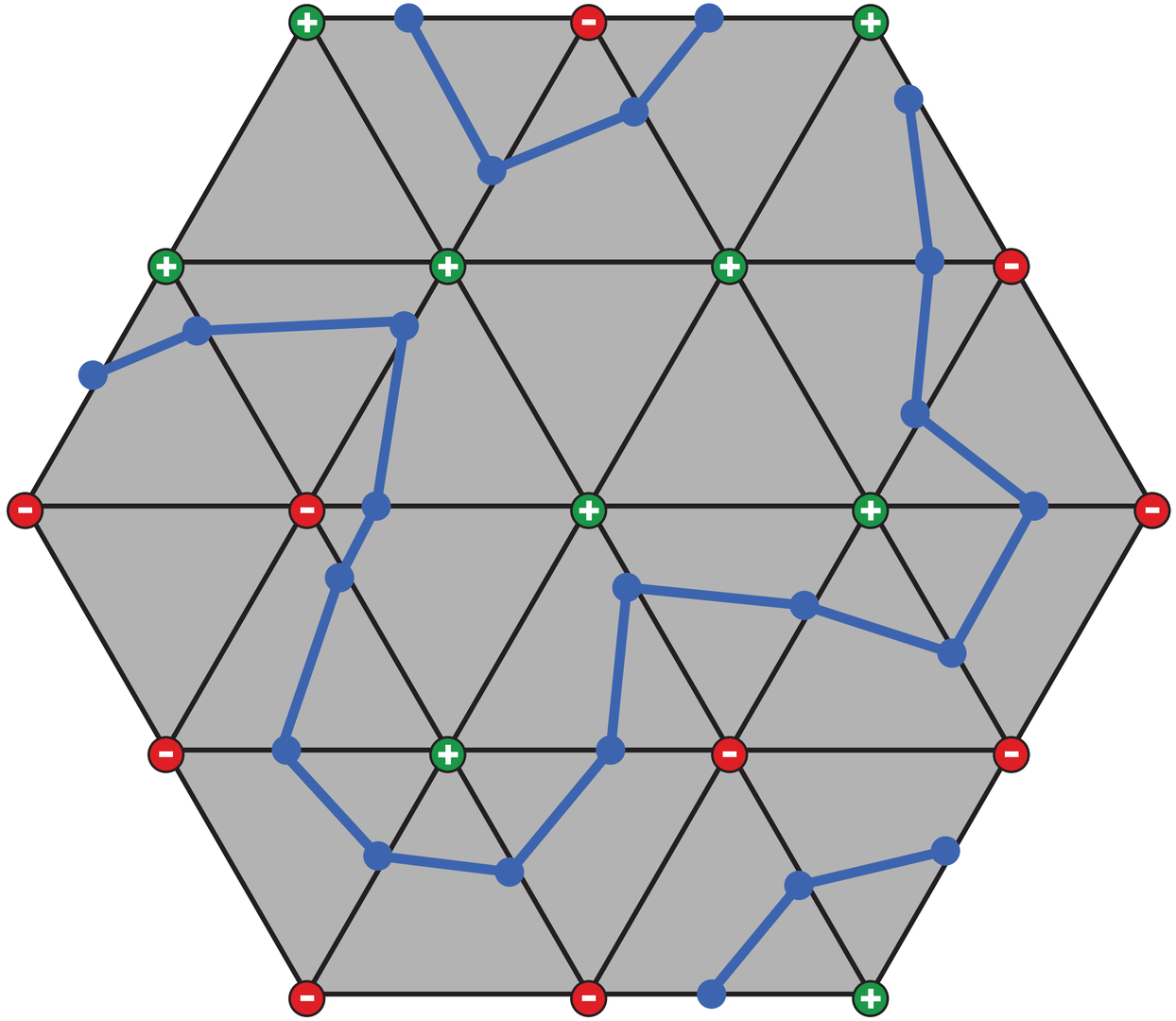
DGP Exercise 5:
Surface Normals & Discrete Differential Geometry

Tizian Zeltner

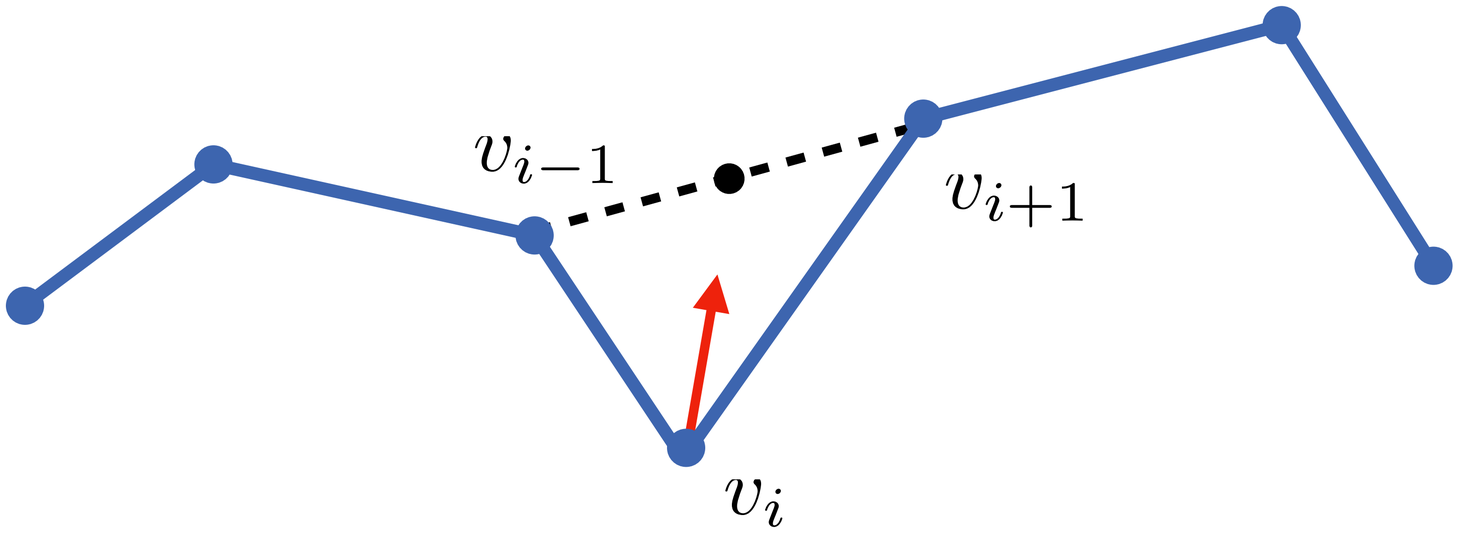
Today's Menu

- Recap for exercise 3: Marching triangles
- Recap for exercise 4: Curve smoothing
- Surface_mesh library
- Exercise 5: Surface normals + discrete differential geometry

Recap Exercise 3: Marching triangles



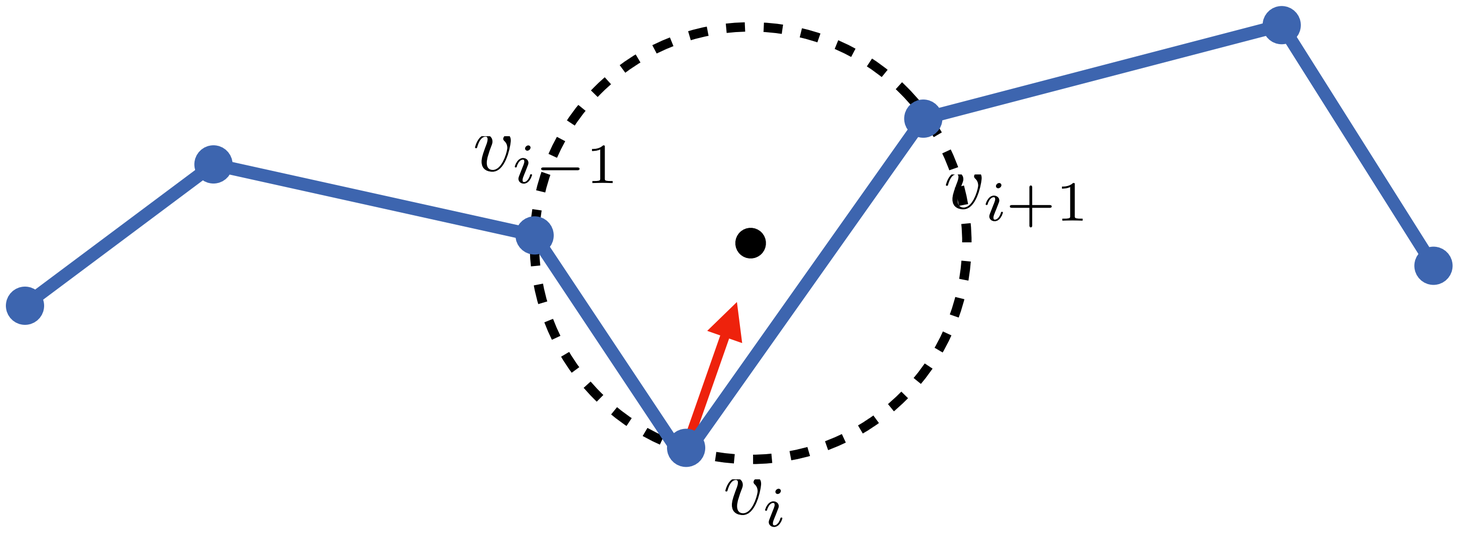
Recap Exercise 4: Curve smoothing



`laplacianSmoothing()`

$$v_i' = (1-\varepsilon)v_i + \varepsilon\frac{v_{i-1}+v_{i+1}}{2}$$

Recap Exercise 4: Curve smoothing

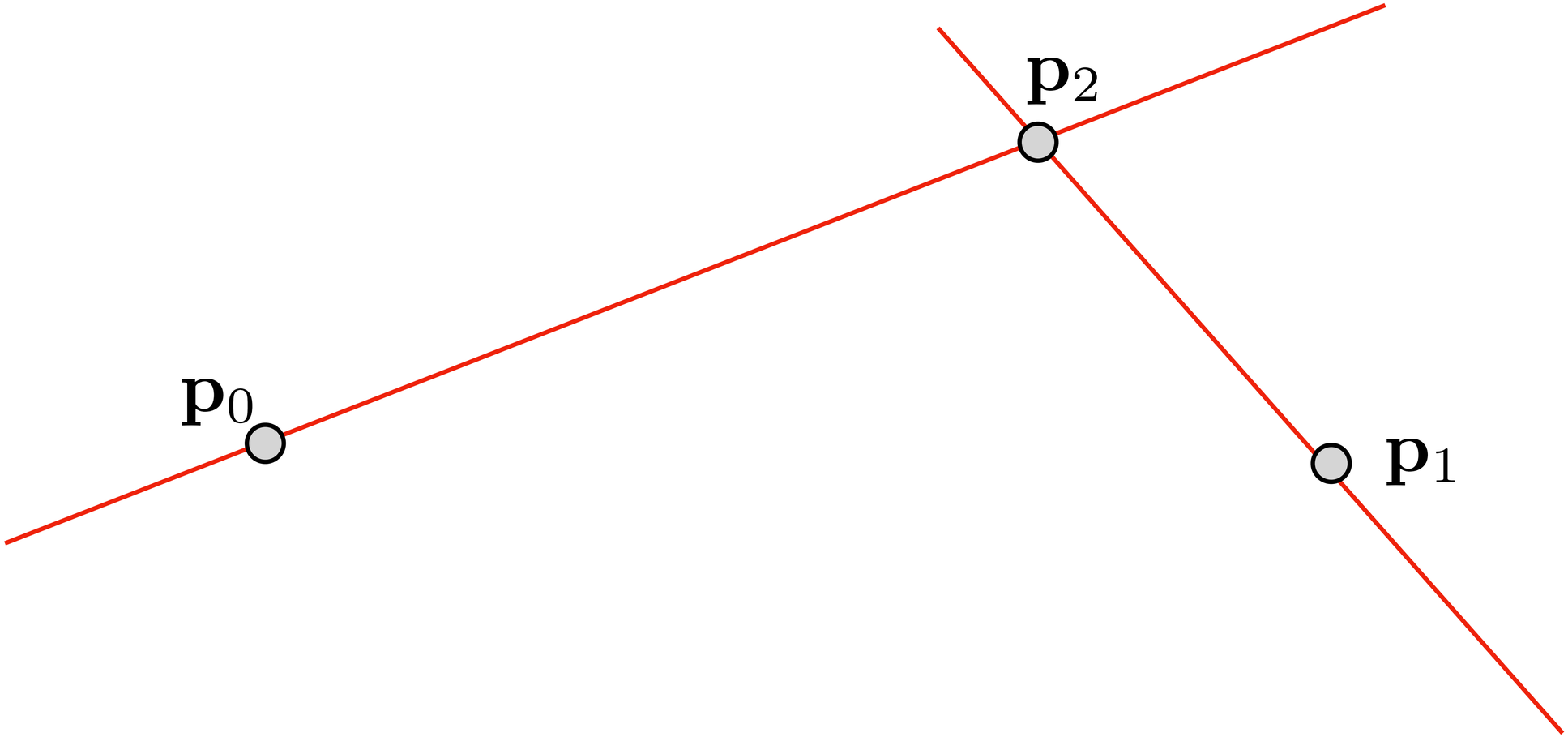


`osculatingCircle()`

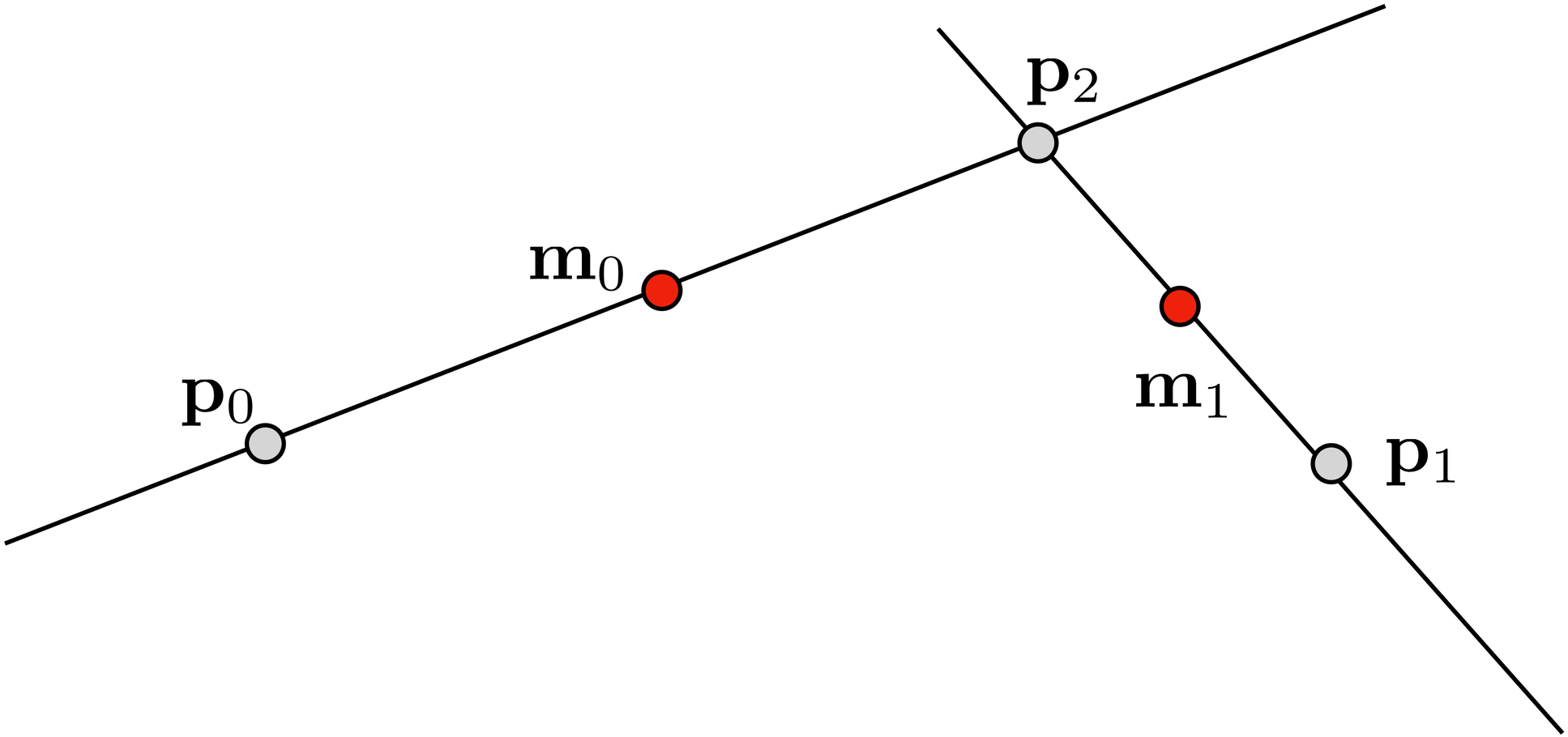
$$v_i' = v_i + \varepsilon \frac{C - v_i}{||C - v_i||^2}$$

Recap Exercise 4: Circle defined by 3 points

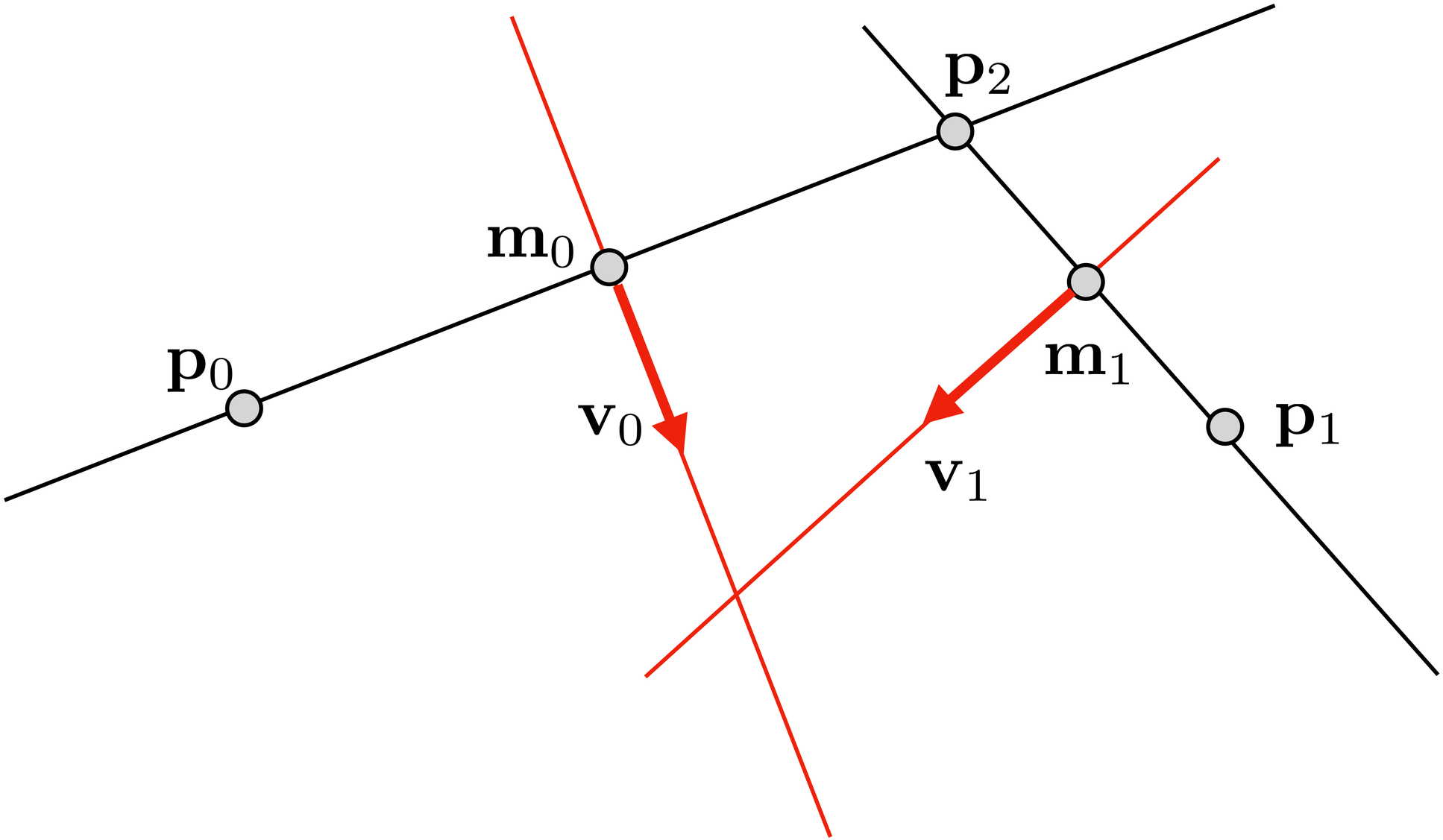
Recap Exercise 4: Circle defined by 3 points



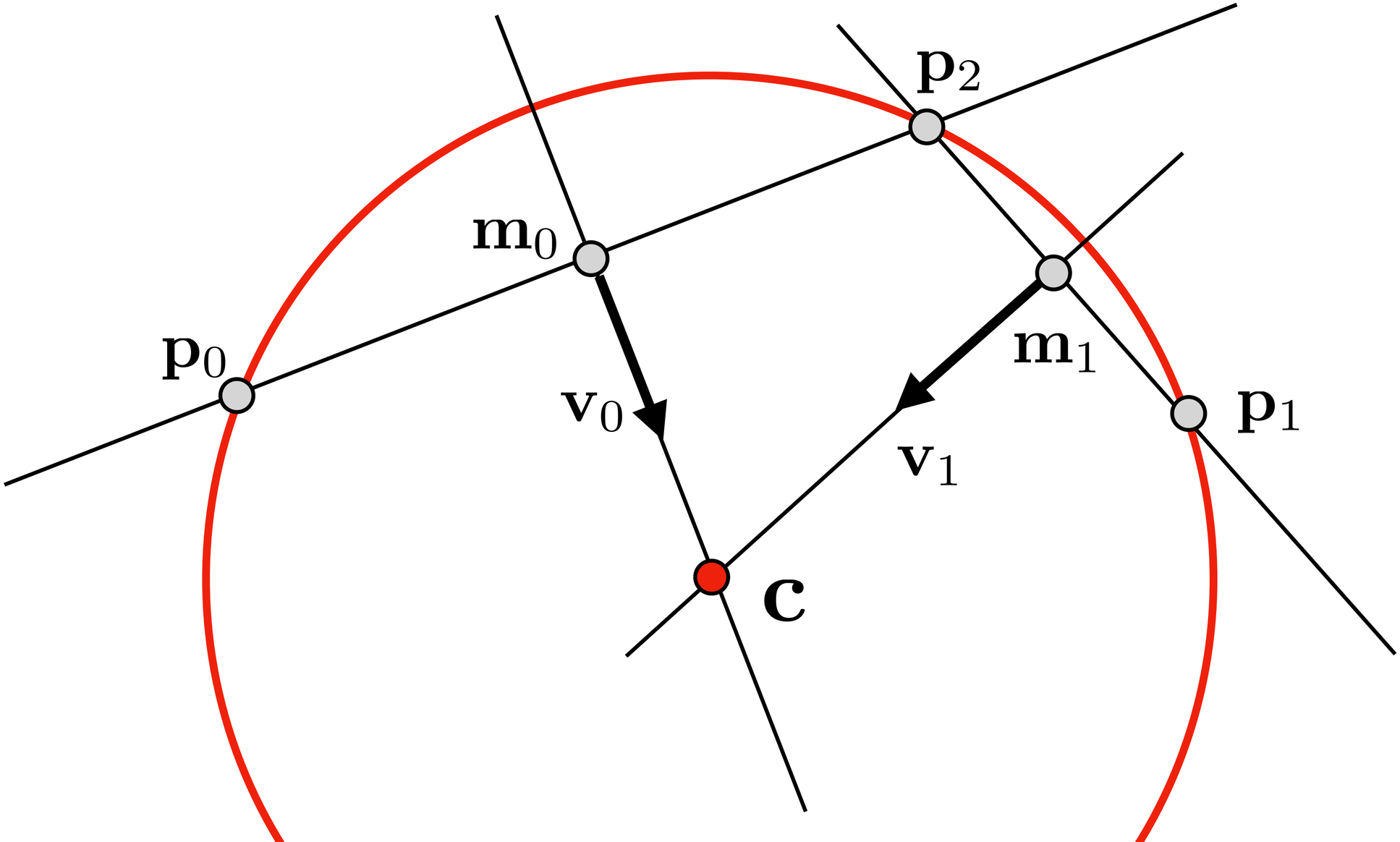
Recap Exercise 4: Circle defined by 3 points



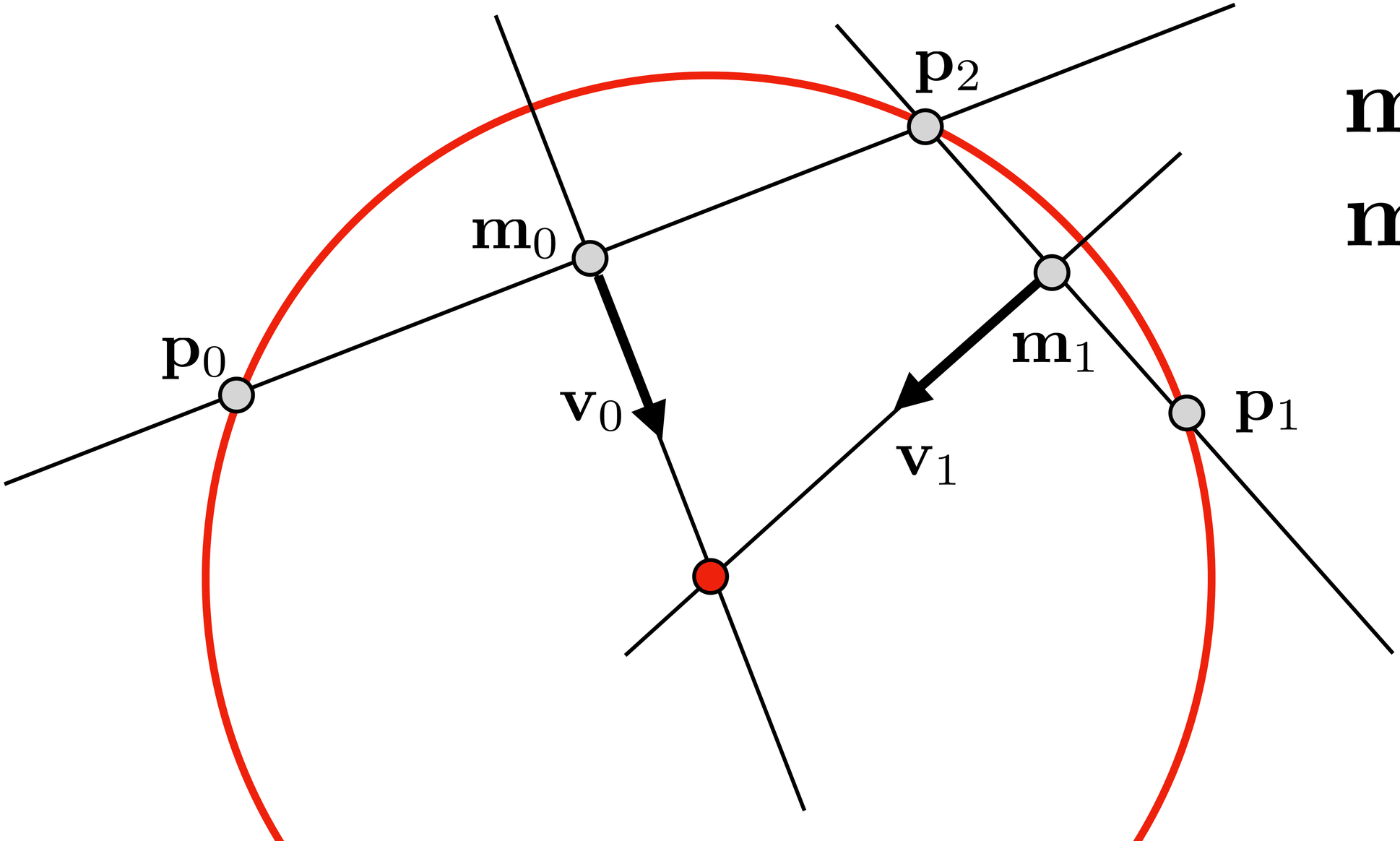
Recap Exercise 4: Circle defined by 3 points



Recap Exercise 4: Circle defined by 3 points



Recap Exercise 4: Circle defined by 3 points



$$\mathbf{m}_0 + s \cdot \mathbf{v}_0$$
$$\mathbf{m}_1 + t \cdot \mathbf{v}_1$$

Recap Exercise 4: Circle defined by 3 points

$$\mathbf{m}_0 + s \cdot \mathbf{v}_0 = \mathbf{m}_1 + t \cdot \mathbf{v}_1$$

$$s \cdot \mathbf{v}_0 - t \cdot \mathbf{v}_1 = \mathbf{m}_1 - \mathbf{m}_0$$

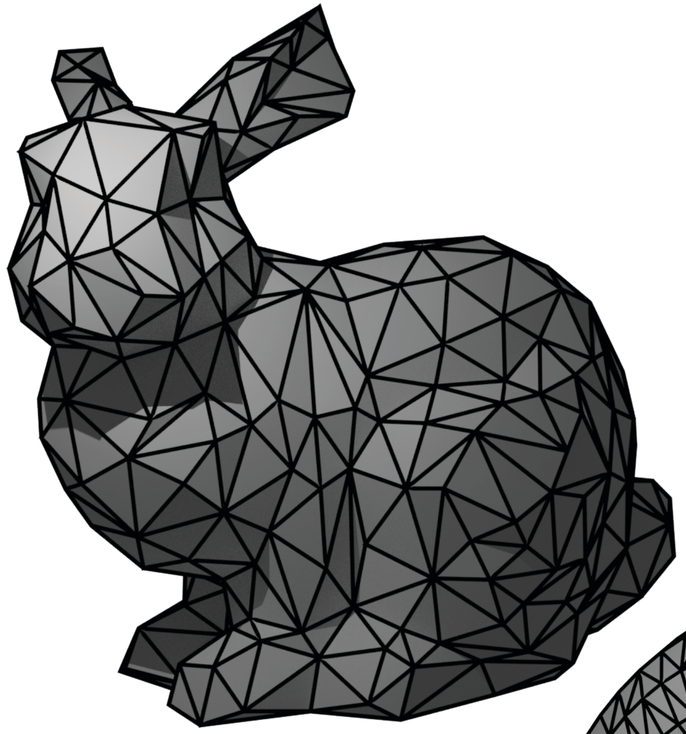
Linear system (2x2):

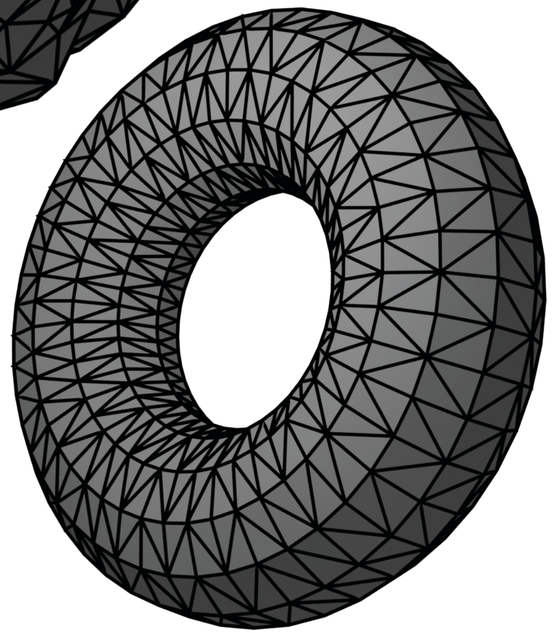
$$\begin{pmatrix} \mathbf{v}_0 & -\mathbf{v}_1 \end{pmatrix} \cdot \begin{pmatrix} s \\ t \end{pmatrix} = \begin{pmatrix} \mathbf{m}_1 - \mathbf{m}_0 \end{pmatrix}$$

$$\mathbf{c} = \mathbf{m}_0 + s \cdot \mathbf{v}_0$$

Surface_mesh library

Polygonal Mesh Processing

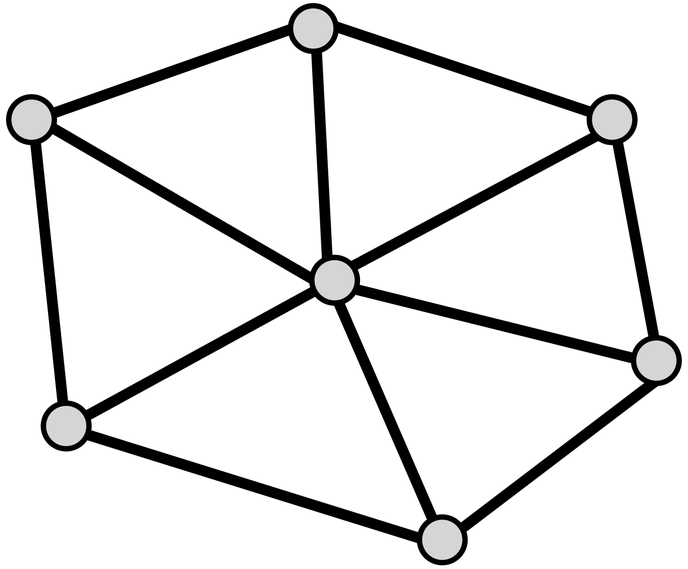




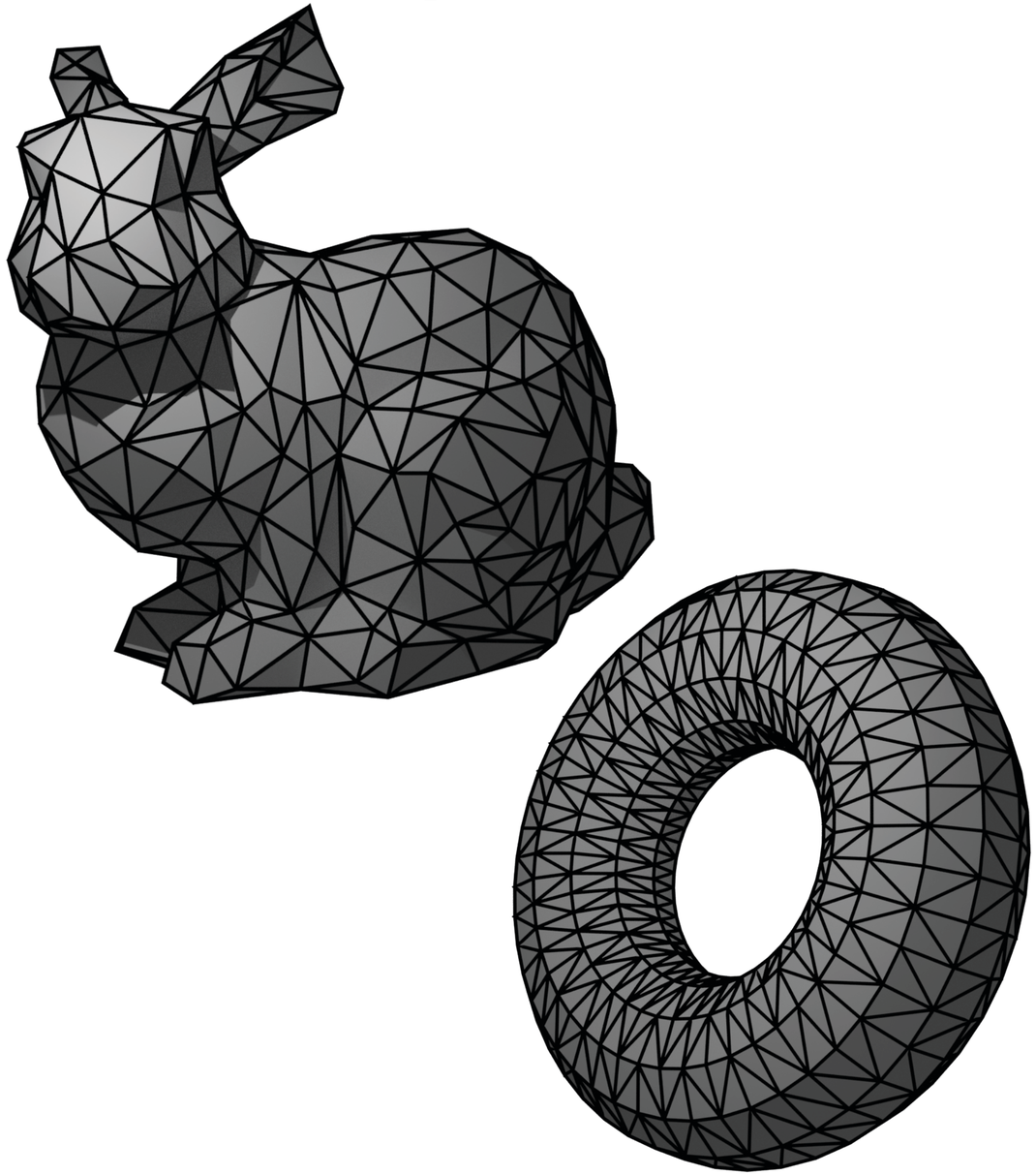
$$\mathcal{M} = (\{\mathbf{v}_i\}, \{e_j\}, \{f_k\})$$

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

topology e_j, f_k

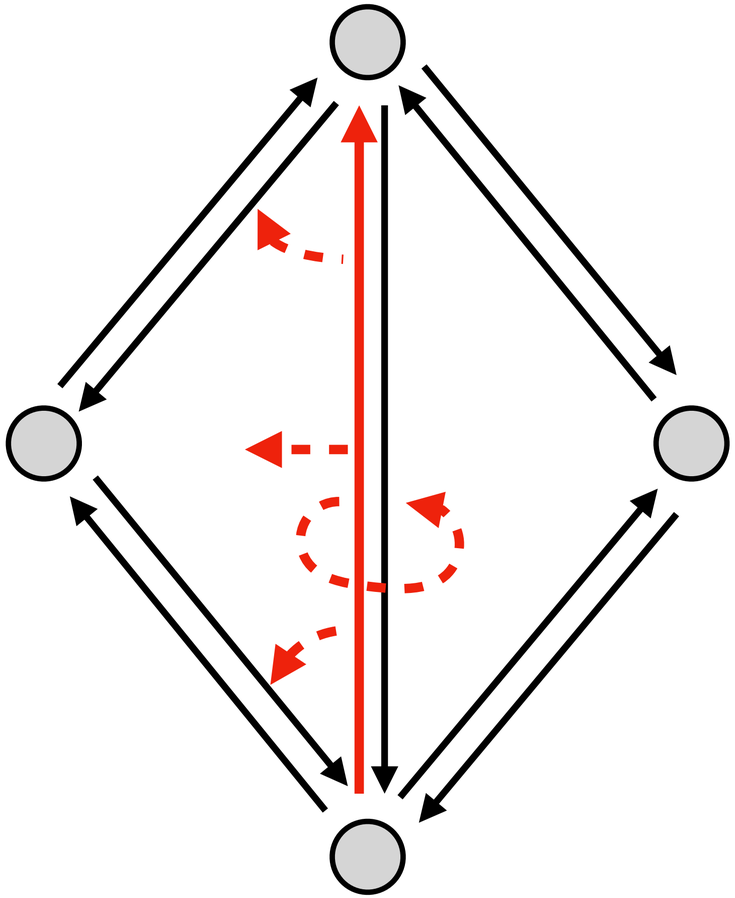


Requirements for a digital representation



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

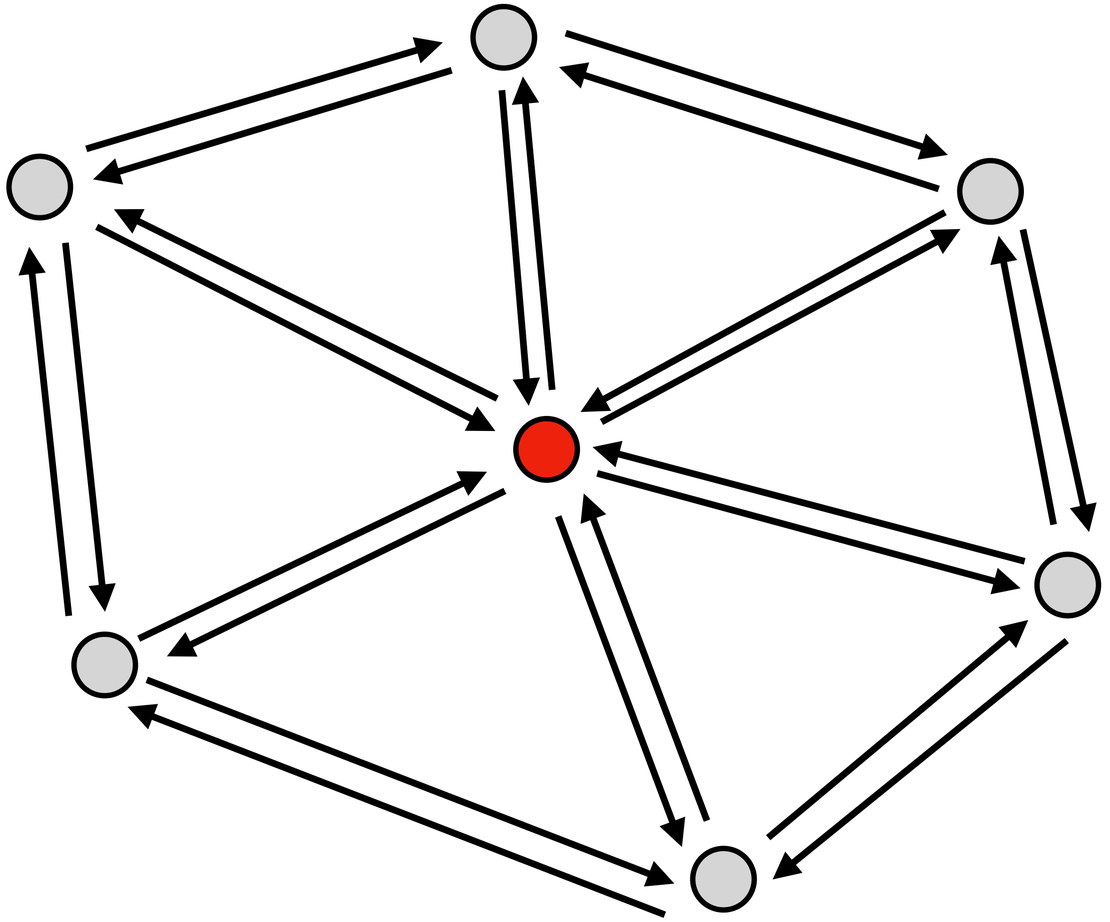
Halfedge data structure



- Vertex
  - one outgoing halfedge
- Halfedge
  - target vertex
  - incident face
  - previous/next/opposite halfedge
- Face
  - one incident halfedge

One-ring neighborhood traversal

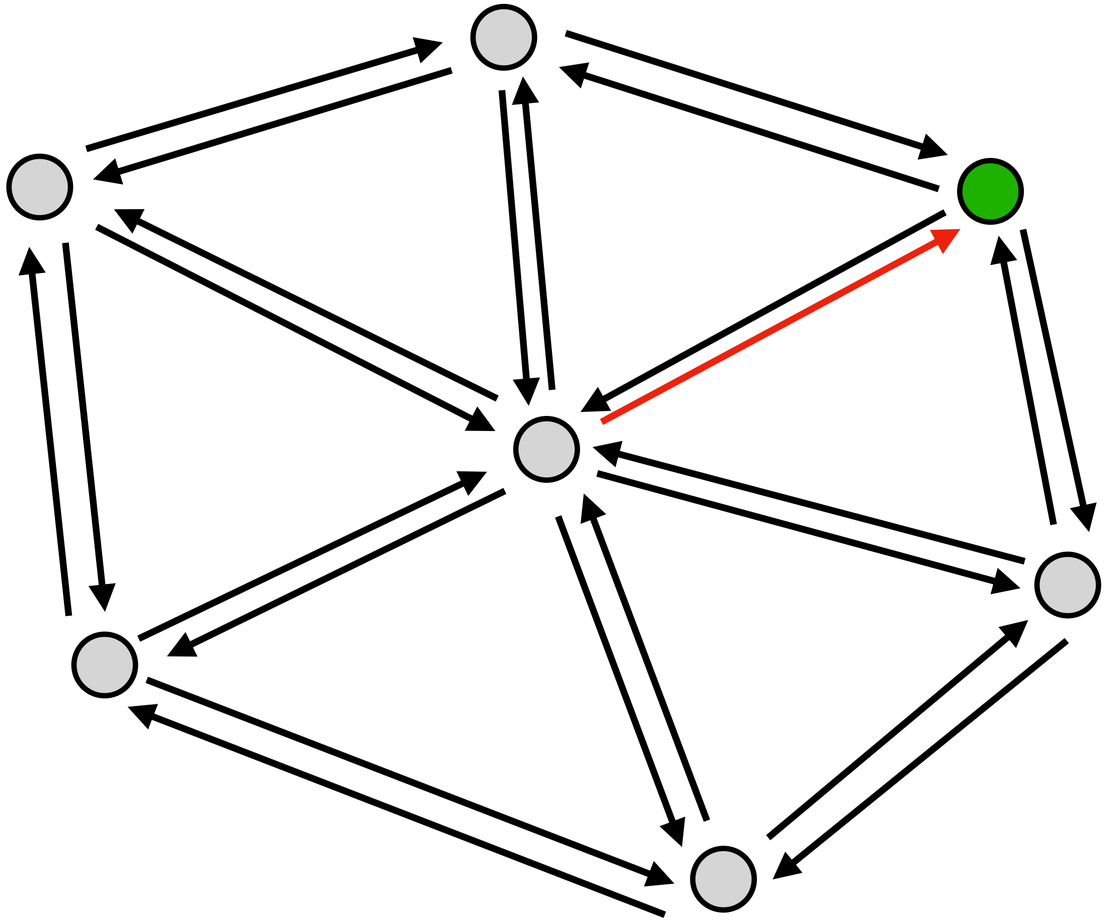
1. Start at vertex



One-ring neighborhood traversal

1. Start at vertex

2. Outgoing halfedge

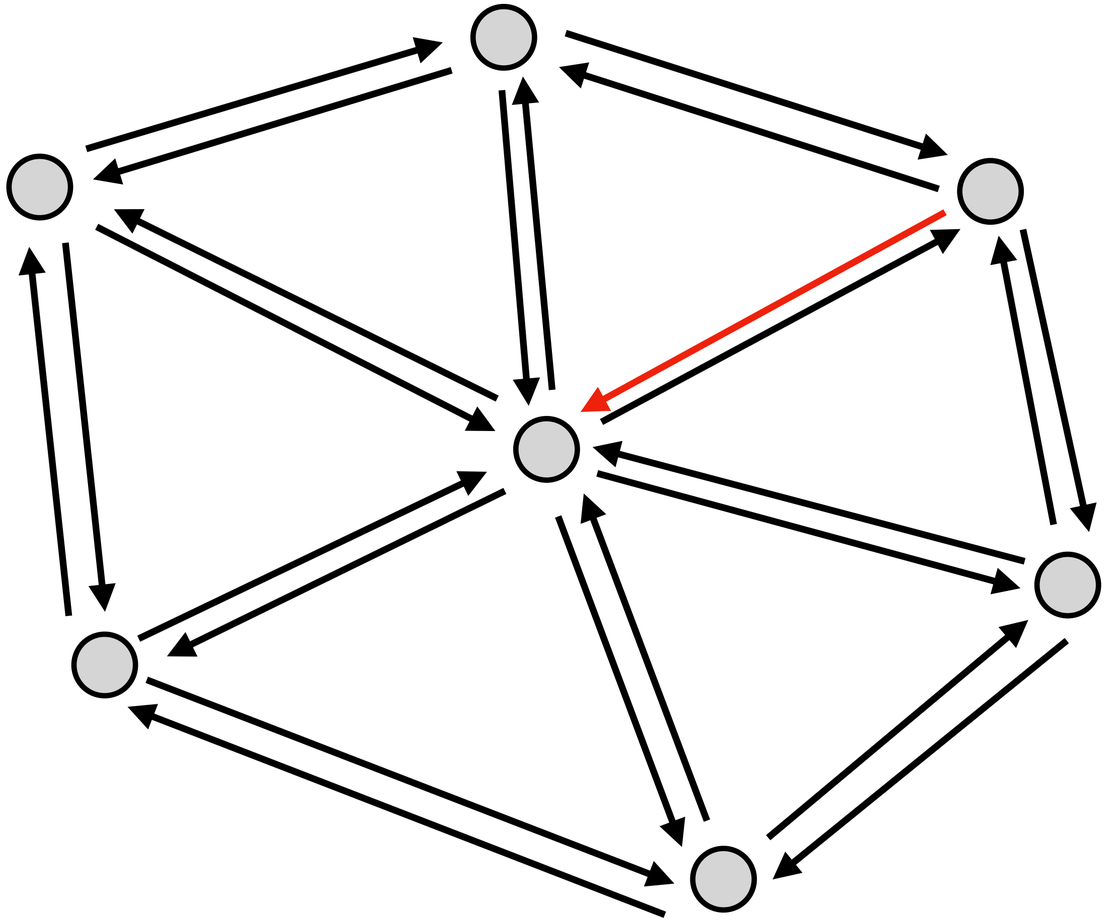


One-ring neighborhood traversal

1. Start at vertex

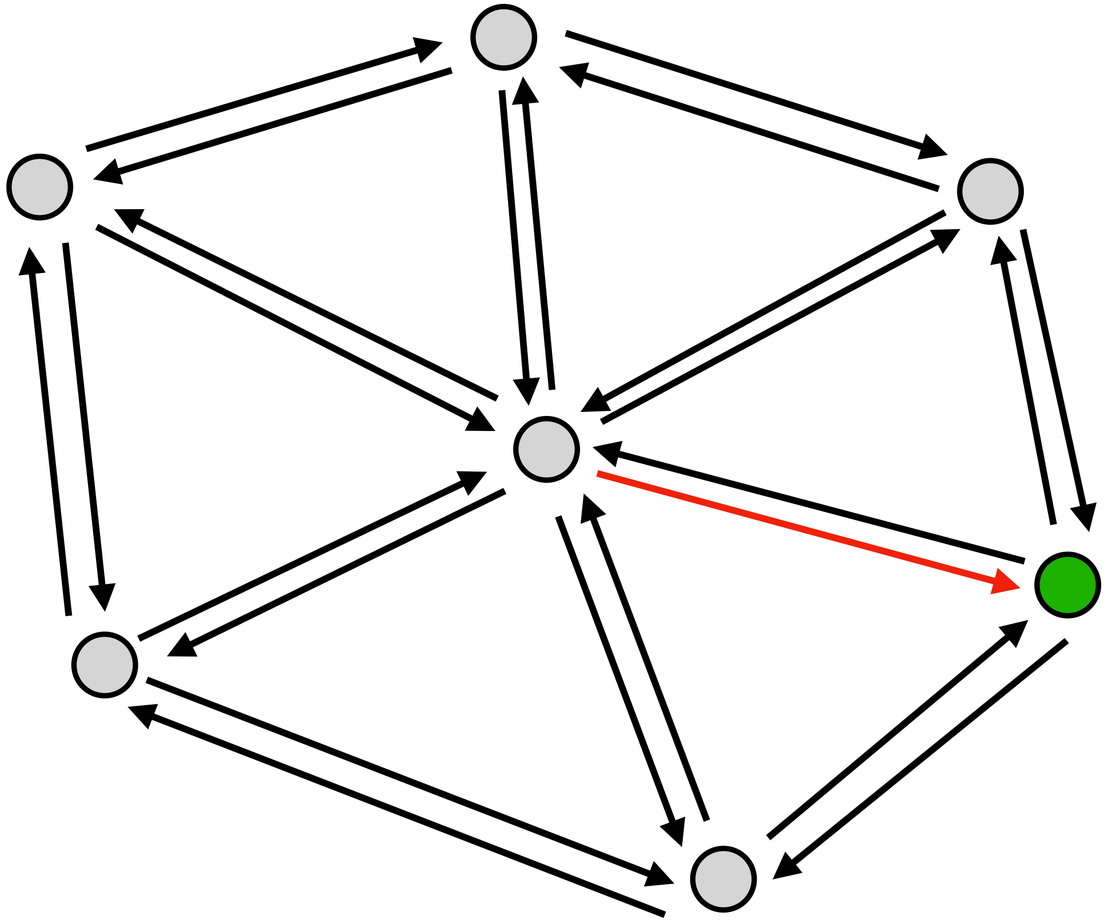
2. Outgoing halfedge

3. Opposite halfedge



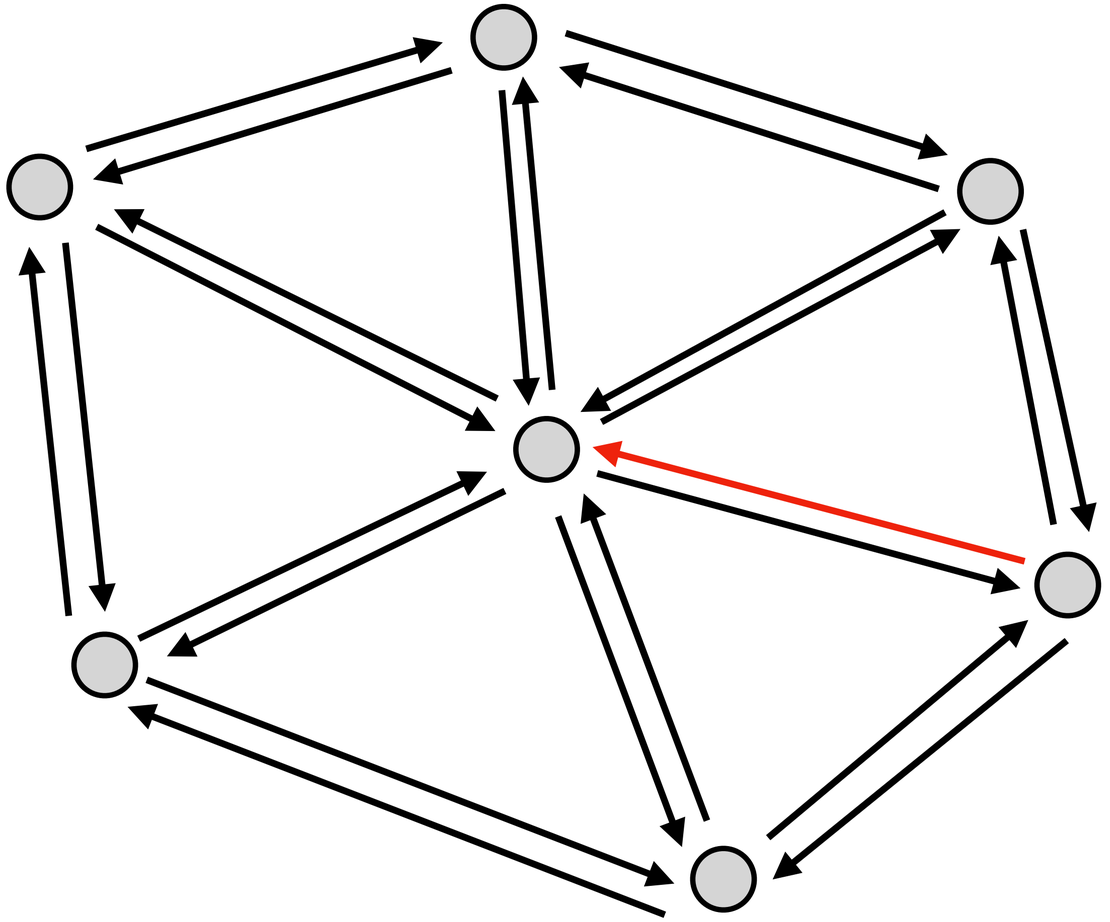
One-ring neighborhood traversal

1. Start at vertex
2. Outgoing halfedge
3. Opposite halfedge
4. Next halfedge



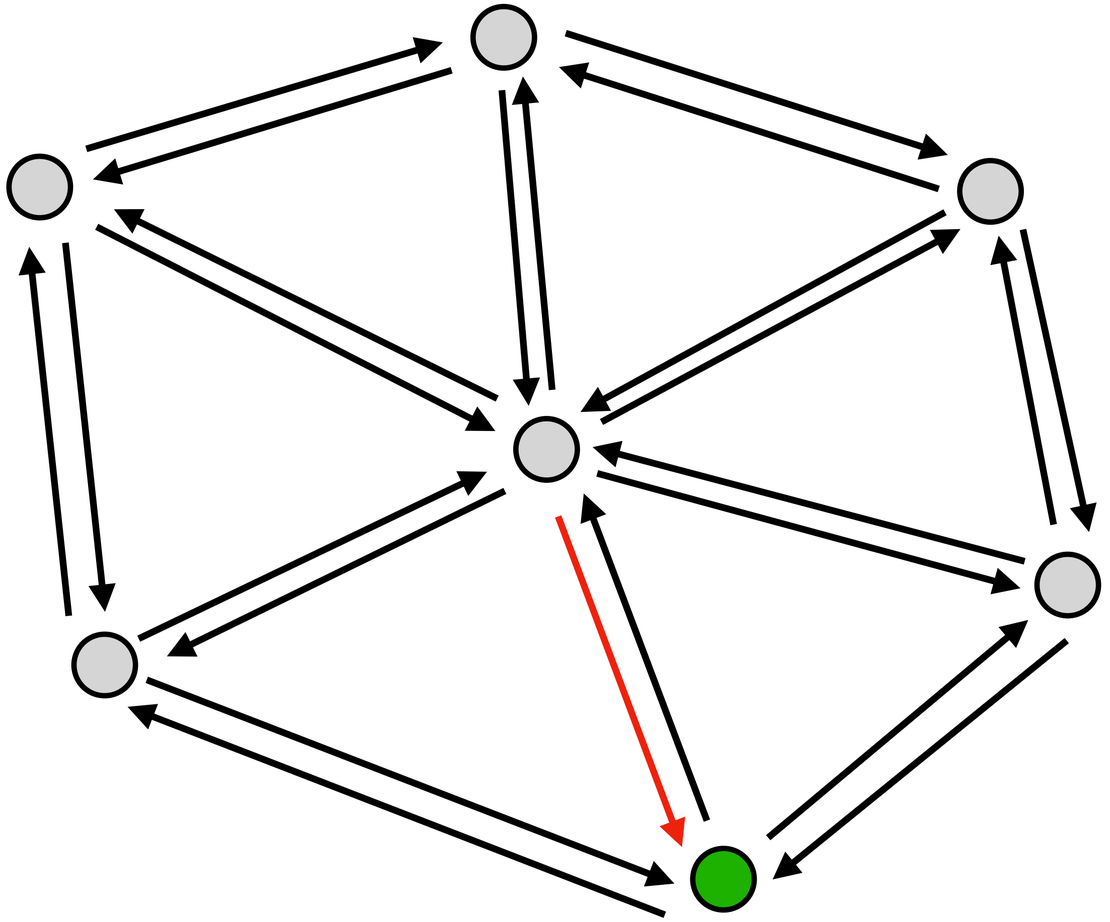
One-ring neighborhood traversal

1. Start at vertex
2. Outgoing halfedge
3. Opposite halfedge
4. Next halfedge
5. Opposite



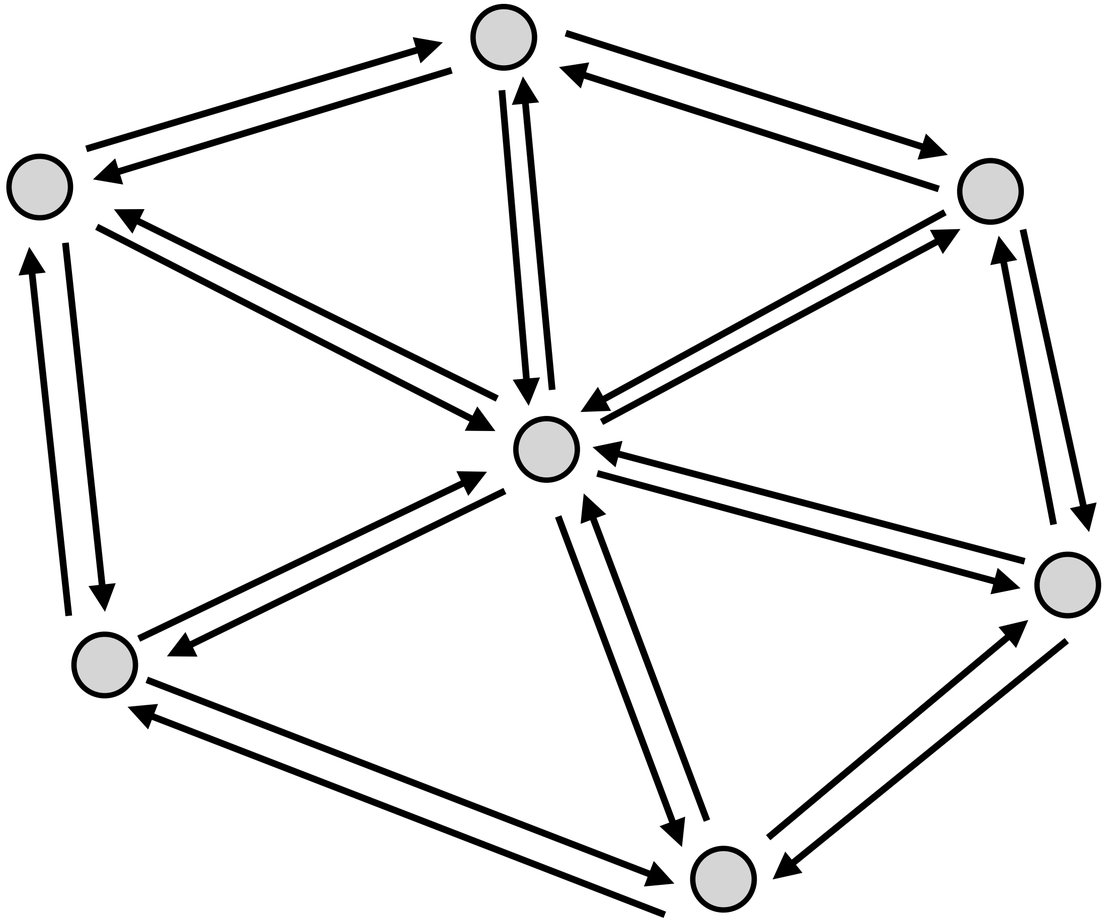
One-ring neighborhood traversal

1. Start at vertex
2. Outgoing halfedge
3. Opposite halfedge
4. Next halfedge
5. Opposite
6. Next



One-ring neighborhood traversal

1. Start at vertex
2. Outgoing halfedge
3. Opposite halfedge
4. Next halfedge
5. Opposite
6. Next
7. ...



Surface_mesh

- Graphics & Geometry Group, Bielefeld University
- C++ Library for mesh representation and processing
  - Implements **halfedge data structure**
  - Integrated **arithmetic / geometric operations**
  - Mesh file reader / writer

Include and namespace

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

using namespace surface_mesh;
typedef Surface_mesh Mesh;
```

Integrated vector operations

```
// Scalar type
typedef float Scalar;

// Vector types
Vec3 v;
Point p;
Normal n;
Color c;
Texture_coordinate u;
```

All of these are equivalent

Integrated vector operations

```
using namespace surface_mesh;

Vec3 x, y;

// ...

Vec3 a = x + y;

Vec3 b = x - y;

Vec3 c = x * 0.5;

Vec3 d = y / 3.0;
```

3D Vectors

Arithmetic operations

Integrated vector operations

```
using namespace surface_mesh;

Vec3 x, y;

// ...

Scalar a = dot(x, y);

Scalar b = norm(y);

Vec3 c = normalize(x);

Vec3 d = cross(x, y);
```

3D Vectors

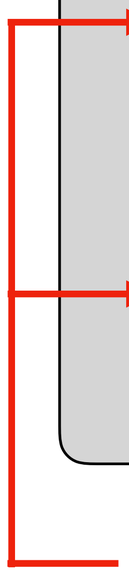
Arithmetic operations

Loading and writing mesh files

```
// Create mesh object
Mesh myMesh;
mesh.read("/input/file")
// Manipulate ...
mesh.write("/output/file")
```

Mesh creation

```
// Adding vertices and faces
Mesh myMesh;

Mesh::Vertex v = myMesh.add_vertex(Point(x,y,z));

Mesh::Vertex v0, v1, v2;
Mesh::Face f = myMesh.add_triangle(v0, v1, v2);
```



Vertex / face *handles*

Accessing and modifying vertex geometry

```
Mesh myMesh;
Mesh::Vertex v;


Point pos = myMesh.position(v);


Point new_pos = ...
myMesh.position(v) = new_pos;
```

Element count functions

```
Mesh myMesh;

unsigned int n1 = myMesh.n_vertices(),
             n2 = myMesh.n_faces(),
             n3 = myMesh.n_edges(),
             n4 = myMesh.n_halfedges();
```

Mesh elements traversal

```
// Iterate over all vertices in the mesh ...
Mesh myMesh;
Mesh::Vertex_iterator v_it, v_begin, v_end;

// Iterator ranges

v_begin = myMesh.vertices_begin();

v_end = myMesh.vertices_end();


for (v_it = v_begin; v_it != v_end; ++v_it) {
    Mesh::Vertex v = *v_it;
    doSomethingWithVertex(v);
}
```

Mesh elements traversal

- Iterating over faces

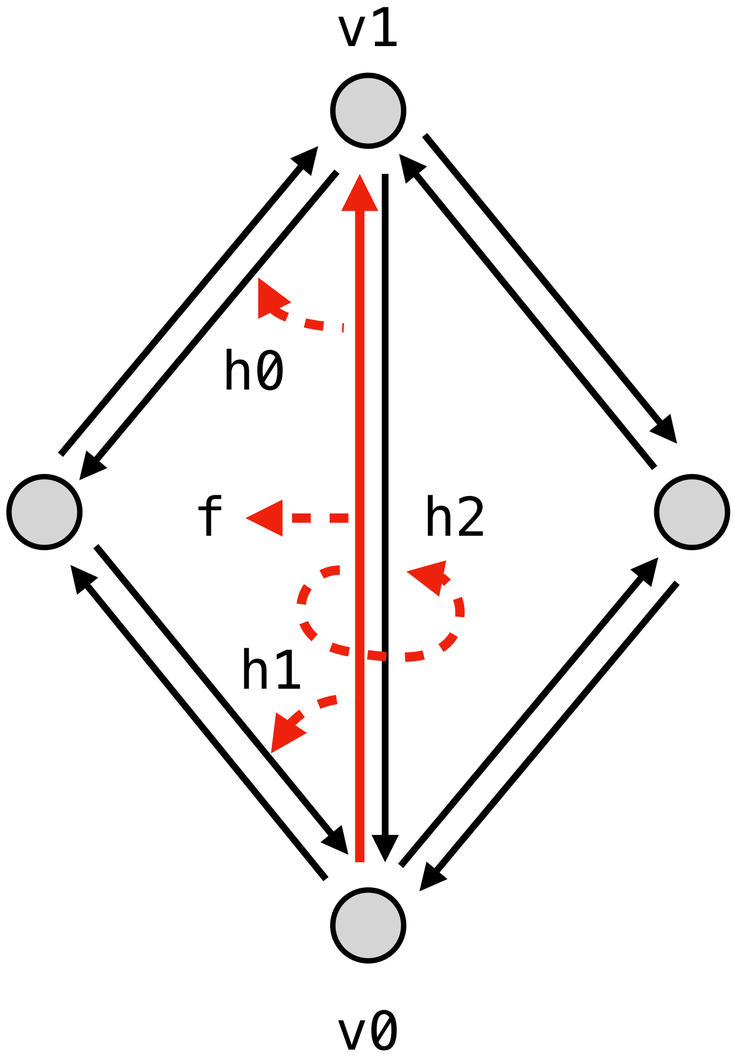
`Mesh::Vertex_iterator`	⟶	`Mesh::Face_iterator`
`vertices_begin()`	⟶	`faces_begin()`
`vertices_end()`	⟶	`faces_end()`

- Similarly:

`Mesh::Edge_iterator` & `Mesh::Halfedge_iterator`

Connectivity / Halfedge queries

```
Mesh myMesh;
Mesh::halfedge h;

Mesh::Vertex v0 = myMesh.from_vertex(h);
Mesh::Vertex v1 = myMesh.to_vertex(h);

Mesh::Face f = myMesh.face(h);

Mesh::Halfedge h0 = myMesh.next_halfedge(h);
Mesh::Halfedge h1 = myMesh.prev_halfedge(h);
Mesh::Halfedge h2 = myMesh.opposite_halfedge(h);
```

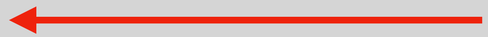


Other useful halfedge things

```
Mesh myMesh;
Mesh::Vertex v;

// Get outgoing halfedge of vertex
Mesh::halfedge h = myMesh.halfedge(v);

// Whether elements are at a boundary
bool b0 = myMesh.is_boundary(v),
     b1 = myMesh.is_boundary(h);
```

Neighborhood query

```
// Traverse all neighboring vertices
Mesh myMesh;

Mesh::Vertex v0;

Mesh::Vertex_around_vertex_circulator vc, vc_end;


vc = myMesh.vertices(v0);

vc_end = c;


do {
    Mesh::Vertex v = *vc;
    doSomethingWithVertex(v);
} while (++vc != vc_end);
```



Note down starting vertex

Neighborhood query

- Other circulators:

```
Mesh::Face_around_vertex_circulator

Mesh::Halfedge_around_face_circulator

Mesh::Halfedge_around_vertex_circulator

Mesh::Vertex_around_face_circulator
```

Properties

```
Mesh myMesh;
// Creation
Mesh::EdgeProperty<float> edge_length
    = myMesh.add_edge_property<float>("property-name");

Mesh::Edge e; float L;
edge_length[e] = L;

// Removal
Mesh::EdgeProperty<Point> edge_points = ...
mesh.remove_edge_property(edge_points);
```

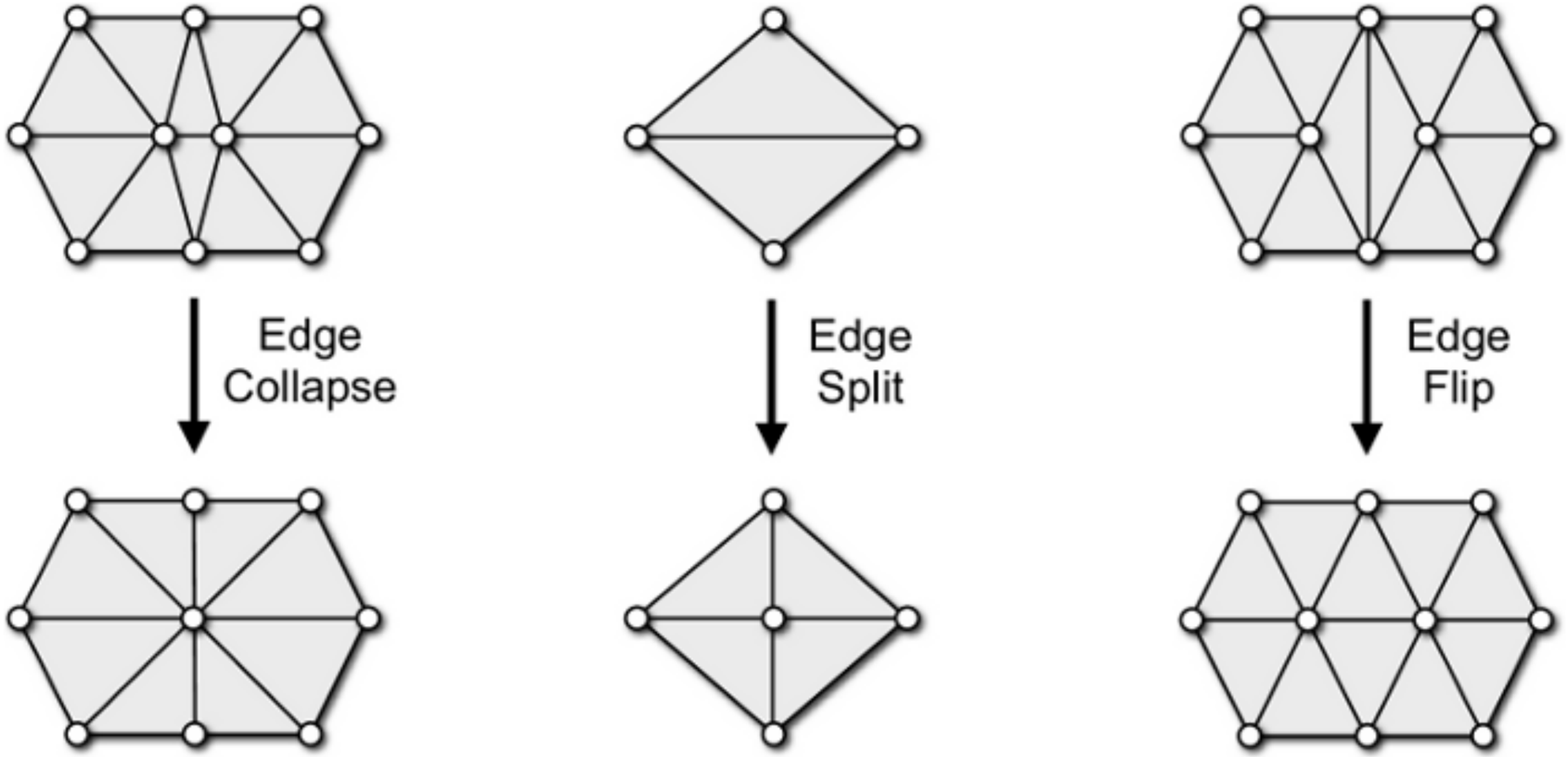
Access like an array

Properties

- Properties for other elements:

`Vertex`	⟶	`Mesh::Vertex_property<T>`
`Face`	⟶	`Mesh::Face_property<T>`
`Halfedge`	⟶	`Mesh::Halfedge_property<T>`

Advanced topics: Modifying topology



References

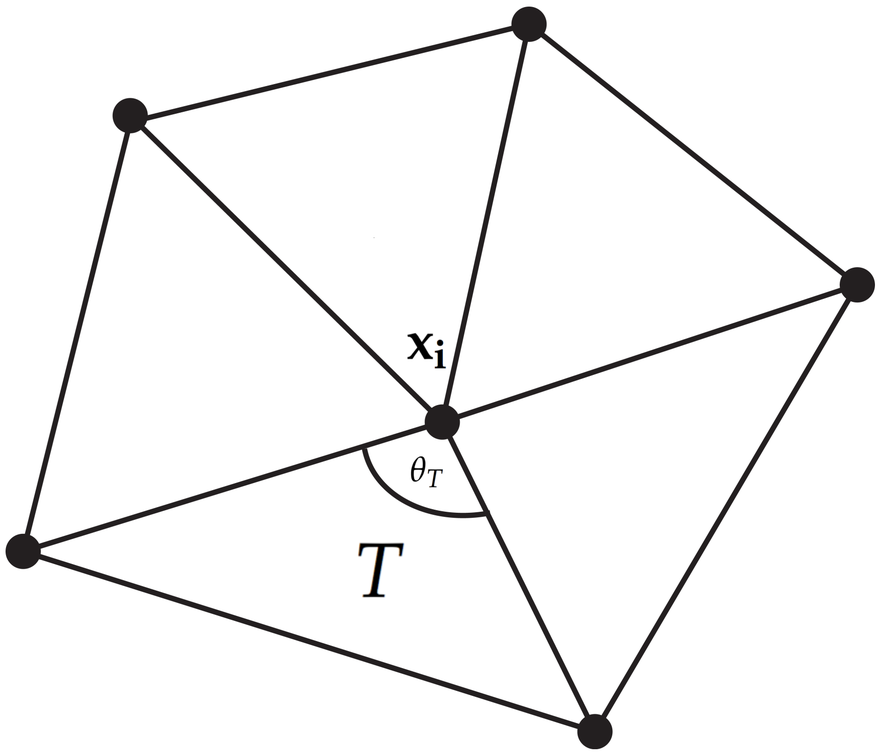
- Surface_mesh documentation
  https://www.labri.fr/perso/pbenard/teaching/surface_mesh/usage.html

- `Surface_mesh.h`

- Design, Implementation, and Evaluation of the Surface_mesh Data Structure, Sieger and Botsch 2011
  https://imr.sandia.gov/papers/imr20/Sieger.pdf

Exercise 5

1) Computing Vertex Normals



- Face normals

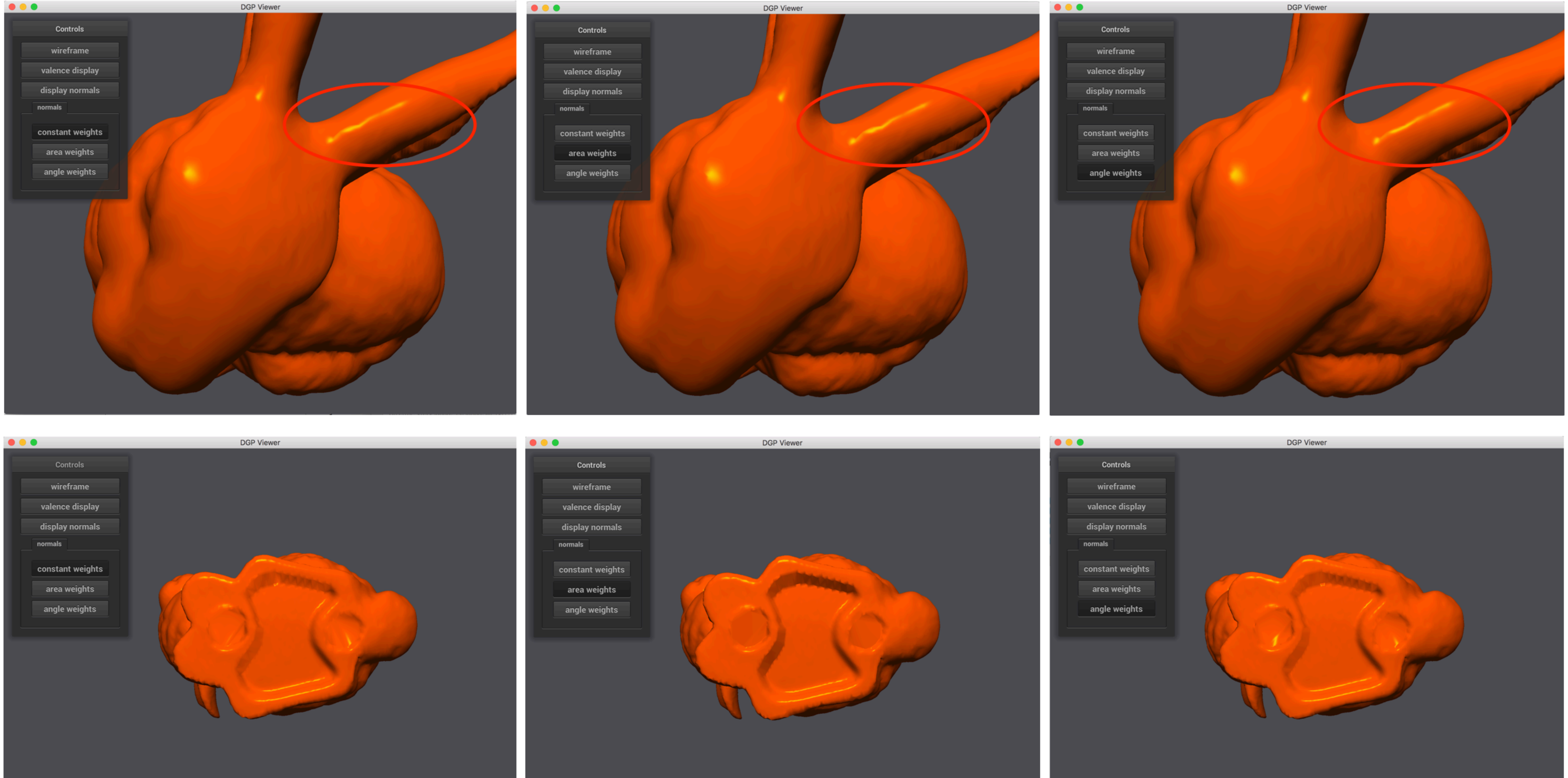
$$\mathbf{n}(T) = \frac{(\mathbf{x}_j - \mathbf{x}_i) \times (\mathbf{x}_k - \mathbf{x}_i)}{||(\mathbf{x}_j - \mathbf{x}_i) \times (\mathbf{x}_k - \mathbf{x}_i)||}$$

- Vertex normals

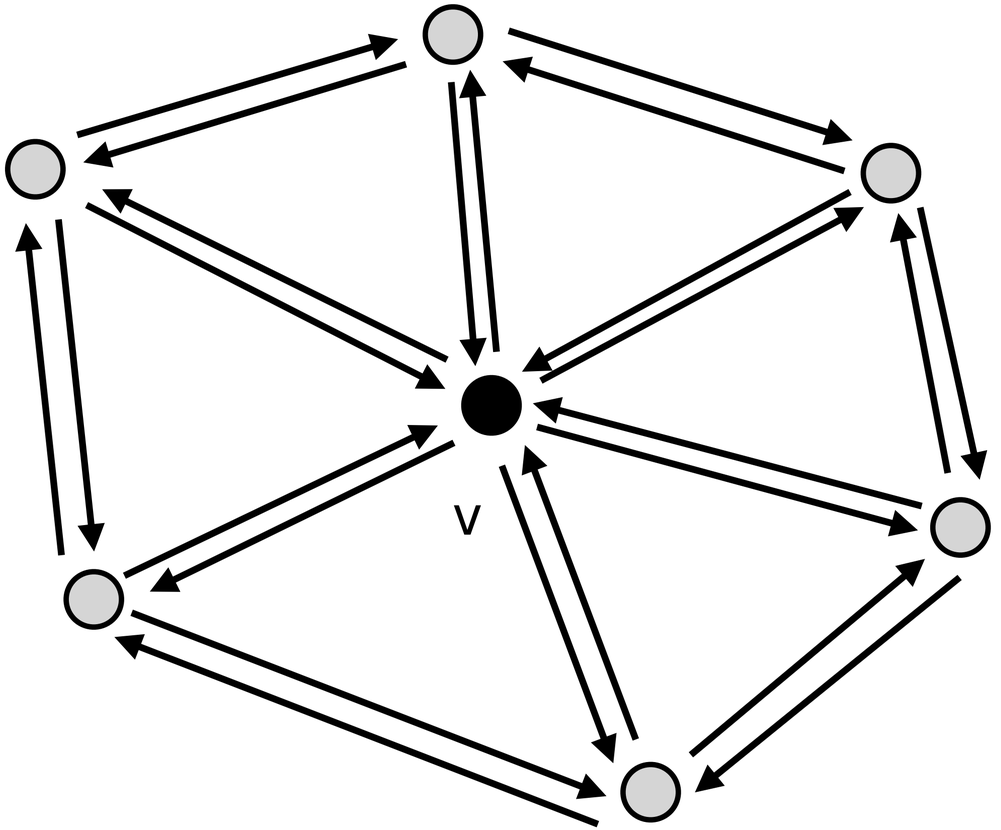
$$\mathbf{n}(x_i) = \frac{\sum_{T \in \mathcal{N}_1(x_i)} \alpha_T \ \mathbf{n}(T)}{||\sum_{T \in \mathcal{N}_1(x_i)} \alpha_T \ \mathbf{n}(T)||}$$

- Weights
  - constant
  - area **Hint: cross product**
  - angle

1) Computing Vertex Normals

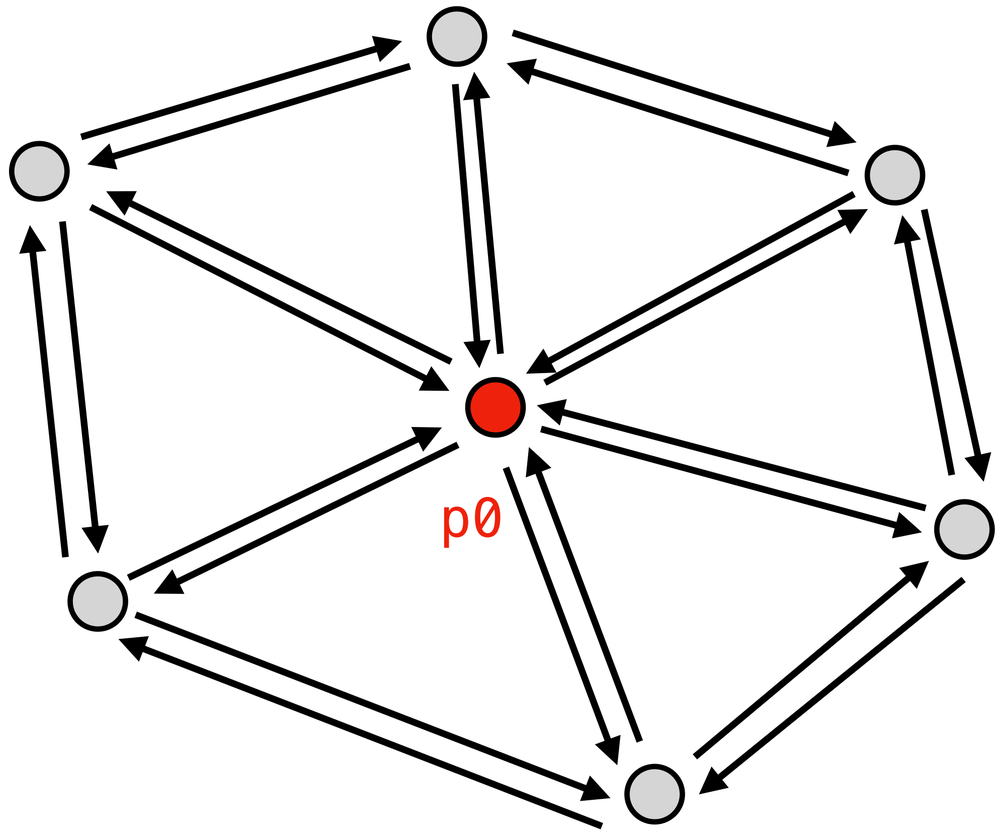


Example: Compute angles using `Surface_mesh`



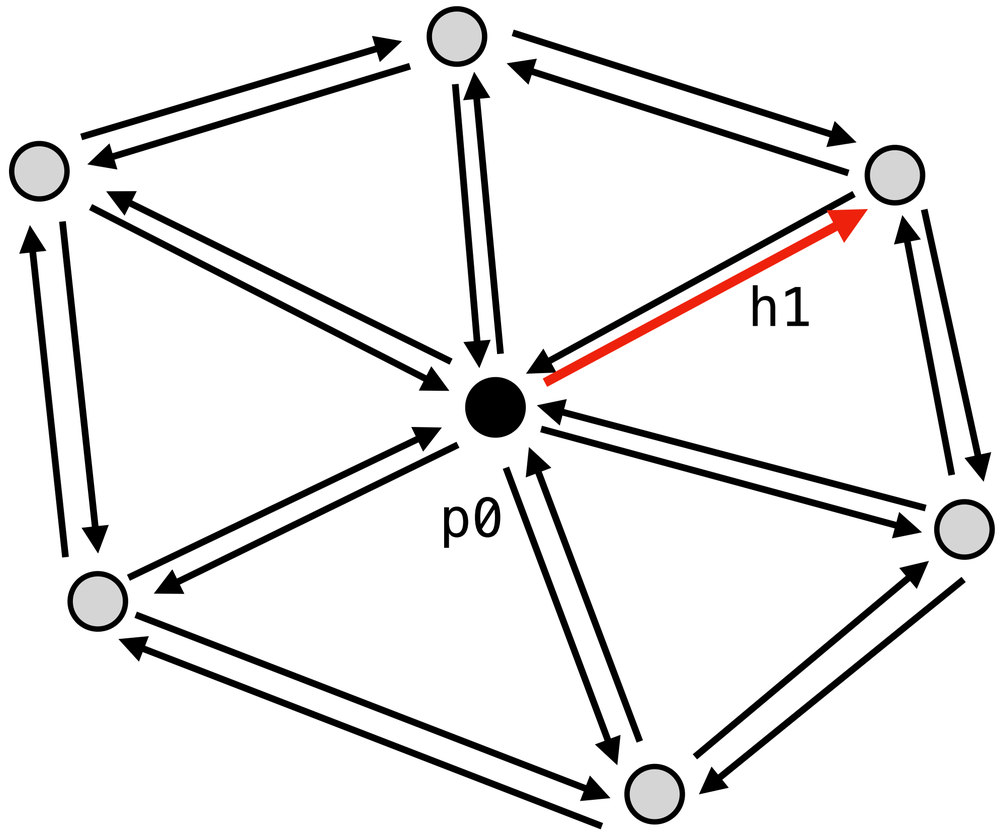
Example: Compute angles using `Surface_mesh`

```
Point p0 = mesh.position(v);
```



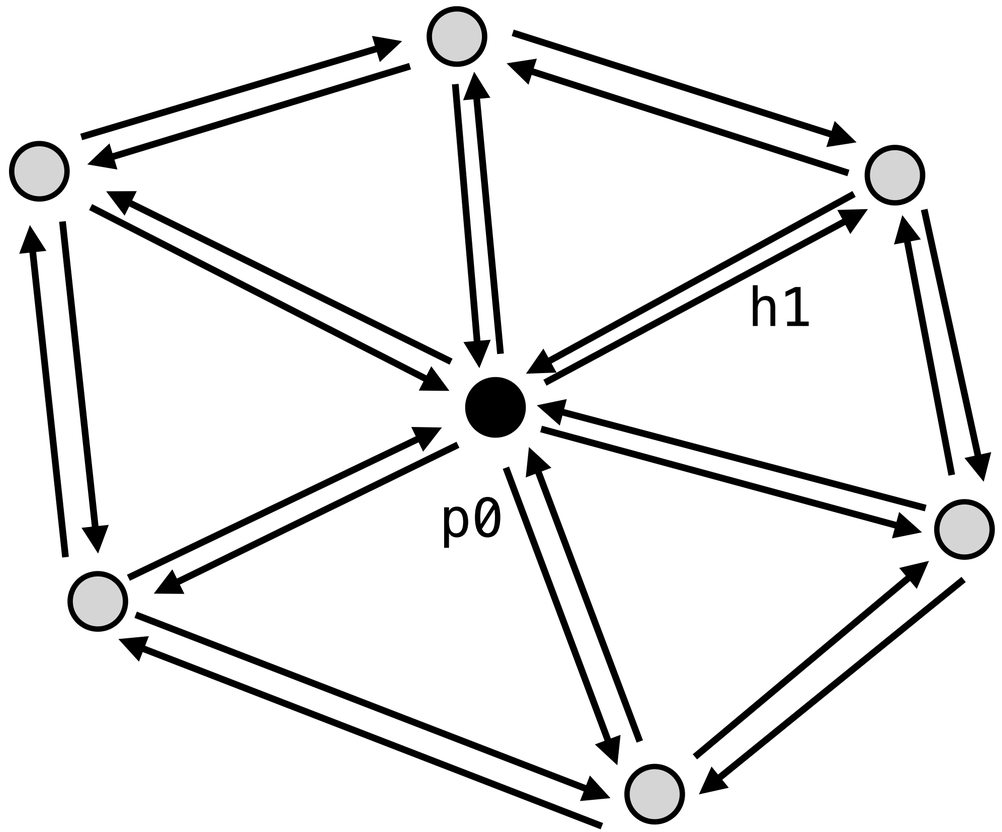
Example: Compute angles using `Surface_mesh`

```
Point p0 = mesh.position(v);
Mesh::Halfedge h1 = mesh.halfedge(v), hend = h1;
Normal result(0, 0, 0);
```



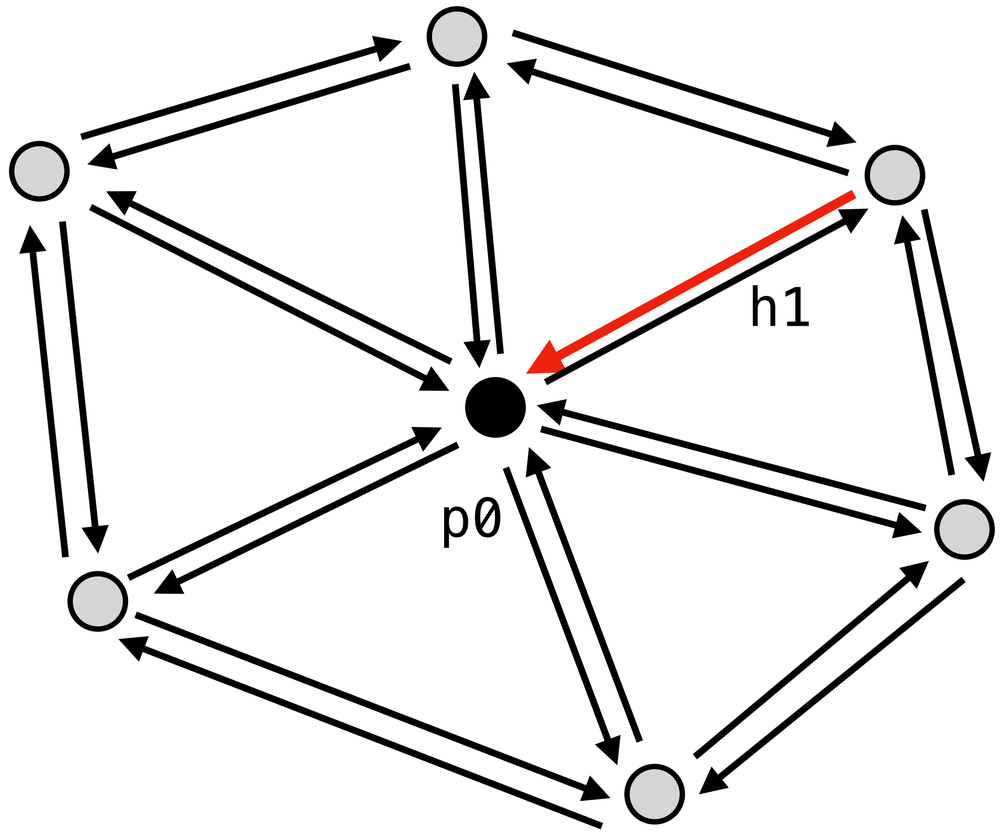
Example: Compute angles using `Surface_mesh`

```
Point p0 = mesh.position(v);
Mesh::Halfedge h1 = mesh.halfedge(v), hend = h1;
Normal result(0, 0, 0);
do {









 } while (h1 != hend)
```



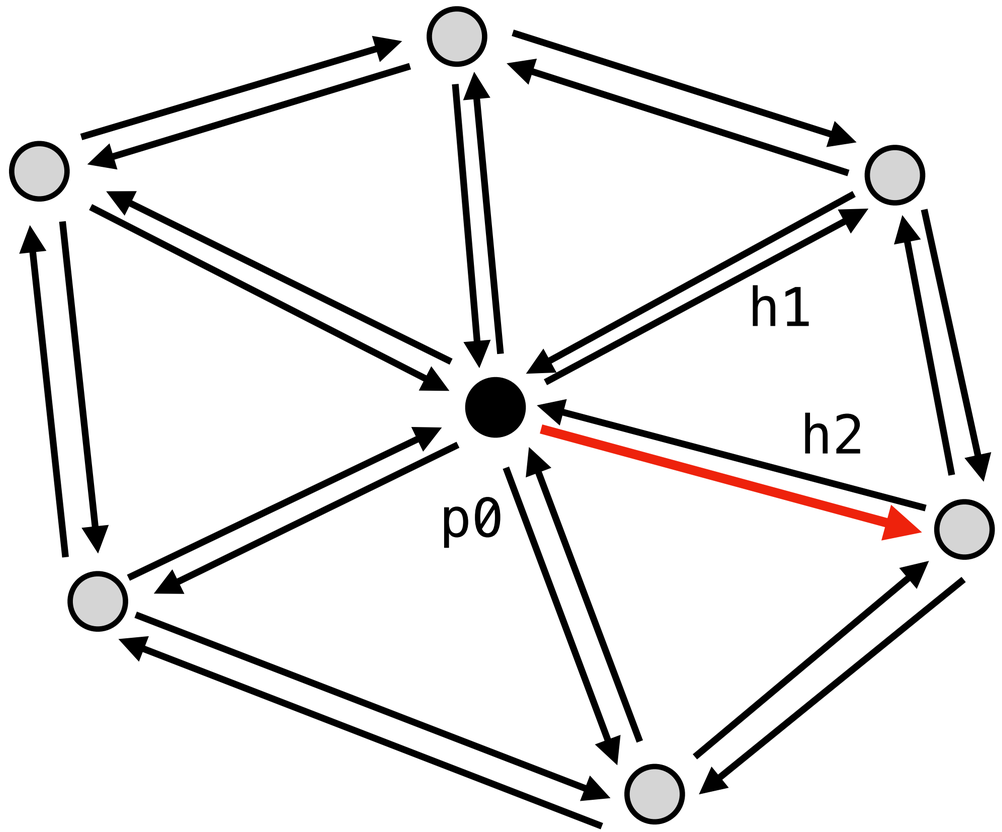
Example: Compute angles using `Surface_mesh`

```
Point p0 = mesh.position(v);
Mesh::Halfedge h1 = mesh.halfedge(v), hend = h1;
Normal result(0, 0, 0);
do {
    Mesh::Halfedge h2 = mesh.next_halfedge(
        mesh.opposite_halfedge(h1));









 } while (h1 != hend)
```



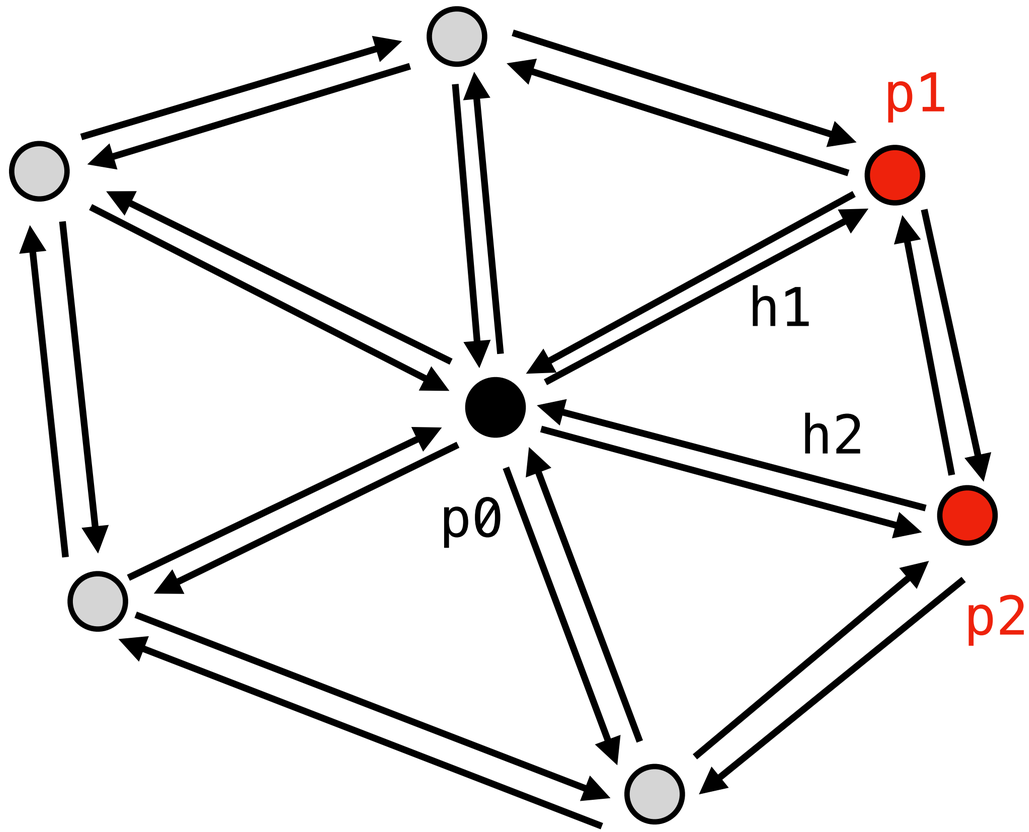
Example: Compute angles using `Surface_mesh`

```
Point p0 = mesh.position(v);
Mesh::Halfedge h1 = mesh.halfedge(v), hend = h1;
Normal result(0, 0, 0);
do {
    Mesh::Halfedge h2 = mesh.next_halfedge(
        mesh.opposite_halfedge(h1));









 } while (h1 != hend)
```



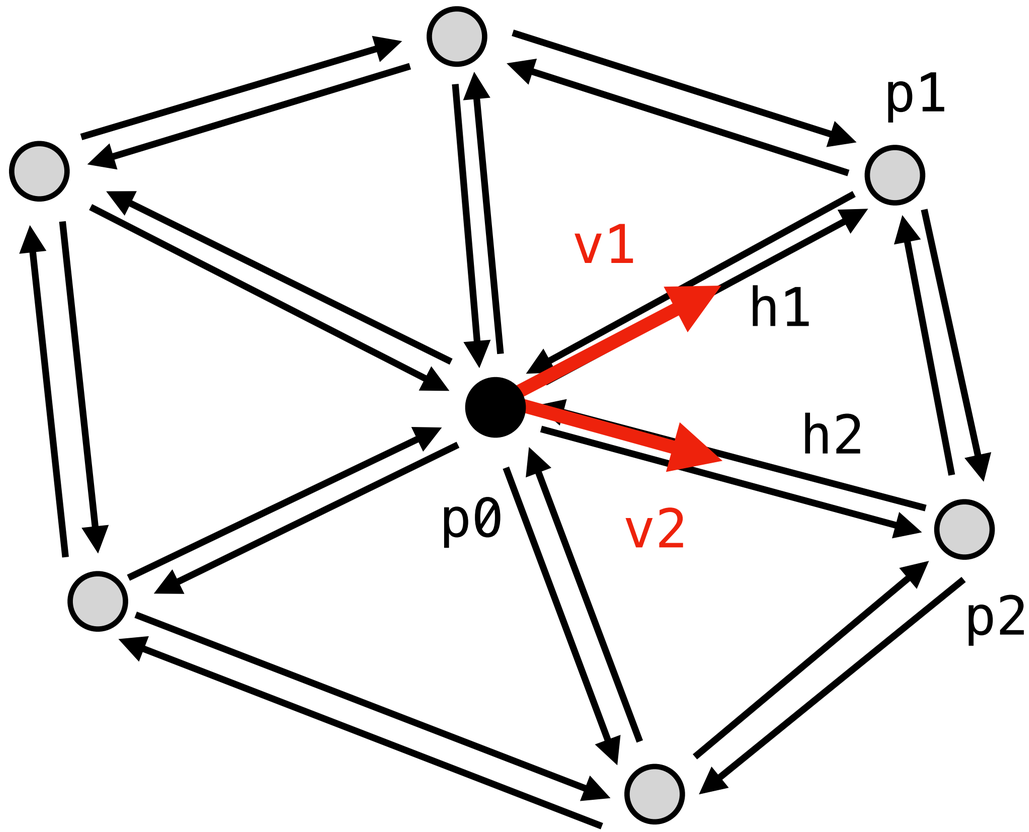
Example: Compute angles using `Surface_mesh`

```
Point p0 = mesh.position(v);
Mesh::Halfedge h1 = mesh.halfedge(v), hend = h1;
Normal result(0, 0, 0);
do {
    Mesh::Halfedge h2 = mesh.next_halfedge(
        mesh.opposite_halfedge(h1));
    Point p1 = mesh.position(mesh.to_vertex(h1)),
          p2 = mesh.position(mesh.to_vertex(h2));
    Vec3 v1 = normalize(p1 - p0),
         v2 = normalize(p2 - p0);









 } while (h1 != hend)
```



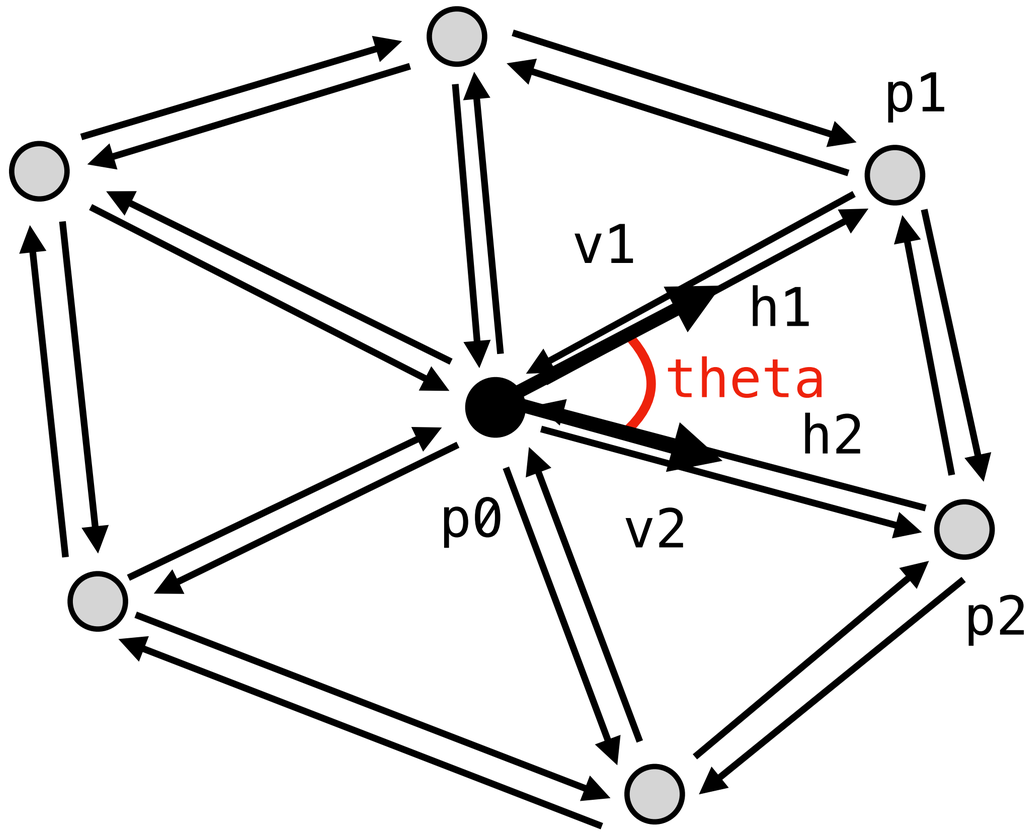
Example: Compute angles using `Surface_mesh`

```
Point p0 = mesh.position(v);
Mesh::Halfedge h1 = mesh.halfedge(v), hend = h1;
Normal result(0, 0, 0);
do {
    Mesh::Halfedge h2 = mesh.next_halfedge(
        mesh.opposite_halfedge(h1));
    Point p1 = mesh.position(mesh.to_vertex(h1)),
          p2 = mesh.position(mesh.to_vertex(h2));
    Vec3 v1 = normalize(p1 - p0),
         v2 = normalize(p2 - p0);









 } while (h1 != hend)
```



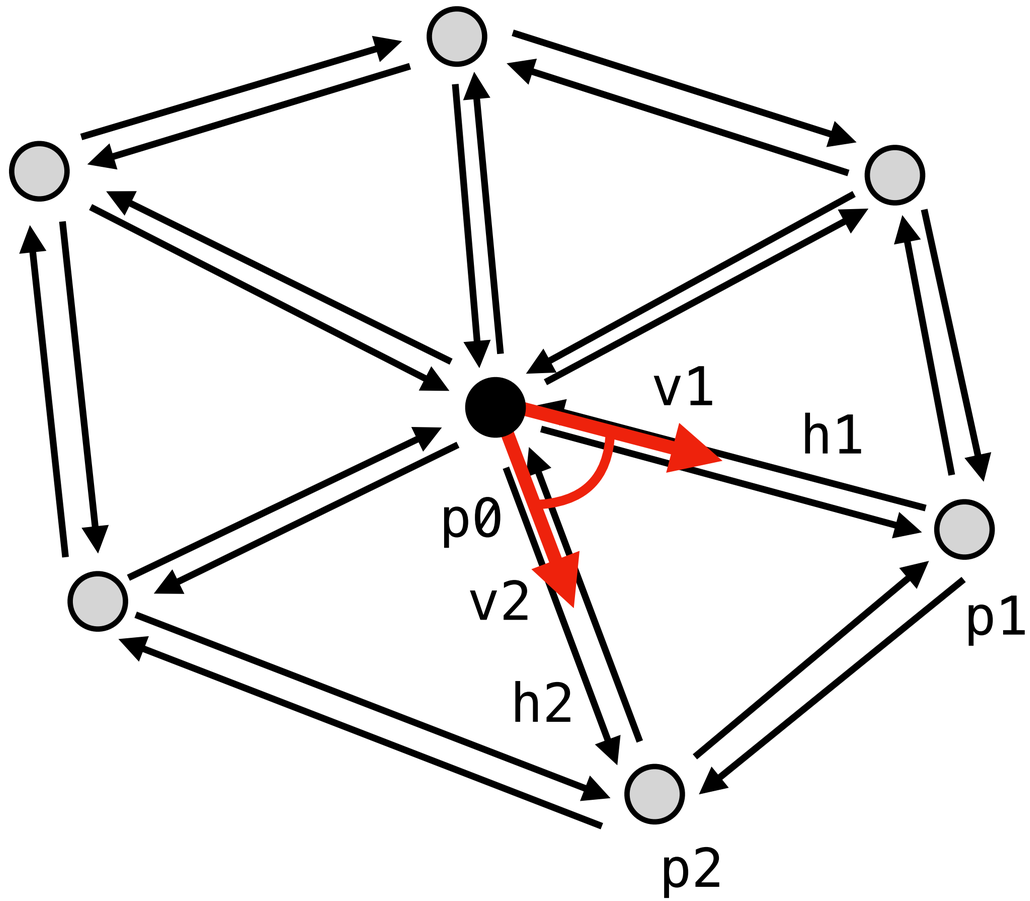
Example: Compute angles using `Surface_mesh`

```
Point p0 = mesh.position(v);
Mesh::Halfedge h1 = mesh.halfedge(v), hend = h1;
Normal result(0, 0, 0);
do {
    Mesh::Halfedge h2 = mesh.next_halfedge(
        mesh.opposite_halfedge(h1));
    Point p1 = mesh.position(mesh.to_vertex(h1)),
          p2 = mesh.position(mesh.to_vertex(h2));
    Vec3 v1 = normalize(p1 - p0),
         v2 = normalize(p2 - p0);
    Scalar theta = ... // Compute angle
    Normal n = ... // Compute face normal
    result += theta * n;

 } while (h1 != hend)
```



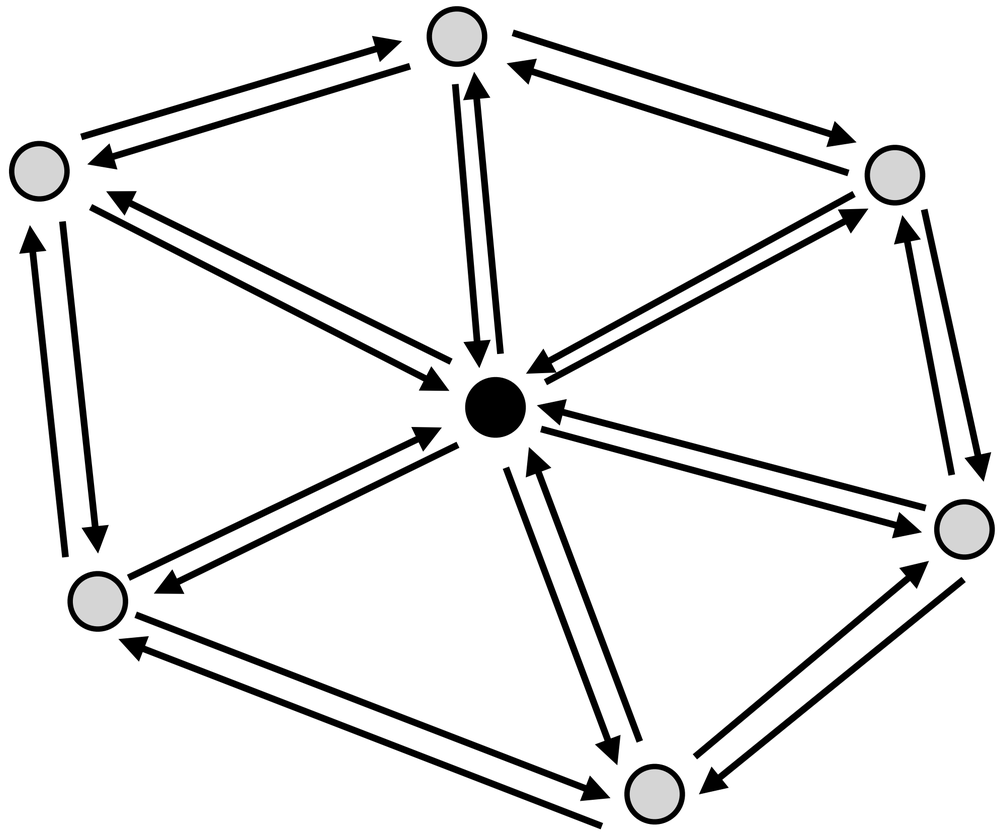
Example: Compute angles using `Surface_mesh`

```
Point p0 = mesh.position(v);
Mesh::Halfedge h1 = mesh.halfedge(v), hend = h1;
Normal result(0, 0, 0);
do {
    Mesh::Halfedge h2 = mesh.next_halfedge(
        mesh.opposite_halfedge(h1));
    Point p1 = mesh.position(mesh.to_vertex(h1)),
          p2 = mesh.position(mesh.to_vertex(h2));
    Vec3 v1 = normalize(p1 - p0),
         v2 = normalize(p2 - p0);
    Scalar theta = ... // Compute angle
    Normal n = ... // Compute face normal
    result += theta * n;
    h1 = h2;
 } while (h1 != hend)
```



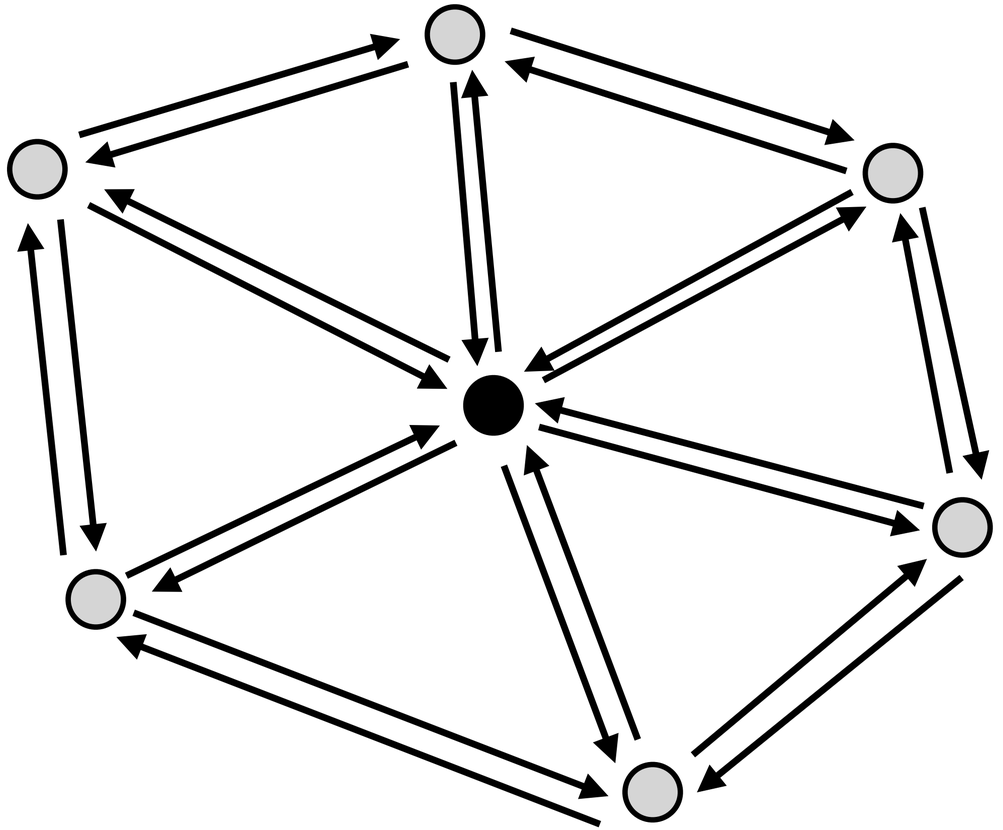
Example: Compute angles using `Surface_mesh`

```
Point p0 = mesh.position(v);
Mesh::Halfedge h1 = mesh.halfedge(v), hend = h1;
Normal result(0, 0, 0);
do {
    Mesh::Halfedge h2 = mesh.next_halfedge(
        mesh.opposite_halfedge(h1));
    Point p1 = mesh.position(mesh.to_vertex(h1)),
          p2 = mesh.position(mesh.to_vertex(h2));
    Vec3 v1 = normalize(p1 - p0),
         v2 = normalize(p2 - p0);
    Scalar theta = ... // Compute angle
    Normal n = ... // Compute face normal
    result += theta * n;
    h1 = h2;
 } while (h1 != hend)
 result = normalize(result);
```



Example: Compute angles using `Surface_mesh`

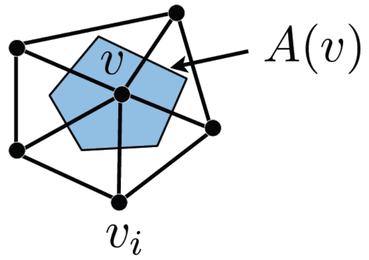
```
Point p0 = mesh.position(v);
Mesh::Halfedge h1 = mesh.halfedge(v), hend = h1;
Normal result(0, 0, 0);
do {
    Mesh::Halfedge h2 = mesh.next_halfedge(
        mesh.opposite_halfedge(h1));
    Point p1 = mesh.position(mesh.to_vertex(h1)),
          p2 = mesh.position(mesh.to_vertex(h2));
    Vec3 v1 = normalize(p1 - p0),
         v2 = normalize(p2 - p0);
    Scalar theta = ... // Compute angle
    Normal n = ... // Compute face normal
    result += theta * n;
    h1 = h2;
 } while (h1 != hend)
 result = normalize(result);
```

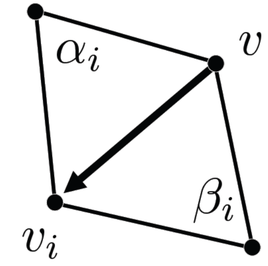


2) Discrete Curvature

- Uniform Laplace Operator

$$L_U(v) = \frac{1}{N} \sum_{i}^{|N|} (v_i - v)$$



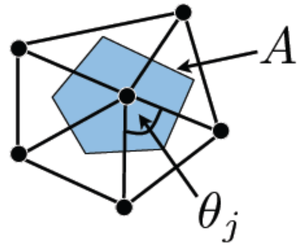


- Laplace-Beltrami Curvature

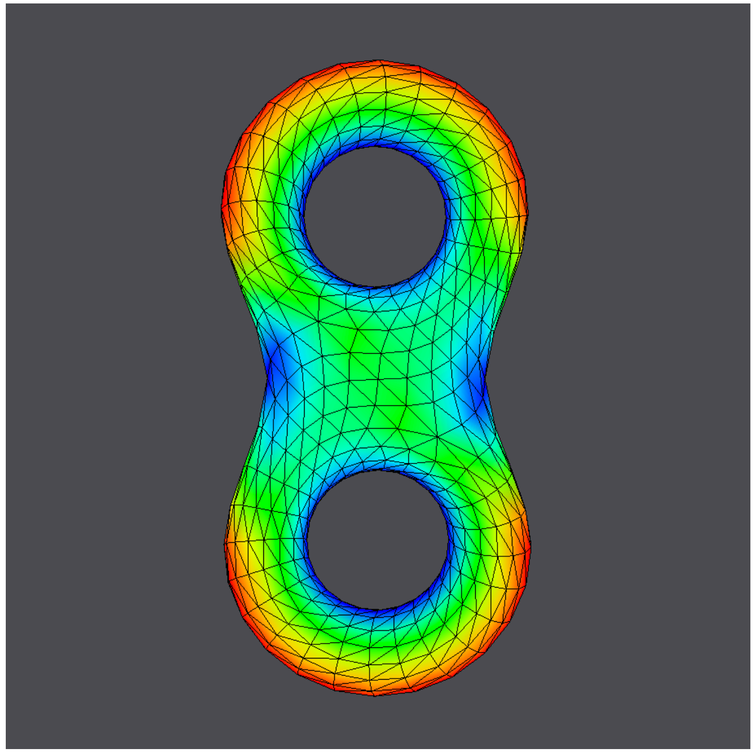
$$L_B(v) = \frac{1}{2A} \sum_{i}^{|N|} (\cot \alpha_i + \cot \beta_i)(v_i - v)$$

- Gaussian Curvature

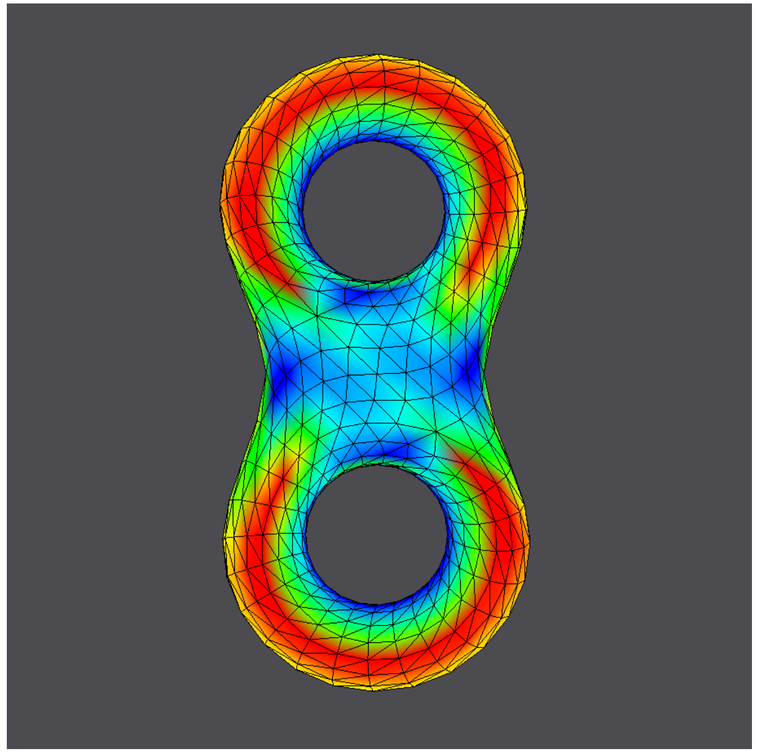
$$K = (2\pi - \sum_{j} \theta_j)/A$$



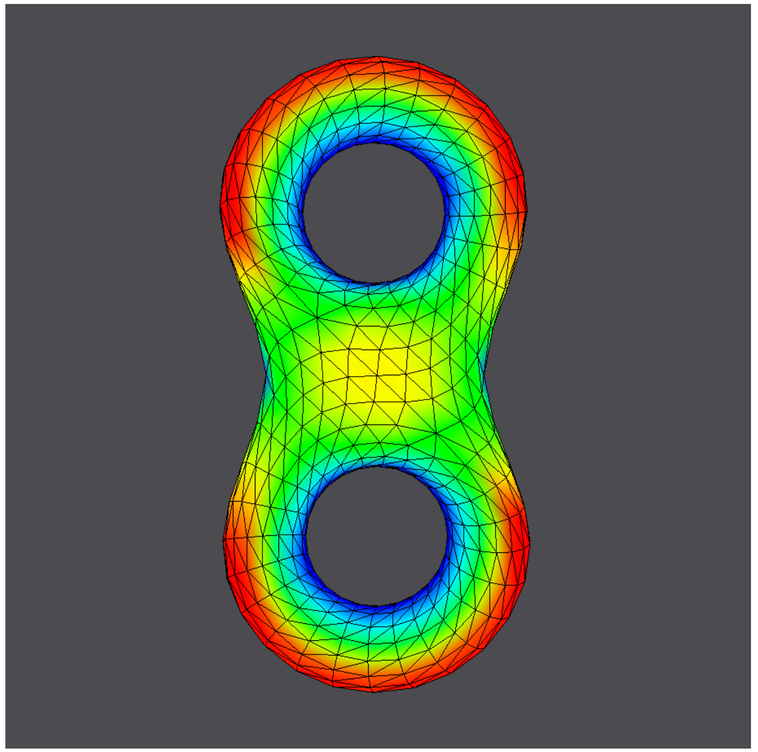
2) Discrete Curvature



Uniform Laplacian curvature



Laplace-Beltrami approximation of mean curvature



Gaussian curvature

Questions?