Organisation of the Course

Sonntag, 19. April 2020 20:57

Algorithms for Scientific Computing

Organisation of the Course

Tobias Neckel
Technical University of Munich

Summer 2020

Content

Space-Filling Curves 1/4 TN

- Hilbert Curve
- Peano Curve
- 2D vs. 3D
- Applications

Hierarchical Numerical Methods 1/2

- Hierarchical Basis (1D, *N*D)
- Sparse Grids
- Applications

Discrete Fourier Transform & related Transforms 1/4

- Fast Fourier Transform
- Fast Discrete Cosine/Sine Transform
- Applications

Persons

Lecture

- Tobias Neckel
- Felix Dietrich
- Christian Mendl

Tutorials

- Jean-Matthieu Gallard
- Santiago Narvaez

Corona Aspects

Lecture

- Asynchronous: Recorded videos
- Will be uploaded shortly before the corresponding lecture slot
- If desired: virtual office hours of lecturer once per week

Tutorials

- Asynchronous: WS + solutions published
- Synchronous: Interactvive Q & A tutorial session
- For more details: See Moodle page

Exams

- State / TUM policy not yet clear
- Probably: usual format (written) in usual period (end of term) in unusual mode (other rooms, more distance)

Technical Aspects

- Moodle: Landing page, contains everything
- Interactive parts: either via zoom or BigBlueButton (will be announced via Moodle)

Lecture 1 - From Quadtrees to Space-Filling Orders

Donnerstag, 16. April 2020 16:21

3 parts:

I. Quadtrees
II. Hilbert Orders
III. Applications of Space-Filling Orders

slides

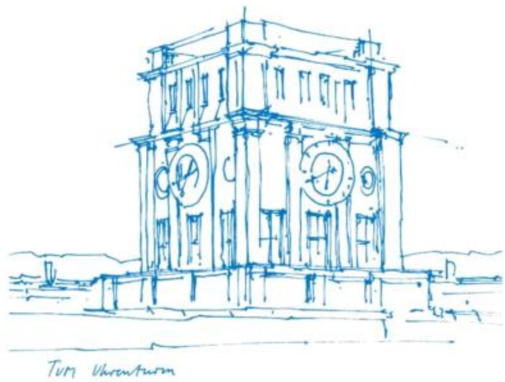
Sonntag, 19. April 2020 09:33

Algorithms for Scientific Computing

From Quadtrees to Space-Filling Orders

Tobias Neckel
Technical University of Munich

Summer 2020



Part I

Quadtrees

Overview: Modelling of Geometric Objects

Surface-oriented models:

CAD

- wire-frame models
- augmented models using Bezier curves and planes
- typically described by graphs on nodes, edges, and faces

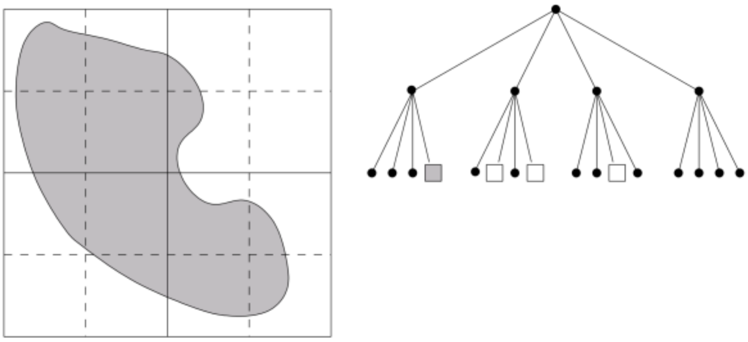
Volume-oriented models:

- Constructive Solid Geometry (boolean operations on primitives)
- voxel models: place object in a grid
- octrees: recursive refinement of voxel grids
- quadtrees: 2D analogon (voxel $\rightsquigarrow$ pixel)

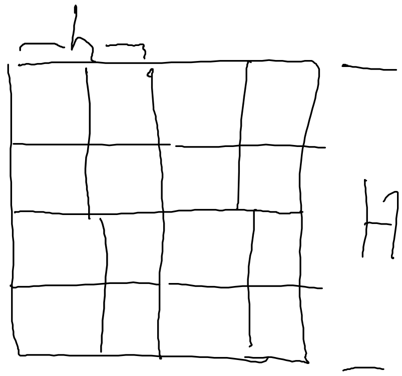
Seite 4

Sonntag, 19. April 2020 09:33

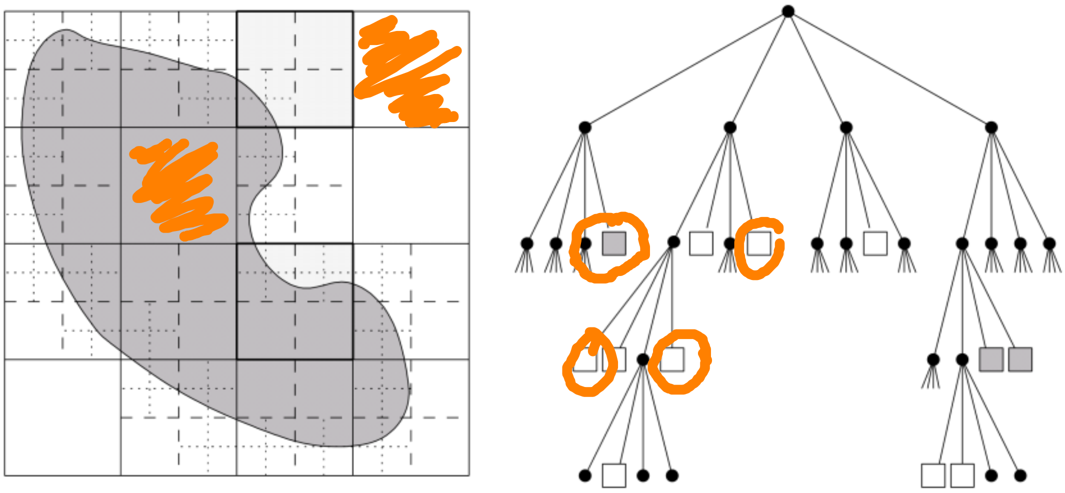
Quadtrees to Describe Geometric Objects



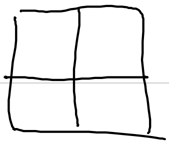
- start with an initial square (covering the entire domain)
- recursive substructuring in four subsquares



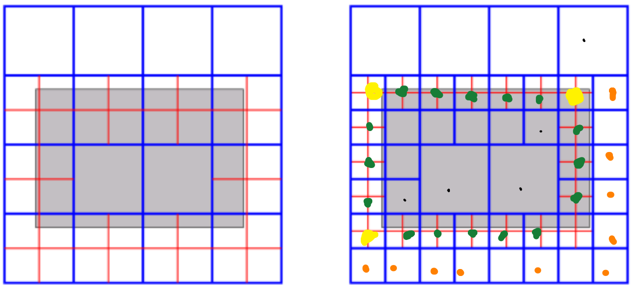
Quadtrees to Describe Geometric Objects



- start with an initial square (covering the entire domain)
- recursive substructuring in four subsquares
- adaptive refinement possible
- terminate, if squares entirely within or outside domain



Number of Quadtree Cells to Store a Rectangle



k=2 k=3

Terminal (t_k) and boundary (b_k) cells after k refinement steps (for $k > 2$):

$$\begin{aligned} b_k &= 2 \cdot b_{k-1} \\ t_k &= t_{k-1} + 2 \cdot b_{k-1} \end{aligned} \quad \Rightarrow \quad \begin{aligned} b_k &= 2^{k-2} \cdot b_2 = \tfrac{5}{2} \cdot 2^k \\ t_k &= \ldots = 5 \cdot 2^k - 14 \end{aligned}$$

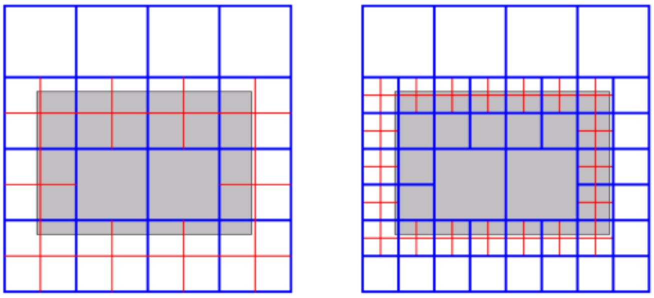
Tobias Neckel | Algorithms for Scientific Computing | Quadtrees & Space-Filling Orders | Summer 2020 5

k	b	t
	$b_0 = 1$	$t_0 = 0$
	$b_1 = 4$	$t_1 = 0$
	$b_2 = 10$	$t_2 = 6$
	$b_3 = 4 + 16 = 20$	$t_3 = 6 + 7 + 13 = 26$
	$b_4 = 4 + 36 = 40$	$t_4 = 26 + 17 + 23 = 66$

$$b_k = 2 b_{k-1} = \underbrace{2 \cdot 2 \cdots 2}_{k-2} b_2$$

$$= 2^{k-2} \cdot \frac{5}{2} \cdot 2 \cdot 2 = \frac{5}{2} 2^k$$

Number of Quadtree Cells to Store a Rectangle



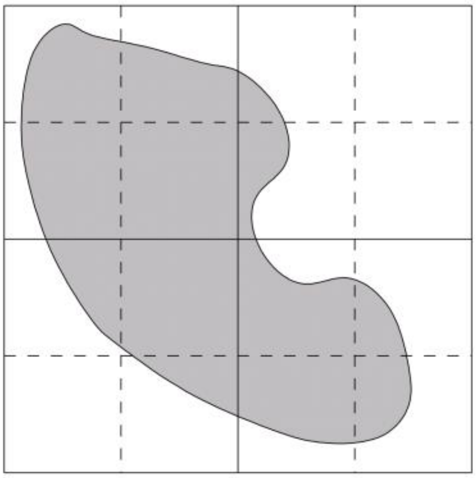
- uniformly ref. voxel-grid (level k): $(2^{d=2})^k = (2^k)^2 =: \mathcal{O}(N^2)$ cells
- quadtree-refined grid (level k): $\frac{15}{2} \cdot 2^k - 14 =: \mathcal{O}(N)$ cells

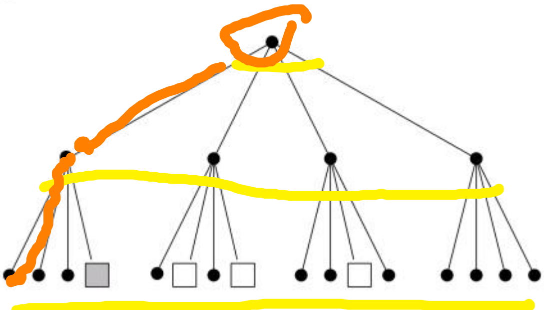
$\Rightarrow$ number of cells proportional to length of boundary ($N := 2^k$)

Tobias Neckel | Algorithms for Scientific Computing | Quadtrees & Space-Filling Orders | Summer 2020 5

$$t_k = t_{k-1} + 2b_{k-1}$$
$$= t_{k-2} + 2b_{k-2} + 2b_{k-1}$$
$$= \dots = t_2 + 2(b_{k-1} + \dots + b_2)$$
$$= t_2 + 2\sum_{j=2}^{k-1} b_j$$
$$= t_2 + 2\sum\left(\frac{5}{2}2^j\right)$$
$$t_2 + 5 \cdot \left(\sum 2^j\right)$$
$$= t_2 + 5\left(\frac{2^k - 1}{2-1} - 2^0 - 2^1\right)$$
$$= t_2 + 5 \cdot 2^k - 20 = 5\,2^k - 14$$

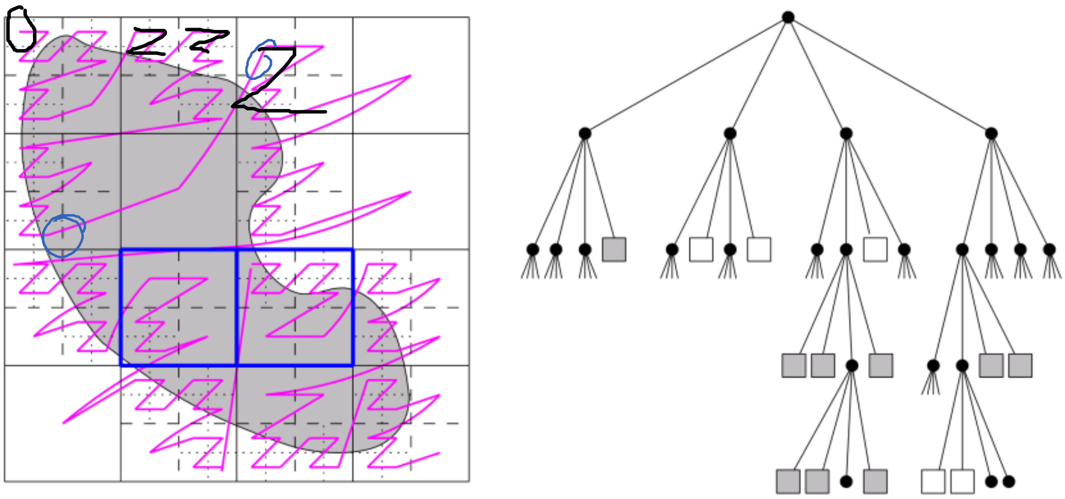
Storing a Quadtree – Sequentialisation





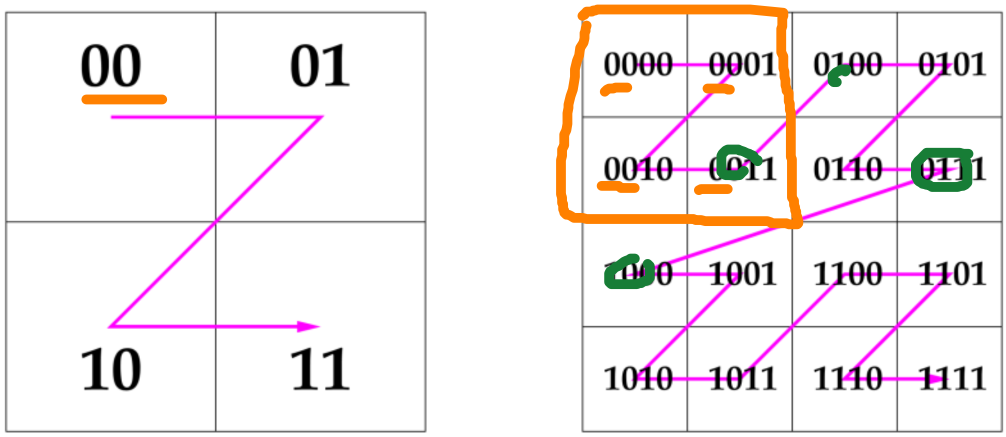
- sequentialise cell information according to **depth-first traversal**
- relative numbering of the child nodes determines sequential order

Storing a Quadtree – Sequentialisation



- sequentialise cell information according to **depth-first traversal**
- relative numbering of the child nodes determines sequential order
- here: leads to so-called **Morton order**

Morton Order ("Z curve")



Relation to bit arithmetics:

- odd digits: position in vertical direction
- even digits: position in horizontal direction

Morton Order and Cantor's Mapping

Georg Cantor (1877):

$$0.01111001\ldots \rightarrow \begin{pmatrix} 0.0110\ldots \\ 0.1101\ldots \end{pmatrix}$$

- **bijective** mapping $[0,1] \rightarrow [0,1]^2$
- proved identical cardinality of $[0,1]$ and $[0,1]^2$
- provoked the question: is there a **continuous** mapping? (i.e. a curve)

Similar: is there a contiguous order on the quadtree cells?

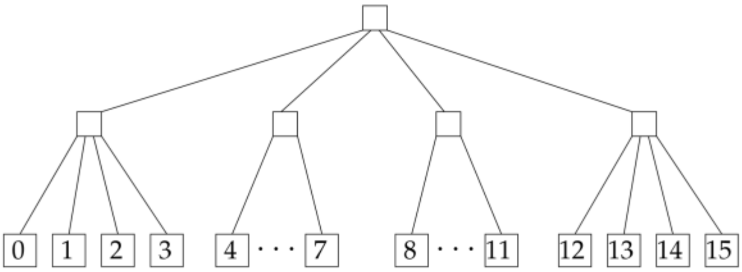
Preserving Neighbourship for a 2D Octree

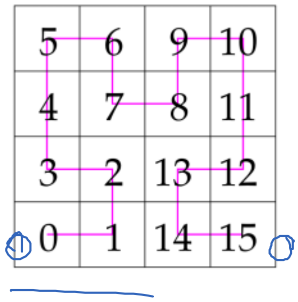
Requirements:

- consider a simple 4×4-grid
- uniformly refined
- subsequently numbered cells should be neighbours in 2D

Shared edges

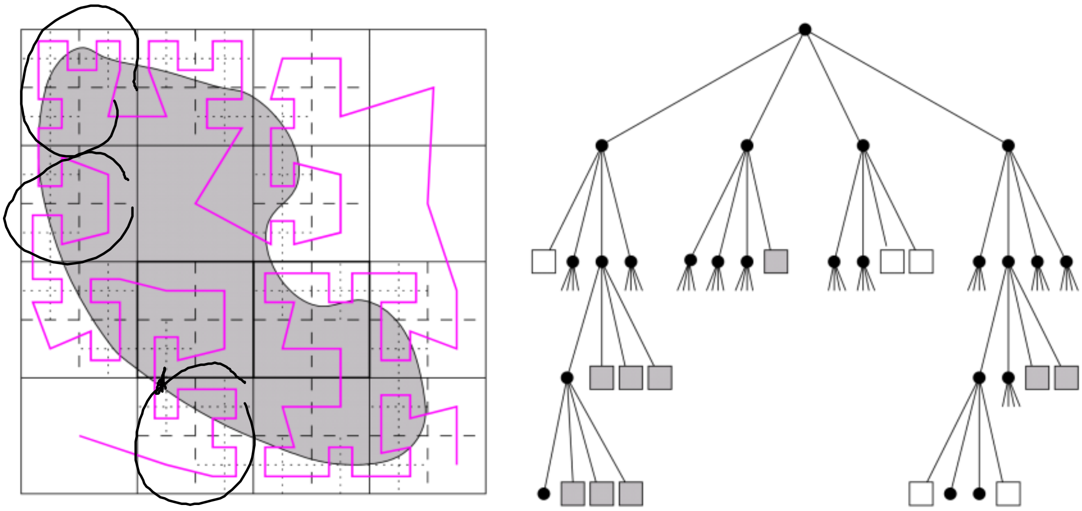
Leads to (more or less unique) numbering of children:





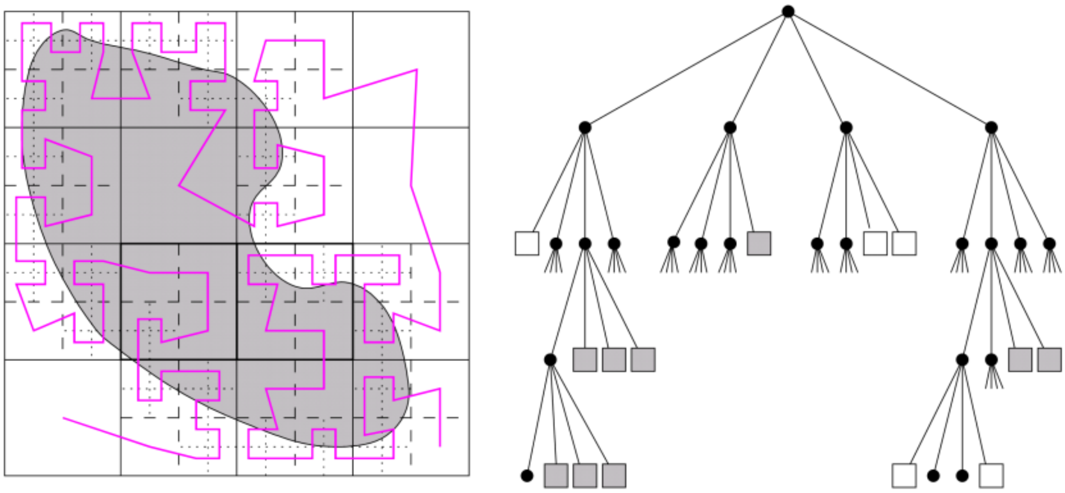
Hilbert

Preserving Neighbourship for a 2D Octree (2)



- adaptive refinement possible
- neighbours in sequential order remain neighbours in 2D

Preserving Neighbourship for a 2D Octree (2)



- adaptive refinement possible
- neighbours in sequential order remain neighbours in 2D
- here: similar to the concept of **Hilbert curves**

Open Questions

Algorithmics:

- How do we describe the sequential order algorithmically?
- What kind of operations are possible?
- Are there further “orderings” with the same or similar properties?

Applications:

- Can we quantify the “neighbour” property?
- In what applications can this property be useful?
- Which other properties and/or operations can be useful?

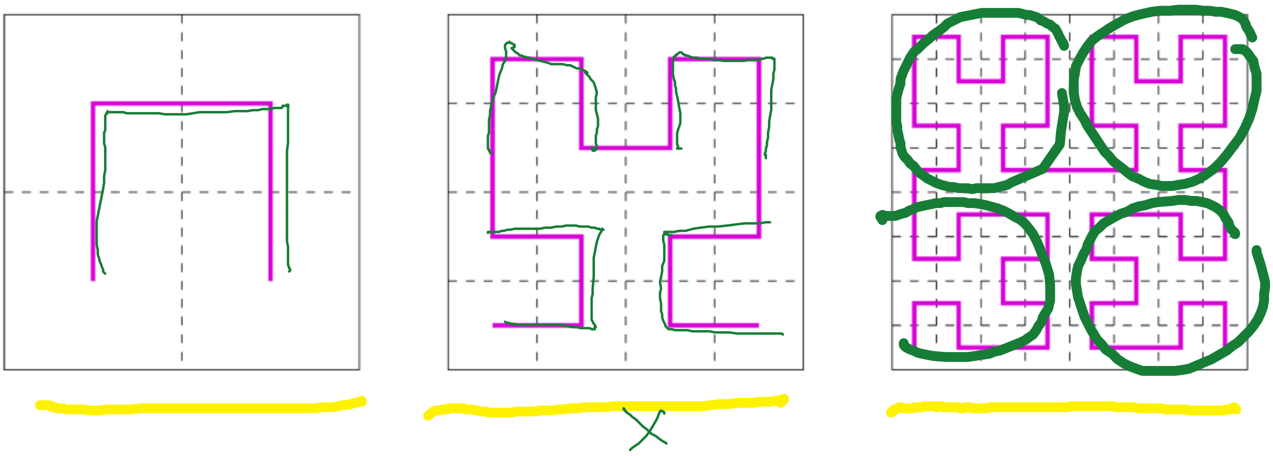
Part II

Hilbert Orders

Seite 17

Sonntag, 19. April 2020 09:33

Construction of the Hilbert Order



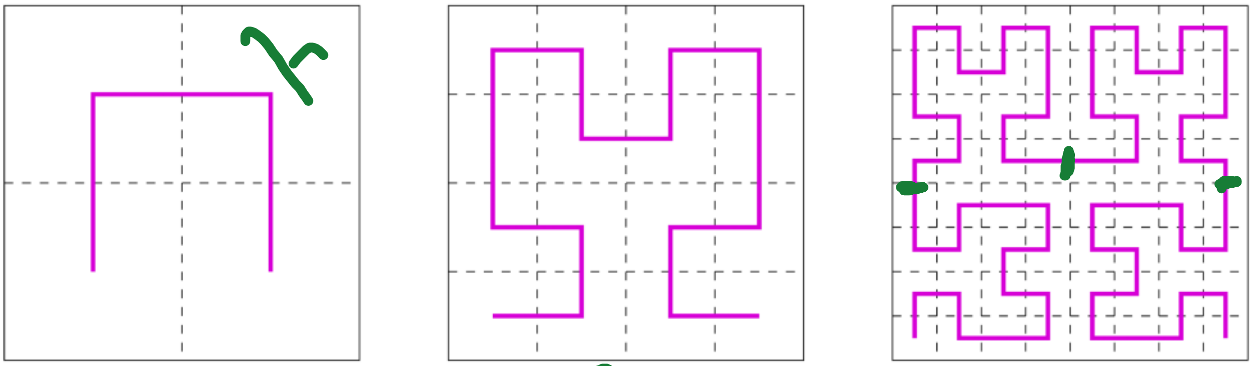
Incremental construction of the Hilbert order:

- start with the basic pattern on 4 subsquares
- combine four numbering patterns to obtain a twice-as-large pattern
- proceed with further iterations

Seite 18

Sonntag, 19. April 2020 09:33

Construction of the Hilbert Order

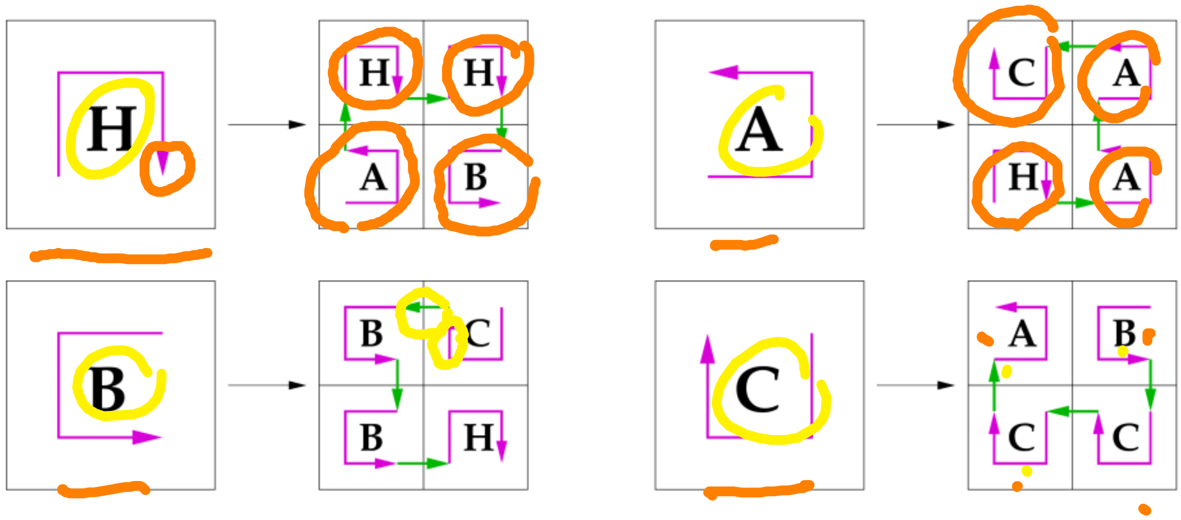


Recursive construction of the Hilbert order:

- start with the basic pattern on 4 subsquares
- for an existing grid and Hilbert order:
  split each cell into 4 congruent subsquares
- order 4 subsquares following the rotated basic pattern,
  such that a contiguous order is obtained

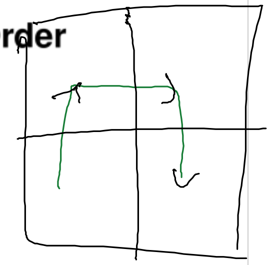
A Grammar for Describing the Hilbert Order

Examine pattern during the construction of the Hilbert order:



→ motivates a **Grammar** to generate the iterations

A Grammar for Describing the Hilbert Order



needs

- Non-terminal symbols: $\{H, A, B, C\}$, start symbol H
- terminal characters: $\{\uparrow, \downarrow, \leftarrow, \rightarrow\}$
- productions:

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

- replacement rule: in any word,
  all non-terminals have to be replaced at the same time
  $\rightarrow$ L-System (Lindenmayer)

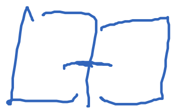
$\Rightarrow$ the arrows describe the **iterations of the Hilbert curve** in "turtle graphics"

Subj pred. obj. } Sentence

Using the Grammar to Describe the Hilbert Curve

The grammar for the Hilbert order prepares the mathematical definition of the curve (and proof of continuity):

- there are only four basic patterns that occur (corresp. to the symbols $\{H, A, B, C\}$ of the grammar)
  $\rightarrow$ **closed recursive system!**
- two subsequent subsquares of the Hilbert-curve construction share a common edge(!)
  $\rightarrow$ follows from the fact that the move operators $\{\uparrow, \downarrow, \leftarrow, \rightarrow\}$ are sufficient to describe the operators
  $\rightarrow$ **contiguous order!** (leads to continuity of the curve)
- last but not least:
  we have formalised the construction of the iterations
  (towards formal definition of a mapping)



Part III

Applications of Space-Filling Orders

Sequentialising Multi-dimensional Data

Examples of multi-dimensional data structures:

- Matrices
- Image data (images, tomographic data, movies, . . .)
- discretisation meshes (to discretise mathematical models in physics/. . . ; PDE, etc.)
- Coordinates (often used in connection with graphs)
- tables (also in data bases)
- in computational finance and financial mathematics: “baskets” of stocks/options/. . .

Sequentialising Multi-dimensional Data (2)

Typical algorithms and operations:

- traversal (update/processing of all data; simulation meshes, e.g.)
- matrix operations (linear algebra, etc.)
- sequentialisation (e.g. to store data on discs or in main memory)
- partitioning of data (for parallelisation or in divide-and-conquer algorithms)
- sorting of data (to simplify further operations)
- in general: nested loops

```
for i from 1 to n do
    for j from 1 to m do          ...
```

Demands on Efficient Sequentialisation

Effective Sequentialisation:

- unique numbering $\Rightarrow$ requires bijective mapping
- sequentialisation without "holes" (for data structures, e.g.)

Efficient Sequentialisation:

- preserve neighbourship properties $\Rightarrow$ data locality
- fast, simple index computation
- "smoothness", stability vs. small changes
- dimensional symmetry (no fast or slow dimensions)
- "clustering" of data

Application Examples

- **range queries** in image and raster data bases
- **image browsing** and **image search** in image collections
- heuristical approaches for graph-based algorithms (nearest neighbour, traveling salesman)
- collision detection
- **parallelisation** of data
- efficient use of **cache memory** (in simulations, e.g.)

Also: everything that involves Quadtrees/Octrees:

- Searching (collision detection, access to surface representations, etc.)
- Fast access to level-of-detail infos (geodata, graphics, games, . . .)
- Dynamically adaptive meshes in scientific computing
- many more . . .