Grid generation

9th SIMLAB Course

Janos Benk

October 4, 2010

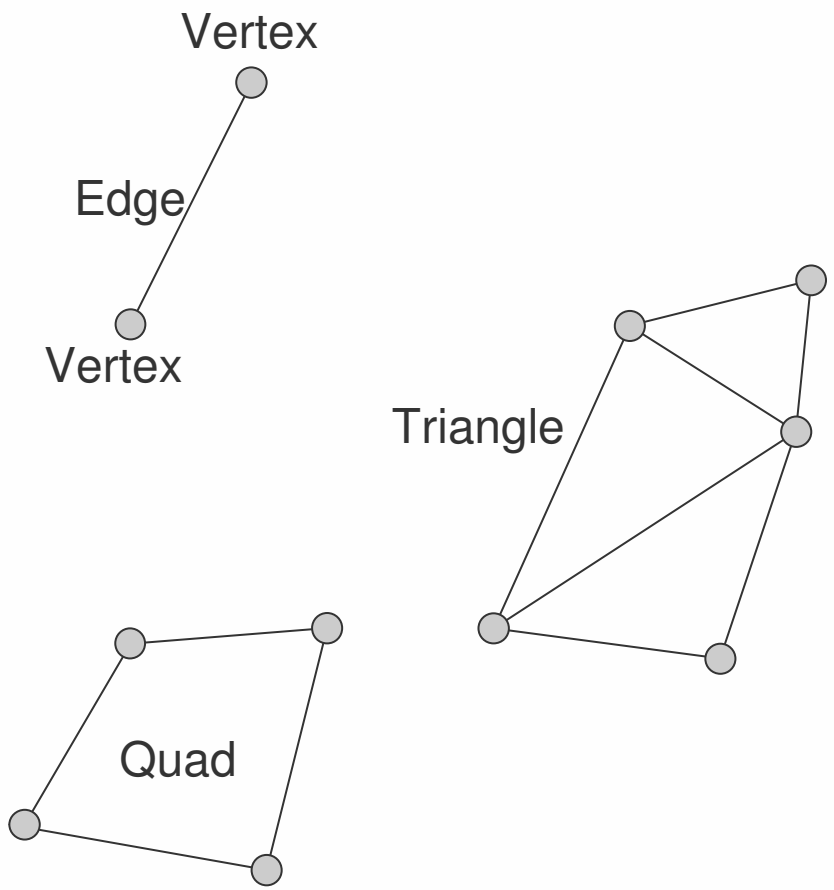
Overview

- Motivation and Introduction to grids (meshes)
- Categoriaztion of grids
- Grid generation
- (Hierachical basis and sparse grids)
- (Space filling curves and cashe efficient grids)

Grid Introduction

- **Vertex** (point) is a 0 dimensional object with a defined position in space
- The grid (mesh) is not just the set of vertexes, but also the connection between them. (Similar to graphs),
- **Edge** is 1D object.
- Topology information
- Vertexes and edges form 2 dimensional objects.

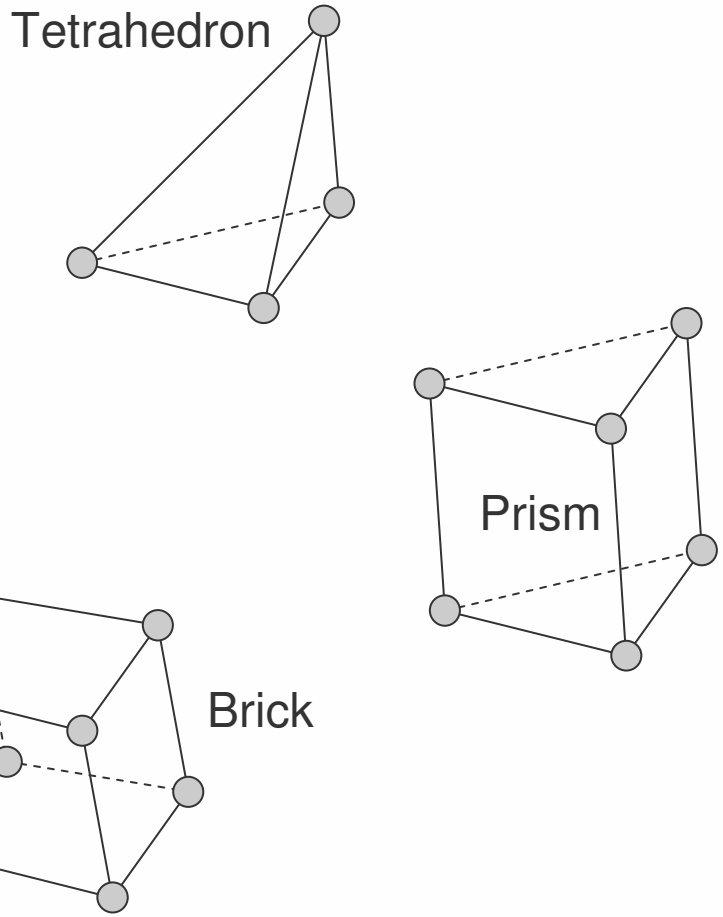
  (Quad = Rectangle)



Grid Introduction

- In 3D there are more combinations
- Triangles and Quads can form different combination of 3D objects
- A grid can allow also the combination of different elements.
- The notion of **facet**, **co-facet**, are sub-dimensional or super-dimensional components

  (Brick = Cuboid)

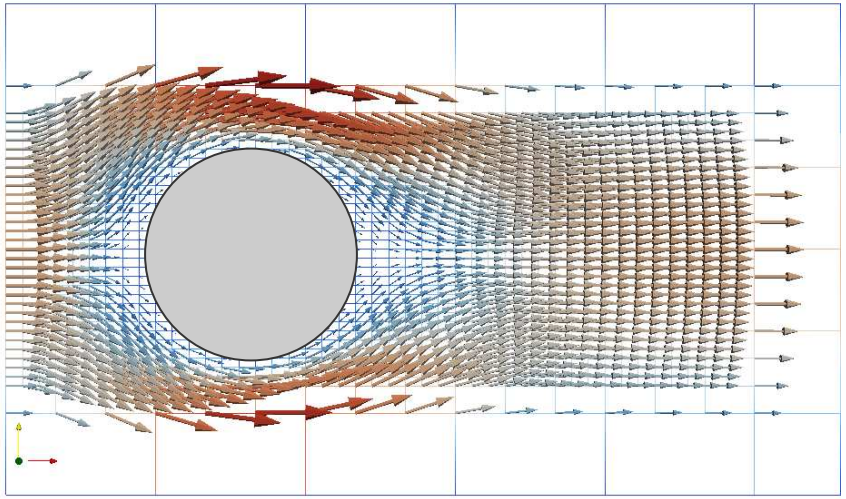


Motivation

- Grids, why?
- PDE solution

(Heat, Mechanical Structure, Fluid, …)

- CAD
- Visualization
- Data mining
- …



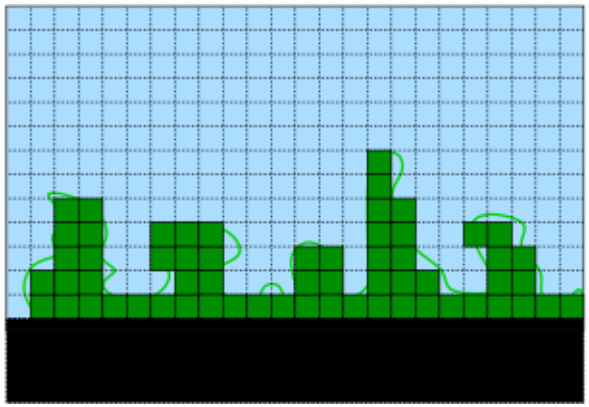
Grid categorization

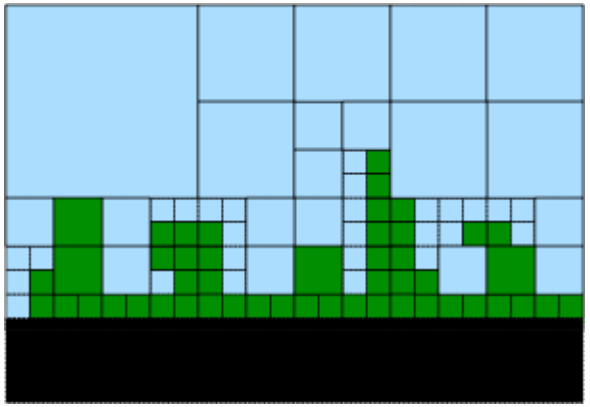
Grid categorization

- Grids can be categorized after different properties, we take the most important.

- **Structured** Grids:
- construction of points or elements follows regular process
- geometric (coordinates) and topological information (neighbour
- relations can be derived (i.e. are not stored)
- memory addresses can be easily computed

- (**Adaptive** Grids, cells with different size)

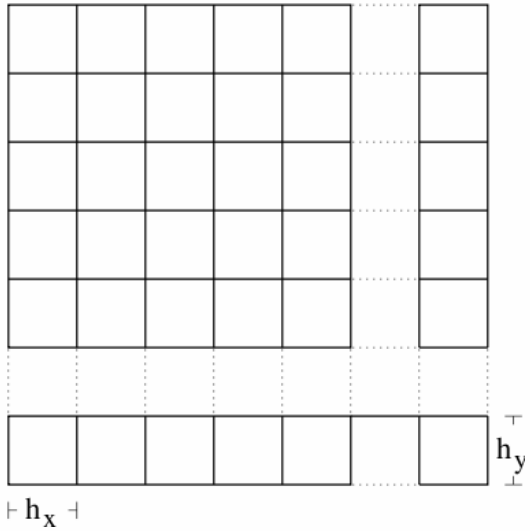


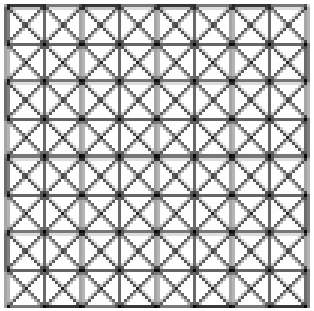


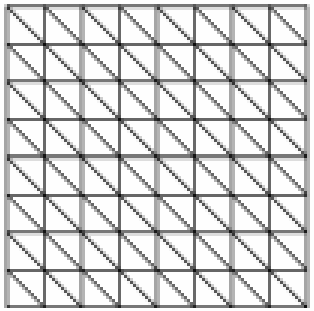
Grid categorization

Regular Structured Grids

- rectangular/cartesian grids:
- rectangles (2D) or cuboids (3D)
- triangular meshes:
- triangles (2D) or tetrahedra (3D)
- row-major or column-major traversal and storage



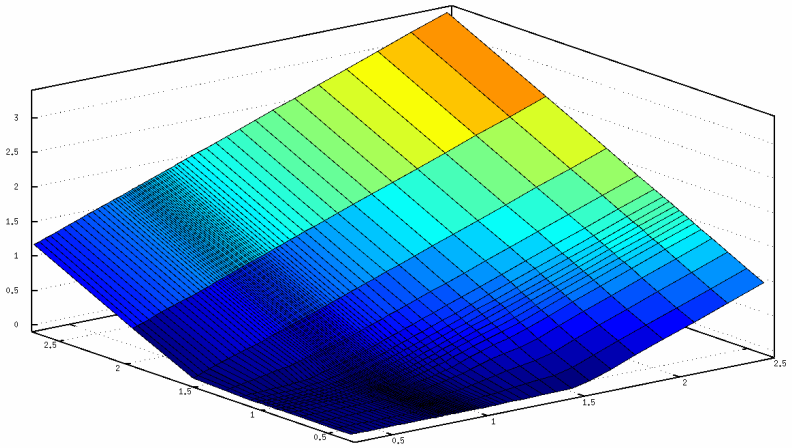


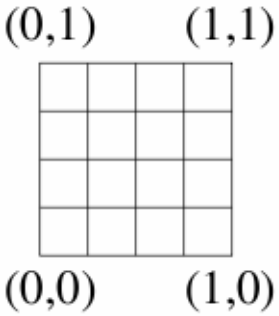


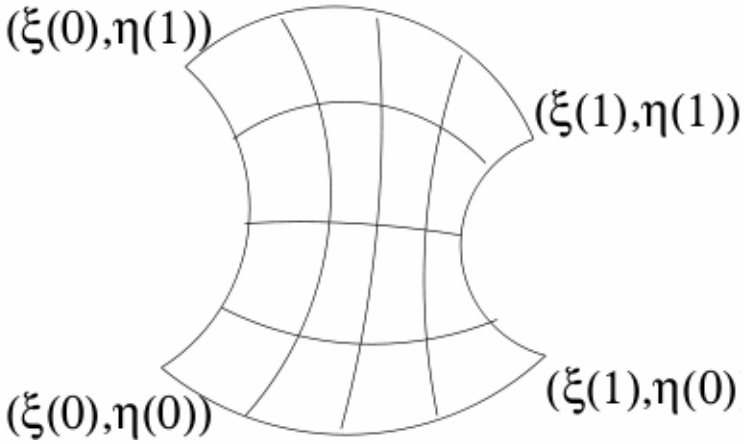
Grid categorization

Transformed Structured Grids

- transformation of the unit square to the computational domain
- regular grid is transformed likewise



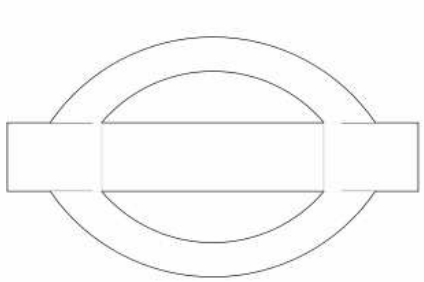


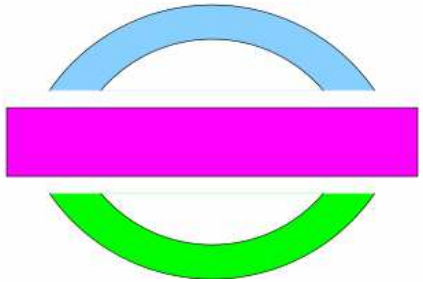


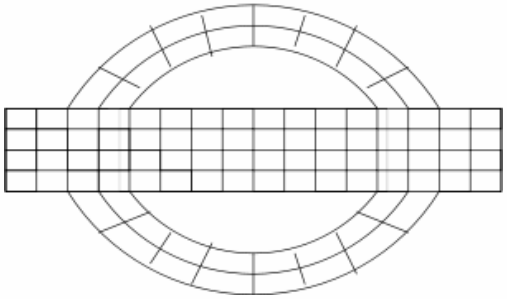
Grid categorization

Composite Structured Grids

- subdivide (complicated) domain into subdomains of simpler form
- and use regular grid on each subdomain
- at interfaces:
- conforming at interface (“glue” required?)
- overlapping grids (chimera grids)



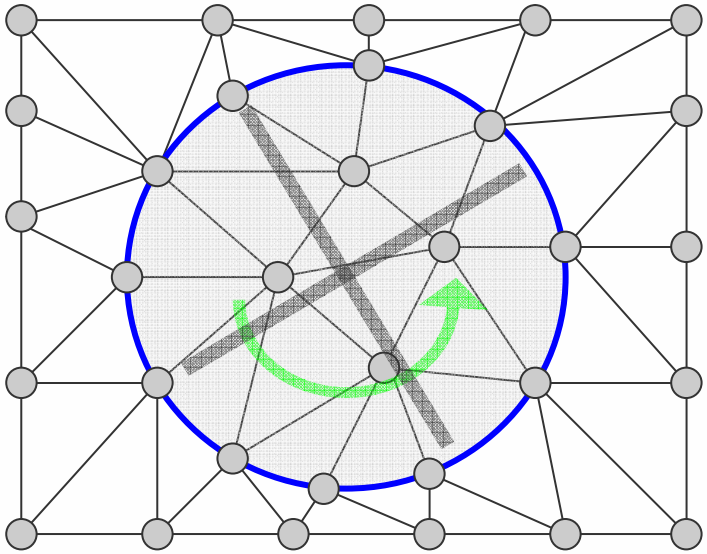




Grid categorization

- **Composite Grids** (similar to composite structural grid) (the example is unstructured, the idea is same)

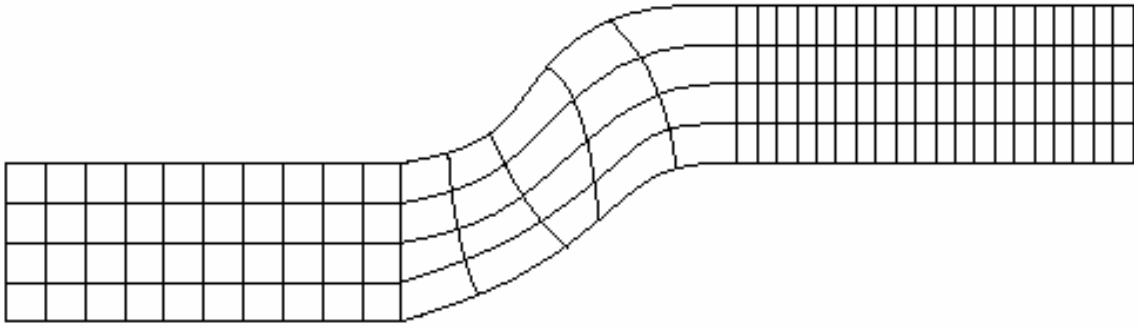
  Example of rotation, e.g. turbine or pump



Grid categorization

Block Structured Grids

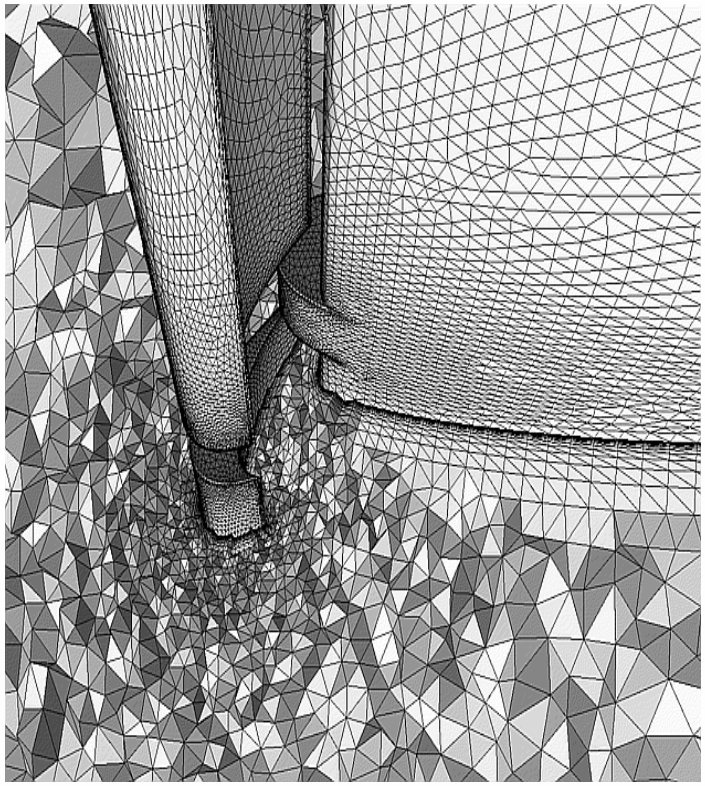
- subdivision into logically rectangular subdomains(with logically rectangular local grids)
- subdomains fit together in an unstructured way, but continuity is ensured (coinciding grid points)
- popular in computational fluid dynamics



Grid categorization

- **UnStructured** Grids:
- (almost) no restrictions, maximum flexibility
- completely irregular generation, even random choice is possible
- explicit storage of basic geometric and topological information
- usually complicated data structures

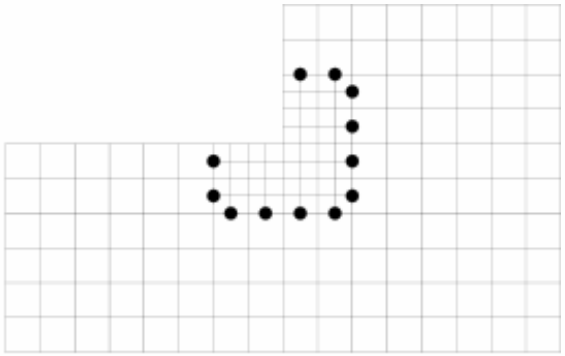
- (adaptivity?)

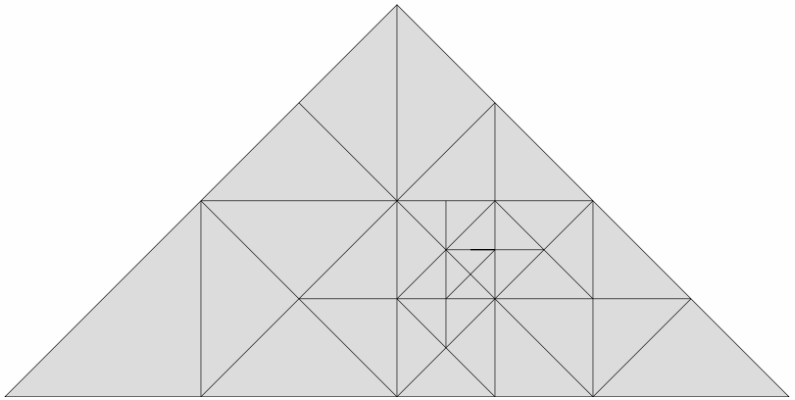


Grid categorization

Adaptive Grids

- Characterization of adaptive grids:
- size of grid cells varies considerably
- to locally improve accuracy
- sometimes requirement from numerics
- Challenge for structured grids:
- efficient storage/traversal
- retrieve structural information (neighbours, etc.)

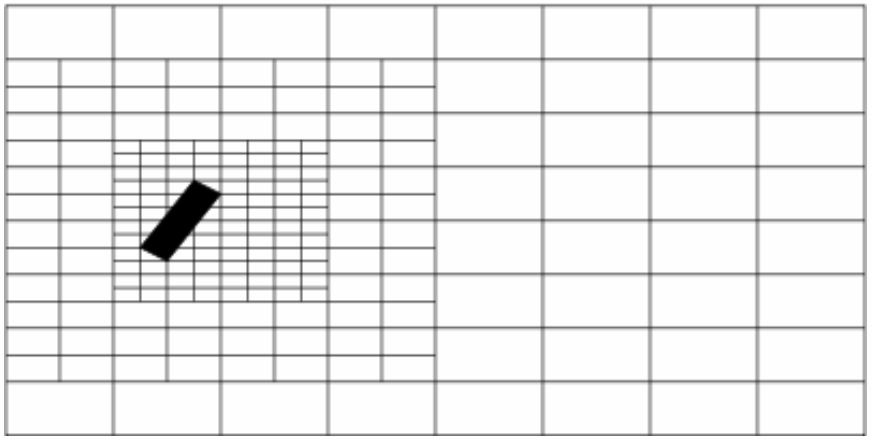




Grid categorization

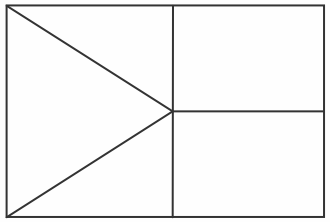
Block Adaptive Grids

- retain regular structure
- refinement of entire blocks
- similar to block structured grids
- efficient storage and processing
- but limited w.r.t. adaptivity

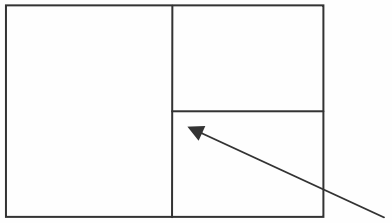


Grid categorization

- **Conform grid** (regular grid)
  with respect to neighbor cell



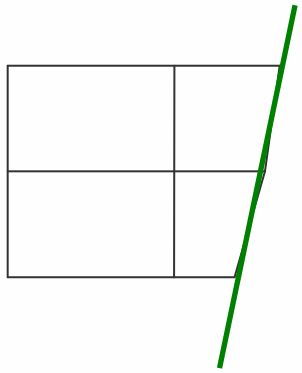
- **Non-conform grid** (irregular)
  with respect to neighbor cell



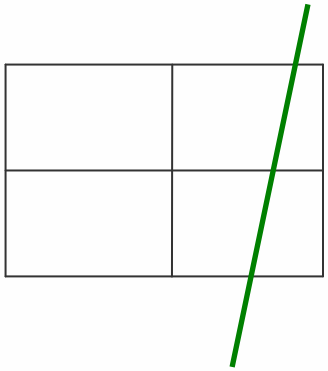
Hanging node

Grid categorization

- **Conform grid** (regular grid)
  with respect to boundary



- **Non-conform grid** (irregular)
  with respect to boundary



Grid Manipulation

- **grid generation**:
  initial placement of grid points or elements
- **grid adaption**:
  need for (additional) grid points often becomes clear only during the computations
  requires possibilities of both refinement and coarsening
- **grid partitioning**:
  standard parallelization techniques are based on some decomposition of the underlying domain

Grid operations

Typical **Operations** on the Grid:

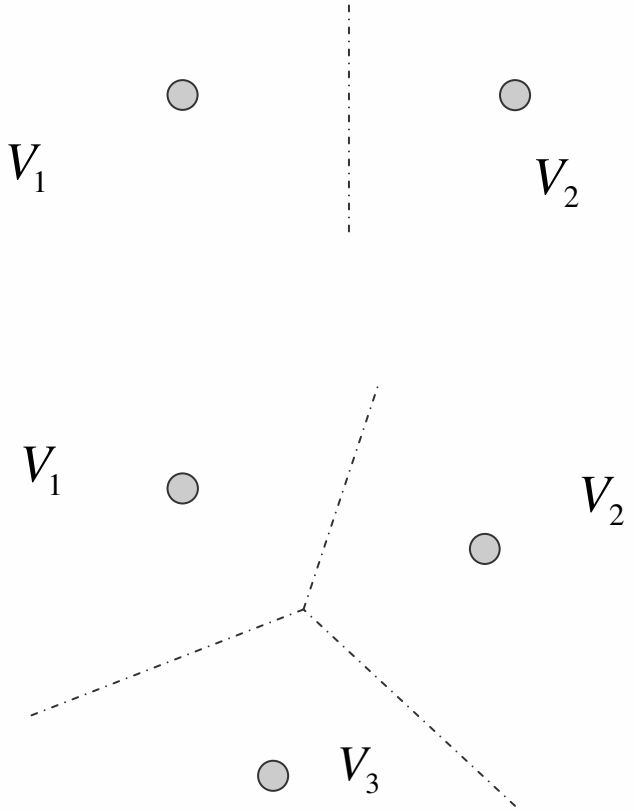
- **numbering** of the nodes/cells
- for traversal/processing of the grid
- for storing the grid
- for partitioning the grid
- **identify** the neighbours of a node/cell
- neighbouring/adjacent nodes/cells/edges
- due to typical discretization techniques

Grid generation

Grid generation

- **UnStructured** Grids in 2D

- assume: grid points are already obtained – how to define elements?
- based on Voronoi diagrams
- Voronoi region around each given grid point:
- Vi = {P : ||P – Pi|| < ||P – Pj|| , j != i }

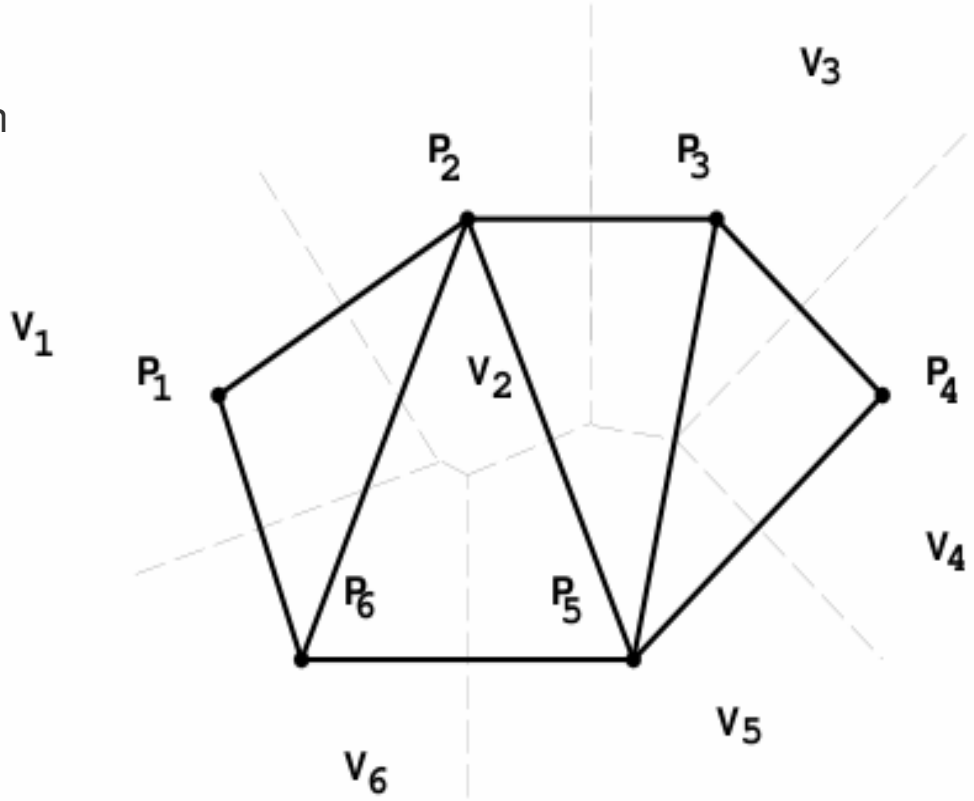


Grid generation

Algorithm (Delaunay triangulation)

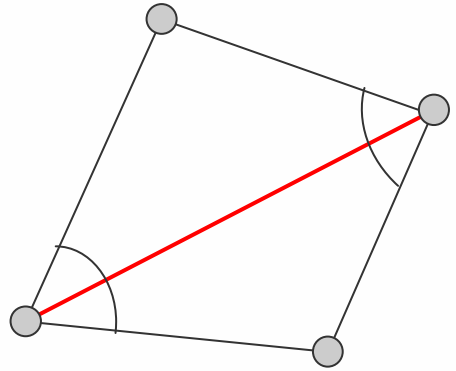
- 1. Voronoi region around each given grid point:

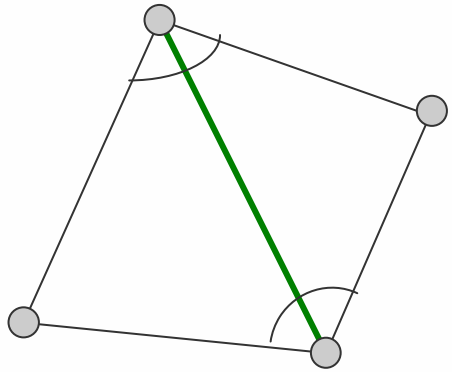
  Vi = {P : ||P – Pi|| < ||P – Pj|| , j != i }
- 2. connect points that are located in adjacent Voronoi regions
- 3. leads to set of disjoint triangles (tetrahedra in 3D)
- Applications:
- closely related to FEM, typically triangles/tetrahedra
- very widespread



Grid generation

- Starting from an initial (non-optimal) triangulation
- Pair wise test the triangles for the “Delaunay condition”
- If the sum of the angles, having the common edge is more than 180
- The flipped triangles satisfy now the “Delaunay condition”



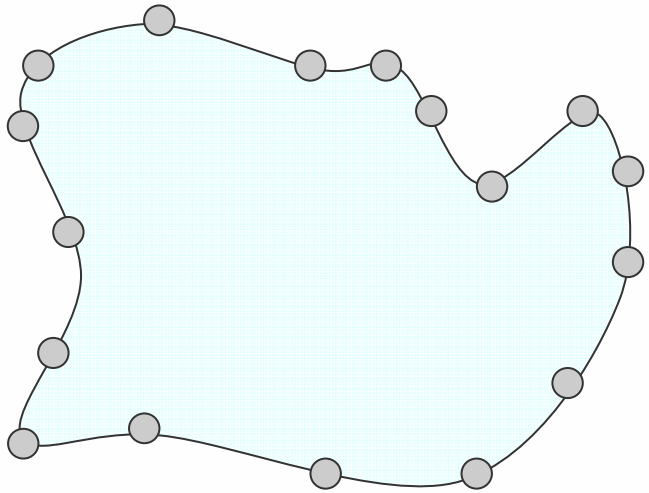


Grid generation

- how do we get the grid points?
- start with a regular grid and refine/modify

boundary-based:

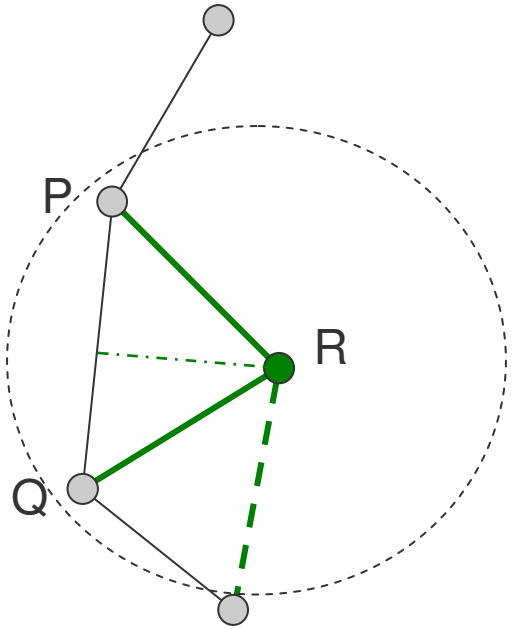
- start with some boundary point distribution, generate Delaunay triangulation, and subdivide (following suitable rules)
- if helpful, add point or lines sources (singularities, bound. layers)
- Advancing Front methods



Grid generation

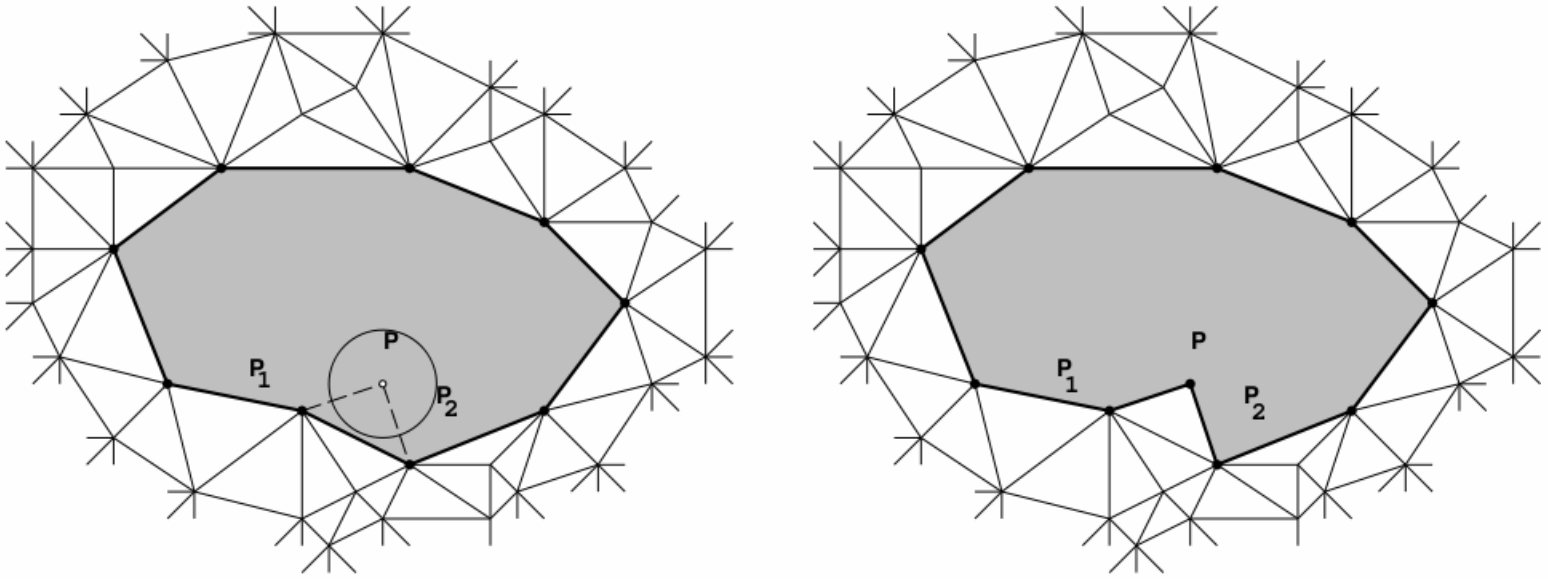
Advancing algorithm:

- 1. choose an edge on the current front, say PQ
- 2. create a new point R at equal distance d from P and Q
- 3. determine all grid points lying within a circle around R, radius r
- 4. order these points w.r.t. distance from R
- 5. for all points, form triangles with P and Q; select the triangles
- 6. add triangles to grid (unless: intersections, . . .)
- 7. update triangulation and front line: add new cell, update edges



Grid generation

- advance a front step-by-step towards interior
- starting from the boundary (starting front)



Grid generation

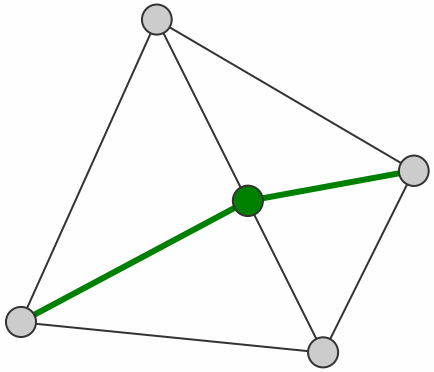
Refinement algorithm:

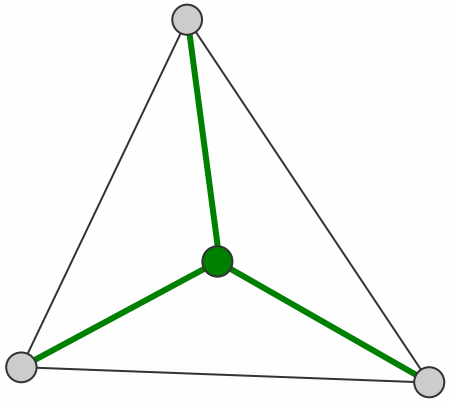
indicator function per cell

avoid the creation of hanging nodes

“longest edge partition”, the most common one

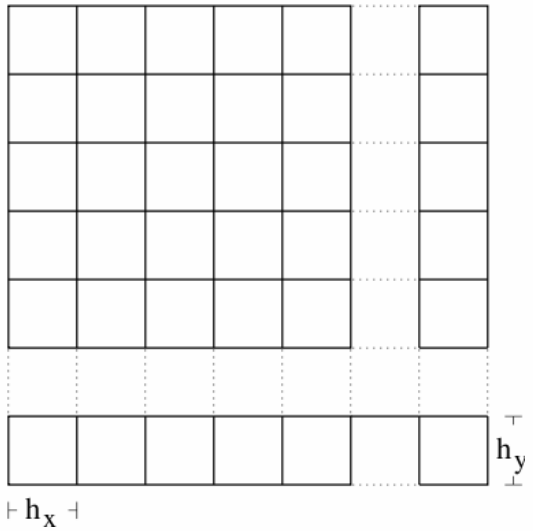
dividing the triangle in 3 other triangles

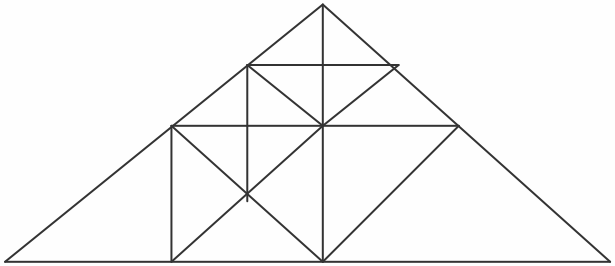


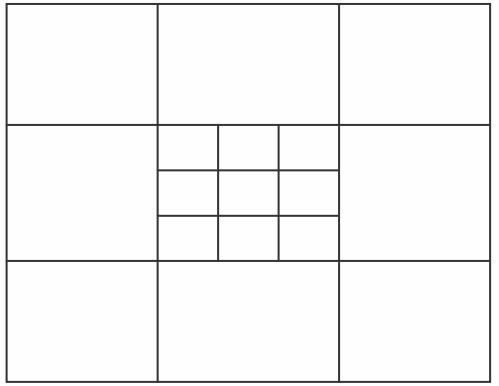


Grid generation

- **Structured grids:**
- Cartesian grids
- Adaptivity in structured grids

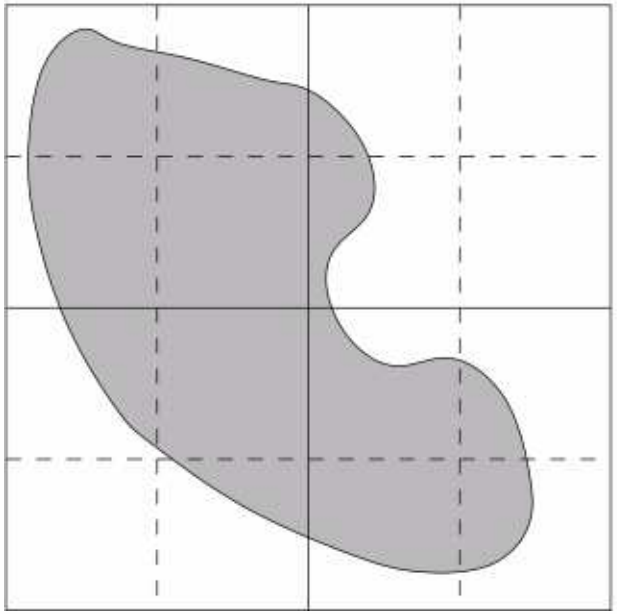


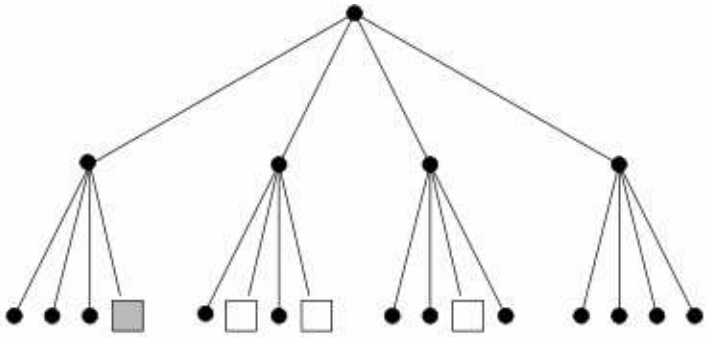




Grid generation

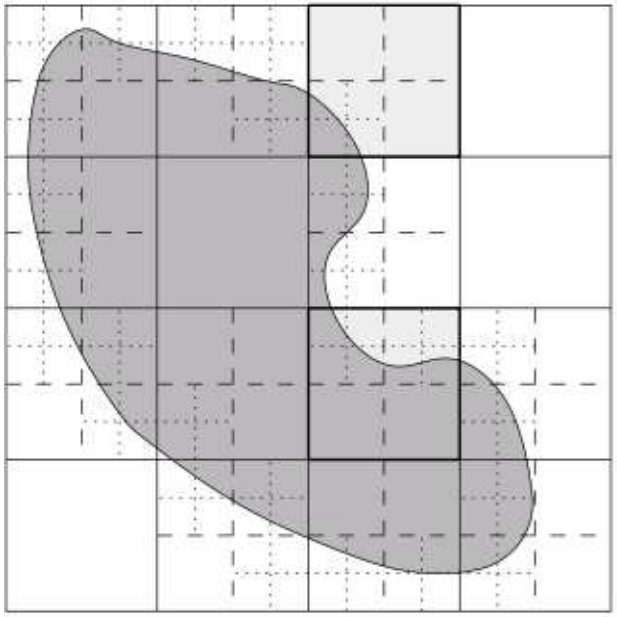
- **Adaptive structured grids:**
- Structure described through a tree (quadtree, octtree in 3D)
- Information per cell is minimal (one bit)
- Such trees can also be used for point localization

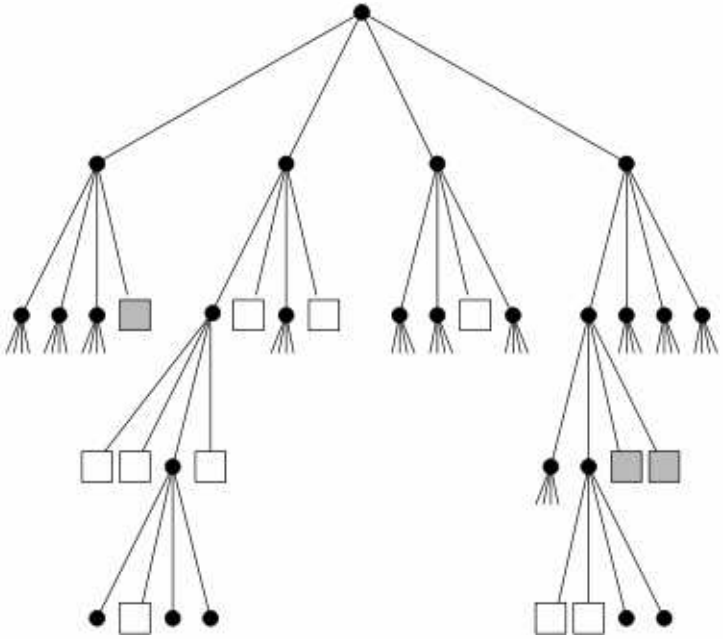




Grid generation

- **Adaptive structured grids:**
- Further refinement

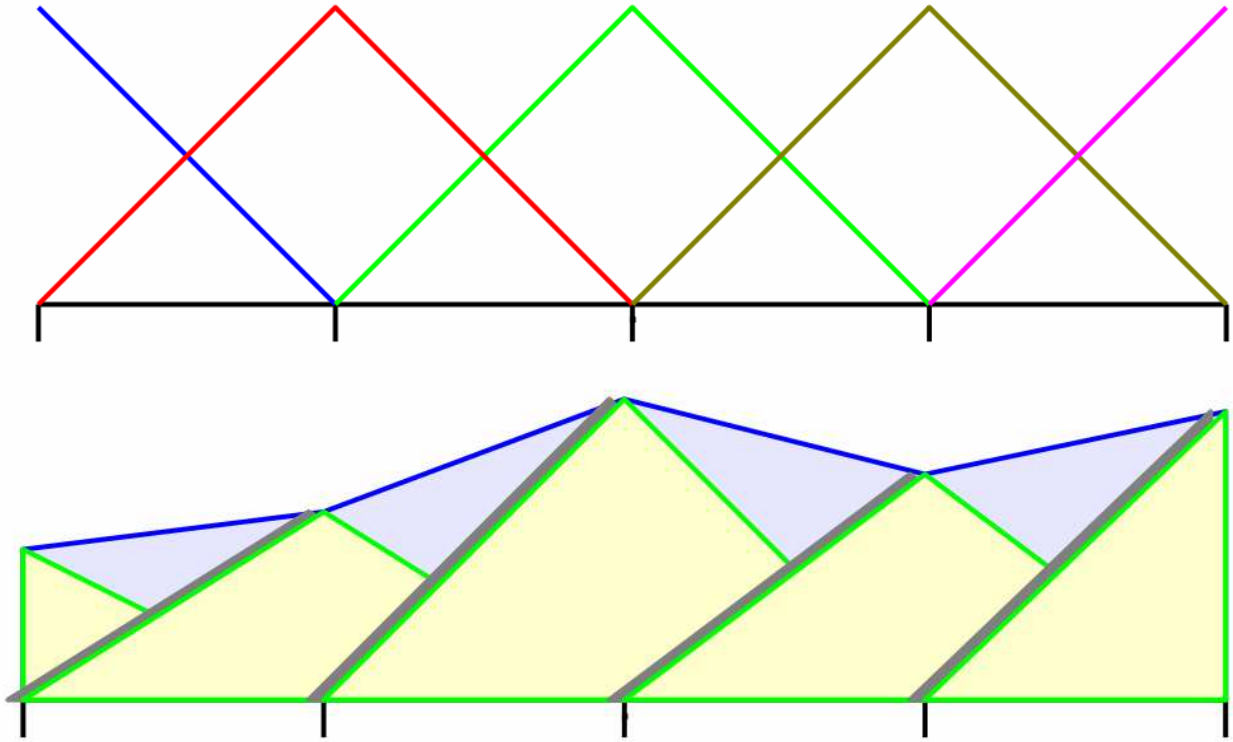




Hierarchical basis and sparse grids

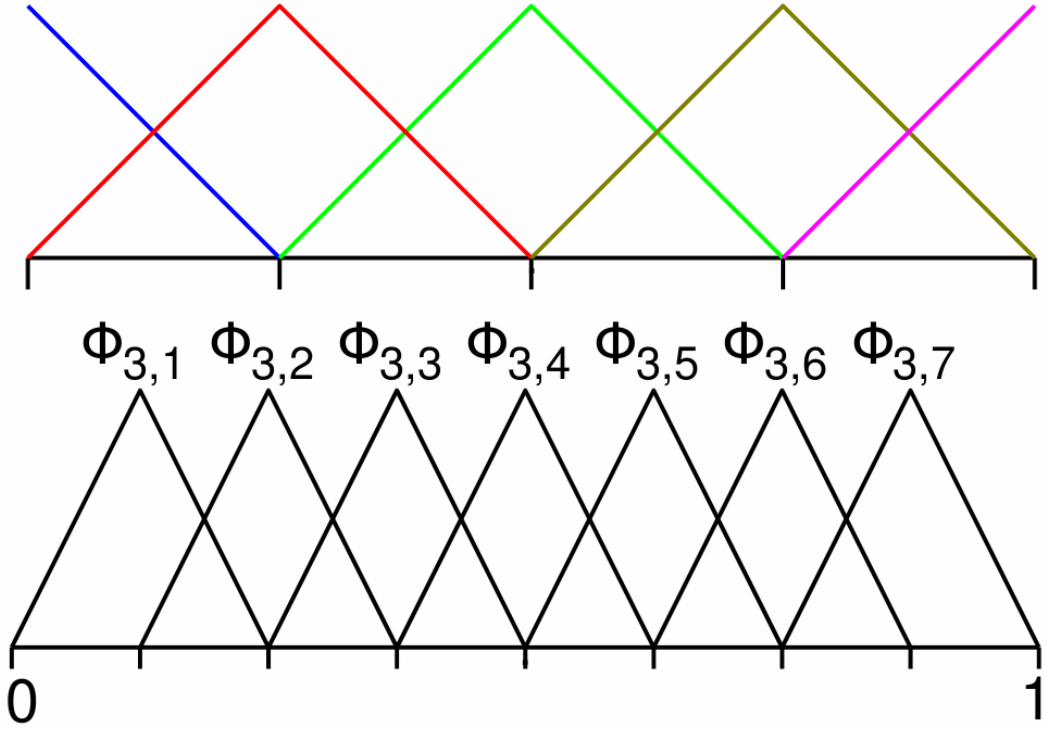
Hierarchical basis

- Linear nodal basis



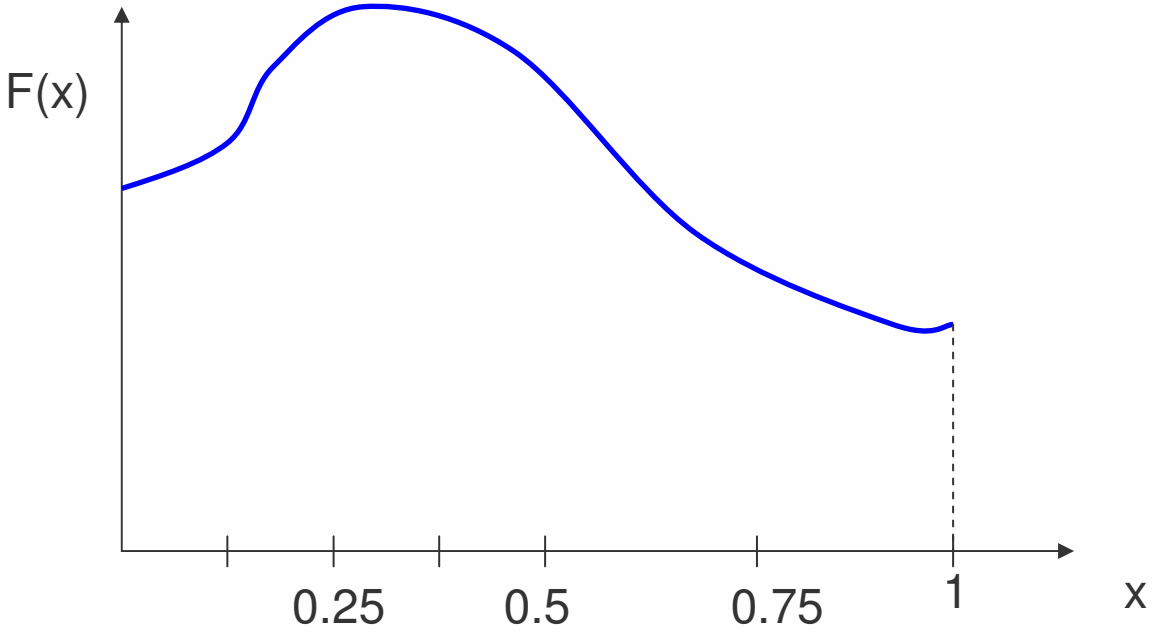
Hierarchical basis

- Linear nodal basis, different refinement stage



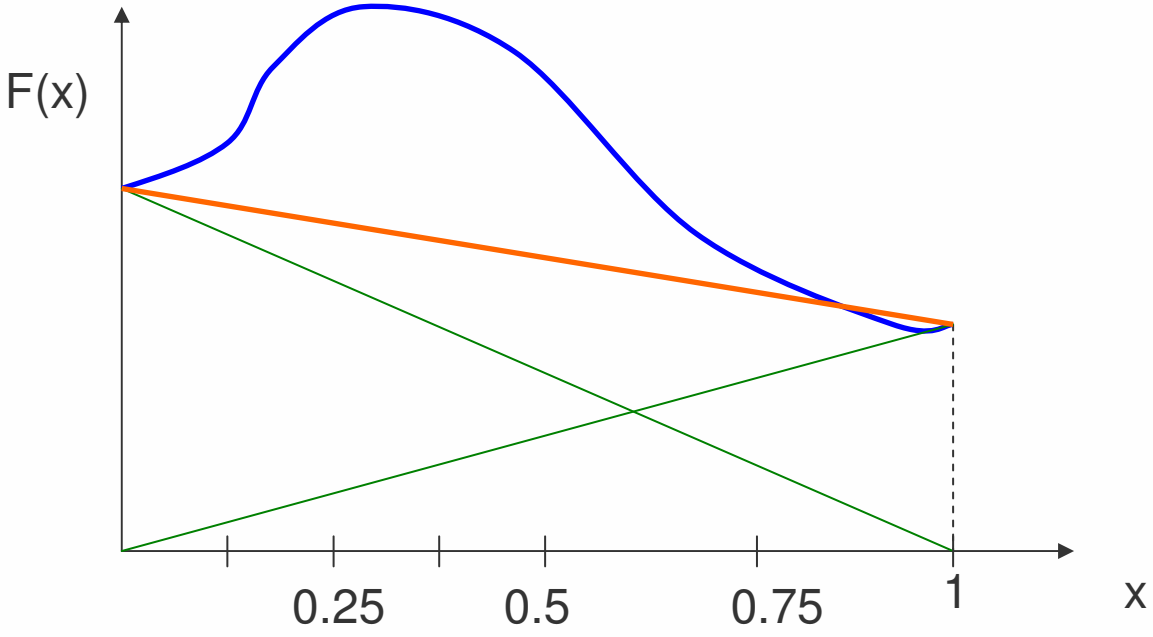
Hierarchical basis

- Do it hierarchically (Archimedes' Hierarchical Approach)



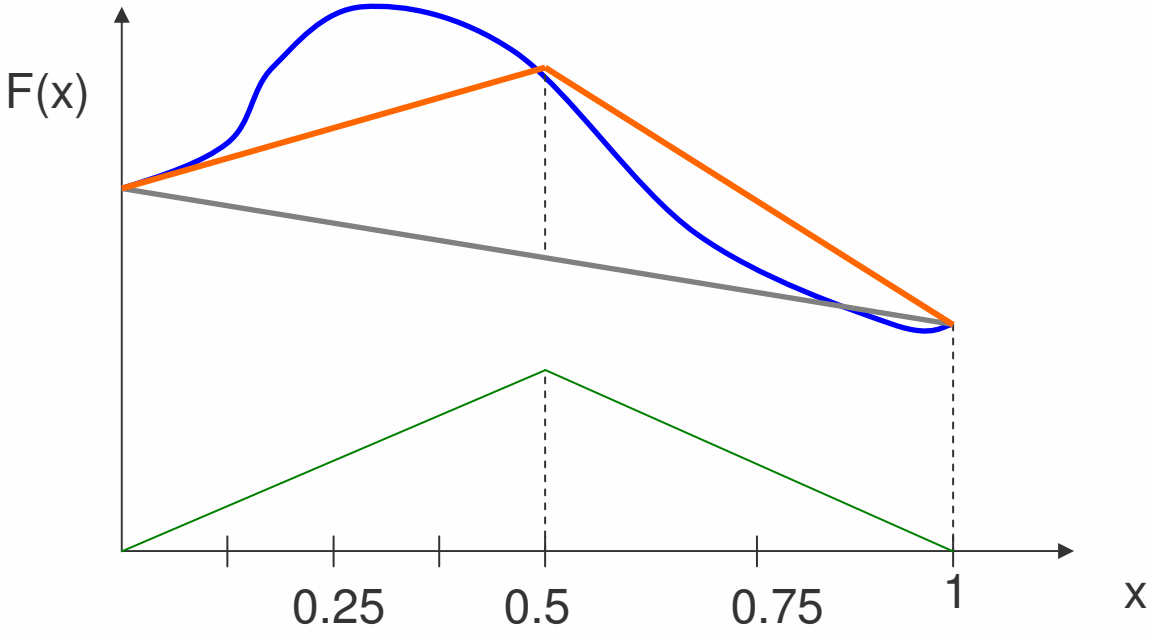
Hierarchical basis

- Do it hierarchically (Archimedes' Hierarchical Approach)



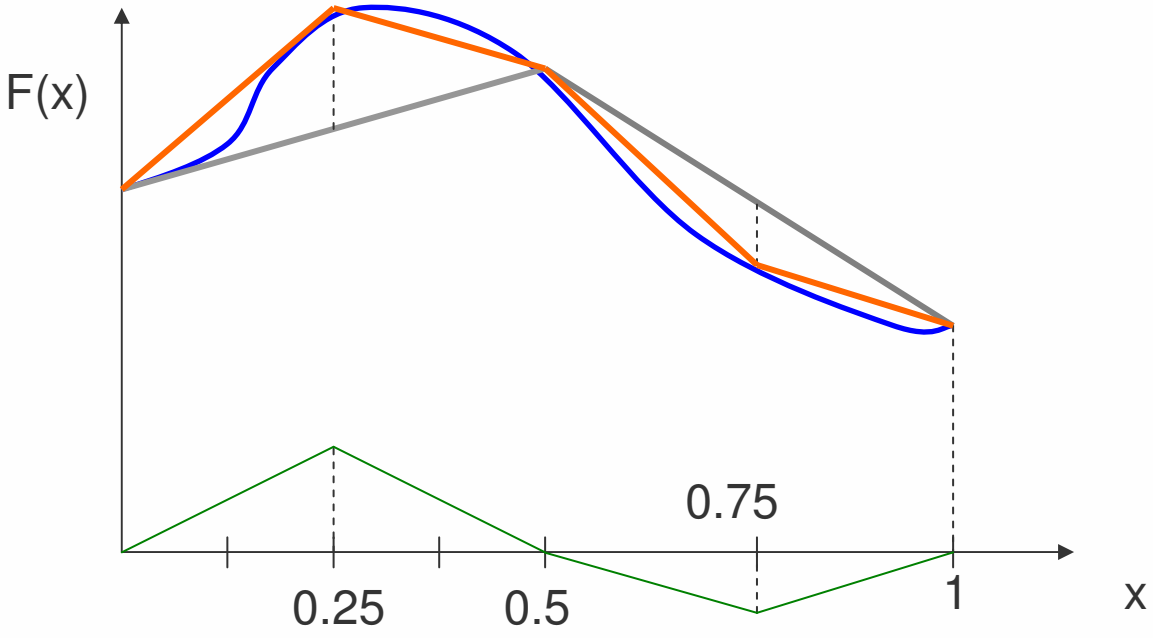
Hierarchical basis

- Do it hierarchically (Archimedes' Hierarchical Approach)



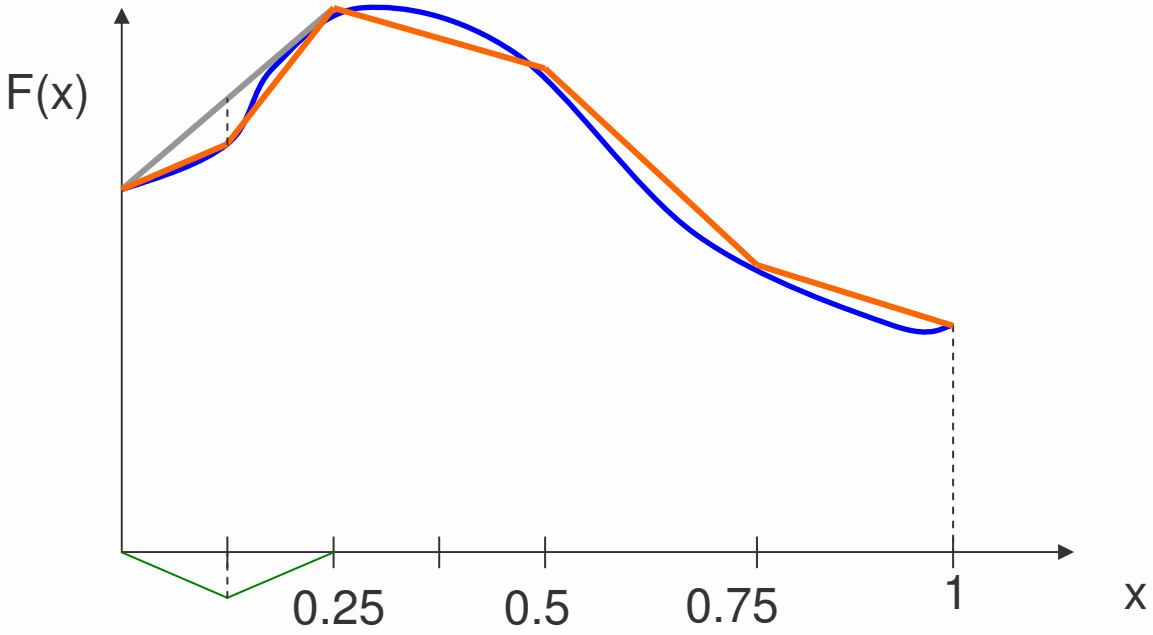
Hierarchical basis

- Do it hierarchically (Archimedes' Hierarchical Approach)



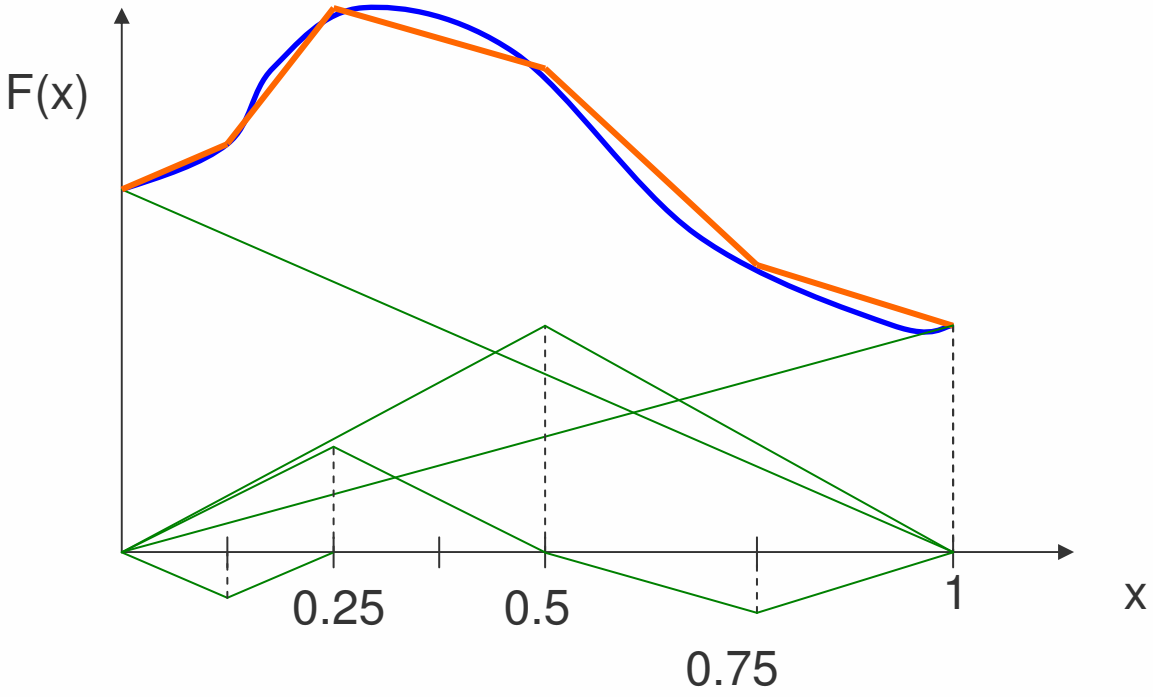
Hierarchical basis

- Do it hierarchically (Archimedes' Hierarchical Approach)



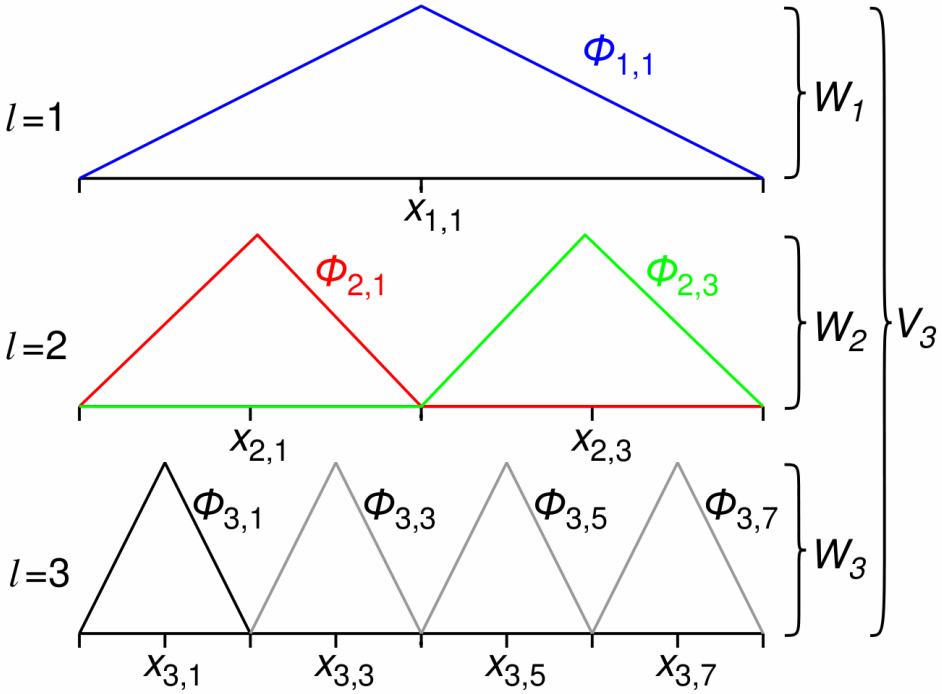
Hierarchical basis

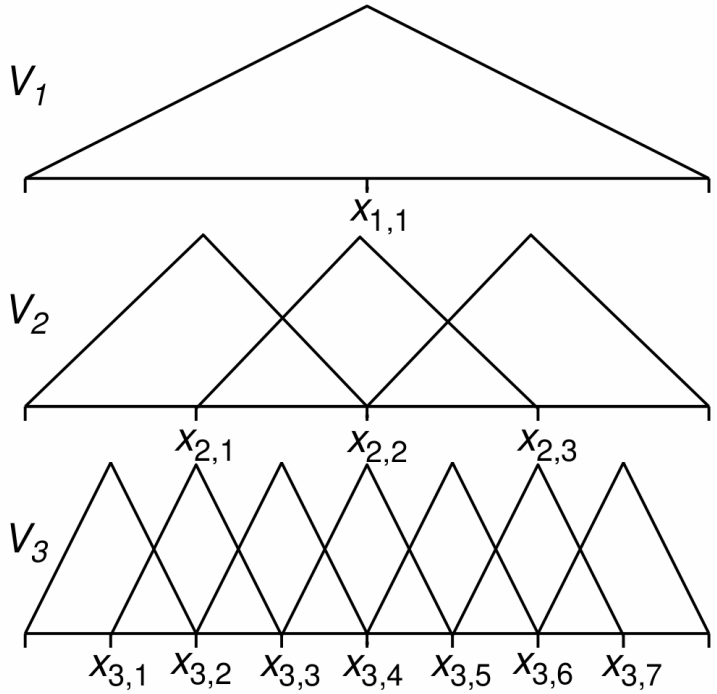
- Do it hierarchically (Archimedes' Hierarchical Approach)



Hierarchical basis

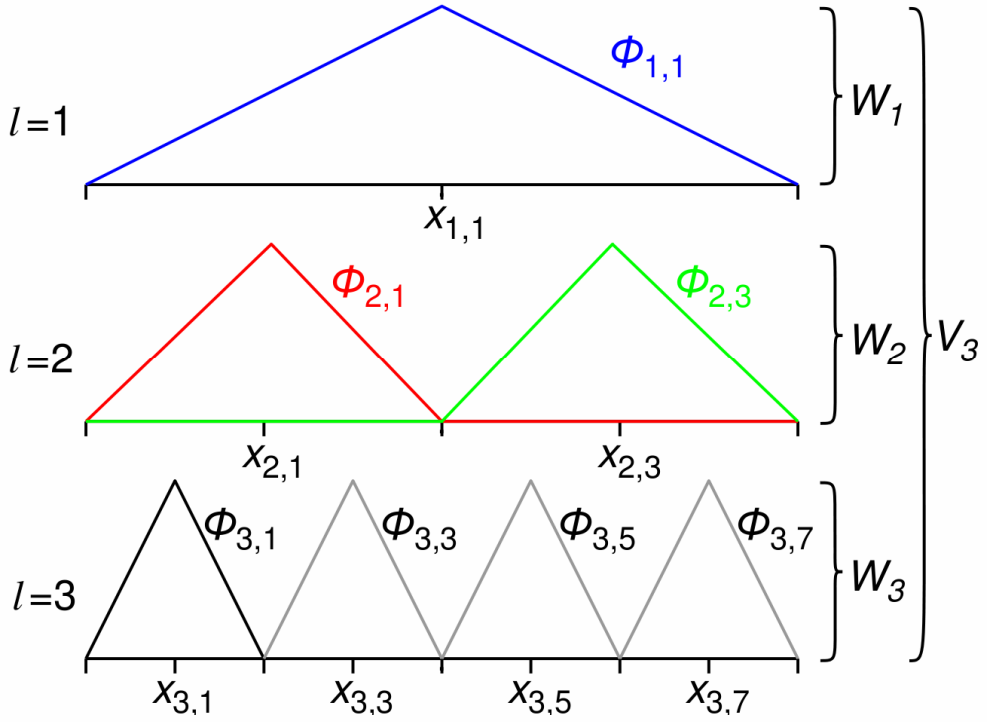
- Hierarchical vs. nodal basis





Hierarchical basis

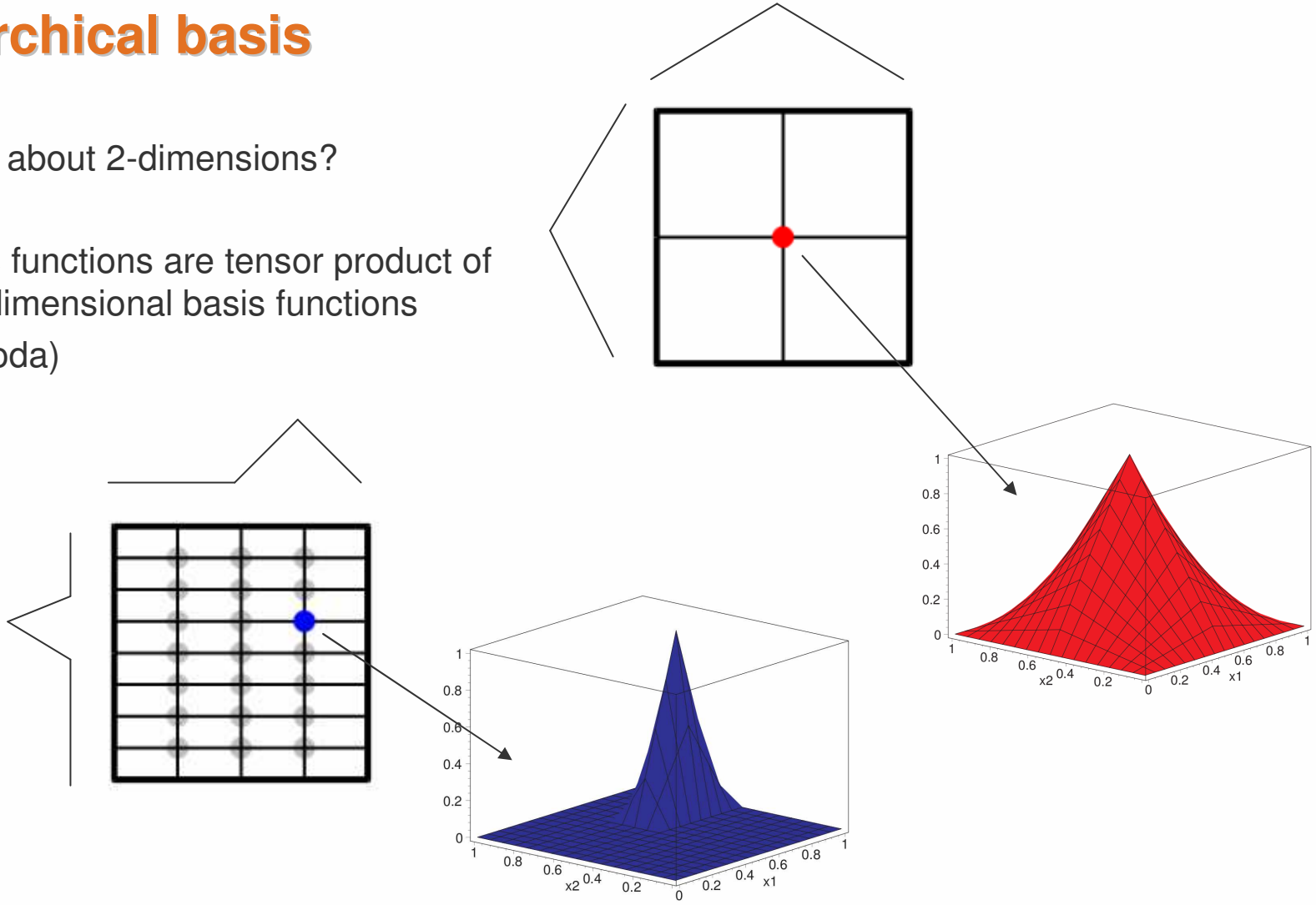
- Hierarchical basis



$$V_n = \bigoplus_{l=1}^{n} W_l$$

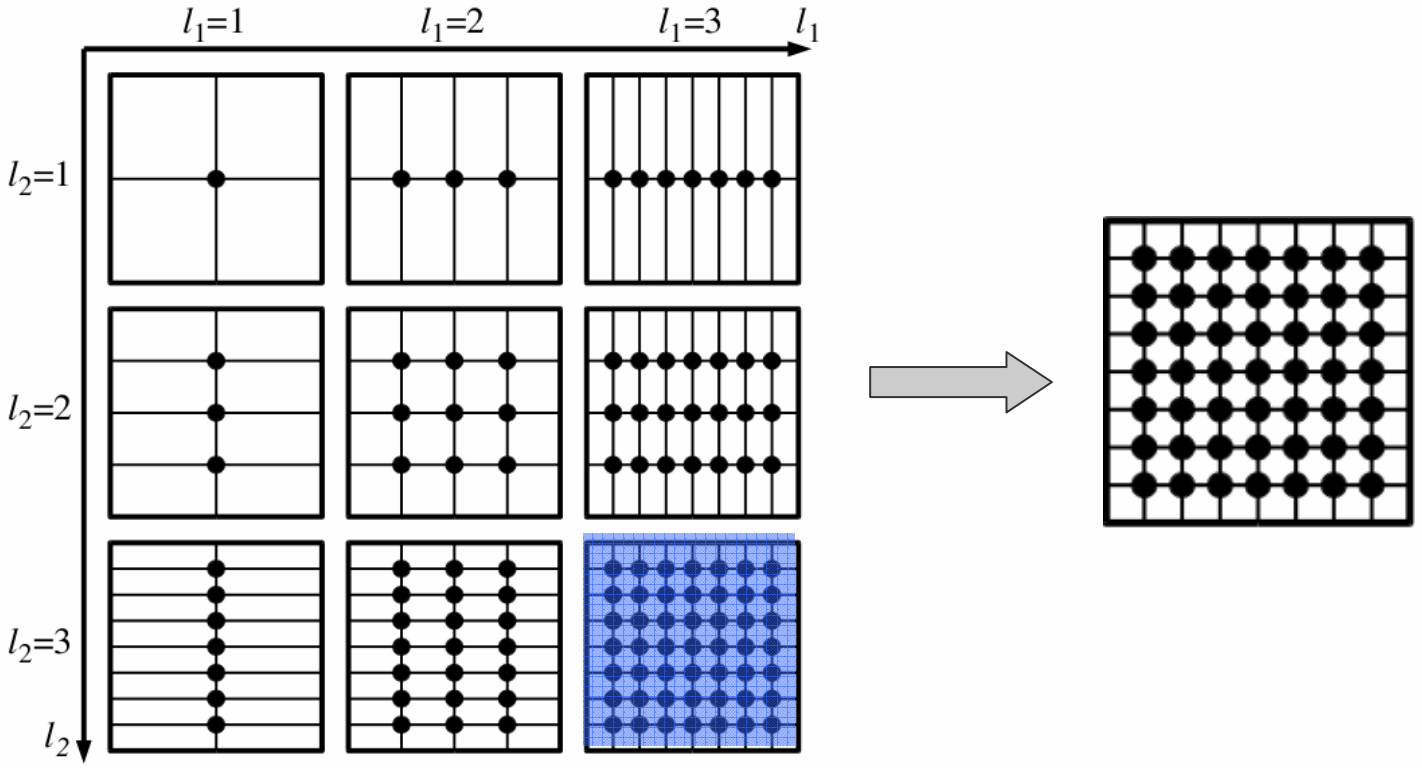
Hierarchical basis

- What about 2-dimensions?
- Basis functions are tensor product of one dimensional basis functions
- (Pagoda)



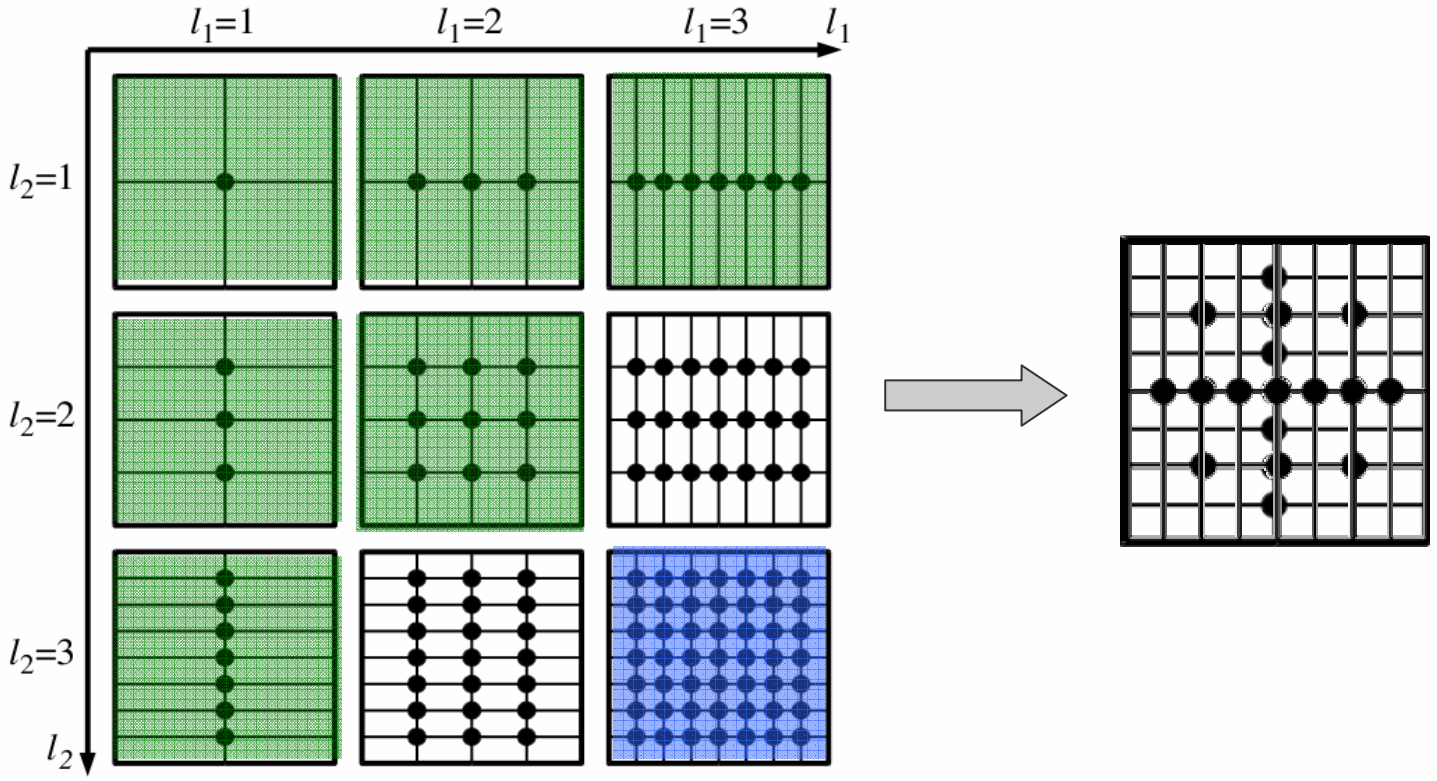
Sparse grids

- 2-dimensions, full grid



Sparse grids

- 2-dimensions, could we use less points (sparse grid)



Sparse grids

- Number of grid points
- 2 dimensions

  l1=l2=10
  Full grid: **1 046 529** vs. Sparse grid: **9 217**

- 3 dimensions

  l1=l2=10
  Full grid: **1 070 590 167** vs. Sparse grid: **47 103**

- In higher dimensions even favorable for sparse grids , reducing the “curse of dimensionality”:

Thank you for your attention!