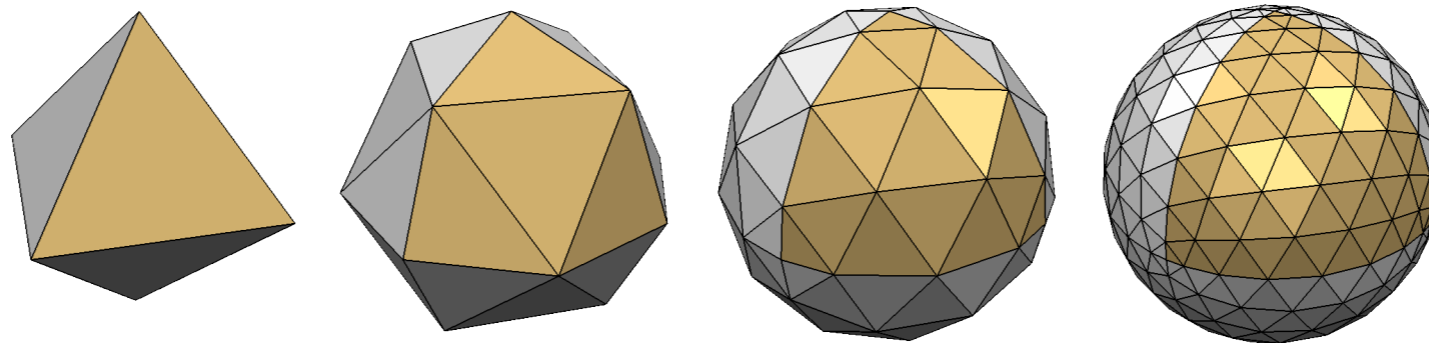


Surface_mesh



Eduard Zell

Computer Graphics & Geometry Processing

Build a Tetrahedron

```
#include <surface_mesh/Surface_mesh.h>

int main(void)
{
    // instantiate a Surface_mesh object
    Surface_mesh mesh;

    // instantiate 4 vertex handles
    Surface_mesh::Vertex v0,v1,v2,v3;

    // add 4 vertices
    v0 = mesh.add_vertex(Point(0,0,0));
    v1 = mesh.add_vertex(Point(1,0,0));
    v2 = mesh.add_vertex(Point(0,1,0));
    v3 = mesh.add_vertex(Point(0,0,1));

    // add 4 triangular faces
    mesh.add_triangle(v0,v1,v3);
    mesh.add_triangle(v1,v2,v3);
    mesh.add_triangle(v2,v0,v3);
    mesh.add_triangle(v0,v2,v1);

    std::cout << "vertices: " << mesh.n_vertices() << std::endl;
    std::cout << "edges: " << mesh.n_edges() << std::endl;
    std::cout << "faces: " << mesh.n_faces() << std::endl;

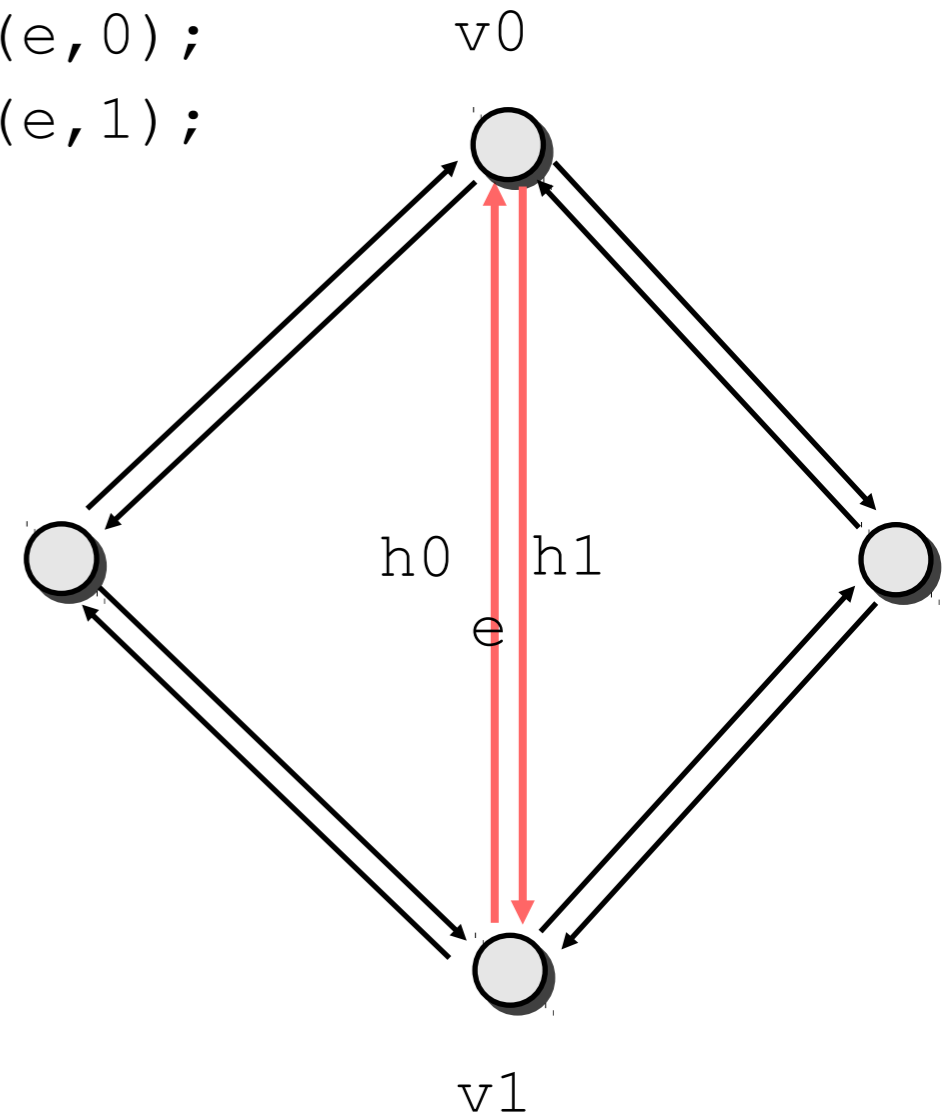
    return 0;
}
```

Geometry: 3D Vectors

```
Point x(0,0,0), y(1,2,3);  
Scalar length = (x-y).norm();  
  
Point n = cross(x,y);           // cross product  
n.normalize();                  // n.norm()==1  
  
Scalar dp = dot(x,y);          // dot product  
Scalar angle = acos( dp / x.norm() / y.norm() );
```

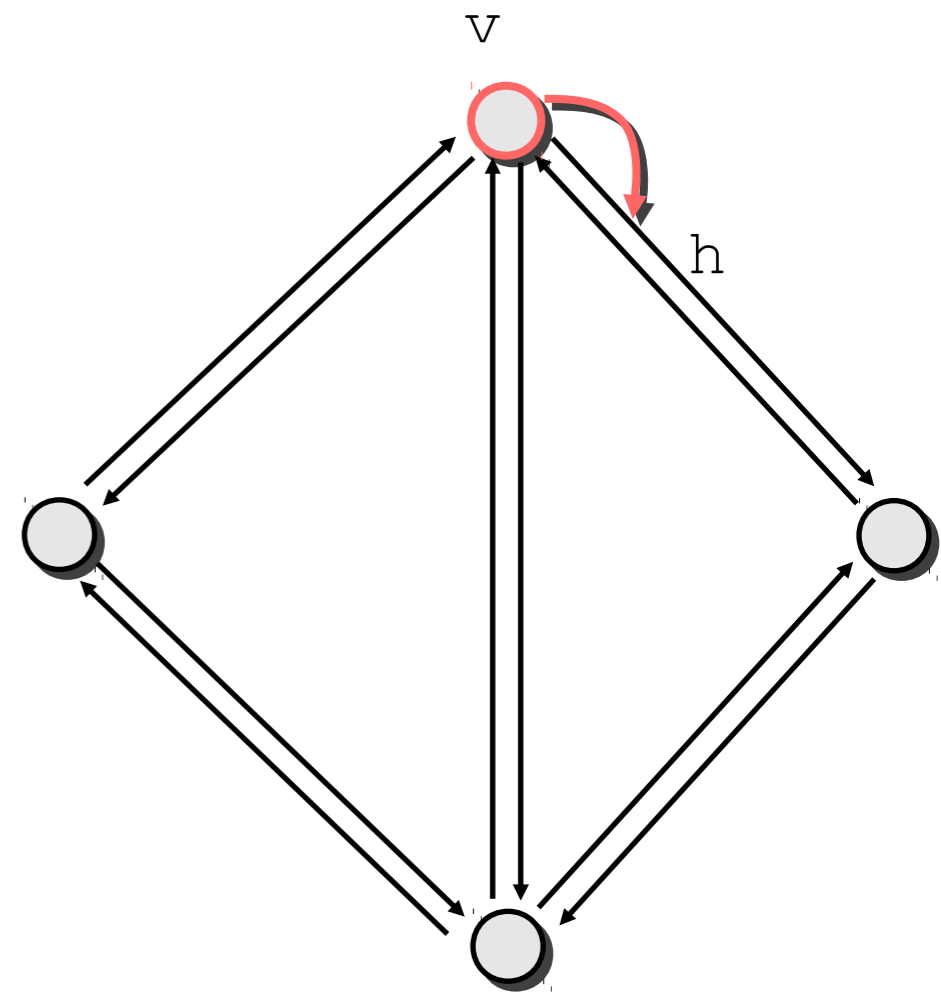

Edge Connectivity

```
Surface_mesh::Edge e;  
Surface_mesh::Halfedge h0 = mesh.halfedge(e, 0);  
Surface_mesh::Halfedge h1 = mesh.halfedge(e, 1);  
Surface_mesh::Vertex v0 = mesh.vertex(e, 0);  
Surface_mesh::Vertex v1 = mesh.vertex(e, 1);
```



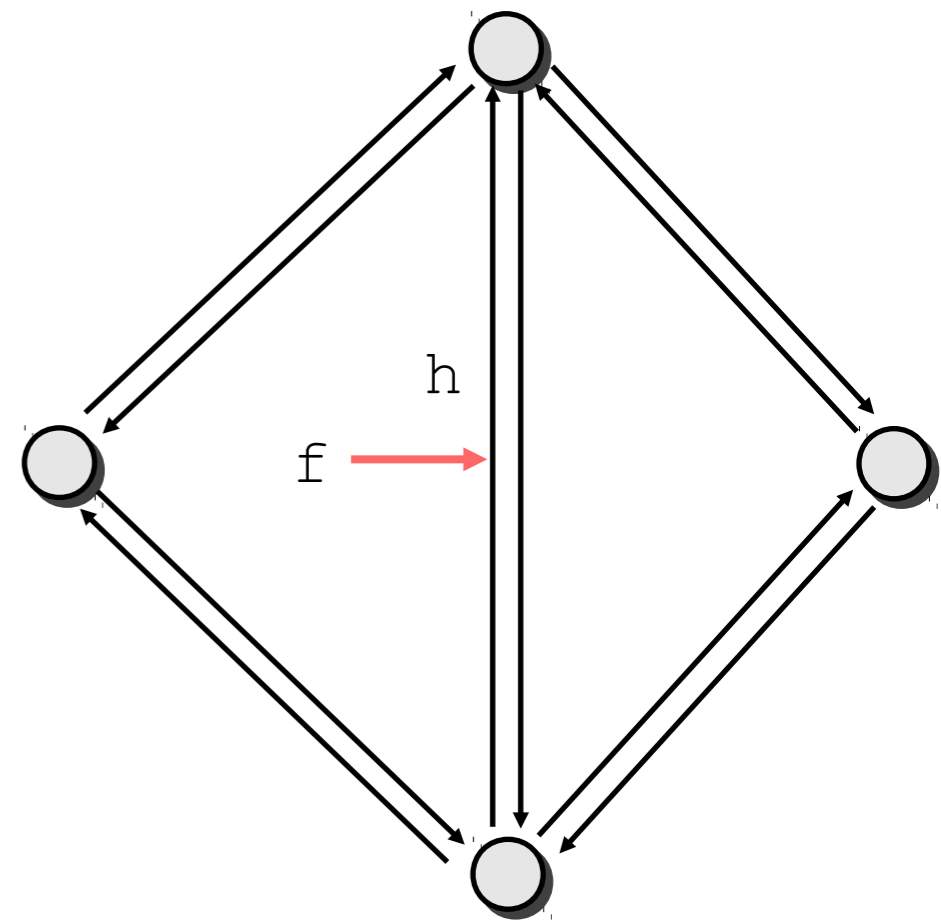
Vertex Connectivity

```
Surface_mesh::Vertex    v;  
Surface_mesh::Halfedge h = mesh.halfedge(v);
```



Face Connectivity

```
Surface_mesh::Face      f;  
Surface_mesh::Halfedge h = mesh.halfedge(f);
```



File I/O

```
#include <surface_mesh/Surface_mesh.h>

int main(int argc, char** argv)
{
    // instantiate a Surface_mesh object
    Surface_mesh mesh;

    // read a mesh specified as the first command line argument
    mesh.read(argv[1]);

    // ...
    // do fancy stuff with the mesh
    // ...

    // write the mesh to the file specified as second argument
    mesh.write(argv[2]);

    return 0;
}
```


Iterators

- Iterate over all vertices

```
Surface_mesh::Vertex_iterator vit;  
  
for (vit = mesh.vertices_begin();  
     vit != mesh.vertices_end();  
     ++vit)  
{  
    ...  
}
```

- Analogous for halfedges, edges, faces

Iterators with C++11

- Iterate over all vertices

```
for (auto v: mesh.vertices())  
{  
    ...  
}
```

- Analogous for halfedges, edges, faces

Vertex Circulators

- Enumerate all vertices, halfedges, faces around a center vertex

```
unsigned int valence(Surface_mesh::Vertex v)
{
    unsigned int count(0);

    Surface_mesh::Vertex_around_vertex_circulator vc = mesh.vertices(v), vc_end = vc;
    do
    {
        ++count;
    }
    while (++vc != vc_end);

    return count;
}
```

Vertex Circulators with C++11

- Enumerate all vertices, halfedges, faces around a center vertex

```
unsigned int valence(Surface_mesh::Vertex v)
{
    unsigned int count(0);

    auto vc = mesh.vertices(v), vc_end = vc;
    do
    {
        ++count;
    }
    while (++vc != vc_end);

    return count;
}
```

Face Circulators

- Enumerate all vertices or halfedges in a face

```
unsigned int valence(Surface_mesh::Face f)
{
    unsigned int count(0);

    Surface_mesh::Vertex_around_face_circulator fvit = mesh.vertices(f), fvend = fvit;
    do
    {
        ++count;
    }
    while (++fvit != fvend);

    return count;
}
```

Face Circulators with C++11

- Enumerate all vertices or halfedges in a face

```
unsigned int valence(Surface_mesh::Face f)
{
    unsigned int count(0);

    auto fvit = mesh.vertices(f), fvend = fvit;
    do
    {
        ++count;
    }
    while (++fvit != fvend);

    return count;
}
```

Custom Properties

```
Surface_mesh::Vertex_property<int> val = mesh.add_vertex_property<int>("v:val");
```

```
for (v_it=mesh.vertices_begin(); v_it!=mesh.vertices_end(); ++v_it)  
    val[*v_it] = compute_valence(v_it);
```

```
// do some work...
```

```
mesh.remove_vertex_property(val);
```

name
should be
unique

```
Vertex_propertyT<Point> points = mesh.vertex_property<Point>("v:point");
```

```
Vertex_propertyT<Normal> normals = mesh.vertex_property<Normal>("v:normal");
```

```
Vertex_propertyT<bool> locked = mesh.vertex_property<bool>("v:locked");
```

```
for (v_it=mesh.vertices_begin(); v_it!=mesh.vertices_end(); ++v_it)  
    if (!locked[*v_it])  
        points[*v_it] += normals[*v_it];
```

“get or add”
property

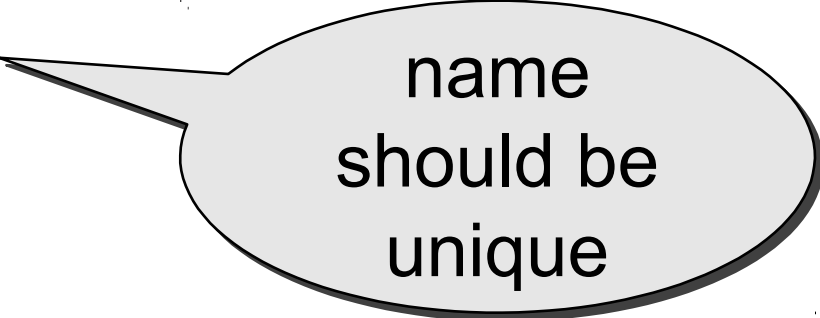
Custom Properties with C++11

```
auto = mesh.add_vertex_property<int>("v:val");
```

```
for (auto v: mesh.vertices())  
    val[v] = compute_valence(v);
```

```
// do some work...
```


```
mesh.remove_vertex_property(val);
```



name
should be
unique

```
auto points = mesh.vertex_property<Point>("v:point");  
auto normals = mesh.vertex_property<Normal>("v:normal");  
auto locked = mesh.vertex_property<bool>("v:locked");
```

```
for (auto v: mesh.vertices())  
    if (!locked[v])  
        points[v] += normals[v];
```



“get or add”
property

Compute Barycenter

```
Point barycenter(Surface_mesh& mesh)
{
    // get (pre-defined) property storing vertex positions
    Surface_mesh::Vertex_property<Point> points =
    mesh.get_vertex_property<Point>("v:point");

    // vertex iterator
    Surface_mesh::Vertex_iterator vit, vend = mesh.vertices_end();

    Point p(0,0,0);

    for (vit = mesh.vertices_begin(); vit != vend; ++vit)
    {
        // access point property like an array
        p += points[*vit];
    }

    p /= mesh.n_vertices();

    return p;
}
```

Compute Barycenter with C++11

```
Point barycenter(Surface_mesh& mesh)
{
    // get (pre-defined) property storing vertex positions
    auto points = mesh.get_vertex_property<Point>("v:point");

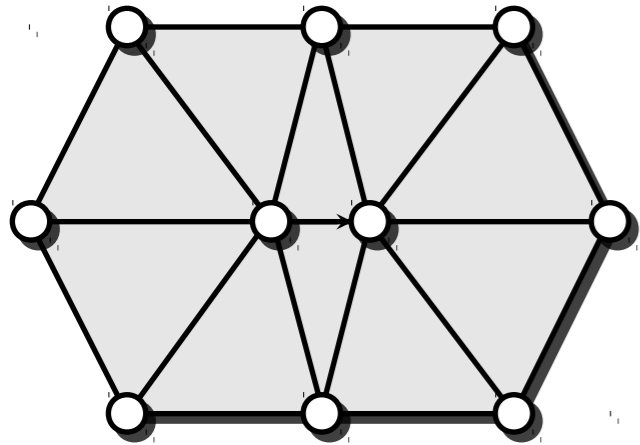
    Point p(0,0,0);

    for (auto v: mesh.vertices())
    {
        // access point property like an array
        p += points[v];
    }

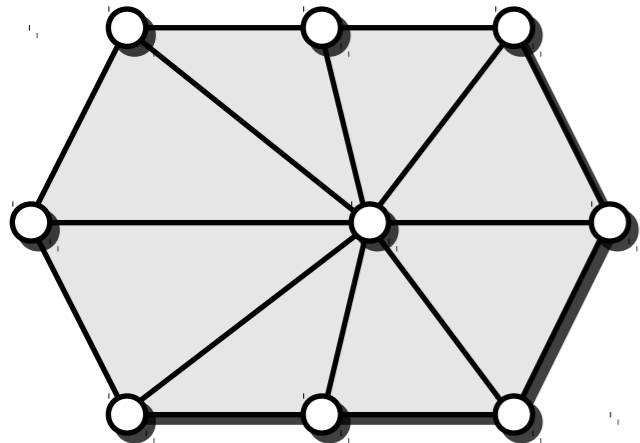
    p /= mesh.n_vertices();

    return p;
}
```

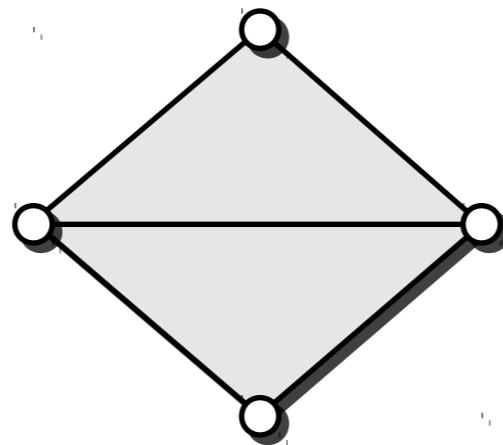
Change Topology



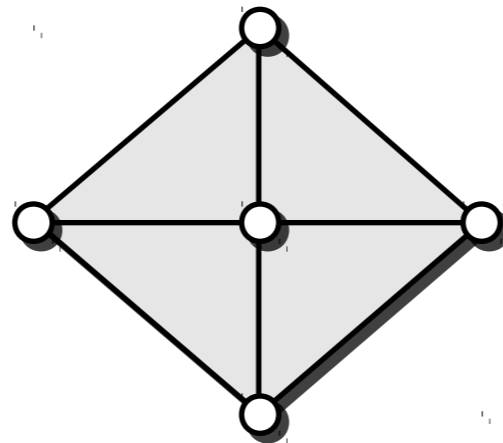
Halfedge
Collapse



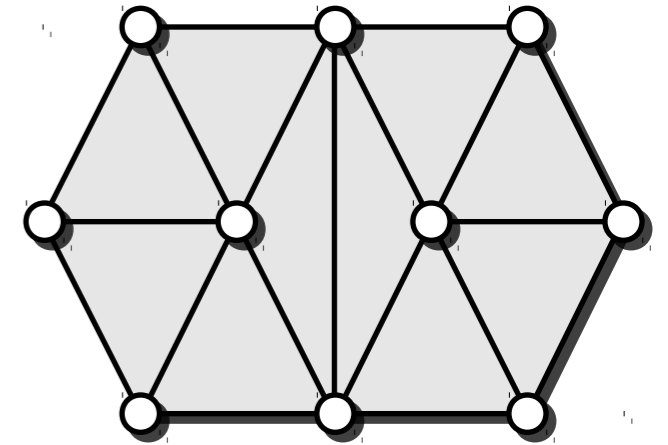
`mesh.collapse (Halfedge)`



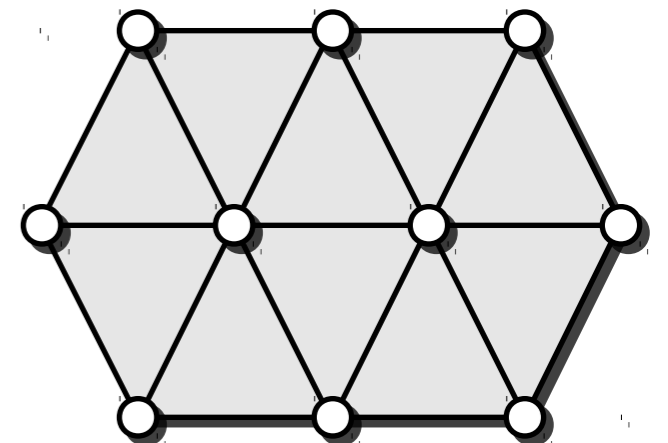
Edge
Split



`mesh.split (Edge, Point)`

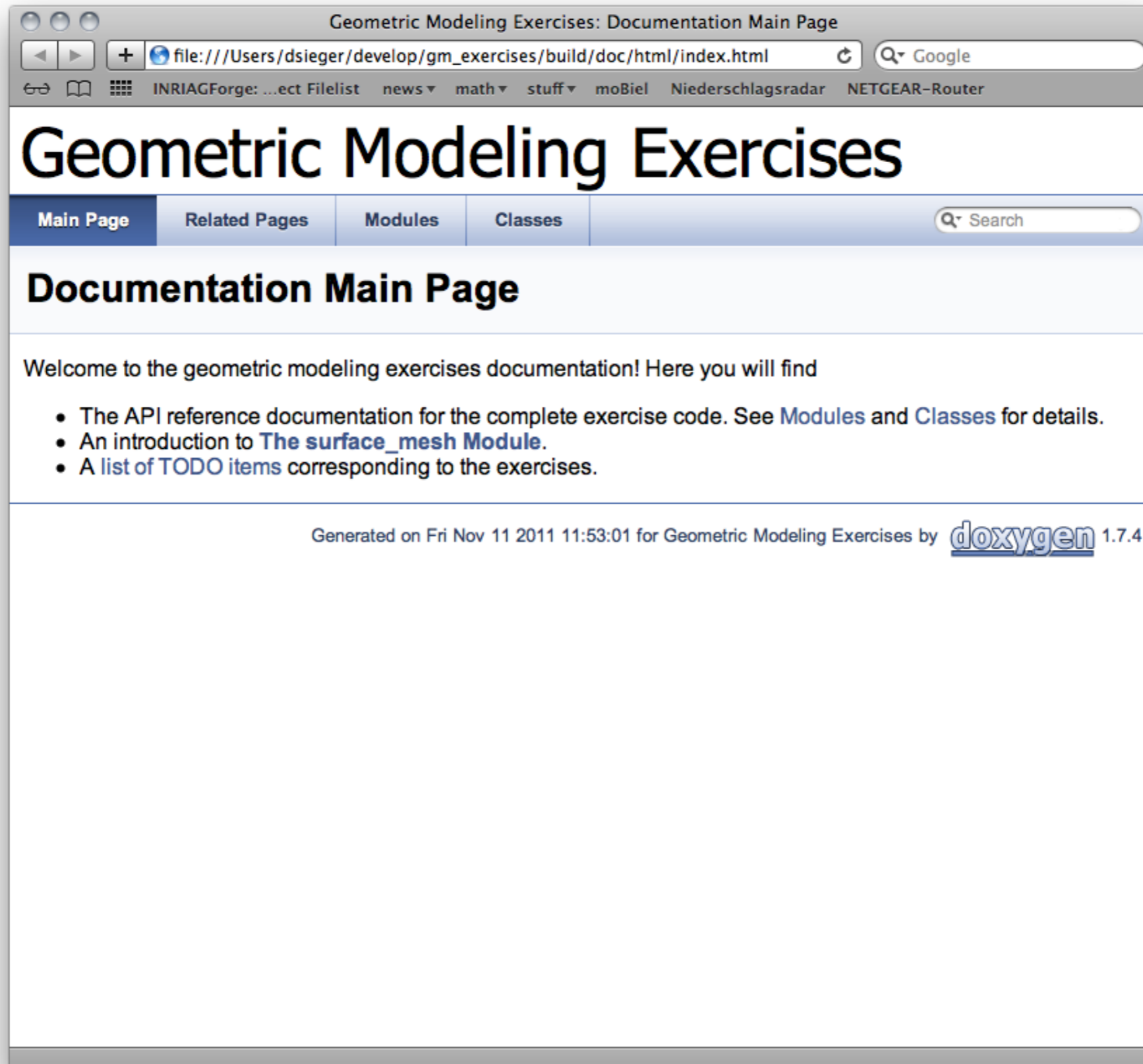


Edge
Flip



`mesh.flip (Edge)`

Documentation



The screenshot shows a web browser window with the title "Geometric Modeling Exercises: Documentation Main Page". The address bar contains the file path "file:///Users/dsieger/develop/gm_exercises/build/doc/html/index.html". The browser's search bar is set to "Google". The page content includes a navigation menu with "Main Page", "Related Pages", "Modules", and "Classes", and a search box. The main heading is "Geometric Modeling Exercises". Below the navigation is a sub-heading "Documentation Main Page". The main text reads: "Welcome to the geometric modeling exercises documentation! Here you will find" followed by a bulleted list: "• The API reference documentation for the complete exercise code. See [Modules](#) and [Classes](#) for details.", "• An introduction to [The surface_mesh Module](#).", and "• A list of [TODO items](#) corresponding to the exercises." At the bottom, it says "Generated on Fri Nov 11 2011 11:53:01 for Geometric Modeling Exercises by [doxygen](#) 1.7.4".

Geometric Modeling Exercises: Documentation Main Page

file:///Users/dsieger/develop/gm_exercises/build/doc/html/index.html

Google

INRIAGForge: ...ect Filelist news math stuff moBiel Niederschlagsradar NETGEAR-Router

Geometric Modeling Exercises

Main Page Related Pages Modules Classes Search

Documentation Main Page

Welcome to the geometric modeling exercises documentation! Here you will find

- The API reference documentation for the complete exercise code. See [Modules](#) and [Classes](#) for details.
- An introduction to [The surface_mesh Module](#).
- A list of [TODO items](#) corresponding to the exercises.

Generated on Fri Nov 11 2011 11:53:01 for Geometric Modeling Exercises by [doxygen](#) 1.7.4