129
numerical simulation on spacetree grids, 215–232
Peano order, 139
regularly refined, 130
sequentialisation, 132
Hilbert order, 135
Spacetree grid
access to neighbour cells, 216–217
conforming refinement, 226–227
ghost cells, 159–160
Spacetree traversal
element-based, 217–219
first-access order, 220
last-access order, 220
Morton order, 229–232
multiple access to unknowns, 218
Peano order, 232
stack-and-stream scheme, 220
Sparse matrix, 211
Hilbert order, 213
Peano order, 211
quadtree storage scheme, 213
spacetree storage scheme, 211–212
Spatial locality, 198
Split traversal, 155
Stack-based traversal
adaptive grid, 226
algorithm, 225–226
dge-located unknowns, 227, 254
memory efficiency, 227–228
old/new classification, 223–224
Peano curves, 229
stack rules, 222–225
Stack property, 219, 228
Hilbert curve, 219
partition boundaries, 161
Peano curve, 221
Sierpinski curve, 221
(violation by) Hilbert order, 229
Surface-oriented geometry modelling, 1
Surjective, 17
Tagged edge, 83, 124
Temporal locality, 198
Terminal symbols. See Grammar
Tetrahedral grids, 181, 184–191
angles of tetrahedra, 188
bisection refinement, 184–188, 191–192
shapes of tetrahedra, 186
longest-edge bisection, 192
red-black refinement, 187, 192
Tetrahedral meshes. See Tetrahedral grids
Tetrahedral strips, 141, 233
Tetrahedron with tagged edge, 124
3D Hilbert curve, 109–116
approximating polygon, 110, 111
arithmeticisation, 113
basic patterns, 110
rotation, 111, 112
face-connected, 109
grammar, 114–116
number of terminals, 114
Hölder continuity, 168
mapping, 113
operators, 113
parameterised by volume, 169
variants, 109–112, 126
number of different curves, 112
3D Hilbert index, 113
3D Hilbert traversal
   palindrome/stack property, 229, 230
3D Peano curve, 116–119
   basic patterns, 116
   dimension recursive, 116, 117
   grammar, 119
   projection property, 202, 203
   switch-back type, 116
3D Sierpinski curve, 123–125
   algorithm, 125
   definition, 124
   face-connected, 125
   Hölder continuity, 184
   tetrahedral strips, 141
3D Sierpinski order, 192
Tiling, 13, 202
Topological monsters, 235
Translation lookaside buffer, 213
Travelling salesman problem, 28, 236
Traversal, 3, 10
   on adaptive spacetrees, 134–135
   algorithm, 34
      Hilbert curve (see Hilbert traversal)
      Peano curve (see Peano traversal)
   computational costs, 36
   depth-first (see Depth-first traversal)
      of a matrix, 200
      modified depth-first, 153
      in SFC order, 28
      turtle-based vs. plotter-based, 42
Tree algorithms, 163
Triangle strips, 141, 233
   swap command, 142
Triangle with tagged edge, 83
Triangular grid, 8, 144, 181–184
   adaptive, 141
red-green refinement, 181–184
Sierpinski order, 140
Triangular meshes, 141
Tuple, 10
Turtle grammar
   Hilbert curve, 222
   Sierpinski curve, 223
Turtle graphics, 33
   βΩ-curve, 102
   Gosper curve, 106
   grammar, 39
UB-trees, 237
Uniformly continuous, 23, 94
Uniqueness
   of inverse mapping, 56
   of SFC mappings, 52–55
Vertex labelling, 84, 89, 125, 183,
   190
Volume-oriented geometry modelling,
   2
Wire-frame model, 1, 2
Working set, 201
Work pool approach, 152
Work stealing, 153
Wunderlich, W., 29
Z-curve, 96, 140
   iterations, 96
   mapping, 96
Z-order. See Z-curve