Dune-DEC

A Discrete Exterior Calculus Module

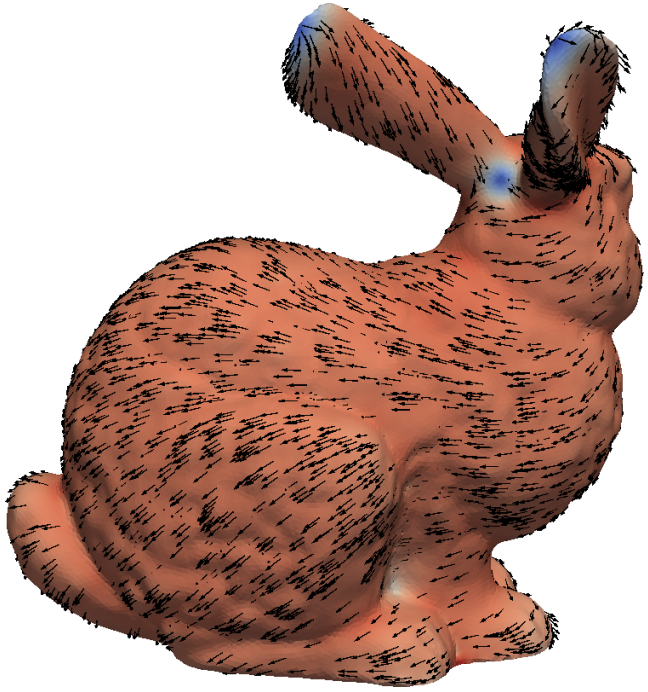
Simon Praetorius *simon.praetorius@tu-dresden.de*

Institute for Scientific Computing
Technische Universität Dresden

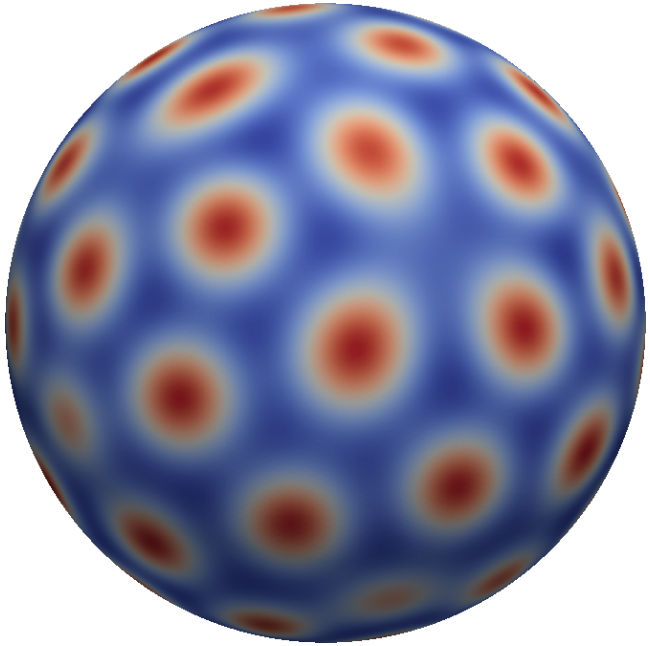
Introduction

1. Vector-field relaxation on manifolds
2. High order scalar PDE: Phase-Field Crystal equation

(1)



(2)



What is Discrete Exterior Calculus

- *Idea:* Construct a discrete theory that mimics a continuous vector calculus theory
- Work with discrete fields and forms living on grid entities directly
- Extend the notion of exterior product and exterior derivatives to the discrete level
- Purely geometry quantities involved
- Generalization of finite-difference/finite-volume to unstructured grids
- Algebraic theory formulated in

A. Hirani: "Discrete Exterior Calculus", PhD thesis (2003)

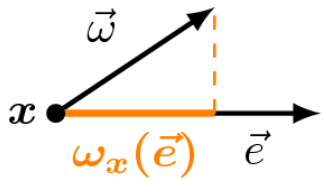
Goal of this talk

- Show you some ideas of the theory
- Requirements for a software framework
- An implementation approach

Differential Geometry

Forms and Fields

- A **differential form** of degree k (k-form): smooth alternating k-multilinear form, maps k (tangent) vectors to $\mathbb{R}$
  - *Example:* determinant (measures signed volume of parallelpiped spanned by k vectors)
  - *Example:* Vectors $\vec{\omega}$ and covectors ω^1 (1-form)



In 2D:
0-forms $\omega^0 \sim$ scalar fields
1-forms $\omega^1 \sim$ vector fields
2-forms $\omega^2 \sim$ scalar fields

- **Musical Isomorphisms** $\sharp$ and $\flat$: Map vectors to forms and vice-versa, i.e.

$$\omega^\sharp := \vec{\omega}, \qquad \vec{\omega}^\flat := \omega$$

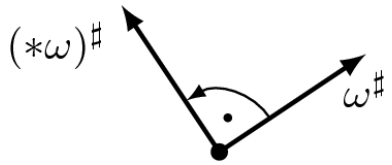
- Identification in $\mathbb{R}^2$:

$$\omega^1(\vec{e}) = \omega^\sharp \cdot \vec{e}, \qquad \omega^2(\vec{e}, \vec{f}) = \omega^\sharp(\vec{e} \times \vec{f})$$

Differential Geometry

Operations on forms

- **Hodge-Star operator** $*$: Maps k-forms to (n-k)-forms.



In 2D: vectors can be uniquely represented by their orthogonal vectors (rotated by 90 degree)
In 3D: (oriented) planes can be indentified with their normals

Example: volume form $*1$:

$$vol(\Omega) = \int_\Omega *1$$

- **Exterior derivative:** Maps k-forms to $(k+1)$-forms, such that

$$\int_\Omega \mathrm{d}\omega := \int_{\partial\Omega} \omega$$

(ω a k-form, Ω a $(k+1)$-dim. orientable manifold) - Stoke's theorem

Differential Geometry

Examples:

$$\nabla f = (\mathrm{d}f)^\sharp$$
$$\nabla \times \vec{\omega} = \left(* \, \mathrm{d} \vec{\omega}^\flat \right)^\sharp$$
$$\nabla \cdot \vec{\omega} = * \mathrm{d} * \vec{\omega}^\flat$$

(with f a 0-form and ω a 1-form with vector proxy $\vec{\omega}$)

- Since $\mathrm{d}f$ is a 1-form, it maps vectors to $\mathbb{R}$, i.e. in vector calculus:

$$\mathrm{d}f(\vec{e}) = \vec{e} \cdot \nabla f$$

- Laplace Beltrami operator:

$$\Delta f = (* \mathrm{d} * \mathrm{d}) f$$

- Laplace de-Rham operator:

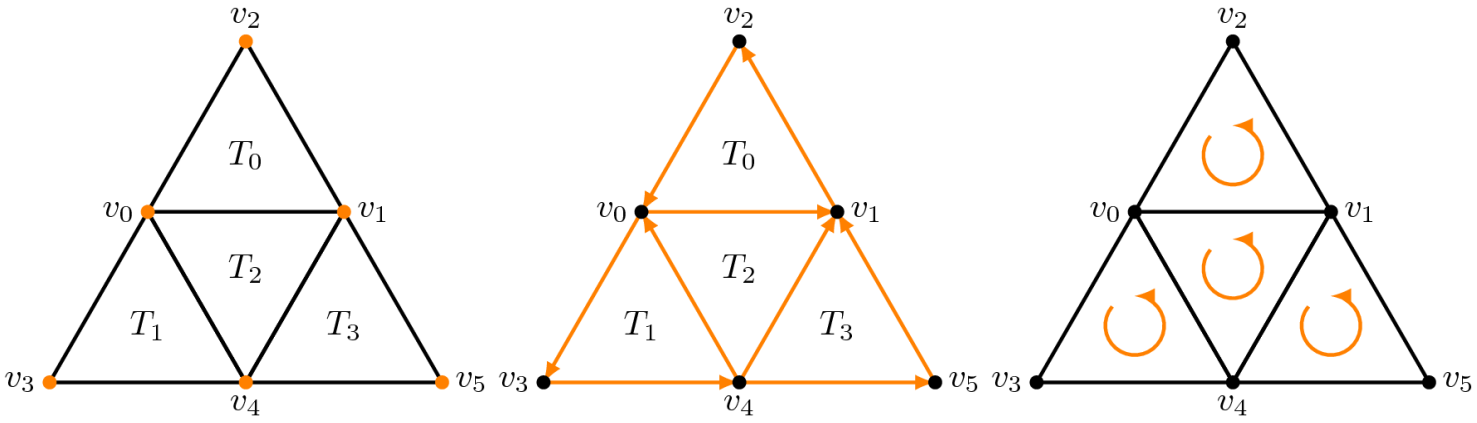
$$\Delta \omega = (* \mathrm{d} * \mathrm{d} + \mathrm{d} * \mathrm{d} *) \omega$$

Discrete Differential Geometry

Simplicial Complex

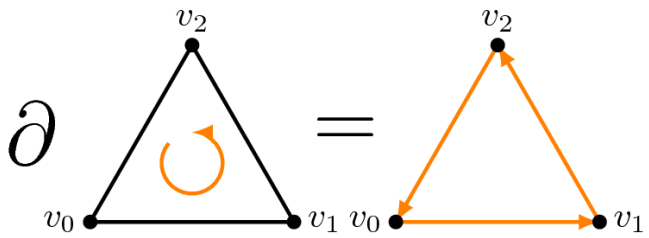
Triangulation Ω_h:

- Collection of **oriented** simplices: vertices, edges, faces
- Every face of a simplex of Ω_h is in Ω_h
- Intersection of any 2 simplices of Ω_h is a face of each of them. (**Conforming** triangulation)
- For simplicity: **well-centered**, i.e. the circumcenter of all simplices are inside the entity



Discrete Differential Geometry

- **Boundary** of a k-simplex $\sigma_k = [v_0, \ldots, v_k]$ is formal sum of facets:



$$\partial[v_0, v_1, v_2] = [v_1, v_2] - [v_0, v_2] + [v_0, v_1]$$

- Let Ω^k denote the k-simplices of Ω_h. A **k-chain** c is a formal (weighted) sum of entities:

$$c = \sum_{\sigma \in \Omega^k} c_\sigma \sigma, \quad \text{with } \partial c = \sum_{\sigma \in \Omega^k} c_\sigma \partial \sigma$$

with $c_\sigma \in \mathbb{Z}$ weights.

Interpretation: Weight encodes how often and in which direction an entity is traversed in a chain

Discrete Differential Geometry

- Discrete differential form: integral value over submanifolds (e.g. edges):

$$\omega_h(e) := \int_e \vec{\omega} \cdot \vec{e} \, \mathrm{d}e$$

with $\vec{e}$ the edge vector (oriented!)

- More generally: a **k-cochain** maps (chains of) k-dimensional entities to $\mathbb{R}$

$$\omega_h(c) := \sum_{\sigma \in \Omega^k} c_\sigma \int_\sigma \omega, \qquad \text{with } \omega \text{ a } k-\text{form}$$

- Exterior derivate for cochains defined by Stokes's theorem

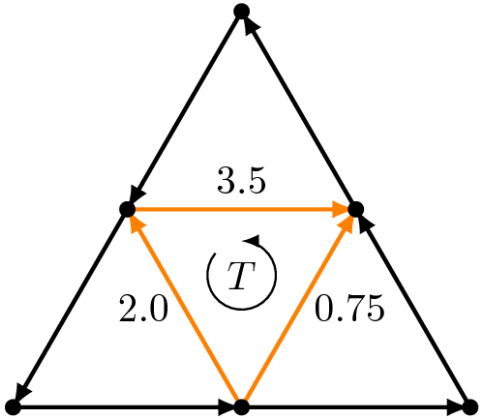
$$\mathrm{d}\omega_h(\sigma) := \int_\sigma \mathrm{d}\omega = \int_{\partial\sigma} \omega = \omega_h(\partial\sigma)$$

for a k-form ω and a $(k+1)$-simplex σ.

Discrete Differential Geometry

Example

The derivative of a discrete 1-form is a 2-form. The 1-form is given by values on the edges, the 2-form by values on the elements:



The boundary 2-chain (given as a set of weights)

$$\partial T = \{1, -1, -1\}$$

The exterior derivative

$$\mathrm{d}\omega_h^1(T) = 1 \cdot 0.75 + (-1) \cdot 3.5 + (-1) \cdot 2 = (1, -1, -1) \cdot (0.75, 3.5, 2)^\top = -4.75$$

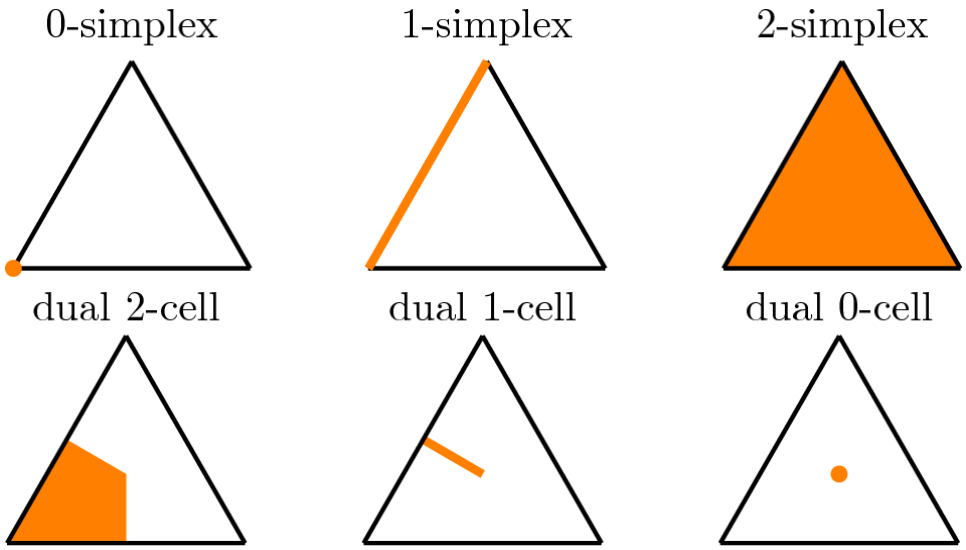
Discrete Differential Geometry

Dual grid

- Incredient to define a discrete Hodge star operator
- Property of continuous operator must be conserved:

$$\int_{\Omega_h} *1_h = vol(\Omega_h)$$

- Each (primal) k-simplex is assigned a dual (n-k)-cell



(dual cells restricted to a triangle)

Discrete Differential Geometry

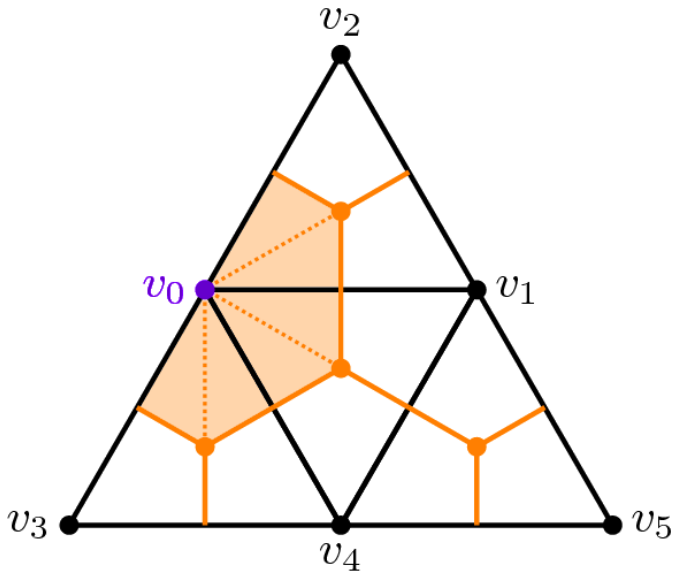
Dual grid

- **Circumcenter** of a simplex σ denoted by $c(\sigma)$
- Formal definition of the **dual cell** $\star\sigma$ of simplex σ:

$$\star\sigma^k := \sum_{\sigma^k \prec \sigma^{k+1} \prec \ldots \prec \sigma^n} s \cdot [c(\sigma^k), c(\sigma^{k+1}), \ldots, c(\sigma^n)]$$

with s a sign that ensures consitency of the orientation with the orientation of the primal simplex, e.g. vertex dual has same orientation as triangles.

$\Rightarrow$ a dual cell is an (n-k)-chain of simplices.



Discrete Differential Geometry

Discrete Hodge-Star Operator

- Maps primal k-cochains to dual (n-k)-cochains (and vice-versa)
- Should conserve some continuous properties, e.g. for k-forms ω, ν

$$
\begin{aligned}
** \omega &= (-1)^{k(n-k)} \omega && \text{(Bijectivity)} \\
*(f_1 \omega + f_2 \nu) &= f_1 \cdot *\omega + f_2 \cdot *\nu && \text{(Linearity)}
\end{aligned}
$$

- Simplest definition (**diagonal Hodge-star**):

$$
*\omega_h(\star\sigma) := \frac{|\star\sigma|}{|\sigma|} \omega_h(\sigma)
$$

This means: average over primal cell equals average over dual cell:

$$
\frac{1}{|\sigma|} \int_\sigma \omega = \frac{1}{|\star\sigma|} \int_{\star\sigma} *\omega
$$

Example: $\omega = 1, \sigma = vertex \Rightarrow 1 = \frac{1}{|\star\sigma|} \int_{\star\sigma} *1$

Implementation

Preliminary draft of Dune module

dune-dec (https://gitlab.math.tu-dresden.de/spraetor/dune-dec)

DEC Software

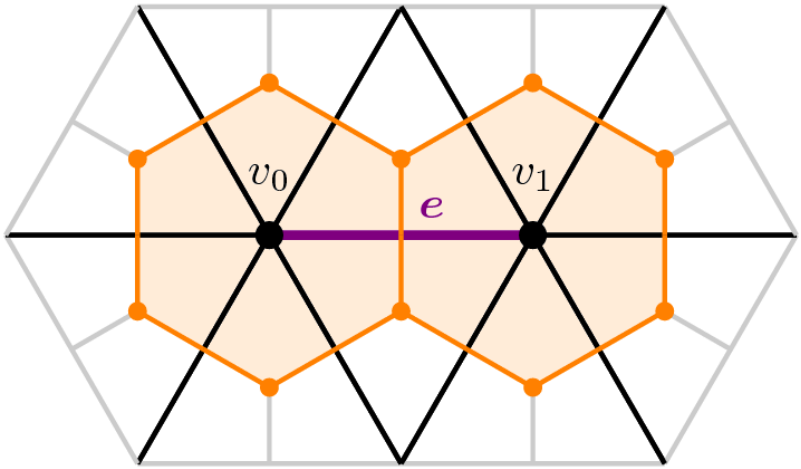
- **PyDEC** (N. Bell, A. Hirani)
  - python framework
  - simplex and cube meshes
  - 2D and 3D
  - No parallelization
- **Kahler** (A. Eftimiades)
  - Hermitean manifolds
  - Simplex grid with metric
- **DGtal**: (D. Coeurjolly, P. Gueth)
  - Cube meshes
  - Only volume grid (no manifolds)

Implementation

- Challenges:
  - Well-centered simplex grid
  - Entities must be oriented
  - calculation of volumes (of primal and dual cells)
  - Extraction of simplex chains

Example: (Grad-Div)-Laplace operator for 1-form ω on edge e

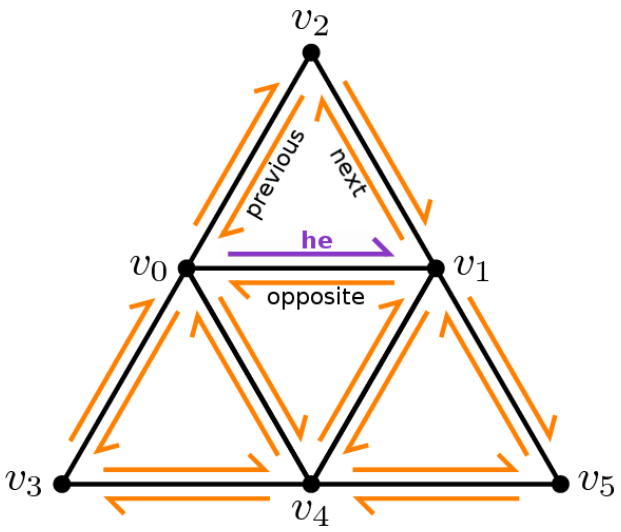
$$\Delta^{\mathrm{GD}}\omega_h(e) = (\mathrm{d} * \mathrm{d} * \omega_h)(e) = -\sum_{v \prec e} \frac{s_{v,e}}{|\star v|} \sum_{e' \succ v} s_{v,e'} \frac{|\star e'|}{|e'|} \omega_h(e')$$



Realization

Half-Edge Datastructure

- Assign to each edge two oppositely oriented **half-edges**
- Provide 3 operations:
  - next
  - previous
  - opposite
- All entities are identified by 1 half-edge … defines orientation

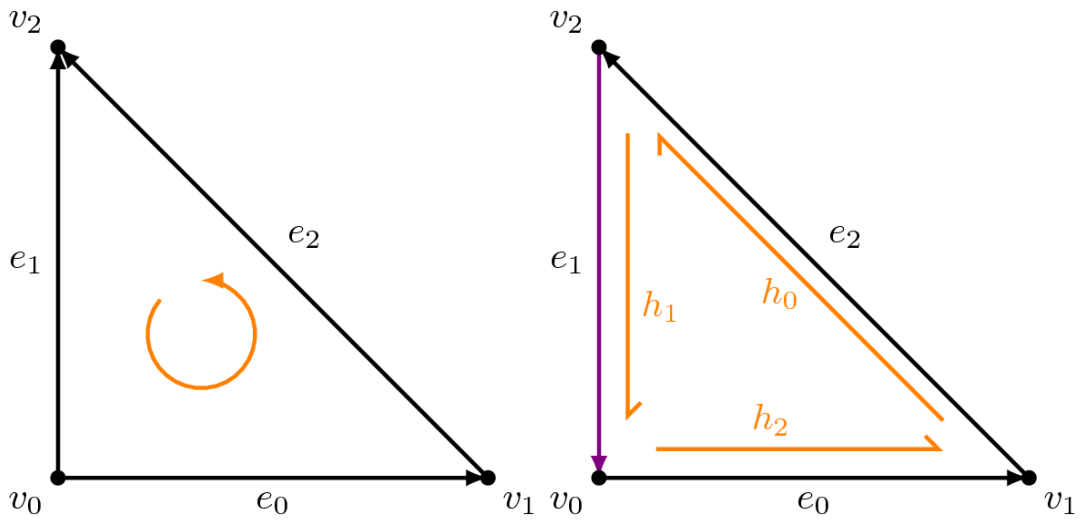



```
struct HalfEdge {
  HalfEdge* next;
  HalfEdge* previous;
  HalfEdge* opposite;

  EntitySeed vertex;
  EntitySeed edge;
  EntitySeed face;
};
```

Reference elements

Dune reference elements $\leftrightarrow$ Oriented reference elements



Reorient edges by induced orientation, i.e.

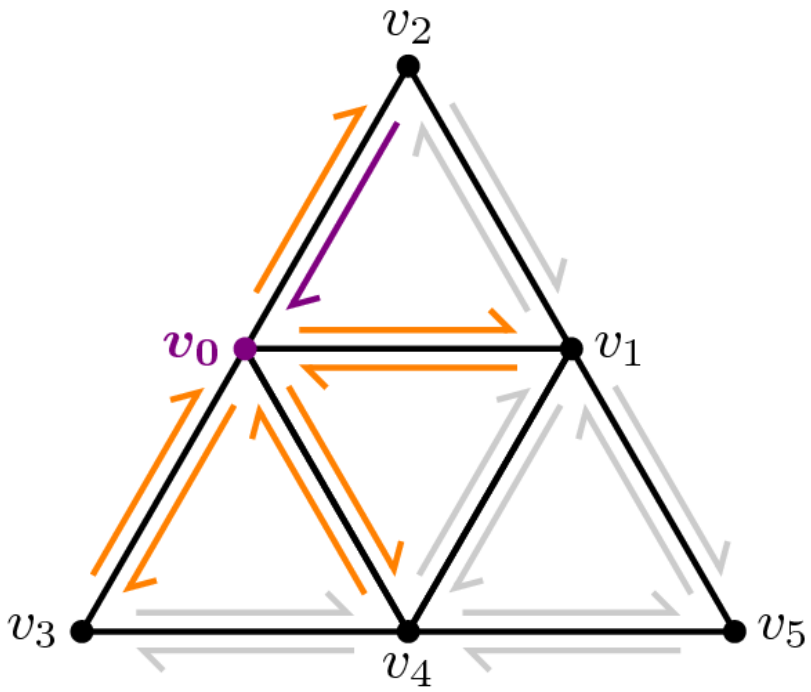
$$
\begin{aligned}
T &= [v_0, v_1, v_2] \\
e_0 &= [v_0, v_1, \widehat{v_2}] = [v_0, v_1] \to h_2 \\
e_1 &= -[v_0, \widehat{v_1}, v_2] = [v_2, v_0] \to h_1 \\
e_2 &= [\widehat{v_0}, v_1, v_2] = [v_1, v_2] \to h_0
\end{aligned}
\qquad
\mathrm{sign}_i = \begin{cases} +1 & i \text{ is even} \\ -1 & i \text{ is odd} \end{cases}
$$

(Symbol $\widehat{v_i}$ means: without v_i)

Half-Edge Iteration

Provide chains of entities by defining an iterator

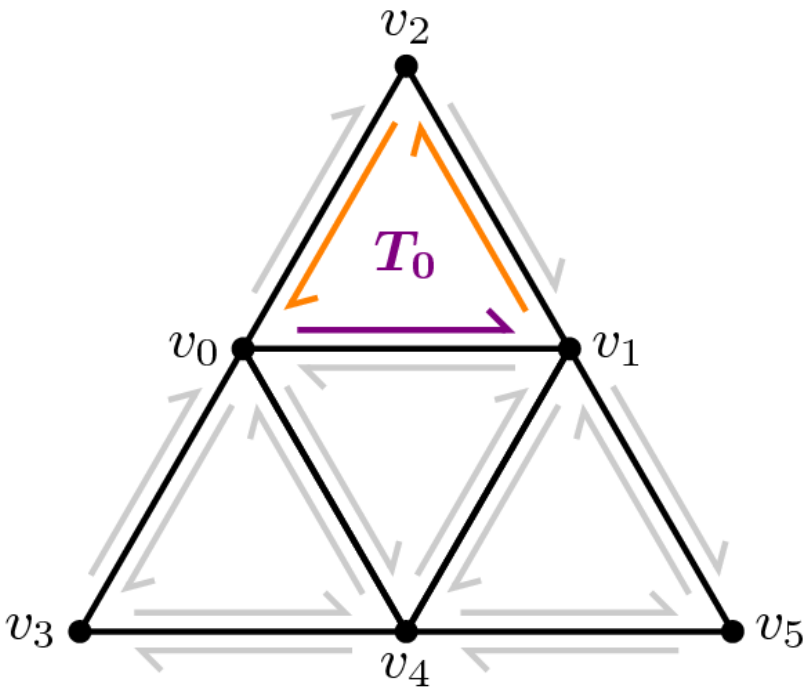
- Iteration over all edges incident to a vertex v_0
- Iterator increment: `he = he->next->opposite`
- Iterator dereferencing: `grid.entity(he->edge)`



Half-Edge Iteration

Provide chains of entities by defining an iterator

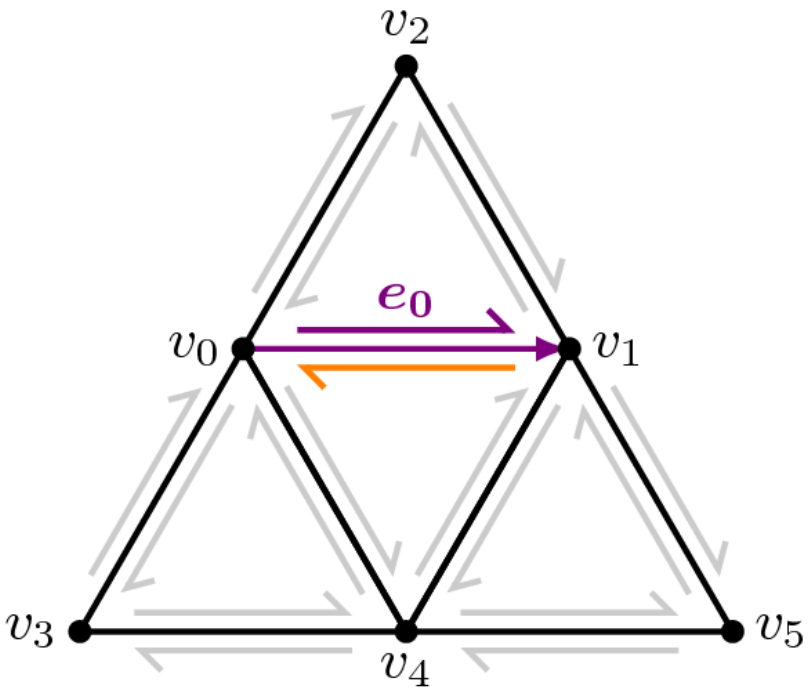
- Iteration over all edges incident to an element T_0
- Iterator increment: `he = he->next`
- Iterator dereferencing: `grid.entity(he->edge)`



Half-Edge Iteration

Provide chains of entities by defining an iterator

- Iteration over all elements incident to an edge e_0
- Iterator increment: `he = he->opposite`
- Iterator dereferencing: `grid.entity(he->face)`



Range generators

- To access the k-chains (ranges of entities) use generator functions
- Syntax similar to grid-view range generators

```
Matrix A(grid.size(1), grid.size(1)); // sparse matrix with num_rows=#edges

auto const& I = grid.leafIndexSet();

// iterator over all edges in the grid
for (auto const& e : edges(grid.leafGridView())) {

  // iterate over all vertices incident to edge e
  for (auto const& v : vertices(e)) {
    auto factor1 = v.sign(e) * grid.dual_volume(v);

    // iterate over all edges incident to vertex v
    for (auto const& e_ : edges(v)) {
      auto factor2 = v.sign(e_) * grid.dual_volume(e_) / grid.volume(e_);

      A(I.index(e), I.index(e_)) += factor1 * factor2;
    }
  }
}
```

Entitys and Geometry

- Extension of the Dune Entity and Geometry interface
- Entity: Relative orientation of entities:

```
// sign(v,e) = { +1 ... e points to v,
//               -1 ... e points away from v,
//                0 ... v not connected to e }
// sign(e,f) = { +1 ... f is on the left side of e,
//               -1 ... f is on the right side of e,
//                0 ... otherwise }
template <int codim>
int sign(Entity<codim,dim,Grid> const& that) const;
```

- Geometry: `center()` gives circumcenter, and new method `dual_volume()`:

```
GlobalCoords center() const;    //< Return the circumcenter of this entity
float_type volume() const;      //< Return the volume of this entity
float_type dual_volume() const; //< Return the volume of the dual entity
```

- Geometry: For edge entities the edge-vector can be extracted:

```
/// return the vector pointing from vertex v0 to v1 of the edge
GlobalCoords vector() const;

/// return the vector pointing from vertex c(T0) to c(T1)
GlobalCoords dual_vector() const;
```

Conclusion and Outlook

Results

- Implemented a half-edge datastructure to provide ranges over incident entities
- Calculations of circumcenter and dual volumes (can be cached)
- Realization of standard differential operators in DEC notation
- Result: Generalization for Finite-Difference / Finite-Volume to unstructured simplex grids
- Surface grids work out-of-the-box
- Implemented as Dune grid-adaptor

TODO

- 3D generalization: half-faces
- Relax well-centeredness requirement:
  - weighted triangulation
  - allow negative volumes (and circumcenters outside of the entity)
- (Local) refinement and adaption of the half-edge structure
- Optimize incidence iterators $\rightarrow$ increase locality
- Parallelization
- Provide full interface for Dune-grids

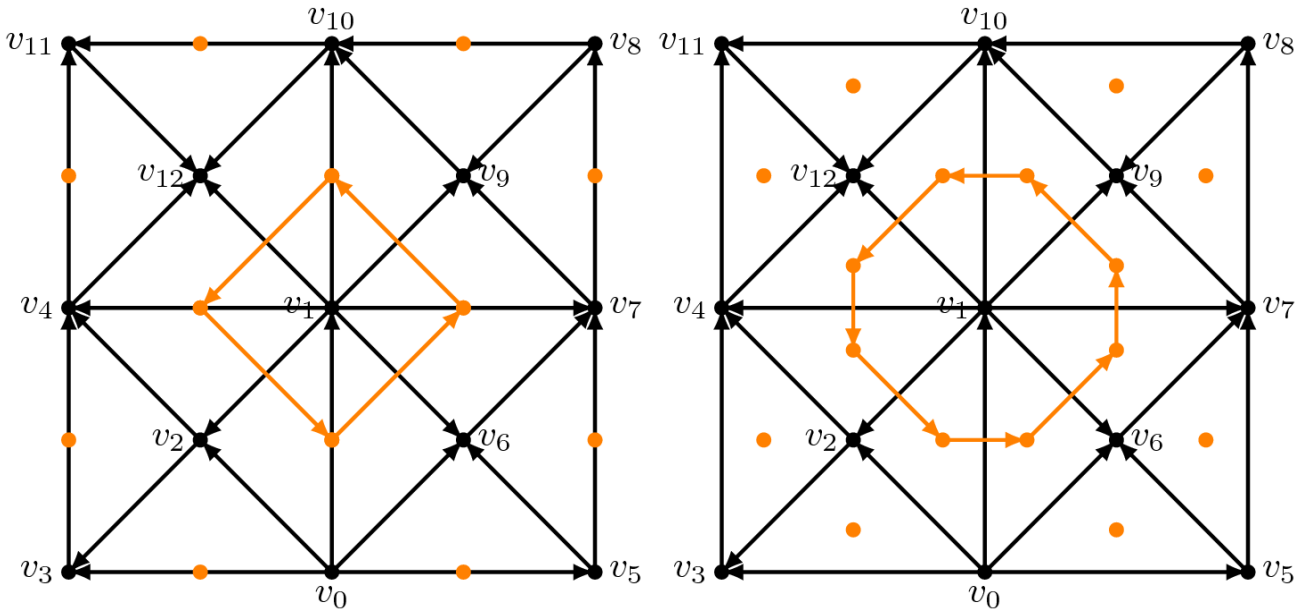
Questions?

Weighted triangulation

- Assign a weight to each vertex
- Move Circumcenter depending on these weights orthogonal to edges:

$$c^* := c(\sigma) + \frac{1}{2} \sum_i w_i \nabla \lambda_i$$

with λ_i linear lagrange basis functions (barycentric coordinates).



Weighted triangulation

- Calculation of weight by Poisson equation:

$$\Delta w = \nabla \cdot (c(\sigma) - b(\sigma))$$

with $b(\sigma)$ the barycenter of the element σ.

- Rhs-vector defined element-wise:

$$\langle \nabla w, \nabla \theta \rangle_{\Omega_h} = \sum_{\sigma \in \Omega_h} \langle c(\sigma) - b(\sigma), \nabla \theta \rangle_{\sigma}, \quad \forall \theta$$

- Idea is based on

F. de Goes, P. Memari, P. Mullen, M. Desbrun: "Weighted Triangulation for Geometry Processing" (2014)

- Problem: For consistency of the scheme we need modification (different Hodge-Star operator)

Half-Edge versus Intersection

- Half-edges provide all information to implement `Dune::Intersection`:

```
/// Return true if intersection is with interior or exterior boundary
bool boundary() const
{
  return he.face == invalid || he.opposite->face == invalid;
}

/// Return true if intersection is shared with another element.
bool neighbour() const { return he.opposite->face != invalid; }

/// Return Entity on the inside of this intersection.
/// That is the Entity where we started this.
Entity inside() const { return grid.entity(he.face); }

/// Return Entity on the outside of this intersection ...
Entity outside() const { return grid.entity(he.opposite->face); }
```

Half-Edge versus Intersection

- Half-edges provide all information to implement `Dune::Intersection`:

```
/// Geometrical information about this intersection in
/// local coordinates of the inside() entity.
LocalGeometry geometryInInside() const { return {he}; }

/// Geometrical information ... of the outside() entity.
LocalGeometry geometryInOutside() const { return {*he.opposite}; }

/// Geometrical information about the intersection in global coordinates.
Geometry geometry() const { return {grid.halfEdge(he.edge)}; }

/// Local index of codim 1 entity in the inside() entity where
/// intersection is contained in
int indexInInside() const { /* little bit more complicated */ }

/// Local index ... in the outside() entity
int indexInOutside() const { /* little bit more complicated */ }

/// Return unit outer normal
GlobalCoords centerUnitOuterNormal()
{
  return grid.entity(he.edge).geometry().dual_vector();
}
```