The DUNE-FEM Module

Global Valued Finite Elements and Discrete Functions

Christoph Gersbacher, Stefan Girke

September 19, 2013

Münster, Germany

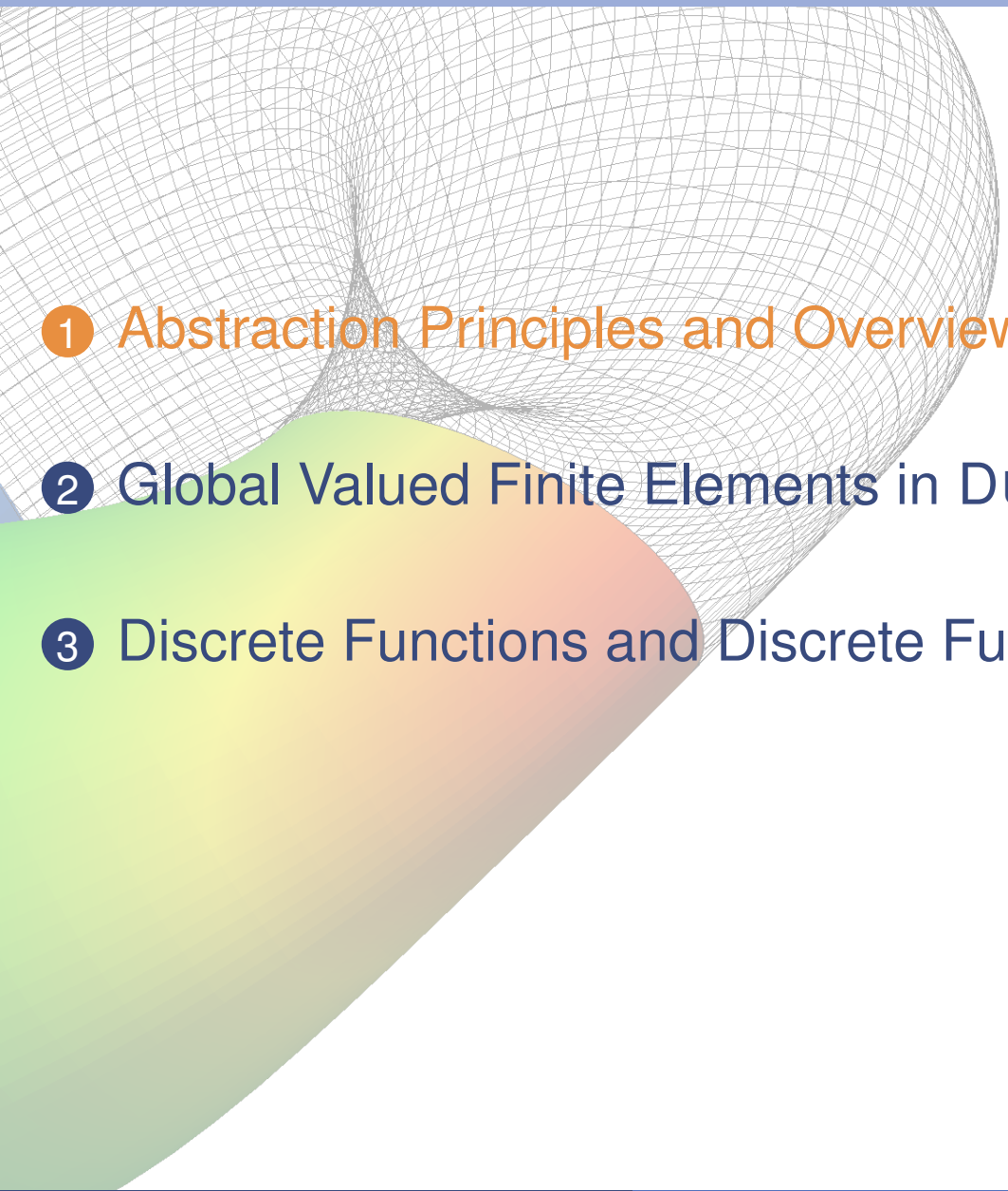
Department of Applied Mathematics
University of Freiburg

www.dune.mathematik.uni-freiburg.de

Department of Computational and Applied Mathematics
University of Münster

WESTFÄLISCHE WILHELMS-UNIVERSITÄT MÜNSTER

Outline



Discrete Function Spaces in DUNE-FEM

- Continuous Lagrange spaces of arbitrary order
- Discontinuous Galerkin spaces
- *p*-adaptive DG and Lagrange space
- Finite volume space
- Rannacher-Turek space using DUNE-LOCALFUNCTIONS
- ...

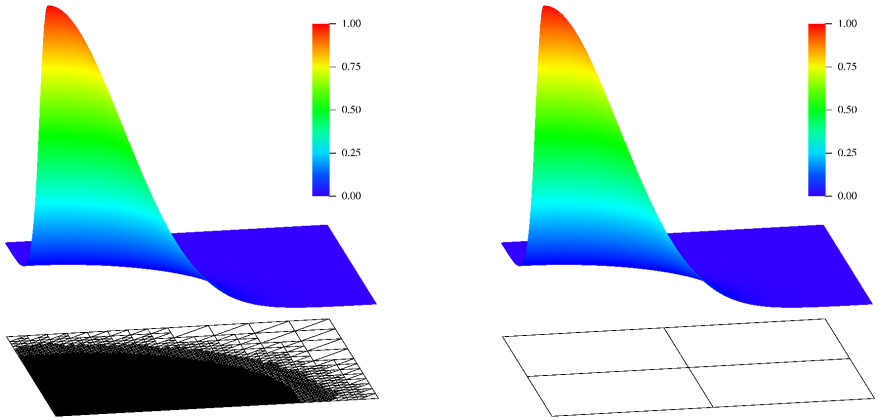


Figure: P_1- and Q_6-Finite-Element solutions of the Poisson Equation

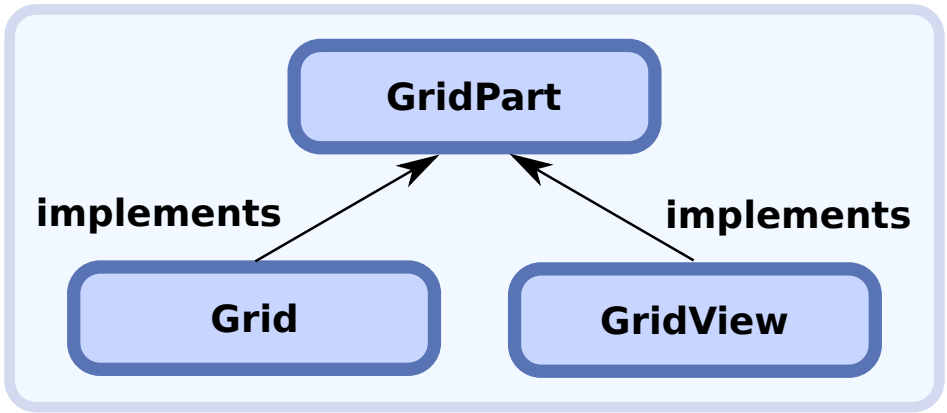
Entity Containers

The main entity container in DUNE-FEM is called `GridPart`. Think of a `GridPart` as a `GridView` providing an extended index set.

In the following, $\mathcal{G}$ denotes a grid part,

$$\mathcal{G} = \{E_i \mid i = 1, \ldots, N\},$$

with codimension 0 entities E_i, $i = 1, \ldots, N$. The domain covered by the grid part's elements is $\Omega^{\mathcal{G}} = \bigcup_{E \in \mathcal{G}} E$.



Discrete Function Spaces

Consider a finite-dimensional function space $V^{\mathcal{G}} : \Omega^{\mathcal{G}} \to \mathcal{U}$ spanned by a global basis

$$\mathcal{B}^{\mathcal{G}} = \{\varphi_\alpha^{\mathcal{G}} : \Omega^{\mathcal{G}} \to \mathcal{U} \mid \alpha = 0, \dots, N^{\mathcal{G}} - 1\}.$$

We assume that:

- For each element $E \in \mathcal{G}$ we have a set of *local basis functions* $\mathcal{B}^E = \{\varphi_i^E : E \to \mathcal{U} \mid i = 0, \dots, N^E - 1\}$.
- For each local basis function $\varphi_i^E \in \mathcal{B}^E$ there exists a unique $\varphi_\alpha^{\mathcal{G}} \in \mathcal{B}^{\mathcal{G}}$ such that

  $$\varphi_i^E = \varphi_{\mu_E(i)}^{\mathcal{G}}|_E,$$

  which simultaneously defines a bijective *index mapping* μ_E from local to global to degrees of freedom (Dofs).

Discrete Functions

Let $u \in V^{\mathcal{G}} = \operatorname{span} \mathcal{B}^{\mathcal{G}}$, i.e.,

$$u(x) = \sum_{\alpha} u_\alpha \varphi_\alpha^{\mathcal{G}}(x).$$

We call $(u_\alpha)_{\alpha=0}^{N^{\mathcal{G}}-1}$ the *global Dof vector*.

The restriction of u to a grid element E yields the *local discrete function*:

$$u|_E = \sum_{\alpha} u_{\varphi_\alpha} \varphi_\alpha|_E = \sum_{i} u_{\mu_E(i)} \varphi^E_{\mu_E(i)}$$

and the corresponding *local Dof vector* $(u_{\mu_E(i)})_{i=0}^{N^E-1}$.

Interface Classes and Relations in DUNE-FEM

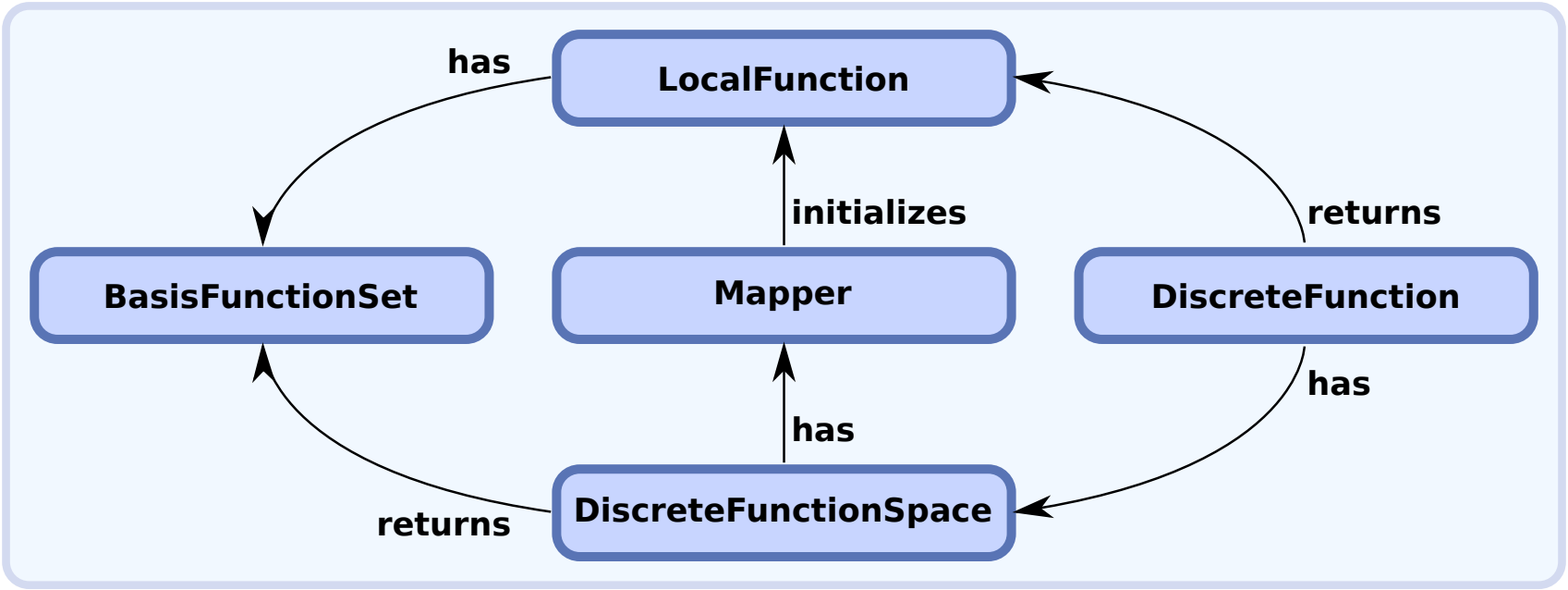
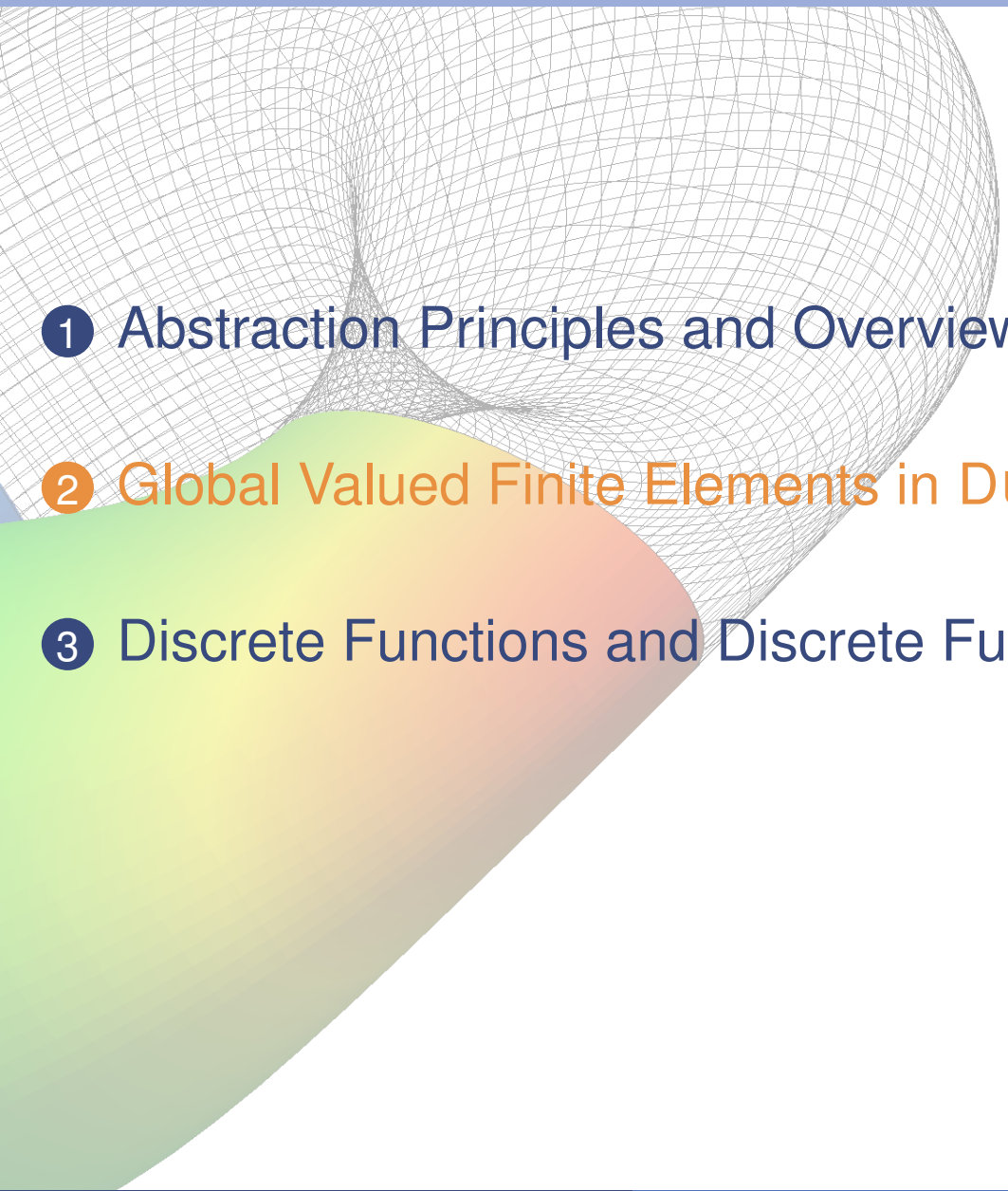


Figure: Class diagram for discrete function spaces in DUNE-FEM

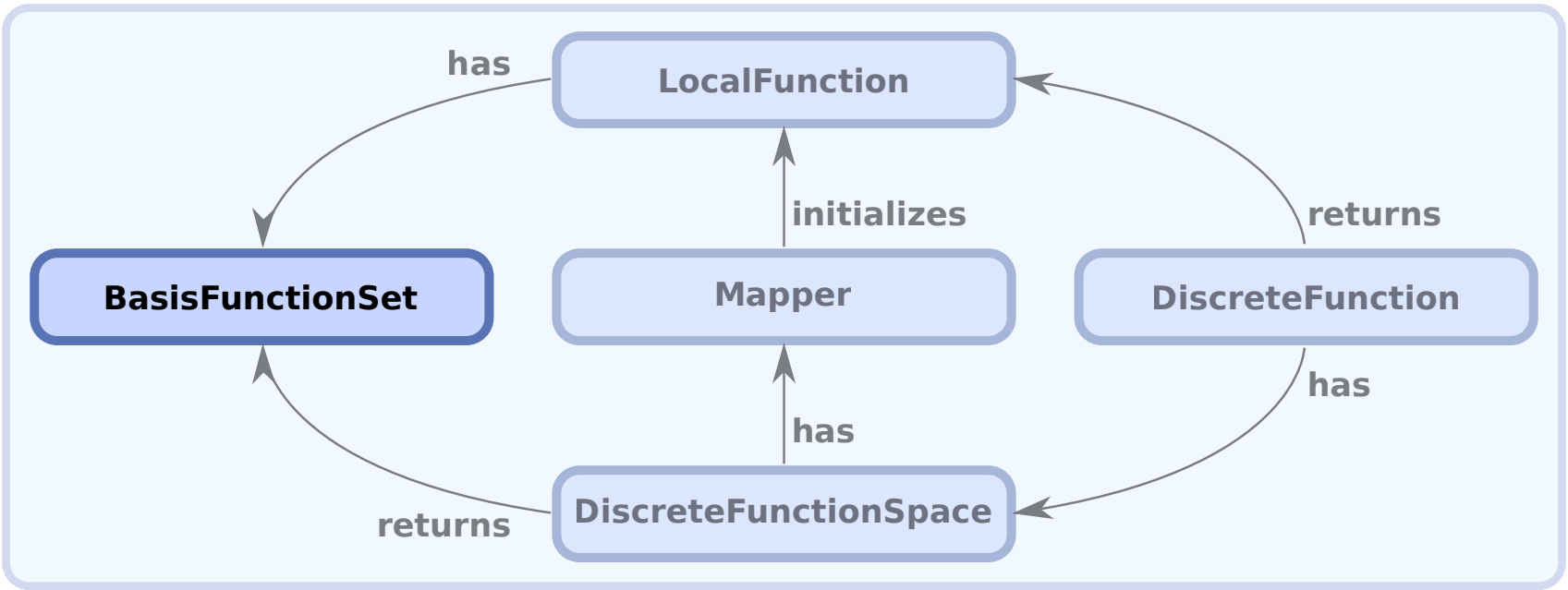
For a complete description of the DUNE-FEM interfaces see:

A. Dedner, R. Klöfkorn, M. Nolte, M. Ohlberger. *A Generic Interface for Parallel and Adaptive Discretization Schemes: Abstraction Principles and the* DUNE-FEM *Module*. Computing, 2010.

Outline



Global Valued Finite Elements



The `BasisFunctionSet` Interface

```
template<class EntityType, class RangeType>
struct BasisFunctionSet
{
  const EntityType &entity() const; // return entity
  int size() const; // number of basis functions
  int order() const; // order of basis functions

  // evaluate basis functions in Point (e.g., a local coordinate)
  template<class Point, class RangeArray>
  void evaluateAll(const Point &x, const RangeArray &values) const;

  // evaluate linear combination
  template<class P, class DV>
  void evaluateAll(const P &x, const DV &dofs, RangeType &value) const;

  // evaluate linear combination for quadrature rule
  template<class Quadrature, class DV, class RA>
  void evaluateAll(const Quadrature &quadrature,
                   const DV &dofs, RA &values);

  // jacobianAll(...), hessianAll(...), axpy(...)
};
```

Implementing `BasisFunctionSet`

Few discrete function spaces implement the `BasisFunctionSet` interface directly:

- `FiniteVolumeSpace`: In case of the piecewise constant function space $V^{\mathcal{G}} : \Omega^{\mathcal{G}} \to \mathcal{U} \subset \mathbb{R}^r$ we set for $E \in \mathcal{G}$:

$$\mathcal{B}^E = \{\varphi_i^E(x) = 1 \mid i = 1, \dots, r\}.$$

- `FourierDiscreteFunctionSpace`: In the scalar case the global basis function set is given by

$$\mathcal{B}^{\mathcal{G}} = \left\{ \frac{1}{2}, \cos(x), \sin(x), \dots, \cos(Nx), \sin(Nx) \right\}$$

for some $N \in \mathbb{N}$. Note that we have $\mathcal{B}^{\mathcal{G}} = \mathcal{B}^E$ for all E in $\mathcal{G}$.

Shape Functions

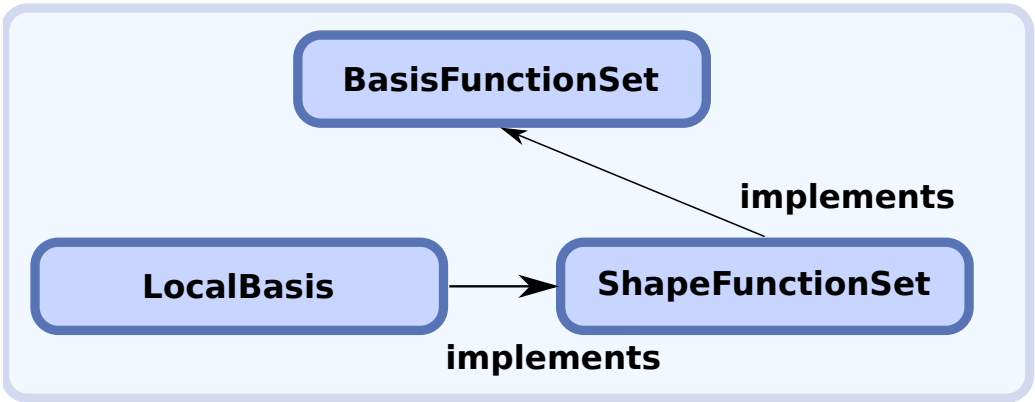
Prominent examples of discrete function spaces can be based on fixed sets of shape functions

$$\{\hat{\varphi}_i^{\hat{E}} \mid i = 0, \ldots, N^{\hat{E}-1}\}$$

defined for each reference element $\hat{E} \subset \mathbb{R}^d$ in a grid part $\mathcal{G}$.

In the following, we assume that for each element $E \in \mathcal{G}$ with geometry $F_E : \hat{E} \to E$ the local basis $\mathcal{B}^E = \{\varphi_i^E\}$ takes the form

$$\varphi_i^E = \hat{\varphi}_i^{\hat{E}} \circ F_E^{-1} \qquad (i = 0, \ldots N^{\hat{E}} - 1).$$



The `ShapeFunctionSet` Interface

```
struct ShapeFunctionSet
{
  ... // we omit some typedefs

  std::size_t size() const; // number of shape functions
  int order() const;  // polynomial order of shape functions

  // What is a functor? See below...
  template<class Point, class Functor>
  void evaluateEach(const Point &x, Functor functor) const;

  // And a Point? Think of a local coordinate.
  template<class Point, class Functor>
  void jacobianEach(const Point &x, Functor functor) const;

  template<class Point, class Functor>
  void hessianEach(const Point &x, Functor functor) const;
};
```

Evaluating the Shape Function Set

The values of all shape functions are needed when assembling, e.g., the local mass matrix $M = (m_{ij})_{i,j}$,

$$m_{ij} = \int_E \varphi_i^E \, \varphi_j^E = \int_{\hat{E}} \hat{\varphi}_i^{\hat{E}} \, \hat{\varphi}_j^{\hat{E}} \cdot |\det DF_E|.$$

On the other hand, e.g., when evaluating a local discrete function

$$u|_E = \sum_i u_i \varphi_i^E(x) \qquad (u_i \in \mathbb{R}),$$

$$u|_E(x) = \sum_i u_i \hat{\varphi}_i^{\hat{E}}(\hat{x}) \qquad (\hat{x} = F_E^{-1}(x)),$$

there is no need to store the vector $\left(\hat{\varphi}_i^{\hat{E}}(F_E^{-1}(x))\right)_i$ of evaluated shape functions.

Evaluating the Global Finite Element

```
template<class Scalars, class Value>
struct AxpyFunctor
{
  AssignFunctor(const Scalars &c, Value &result)
  : c_(c), result_(result)
  {}

  void operator()(const int i, const Value &value)
  {
    result_.axpy(c_[i], value);
  }

private:
  const Scalars &c_;
  Value &result_;
};
```

Evaluating the Global Finite Element (cont.)

```
template<class Entity, class ShapeFunctionSet>
class DefaultBasisFunctionSet
{
  struct AxpyFunctor;

public:
  template<class Point, class DofVector>
  void evaluateAll(const Point &x, const DofVector &dofs,
                   RangeType &value) const
  {
    value = RangeType(0);
    AxpyFunctor<DofVector, RangeType> f(dofs, value);
    shapeFunctionSet_.evaluateEach(x, f);
  }

  // ...

private:
  ShapeFunctionSet shapeFunctionSet_;
};
```

Implementations of `ShapeFunctionSet`

A number of `ShapeFunctionSet` implementations are available in DUNE-FEM:

- Scalar shape function sets:

  `LegendreSFS`, `LagrangeSFS`, `OrthonormalSFS`, `...`

- Wrapper for `LocalBasis` from DUNE-LOCALFUNCTIONS:

  `LocalFunctionsShapeFunctionSet<LocalFunctions>`

- Turn scalar valued shape function set to vector valued one:

  `VectorialShapeFunctionSet<ScalarSFS, Range>`

- Tensor product shape function set for cartesian grids:

  `TensorproductShapeFunctionSet<SFSTuple>`

Discrete Function Spaces in Dune-Fem

Most implementations of discrete function spaces in DUNE-FEM are based on shape function sets.

- Continuous Lagrange spaces for hybrid grids:

  `LagrangeSpace<FunctionSpace, GridPart, order>`

- Generic Discontinuous Galerkin spaces:

  `DiscontinuousGalerkinSpace<FS, GP, order, ShapeFunctionSet>`

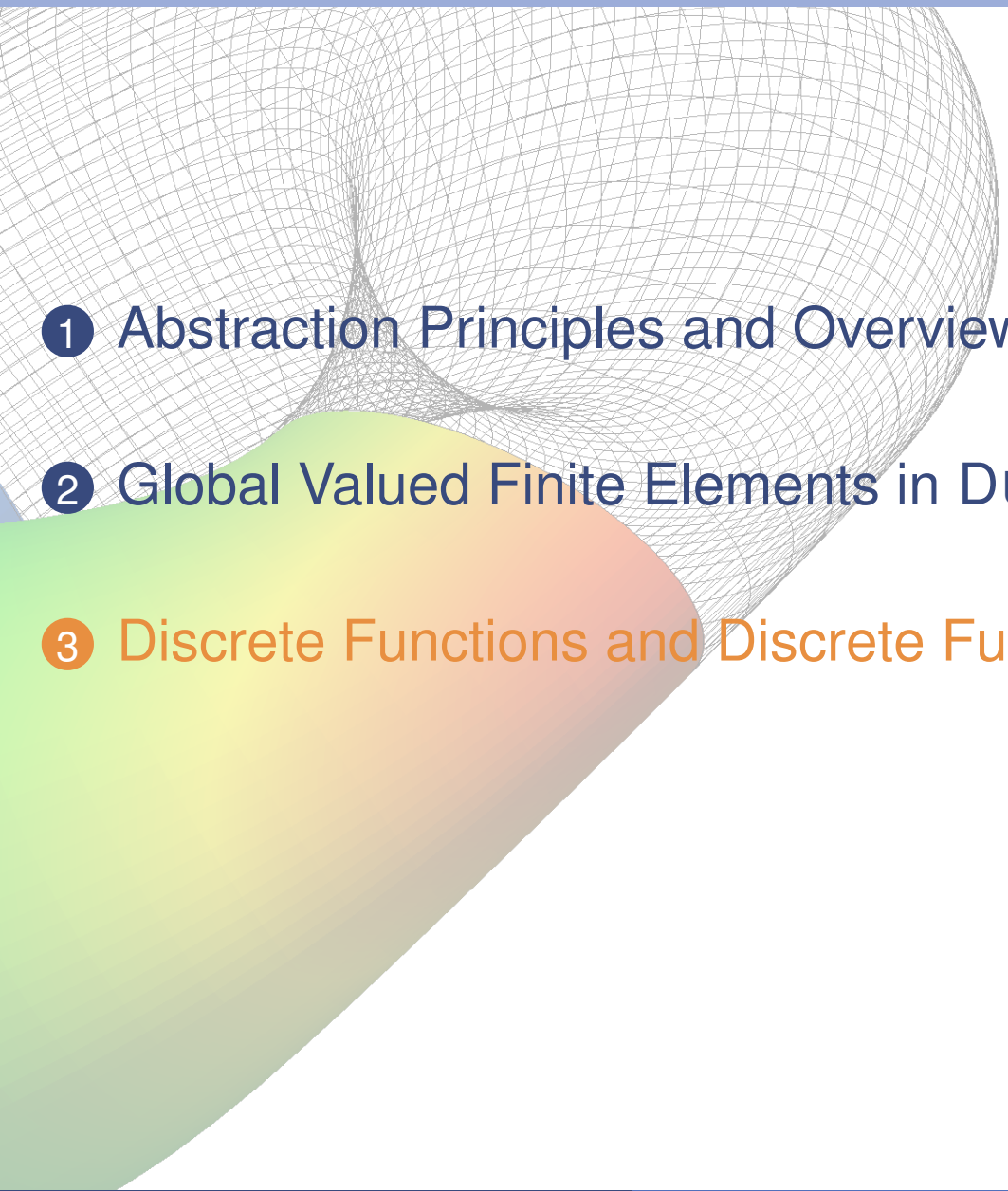
- Base class for *p*-adaptive DG and Lagrange space:

  `GenericDiscreteFunctionSpace<FS, GP, order, ShapeFunctionSet>`

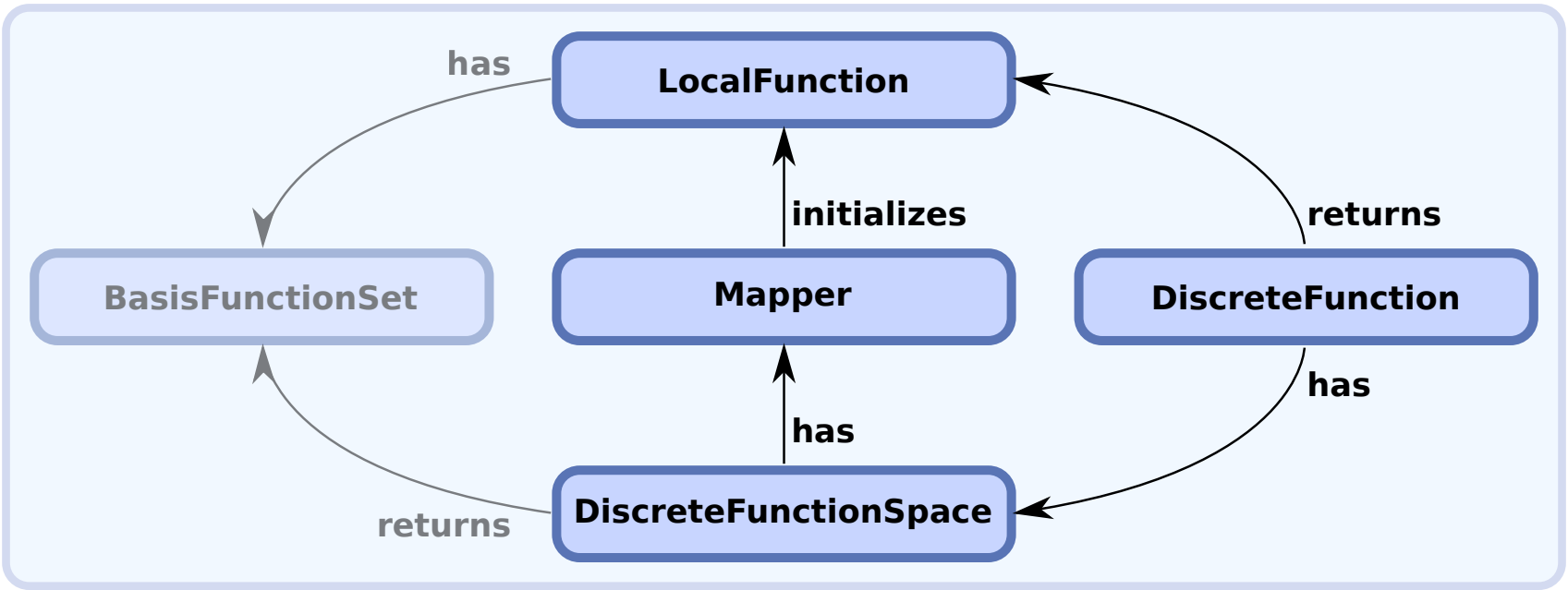
- Example implementation using Dune-LocalFunctions:

  `RannacherTurekSpace<FS, order>`

Outline



Discrete Function Spaces



Discrete Function Spaces

```
struct DiscreteFunctionSpace
{
  int size() const; // number of global basis functions
  int order() const; // overall polynomial order

  // get local basis function set
  BasisFunctionSet basisFunctionSet(const EntityType &entity) const;

  // returns dof mapper, see below...
  BlockMapperType &blockMapper() const;

  // communicate dofs and apply operation (copy, assignment, etc.)
  template<class DiscreteFunction, class Operation>
  void communicate(DiscreteFunction &dF, const Operation &) const;

  // interpolate local function and write to vector
  template<class LocalFunction, class LocalDofVector>
  void interpolate(const LocalFunction &lF, LocalDofVector &d) const;

  // get grid part
  GridPartType &gridPart();
};
```

Discrete Functions

```
struct DiscreteFunction
{
  // size of a discrete function equals number of dofs
  int size() const; // return number of dofs

  // return reference to discrete function space
  const DiscreteFunctionSpaceType &space() const;

  // return local representation of the discrete function
  LocalFunctionType localFunction(const EntityType &entity);

   // type used to identity blocks on global dof vector
  typedef ImplementationDefined GlobalKeyType;
  // access block of dofs
  DofBlockType &block(const GlobalKey &key);

  // global dof vector modifiers
  void clear(); // set all dofs to zero
  DiscreteFunction &operator+=(const DiscreteFunction &other);
  DiscreteFunction &operator-=(const DiscreteFunction &other);
  // ... assignment, scaling, etc.
};
```

Local Discrete Functions

A `LocalFunction` is a local dof vector plus a local basis function set.

```
template<class Traits>
struct LocalFunction
{
  // return local basis function set
  BasisFunctionSetType basisFunctionSet() const;

  // evaluation
  template<class PointType>
  void evaluate(const PointType &x, RangeType &ret) const;
  template<class PointType>
  void jacobian(const PointType &x, JacobianRangeType &ret) const;

  // dof access
  DofType &operator[](int i);
  const DofType &operator[](int i) const;

  // modifiers
  template<class Other>
  LocalFunctionType &operator+=(const LocalFunction<Other> &);
  // ...
};
```

Dof Mappers

```
template<class GridPartType>
struct DofMapper
{
  int size() const; // global number of dofs

  // return number of dofs for given entity
  template<class Entity>
  int numEntityDofs(const Entity &entity) const;
  // visit dofs for entity
  template<class Entity, class Functor>
  void mapEachEntityDof(const Entity &entity, Functor functor) const;

  // return sum over all subentity dofs of codimension 0 entity
  int numDofs(const ElementType &element) const;
  // visit all subentity dofs
  template<class Functor>
  void mapEach(const ElementType &element, Functor functor) const;
};
```

Initializing Local Dof Vectors

```
// constructor; initializes dofs of local function for given
    codimension 0 entity
LocalFunction::LocalFunction(DiscreteFunction &discreteFunction,
                             const EntityType &entity)
: space_(discreteFunction.space())
  basisFunctionSet_(space_.basisFunctionSet(entity)),
  dofs_(basisFunctionSet_.size())
{
  AssignDofsFunctor functor(discreteFunction,dofs_);
  space.blockMapper().mapEach(entity,functor);
}

template<class DiscreteFunction, class GlobalKey>
inline void AssignDofsFunctor<DiscreteFunction>
  ::operator()(int local, const GlobalKey &key)
{
  typedef typename DiscreteFunction::DofBlockType DofBlockType;
  DofBlockType block = discreteFunction_.block(key);
  for(int j = 0; j < block.size(); ++j)
    dofs_[i_++] = block[j]; // i_: internal counter
}
```