Hierarchical Grid Adaptation for Hybrid Meshes

Matthias Möller

matthias.moeller@math.uni-dortmund.de

Institute of Applied Mathematics (LS III)

Technische Universität Dortmund

Germany

WCCM/ECCOMAS, Venice – 2008

Overview

1. Motivation
2. Grid adaptation algorithm
   - Red-green refinement
   - Re-coarsening
3. Implementation details
4. Numerical examples
   - Scalar transport problems
   - Compressible Euler equations
5. Conclusions

Why use hybrid grids?

Unstructured grids (triangles/quadrilaterals and tetrahedra/hexahedra)

- triangulation of complex domains (Delaunay, advancing front)
- prevent distorted grids (e.g., near singular points)
- overhead costs due to indirect addressing

Structured grids (Cartesian and/or generalized tensor-product grids)

- efficient numerics based on line-wise numbering (SBBLAS)
- orthogonal grids to resolve boundary layers
- unflexible/impractical for complex domains

Hierarchical hybrid grids

Use **globally unstructured** coarse grid and employ **locally structured** and/or **locally unstructured** meshes for triangulation in each patch.

Why use hybrid grids?

Unstructured grids (triangles/quadrilaterals and tetrahedra/hexahedra)

- triangulation of complex domains (Delaunay, advancing front)
- prevent distorted grids (e.g., near singular points)
- overhead costs due to indirect addressing

Structured grids (Cartesian and/or generalized tensor-product grids)

- efficient numerics based on line-wise numbering (SBBLAS)
- orthogonal grids to resolve boundary layers
- unflexible/impractical for complex domains

Hierarchical hybrid grids

Use **globally unstructured** coarse grid and employ **locally structured** and/or **locally unstructured** meshes for triangulation in each patch.

Design goals for adaptivity

- Adaptation strategy should lead to **conforming triangulations**
- Mesh quality should not deteriorate due to **grid refinement**
- **Grid coarsening** should 'undo' previous refinement steps
- **Coarse/initial grid** should be preserved for all times
- Dynamic grid adaptation must be **efficient** for transient flows

Literature

Bank83 R. E. Bank, A. H. Sherman and A. Weiser, *Refinement algorithms and data structures for regular local mesh refinements*, in: Scientific Computing, eds. R. Stepleman et al. (IMACS, North-Holland, Amsterdam, 1983) pp. 3-17

Hem99 D. Hempel, *Rekonstruktionsverfahren auf unstrukturierten Gittern zur numerischen Simulation von Erhaltungsprinzipien*, PhD (in German), Hamburg 1999

Design goals for adaptivity

- Adaptation strategy should lead to **conforming triangulations**
- Mesh quality should not deteriorate due to **grid refinement**
- **Grid coarsening** should 'undo' previous refinement steps
- **Coarse/initial grid** should be preserved for all times
- Dynamic grid adaptation must be **efficient** for transient flows

Literature

Bank83 R. E. Bank, A. H. Sherman and A. Weiser, *Refinement algorithms and data structures for regular local mesh refinements*, in: Scientific Computing, eds. R. Stepleman et al. (IMACS, North-Holland, Amsterdam, 1983) pp. 3-17

Hem99 D. Hempel, *Rekonstruktionsverfahren auf unstrukturierten Gittern zur numerischen Simulation von Erhaltungsprinzipien*, PhD (in German), Hamburg 1999

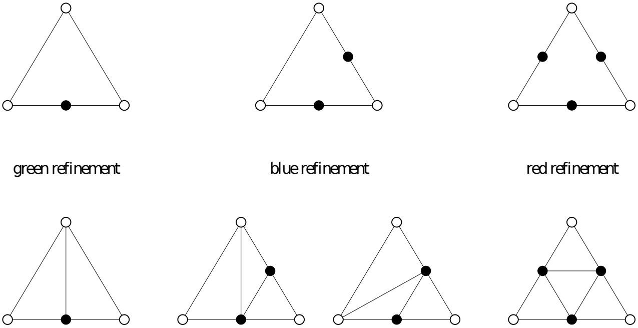
Principle: *Regular refinement in 2D [Bank83]*

- Subdivide each marked cell into four similar cells (red refinement)
- Eliminate ‘hanging nodes’ by introducing transition elements that are removed prior to performing further refinement (green refinement)

Red-green refinement revisited

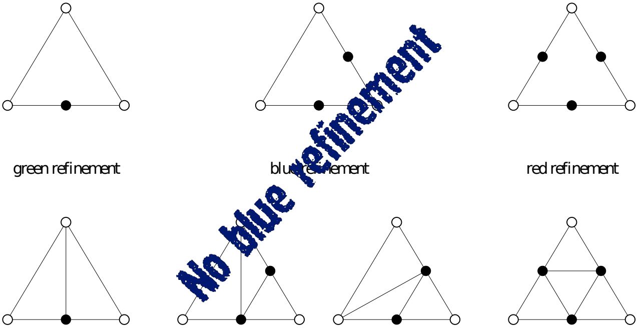
Principle: *Regular refinement in 2D [Bank83]*

- Subdivide each marked cell into four similar cells (red refinement)
- Eliminate 'hanging nodes' by introducing transition elements that are removed prior to performing further refinement (green refinement)



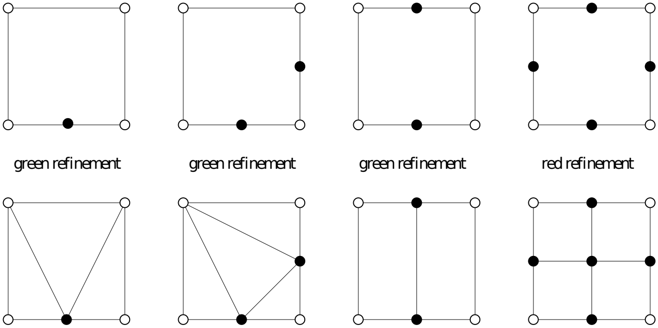
Principle: *Regular refinement in 2D [Bank83]*

- Subdivide each marked cell into four similar cells (red refinement)
- Eliminate 'hanging nodes' by introducing transition elements that are removed prior to performing further refinement (green refinement)



Principle: *Regular refinement in 2D [Bank83]*

- Subdivide each marked cell into four similar cells (red refinement)
- Eliminate 'hanging nodes' by introducing transition elements that are removed prior to performing further refinement (green refinement)



Initial triangulation $\mathcal{T}_0(\mathcal{E}_0, \mathcal{V}_0)$ consists of

- a set of **elements** $\mathcal{E}_0 = \{\Omega_k \,:\, k = 1, \dots N_\mathrm{E}\}$ and
- a set of **vertices** $\mathcal{V}_0 = \{v_i \,:\, i = 1, \dots, N_\mathrm{V}\}$

The red-green refinement algorithm is used to transform a conforming triangulation $\mathcal{T}_{m-1}$ into a conforming triangulation $\mathcal{T}_m$, $m = 1, 2, \dots$.

Generation function: $g : \mathcal{V}_m \to \mathbb{N}_0$ *(triangles in 2D [Hem99])*

$$g(v_i) := \begin{cases} 0 & \text{if} \quad v_i \in \mathcal{V}_0 & \text{initial triangulation} \\ \max\limits_{v_j \in \Omega_k \cap \Omega_l} g(v_j) + 1 & \text{if} \quad v_i \in \Omega_k \cap \Omega_l & \text{face/edge vertex} \\ \max\limits_{v_j \in \Omega_k} g(v_j) + 1 & \text{if} \quad v_i \in \Omega_k \setminus \partial\Omega_k & \text{interior vertex} \end{cases}$$

Generation function can be used to prescribe maximum refinement level!

Initial triangulation $\mathcal{T}_0(\mathcal{E}_0, \mathcal{V}_0)$ consists of

- a set of **elements** $\mathcal{E}_0 = \{\Omega_k \ : \ k = 1, \dots N_{\mathrm{E}}\}$ and
- a set of **vertices** $\mathcal{V}_0 = \{v_i \ : \ i = 1, \dots, N_{\mathrm{V}}\}$

The red-green refinement algorithm is used to transform a conforming triangulation $\mathcal{T}_{m-1}$ into a conforming triangulation $\mathcal{T}_m$, $m = 1, 2, \dots$.

Generation function: $g : \mathcal{V}_m \to \mathbb{N}_0$ *(triangles in 2D [Hem99])*

$$g(v_i) := \begin{cases} 0 & \text{if} \quad v_i \in \mathcal{V}_0 & \text{initial triangulation} \\ \max\limits_{v_j \in \Omega_k \cap \Omega_l} g(v_j) + 1 & \text{if} \quad v_i \in \Omega_k \cap \Omega_l & \text{face/edge vertex} \\ \max\limits_{v_j \in \Omega_k} g(v_j) + 1 & \text{if} \quad v_i \in \Omega_k \setminus \partial\Omega_k & \text{interior vertex} \end{cases}$$

Generation function can be used to prescribe maximum refinement level!

Initial triangulation $\mathcal{T}_0(\mathcal{E}_0, \mathcal{V}_0)$ consists of

- a set of **elements** $\mathcal{E}_0 = \{\Omega_k \,:\, k = 1, \ldots N_{\mathrm{E}}\}$ and
- a set of **vertices** $\mathcal{V}_0 = \{v_i \,:\, i = 1, \ldots, N_{\mathrm{V}}\}$

The red-green refinement algorithm is used to transform a conforming triangulation $\mathcal{T}_{m-1}$ into a conforming triangulation $\mathcal{T}_m$, $m = 1, 2, \ldots$.

Generation function: $g : \mathcal{V}_m \to \mathbb{N}_0$ *(triangles in 2D [Hem99])*

$$g(v_i) := \begin{cases} 0 & \text{if} \quad v_i \in \mathcal{V}_0 & \text{initial triangulation} \\ \max\limits_{v_j \in \Omega_k \cap \Omega_l} g(v_j) + 1 & \text{if} \quad v_i \in \Omega_k \cap \Omega_l & \text{face/edge vertex} \\ \max\limits_{v_j \in \Omega_k} g(v_j) + 1 & \text{if} \quad v_i \in \Omega_k \setminus \partial\Omega_k & \text{interior vertex} \end{cases}$$

Generation function can be used to prescribe maximum refinement level!

Initial triangulation $\mathcal{T}_0(\mathcal{E}_0, \mathcal{V}_0)$ consists of

- a set of **elements** $\mathcal{E}_0 = \{\Omega_k \;:\; k = 1, \dots N_\mathrm{E}\}$ and
- a set of **vertices** $\mathcal{V}_0 = \{v_i \;:\; i = 1, \dots, N_\mathrm{V}\}$

The red-green refinement algorithm is used to transform a conforming triangulation $\mathcal{T}_{m-1}$ into a conforming triangulation $\mathcal{T}_m$, $m = 1, 2, \dots$.

Generation function: $g : \mathcal{V}_m \to \mathbb{N}_0$ *(triangles in 2D [Hem99])*

$$g(v_i) := \begin{cases} 0 & \text{if} \quad v_i \in \mathcal{V}_0 & \text{initial triangulation} \\ \max\limits_{v_j \in \Omega_k \cap \Omega_l} g(v_j) + 1 & \text{if} \quad v_i \in \Omega_k \cap \Omega_l & \text{face/edge vertex} \\ \max\limits_{v_j \in \Omega_k} g(v_j) + 1 & \text{if} \quad v_i \in \Omega_k \setminus \partial\Omega_k & \text{interior vertex} \end{cases}$$

Generation function can be used to prescribe maximum refinement level!

Goal: Each refinement operation is reversed in a single step
$\Rightarrow$ re-coarsening must start from the red elements

Locking: $d : \mathcal{V}_m \to \mathbb{Z}; \quad d(v_i) := g(v_i)\ \forall v_i \in \mathcal{V}_m$ [Hem99]

- Vertex v_i is locked, i.e. $d(v_i) := -|d(v_i)|$ if it belongs to
  - an element that is marked for further refinement
  - an edge ij and there is a vertex v_j such that $g(v_j) > g(v_i)$
  - a red element which should not be coarsened due to accuracy
- Vertices are locked if blue elements would be created otherwise
  - If two nodes of an inner red triangle are locked, then lock third vertex
  - If the 'interior' node of a patch of four quadrilaterals is locked or if more than six nodes are locked, then lock all nodes of that patch

Result: Vertices $v_i \in \mathcal{V}_m$ with $d(v_i) \leq 0$ are locked; all other nodes are 'free' and can be deleted.

$d(v_i) = 0 \quad \forall v_i \in \mathcal{V}_0$

Re-coarsening algorithm

Goal: Each refinement operation is reversed in a single step
$\Rightarrow$ re-coarsening must start from the red elements

Locking: $d : \mathcal{V}_m \to \mathbb{Z}; \quad d(v_i) := g(v_i)\ \forall v_i \in \mathcal{V}_m$ [Hem99]

1. Vertex v_i is locked, i.e. $d(v_i) := -|d(v_i)|$ if it belongs to
   - an element that is marked for further refinement
   - an edge ij and there is a vertex v_j such that $g(v_j) > g(v_i)$
   - a red element which should not be coarsened due to accuracy
2. Vertices are locked if blue elements would be created otherwise
   - If two nodes of an inner red triangle are locked, then lock third vertex
   - If the 'interior' node of a patch of four quadrilaterals is locked or if more than six nodes are locked, then lock all nodes of that patch

Result: Vertices $v_i \in \mathcal{V}_m$ with $d(v_i) \leq 0$ are locked; all other nodes are 'free' and can be deleted.

$d(v_i) = 0$
$\forall v_i \in \mathcal{V}_0$

Goal: Each refinement operation is reversed in a single step
$\Rightarrow$ re-coarsening must start from the red elements

Locking: $d : \mathcal{V}_m \rightarrow \mathbb{Z}; \quad d(v_i) := g(v_i) \; \forall v_i \in \mathcal{V}_m$ *[Hem99]*

1. Vertex v_i is locked, i.e. $d(v_i) := -|d(v_i)|$ if it belongs to
   - an element that is marked for further refinement
   - an edge ij and there is a vertex v_j such that $g(v_j) > g(v_i)$
   - a red element which should not be coarsened due to accuracy
2. Vertices are locked if blue elements would be created otherwise
   - If two nodes of an inner red triangle are locked, then lock third vertex
   - If the 'interior' node of a patch of four quadrilaterals is locked or if more than six nodes are locked, then lock all nodes of that patch

Result: Vertices $v_i \in \mathcal{V}_m$ with $d(v_i) \leq 0$ are locked;
all other nodes are 'free' and can be deleted.

$d(v_i) = 0$
$\forall \, v_i \in \mathcal{V}_0$

Goal: Each refinement operation is reversed in a single step
$\Rightarrow$ re-coarsening must start from the red elements

Locking: $d : \mathcal{V}_m \to \mathbb{Z}; \quad d(v_i) := g(v_i)\ \forall v_i \in \mathcal{V}_m$ [Hem99]

1. Vertex v_i is locked, i.e. $d(v_i) := -|d(v_i)|$ if it belongs to
   - an element that is marked for further refinement
   - an edge ij and there is a vertex v_j such that $g(v_j) > g(v_i)$
   - a red element which should not be coarsened due to accuracy
2. Vertices are locked if blue elements would be created otherwise
   - If two nodes of an inner red triangle are locked, then lock third vertex
   - If the 'interior' node of a patch of four quadrilaterals is locked or if more than six nodes are locked, then lock all nodes of that patch

Result: Vertices $v_i \in \mathcal{V}_m$ with $d(v_i) \leq 0$ are locked; all other nodes are 'free' and can be deleted.

$d(v_i) = 0 \ \forall v_i \in \mathcal{V}_0$

Goal: Each refinement operation is reversed in a single step
$\Rightarrow$ re-coarsening must start from the red elements

Locking: $d : \mathcal{V}_m \to \mathbb{Z}; \quad d(v_i) := g(v_i) \ \forall v_i \in \mathcal{V}_m$ [Hem99]

1. Vertex v_i is locked, i.e. $d(v_i) := -|d(v_i)|$ if it belongs to
   - an element that is marked for further refinement
   - an edge ij and there is a vertex v_j such that $g(v_j) > g(v_i)$
   - a red element which should not be coarsened due to accuracy
2. Vertices are locked if blue elements would be created otherwise
   - If two nodes of an inner red triangle are locked, then lock third vertex
   - If the 'interior' node of a patch of four quadrilaterals is locked or if more than six nodes are locked, then lock all nodes of that patch

Result: Vertices $v_i \in \mathcal{V}_m$ with $d(v_i) \leq 0$ are locked; all other nodes are 'free' and can be deleted.

$d(v_i) = 0 \ \forall v_i \in \mathcal{V}_0$

Goal: Each refinement operation is reversed in a single step
$\Rightarrow$ re-coarsening must start from the red elements

Locking: $d : \mathcal{V}_m \to \mathbb{Z}; \quad d(v_i) := g(v_i)\ \forall v_i \in \mathcal{V}_m$ [Hem99]

1. Vertex v_i is locked, i.e. $d(v_i) := -|d(v_i)|$ if it belongs to
   - an element that is marked for further refinement
   - an edge ij and there is a vertex v_j such that $g(v_j) > g(v_i)$
   - a red element which should not be coarsened due to accuracy
2. The following rules are applied in 2D:
   - If two nodes of an inner red triangle are locked, then lock third vertex
   - If the 'interior' node of a patch of four quadrilaterals is locked or if more than six nodes are locked, then lock all nodes of that patch

Result: Vertices $v_i \in \mathcal{V}_m$ with $d(v_i) \leq 0$ are locked;
all other nodes are 'free' and can be deleted.

$d(v_i) = 0$
$\forall v_i \in \mathcal{V}_0$

Re-coarsening algorithm

Goal: Each refinement operation is reversed in a single step
$\Rightarrow$ re-coarsening must start from the red elements

Locking: $d : \mathcal{V}_m \to \mathbb{Z}; \quad d(v_i) := g(v_i) \ \forall v_i \in \mathcal{V}_m$ [Hem99]

1. Vertex v_i is locked, i.e. $d(v_i) := -|d(v_i)|$ if it belongs to
   - an element that is marked for further refinement
   - an edge ij and there is a vertex v_j such that $g(v_j) > g(v_i)$
   - a red element which should not be coarsened due to accuracy
2. The following rules are applied in 2D:
   - If two nodes of an inner red triangle are locked, then lock third vertex
   - If the 'interior' node of a patch of four quadrilaterals is locked or if more than six nodes are locked, then lock all nodes of that patch

Result: Vertices $v_i \in \mathcal{V}_m$ with $d(v_i) \leq 0$ are locked; all other nodes are 'free' and can be deleted.

$d(v_i) = 0 \ \forall v_i \in \mathcal{V}_0$

Re-coarsening algorithm

Goal: Each refinement operation is reversed in a single step
$\Rightarrow$ re-coarsening must start from the red elements

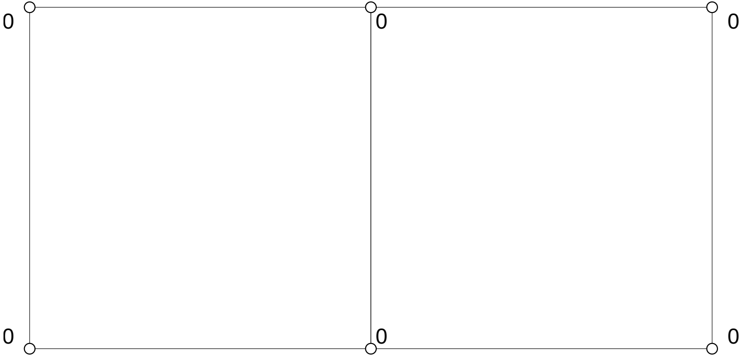
Locking: $d : \mathcal{V}_m \to \mathbb{Z}; \quad d(v_i) := g(v_i) \ \forall v_i \in \mathcal{V}_m$ [Hem99]

1. Vertex v_i is locked, i.e. $d(v_i) := -|d(v_i)|$ if it belongs to
   - an element that is marked for further refinement
   - an edge ij and there is a vertex v_j such that $g(v_j) > g(v_i)$
   - a red element which should not be coarsened due to accuracy
2. The following rules are applied in 2D:
   - If two nodes of an inner red triangle are locked, then lock third vertex
   - If the 'interior' node of a patch of four quadrilaterals is locked or if more than six nodes are locked, then lock all nodes of that patch

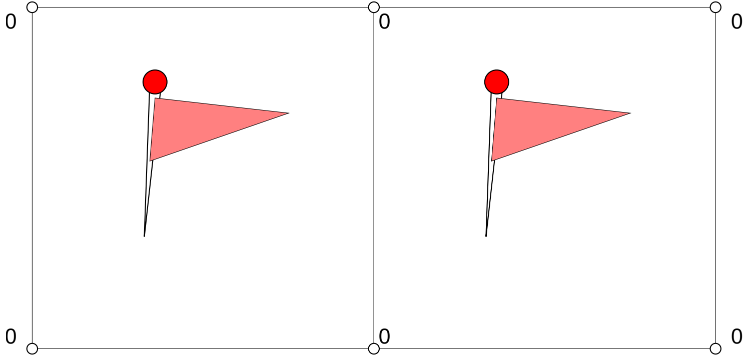
Result: Vertices $v_i \in \mathcal{V}_m$ with $d(v_i) \leq 0$ are locked; all other nodes are 'free' and can be deleted.

$d(v_i) = 0$
$\forall\, v_i \in \mathcal{V}_0$

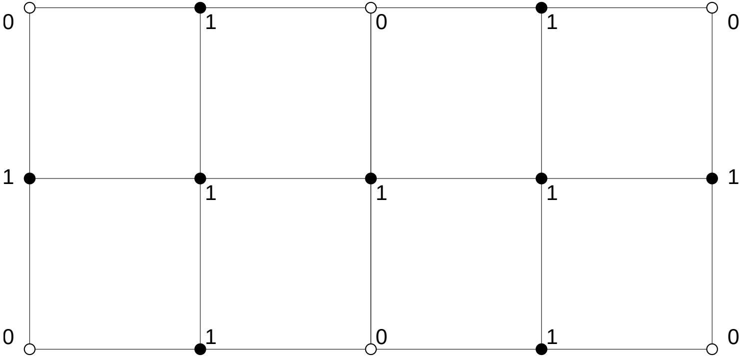
Refinement algorithm: initial grid



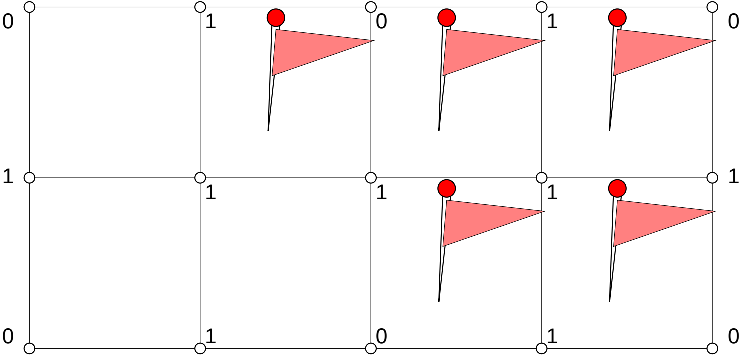
Refinement algorithm: mark elements for regular refinement



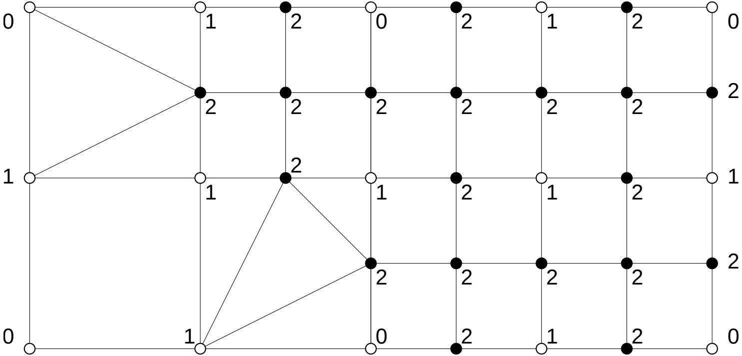
Refinement algorithm: perform regular refinement



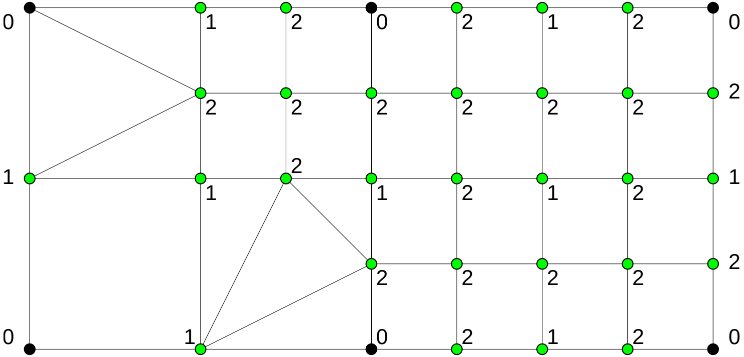
Refinement algorithm: mark elements for regular refinement



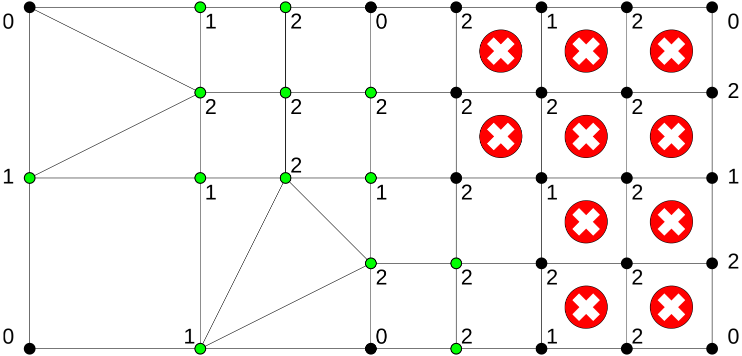
Refinement algorithm: perform regular refinement + transition cells



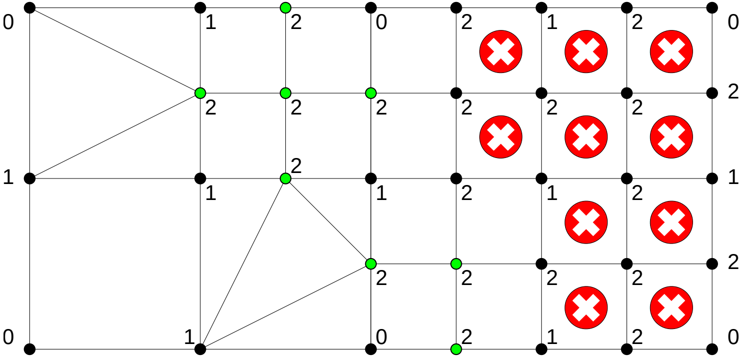
Re-coarsening algorithm: vertices from initial grid are locked



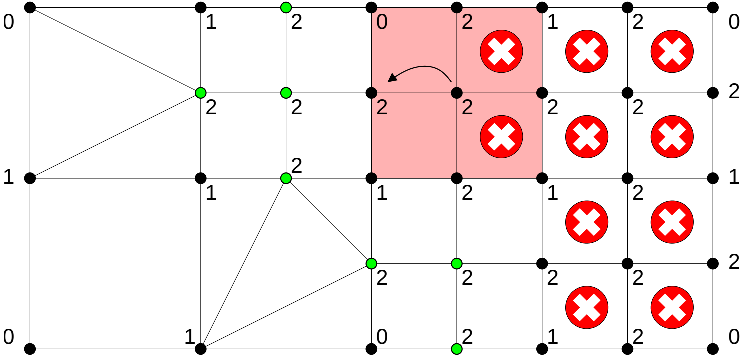
Re-coarsening algorithm: keep cells and lock connected vertices



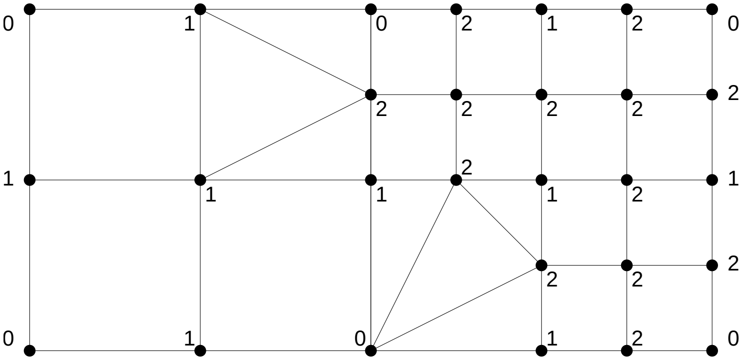
Re-coarsening algorithm: *"lock vertex if there are younger ones"*



Re-coarsening algorithm: *"prevent creation of blue elements"*



Re-coarsening algorithm: remove vertices and update elements



Identification of elements

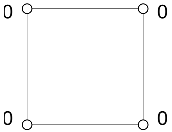
technische universität dortmund

Characterization of triangles/quadrilaterals in 2D

If the triangulation $\mathcal{T}_m = (\mathcal{E}_m, \mathcal{V}_m)$ is constructed from $\mathcal{T}_0 = (\mathcal{E}_0, \mathcal{V}_0)$ by performing m red-green refinement steps then each element $\Omega_k \in \mathcal{E}_m$ can be uniquely characterized by the generation number of the vertices.

Example: quadrilaterals

- If all vertices of Ω_k have zero age, then $\Omega_k \in \mathcal{E}_0$
- If three vertices of Ω_k have the same generation number, then the element is a red quadrilateral
- If exactly two consecutive vertices of Ω_k have largest generation number (in the cell), then the element is a green quadrilateral and the adjacent element Ω_l is the green neighbor

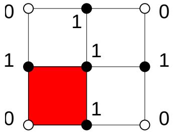


Characterization of triangles/quadrilaterals in 2D

If the triangulation $\mathcal{T}_m = (\mathcal{E}_m, \mathcal{V}_m)$ is constructed from $\mathcal{T}_0 = (\mathcal{E}_0, \mathcal{V}_0)$ by performing m red-green refinement steps then each element $\Omega_k \in \mathcal{E}_m$ can be uniquely characterized by the generation number of the vertices.

Example: quadrilaterals

- If all vertices of Ω_k have zero age, then $\Omega_k \in \mathcal{E}_0$
- If three vertices of Ω_k have the same generation number, then the element is a red quadrilateral
- If exactly two consecutive vertices of Ω_k have largest generation number (in the cell), then the element is a green quadrilateral and the adjacent element Ω_l is the green neighbor

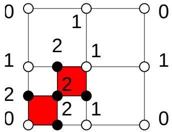


Characterization of triangles/quadrilaterals in 2D

If the triangulation $\mathcal{T}_m = (\mathcal{E}_m, \mathcal{V}_m)$ is constructed from $\mathcal{T}_0 = (\mathcal{E}_0, \mathcal{V}_0)$ by performing m red-green refinement steps then each element $\Omega_k \in \mathcal{E}_m$ can be uniquely characterized by the generation number of the vertices.

Example: quadrilaterals

- If all vertices of Ω_k have zero age, then $\Omega_k \in \mathcal{E}_0$
- If three vertices of Ω_k have the same generation number, then the element is a red quadrilateral
- If exactly two consecutive vertices of Ω_k have largest generation number (in the cell), then the element is a green quadrilateral and the adjacent element Ω_l is the green neighbor

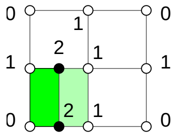


Characterization of triangles/quadrilaterals in 2D

If the triangulation $\mathcal{T}_m = (\mathcal{E}_m, \mathcal{V}_m)$ is constructed from $\mathcal{T}_0 = (\mathcal{E}_0, \mathcal{V}_0)$ by performing m red-green refinement steps then each element $\Omega_k \in \mathcal{E}_m$ can be uniquely characterized by the generation number of the vertices.

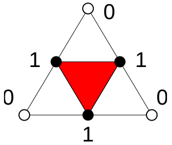
Example: quadrilaterals

- If all vertices of Ω_k have zero age, then $\Omega_k \in \mathcal{E}_0$
- If three vertices of Ω_k have the same generation number, then the element is a red quadrilateral
- If exactly two consecutive vertices of Ω_k have largest generation number (in the cell), then the element is a green quadrilateral and the adjacent element Ω_l is the green neighbor



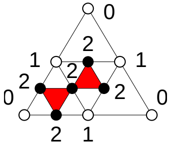
Example: triangles

- If all vertices of Ω_k have the same generation number, then the element is an inner red triangle
- If exactly two vertices of Ω_k have largest generation number (in the cell), then the element is an outer red triangle provided that the adjacent element is an inner red triangle. Otherwise, it is a green triangle
- If exactly one vertex v_i of Ω_k has largest generation number (in the cell), then the element is a green triangle and three difference cases are possible:
  - Adjacent element Ω_l has one vertex with largest generation number and it is the common node v_i
  - There exists **one** neighboring green triangle with exactly two vertices with largest generation number
  - There exist **two** neighboring green triangles sharing the common node v_i with largest generation number



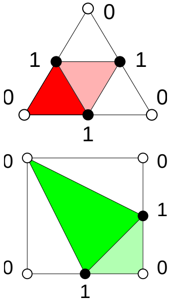
Example: triangles

- If all vertices of Ω_k have the same generation number, then the element is an inner red triangle
- If exactly two vertices of Ω_k have largest generation number (in the cell), then the element is an outer red triangle provided that the adjacent element is an inner red triangle. Otherwise, it is a green triangle
- If exactly one vertex v_i of Ω_k has largest generation number (in the cell), then the element is a green triangle and three difference cases are possible:
  - Adjacent element Ω_l has one vertex with largest generation number and it is the common node v_i
  - There exists **one** neighboring green triangle with exactly two vertices with largest generation number
  - There exist **two** neighboring green triangles sharing the common node v_i with largest generation number



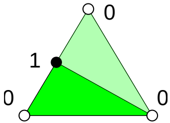
Example: triangles

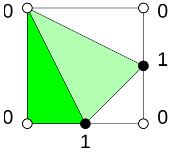
- If all vertices of Ω_k have the same generation number, then the element is an inner red triangle
- If exactly two vertices of Ω_k have largest generation number (in the cell), then the element is an outer red triangle provided that the adjacent element is an inner red triangle. Otherwise, it is a green triangle
- If exactly one vertex v_i of Ω_k has largest generation number (in the cell), then the element is a green triangle and three difference cases are possible:
  - Adjacent element Ω_l has one vertex with largest generation number and it is the common node v_i
  - There exists **one** neighboring green triangle with exactly two vertices with largest generation number
  - There exist **two** neighboring green triangles sharing the common node v_i with largest generation number

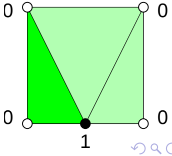


Example: triangles

- If all vertices of Ω_k have the same generation number, then the element is an inner red triangle
- If exactly two vertices of Ω_k have largest generation number (in the cell), then the element is an outer red triangle provided that the adjacent element is an inner red triangle. Otherwise, it is a green triangle
- If exactly one vertex v_i of Ω_k has largest generation number (in the cell), then the element is a green triangle and three difference cases are possible:
  - Adjacent element Ω_l has one vertex with largest generation number and it is the common node v_i
  - There exists **one** neighboring green triangle with exactly two vertices with largest generation number
  - There exist **two** neighboring green triangles sharing the common node v_i with largest generation number

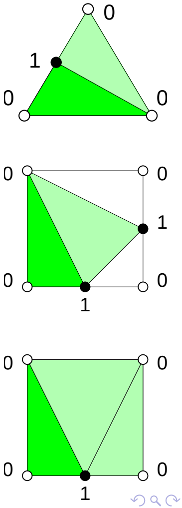






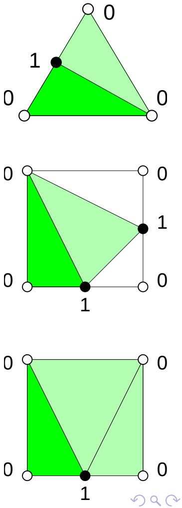
Example: triangles

- If all vertices of Ω_k have the same generation number, then the element is an inner red triangle
- If exactly two vertices of Ω_k have largest generation number (in the cell), then the element is an outer red triangle provided that the adjacent element is an inner red triangle. Otherwise, it is a green triangle
- If exactly one vertex v_i of Ω_k has largest generation number (in the cell), then the element is a green triangle and three difference cases are possible:
    - Adjacent element Ω_l has one vertex with largest generation number and it is the common node v_i
    - There exists **one** neighboring green triangle with exactly two vertices with largest generation number
    - There exist **two** neighboring green triangles sharing the common node v_i with largest generation number



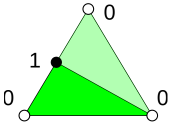
Example: triangles

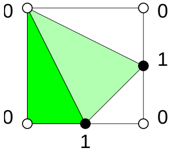
- If all vertices of Ω_k have the same generation number, then the element is an inner red triangle
- If exactly two vertices of Ω_k have largest generation number (in the cell), then the element is an outer red triangle provided that the adjacent element is an inner red triangle. Otherwise, it is a green triangle
- If exactly one vertex v_i of Ω_k has largest generation number (in the cell), then the element is a green triangle and three difference cases are possible:
  - Adjacent element Ω_l has one vertex with largest generation number and it is the common node v_i
  - There exists **one** neighboring green triangle with exactly two vertices with largest generation number
  - There exist **two** neighboring green triangles sharing the common node v_i with largest generation number

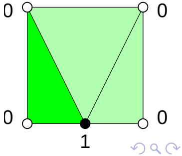


Example: triangles

- If all vertices of Ω_k have the same generation number, then the element is an inner red triangle
- If exactly two vertices of Ω_k have largest generation number (in the cell), then the element is an outer red triangle provided that the adjacent element is an inner red triangle. Otherwise, it is a green triangle
- If exactly one vertex v_i of Ω_k has largest generation number (in the cell), then the element is a green triangle and three difference cases are possible:
  - Adjacent element Ω_l has one vertex with largest generation number and it is the common node v_i
  - There exists **one** neighboring green triangle with exactly two vertices with largest generation number
  - There exist **two** neighboring green triangles sharing the common node v_i with largest generation number







Idea I: State function $s : \mathcal{E}_m \to \mathbb{Z}$ (in MSB representation)

- Set `Bit[0]` to 1 for quadrilateral, otherwise set it to zero
- Set `Bit[k=1..4]` to 1 if both endpoints of edge k have same age
- If no two endpoints have same age, then find local position k of the youngest vertex, set `Bit[k]` to 1 and negate the state

Idea II: Define local ordering strategy within each element *a priori*

element characterization	element state
triangle/quadrilateral from $\mathcal{T}_0$	0/1
green quadrilateral	3, 5, 9, 11, 17, 21
red quadrilateral	7, 13, 19, 25
inner red triangle	14
'other' triangle	2, 4, 8
green triangle	-8, -4, -2

Idea I: State function $s : \mathcal{E}_m \rightarrow \mathbb{Z}$ (in MSB representation)

- Set `Bit[0]` to 1 for quadrilateral, otherwise set it to zero
- Set `Bit[k=1..4]` to 1 if both endpoints of edge k have same age
- If no two endpoints have same age, then find local position k of the youngest vertex, set `Bit[k]` to 1 and negate the state

Idea II: Define local ordering strategy within each element *a priori*

element characterization	element state
triangle/quadrilateral from $\mathcal{T}_0$	0/1
green quadrilateral	3, 5, 9, 11, 17, 21
red quadrilateral	7, 13, 19, 25
inner red triangle	14
'other' triangle	2, 4, 8
green triangle	-8, -4, -2

Idea I: State function $s : \mathcal{E}_m \rightarrow \mathbb{Z}$ (in MSB representation)

- Set `Bit[0]` to 1 for quadrilateral, otherwise set it to zero
- Set `Bit[k=1..4]` to 1 if both endpoints of edge k have same age
- If no two endpoints have same age, then find local position k of the youngest vertex, set `Bit[k]` to 1 and negate the state

Idea II: Define local ordering strategy within each element *a priori*

element characterization	element state
triangle/quadrilateral from $\mathcal{T}_0$	0/1
green quadrilateral	3, 5, 9, 11, 17, 21
red quadrilateral	7, 13, 19, 25
inner red triangle	14
'other' triangle	2, 4, 8
green triangle	-8, -4, -2

Idea I: State function $s : \mathcal{E}_m \rightarrow \mathbb{Z}$ (in MSB representation)

- Set `Bit[0]` to 1 for quadrilateral, otherwise set it to zero
- Set `Bit[k=1..4]` to 1 if both endpoints of edge k have same age
- If no two endpoints have same age, then find local position k of the youngest vertex, set `Bit[k]` to 1 and negate the state

Idea II: Define local ordering strategy within each element *a priori*

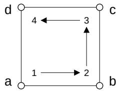
element characterization	element state
triangle/quadrilateral from $\mathcal{T}_0$	0/1
green quadrilateral	3, 5, 9, 11, 17, 21
red quadrilateral	7, 13, 19, 25
inner red triangle	14
'other' triangle	2, 4, 8
green triangle	-8, -4, -2

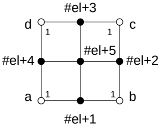
Idea I: State function $s : \mathcal{E}_m \rightarrow \mathbb{Z}$ (in MSB representation)

- Set `Bit[0]` to 1 for quadrilateral, otherwise set it to zero
- Set `Bit[k=1..4]` to 1 if both endpoints of edge k have same age
- If no two endpoints have same age, then find local position k of the youngest vertex, set `Bit[k]` to 1 and negate the state

Idea II: Define local ordering strategy within each element *a priori*

element characterization	element state
triangle/quadrilateral from $\mathcal{T}_0$	0/1
green quadrilateral	5, 21
red quadrilateral	13
inner red triangle	14
'other' triangle	4
green triangle	-8, -4, -2





Convection in $\Omega = (0,1)^2$

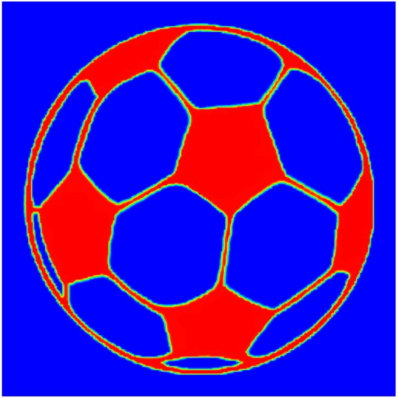
$$\frac{\partial u}{\partial t} + \nabla \cdot (\mathbf{v}u) = 0$$

with velocity $\mathbf{v}(t)$, $t \in (0, T)$

$$\begin{aligned} v_x &= \sin^2(\pi x)\sin(2\pi y)g(t) \\ v_y &= -\sin^2(\pi y)\sin(2\pi x)g(t) \\ g(t) &= \cos(\pi t/T), \quad T = 1.5 \end{aligned}$$

32×32 coarse grid + 3 ref. steps

29.345 cells, 26.189 vertices



Time 0.0

Convection in $\Omega = (0,1)^2$

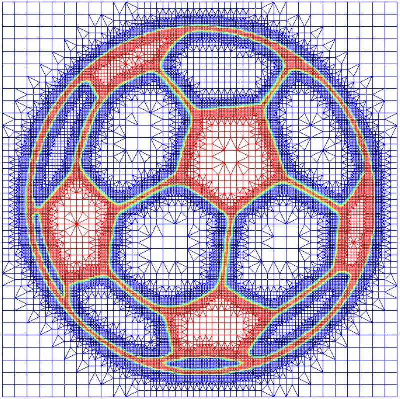
$$\frac{\partial u}{\partial t} + \nabla \cdot (\mathbf{v}u) = 0$$

with velocity $\mathbf{v}(t)$, $t \in (0, T)$

$$\begin{aligned} v_x &= \sin^2(\pi x)\sin(2\pi y)g(t) \\ v_y &= -\sin^2(\pi y)\sin(2\pi x)g(t) \\ g(t) &= \cos(\pi t/T), \quad T = 1.5 \end{aligned}$$

32×32 coarse grid + 3 ref. steps

29.345 cells, 26.189 vertices



Time 0.0

Convection in $\Omega = (0,1)^2$

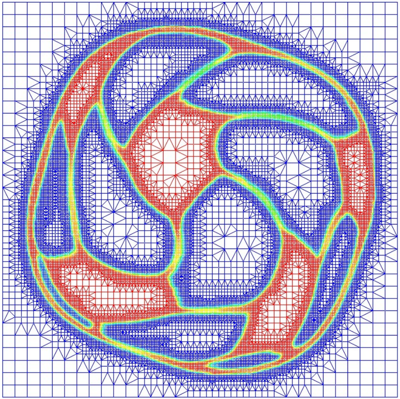
$$\frac{\partial u}{\partial t} + \nabla \cdot (\mathbf{v}u) = 0$$

with velocity $\mathbf{v}(t)$, $t \in (0, T)$

$$\begin{aligned} v_x &= \sin^2(\pi x)\sin(2\pi y)g(t) \\ v_y &= -\sin^2(\pi y)\sin(2\pi x)g(t) \\ g(t) &= \cos(\pi t/T), \quad T = 1.5 \end{aligned}$$

32×32 coarse grid + 3 ref. steps

30.798 cells, 27.452 vertices



Time 0.25

Convection in $\Omega = (0,1)^2$

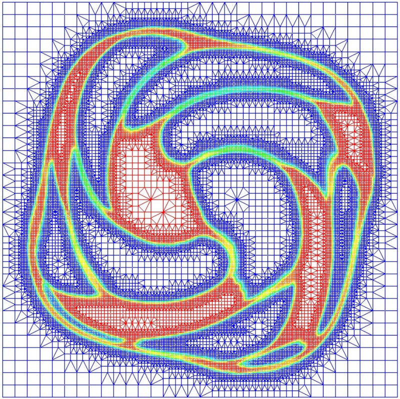
$$\frac{\partial u}{\partial t} + \nabla \cdot (\mathbf{v}u) = 0$$

with velocity $\mathbf{v}(t)$, $t \in (0, T)$

$$\begin{aligned} v_x &= \sin^2(\pi x)\sin(2\pi y)g(t) \\ v_y &= -\sin^2(\pi y)\sin(2\pi x)g(t) \\ g(t) &= \cos(\pi t/T), \quad T = 1.5 \end{aligned}$$

32×32 coarse grid + 3 ref. steps

33.080 cells, 29.516 vertices



Time 0.5

Example: *Swirling flow*

Convection in $\Omega = (0,1)^2$

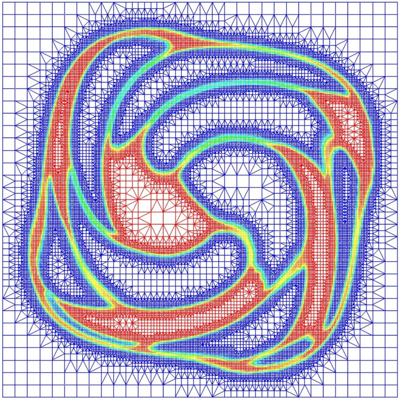
$$\frac{\partial u}{\partial t} + \nabla \cdot (\mathbf{v} u) = 0$$

with velocity $\mathbf{v}(t)$, $t \in (0, T)$

$$\begin{aligned} v_x &= \sin^2(\pi x)\sin(2\pi y) g(t) \\ v_y &= -\sin^2(\pi y)\sin(2\pi x) g(t) \\ g(t) &= \cos(\pi t/T), \quad T = 1.5 \end{aligned}$$

32×32 coarse grid + 3 ref. steps

33.371 cells, 29.817 vertices



Time 0.75

Convection in $\Omega = (0,1)^2$

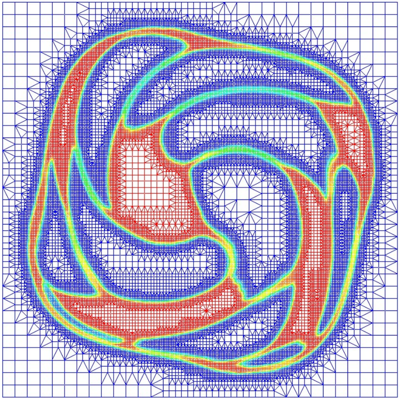
$$\frac{\partial u}{\partial t} + \nabla \cdot (\mathbf{v}u) = 0$$

with velocity $\mathbf{v}(t)$, $t \in (0, T)$

$$\begin{aligned} v_x &= \sin^2(\pi x)\sin(2\pi y)g(t) \\ v_y &= -\sin^2(\pi y)\sin(2\pi x)g(t) \\ g(t) &= \cos(\pi t/T), \quad T = 1.5 \end{aligned}$$

32×32 coarse grid + 3 ref. steps

33.639 cells, 30.089 vertices



Time 1.0

Example: *Swirling flow*

Convection in $\Omega = (0,1)^2$

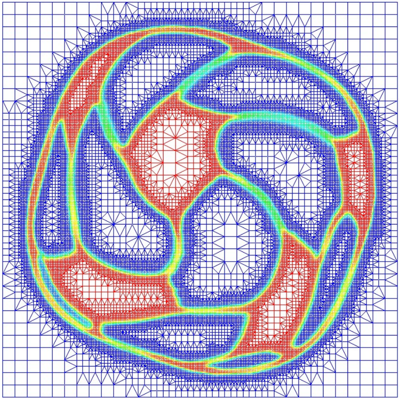
$$\frac{\partial u}{\partial t} + \nabla \cdot (\mathbf{v}u) = 0$$

with velocity $\mathbf{v}(t)$, $t \in (0, T)$

$$\begin{aligned} v_x &= \sin^2(\pi x)\sin(2\pi y)g(t) \\ v_y &= -\sin^2(\pi y)\sin(2\pi x)g(t) \\ g(t) &= \cos(\pi t/T), \quad T = 1.5 \end{aligned}$$

32×32 coarse grid + 3 ref. steps

32.456 cells, 29.177 vertices



Time 1.25

Example: *Swirling flow*

Convection in $\Omega = (0,1)^2$

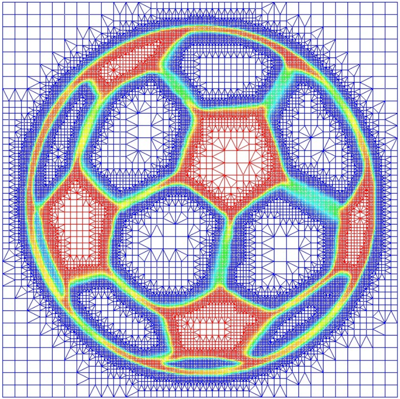
$$\frac{\partial u}{\partial t} + \nabla \cdot (\mathbf{v}u) = 0$$

with velocity $\mathbf{v}(t)$, $t \in (0, T)$

$$\begin{aligned} v_x &= \sin^2(\pi x)\sin(2\pi y)g(t) \\ v_y &= -\sin^2(\pi y)\sin(2\pi x)g(t) \\ g(t) &= \cos(\pi t/T), \quad T = 1.5 \end{aligned}$$

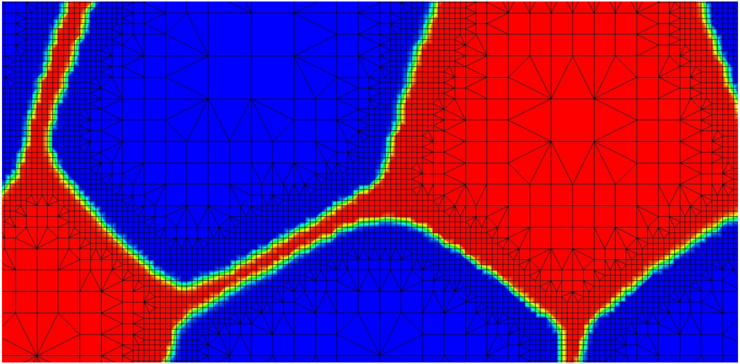
32×32 coarse grid + 3 ref. steps

30.774 cells, 27.691 vertices

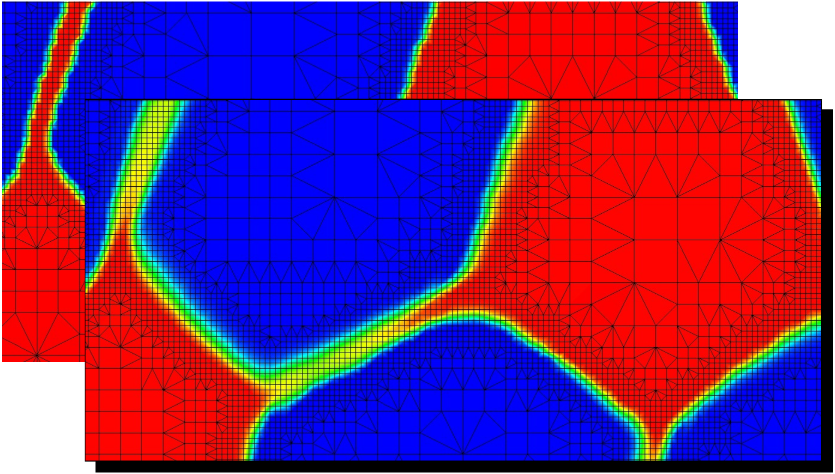


Time 1.5

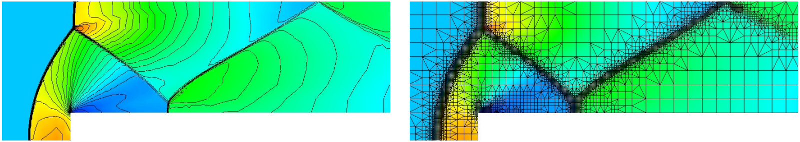
Initial profile



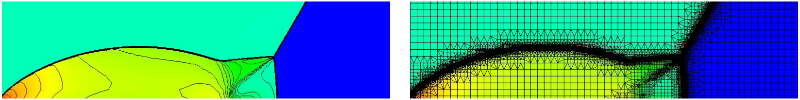
Initial profile and final solution computed by semi-implicit FEM-FCT



Forward facing step, $t = 4.0$, 30 contour lines

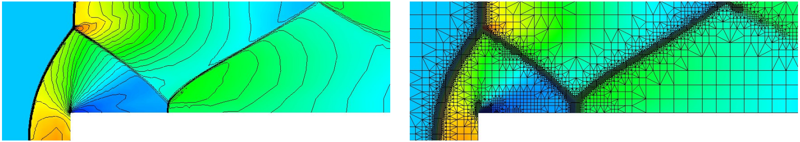


Double Mach reflection, $t = 0.2$, 30 contour lines

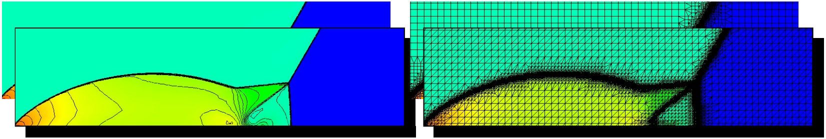


- solutions are computed by algebraic flux correction scheme; *Kuzmin*
- Crank-Nicolson time stepping approach is adopted

Forward facing step, $t = 4.0$, 30 contour lines



Double Mach reflection, $t = 0.2$, 30 contour lines



- solutions are computed by algebraic flux correction scheme; *Kuzmin*
- Crank-Nicolson time stepping approach is adopted

Conclusions and Outlook

Conclusions

- hierarchical grid adaptivity can be based on red-green strategy
- grid re-coarsening can be based on locking of vertices
- grid genealogy can be recovered from nodal generation numbers
- cells can be easily identified from element state function

Future work

- Combine local grid adaptation with structured grids (HPC)
- Implement refinement/re-coarsening algorithms in 3D
- Improve error indicators/estimators also for time-dependent flows

Conclusions

- hierarchical grid adaptivity can be based on red-green strategy
- grid re-coarsening can be based on locking of vertices
- grid genealogy can be recovered from nodal generation numbers
- cells can be easily identified from element state function

Future work

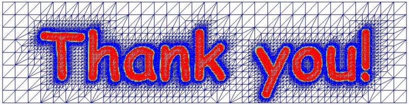
- Combine local grid adaptation with structured grids (HPC)
- Implement refinement/re-coarsening algorithms in 3D
- Improve error indicators/estimators also for time-dependent flows

Conclusions

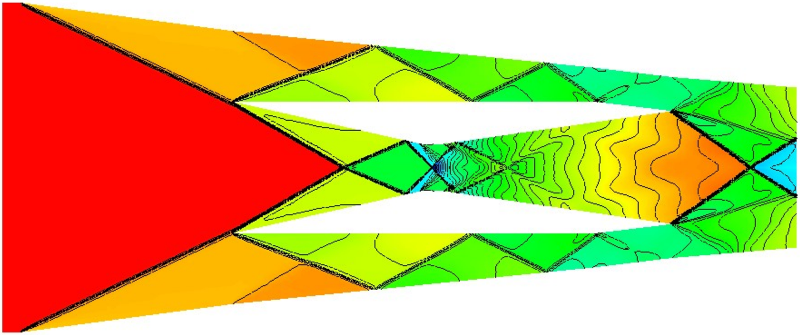
- hierarchical grid adaptivity can be based on red-green strategy
- grid re-coarsening can be based on locking of vertices
- grid genealogy can be recovered from nodal generation numbers
- cells can be easily identified from element state function

Future work

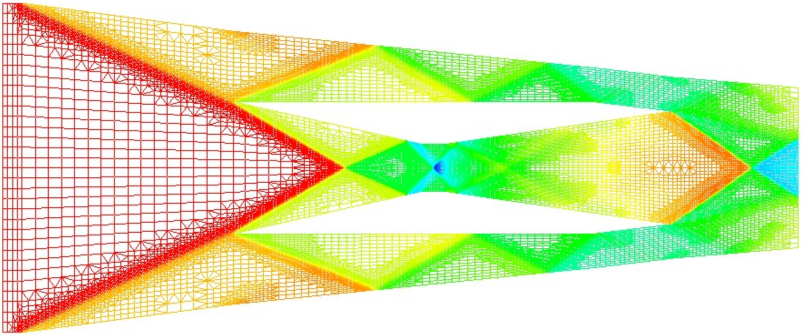
- Combine local grid adaptation with structured grids (HPC)
- Implement refinement/re-coarsening algorithms in 3D
- Improve error indicators/estimators also for time-dependent flows



Scramjet inlet at $Ma = 3.0$, $\alpha = 0^\circ$



Scramjet inlet at $Ma = 3.0$, $\alpha = 0^\circ$



Scramjet inlet at $Ma = 3.0$, $\alpha = 0^\circ$