A tetrahedral space-filling curve via bitwise interleaving

Presentation · April 2016

DOI: 10.13140/RG.2.2.22831.87203

2 authors:

Carsten Burstedde
University of Bonn
52 PUBLICATIONS **4,665** CITATIONS
SEE PROFILE

Johannes Holke
German Aerospace Center (DLR)
35 PUBLICATIONS **75** CITATIONS
SEE PROFILE

A tetrahedral space-filling curve via bitwise interleaving

Carsten Burstedde, Johannes Holke

April 13, 2016

Institute for Numerical Simulation
Rheinische Friedrich-Wilhelms-Universität Bonn

hausdorff center for mathematics

BIGS
BONN INTERNATIONAL
GRADUATE SCHOOL
OF MATHEMATICS

Table of contents

Introduction

Space-filling curve partitioning

Construct a curve that passes each mesh element only once. Cut the curve in equal pieces.

- Runs in linear time.
- Works on arbitrary meshes.
- Can create non-connected domains.

Introduction

Space-filling curve partitioning

Construct a curve that passes each mesh element only once. Cut the curve in equal pieces.

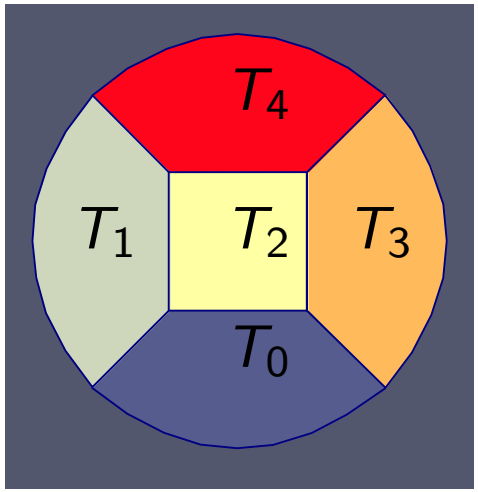
- Runs in linear time.
- Works on arbitrary meshes.
- Can create non-connected domains.

SFC partitioning has proven to give good results in an optimal runtime. Well suited for applications where refinement/coarsening is done frequently.

How to construct SFC

A mesh actually consists of two layers:[1]

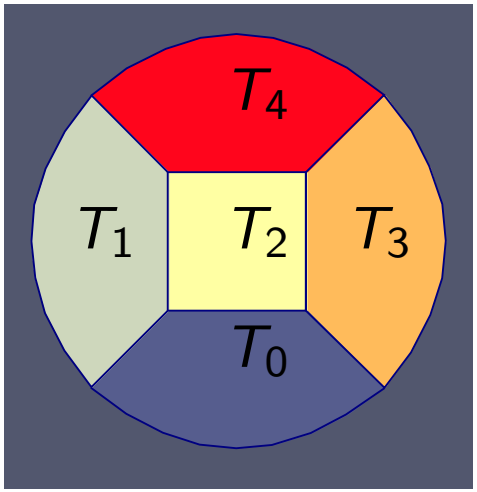
- The coarse mesh.
  - Stores domain geometry.
  - Constructed either by the user or a mesh generator.
  - Cannot be coarsened.
  - Pieces are called trees and are already ordered.

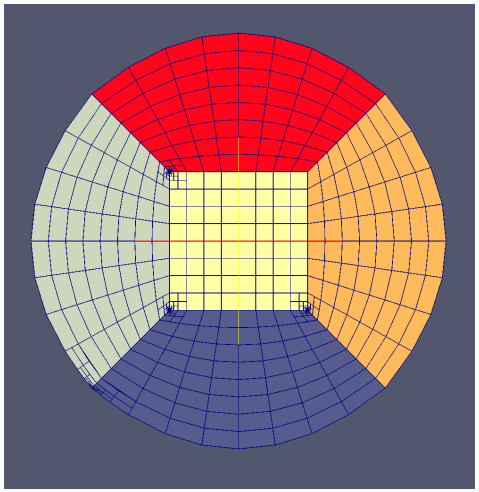


How to construct SFC

A mesh actually consists of two layers:[1]

- The coarse mesh.
  - Stores domain geometry.
  - Constructed either by the user or a mesh generator.
  - Cannot be coarsened.
  - Pieces are called trees and are already ordered.
- The fine mesh.
  - Consists of the actual mesh elements.
  - Is used in the numerical computation.





How to construct SFC

Since each element belongs to a tree and the trees are already ordered, we only need to order all elements within each tree.

How to construct SFC

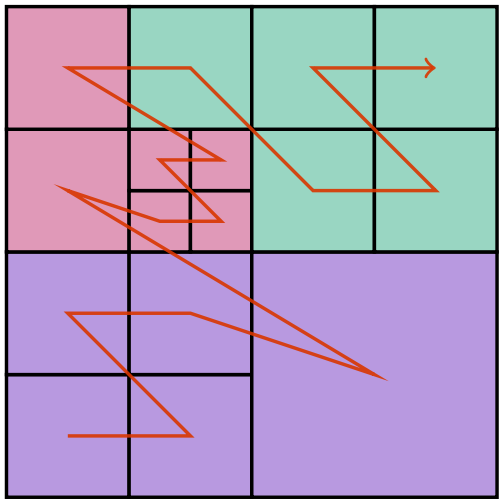
Since each element belongs to a tree and the trees are already ordered, we only need to order all elements within each tree.

$\Rightarrow$ Define SFC only on a single reference tree.

The Morton code for cubes

Reminder

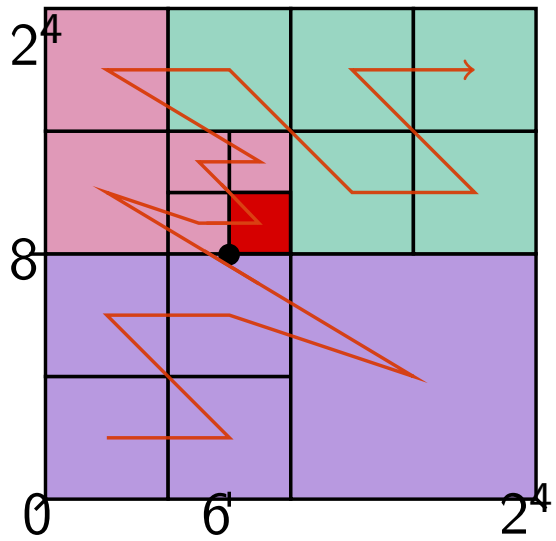
The Z-curve for quadrilateral/hexahedral meshes is constructed via the Morton code (Morton 1966).



The Morton code for cubes

Reminder

The Z-curve for quadrilateral/hexahedral meshes is constructed via the Morton code (Morton 1966).



Morton code of Q:

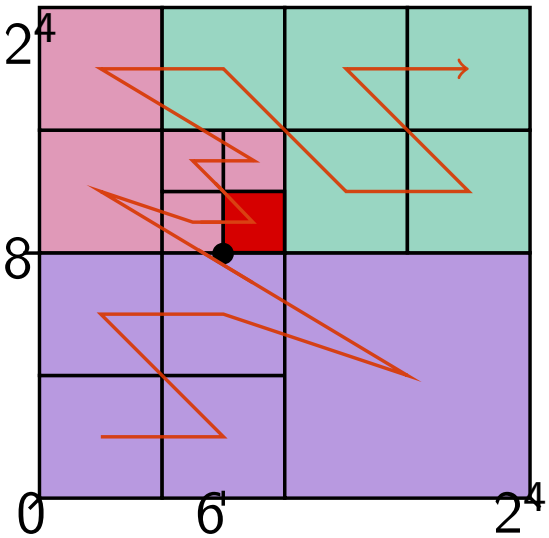
$$\begin{aligned} x(Q) &= 6 = (0110)_2 \\ y(Q) &= 8 = (1000)_2 \\ \ell(Q) &= 3 \end{aligned} \tag{1}$$

Maximum refinement level $\mathcal{L} = 4$.

The Morton code for cubes

Reminder

The Z-curve for quadrilateral/hexahedral meshes is constructed via the Morton code (Morton 1966).



Maximum refinement level $\mathcal{L} = 4$.

Morton code of Q:

$$\begin{aligned} x(Q) &= 6 = (0110)_2 \\ y(Q) &= 8 = (1000)_2 \\ \ell(Q) &= 3 \end{aligned} \tag{1}$$

$$\Rightarrow m(Q) = (10010100)_2$$

The Morton code for cubes

Cell identifier

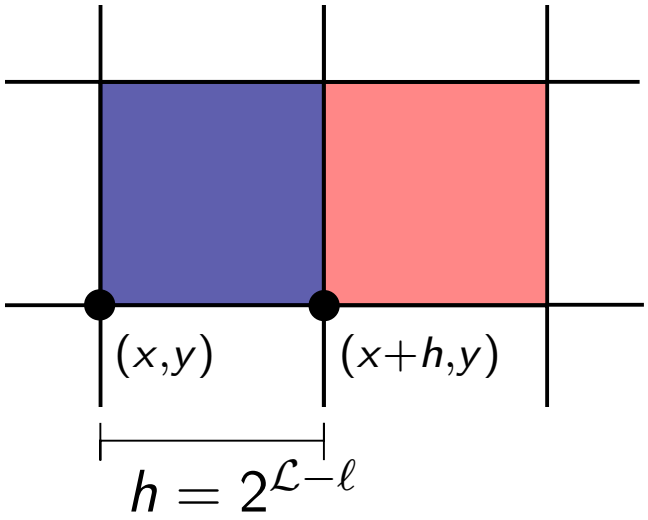
$m(Q)$ and $\ell(Q)$ define the SFC and are a unique identifier of Q.

The Morton code for cubes

Using the Morton index, many low-level algorithms can be computed efficiently. Two examples:

1) Computing face-neighbors

Add or subtract $h = 2^{\mathcal{L}-\ell}$ from the appropriate coordinate.



The Morton code for cubes

Using the Morton index, many low-level algorithms can be computed efficiently. Two examples:

2) Computing parents

Set the ℓth bits of each coordinate to 0 and change the level to $\ell - 1$.

$$\begin{aligned} x_P &= x \,\&\, \sim h \\ y_P &= y \,\&\, \sim h \\ z_P &= z \,\&\, \sim h \\ \ell_P &= \ell - 1 \end{aligned}$$

The Morton code for cubes

Advantageous features of the Morton code

- Computable in a fast manner via bitwise interleaving.
- Easy to implement.
- Memory efficient
  - storage per element: coordinates of one node, level ($4d + 1$ Bytes).
- Children, parent, face-neighbor, etc.
  computed in constant time.

Question

Can we transfer these properties to simplical AMR?

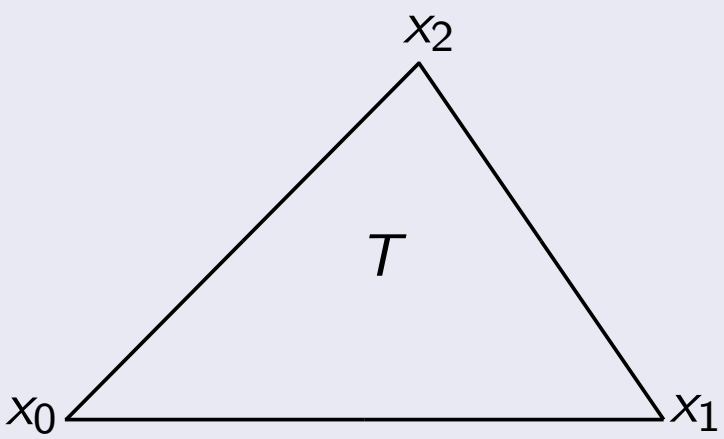
Question

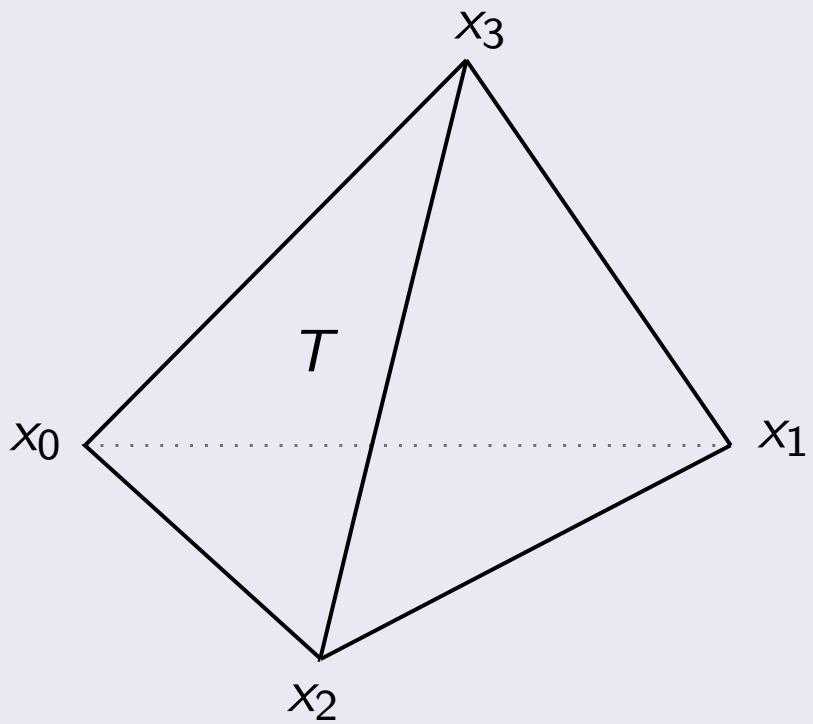
Can we transfer these properties to simplical AMR?

Yes, mostly. We will demonstrate the construction of a Morton code for non-conforming simplicial mesh refinement.

The refinement rule

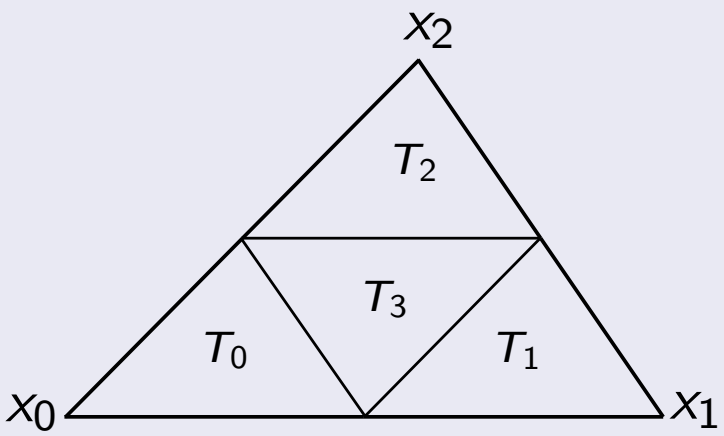
We use the red refinement rules described by Bey [**?**].

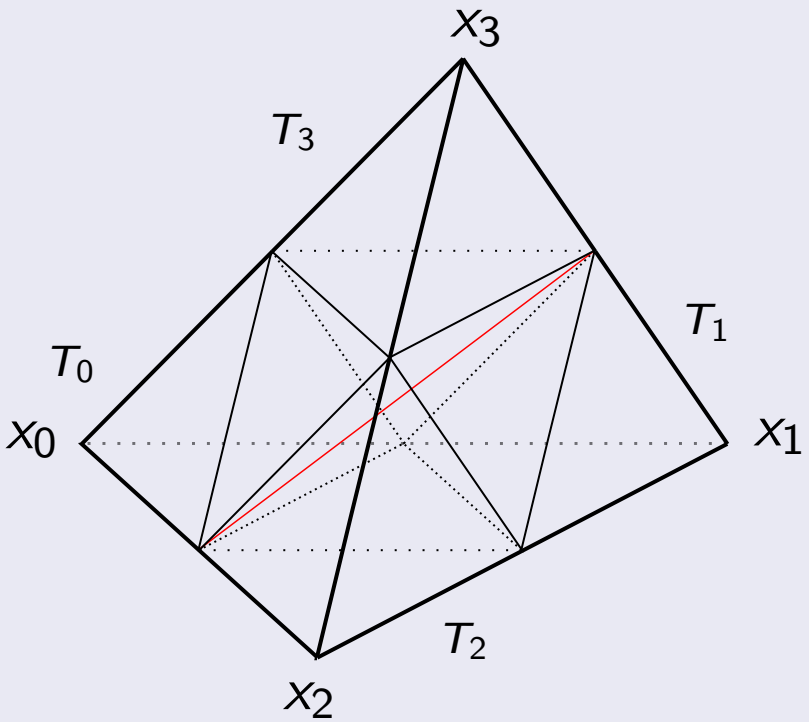




The refinement rule

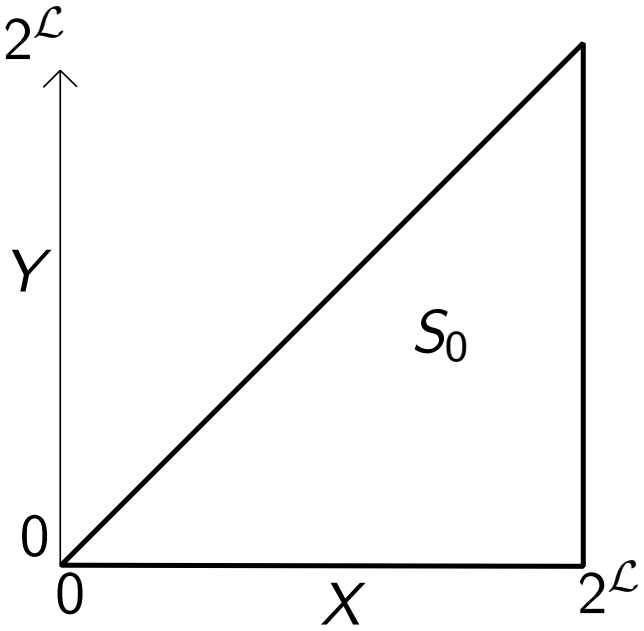
We use the red refinement rules described by Bey [**?**].





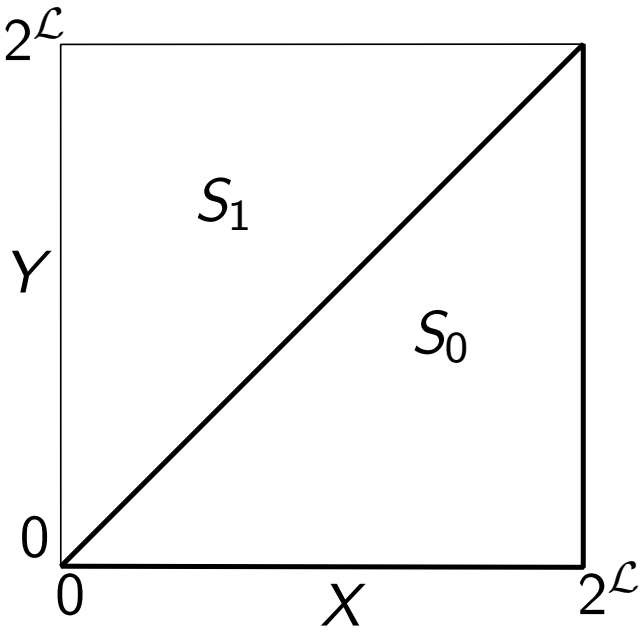
- Refinement and coarsening become near-trivial operations.
- Hanging nodes will occur.

The reference tree and the type (2d)



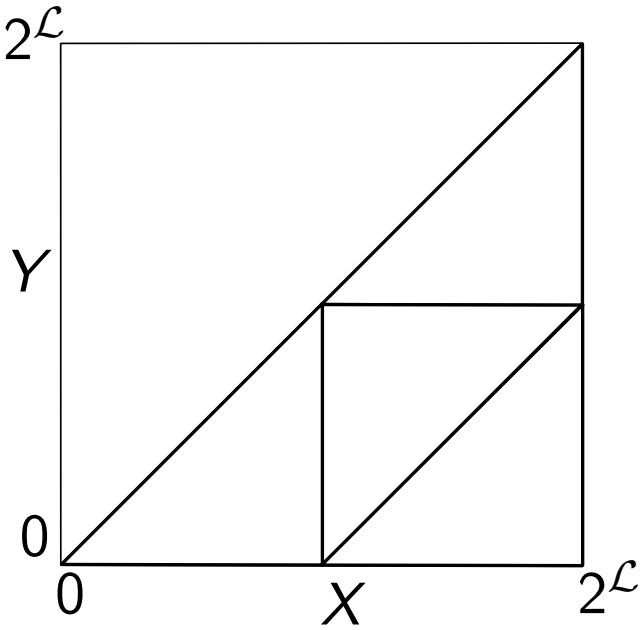
The right triangle S_0 serves as our reference tree. $\mathcal{L}$ is a given maximum refinement level.

The reference tree and the type (2d)



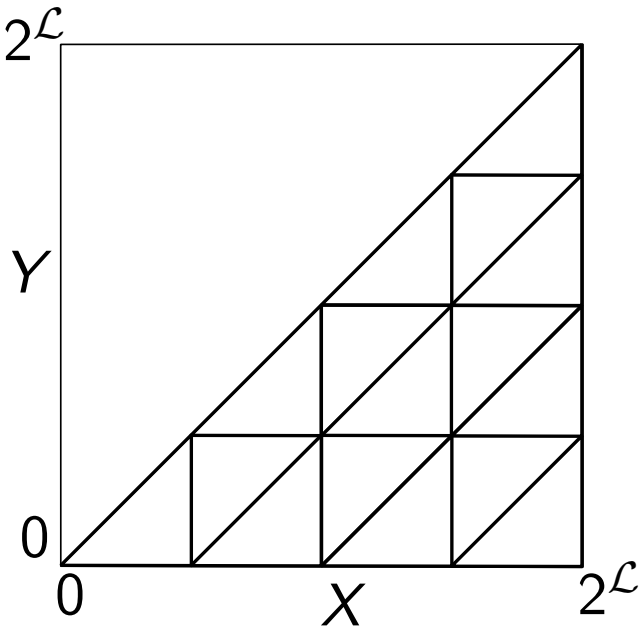
We embed this triangle in the square $[0, 2^{\mathcal{L}}]^2$, which is divided along its diagonal; obtaining a second triangle S_1.

The reference tree and the type (2d)



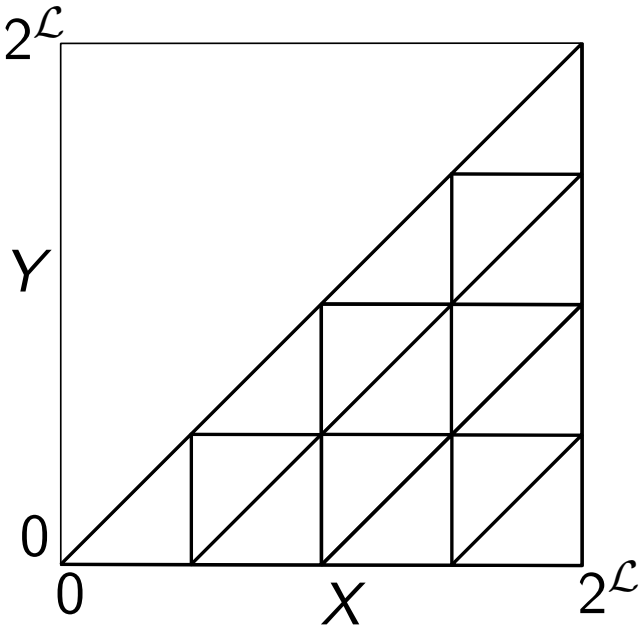
Bey's observation:
When refining S_0, each occurring triangle T is equivalent to S_0 or S_1.

The reference tree and the type (2d)



Bey's observation:
When refining S_0, each occurring triangle T is equivalent to S_0 or S_1.

The reference tree and the type (2d)

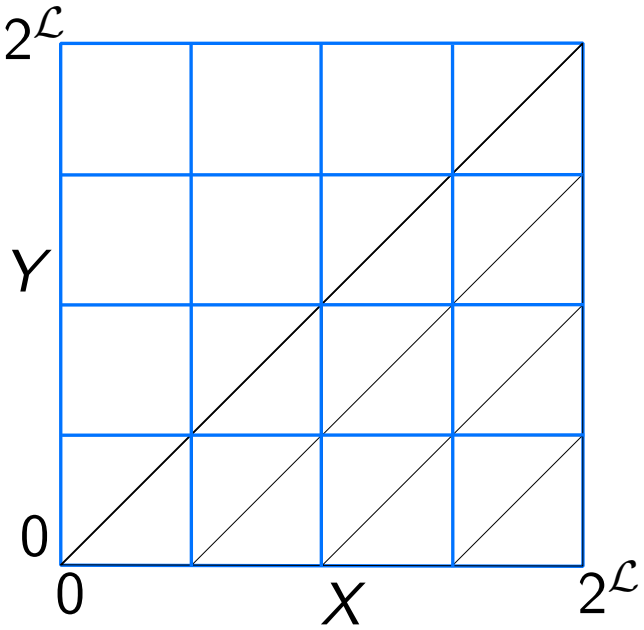


Bey's observation:
When refining S_0, each occurring triangle T is equivalent to S_0 or S_1.

Definition

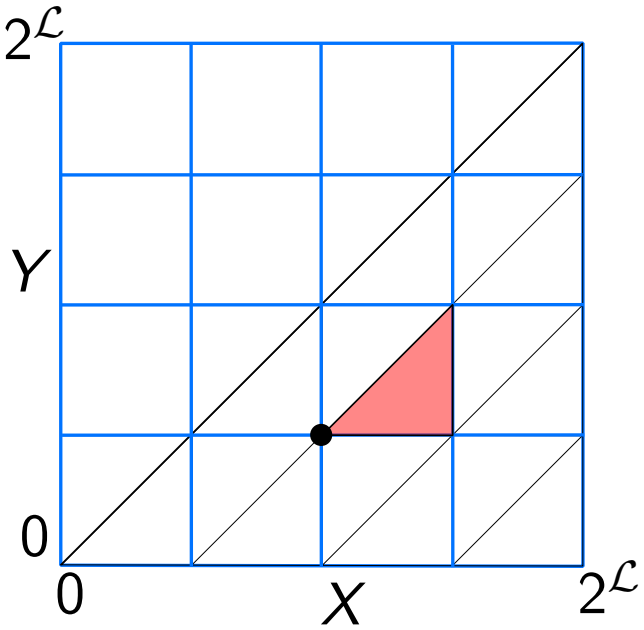
$\text{Type}(T) = i \;:\Leftrightarrow\; T \simeq S_i$.

The reference tree and the type (2d)



Furthermore, there is also an underlying quadrilateral mesh.

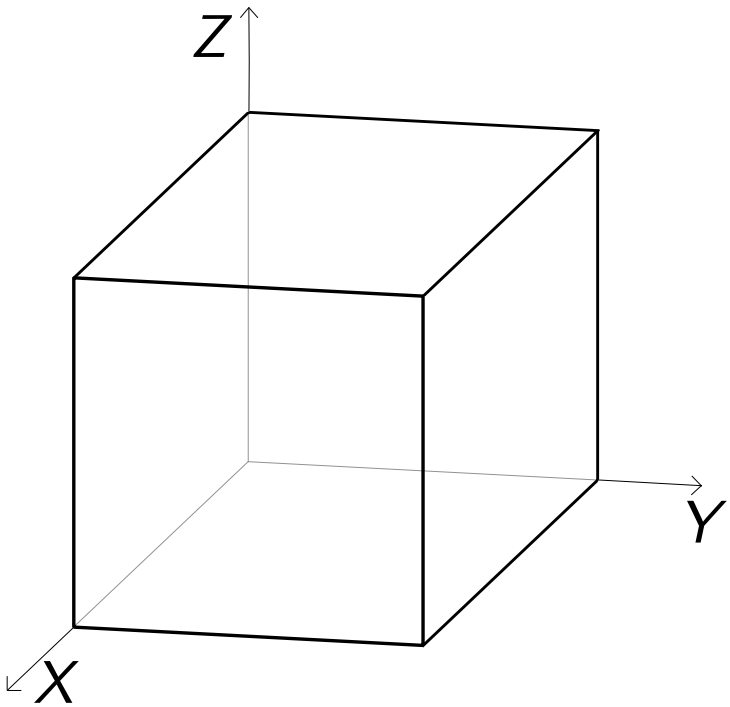
The reference tree and the type (2d)



A triangle T in a refinement is uniquely identified by the coordinates of one node plus its level plus its type.

The reference element and the type (3d)

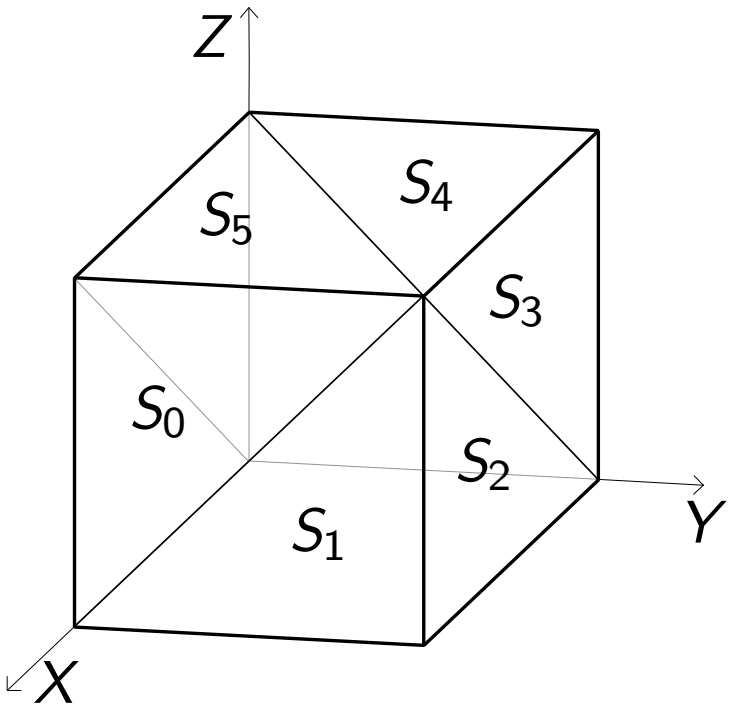
The same constructions applies in 3d.
Here we have six different types.



Each tetrahedron T in a refinement of S_0 is equivalent to one of $S_0, \ldots, S_5$.

The reference element and the type (3d)

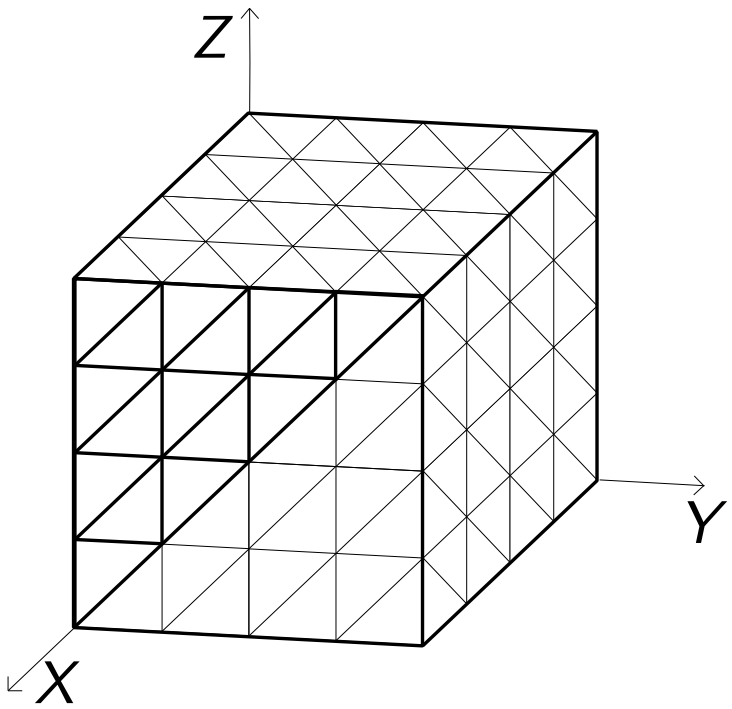
The same constructions applies in 3d.
Here we have six different types.



Each tetrahedron T in a refinement of S_0 is equivalent to one of $S_0, \ldots, S_5$.

The reference element and the type (3d)

The same constructions applies in 3d.
Here we have six different types.



Each tetrahedron T in a refinement of S_0 is equivalent to one of $S_0, \ldots, S_5$.
It is uniquely identified by the coordinates of one node plus its level plus its type.

The tetrahedral Morton code

Cell identifier

The tuple

$$(x(T), y(T), z(T), \text{type}(T), \ell(T))$$

uniquely identifies a simplex in a refinement of S_0.

The tetrahedral Morton code

Cell identifier

The tuple

$$(x(T), y(T), z(T), \text{type}(T), \ell(T))$$

uniquely identifies a simplex in a refinement of S_0.

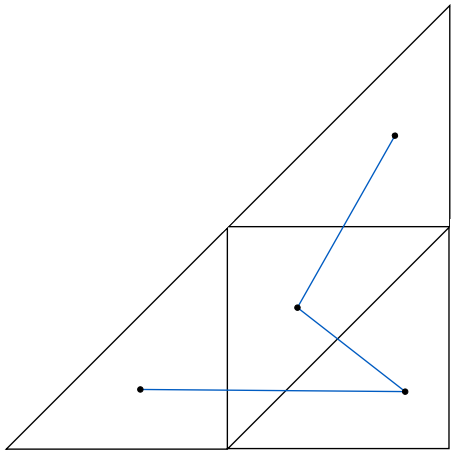
We only need to store this data per element.
Thus, with $\mathcal{L} = 32$:

$$\begin{aligned} 2 \times 4 + 1 + 1 &= 10 \quad \text{Bytes per Triangle,} \\ 3 \times 4 + 1 + 1 &= 14 \quad \text{Bytes per Tetrahedron.} \end{aligned}$$

The SFC in 2 dimensions

Are we done yet?

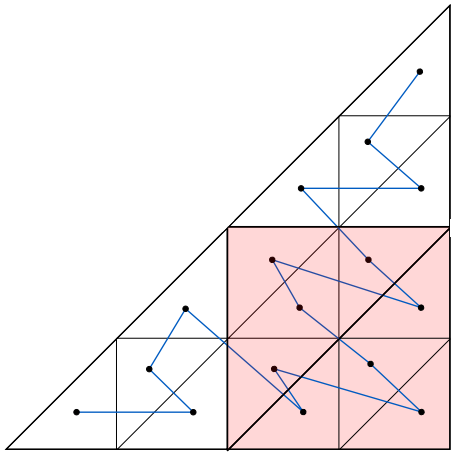
We could define the SFC index of T to be the interleaving of z, y, x coordinates and account for the type in the last step.



The SFC in 2 dimensions

Are we done yet?

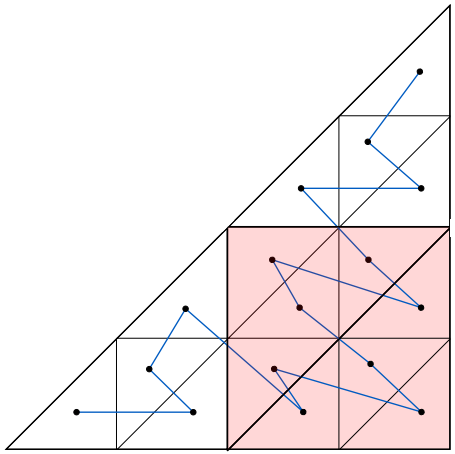
We could define the SFC index of T to be the interleaving of z, y, x coordinates and account for the type in the last step.



The SFC in 2 dimensions

Are we done yet?

We could define the SFC index of T to be the interleaving of z, y, x coordinates and account for the type in the last step.



No!

- Locality is not preserved.

The tetrahedral Morton code

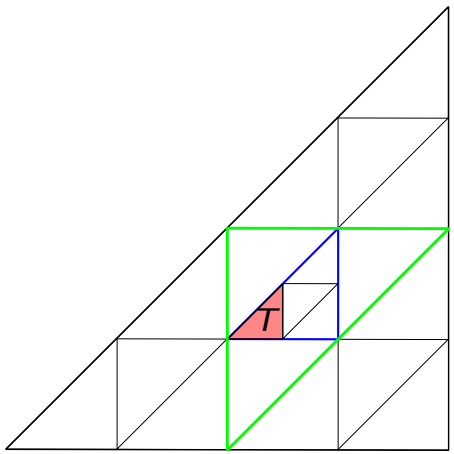
Definition

For a given simplex T in a refinement let $B = (b_{\mathcal{L}-1}, b_{\mathcal{L}-2}, \ldots, b_0)$ be the sequence of the types of $T's$ ancestors of levels $1, 2, \ldots, \mathcal{L}$.

The tetrahedral Morton code

Definition

For a given simplex T in a refinement let $B = (b_{\mathcal{L}-1}, b_{\mathcal{L}-2}, \ldots, b_0)$ be the sequence of the types of $T's$ ancestors of levels $1, 2, \ldots, \mathcal{L}$.

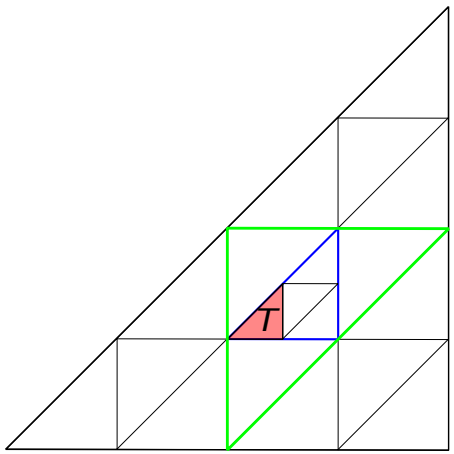


$$B = (1, 0, 0, 0, \ldots, 0) \qquad (2)$$

The tetrahedral Morton code

Definition

For a given simplex T in a refinement let $B = (b_{\mathcal{L}-1}, b_{\mathcal{L}-2}, \ldots, b_0)$ be the sequence of the types of $T's$ ancestors of levels $1, 2, \ldots, \mathcal{L}$.



$$B = (1, 0, 0, 0, \ldots, 0) \qquad (2)$$

We do not store B, but compute it in $\mathcal{O}(\text{level})$ when necessary.

The tetrahedral Morton code

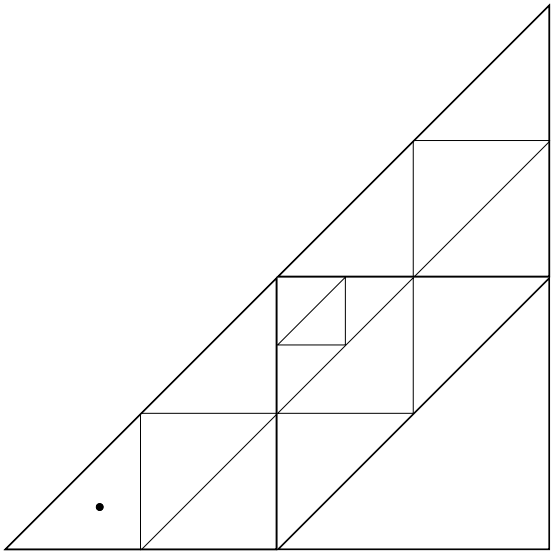
Definition

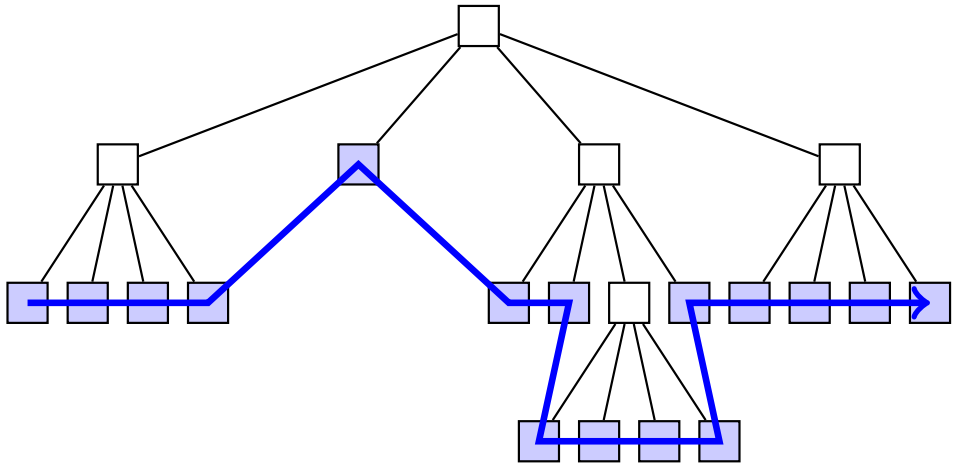
The tetrahedral Morton index of a simplex T is contructed by interleaving $(z,)y, x$ coordinates of the anchor node of T with B.

$(y_{\mathcal{L}-1}, \ldots, y_0)_2 \qquad (x_{\mathcal{L}-1}, \ldots, x_0)_2 \qquad (b_{\mathcal{L}-1}, \ldots, b_0)_4$

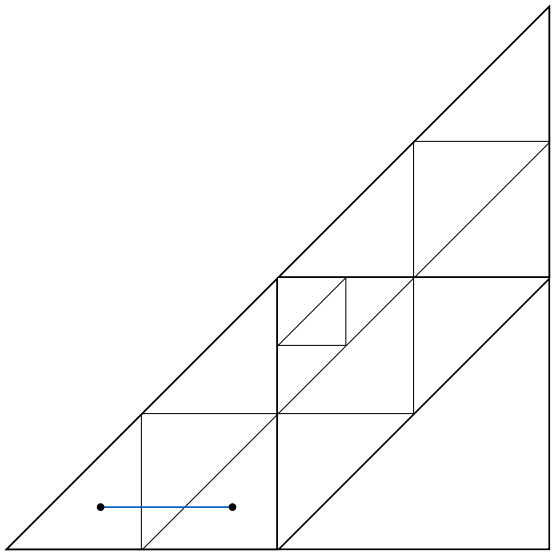
$$m(T) = (y_{\mathcal{L}-1}x_{\mathcal{L}-1}, b_{\mathcal{L}-1}, \ldots, y_0 x_0, b_0)_4$$

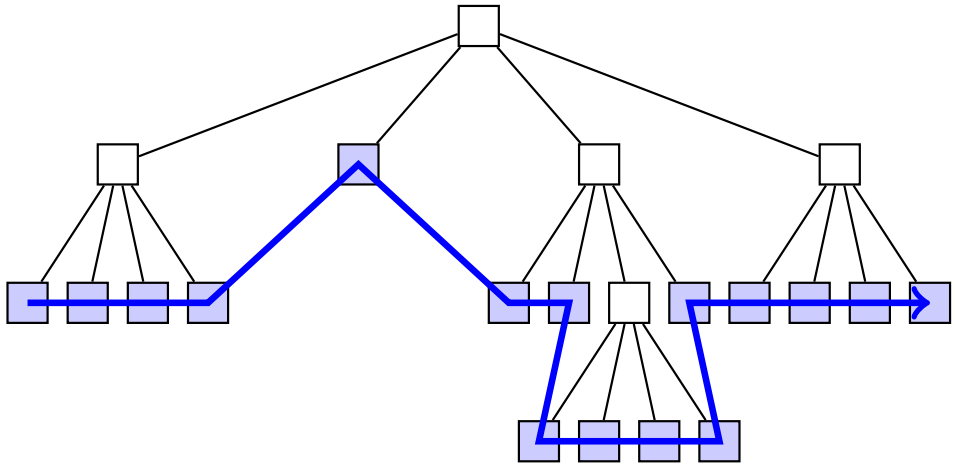
The SFC in 2 dimensions



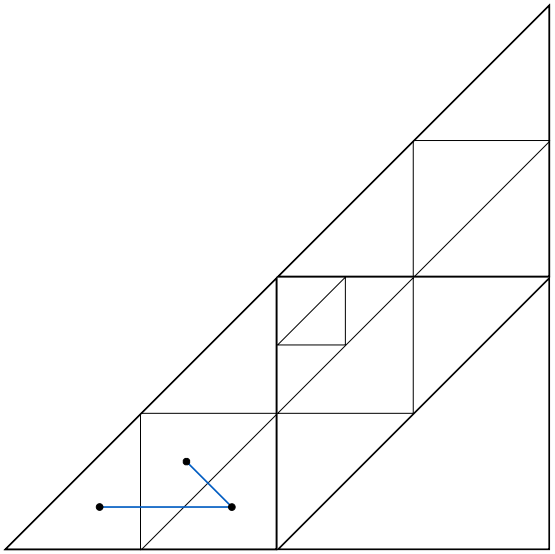


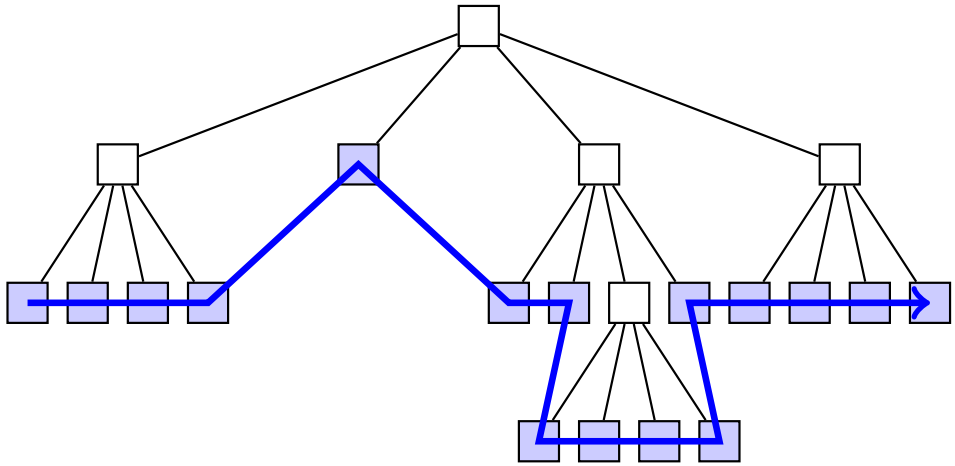
The SFC in 2 dimensions



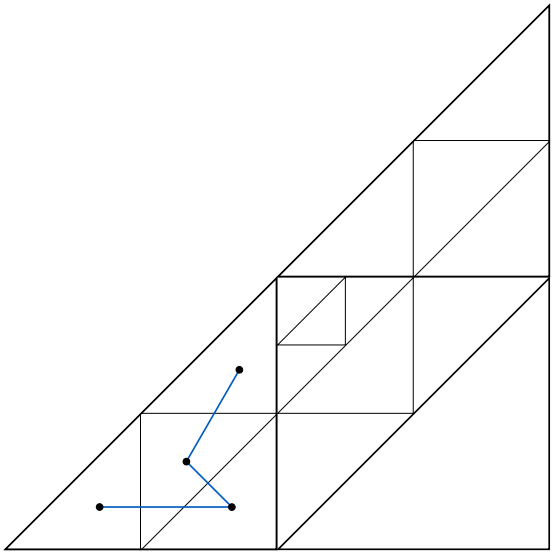


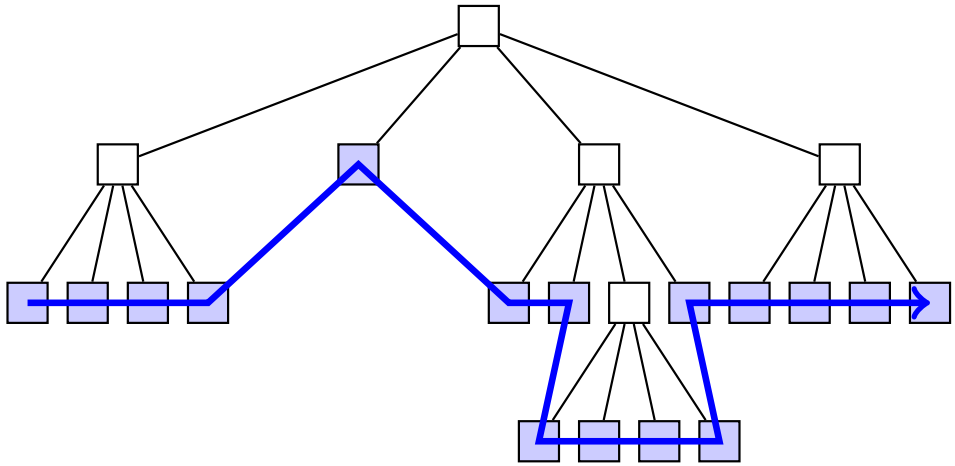
The SFC in 2 dimensions



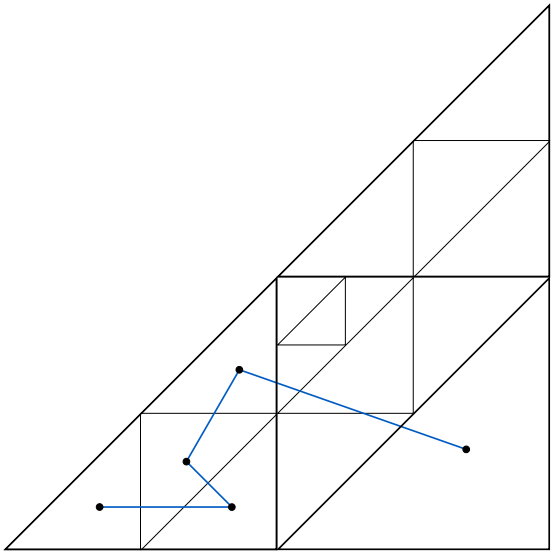


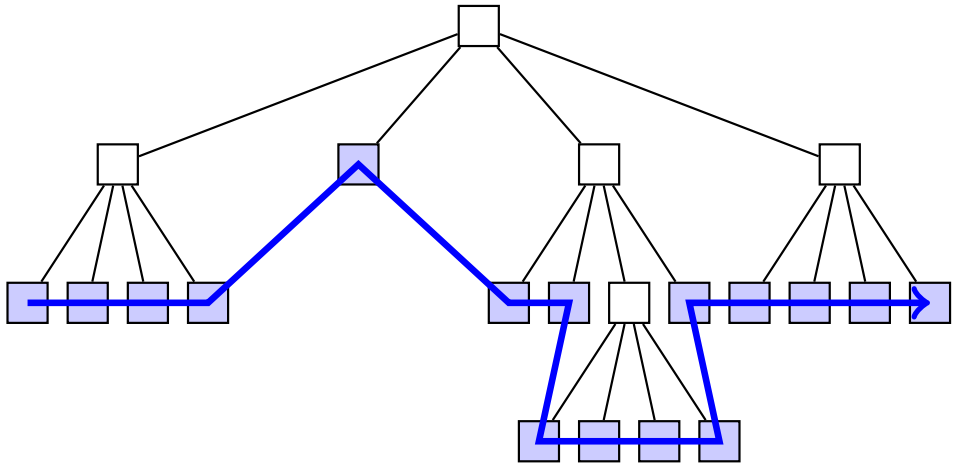
The SFC in 2 dimensions



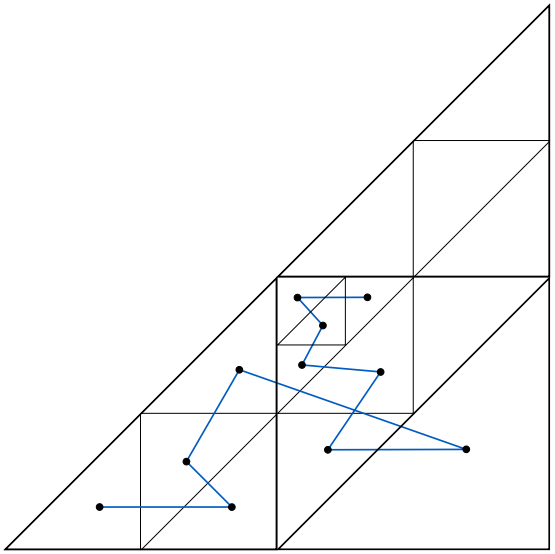


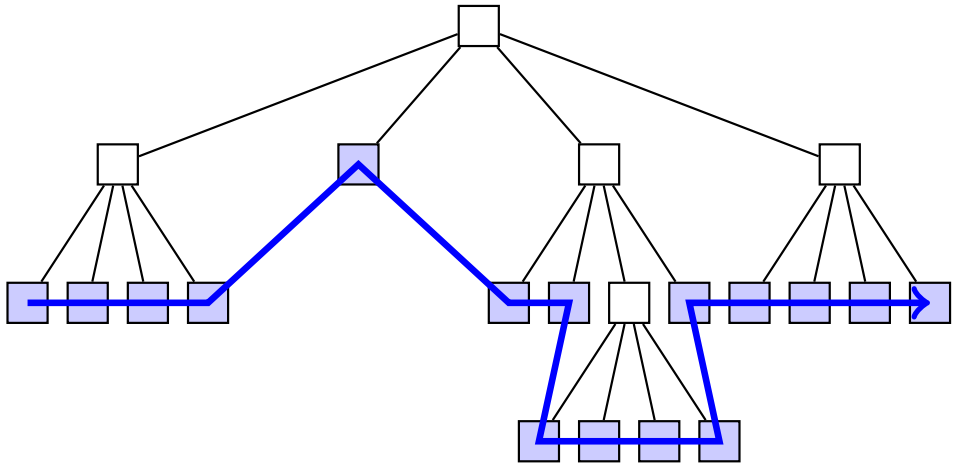
The SFC in 2 dimensions



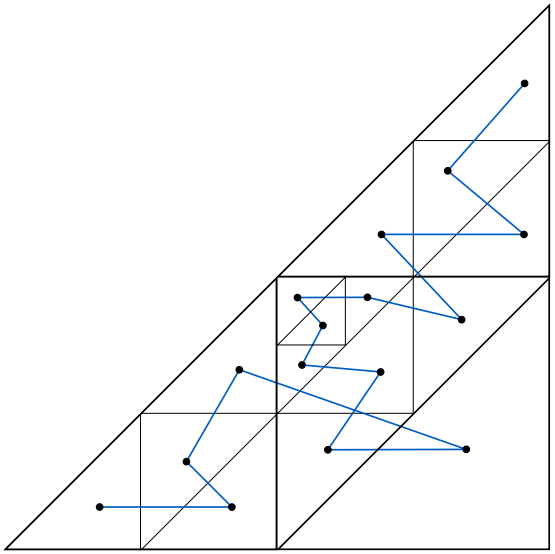


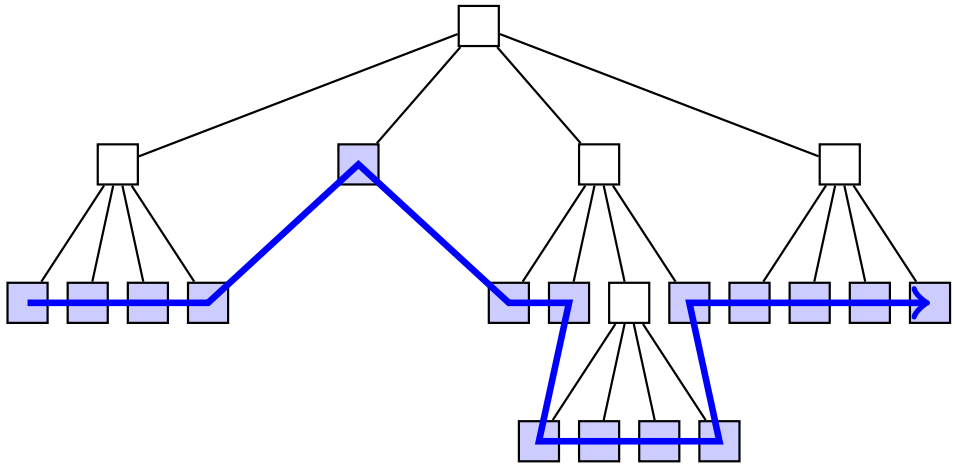
The SFC in 2 dimensions





The SFC in 2 dimensions



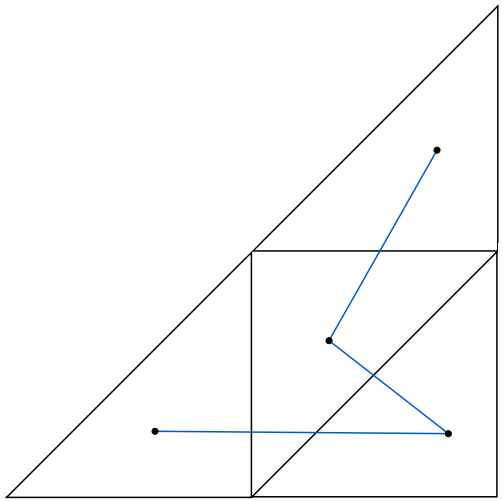


The SFC in 2 dimensions

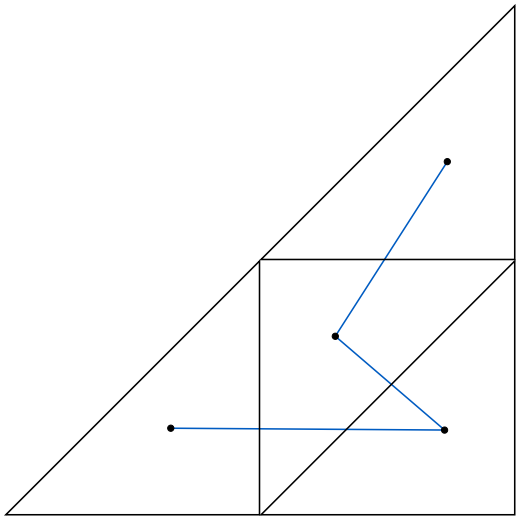
How is this different from just taking the Morton curve over the underlying quadgrid?

The SFC in 2 dimensions

How is this different from just taking the Morton curve over the underlying quadgrid?



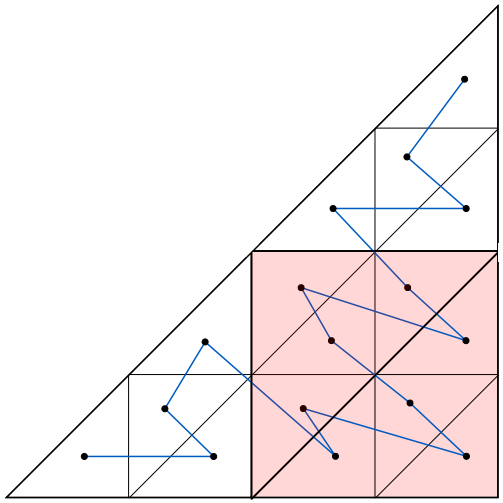
The Morton curve over quadgrid and types.



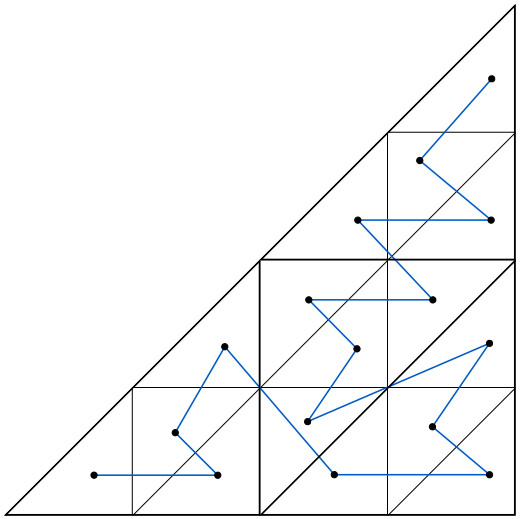
Our new Morton curve.

The SFC in 2 dimensions

How is this different from just taking the Morton curve over the underlying quadgrid?



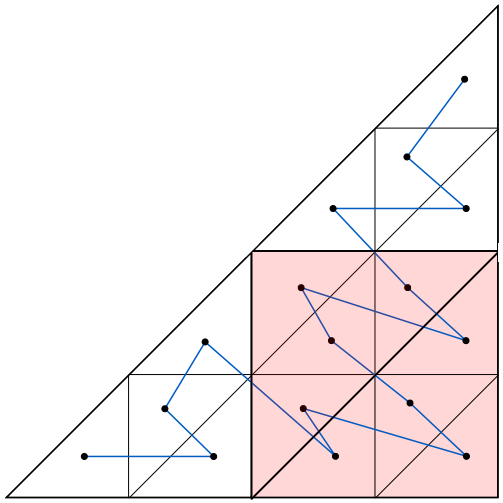
The Morton curve over quadgrid and types.



Our new Morton curve.

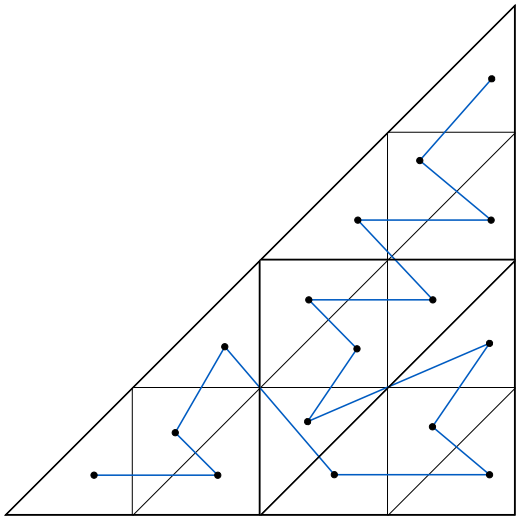
The SFC in 2 dimensions

How is this different from just taking the Morton curve over the underlying quadgrid?



The Morton curve over quadgrid and types.

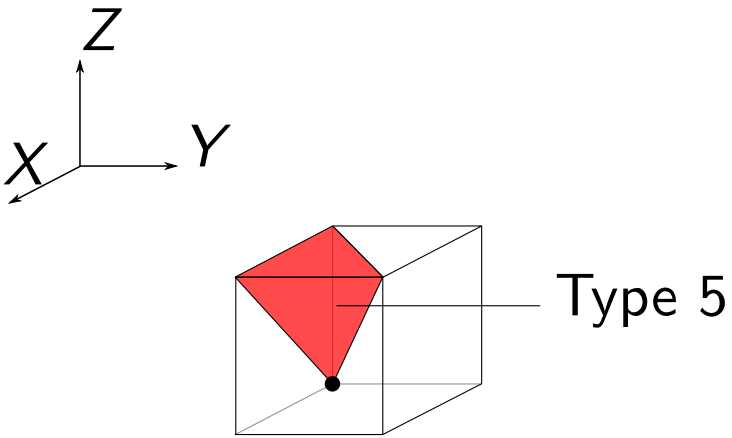
- Locality not preserved.



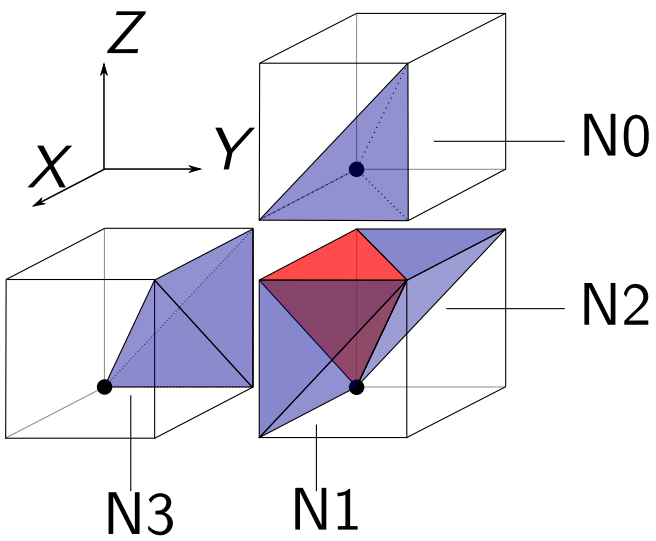
Our new Morton curve.

- Locality preserved.

Computing face-neighbors, an example



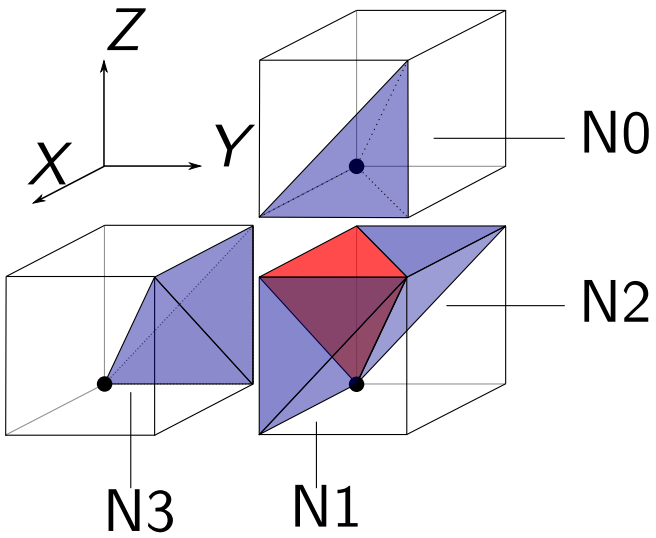
Computing face-neighbors, an example



Computation of N0

Add $2^{\mathcal{L}-\ell}$ to z-coordinate, change type to 1.

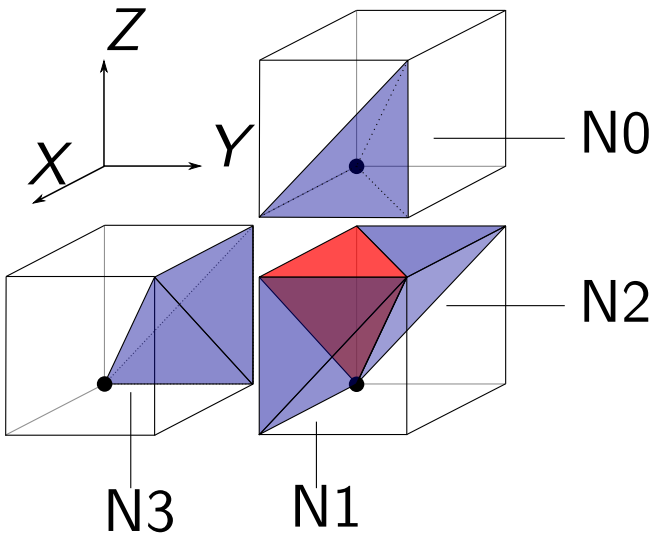
Computing face-neighbors, an example



Computation of N1

Change type to 0.

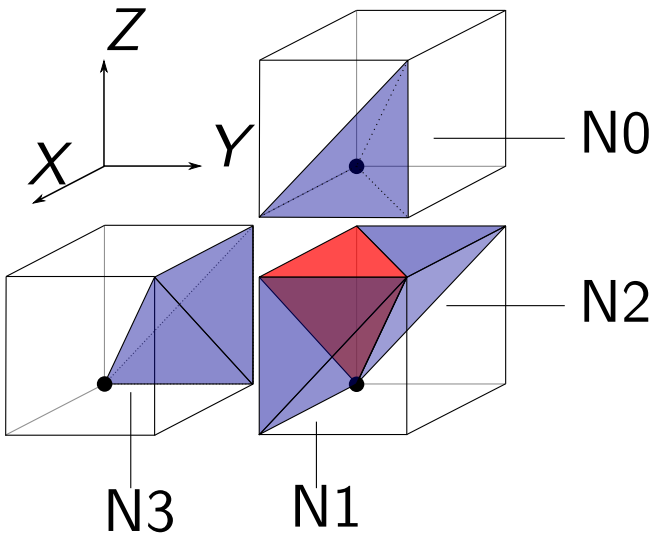
Computing face-neighbors, an example



Computation of N2

Change type to 4.

Computing face-neighbors, an example



Computation of N3

Substract $2^{\mathcal{L}-\ell}$ from y-coordinate, change type to 3.

Computing face-neighbors

We do this once for each type and obtain a look-up table.
Computing face-neighbors is as easy as looking up the values in the table.

Computing face-neighbors

We do this once for each type and obtain a look-up table.
Computing face-neighbors is as easy as looking up the values in the table.

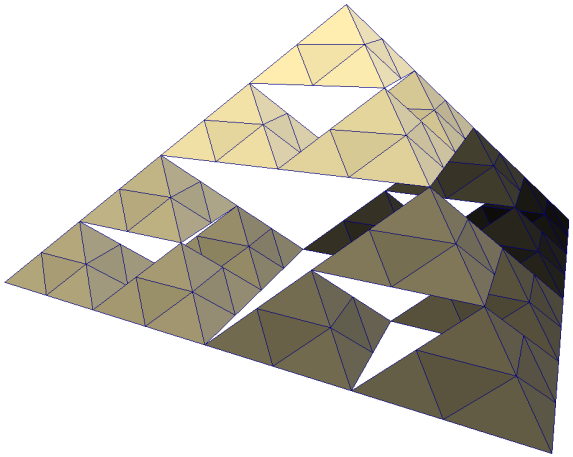
Parent, children and node coordinates can be computed similarly.

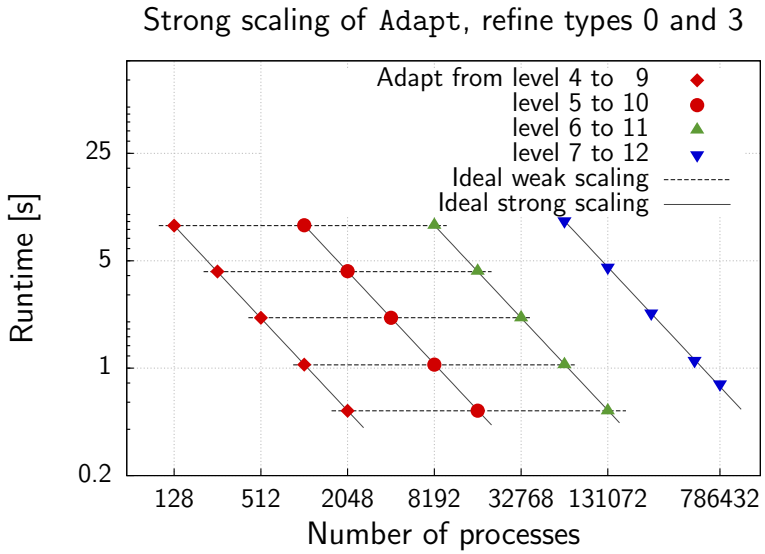
Outlook

We currently develop the AMR library `t8code`, implementing high-level AMR algorithms based on the tetrahedral Morton curve.

- `Adapt`
- `Balance`
- `Partition`
- `Ghost`
- `Iterate`

Timing results for Refinement





Timings on up to $131k$ processes on JUQUEEN (FSZ Jülich).
Up to $786k$ on MIRA (ANL).
Biggest mesh in this test:
$\sim 8.6 \times 1e11$ Tets on 65k MPI ranks $\approx$ 13M per rank.

Acknowledgments

Thank you for your attention!