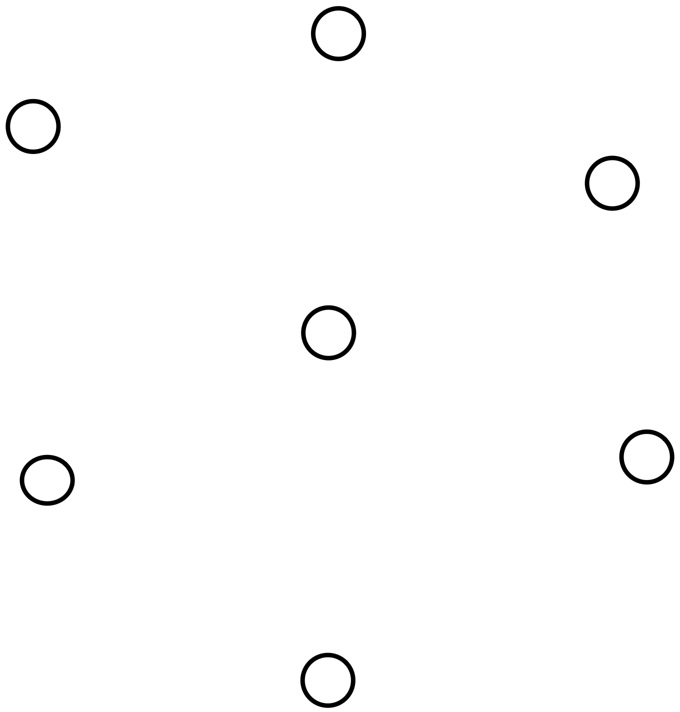
Administrative Stuff

- Exercise grade are 25% of the final grade
- Six programming exercises
- One assignment every two week
- In odd weeks, we discuss solutions
- You can work alone or in group of two
- Send solutions to surgem@inf.ethz.ch

Polygonal mesh processing

$$\mathcal{M} = (\{\mathbf{v}_i\}, \{e_j\}, \{f_k\})$$

geometry $\mathbf{v}_i \in \mathbb{R}^3$

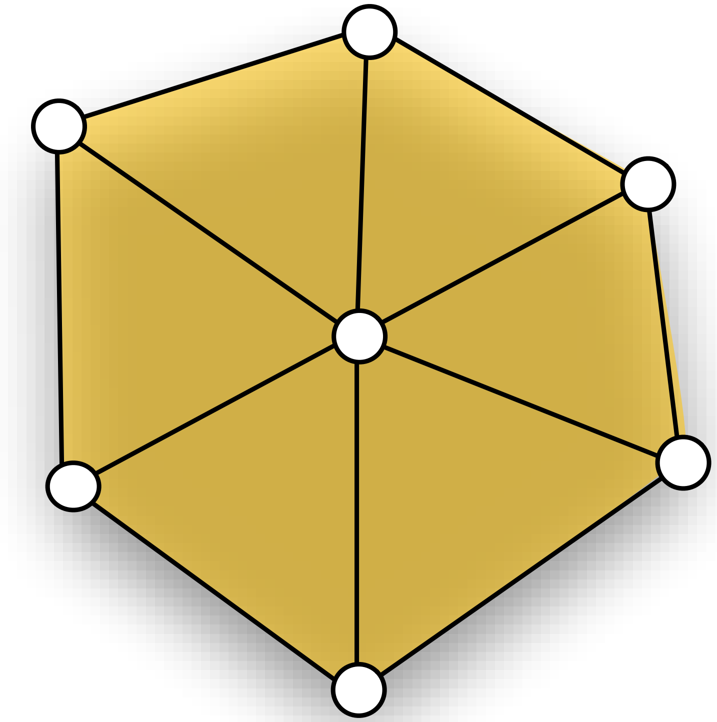


Polygonal mesh processing

$$\mathcal{M} = (\{\mathbf{v}_i\}, \{e_j\}, \{f_k\})$$

geometry $\mathbf{v}_i \in \mathbb{R}^3$

topology $e_i, f_i \subset \mathbb{R}^3$



How do we represent geometric entities?

Requirements

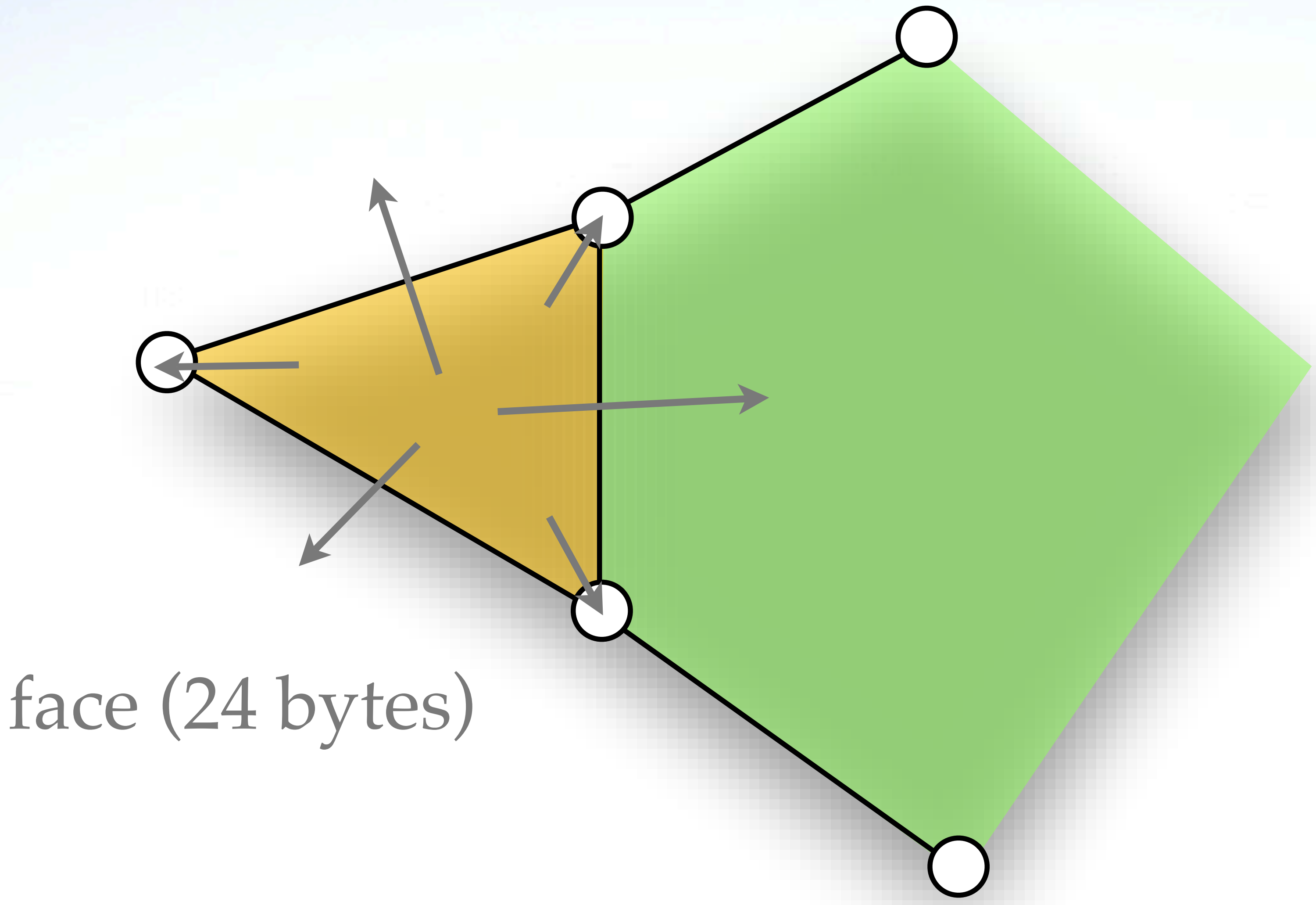
- Random access to vertices, edges and faces
- Fast mesh traversal
- Fast neighborhood query
- Memory efficiency

Different data structures
Different topological data storage

- Two main approaches: Face and edge-based (since they encode connectivity)

- Design decision ~ Memory / speed trade-off

Why not face-based data structure?



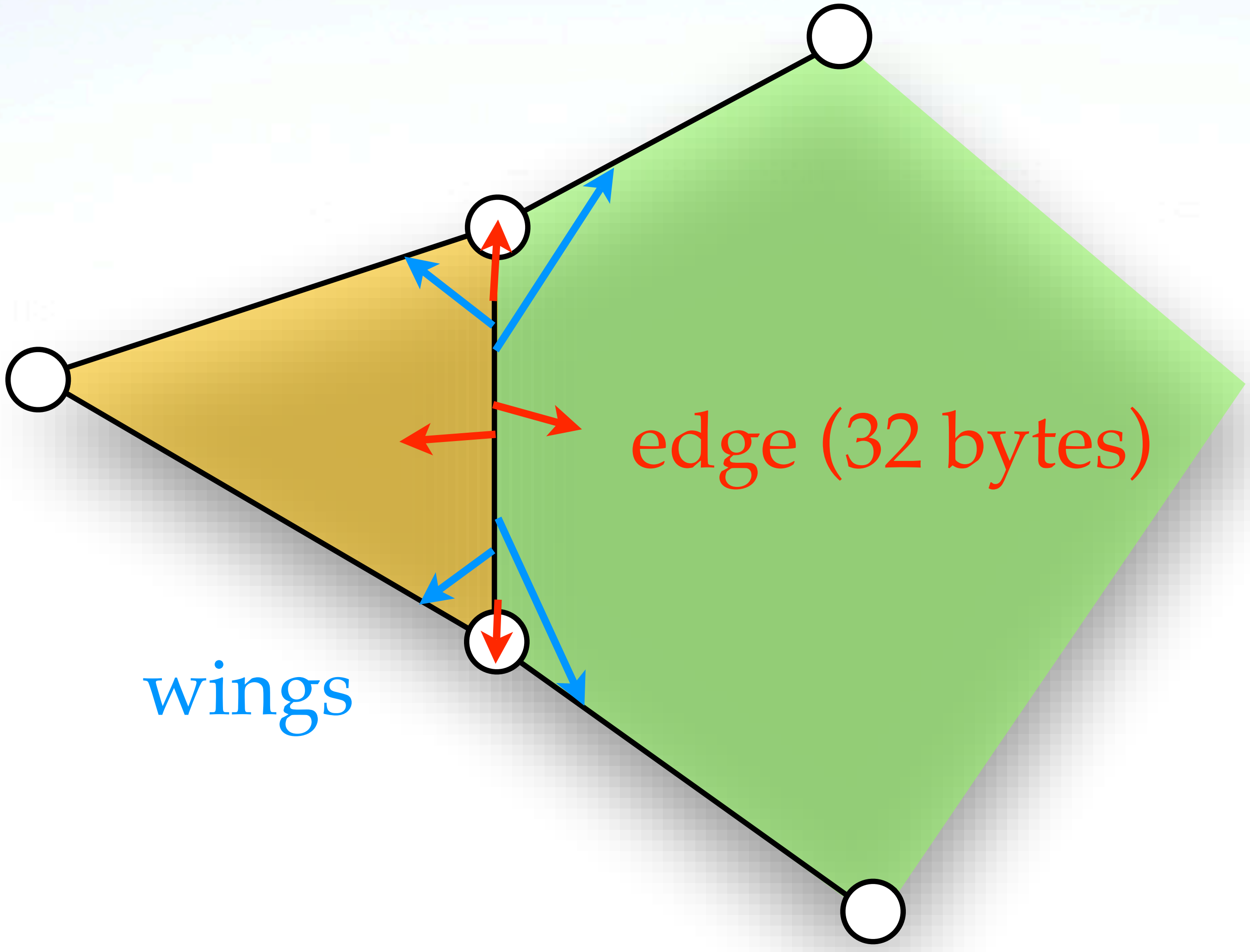
Arbitrary polygons → special case handling

Edges always have the same topological structure

↓

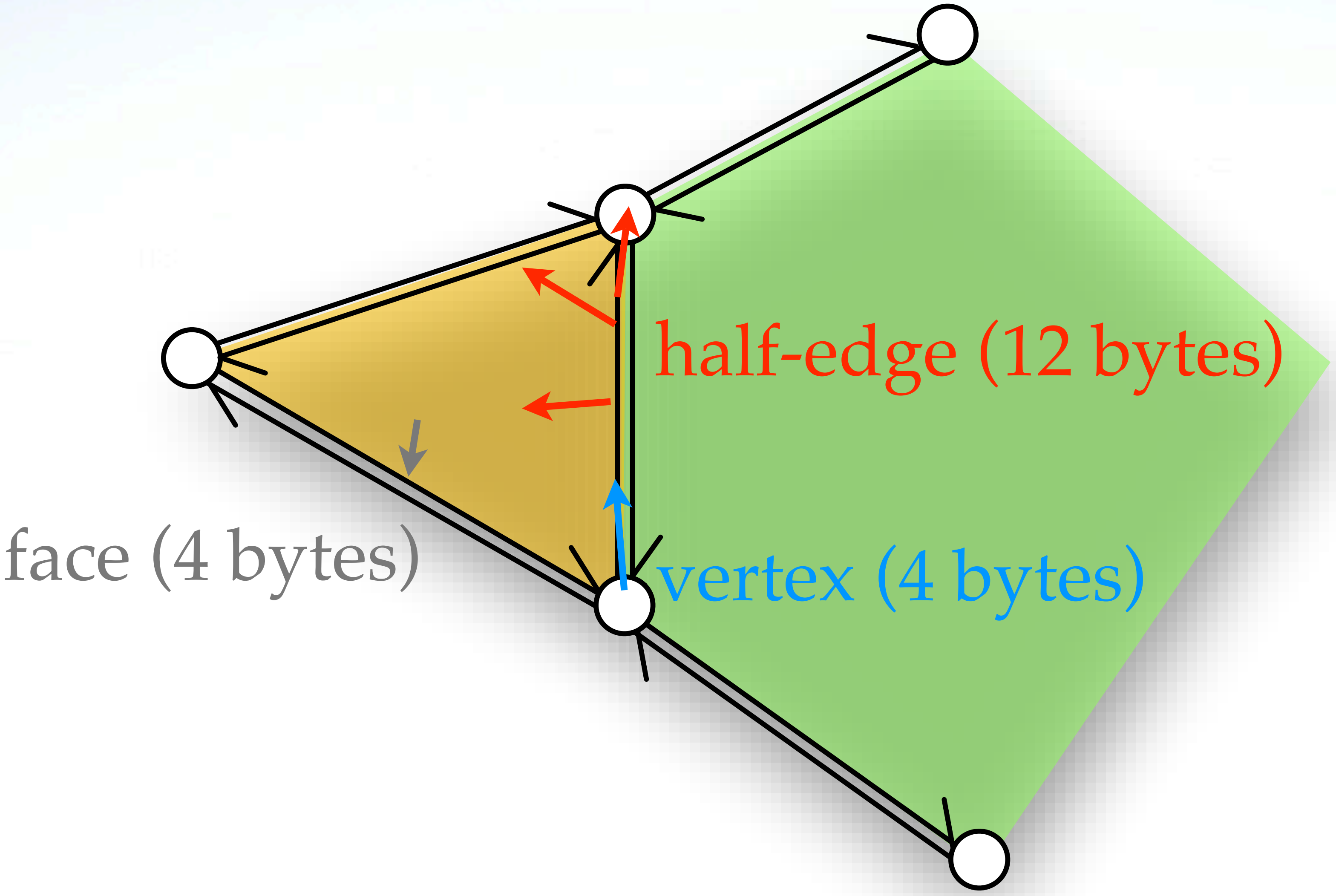
Efficient handling of polygons with variable valence

Why not winged-edge data structure?



Edges do not encode orientation → special case handling for neighborhood traversal

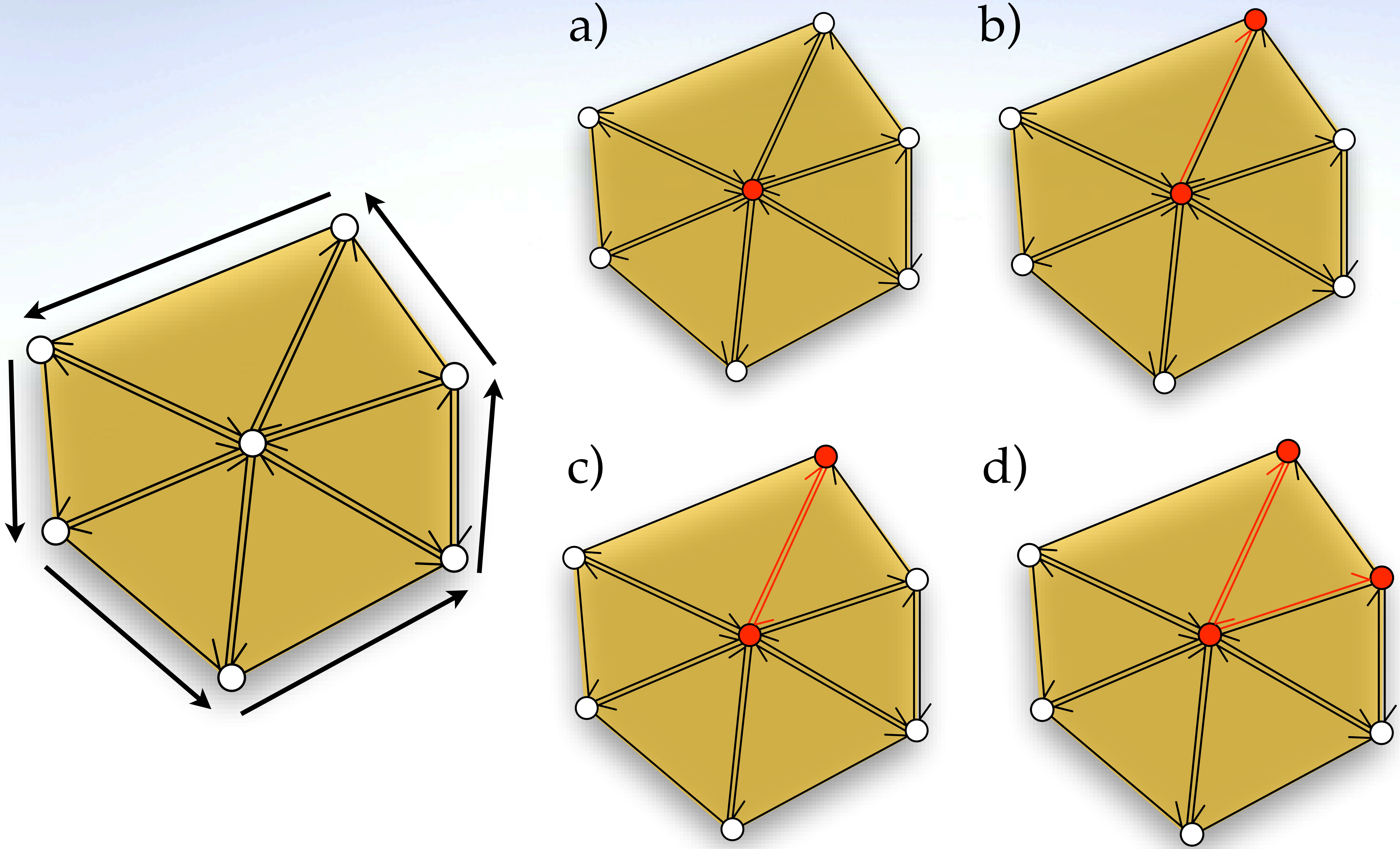
Why half-edge data structure?



Stores main connectivity informations

No run-time overhead due to arbitrary faces

One-ring neighborhood traversal in O(1)



opposite half-edge implicitly encoded

*Open*Mesh 1.0

- ACG – RWTH Aachen
- C++ library
- Implements **half-edge** data structure
- Integrated **basic geometric operations**
- 3-D model file reader/writer

why *Open*Mesh ?

Flexible

- Random access to vertices, edges, and faces
- Arbitrary scalar types
- Arrays or lists as underlying kernels

Efficient in space and time

- Dynamic memory management for array-based meshes (not in CGAL)

- Extendable to specialized kernels for non-manifold meshes (not in CGAL)

It is easy to use...

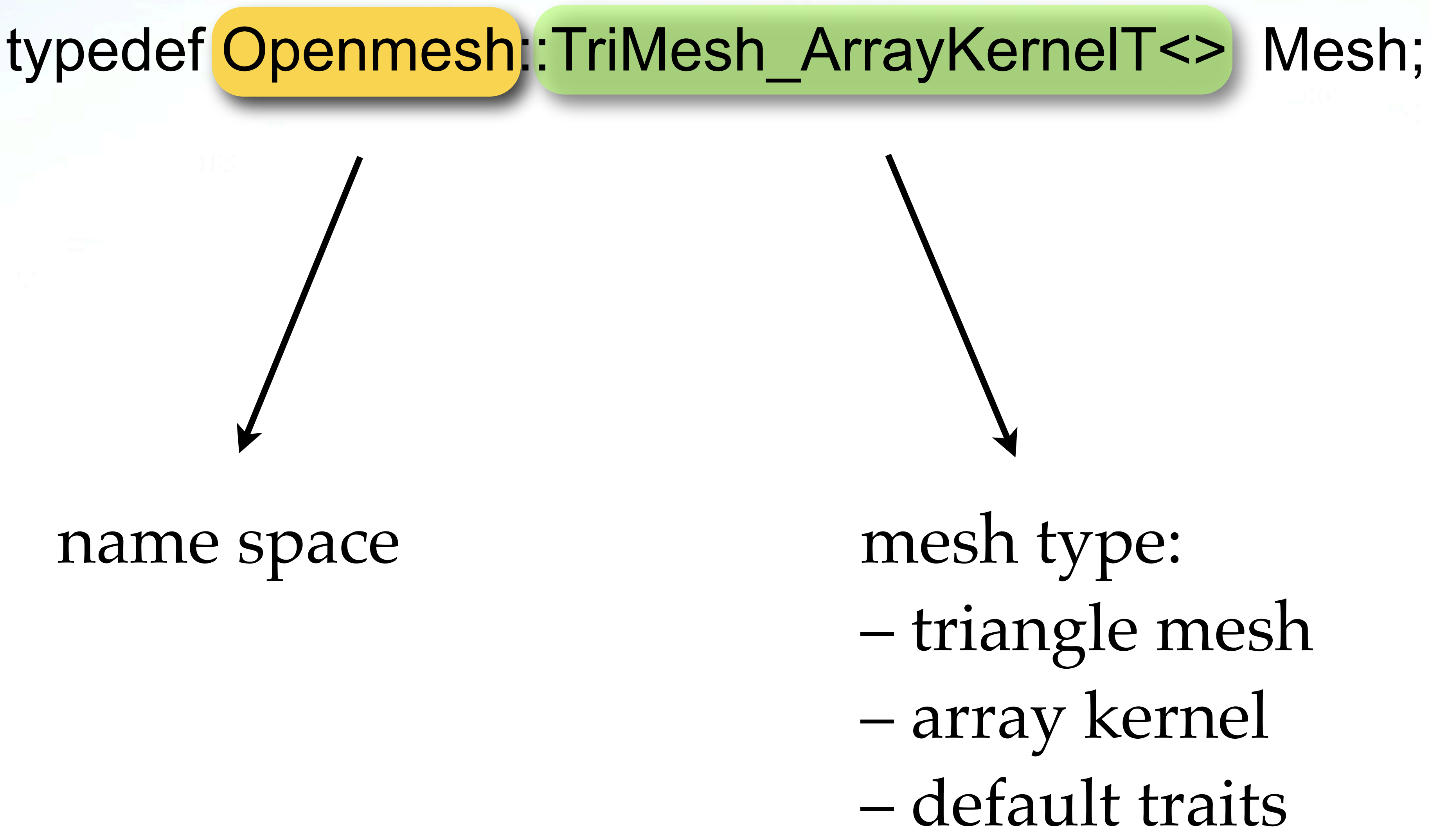
Integrated geometric operations

```
OpenMesh::Vec3f x,y,n,crossproductXY;
...
l = (x-y).length();

n = x.normalize();
scalarProductXY = (x | y);
crossProductXY = x % y;
...
```

Mesh definition

```
#include <OpenMesh/Core/IO/MeshIO.hh>
#include <OpenMesh/Core/Mesh/Types/TriMesh_ArrayKernelT.hh>
```



```
typedef Openmesh::TriMesh_ArrayKernelT<>  Mesh;
```

name space

mesh type:
– triangle mesh
– array kernel
– default traits

Loading and writing a mesh

```
Mesh * myMesh;

OpenMesh::IO::Options readOptions;

OpenMesh::IO::read_mesh(*myMesh,”/path/to/bunny.off”,readOptions)
```

reader/writer settings:

– enable vertex normals/colors / texture coordinates?

– enable face normals/colors?

Adding attributes

```
Mesh * myMesh;

OpenMesh::IO::Options readOptions;

OpenMesh::IO::read_mesh(*myMesh,”/path/to/bunny.off”,readOptions)

if(!readOptions.check(OpenMesh::IO::Options::FaceNormal))
{
   myMesh->update_face_normals();
}

if(! readOptions.check(OpenMesh::IO::Options::VertexNormal))
{
   myMesh->update_vertex_normals();
}
```

Iterating over vertices

```
typedef Openmesh::TriMesh_ArrayKernelT<> Mesh;
Mesh * myMesh;
```

```
Mesh::VertexIter vIt,vBegin,vEnd;

vBegin = myMesh->vertices_begin();
vEnd = myMesh->vertices_end();

for( vIt = vBegin ; vIt != vEnd; ++vIt )
{
   doSomethingWithVertex(vIt.handle());
}
```

mesh processing

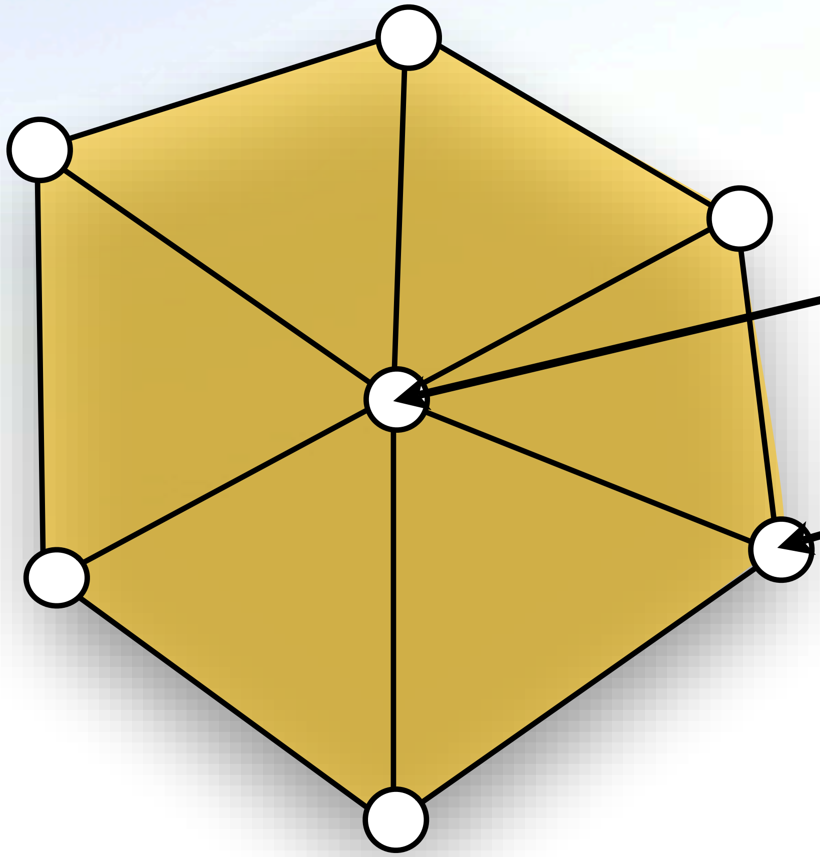
Iterating over faces

Mesh::VertexIter → Mesh::FaceIter

vertices_begin() → faces_begin()

vertices_end() → faces_end()

Circulating over faces around a vertex



```
Mesh::VertexIter vIt,vBegin,vEnd;

vBegin = myMesh->vertices_begin();
vEnd = myMesh->vertices_end();

for( vIt = vBegin ; vIt != vEnd; ++vIt )
{
  Mesh::VertexFaceIter vfIt,vfBegin;
  vfBegin = myMesh->vf_iter(vIt);

  for( vfIt = vfBegin ; vfIt ; ++vfIt)
  {
    doSomethingWithFace(vfIt.handle());
  }
}
```

returns false after a complete circulation round

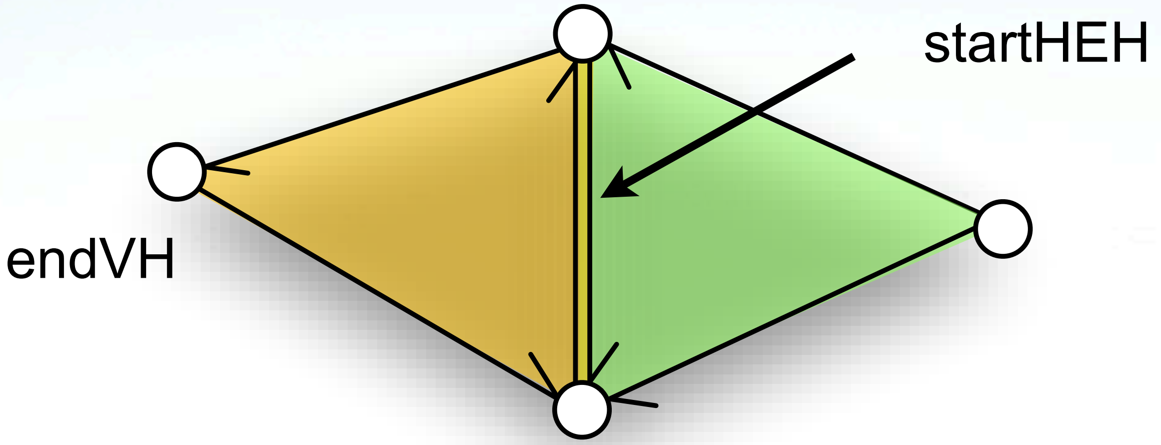
Triangle geometry

```
void analyzeTriangle(OpenMesh::FaceHandle & _fh)
{
      OpenMesh::Vec3f pointA,pointB,pointC;
      Mesh::ConstFaceVertexIter cfvIt;

      cfvIt = myMesh->cfv_iter(_fh);
      pointA = myMesh->point(cfvIt.handle());
      pointB = myMesh->point((++cfvIt).handle());
      pointC = myMesh->point((++cfvIt).handle());

      perimeter(pointA,pointB,pointC);
      area(pointA,pointB,pointC)
}
```

Neighborhood access in O(1)



```
OpenMesh::VertexHandle endVH;
OpenMesh::HalfEdgeHandle startHEH,oppositeHEH,nextHEH;

startHEH = hehIt.handle();

oppositeHEH = myMesh->opposite_halfedge_handle(startHEH);
nextHEH = myMesh->next_halfedge_handle(oppositeHEH);
endVH =  myMesh->to_vertex_handle(nextHEH);
```

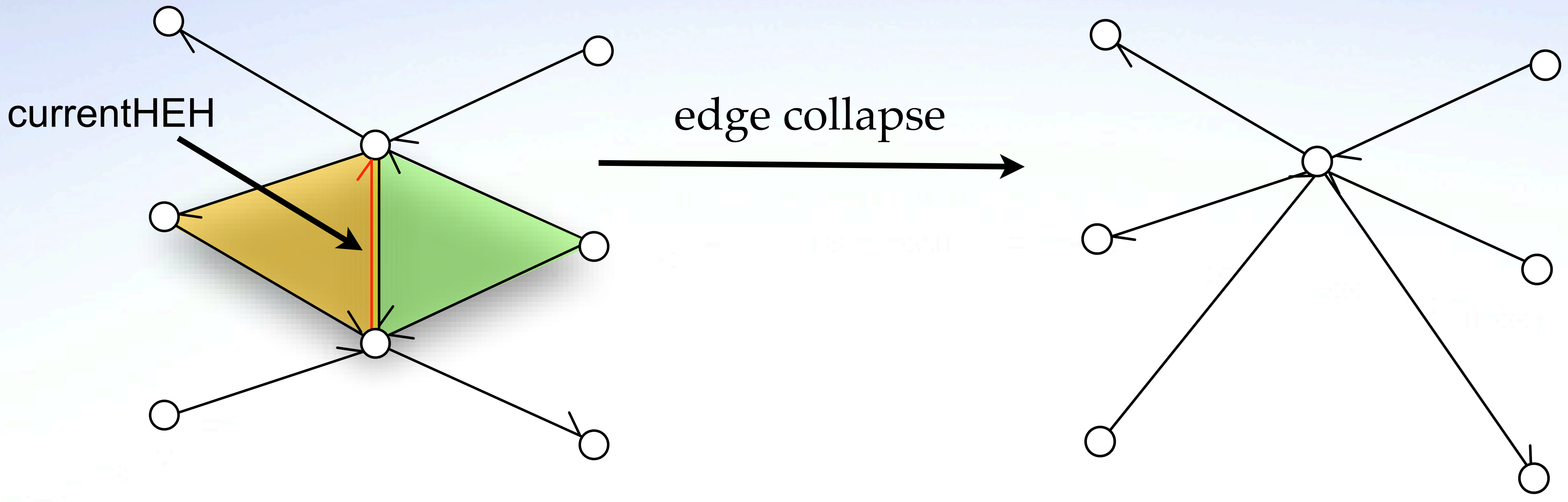
mesh topology
involved

Modifying the geometry

```
for( vIt = vBegin ; vIt != vEnd; ++vIt )
{
   scale(vIt.handle(),2.0);
}
```

```
void scale(OpenMesh::VertexHandle & _vh,double _alpha)
{
        OpenMesh::Vec3f newCoordinate;
        newCoordinate = myMesh->point(_vh);
        myMesh->set_point(_vh, newCoordinate * _alpha);
}
```

Changing the topology



```
myMesh->request_vertex_status();
myMesh->request_edge_status();
myMesh->request_face_status();
```

```
OpenMesh::HalfedgeHandle currentHEH = helt.handle();
```

```
myMesh->collapse(currentHEH);
myMesh->garbage_collection();
```

Customizing the Mesh

- Face type with predefined array kernel

```
typedef Openmesh::TriMesh_ArrayKernelT<> Mesh;
typedef Openmesh::PolyMesh_ArrayKernelT<> Mesh;
```

- Traits

predefined attributes:
– normals / colors
– coordinate types: 2-D, 3-D, ..., *n*D
– scalar types: float, double, ...

custom attributes: centerOfGravity, ...

Traits – static customization

```
#include <OpenMesh/Core/IO/MeshIO.hh>
#include <OpenMesh/Core/Mesh/Types/TriMesh_ArrayKernelT.hh>
```

```
struct myMeshTraits : public OpenMesh::DefaultTraits
{
    typedef OpenMesh::Vec4f Color;

    VertexAttributes (
                OpenMesh::Attributes::Normal |
                OpenMesh::Attributes::Color);

    FaceAttributes (
                OpenMesh::Attributes::Normal |
                OpenMesh::Attributes::Color);

}
```

```
typedef Openmesh::TriMesh_ArrayKernelT<myMeshTraits> Mesh;
```

Dynamic customization of predefined attributes

```
typedef Openmesh::TriMesh_ArrayKernelT<> Mesh;

Mesh * myMesh;

... // load file into myMesh

myMesh->request_vertex_normals();
myMesh->request_vertex_colors();
myMesh->request_face_normals();

...

myMesh->set_color(currentVH,Mesh::Color(0,0,255));

blueColor = myMesh->color(currentVH);
```

Dynamic customization of custom attributes

```
OpenMesh::FPropHandleT<bool> marked;
myMesh->add_property(marked);

for(fIt = fBegin; fIt != fEnd; ++fIt)
{
     if(shouldMark(fIt))
       myMesh->property(marked,fIt) = true;
     else
       myMesh->property(marked,fIt) = false;
}

for(fIt = fBegin; fIt != fEnd; ++fIt)
{
  if(myMesh->property(marked,fIt))
     doSomething(fIt);
}
```

Three important links

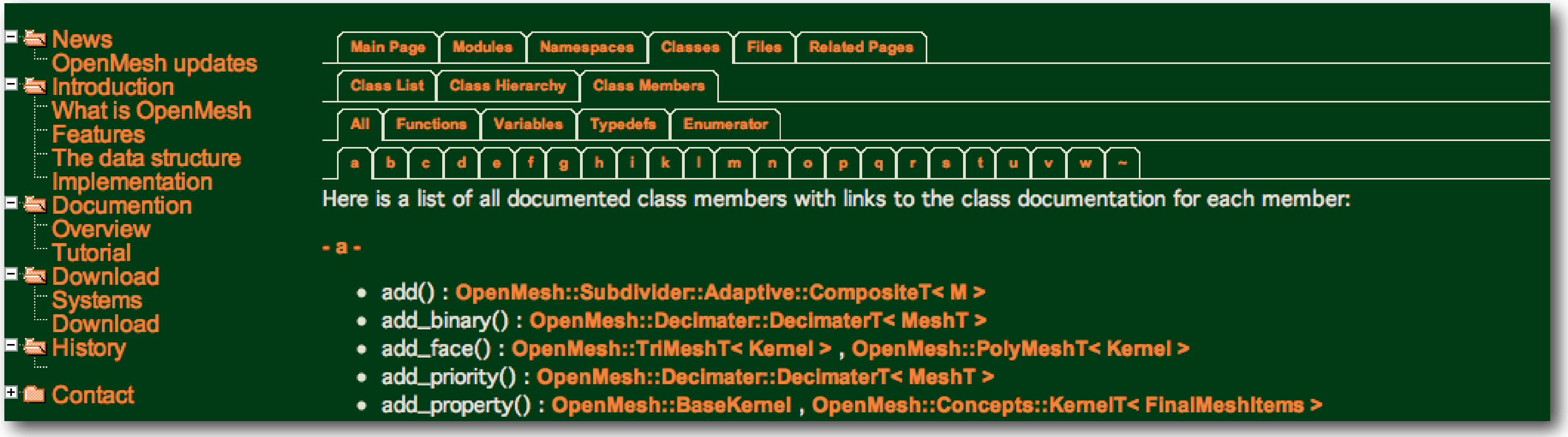
www.openmesh.org → Overview

www.openmesh.org → Tutorial

www.openmesh.org → Documentation

→ Classes

→ Class Members



Here is a list of all documented class members with links to the class documentation for each member:

- a -

- add() : OpenMesh::Subdivider::Adaptive::CompositeT< M >
- add_binary() : OpenMesh::Decimater::DecimaterT< MeshT >
- add_face() : OpenMesh::TriMeshT< Kernel > , OpenMesh::PolyMeshT< Kernel >
- add_priority() : OpenMesh::Decimater::DecimaterT< MeshT >
- add_property() : OpenMesh::BaseKernel , OpenMesh::Concepts::KernelT< FinalMeshItems >

Further readings

- Documention: http://www.openmesh.org/
- OpenMesh – a generic and efficient polygon mesh data structure [Botsch et al. 2002]

?

hao@inf.ethz.ch
balint@inf.ethz.ch