HPC – Algorithms and Applications

Structured Grids and Space-Filling Curves

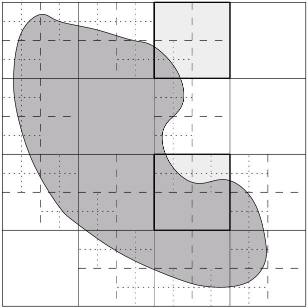
Michael Bader

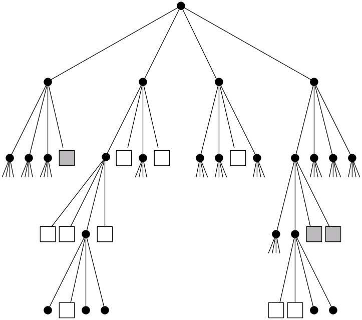
Winter 2013/2014

Part I

From Quadtrees to Space-Filling Curves

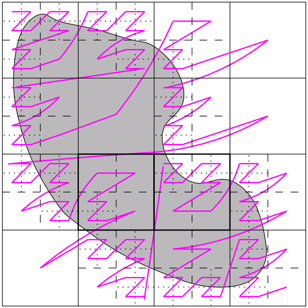
Quadtrees to Describe Geometric Objects

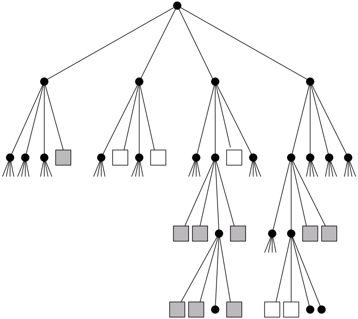




- start with an initial square (covering the entire domain)
- recursive substructuring in four subsquares
- adaptive refinement possible
- terminate, if squares entirely within or outside domain

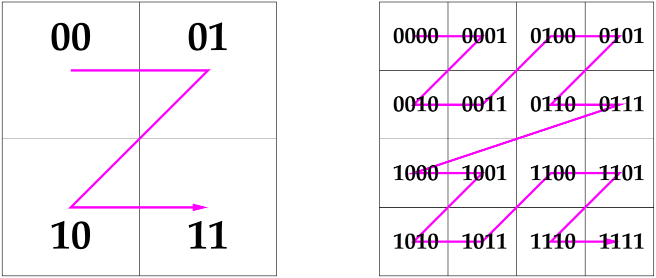
Storing a Quadtree – Sequentialisation





- 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



Relation to bit arithmetics:

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

Morton Order and Cantor's Mapping

Bit interleaving:

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

Georg Cantor (1877):

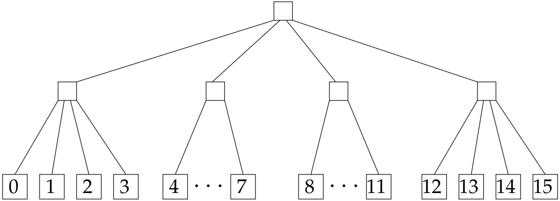
- a **bijective** mapping $[0,1] \rightarrow [0,1]^2$ exists
- proved identical cardinality of $[0,1]$ and $[0,1]^2$
  (Cantor: *"I see it but I don't believe it!"*)
- provoked the question: is there a **continuous** mapping?
  (i.e. a curve)

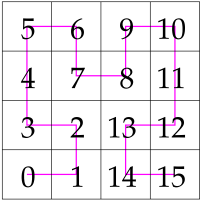
Preserving Neighbourship for a 2D Octree

Requirements:

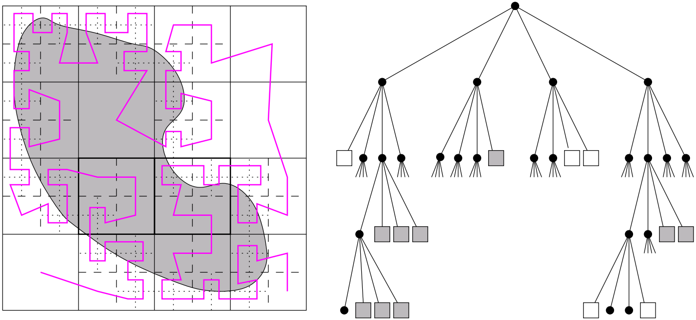
- consider a simple 4×4-grid
- uniformly refined
- subsequently numbered cells should be neighbours in 2D

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
- 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?
- What further operations

Part II

Space-Filling Curves

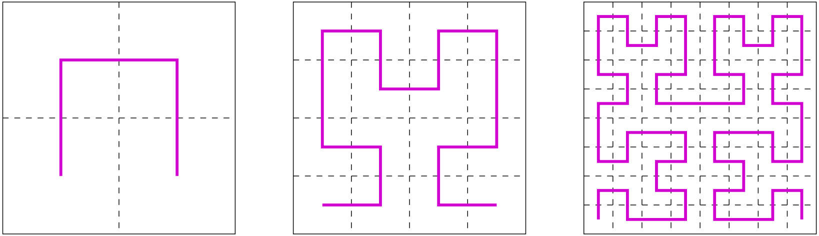
Definition of a Space-filling Curve

Given a continuous, surjective mapping $f\colon \mathcal{I} \to \mathcal{Q} \subset \mathbb{R}^n$, then $f_*(\mathcal{I})$ is called a *space-filling curve*, if $|\mathcal{Q}| > 0$.

Comments:

- a *curve* is defined as the image $f_*(\mathcal{I})$ of a continuous mapping $f\colon \mathcal{I} \to \mathbb{R}^n$
- *surjective*: every element in $\mathcal{Q}$ occurs as a value of f, i.e., $\mathcal{Q} = f_*(\mathcal{I})$
- $\mathcal{I} \subset \mathbb{R}$ and $\mathcal{I}$ is compact, typically $\mathcal{I} = [0, 1]$
- if $\mathcal{Q}$ is a smooth manifold, then there can be *no bijective* space-filling mapping $f\colon \mathcal{I} \to \mathcal{Q} \subset \mathbb{R}^n$ (theorem: E. Netto, 1879).

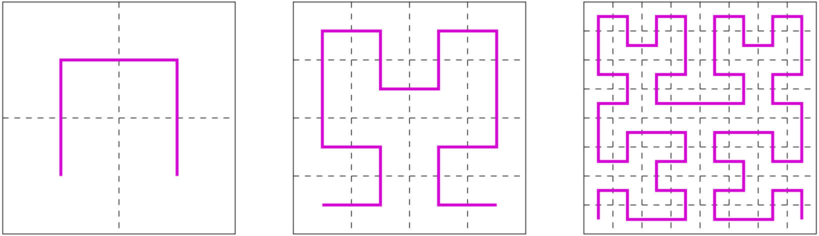
Example: Construction of the Hilbert curve



Iterations of the Hilbert curve:

- start with an iterative numbering of 4 subsquares
- combine four numbering patterns to obtain a twice-as-large pattern
- proceed with further iterations

Example: Construction of the Hilbert curve

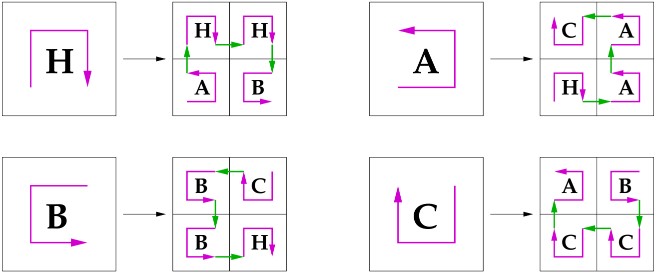


Recursive construction of the *iterations*:

- split the quadratic domain into 4 congruent subsquares
- find a space-filling curve for each subdomain
- join the four subcurves in a suitable way

A Grammar for Describing the Hilbert Curve

Construction of the iterations of the Hilbert curve:



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

A Grammar for Describing the Hilbert Curve

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

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

- 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"

Definition of the Hilbert Curve's Mapping

Definition: (Hilbert curve)

- each parameter $t \in \mathcal{I} := [0, 1]$ is contained in a sequence of intervals

$$\mathcal{I} \supset [a_1, b_1] \supset \ldots \supset [a_n, b_n] \supset \ldots,$$

  where each interval result from a division-by-four of the previous interval.
- each such sequence of intervals can be uniquely mapped to a corresponding sequence of 2D intervals (subsquares)
- the 2D sequence of intervals converges to a unique point q in $q \in \mathcal{Q} := [0, 1] \times [0, 1]$ – q is defined as $h(t)$.

Theorem

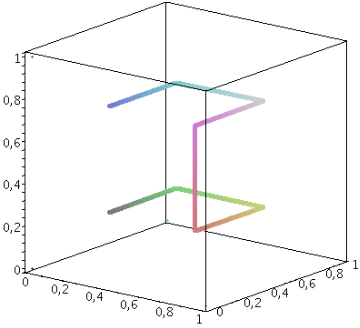
$h : \mathcal{I} \to \mathcal{Q}$ defines a space-filling curve, the Hilbert curve.

Claim: *h* defines a Space-filling Curve

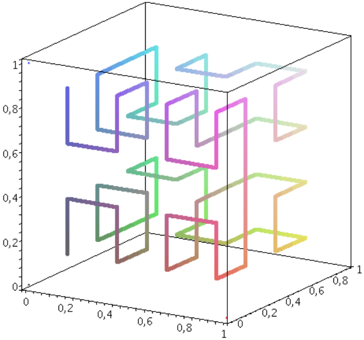
We need to prove:

- h is a mapping, i.e. each $t \in \mathcal{I}$ has a *unique* function value $h(t) \rightarrow$ OK, if $h(t)$ is independent of the choice of the sequence of intervals (proof skipped)
- $h\colon \mathcal{I} \rightarrow \mathcal{Q}$ is *surjective*:
  - for each point $q \in \mathcal{Q}$, we can construct an appropriate sequence of 2D-intervals
  - the 2D sequence corresponds in a unique way to a sequence of intervals in $\mathcal{I}$ – this sequence defines an original value of q
    $\Rightarrow$ every $q \in \mathcal{Q}$ occurs as an image point.
- h is *continuous* $\rightarrow$ see proof of *Hölder continuity*

3D Hilbert Curves – Iterations



1st iteration



2nd iteration

Part III

Parallelisation Using Space-Filling Curves

Generic Space-filling Heuristic

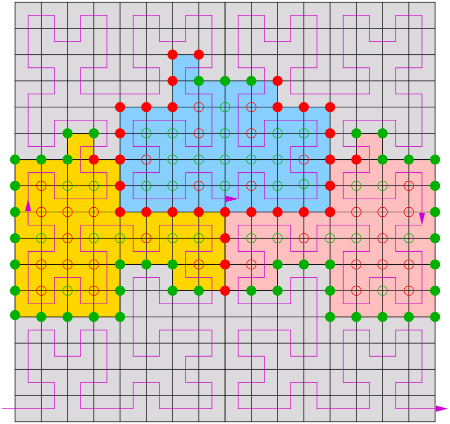
Bartholdi & Platzman (1988):

1. Transform the problem in the unit square, via a space-filling curve, to a problem on the unit interval
2. Solve the (easier) problem on the unit interval

For parallelisation: strategy to determine partitions

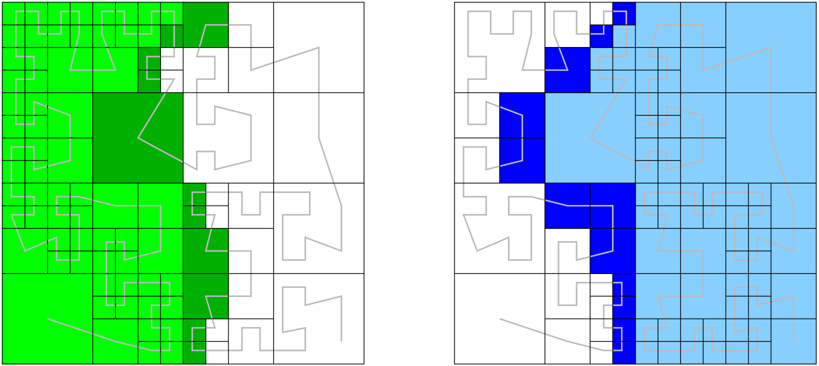
1. use a space-filling curve to generate a sequential order on the grid cells
2. do a 1D partitioning on the list of cells
   (cut into equal-sized pieces, or similar)

Hilbert-Curve Partitions on a Cartesian Grid



- Hilbert curve splits vertices into right/left (red/green) set
- Hilbert order traversal provides boundary vertices in sequential order

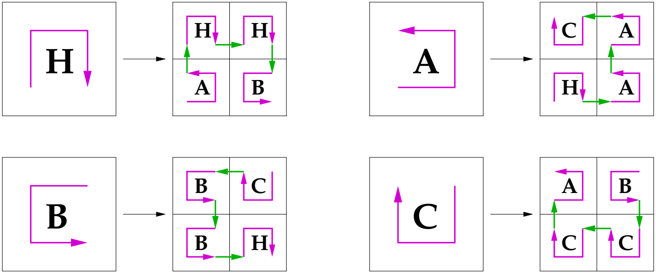
Example: Hilbert-Curve Partitions on Quadtrees



- here: with ghost cells
  (processed in identical order in both partitions)

Recall: Grammar to Describe the Hilbert Curve

Construction of the iterations of the Hilbert curve:



Can this grammar be used to generate *adaptive* Hilbert orders?

A Grammar for Hilbert Orders on Quadtrees

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

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

$\Rightarrow$ arrows describe the iterations of the Hilbert curve in “turtle graphics”

$\Rightarrow$ terminals (and) mark change of levels: “up” and “down”

Hölder Continuity

A function $f : \mathcal{I} \to \mathbb{R}^n$ is (uniformly) *continuous*, if

for each $\epsilon > 0$ there is a $\delta > 0$, such that:

for all $t_1, t_2 \in \mathcal{I}$ with $|t_1 - t_2| < \delta$,

the image points have a distance of $\|f(t_1) - f(t_2)\|_2 < \epsilon$

Hölder Continuity:

f is called *Hölder continuous with exponent* r on $\mathcal{I}$,
if a constant $C > 0$ exists, such that for all $t_1, t_2 \in I$:

$$\|f(t_1) - f(t_2)\|_2 \leq C\,|t_1 - t_2|^r$$

- case $r = 1$ is equivalent to Lipschitz continuity
- Hölder continuity implies uniform continuity

Hölder Continuity and Parallelisation

$$\|f(t_1) - f(t_2)\|_2 \leq C\,|t_1 - t_2|^r$$

Interpretation:

- $\|f(t_1) - f(t_2)\|_2$ is the distance of the image points
- $|t_1 - t_2|$ is the distance of the indices
- also: $|t_1 - t_2|$ is the area of the respective space-filling-curve partition
- hence: relation between volume (number of grid cells/points) and extent (e.g. radius) of a partition

$\Rightarrow$ Hölder continuity gives a quantitative estimate for **compactness of partitions**

Hölder Continuity of the Hilbert Curve

Proof:

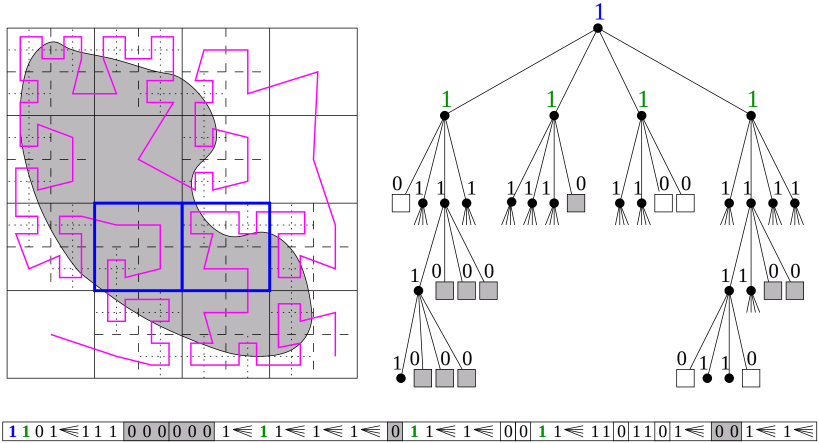
- given $t_1, t_2 \in \mathcal{I}$; choose n, such that $4^{-(n+1)} < |t_1 - t_2| < 4^{-n}$
- 4^{-n} is interval length for the n-th iteration $\Rightarrow [t_1, t_2]$ overlaps at most two neighbouring(!) intervals.
- due to construction of the Hilbert curve, $h(t_1)$ and $h(t_2)$ are in neighbouring subsquares with face length 2^{-n}.
- these two subsquares build a rectangle with a diagonal of length $2^{-n} \cdot \sqrt{5}$; therefore: $\|h(t_1) - h(t_2)\|_2 \leq 2^{-n}\sqrt{5}$
- as $4^{-(n+1)} < |t_1 - t_2|$, we have $2 \cdot 2^{-n} < \sqrt{|t_1 - t_2|}$

$\Rightarrow$ result: $\|h(t_1) - h(t_2)\|_2 \leq \frac{1}{2}\sqrt{5}\,|t_1 - t_2|^{1/2}$

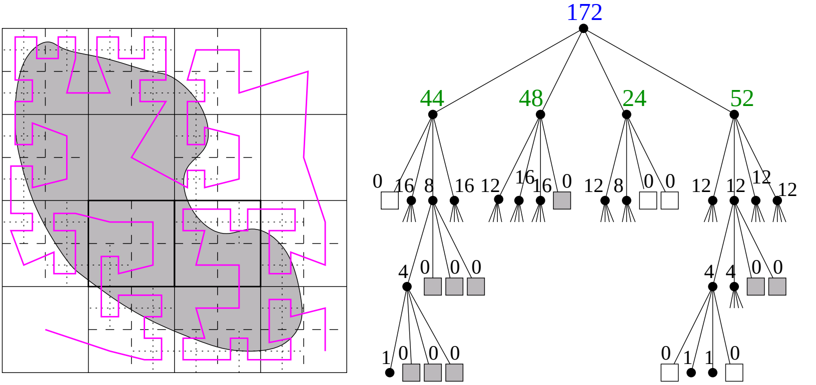
Part IV

Outlook: Parallelisation with Space-Filling Curves and Refinement Trees

Hilbert-Order Bitstream-Encoding of a Quadtree



Refinement-Tree Encoding of a Quadtree



172	44	0	16 ⋘	8	4	1	0 0 0	0 0 0	16 ⋘	48	12 ⋘	16 ⋘	16 ⋘	0	24	12 ⋘	8 ⋘	0	0	52	12 ⋘	12 ⋘	12 ⋘	12 ⋘

Refinement-Tree Encoding of a Quadtree (2)

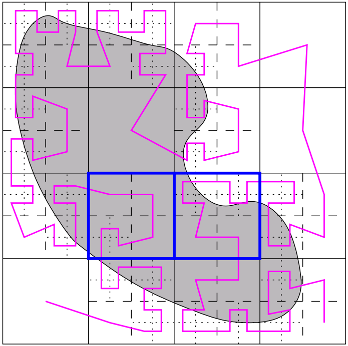
REFTREE algorithm for partitioning:

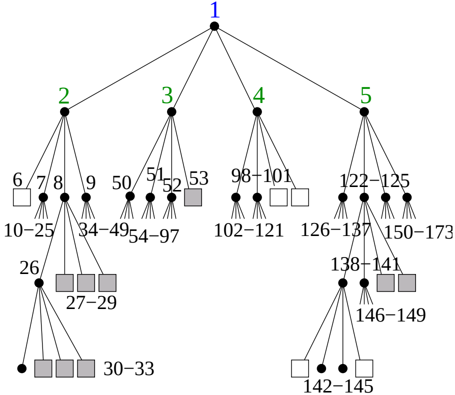
- attributed quadtree with number of leaves/nodes of the subtree for each node
- allows to determine whether a certain node/subtree may be skipped by the current partition (if index of first & last leave/node are given)
- disadvantage of data structure: required information spread across several locations in the stream
  $\Rightarrow$ may be fixed by modified depth-first order

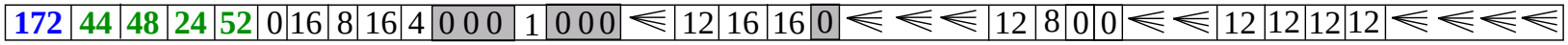
cmp. algorithm in Maple worksheet

`reftree_hilbert_vertexlab.mw`

Refinement-Tree Encoding with Modified Depth-First Order





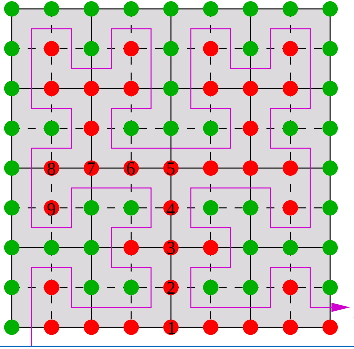


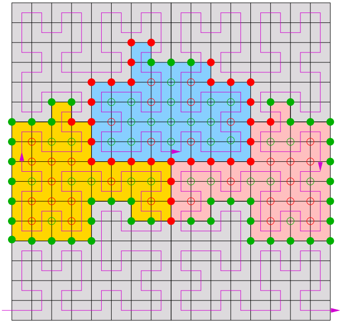
(numbers in the tree represent position of resp. node information in the stream)

Parallelisation vs. Partitioning with SFC

Besides partitioning, parallelisation has to deal with:

- data exchange between partitions:
  $\rightarrow$ unknowns on separator between partitions
  $\rightarrow$ synchronize refinement status at partition boundaries
- may exploit "stack property" of Hilbert curves





How to Determine Left and Right

Turtle Grammars Revisited