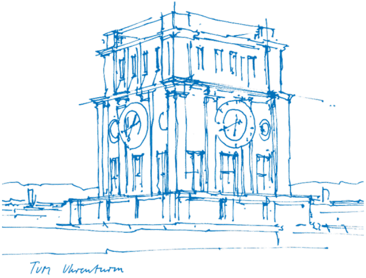
Algorithms for Scientific Computing

From Quadtrees to Space-Filling Orders

Michael Bader
Technical University of Munich

Summer 2022



Part I

Quadtrees

Overview: Modelling of Geometric Objects

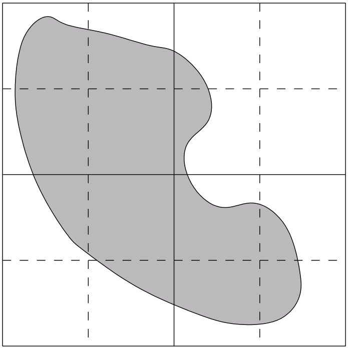
Surface-oriented models:

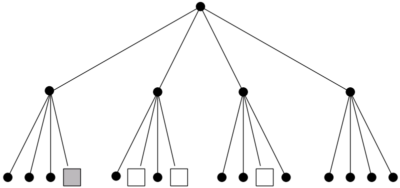
- wire-frame models
- augmented models using Bezier curves and planes
- typically described by graphs on nodes, edges, and faces

Volume-oriented models:

- Constructive Solid Geometry (boolean operations on primitives)
- voxel models: place object in a grid
- octrees: recursive refinement of voxel grids

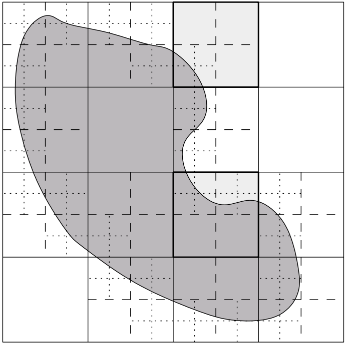
Quadtrees to Describe Geometric Objects

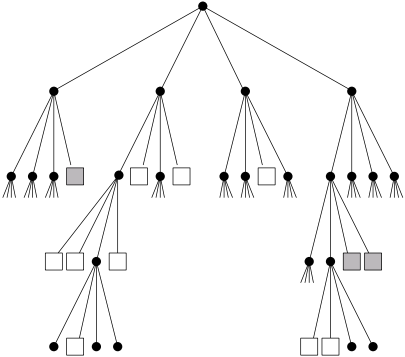




- start with an initial square (covering the entire domain)
- recursive substructuring in four subsquares

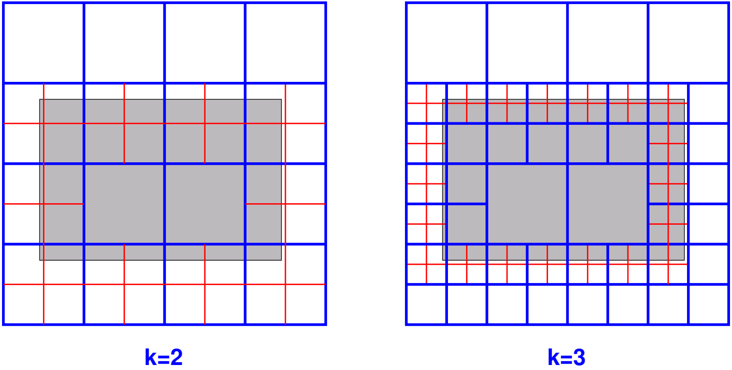
Quadtrees to Describe Geometric Objects





- 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

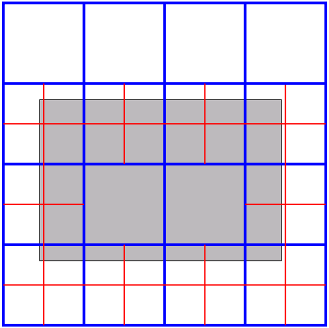
Number of Quadtree Cells to Store a Rectangle



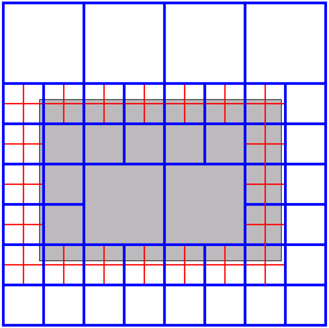
Terminal (t_k) and boundary (b_k) cells after k refinement steps:

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

Number of Quadtree Cells to Store a Rectangle



k=2

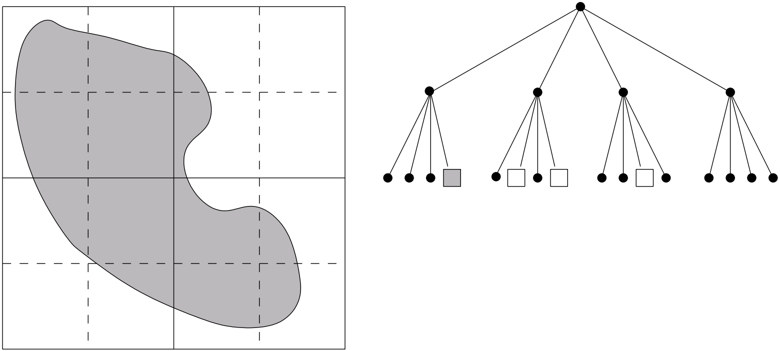


k=3

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

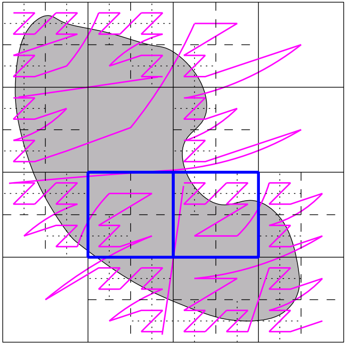
  $\Rightarrow$ number of cells proportional to length of boundary ($N := 2^k$)

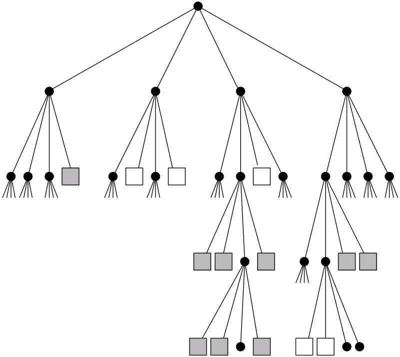
Storing a Quadtree – Sequentialisation



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

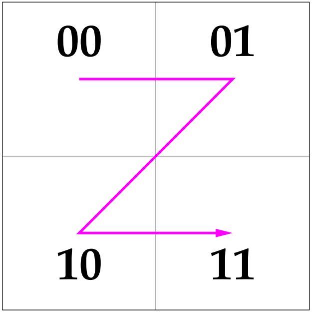
Storing a Quadtree – Sequentialisation

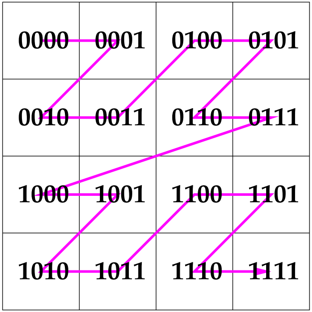




- sequentialise cell information according to **depth-first traversal**
- relative numbering of the child nodes determines sequential order
- here: leads to so-called **Morton order**

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)

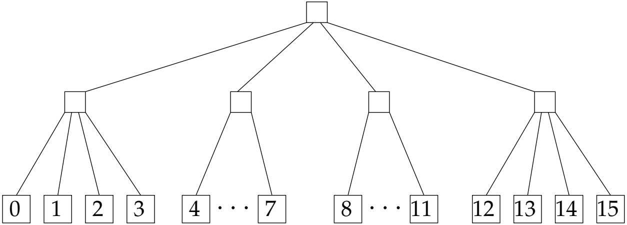
Similar: is there a contiguous order on the quadtree cells?

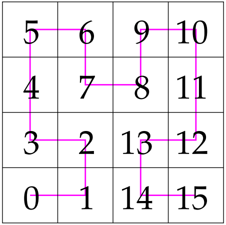
Preserving Neighbourship for a 2D Octree

Requirements:

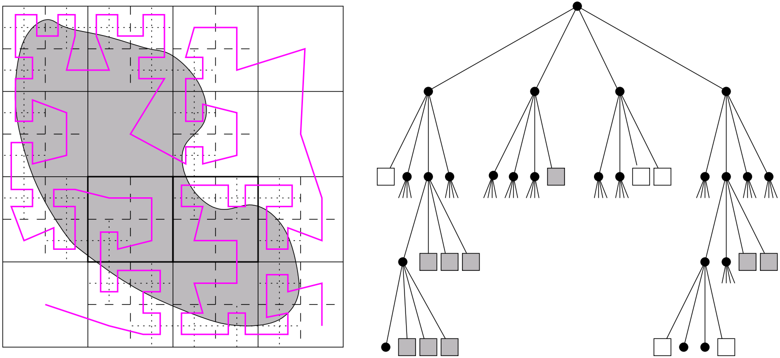
- consider a simple 4×4-grid
- uniformly refined
- subsequently numbered cells should be neighbours in 2D

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



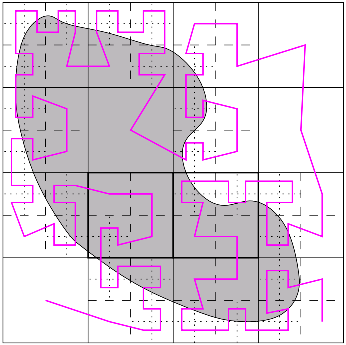
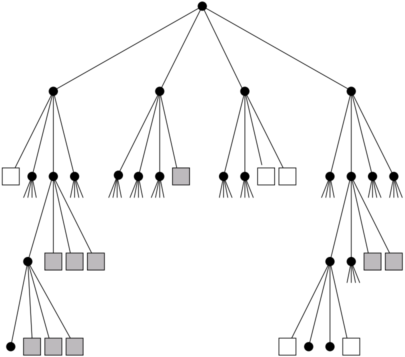


Preserving Neighbourship for a 2D Octree (2)



- 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?

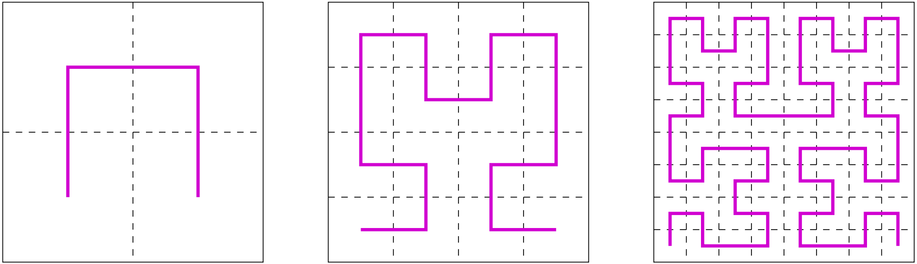
Applications:

- 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

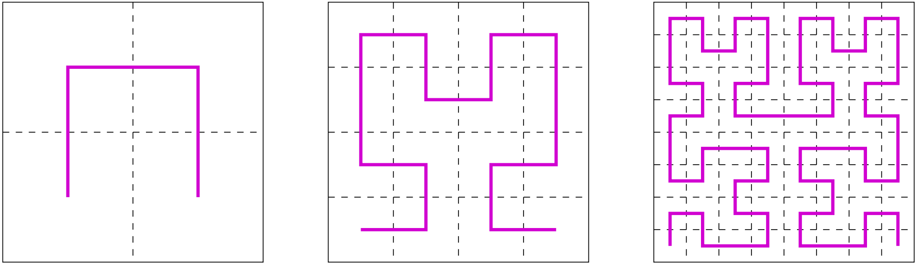
Construction of the Hilbert Order



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

Construction of the Hilbert Order

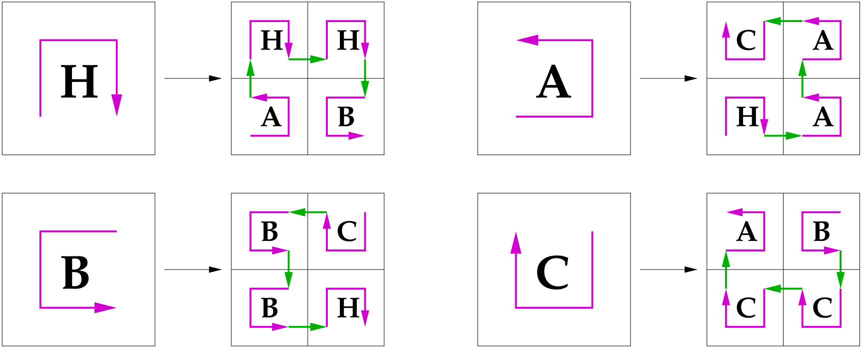


Recursive construction of the Hilbert order:

- 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:



$\rightarrow$ motivates a **Grammar** to generate the iterations

A Grammar for Describing the Hilbert Order

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

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

- 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"

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 . . .