A2: MeshEdit

“I hate meshes. I cannot believe how hard this is.

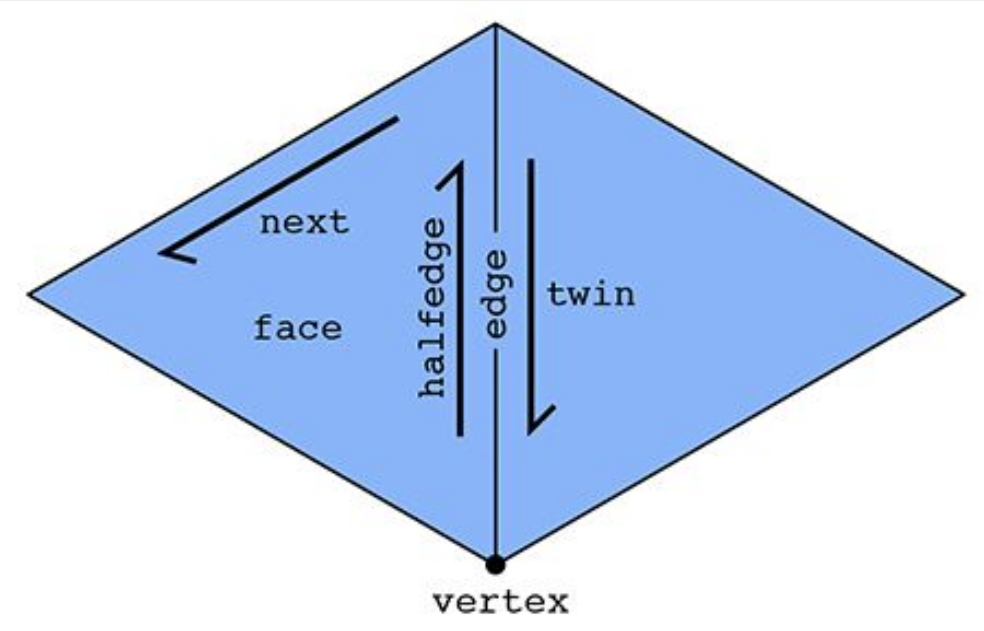
Geometry is hard.”

— David Baraff, Senior Research Scientist, Pixar Animation Studios

- The Half-Edge Data Structure
- Local Mesh Operations
- Debugging Strategies
- Bisect Edge Coding Demo

- The Half-Edge Data Structure
- Local Mesh Operations
- Debugging Strategies
- Bisect Edge Coding Demo

Halfedge



Halfedge Data Structure

```
class Halfedge {
public:
        //connectivity:
        HalfedgeRef twin; //halfedge on the other side of edge
        HalfedgeRef next; //next halfedge ccw around face
        VertexRef vertex; //vertex this halfedge is leaving
        EdgeRef edge; //edge this halfedge is half of
        FaceRef face; //face this halfedge borders
```

Vertex

```
class Vertex {
public:
        //connectivity:
        HalfedgeRef halfedge; //some halfedge that starts at this vertex

        //data:
        uint32_t id; //unique-within-the-mesh ID for this vertex
        Vec3 position; //location of the vertex

        //helpers:
        bool on_boundary() const; //is vertex on a boundary loop?
        uint32_t degree() const; //number of faces adjacent to the vertex, including boundary faces
        Vec3 normal() const; //area-weighted normal vector at the vertex (excluding any boundary faces)
        Vec3 neighborhood_center() const; //center of adjacent vertices
        float angle_defect() const; //difference between sum of adjacent face angles and 2pi
        float gaussian_curvature() const; //product of principle curvatures
```

Edge

```
class Edge {
public:
        //connectivity:
        HalfedgeRef halfedge; //one of the two halfedges adjacent to this edge

        //data:
        uint32_t id; //unique-in-this-mesh id
        bool sharp = false; //should this edge be considered sharp when computing shading normals?

        //helpers:
        bool on_boundary() const; // Is edge on a boundary loop?
        Vec3 center() const; // The midpoint of the edge
        Vec3 normal() const; // The average of the face normals on either side of this edge
        float length() const; // The length of the edge
```

Face

```
class Face {
public:
        //connectivity:
        HalfedgeRef halfedge; // some halfedge in this face

        //data:
        uint32_t id; //unique-in-this-mesh id
        bool boundary = false; //is this a boundary loop?

        //helpers:
        Vec3 center() const; // centroid of this face
        Vec3 normal() const; // area-weighted face normal
        uint32_t degree() const; // number of vertices/edges in this face
        float area() const; // area of this face
```

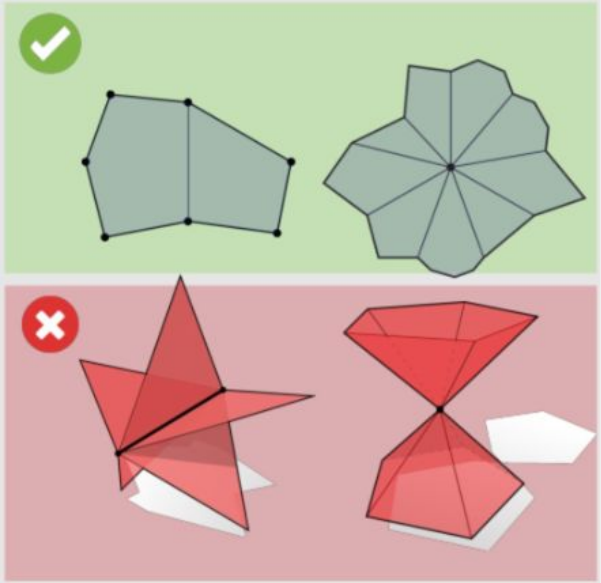
Iterating Over Halfedges

```
void printVertexPositions(FaceRef f) {
  HalfedgeRef h = f->halfedge; // get the first halfedge of the face
  do {
    VertexRef v = h->vertex;   // get the vertex of the current halfedge
    std::cout << v->position << std::endl;      // print the vertex position
    h = h->next;               // move to the next halfedge around the face
  } while (h != f->halfedge);  // keep going until we're back at the beginning
}
```

Iterating Over Halfedges

```
void printNeighborPositions(VertexRef v) {
  HalfedgeRef h = v->halfedge;    // get one of the outgoing halfedges of the vertex
  do {
    HalfedgeRef h_twin = h->twin;   // get the vertex of the current halfedge
    VertexRef vN = h_twin->vertex;  // vertex is 'source' of the half edge.
                                      // so h->vertex is v,
                                      // whereas h_twin->vertex is the neighbor vertex.
    std::cout << vN->position << std::endl;          // print the vertex position
    h = h_twin->next;               // move to the next outgoing halfedge of the vertex.
  } while(h != v->halfedge);        // keep going until we're back at the beginning
}
```

Manifold vs Non Manifold

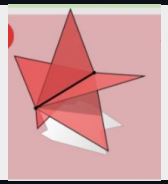


Blender BMesh (for non manifold meshes):
https://developer.blender.org/docs/features/objects/mesh/bmesh/

Mesh Validity Rules

1. The mesh is self-contained. All pointers are to elements in the vertices, edges, faces, or halfedges lists.
2. Edges correspond to twinned halfedges. That is, for every edge `e`, `e->halfedge(->twin)^n` is a cycle of exactly two halfedges, and these are exactly the halfedges with `h->edge` equal to `e`.

^No fins !



Exactly two halfedges lie on each edge

3. Faces correspond to cycles of halfedges. That is, for every face `f`, `f->halfedge(->next)^n` is a cycle of at least three halfedges, and these are exactly the halfedges with `h->face` equal to `f`.
4. Vertices correspond to stars of halfedges. That is, for every vertex `v`, `v->halfedge(->twin->next)^n` is a cycle of at least two halfedges, and these are exactly the halfedges with `h->vertex` equal to `v`.
5. Vertices are not orphaned, nor is the surface adjacent to them hourglass-shaped. That is, vertices are adjacent to at least one non-boundary face and at most one boundary face.

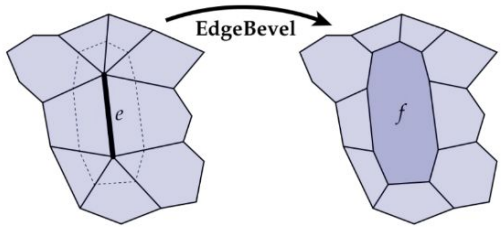
^One fan!

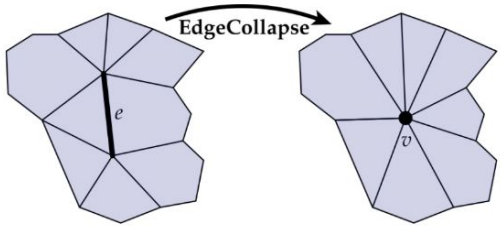
Each vertex can be adjacent to at most one boundary face.

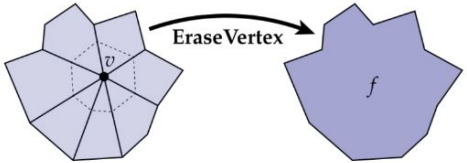
6. Edges not orphaned. That is, edges are adjacent to at least one non-boundary face.
7. Faces are simple. That is, faces touch each vertex and edge at most once.

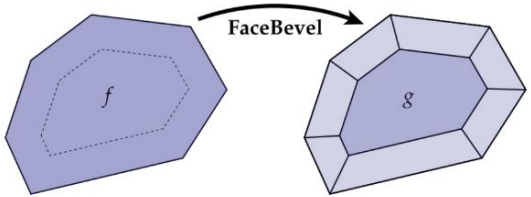
- The Half-Edge Data Structure
- Local Mesh Operations
- Debugging Strategies
- Bisect Edge Coding Demo

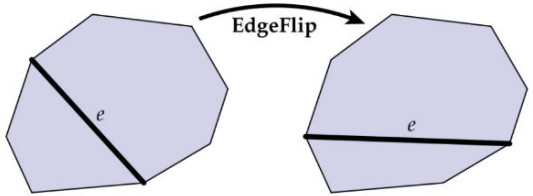
Local Mesh Operations

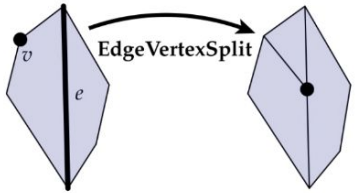


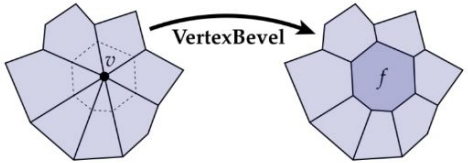


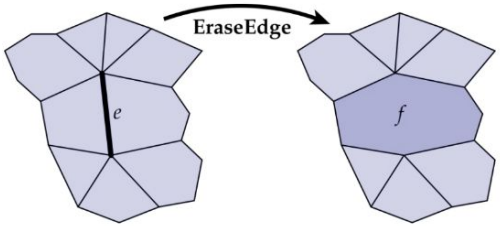


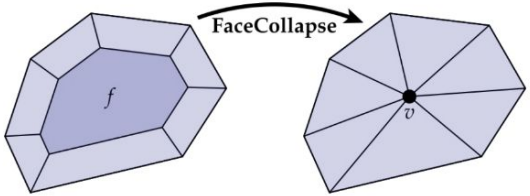








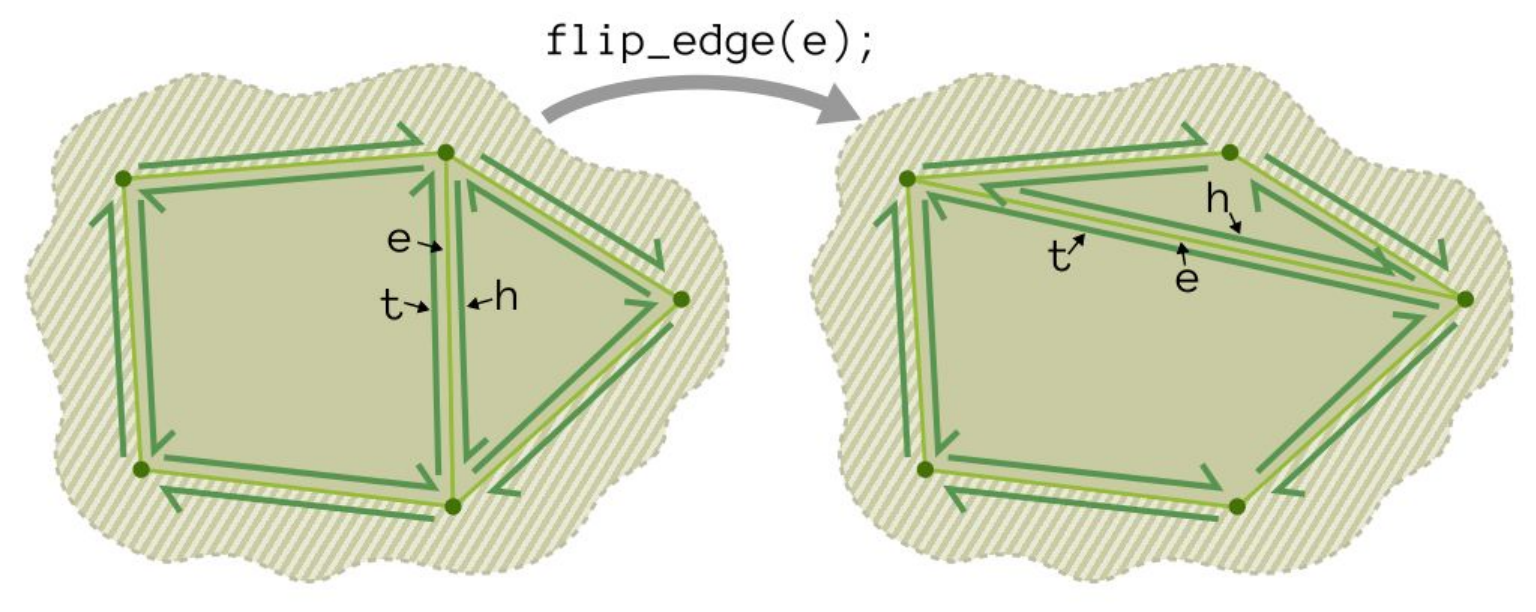




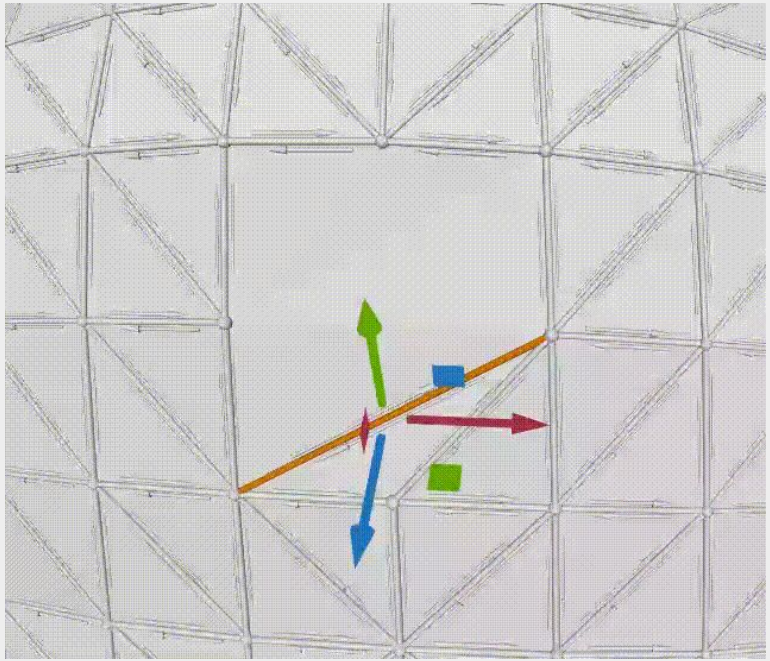
PSA:

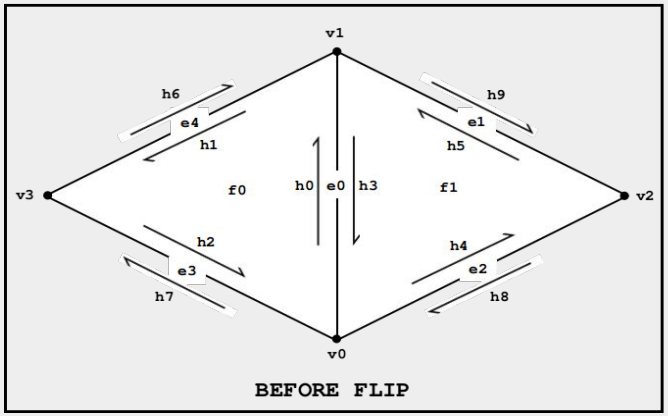
The diagrams in the documentation do not faithfully represent all the cases! Think about different meshes and draw out some diagrams when you are implementing MeshEdit.

Flip Edge

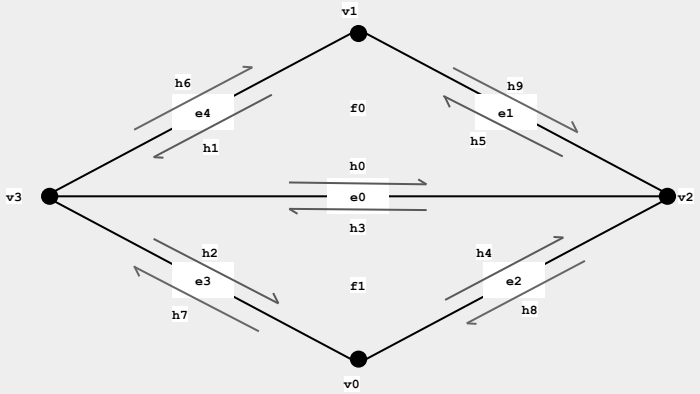


Flip Edge



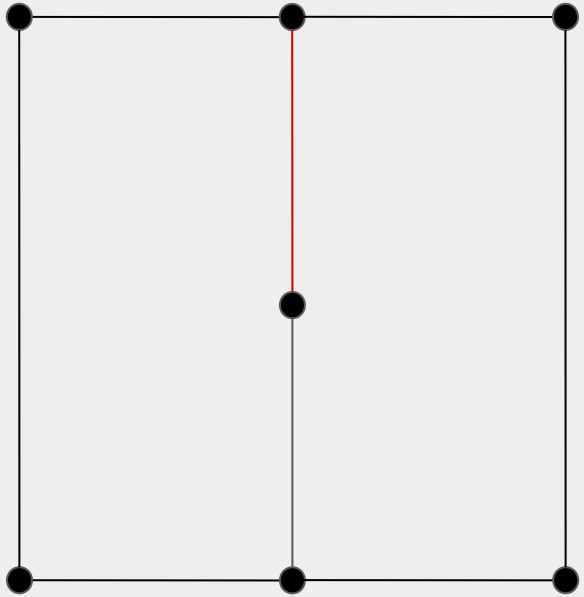


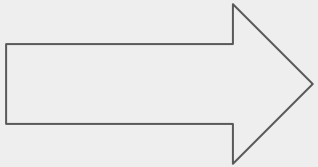
BEFORE FLIP



AFTER FLIP

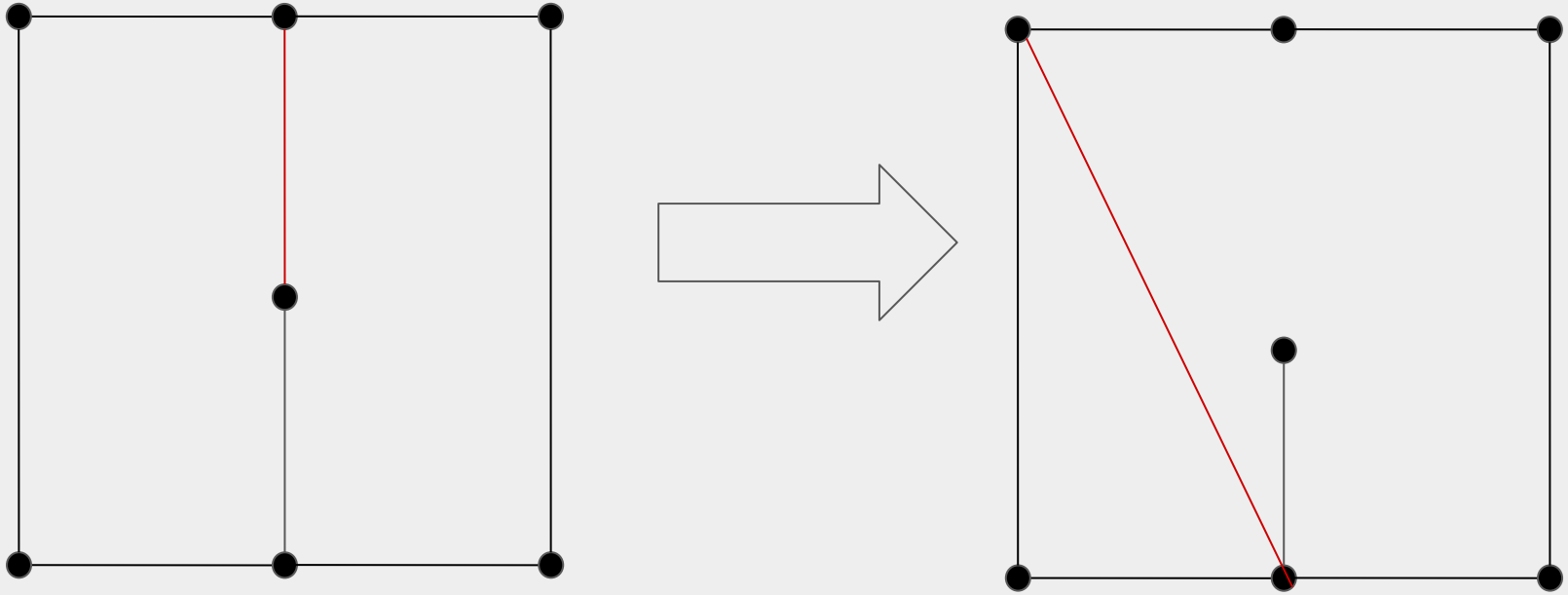
Flip Edge Edge Case Example



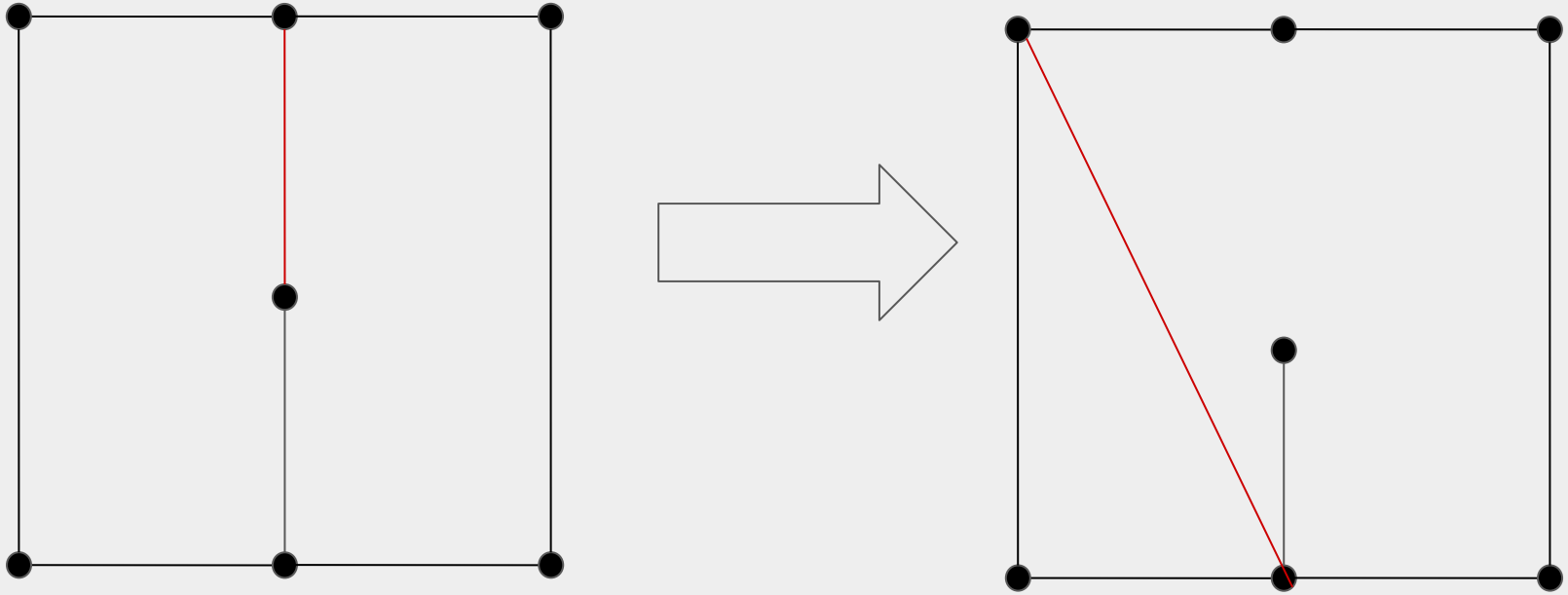


??

Flip Edge Edge Case Example



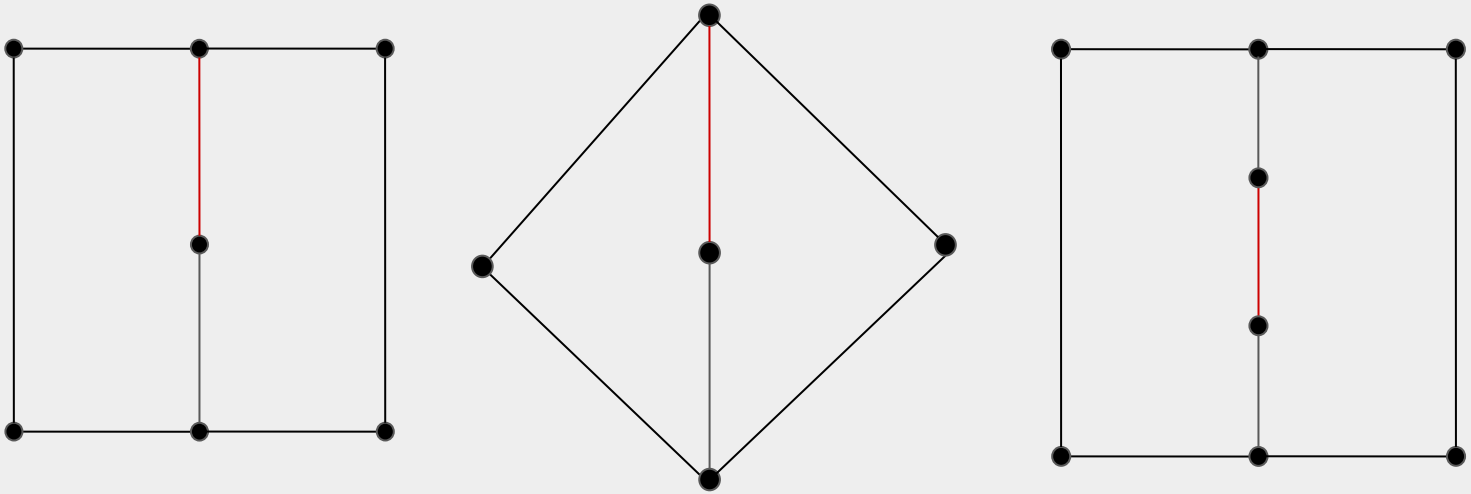
Flip Edge Edge Case Example



REJECT

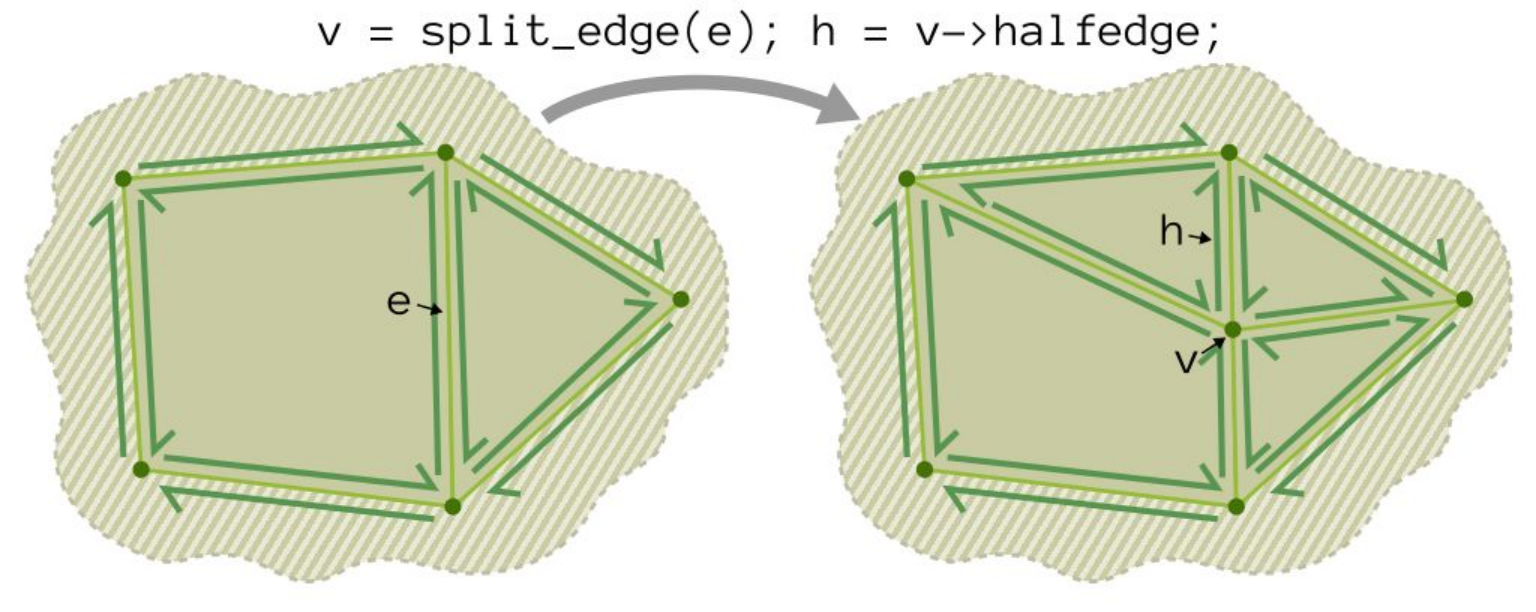
Flip Edge Edge Case Example

Make sure to generalize edge cases – don't hard code!

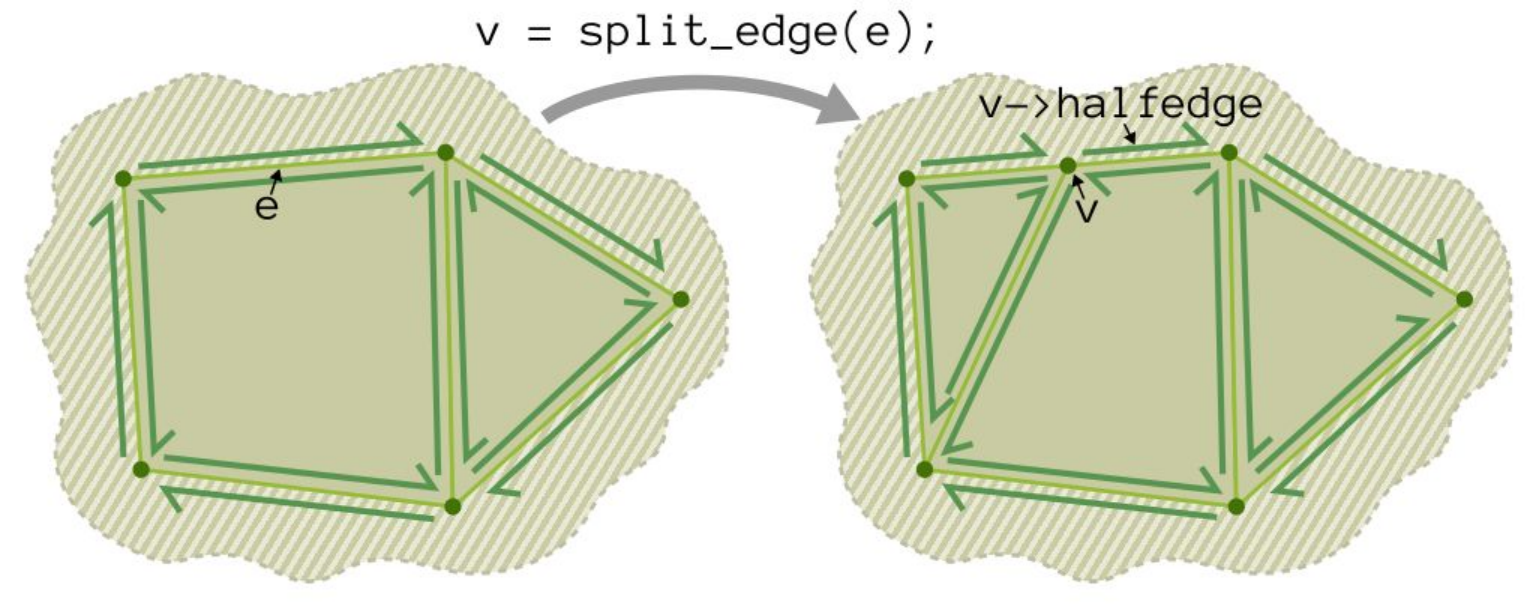


ALL SHOULD REJECT

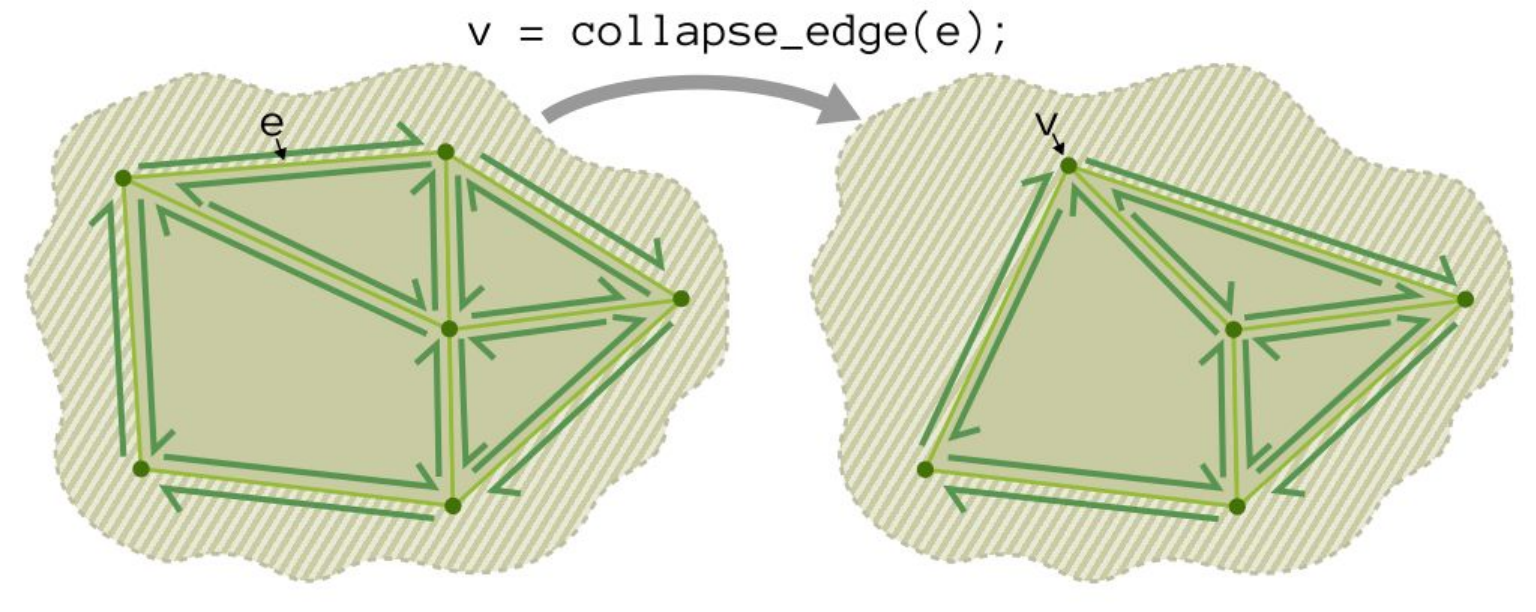
Split Edge



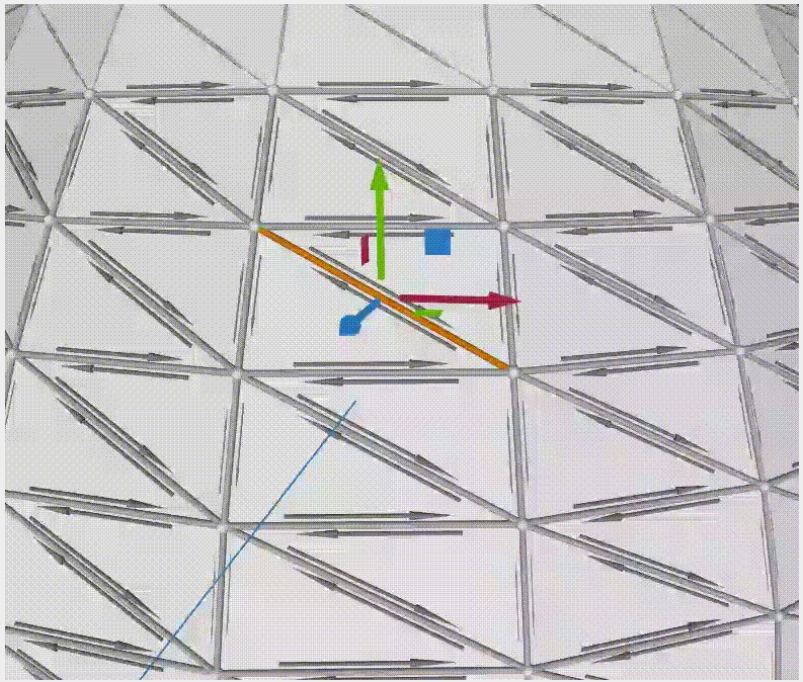
Split Edge

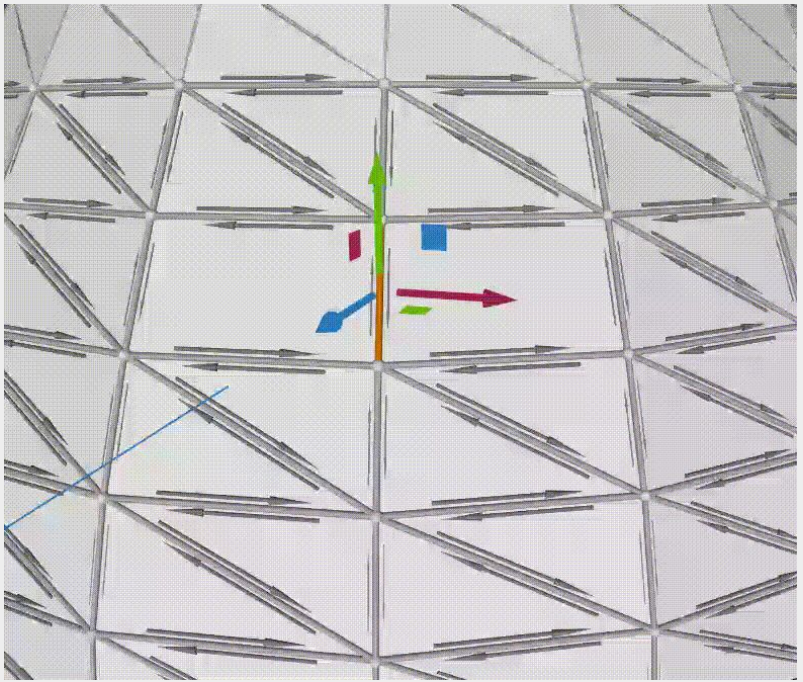


Collapse Edge

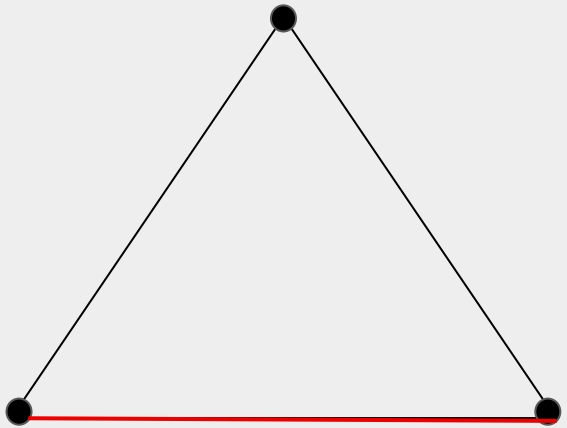


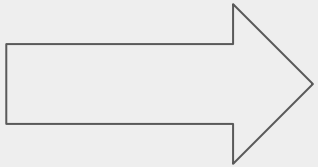
Collapse Edge





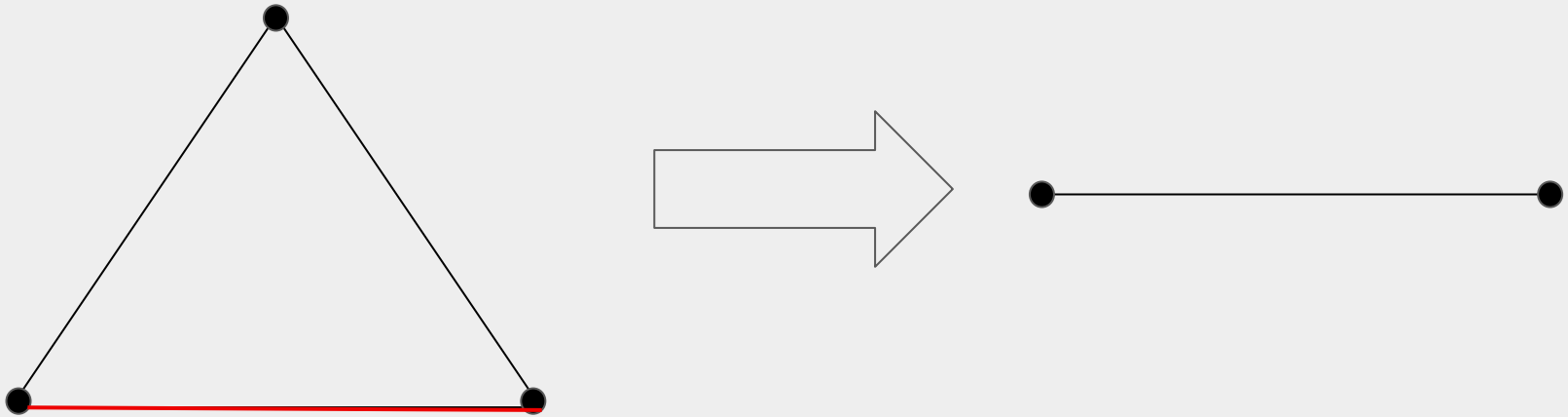
Collapse Edge Edge Case Example (Easy)



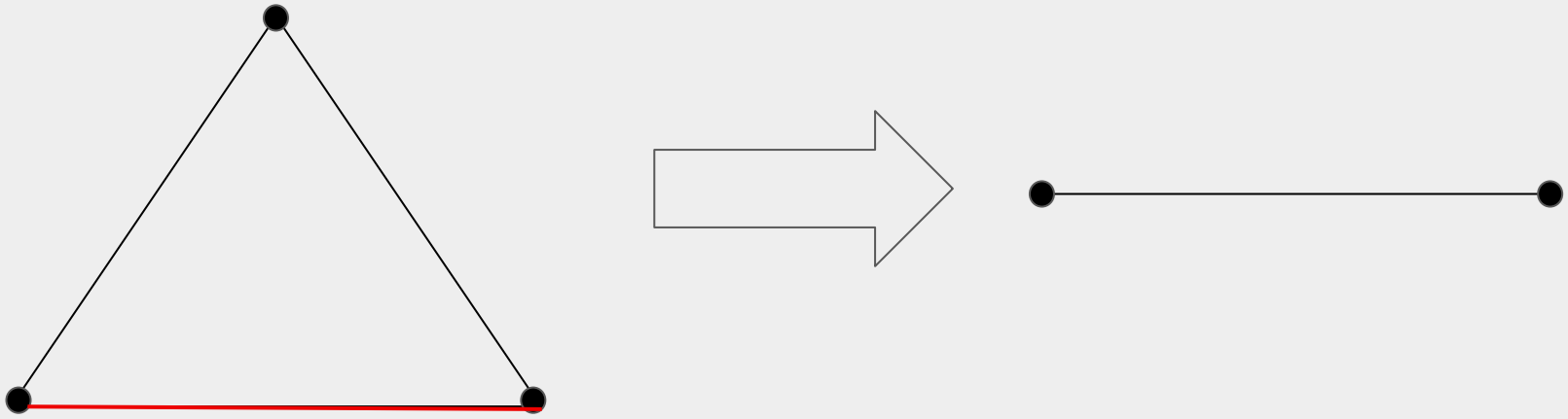


??

Collapse Edge Edge Case Example (Easy)

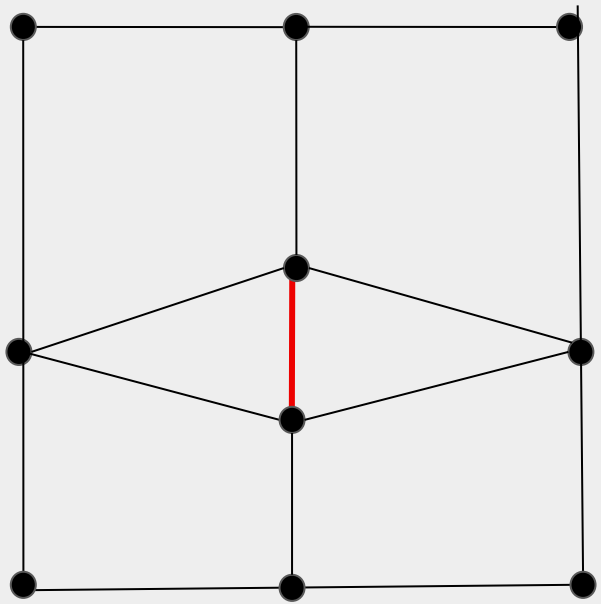


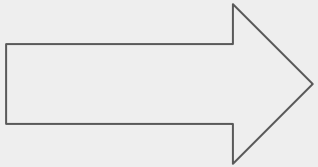
Collapse Edge Edge Case Example (Easy)



REJECT

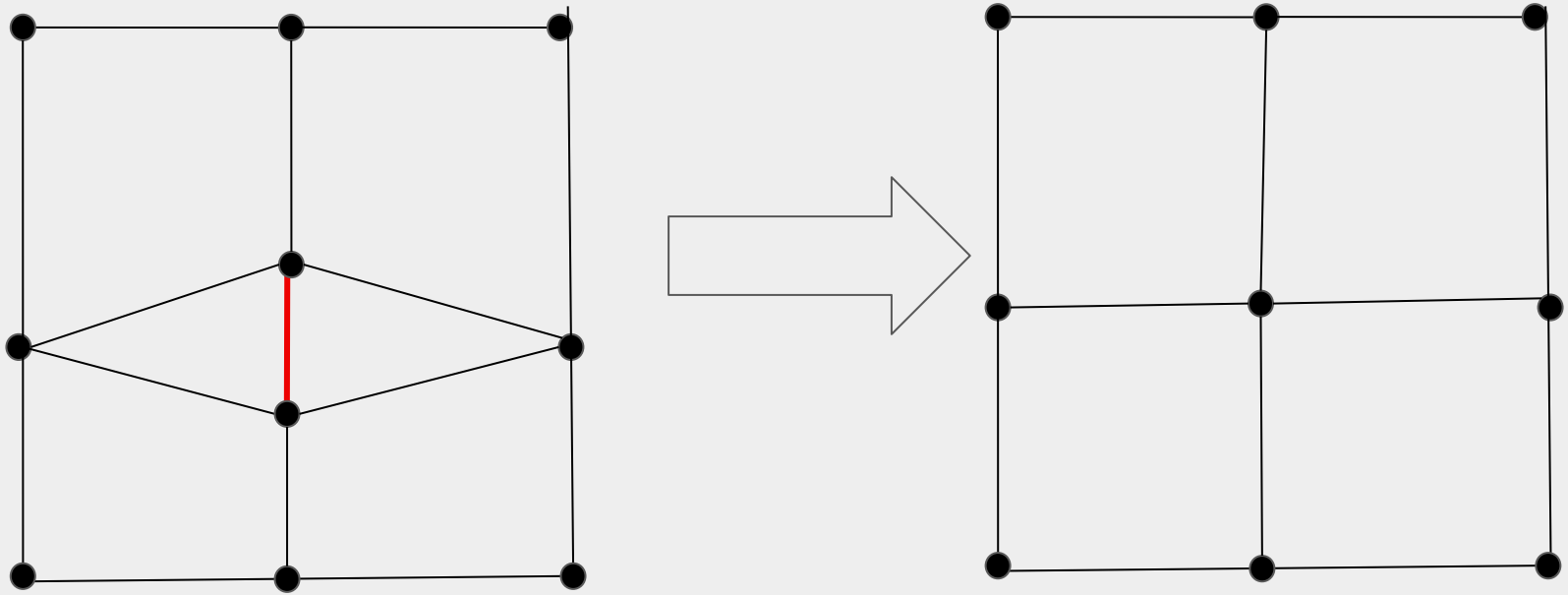
Collapse Edge Edge Case Example (Hard)



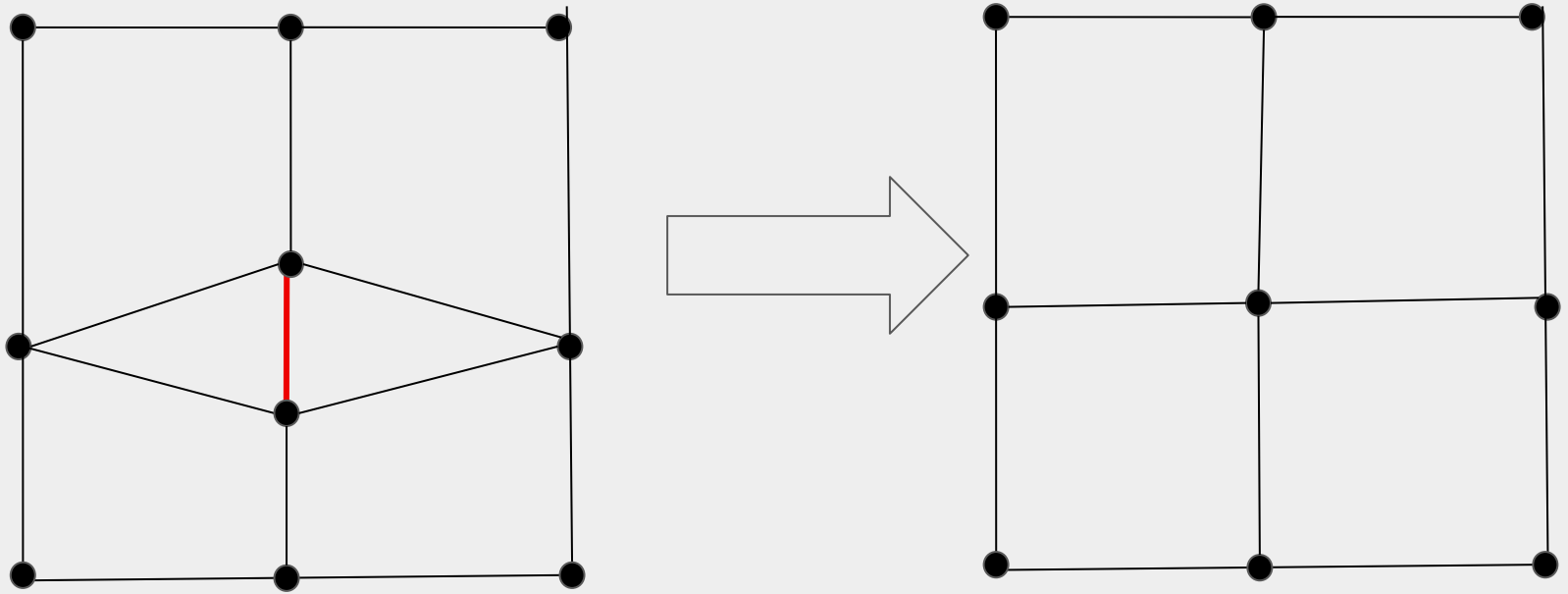


??

Collapse Edge Edge Case Example (Hard)

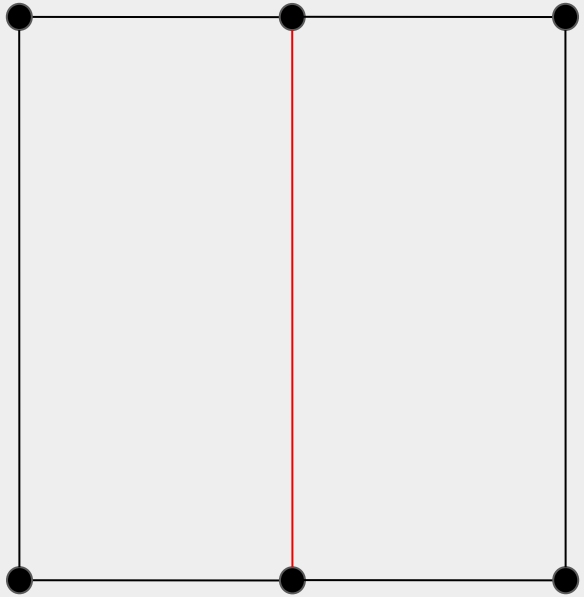


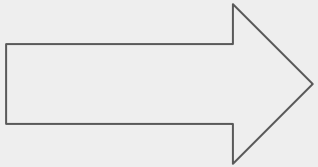
Collapse Edge Edge Case Example (Hard)



ACCEPT

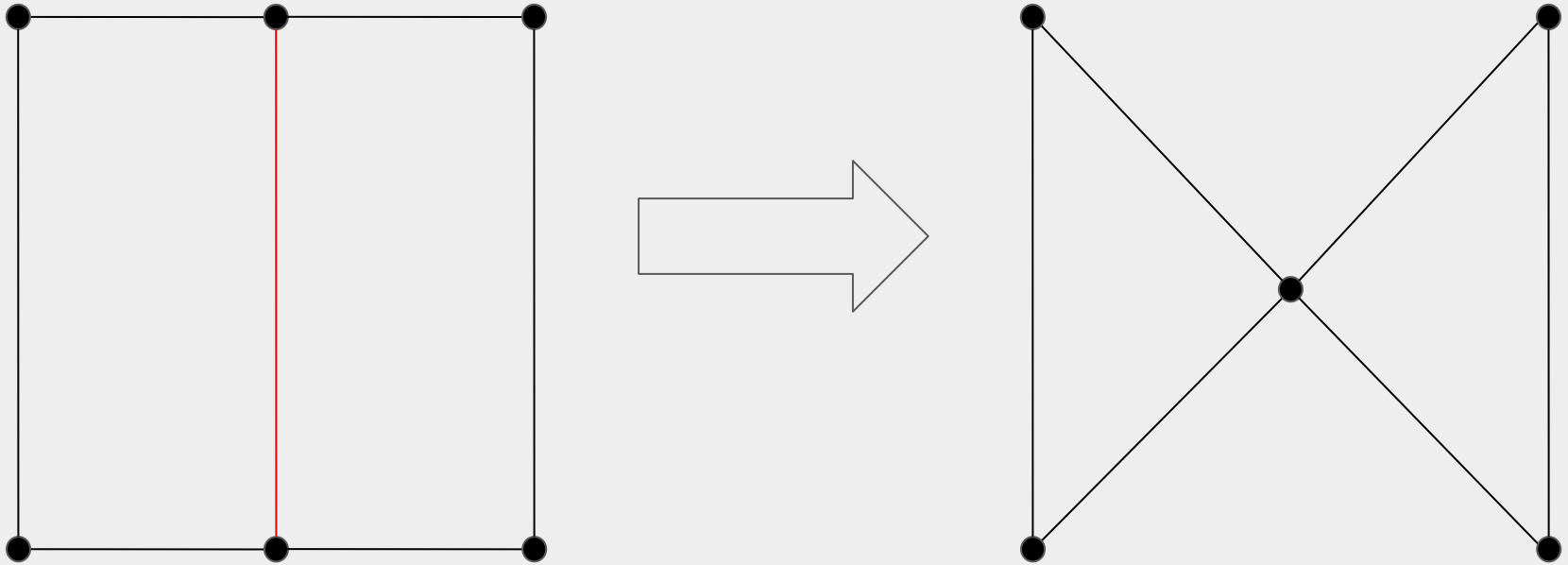
Collapse Edge Edge Case Example (Hard)



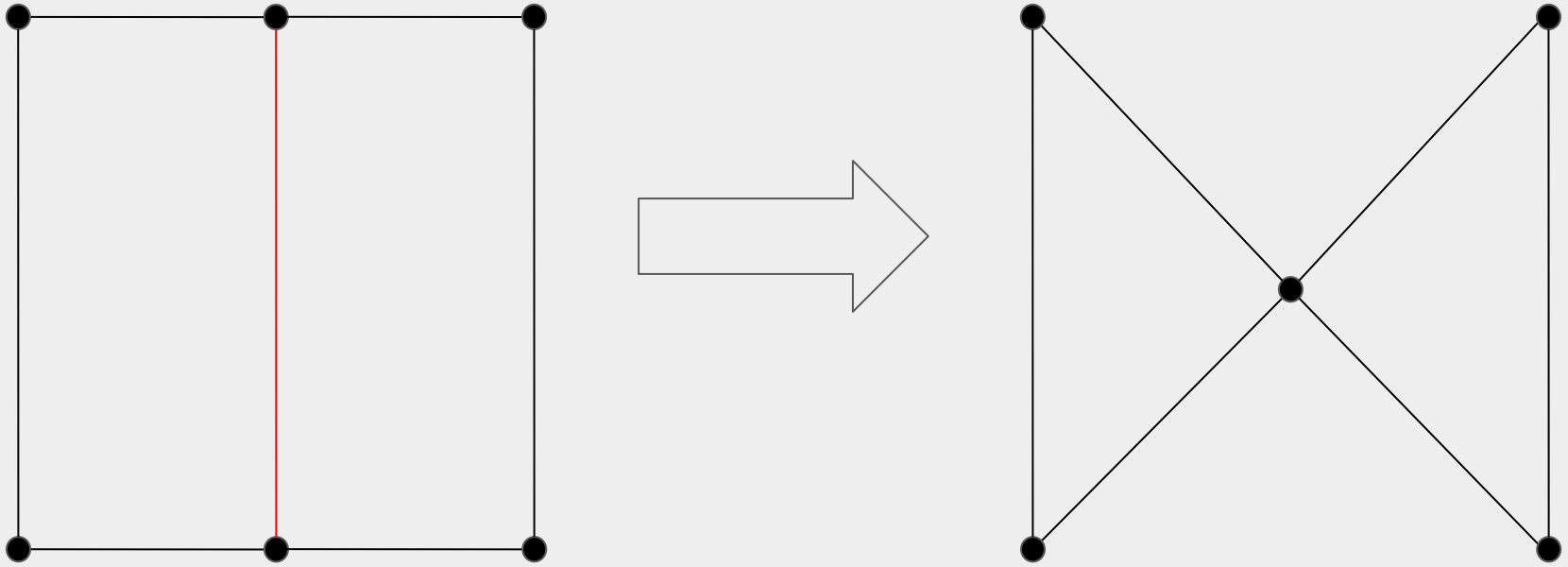


??

Collapse Edge Edge Case Example (Hard)

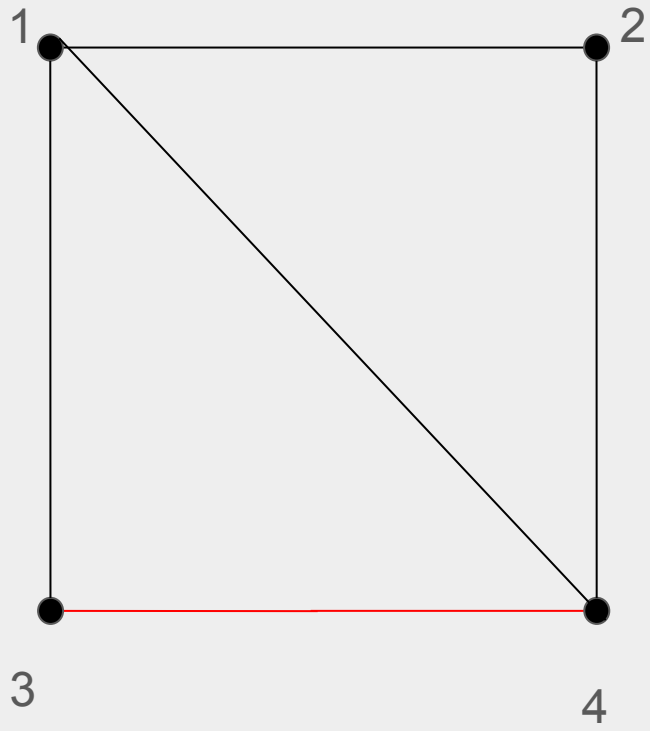


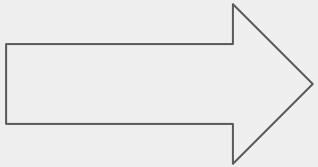
Collapse Edge Edge Case Example (Hard)



REJECT

Collapse Edge Edge Case Example (Hard)

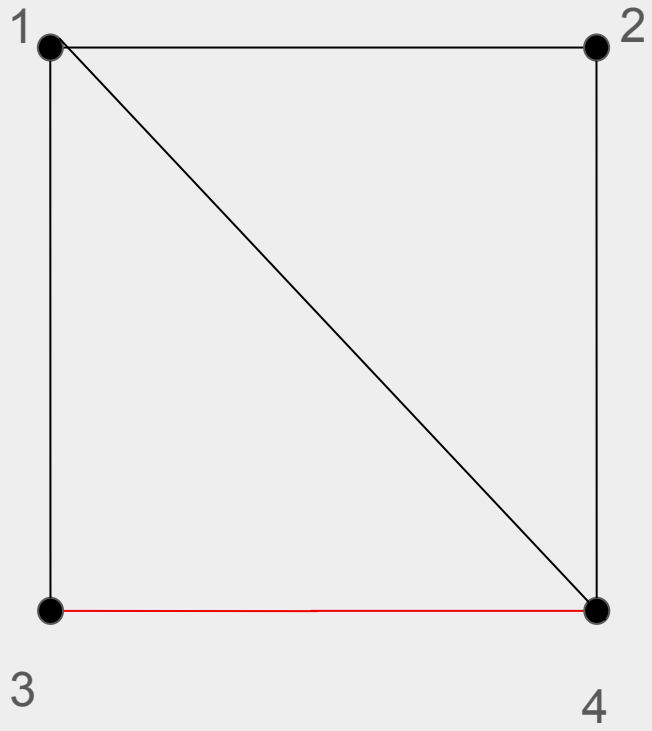




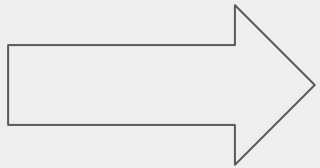
??

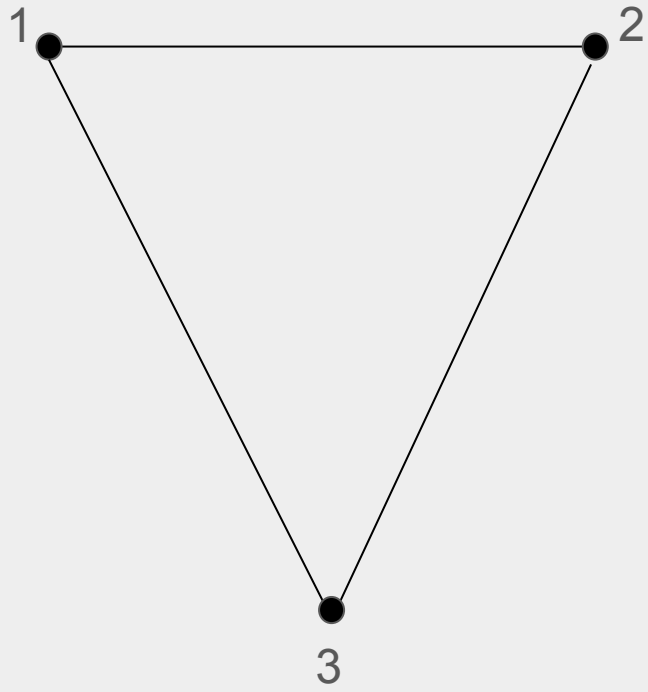
Faces: (1, 3, 4), (1, 2, 4), **(1, 2, 3, 4)**

Collapse Edge Edge Case Example (Hard)



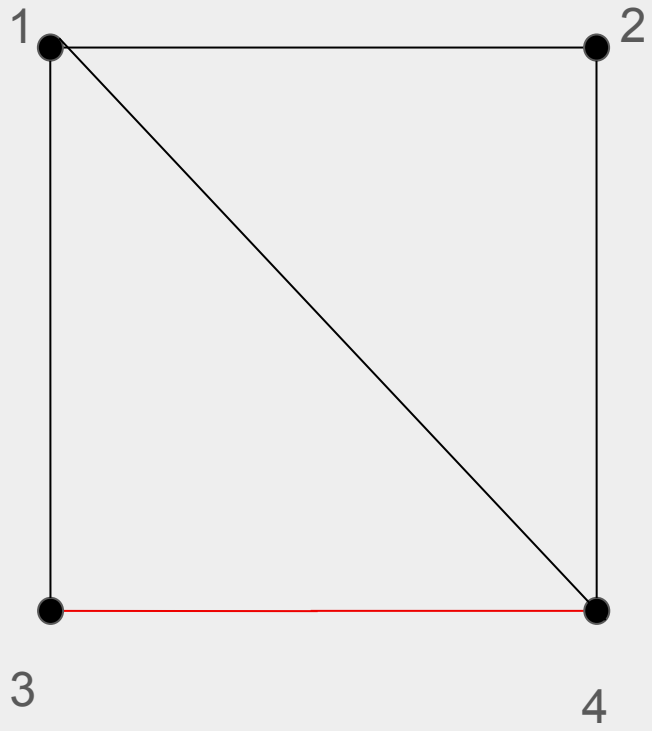
Faces: (1, 3, 4), (1, 2, 4), **(1, 2, 3, 4)**





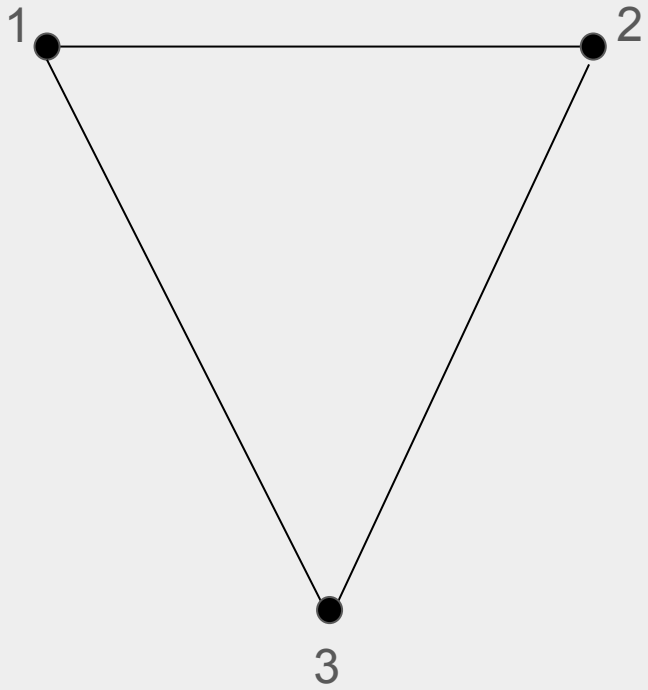
Faces: (1, 2, 3), (3, 2, 1)

Collapse Edge Edge Case Example (Hard)



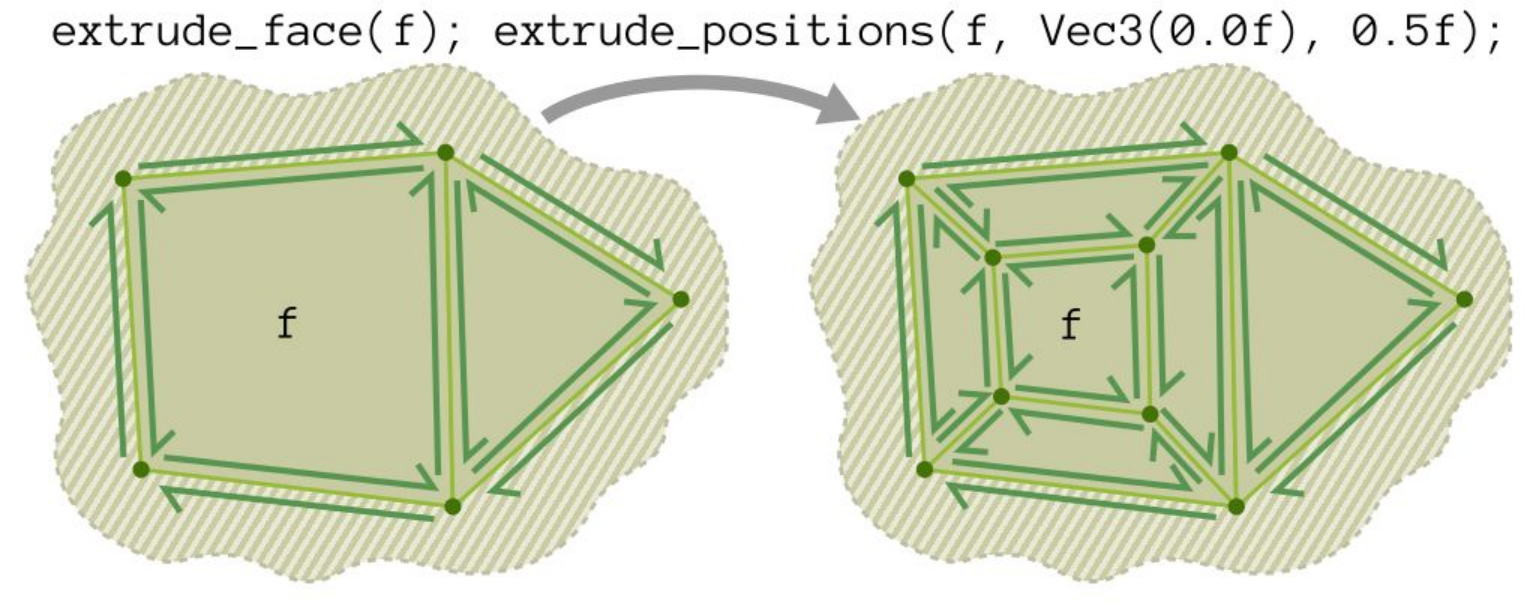
Faces: (1, 3, 4), (1, 4, 2), **(4, 3, 1, 2)**

ACCEPT

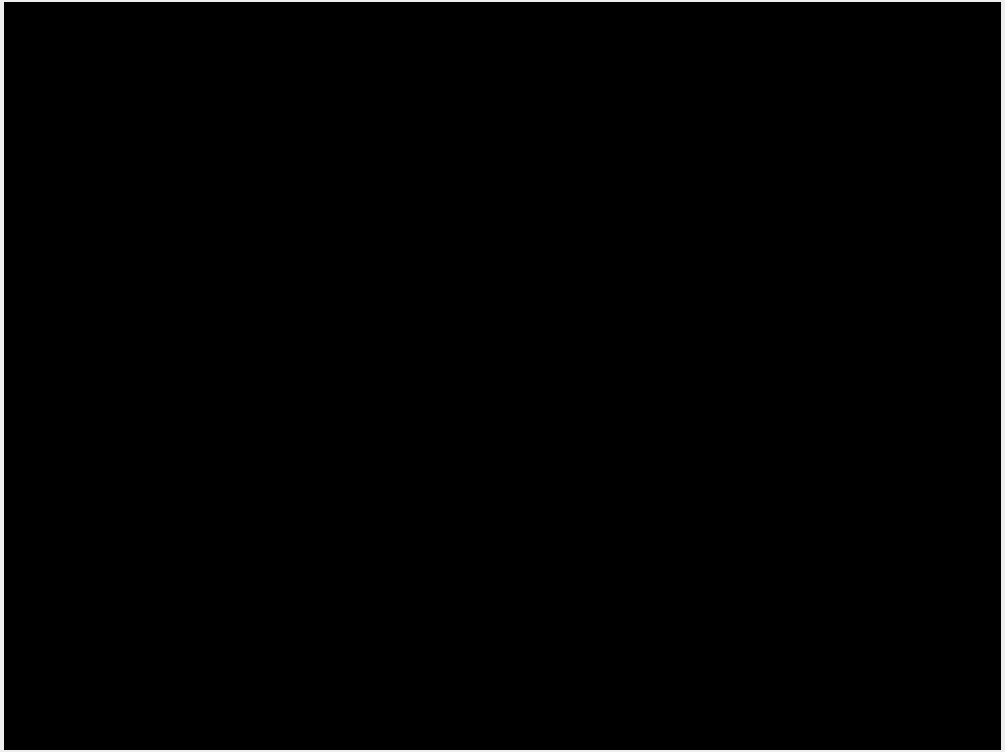


Faces: (3, 2, 1), (1, 2, 3)

Extrude Face



Extrude Face

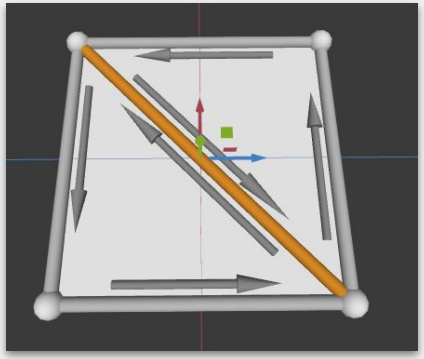


More Notes on Edge Cases

- Your operations must maintain the validity of the mesh
  a. If it is not possible to perform the operation and remain valid, you should not perform it
  b. Your operations should never crash - you should fail “gracefully”
- Don’t hard code edge cases – find the pattern that makes the op invalid, then work from there

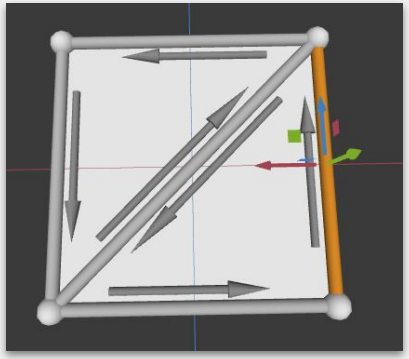
Collapse? **no!**

Flip? **sure**



Collapse? **sure**

Flip? **no!**



- The Half-Edge Data Structure
- Local Mesh Operations
- Debugging Strategies
- Bisect Edge Coding Example

HalfEdge Mesh describe()

```
//a multi-line description of the mesh, suitable for debug output (works on invalid meshes)
std::string describe() const;

//call by putting this anywhere in your mesh operations!
std::cout << describe() << std::endl;
```

HalfEdge Mesh describe()

```
Mesh with 8 halfedges, 4 vertices, 4 edges, and 2 faces:
  [h5] t:h17 n:h7 v0 e6 f4
  [h7] t:h16 n:h9 v1 e8 f4
  [h9] t:h15 n:h11 v3 e10 f4
  [h11] t:h14 n:h5 v2 e12 f4
  [h14] t:h11 n:h15 v0 e12 f13
  [h15] t:h9 n:h16 v2 e10 f13
  [h16] t:h7 n:h17 v3 e8 f13
  [h17] t:h5 n:h14 v1 e6 f13
  [v0] h5 @ {0,0,0}
  [v1] h7 @ {1,0,0}
  [v2] h11 @ {0,1,0}
  [v3] h9 @ {1,1,0}
  [e6] h5
  [e8] h7
  [e10] h9
  [e12] h11
  [f4] h5
  [f13] h14
```

How do we make sense with all this?

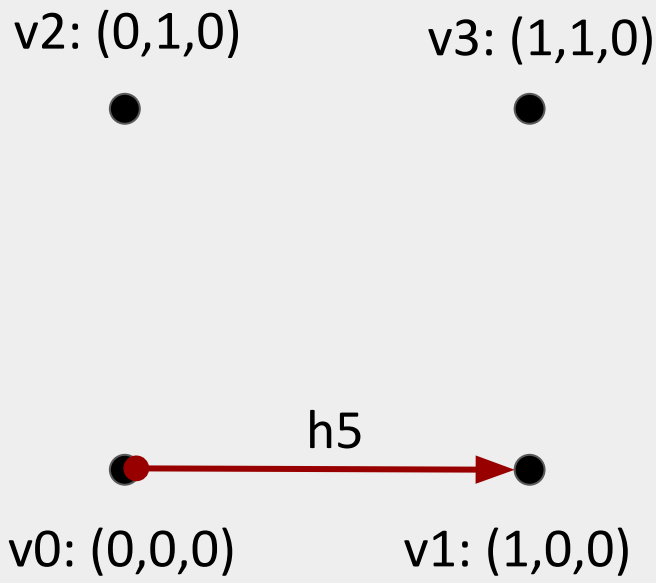
Draw it!

HalfEdge Mesh describe()

```
[h5] t:h17 n:h7 v0 e6 f4
[h7] t:h16 n:h9 v1 e8 f4
[h9] t:h15 n:h11 v3 e10 f4
[h11] t:h14 n:h5 v2 e12 f4
[h14] t:h11 n:h15 v0 e12 f13
[h15] t:h9 n:h16 v2 e10 f13
[h16] t:h7 n:h17 v3 e8 f13
[h17] t:h5 n:h14 v1 e6 f13
[v0] h5 @ {0,0,0}
[v1] h7 @ {1,0,0}
[v2] h11 @ {0,1,0}
[v3] h9 @ {1,1,0}
[e6] h5
[e8] h7
[e10] h9
[e12] h11
[f4] h5
[f13] h14
```

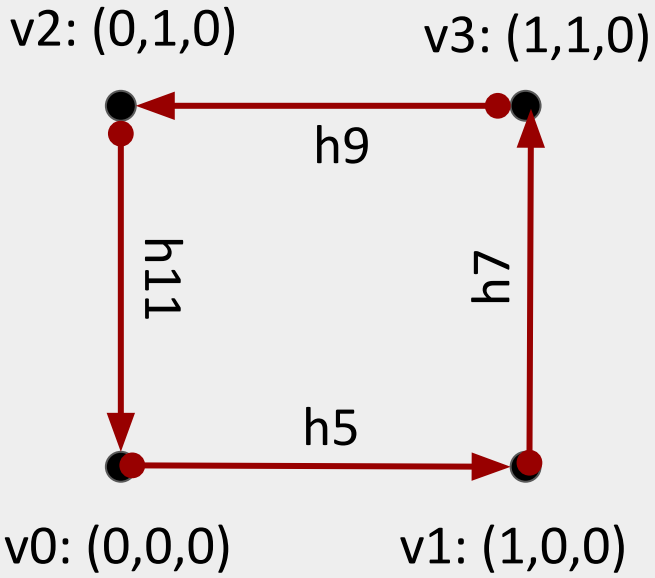
HalfEdge Mesh describe()

```
[h5] t:h17 n:h7 v0 e6 f4
[h7] t:h16 n:h9 v1 e8 f4
[h9] t:h15 n:h11 v3 e10 f4
[h11] t:h14 n:h5 v2 e12 f4
[h14] t:h11 n:h15 v0 e12 f13
[h15] t:h9 n:h16 v2 e10 f13
[h16] t:h7 n:h17 v3 e8 f13
[h17] t:h5 n:h14 v1 e6 f13
[v0] h5 @ {0,0,0}
[v1] h7 @ {1,0,0}
[v2] h11 @ {0,1,0}
[v3] h9 @ {1,1,0}
[e6] h5
[e8] h7
[e10] h9
[e12] h11
[f4] h5
[f13] h14
```



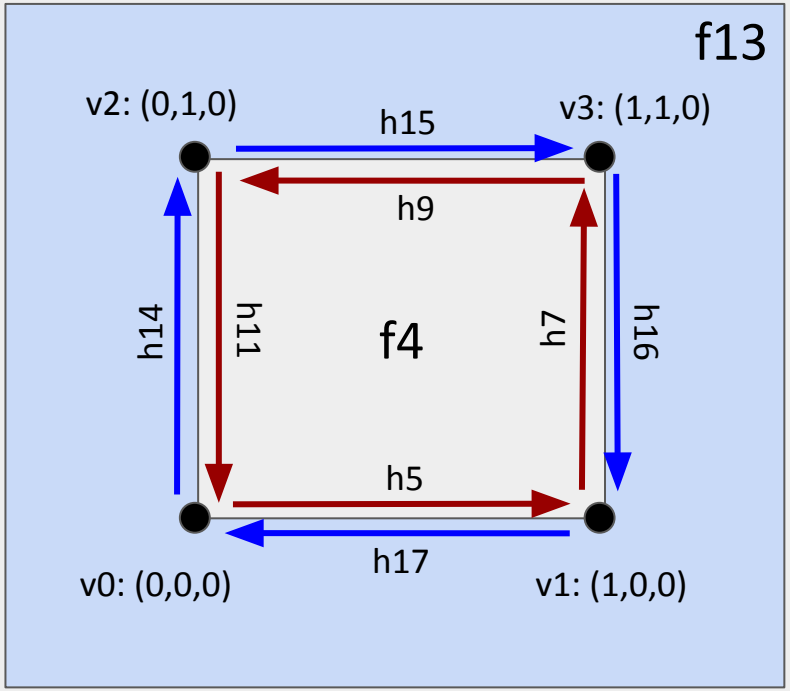
HalfEdge Mesh describe()

```
[h5] t:h17 n:h7 v0 e6 f4
[h7] t:h16 n:h9 v1 e8 f4
[h9] t:h15 n:h11 v3 e10 f4
[h11] t:h14 n:h5 v2 e12 f4
[h14] t:h11 n:h15 v0 e12 f13
[h15] t:h9 n:h16 v2 e10 f13
[h16] t:h7 n:h17 v3 e8 f13
[h17] t:h5 n:h14 v1 e6 f13
[v0] h5 @ {0,0,0}
[v1] h7 @ {1,0,0}
[v2] h11 @ {0,1,0}
[v3] h9 @ {1,1,0}
[e6] h5
[e8] h7
[e10] h9
[e12] h11
[f4] h5
[f13] h14
```



HalfEdge Mesh describe()

```
[h5] t:h17 n:h7 v0 e6 f4
[h7] t:h16 n:h9 v1 e8 f4
[h9] t:h15 n:h11 v3 e10 f4
[h11] t:h14 n:h5 v2 e12 f4
[h14] t:h11 n:h15 v0 e12 f13
[h15] t:h9 n:h16 v2 e10 f13
[h16] t:h7 n:h17 v3 e8 f13
[h17] t:h5 n:h14 v1 e6 f13
[v0] h5 @ {0,0,0}
[v1] h7 @ {1,0,0}
[v2] h11 @ {0,1,0}
[v3] h9 @ {1,1,0}
[e6] h5
[e8] h7
[e10] h9
[e12] h11
[f4] h5
[f13] h14
```



- The Half-Edge Data Structure
- Local Mesh Operations
- Debugging Strategies
- Bisect Edge Coding Demo

Before bisect edge…

FAILED
Invalid mesh: Face with id 3
is referenced by Halfedge with
id 16, which is not in
halfedge(->next)^n.

BEFORE

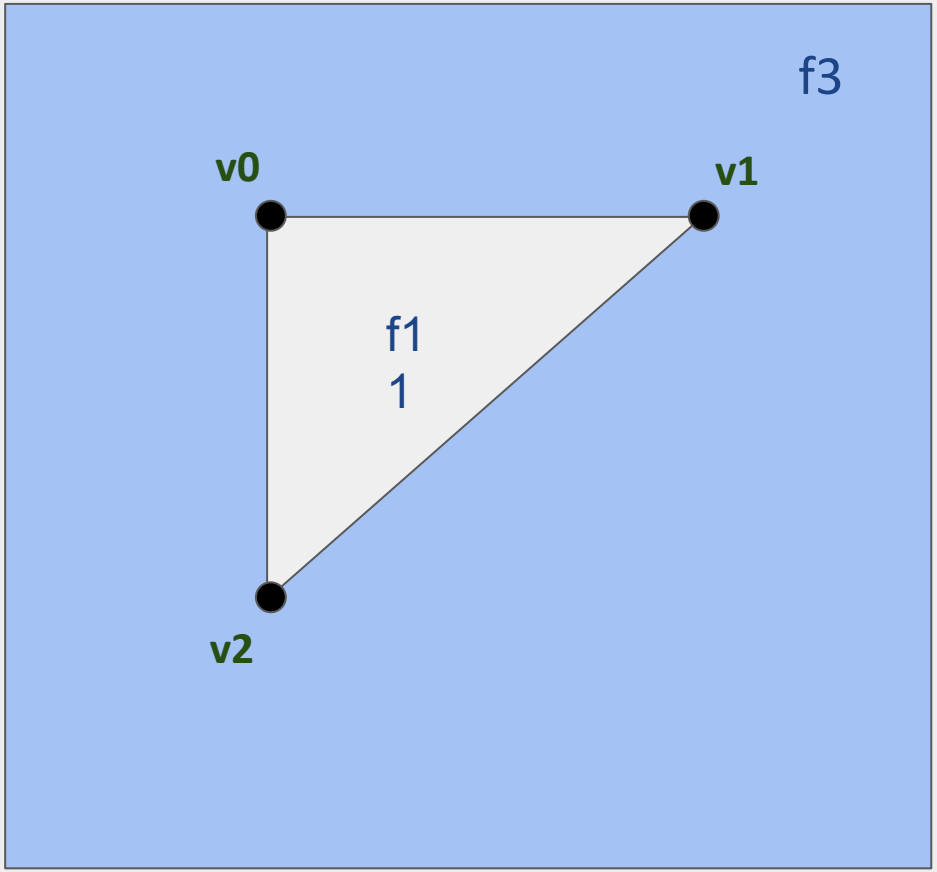
```
[h4] t:h13 n:h6 v0 e5 f3
[h6] t:h12 n:h8 v1 e7 f3
[h8] t:h11 n:h4 v2 e9 f3
[h11] t:h8 n:h12 v0 e9 f10
[h12] t:h6 n:h13 v2 e7 f10
[h13] t:h4 n:h11 v1 e5 f10
[v0] h4 @ {0,0,0}
[v1] h6 @ {1,0,0}
[v2] h8 @ {0,-1,0}
[e5] h4
[e7] h6
[e9] h8
[f3] h4
[f10] h11
```

Before bisect edge…

```
FAILED
Invalid mesh: Face with id 3
is referenced by Halfedge with
id 16, which is not in
halfedge(->next)^n.
```

BEFORE

```
[h4] t:h13 n:h6 v0 e5 f3
[h6] t:h12 n:h8 v1 e7 f3
[h8] t:h11 n:h4 v2 e9 f3
[h11] t:h8 n:h12 v0 e9 f10
[h12] t:h6 n:h13 v2 e7 f10
[h13] t:h4 n:h11 v1 e5 f10
[v0] h4 @ {0,0,0}
[v1] h6 @ {1,0,0}
[v2] h8 @ {0,-1,0}
[e5] h4
[e7] h6
[e9] h8
[f3] h4
[f10] h11
```

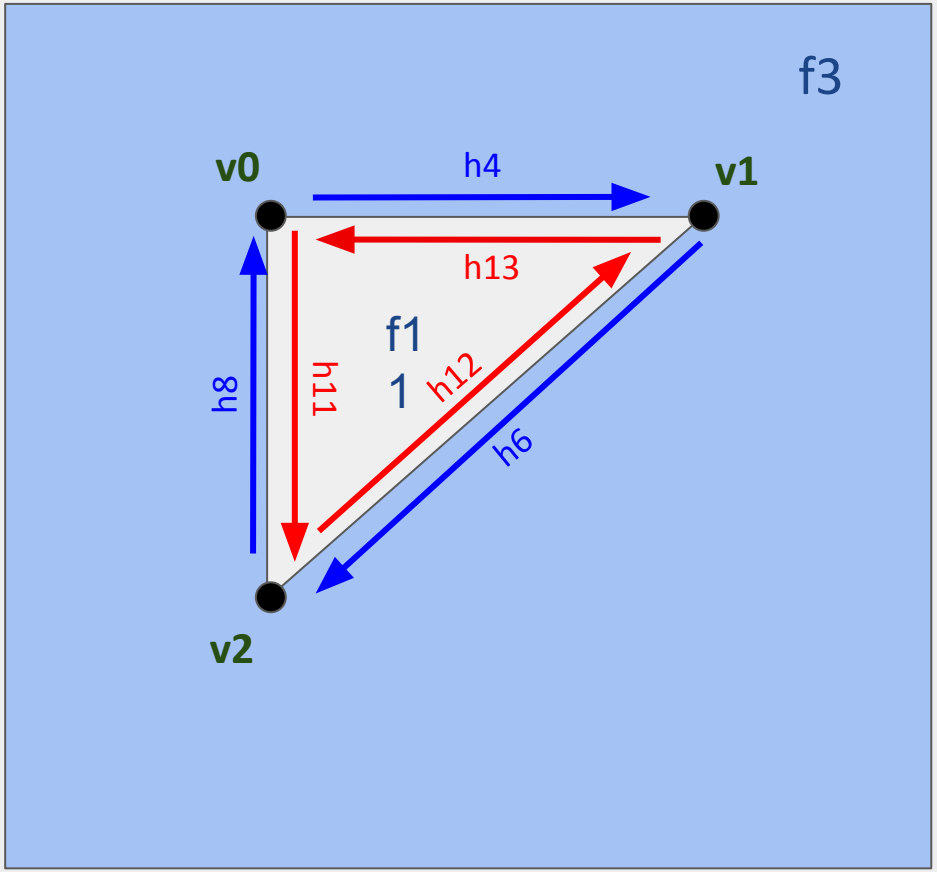


Before bisect edge…

```
FAILED
Invalid mesh: Face with id 3
is referenced by Halfedge with
id 16, which is not in
halfedge(->next)^n.
```

BEFORE

```
[h4] t:h13 n:h6 v0 e5 f3
[h6] t:h12 n:h8 v1 e7 f3
[h8] t:h11 n:h4 v2 e9 f3
[h11] t:h8 n:h12 v0 e9 f10
[h12] t:h6 n:h13 v2 e7 f10
[h13] t:h4 n:h11 v1 e5 f10
[v0] h4 @ {0,0,0}
[v1] h6 @ {1,0,0}
[v2] h8 @ {0,-1,0}
[e5] h4
[e7] h6
[e9] h8
[f3] h4
[f10] h11
```



After bisect edge…

```
FAILED
Invalid mesh: Face with id 3
is referenced by Halfedge with
id 16, which is not in
halfedge(->next)^n.
```

BEFORE

```
[h4] t:h13 n:h6 v0 e5 f3
[h6] t:h12 n:h8 v1 e7 f3
[h8] t:h11 n:h4 v2 e9 f3
[h11] t:h8 n:h12 v0 e9 f10
[h12] t:h6 n:h13 v2 e7 f10
[h13] t:h4 n:h11 v1 e5 f10
[v0] h4 @ {0,0,0}
[v1] h6 @ {1,0,0}
[v2] h8 @ {0,-1,0}
[e5] h4
[e7] h6
[e9] h8
[f3] h4
[f10] h11
```

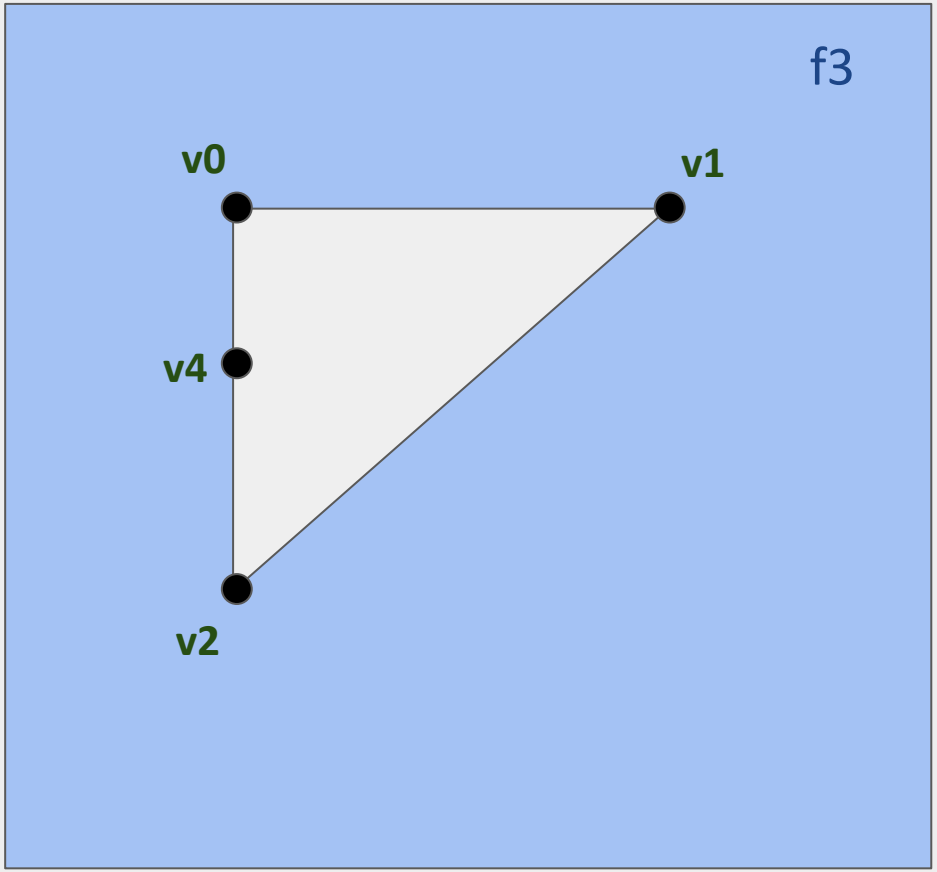
AFTER

```
[h4] t:h13 n:h6 v0 e5 f3
[h6] t:h17 n:h8 v1 e7 f3
[h8] t:h11 n:h4 v2 e9 f3
[h11] t:h8 n:h12 v0 e9 f10
[h12] t:h16 n:h17 v2 e15 f10
[h13] t:h4 n:h11 v1 e5 f10
[h16] t:h11 n:h4 v14 e15 f3
[h17] t:h8 n:h12 v14 e9 f10
[v0] h4 @ {0,0,0}
[v1] h6 @ {1,0,0}
[v2] h8 @ {0,-1,0}
[v14] h16 @ {0,-1,0}
[e5] h4
[e7] h6
[e9] h8
[e15] h16
[f3] h4
[f10] h11
```

After bisect edge…

```
FAILED
Invalid mesh: Face with id 3
is referenced by Halfedge with
id 16, which is not in
halfedge(->next)^n.
```

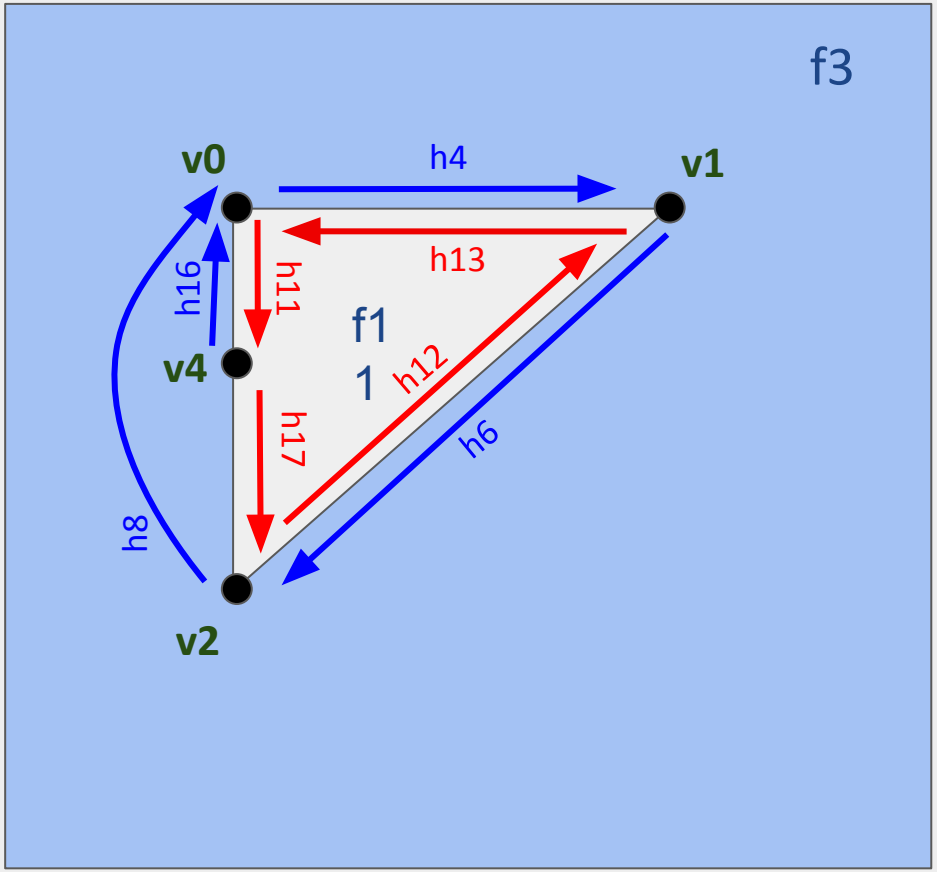
```
AFTER
[h4] t:h13 n:h6 v0 e5 f3
[h6] t:h17 n:h8 v1 e7 f3
[h8] t:h11 n:h4 v2 e9 f3
[h11] t:h16 n:h17 v0 e15 f10
[h12] t:h6 n:h13 v2 e7 f10
[h13] t:h4 n:h11 v1 e5 f10
[h16] t:h11 n:h4 v14 e15 f3
[h17] t:h8 n:h12 v14 e9 f10
[v0] h4 @ {0,0,0}
[v1] h6 @ {1,0,0}
[v2] h8 @ {0,-1,0}
[v14] h16 @ {0,-1,0}
[e5] h4
[e7] h6
[e9] h8
[e15] h16
[f3] h4
[f10] h11
```



After bisect edge…

```
FAILED
Invalid mesh: Face with id 3
is referenced by Halfedge with
id 16, which is not in
halfedge(->next)^n.
```

```
AFTER
[h4] t:h13 n:h6 v0 e5 f3
[h6] t:h17 n:h8 v1 e7 f3
[h8] t:h11 n:h4 v2 e9 f3
[h11] t:h16 n:h17 v0 e15 f10
[h12] t:h6 n:h13 v2 e7 f10
[h13] t:h4 n:h11 v1 e5 f10
[h16] t:h11 n:h4 v14 e15 f3
[h17] t:h8 n:h12 v14 e9 f10
[v0] h4 @ {0,0,0}
[v1] h6 @ {1,0,0}
[v2] h8 @ {0,-1,0}
[v14] h16 @ {0,-1,0}
[e5] h4
[e7] h6
[e9] h8
[e15] h16
[f3] h4
[f10] h11
```

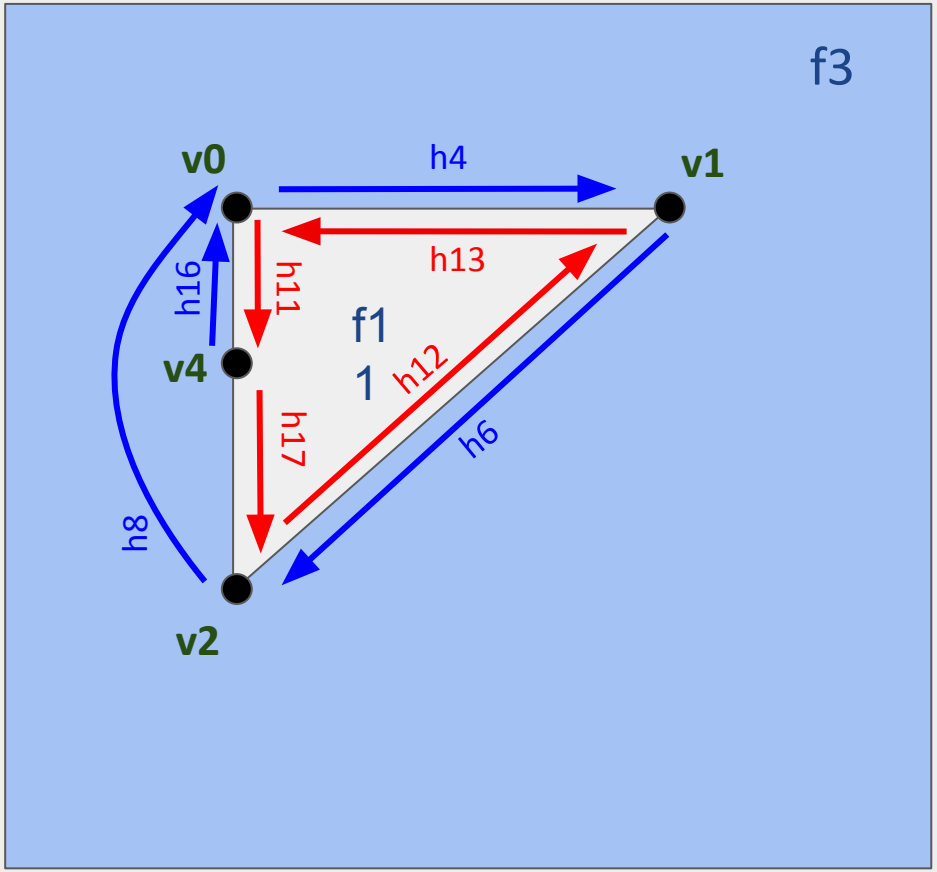


After bisect edge…

We forgot to set h8->next to our new halfedge h16!

AFTER

```
[h4] t:h13 n:h6 v0 e5 f3
[h6] t:h17 n:h8 v1 e7 f3
[h8] t:h11 n:h4 v2 e9 f3
[h11] t:h16 n:h17 v0 e15 f10
[h12] t:h6 n:h13 v2 e7 f10
[h13] t:h4 n:h11 v1 e5 f10
[h16] t:h11 n:h4 v14 e15 f3
[h17] t:h8 n:h12 v14 e9 f10
[v0] h4 @ {0,0,0}
[v1] h6 @ {1,0,0}
[v2] h8 @ {0,-1,0}
[v14] h16 @ {0,-1,0}
[e5] h4
[e7] h6
[e9] h8
[e15] h16
[f3] h4
[f10] h11
```



After bisect edge…

Fixed!

AFTER

```
[h4] t:h13 n:h6 v0 e5 f3
[h6] t:h17 n:h8 v1 e7 f3
[h8] t:h11 n:h16 v2 e9 f3
[h11] t:h16 n:h17 v0 e15 f10
[h12] t:h6 n:h13 v2 e7 f10
[h13] t:h4 n:h11 v1 e5 f10
[h16] t:h11 n:h4 v14 e15 f3
[h17] t:h8 n:h12 v14 e9 f10
[v0] h4 @ {0,0,0}
[v1] h6 @ {1,0,0}
[v2] h8 @ {0,-1,0}
[v14] h16 @ {0,-1,0}
[e5] h4
[e7] h6
[e9] h8
[e15] h16
[f3] h4
[f10] h11
```