# The DUNE PrismGrid Module

CHRISTOPH GERSBACHER

DUNE User Meeting

October 8, 2010 Stuttgart

Albert-Ludwigs-Universität Freiburg

UNI FREIBURG

## Meta grids developed in Freiburg

- GeometryGrid (M. Nolte)
- IdGrid (M. Nolte)
- ParallelGrid (R. Klöfkorn)
- PrismGrid

## Observations

- Easy to implement (?)
- Performance loss

## Meta grids developed in Freiburg

- GeometryGrid (M. Nolte)
- IdGrid (M. Nolte)
- ParallelGrid (R. Klöfkorn)
- PrismGrid

## Observations

- Easy to implement (?)
- Performance loss

## Project History

- First version developed in 2008 *(2D unstructured simplex grids to 3D prismatic grid, suitable for parallel computations)*
- New generic version since 2009
- Project website launch in 2010
- For 2011: make module available

Figure: The DUNE PrismGrid logo

## Features

- Generic prismatic elements over arbitrary *d*-dimensional DUNE grid *(the host grid)*
- Structured in vertical direction with flat upper and lower boundaries
- Periodic in vertical direction and horizontal directions *(if host grid is periodic)*
- Access to host grid

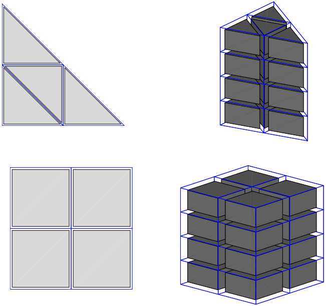


Figure: 2d host grids and resulting 3d prismatic grids

## Features

- Access to several iterators for columns and layers
- Entities of all codimensions *(independent of the host grid implementation)*
- DGF support *(including* `IntervalBlock`*)*
- Two geometry implementations *(original implementation and generic geometries)*

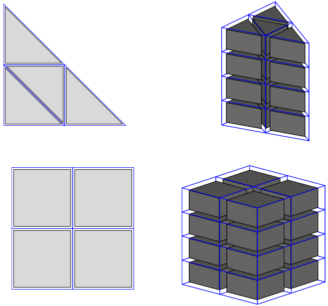


Figure: 2d host grids and resulting 3d prismatic grids

## Open Issues

- Adaptivity
- Parallel support
- I/O
- Performance

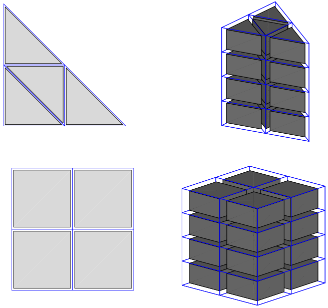


Figure: 2d host grids and resulting 3d prismatic grids

# Generic Reference Elements and Construction of Prisms

- New in DUNE 2.0: Generic reference elements
- Inductive construction rule *(pyramids and prisms)*
- In PrismGrid: Allows generic mapping from PrismGrid geometry types to host grid geometry types
- Generic Geometries

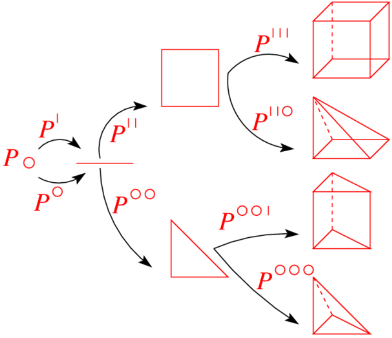



Figure: Construction of reference elements *(A. Dedner)*

# Basic Design of PrismGrid

```
template< class HostGrid >
class PrismGrid
: GridDefaultImplementation < .. >
{
  ...

  // dimension of grid
  enum { dimension = HostGrid::dimension + 1 };

  // dimension of world
  enum { dimensionworld = HostGrid::dimensionworld + 1 };

  // constructor
  PrismGrid ( HostGrid * hostgrid, LineGrid * linegrid );

  // export type of underlying host grid
  typedef typename GridFamily::HostGrid HostGrid;
  HostGrid * hostGrid_;

  // type of line grid
  typedef typename GridFamily::LineGrid LineGrid;
  LineGrid * lineGrid_;

  ...
};
```

# The LineGrid

A LineGrid is a container of intervals with iterators and geometry:

```
template< class ctype >
class LineGrid
{
  // constructor
  LineGrid ( const int n, const ctype left, const ctype right,
             ...
           );

  // return iterator for given direction
  IteratorType iterator ( int direction ) const
  {
    if ( direction == 1 )
      return up_iterator();
    else
      return down_iterator();
  }

  // return end iterator for given direction
  IteratorType end_iterator ( int direction ) const;

  ...
}
```

# DGF Support for PrismGrid

A PrismGrid can be constructed from a DGF-file of the following form:

```
DGF

HOSTGRID
hostgrid.dgf     % host grid dgf file
#

LINEGRID
0. 1. 2          % [ 0., 1. ], 2 cells
55 66            % bottomId = 55, topId = 66
0                % no periodicity
#
```

# GRIDTYPE Typedefs via GridSelector

The following `GRIDTYPE` typedefs are defined during `./configure`:

```
PRISMGRID_SGRID
PRISMGRID_YASPGRID
PRISMGRID_ONEDGRID
PRISMGRID_ALBERTA
PRISMGRID_ALUGRID_CONFORM
PRISMGRID_ALUGRID_CUBE
PRISMGRID_ALUGRID_SIMPLEX
```

Compile:

```
make GRIDTYPE=PRISMGRID_SGRID GRIDDIM=3 WORLDDIM=4 ...
// host grid is SGRID< 3, 4 >
```

There are several possible iterators:

- Columnwise
- Layerwise
- From lower to upper
- From upper to lower

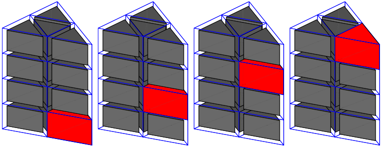


Figure: Iteration upwards a column

Which iterator shall be implemented / used / chosen?

There are several possible iterators:

- Columnwise
- Layerwise
- From lower to upper
- From upper to lower

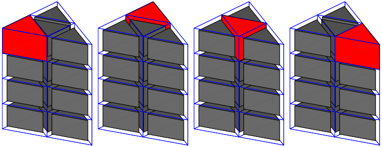


Figure: Iteration over top layer

Which iterator shall be implemented / used / chosen?

# Element Iterator

There are several possible iterators:

- Columnwise
- Layerwise
- From lower to upper
- From upper to lower

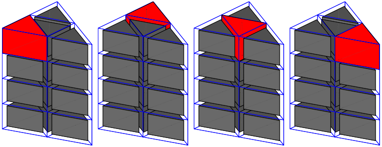


Figure: Iteration over top layer

Which iterator shall be implemented / used / chosen?

There are several possible iterators:

- Columnwise
- Layerwise
- From lower to upper
- From upper to lower

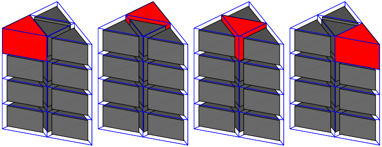


Figure: Iteration over top layer

Which iterator shall be implemented / used / chosen?

# Defining the Behaviour of PrismGrid



```
template< ... >
struct PrismGridSettings
{
  // define the iteration implementation to be used
  static const prismgrid::PrismGridIteratorImplementation
    IteratorImplementation = prismgrid::ColumnWise;
    // IteratorImplementation = prismgrid::LayerWise;

  // define the geometry implementation to be used
  static const prismgrid::PrismGridGeometryImplementation
    GeometryImplementation = prismgrid::OriginalGeometry;
    // GeometryImplementation = prismgrid::GenericGeometry;

  ...
};
```

# Access to Host Grid and Host Grid Entities

- In many applications, the meta grid and the host grid are used simultaneously.
- The `HostGridAccess` structure makes the host grid available through the meta grid and gives access to host grid entities from meta grid entities.
- Meta grids implementing the host grid access: GeometryGrid, IDGrid, PrismGrid, ...

# Access to Host Grid and Host Grid Entities

```
template< class HostGrid >
struct HostGridAccess< PrismGrid< HostGrid > >
{
  ...

  // return reference to host grid
  static const HostGrid & hostGrid ( const PrismGrid & grid )
  {
    return grid.hostGrid();
  }

  // get host grid entity
  template< class Entity >
  static const typename Codim< Entity::codimension >::HostEntity &
  hostEntity ( const Entity &entity );

  ...
};
```

# Performance Check: Explicit Finite-Volumes in 3D

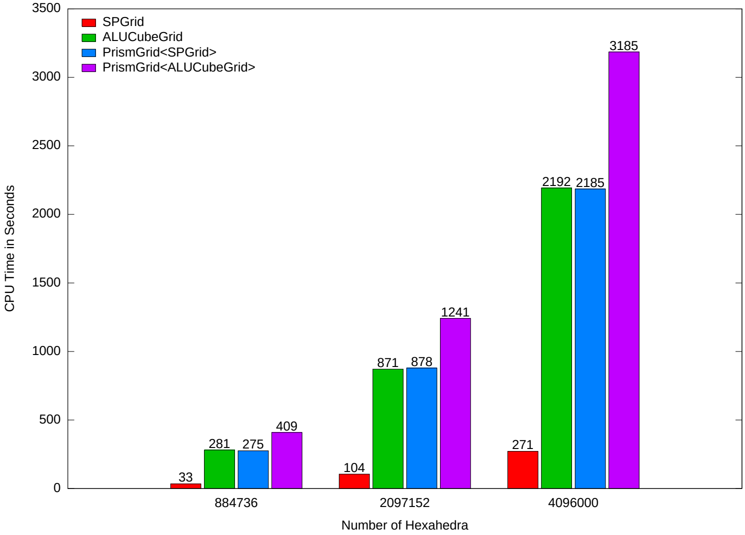
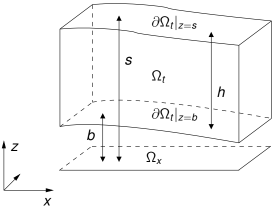



*Chart provided by M. Nolte*

- Geometry: Copy $d \times d$-FieldMatrix into $(d+1) \times (d+1)$-FieldMatrix for JacobianTransposed, JacobianInverseTransposed, ...
- Methods returning references
- Hold as few entity pointers as possible!

We consider the $d$-dimensional incompressible Navier-Stokes equations for shallow flows with a free surface ($d = 2, 3$):

$$
\begin{aligned}
\partial_t u_x + (u \cdot \nabla) u_x + \nabla_x p &= \mu \partial_z^2 u_x && \text{in } \Omega_t, \\
\nabla \cdot u &= 0 && \text{in } \Omega_t, \\
\partial_z p &= -g && \text{in } \Omega_t, \\
\partial_t s + u_x \cdot \nabla_x s &= u_z && \text{on } \partial\Omega_t|_{z=s} \\
u_x \cdot \nabla_x b &= u_z && \text{on } \partial\Omega_t|_{z=b}, \\
\mu \partial_z u_x &= 0 && \text{on } \partial\Omega_t|_{z=s}, \\
\mu \partial_z u_x &= \kappa u_x && \text{on } \partial\Omega_t|_{z=b}.
\end{aligned}
$$



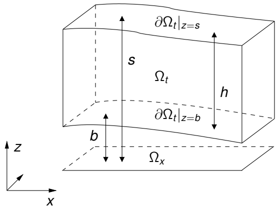
For the discretization, the so called $\sigma$-transformation is applied: Let

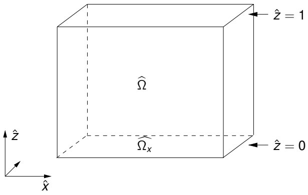
$$\hat{t} = t, \quad \hat{x} = x, \quad \text{and} \quad \hat{z} = \sigma(t, x, z) = \frac{z - b(x)}{h(t, x)}.$$

Then, for all times $t$ it holds

$$\widehat{\Omega_t} = \{(\hat{x}, \hat{z}) \mid (x, z) \in \Omega_t\} = \widehat{\Omega_x} \times (0, 1),$$

i. e. the transformed domain is fixed in time.



# LDG for Hydrostatic Incompressible Free Surface Flows

- LDG solver in DUNE-FEM
- used an early version of PrismGrid
- combination of PrismGrid and GeometryGrid for visualization

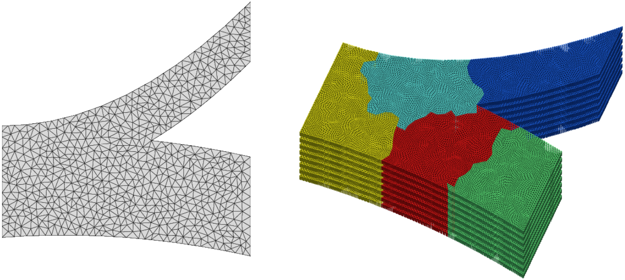


Figure: Two dimensional unstructured simplex grid (left) and resulting three dimensional prismatic grid with five partitions for parallel computations (right)

# LDG for Hydrostatic Incompressible Free Surface Flows

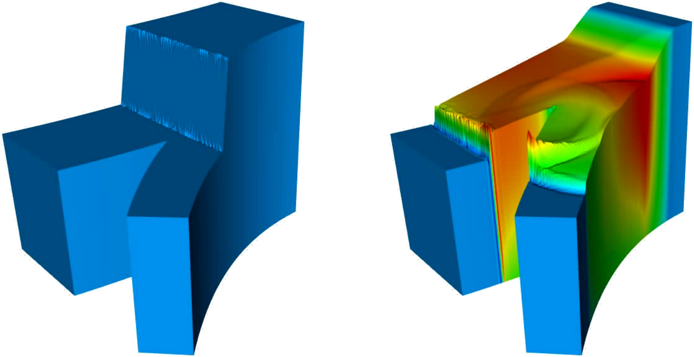


Figure: *(Left)* 3D representation of initial conditions and *(Right)* solution to a latter time

## Conclusion

- Meta grids increase the number of grids available in DUNE
- PrismGrid: meta grid with prismatic elements and additional functionality *(iterators, host grid acces)*
- Importance of generic reference elements for meta grids

## Outlook

- Parallelization
- Adaptivity
- Performance
- Documentation
- ...

Thank you for your attention!

# Conclusion and Outlook

## Conclusion

- Meta grids increase the number of grids available in DUNE
- PrismGrid: meta grid with prismatic elements and additional functionality *(iterators, host grid acces)*
- Importance of generic reference elements for meta grids

## Outlook

- Parallelization
- Adaptivity
- Performance
- Documentation
- ...

Thank you for your attention!

# Conclusion and Outlook



## Conclusion

- Meta grids increase the number of grids available in DUNE
- PrismGrid: meta grid with prismatic elements and additional functionality *(iterators, host grid acces)*
- Importance of generic reference elements for meta grids

## Outlook

- Parallelization
- Adaptivity
- Performance
- Documentation
- ...

Thank you for your attention!