Status of anisotropic refinement in deal.II

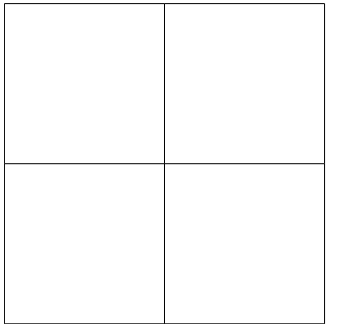
Ralf Hartmann

Numerical Methods
Institute of Aerodynamics and Flow Technology
German Aerospace Center (DLR), Braunschweig

Anisotropic refinement: Motivation

- Resolution of anisotropic solution features like: shocks, shear layers, boundary layers, etc.
- Compressible flows: about 50% of all elements are located in the boundary layer

Anisotropic refinement: Motivation

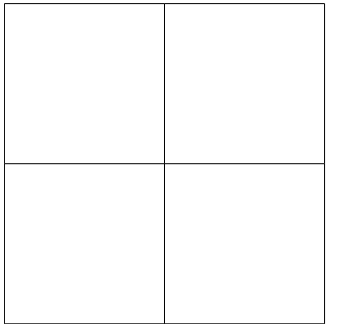


- Resolution of anisotropic solution features like: shocks, shear layers, boundary layers, etc.
- Compressible flows: about 50% of all elements are located in the boundary layer

Assume: wanted size of h_y at boundary: $h_y = 0.005$

Ralf Hartmann, Institut für Aerodynamik und Strömungstechnik

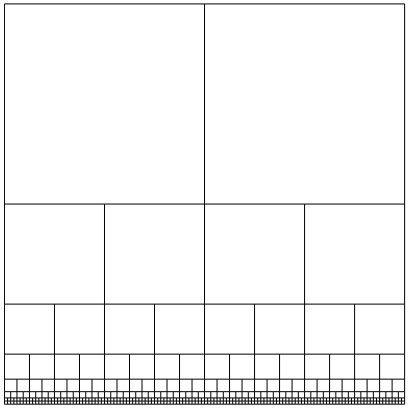
Anisotropic refinement: Motivation



- Resolution of anisotropic solution features like: shocks, shear layers, boundary layers, etc.
- Compressible flows: about 50% of all elements are located in the boundary layer

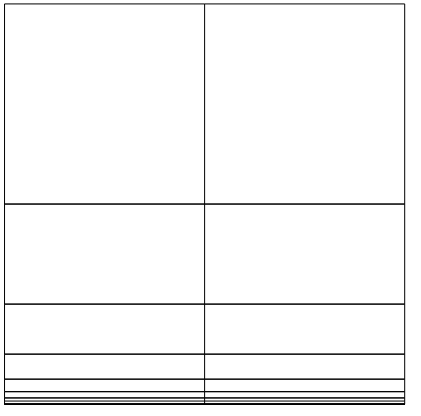
Assume: wanted size of h_y at boundary: $h_y = 0.005$

isotropic



766 cells

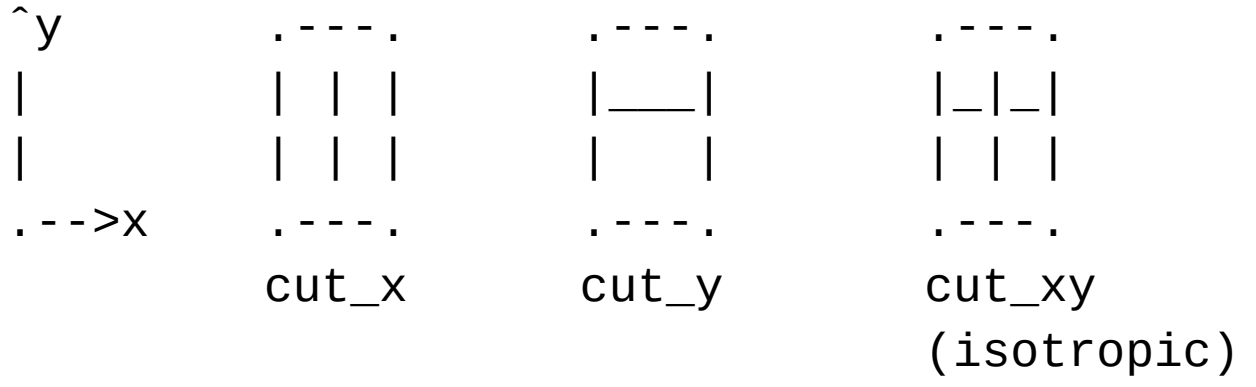
anisotropic



18 cells (aspect ratio: 128)

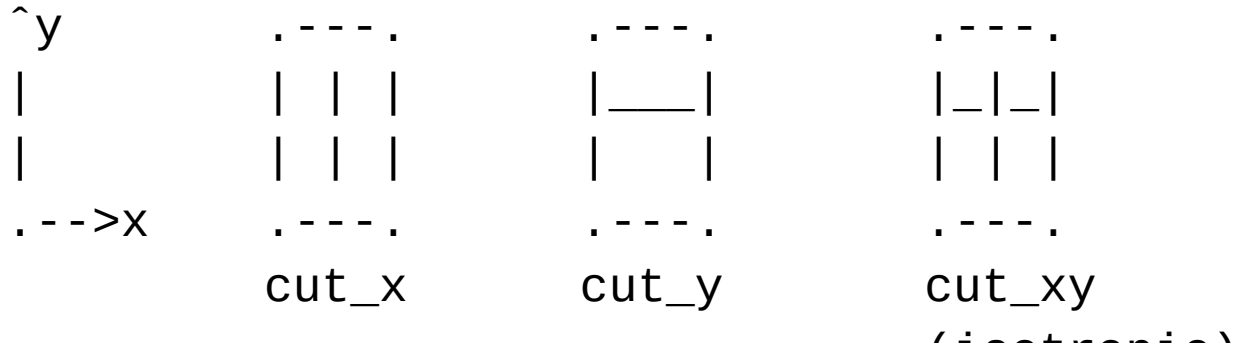
Typical aspect ratios in aerodynamics: $10^4 - 10^6$

Refinement cases in 2d



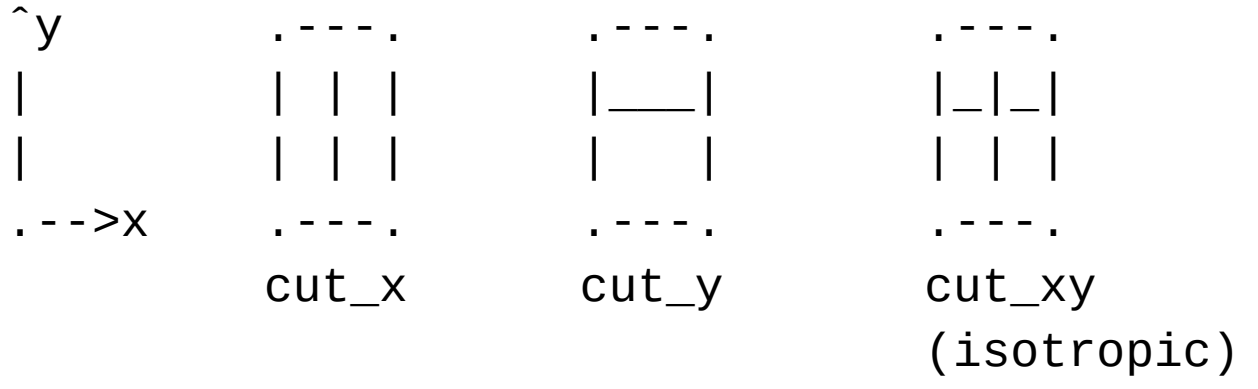
Ralf Hartmann, Institut für Aerodynamik und Strömungstechnik

Refinement cases in 2d




```
namespace RefineCase
{
  enum Type
  {
        no_refinement = 0,
        cut_x         = 1,
        cut_y         = 2,
        cut_xy        = 3
  };
}
```

Refinement cases in 2d



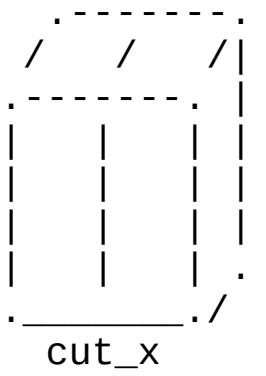

```
namespace RefineCase
{
  enum Type
  {
        no_refinement = 0,
        cut_x         = 1,
        cut_y         = 2,
        cut_xy        = 3
  };
}
```

Note, that e.g.

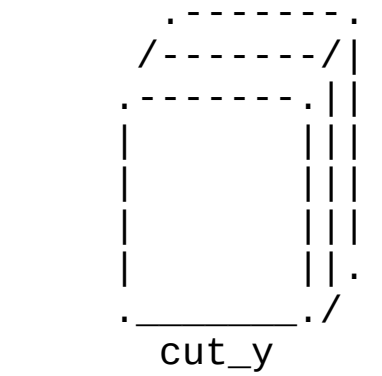
```
cut_xy == cut_x | cut_y
```


Ralf Hartmann, Institut für Aerodynamik und Strömungstechnik

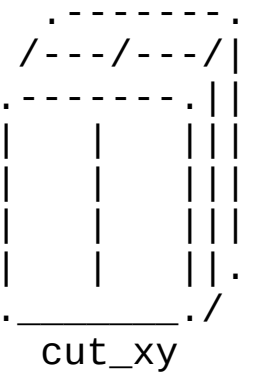
Refinement cases in 3d



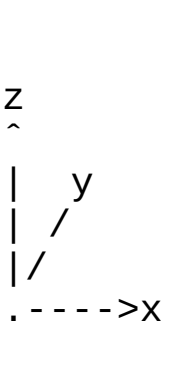
cut_x

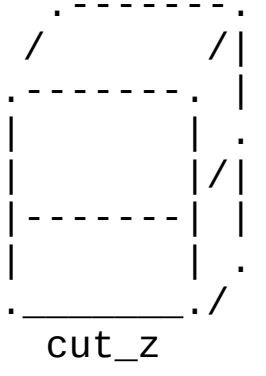


cut_y

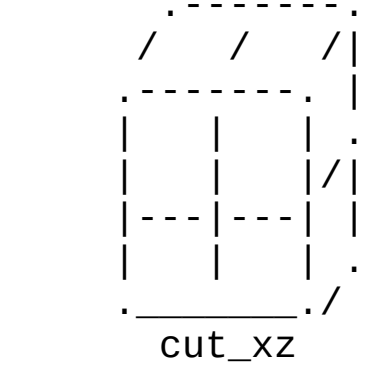


cut_xy

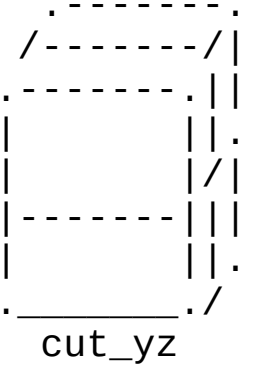




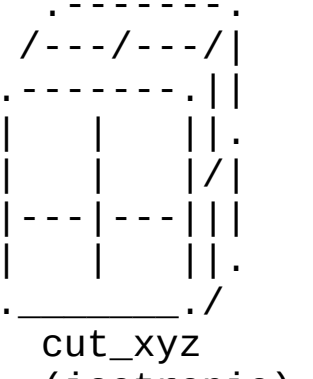
cut_z



cut_xz



cut_yz



cut_xyz
(isotropic)

Ralf Hartmann, Institut für Aerodynamik und Strömungstechnik

Refinement cases in 3d

```
namespace RefineCase
{
  enum Type
  {
        no_refinement = 0,
        cut_x         = 1,
        cut_y         = 2,
        cut_xy        = 3,
        cut_z         = 4,
        cut_xz        = 5,
        cut_yz        = 6,
        cut_xyz       = 7
  };
}
```

Refinement cases in 3d

```
namespace RefineCase
{
  enum Type
  {
        no_refinement = 0,
        cut_x         = 1,
        cut_y         = 2,
        cut_xy        = 3,
        cut_z         = 4,
        cut_xz        = 5,
        cut_yz        = 6,
        cut_xyz       = 7
  };
}
```

note, that e.g.

```
  cut_yz  == cut_y  | cut_z
  cut_xyz == cut_yz | cut_x
```

Local refinement in deal.II

Isotropic refinement (current state):

```
cell->set_refine_flag();
[....]
tria.execute_coarsening_and_refinement();
```

Anisotropic refinement:

```
cell->set_refine_flag(RefineCase::cut_x);
```

Identical calls for isotropic refinement (for 2d):

```
cell->set_refine_flag(RefineCase::cut_xy);
cell->set_refine_flag(GeometryInfo<dim>::isotropic_refinement);
cell->set_refine_flag();                              // backward compatibility!!!
```

Implementation:

```
template <int dim>
class CellAccessor {
  public:
    void set_refine_flag (RefineCase::Type ref_case=
                          GeometryInfo<dim>::isotropic_refinement) const;
};
```

Changes required in user codes

Incompatible changes:

- **Changes in `deal.II` which force user codes to be changed (e.g. to make them compile or to recover old functionality).**

Changes for new functionality:

- **Changes required in user code for gaining a new functionality (e.g. anisotropic refinement)**

Incompatible changes (1)

None, up to now! E.g. step-12 (isotropic refinement) runs without any change!

The interfaces of some internal functions have been extended. E.g.

```
template <int dim>
unsigned int
GeometryInfo<dim>::child_cell_on_face (const unsigned int face,
                                       const unsigned int subface,
                                       const bool face_orientation);
```

has been extended to

```
template <int dim>
unsigned int
GeometryInfo<dim>::child_cell_on_face (const RefineCase::Type ref_case,
                                       const unsigned int face,
                                       const unsigned int subface,
                                       const bool face_orientation);
```

However, this function should not be used in user codes anyway. The right function to be used is e.g. `cell->neighbor_child_on_subface(...)`.

Incompatible changes (2)

Variable names:

- `GeometryInfo<dim>::children_per_cell` **and**
- `GeometryInfo<dim>::subfaces_per_face`

Incompatible changes (2)

Variable names:

- `GeometryInfo<dim>::children_per_cell` **and**
- `GeometryInfo<dim>::subfaces_per_face`

Three possibilities:

- **keep them unchanged, or**
- **change them to** `GeometryInfo<dim>::max_children_per_cell` **and** `GeometryInfo<dim>::max_children_per_face` **(incompatible change!), or**
- **change them to the new names and keep the old ones for backward compatibilities (at least for a while)**

Changes required for new functionality (anisotropic ref.)

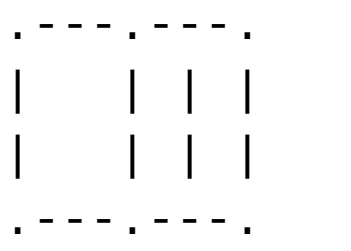
replace ...	by ...
`GeometryInfo<dim>::children_per_cell`	`cell->n_children()`
`GeometryInfo<dim>::subfaces_per_face`	`face->n_children()`

In the context of face integrals (`FEFaceValues`, `FESubfaceValues`)
replace `cell->neighbor(face_no)->has_children()`
by `cell->face(face_no)->has_children()`

E.g. in step-12 (assemble_system2) only one line of code must be added

```
<         if (neighbor->level() == cell->level() &&
<             neighbor->index() > cell->index())
---
>         if ((neighbor->level() == cell->level() &&
>              neighbor->index() > cell->index()) ||
>             neighbor->level() > cell->level())
```

to include the case



Ralf Hartmann, Institut für Aerodynamik und Strömungstechnik

Current implementation

Situation: `cell` **(on** `level`**), neighboring cell** `n` **has children** `c0, c1` **(on** `level+1`**).**
Problem: faces of `cell` **are on** `level`**, faces of** `c0` **are on** `level+1`**,**
line `l` **is on** `level` **and** `level+1` **(!!)**

Currently, faces in `deal.II` **are stored in** `TriangulationLevel`**. Therefore, line** `l` **on** `level` **must be duplicated to be stored also on** `level+1`**.**

Ralf Hartmann, Institut für Aerodynamik und Strömungstechnik

Current implementation

Situation: `cell` **(on** `level`**), neighboring cell** `n` **has children** `c0, c1` **(on** `level+1`**).**
Problem: faces of `cell` **are on** `level`**, faces of** `c0` **are on** `level+1`**,**
line `l` **is on** `level` **and** `level+1` **(!!)**

Currently, faces in `deal.II` **are stored in** `TriangulationLevel`**. Therefore, line** `l` **on** `level` **must be duplicated to be stored also on** `level+1`**.**

Problems of duplicated lines:

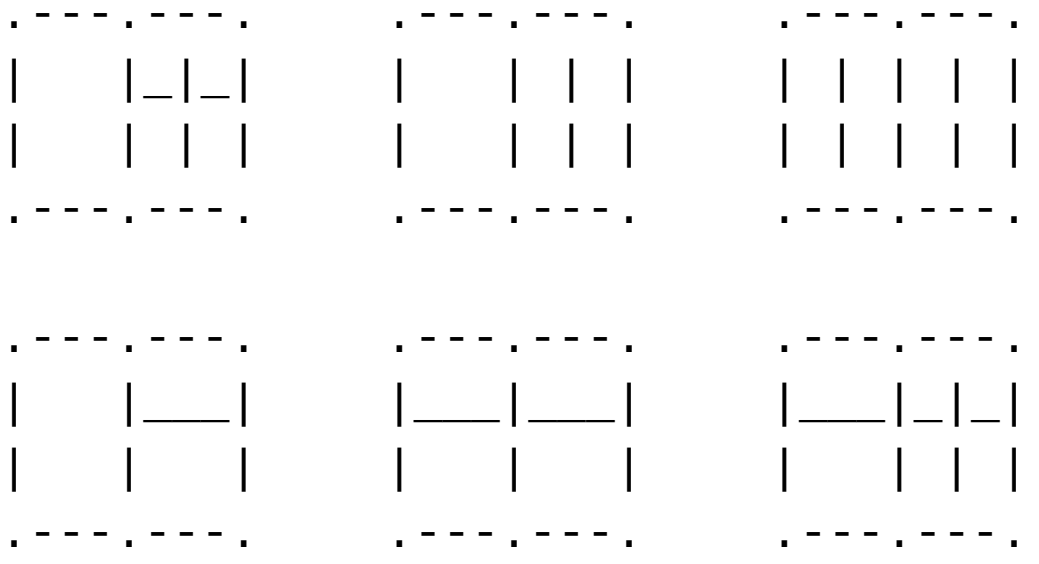
- **Multiple storage of (same) data on two (max.** `n_levels()`**) levels**
- **Restriction on refinement cases (see next slide)**

Therefore, association of faces with levels is a design flaw!

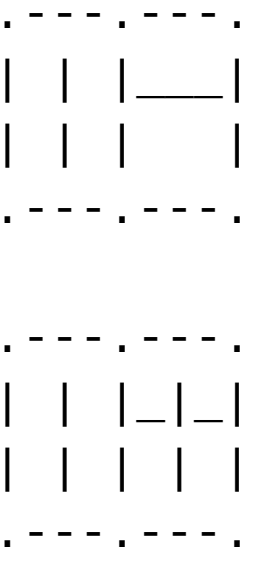
Duplicated faces: restriction of refinement cases

Refinement cases of two neighboring cells

Can be realized



Can NOT be realized



Ralf Hartmann, Institut für Aerodynamik und Strömungstechnik

Problem of non-realizable refinement cases

Can not be realized:

Situation: `cell` **(on** `level`**) with children** `c0, c1`**,**
neighboring cell `n` **with children** `n0,...,n3`**.**
`cell` **and** `n` **on** `level`**, children on** `level+1`**.**

Problem:

line `l=cell->line(1)` **on** `level` **with** `l->has_children()==true`**,**
duplicate line `dl=c1->line(1)` **on** `level+1`**,**
BUT: `dl->has_children()!=true`

Ralf Hartmann, Institut für Aerodynamik und Strömungstechnik

Change of internal data structures

Separate faces from `TriangulationLevel`.

- **This change does not effect the current functionality or interface of `deal.II`**
- **This change allows anisotropic refinement without duplicate faces**
- **Removes restriction on refinement cases**

Ralf Hartmann, Institut für Aerodynamik und Strömungstechnik

Sketch of old internal data structure

```
TriangulationLevel<0> {...};

TriangulationLevel<1>: public TriangulationLevel<0> {
  LinesData lines;
  void reserve_space();
  ...
};

TriangulationLevel<2>: public TriangulationLevel<1> {
  QuadsData quads;
  void reserve_space();
  ...
};

TriangulationLevel<3>: public TriangulationLevel<2> {
  HexesData hexes;
  void reserve_space();
  ...
};

template <int dim>
class Triangulation {
  private:
    std::vector<TriangulationLevel<dim>* > levels;
    ...
};
```

Sketch of new internal data structure

```
template<typename G>
class Objects {
  vector<G> cells;
  vector<bool> used;
  void reserve_space(); ... };

class ObjectsHex: public Objects<Hexahedron> {
  vector<bool> face_orientations; };

class TriangulationLevel<0> {...};           // unchanged
class TriangulationLevel<1>: public Objects<Line>,
                             public TriangulationLevel<0> {...};
class TriangulationLevel<2>: public Objects<Quad>,
                             public TriangulationLevel<0> {...};
class TriangulationLevel<3>: public ObjectsHex,
                             public TriangulationLevel<0> {...};

class TriangulationFaces<1> {};              // no data
class TriangulationFaces<2>: public Objects<Line> {...};
class TriangulationFaces<3>: public Objects<Quads> {...};

template <int dim>
class Triangulation {
  private:
    std::vector<TriangulationLevel<dim>* > levels;
    TriangulationFaces<dim> faces;           // new!
};
```