Computational Geometry

Graham Rhodes
Senior Software Developer, Applied Research Associates, Inc.

GAME DEVELOPERS CONFERENCE
SAN FRANCISCO, CA
MARCH 5-9, 2012
EXPO DATES: MARCH 7-9
2012

What is Computational Geometry?

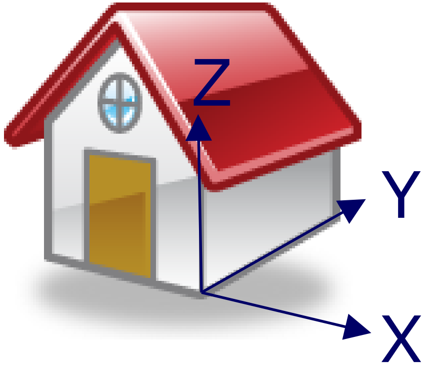
- Manipulation and interrogation of shapes
- Examples:
  - “What is the intersection of a line and a triangle mesh”
  - “What is the minimum distance separating two objects”
  - “Break a mesh into pieces”

Typical Application in Games

- Interrogation of Geometry
  - Collision detection for physics
  - Proximity triggers for game logic
  - Pathfinding, visibility, and other AI operations
- Manipulation of geometry
  - Creation of game assets in 3ds max/Maya/etc
  - User-generated content (e.g., Little Big Planet)
  - Destruction (Bad Company 2, etc., etc.)

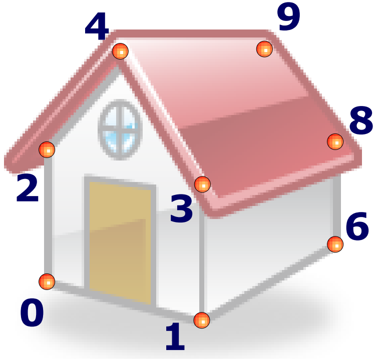
Some Perspective

- Game levels are usually made of meshes
  - Typically made of triangles
  - Indexed triangle meshes



Some Perspective

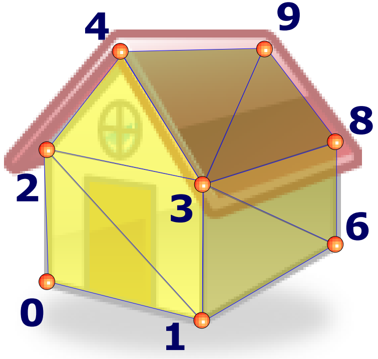
- Game levels are usually made of meshes
  - Typically made of triangles
  - Indexed triangle meshes



0	<-.5, 0, 0>
1	<0, 0, 0>
2	<-.5, 0, .5>
3	<0, 0, .5>
4	<-.25, 0, 1>
5	<-.5, 1, 0>
6	<0, 1, 0>
7	<-.5, 1, .5>
8	<0, 1, .5>
9	<-.25, 1, 1>

Some Perspective

- Game levels are usually made of meshes
  - Typically made of triangles
  - Indexed triangle meshes

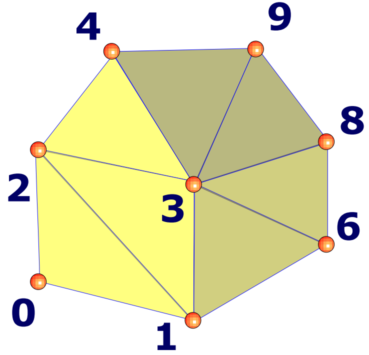


0	<-.5, 0, 0>
1	<0, 0, 0>
2	<-.5, 0, .5>
3	<0, 0, .5>
4	<-.25, 0, 1>
5	<-.5, 1, 0>
6	<0, 1, 0>
7	<-.5, 1, .5>
8	<0, 1, .5>
9	<-.25, 1, 1>

Triangle Indices
0, 1, 2
1, 3, 2
2, 3, 4
1, 6, 3
3, 6, 8
3, 8, 9
3, 9, 4
…

Some Perspective

- Game levels are usually made of meshes
  - Typically made of triangles
  - Indexed triangle meshes



0	<-.5, 0, 0>
1	<0, 0, 0>
2	<-.5, 0, .5>
3	<0, 0, .5>
4	<-.25, 0, 1>
5	<-.5, 1, 0>
6	<0, 1, 0>
7	<-.5, 1, .5>
8	<0, 1, .5>
9	<-.25, 1, 1>

Triangle Indices
0, 1, 2
1, 3, 2
2, 3, 4
1, 6, 3
3, 6, 8
3, 8, 9
3, 9, 4
…

Getting Ready

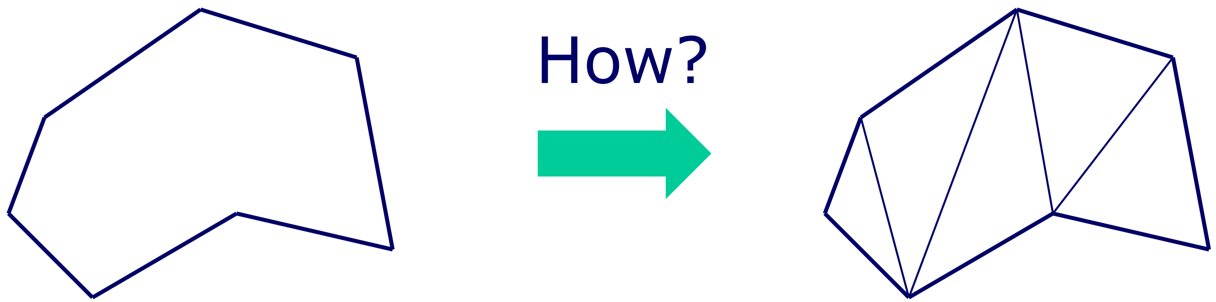
Computational geometry requires appropriate data structures

Categories of Data Structures

- Spatial
  - Find things fast
  - BSP tree, octree, Kd-tree, spatial hashing
  - Etc…
- **Geometry + topology**
  - Change the shape of objects
  - Focus of this talk!

A Computational Geometry Problem

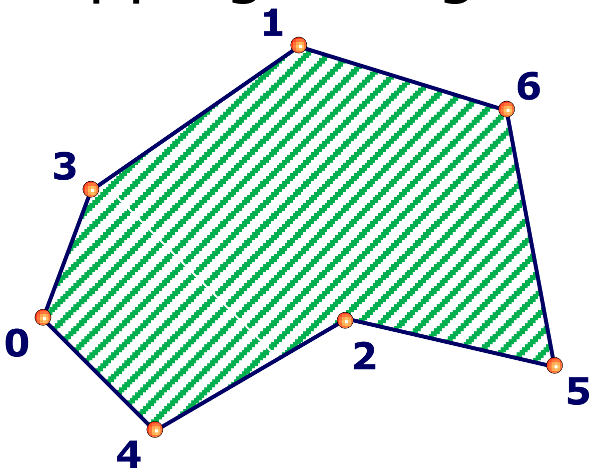
- We have polygons in our game level
- The graphics card requires triangles
  - Triangulation converts polygons into triangles



The Polygon Triangulation Problem

- Intuitive approach: ear-clipping
- Fast approach: monotone decomposition
- Both involve chopping triangles off in a sequence

Number of Polygons:
1

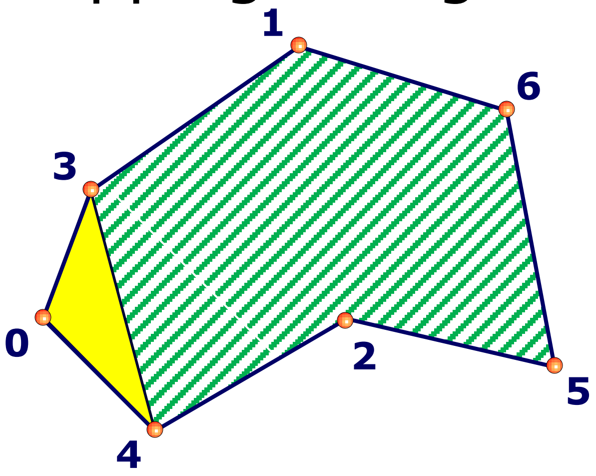


Polygon Indices
6,1,3,0,4,2,5

The Polygon Triangulation Problem

- Intuitive approach: ear-clipping
- Fast approach: monotone decomposition
- Both involve chopping triangles off in a sequence

Number of Polygons:
2

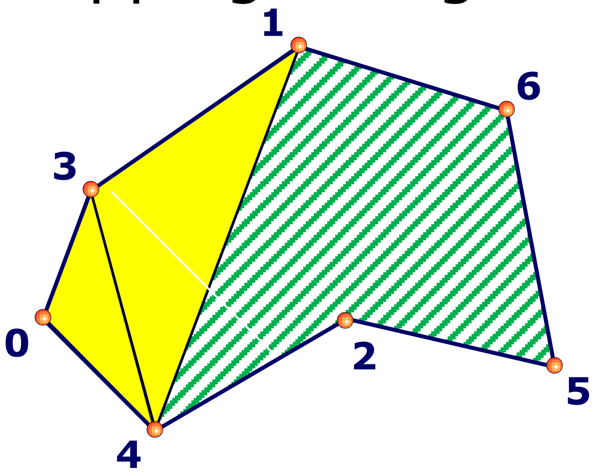


Polygon Indices
6,1,3,4,2,5
0,4,3

The Polygon Triangulation Problem

- Intuitive approach: ear-clipping
- Fast approach: monotone decomposition
- Both involve chopping triangles off in a sequence

Number of Polygons:
3

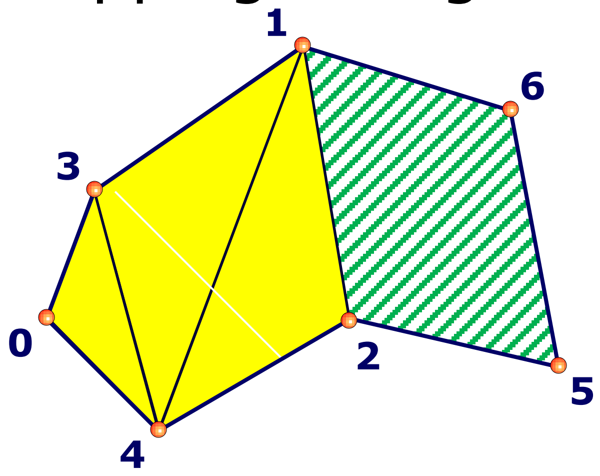


Polygon Indices
6,1,4,2,5
0,4,3
4,1,3

The Polygon Triangulation Problem

- Intuitive approach: ear-clipping
- Fast approach: monotone decomposition
- Both involve chopping triangles off in a sequence

Number of Polygons: 4

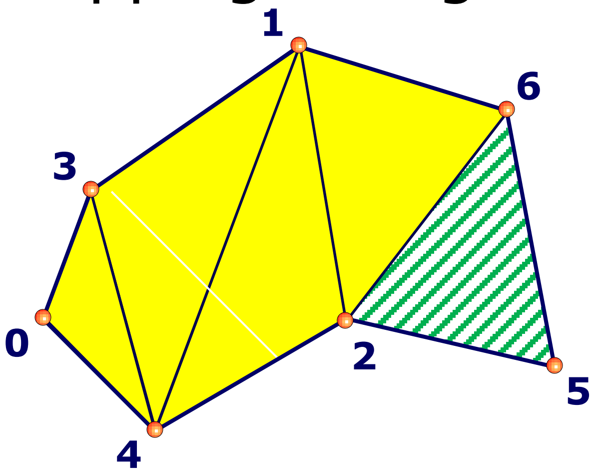


Polygon Indices
6,1,2,5
0,4,3
4,1,3
4,2,1

The Polygon Triangulation Problem

- Intuitive approach: ear-clipping
- Fast approach: monotone decomposition
- Both involve chopping triangles off in a sequence

Number of Polygons:
5

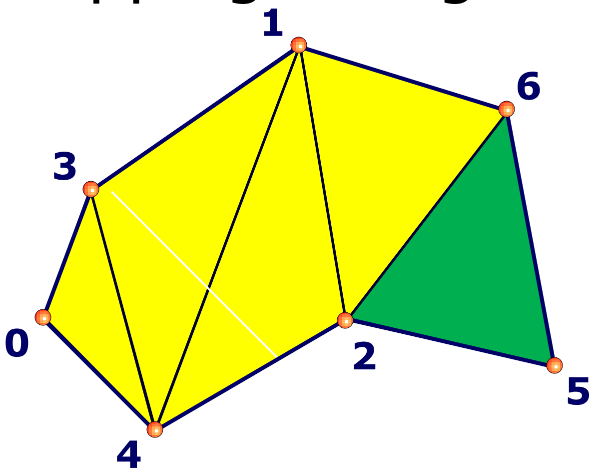


Polygon Indices
6,2,5
0,4,3
4,1,3
4,2,1
2,6,1

The Polygon Triangulation Problem

- Intuitive approach: ear-clipping
- Fast approach: monotone decomposition
- Both involve chopping triangles off in a sequence

Number of Polygons:
5



Polygon Indices
6,2,5
0,4,3
4,1,3
4,2,1
2,6,1

The Polygon Triangulation Problem

- Was maintaining the index list convenient?
- NO!
  - Original polygon: 6,1,3,0,4,2,5
  - Final polygon: 2,5,6
  - All the removed points were in the middle of the list!
  - Maintaining the list can be error prone, and slow for complex models
  - Inelegant

Getting Ready

Computational geometry requires appropriate data structures!

(Lets take a look at one)

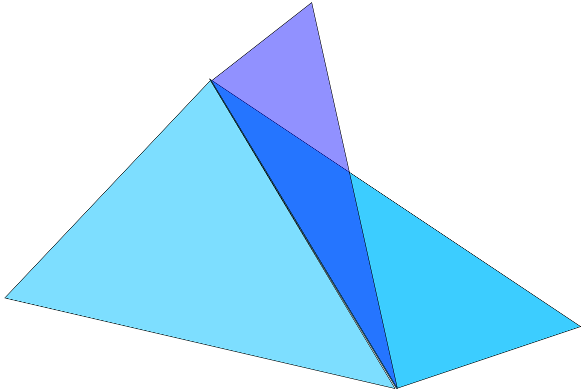
Triangulation Demo Part 1

Geometric Model Representation

- Geometry describes the shape of model elements (triangles)
- Topology describes how the elements are connected

Manifold Topology

- Each edge joins exactly two faces
  - Model is watertight
- Open edges that join to one face are allowed
- Modeling operation consistency rules
  - "Invariants"



Non-manifold Topology

Topological Data Structures

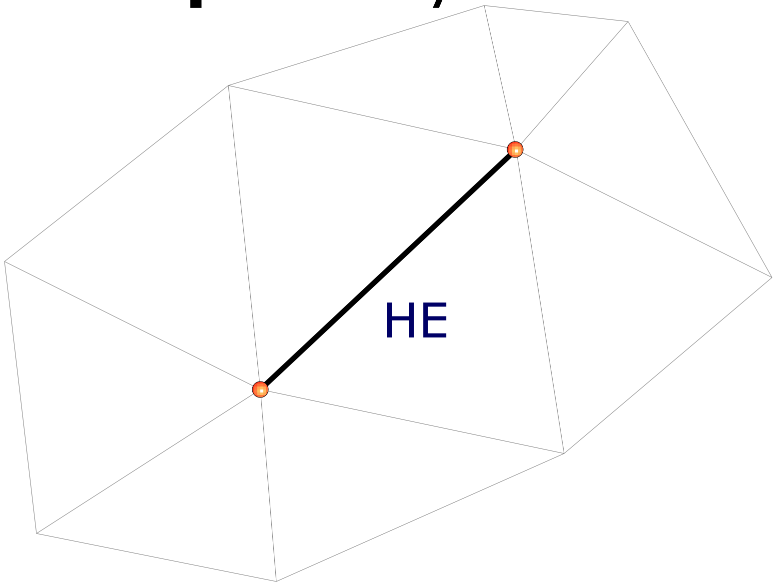
- Enable elegant and fast traversals
  - "Which edges surround a polygon?"
  - "Which polygons surround this vertex?"
- Easy to modify geometry
  - Split an edge or face to add a new vertex
  - Collapse an edge to simplify a mesh

Other Topological Data Structures

- Manifold
  - Winged Edge (Baumgart, 1972)
  - Half Edge (presented here)
  - Quad edge
- Non-manifold
  - Radial edge
  - Generalized non-manifold

Half Edge Data Structure (HDS)

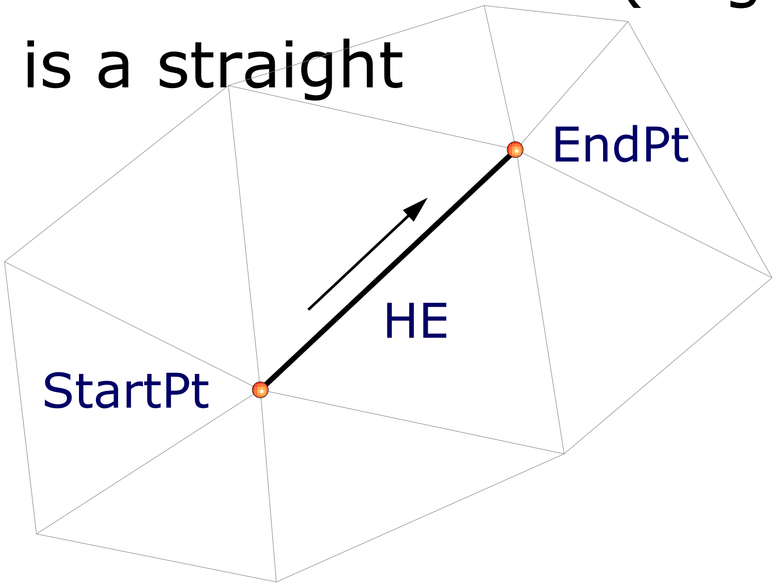
- Basic **topological** element is a half edge (HE)
- Geometry is **implied** by connections*



Half Edge Properties

Half Edge Data Structure (HDS)

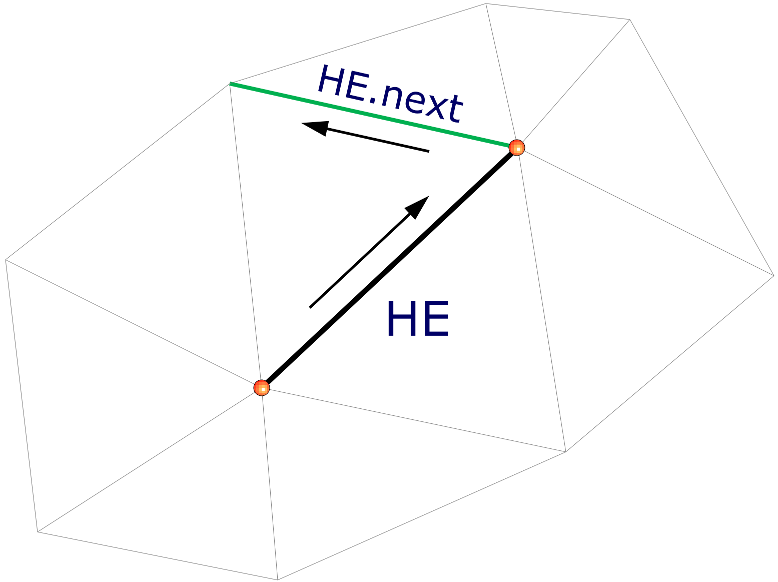
- HE **connects** a Start point to an End point
- Traversal is StartPt to EndPt (edge is oriented)
- Geometry is a straight



Half Edge Properties
EndPt

Half Edge Data Structure (HDS)

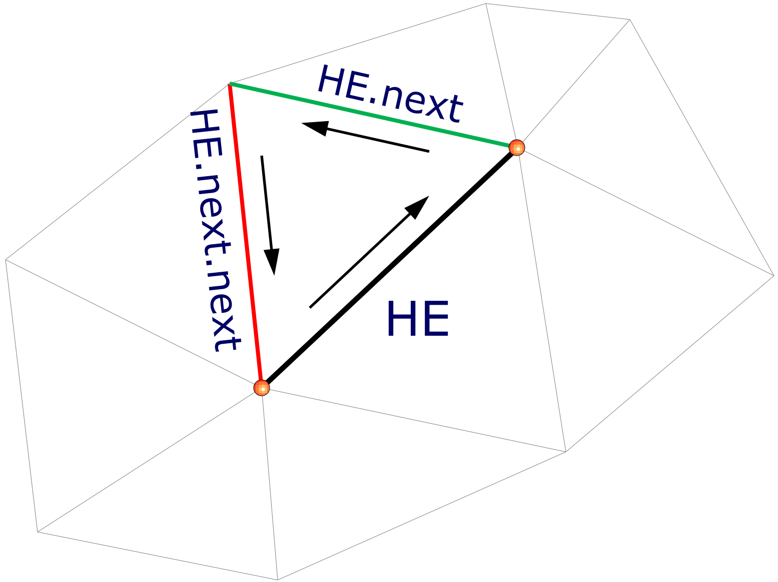
- HE points to next half edge in traversal direction
- Start point of HE.next is HE.EndPt



Half Edge Properties
EndPt
Next

Half Edge Data Structure (HEDS)

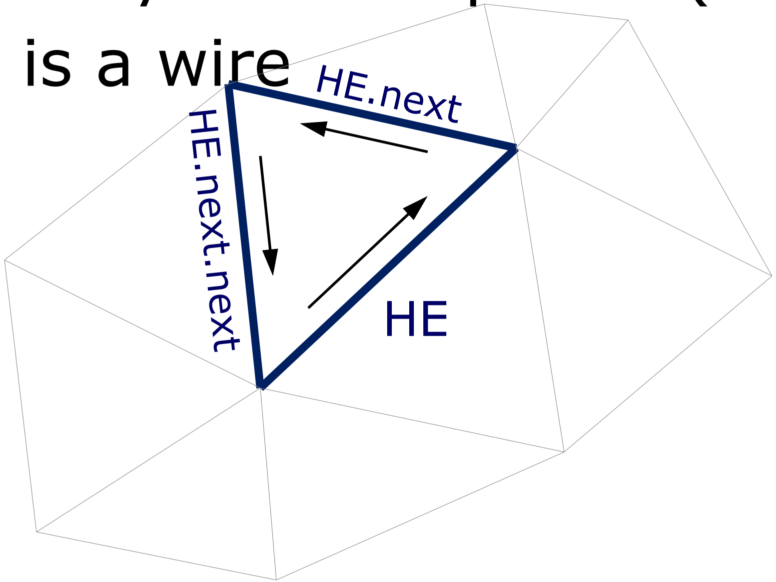
- Traversal directions are consistent



Half Edge Properties
EndPt
Next

Half Edge Data Structure (HEDS)

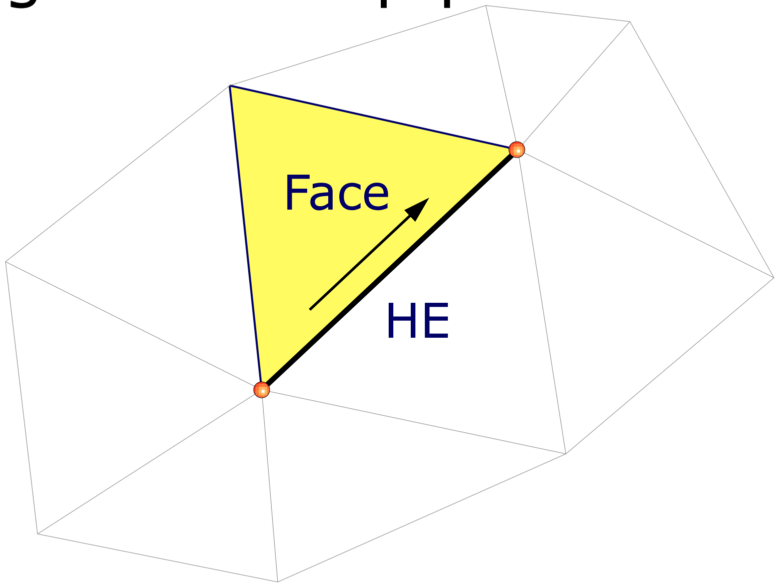
- Note that sequence of half edges forms a loop!
- So far, we only connect points (no polygons yet!)
- Geometry is a wire



Half Edge Properties
EndPt
Next

Half Edge Data Structure (HEDS)

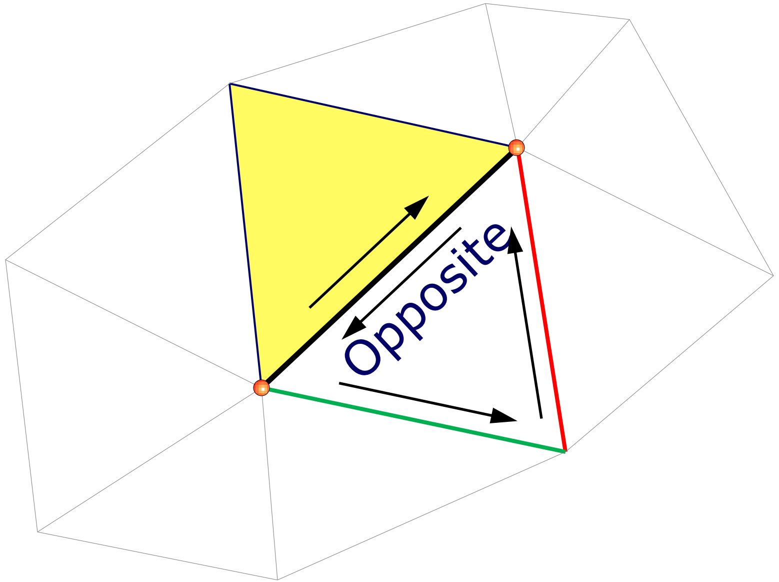
- HE ***may*** point to a face on its **left** side
- All half edges in a loop point to same face



Half Edge Properties
EndPt
Next
Face

Half Edge Data Structure (HEDS)

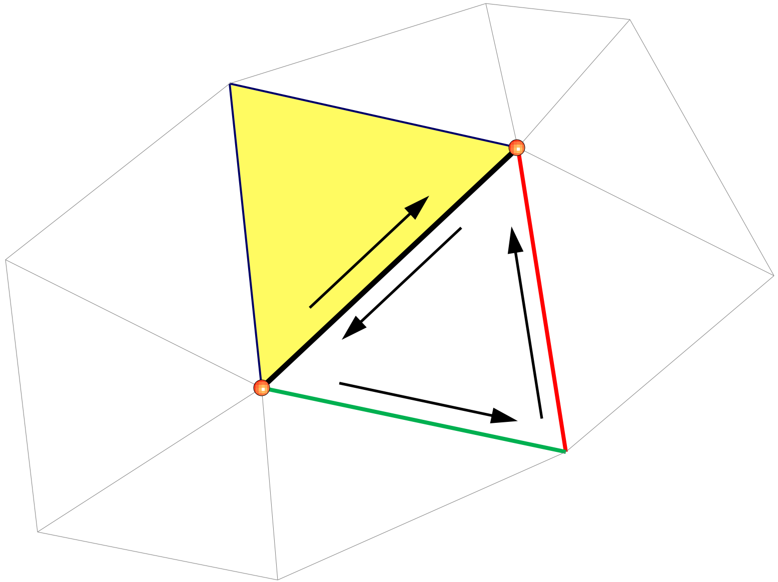
- HE points to its opposite half edge
- Which is attributed as above



Half Edge Properties
EndPt
Next
Face
Opposite

Half Edge Data Structure (HEDS)

- It is useful to store user data and a marker



Half Edge Properties
EndPt
Next
Face
Opposite
UserData
Marker

Simple C++ HDS class definition

```
struct HalfEdge
{
  HalfEdgeVert *endPt;
  HalfEdge *next;
  HalfEdge *opposite;
  HalfEdgeFace *face;
  void *userData;
  unsigned char marker;
};
```

```
struct HalfEdgeFace
{
  HalfEdge *halfEdge;
  unsigned char marker;
};
struct HalfEdgeVert
{
  HalfEdge *halfEdge;
  int index;
  unsigned char marker;
};
```

HDS Invariants

- Strict
  - halfEdge != halfEdge->opposite
  - halfEdge != halfEdge->next
  - halfEdge == halfEdge->opposite->opposite
  - startPt(halfEdge) == halfEdge->opposite->endPt
  - There are a few others…
- Convenience
  - Vertex == Vertex->halfEdge->endPt

Simple Traversals

Find vertex loop defined by a half edge

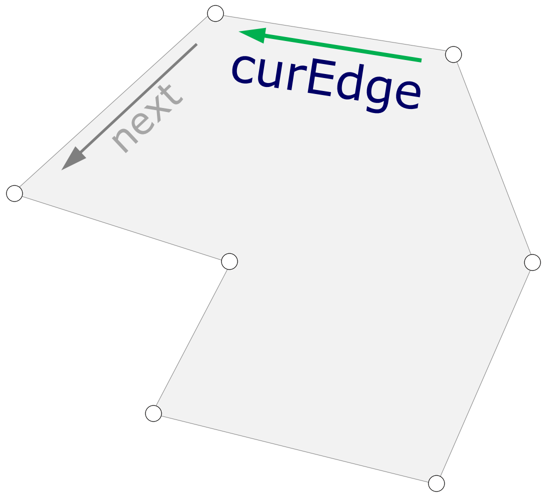
```
IndexList FindVertexLoop(HalfEdge *edge)
{
  IndexList loop;
  HalfEdge *curEdge = edge;
  do {
    loop.push_back(edge.endPt->index);
    curEdge = curEdge->next;
  } while (curEdge != edge);
  return loop;
};
```

Triangulation Demo
Part 2

Simple Traversals

Find vertex loop defined by a half edge

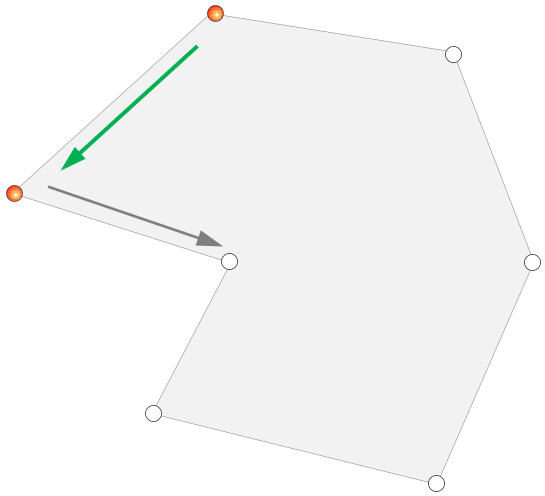
```
IndexList FindVertexLoop(HalfEdge *edge)
{
  IndexList loop;
  HalfEdge *curEdge = edge;
  do {
    loop.push_back(edge.endPt->index);
    curEdge = curEdge->next;
  } while (curEdge != edge);
  return loop;
};
```



Simple Traversals

Find vertex loop defined by a half edge

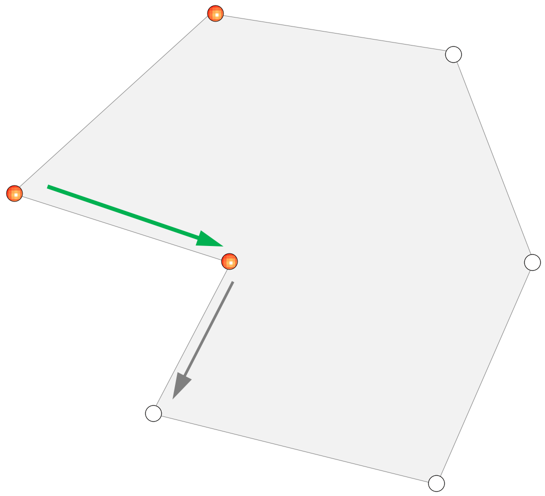
```
IndexList FindVertexLoop(HalfEdge *edge)
{
  IndexList loop;
  HalfEdge *curEdge = edge;
  do {
    loop.push_back(edge.endPt->index);
    curEdge = curEdge->next;
  } while (curEdge != edge);
  return loop;
};
```



Simple Traversals

Find vertex loop defined by a half edge

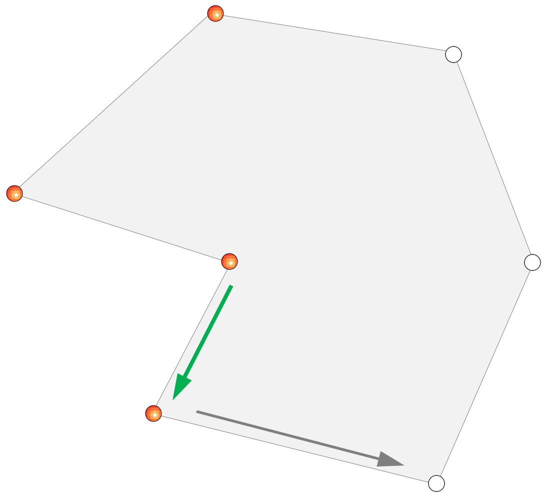
```
IndexList FindVertexLoop(HalfEdge *edge)
{
  IndexList loop;
  HalfEdge *curEdge = edge;
  do {
    loop.push_back(edge.endPt->index);
    curEdge = curEdge->next;
  } while (curEdge != edge);
  return loop;
};
```



Simple Traversals

Find vertex loop defined by a half edge

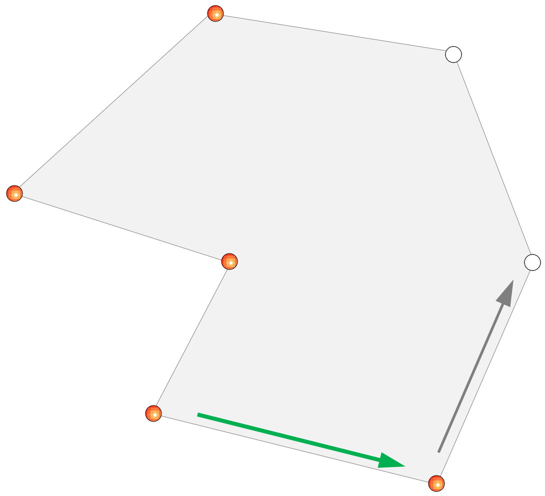
```
IndexList FindVertexLoop(HalfEdge *edge)
{
  IndexList loop;
  HalfEdge *curEdge = edge;
  do {
    loop.push_back(edge.endPt->index);
    curEdge = curEdge->next;
  } while (curEdge != edge);
  return loop;
};
```



Simple Traversals

Find vertex loop defined by a half edge

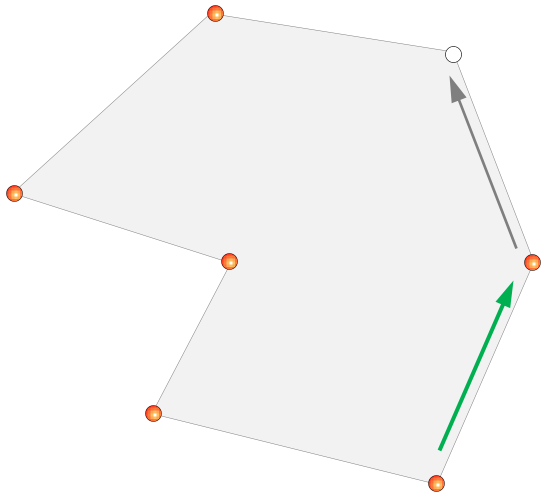
```
IndexList FindVertexLoop(HalfEdge *edge)
{
  IndexList loop;
  HalfEdge *curEdge = edge;
  do {
    loop.push_back(edge.endPt->index);
    curEdge = curEdge->next;
  } while (curEdge != edge);
  return loop;
};
```



Simple Traversals

Find vertex loop defined by a half edge

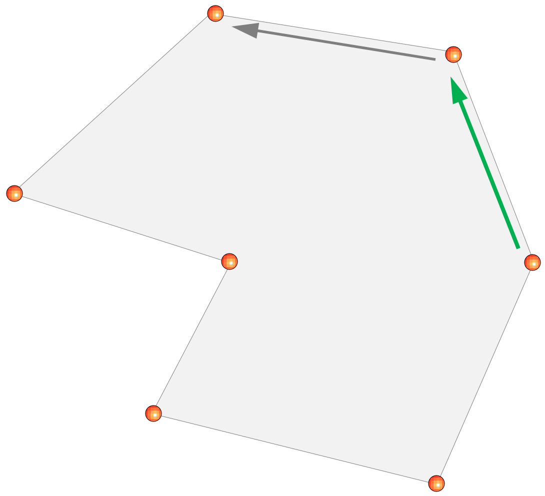
```
IndexList FindVertexLoop(HalfEdge *edge)
{
  IndexList loop;
  HalfEdge *curEdge = edge;
  do {
    loop.push_back(edge.endPt->index);
    curEdge = curEdge->next;
  } while (curEdge != edge);
  return loop;
};
```



Simple Traversals

Find vertex loop defined by a half edge

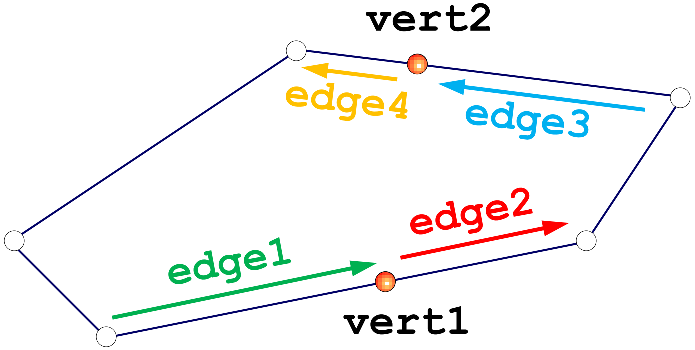
```
IndexList FindVertexLoop(HalfEdge *edge)
{
  IndexList loop;
  HalfEdge *curEdge = edge;
  do {
    loop.push_back(edge.endPt->index);
    curEdge = curEdge->next;
  } while (curEdge != edge);
  return loop;
};
```



Simple Operations

Split a face

```
HalfEdge edge1 = vert1.halfEdge;
HalfEdge edge2 = edge1.next;
HalfEdge edge3 = vert2.halfEdge;
HalfEdge edge4 = edge3.next;
```



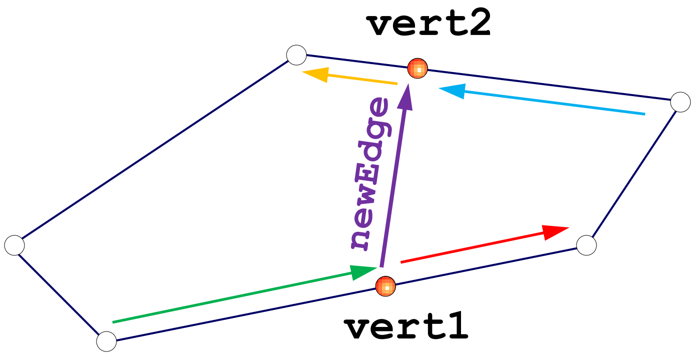
*See speaker notes below slide for an important consideration!

Simple Operations

Split a face

```
HalfEdge edge1 = vert1.halfEdge;
HalfEdge edge2 = edge1.next;
HalfEdge edge3 = vert2.halfEdge;
HalfEdge edge4 = edge3.next;
HalfEdge newEdge = new HalfEdge;

edge1.next = newEdge;
newEdge.next = edge4;
newEdge.face = edge1.face;
newEdge.endPt = vert2;
```

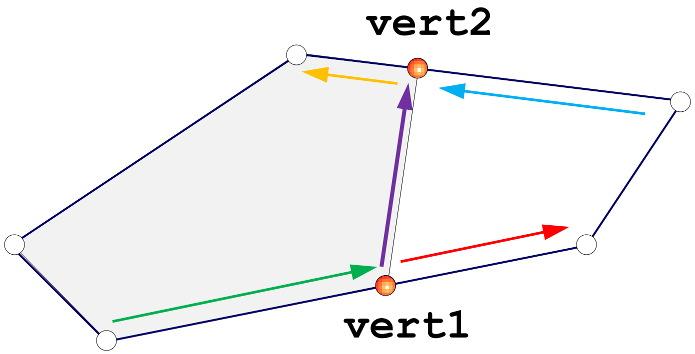


Simple Operations

Split a face

```
HalfEdge edge1 = vert1.halfEdge;
HalfEdge edge2 = edge1.next;
HalfEdge edge3 = vert2.halfEdge;
HalfEdge edge4 = edge3.next;
HalfEdge newEdge = new HalfEdge;

edge1.next = newEdge;
newEdge.next = edge4;
newEdge.face = edge1.face;
newEdge.endPt = vert2;
edge1.face.halfEdge = edge1;
```



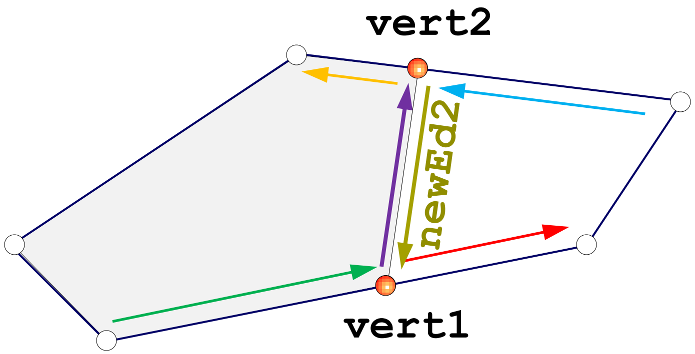
Simple Operations

Split a face

```
HalfEdge newEd2 = new HalfEdge;

newEd2.next = edge2;
newEd2.endPt = vert1;
edge3.next = newEd2;

newEdge.opposite = newEd2;
newEd2.opposite = newEdge;
```

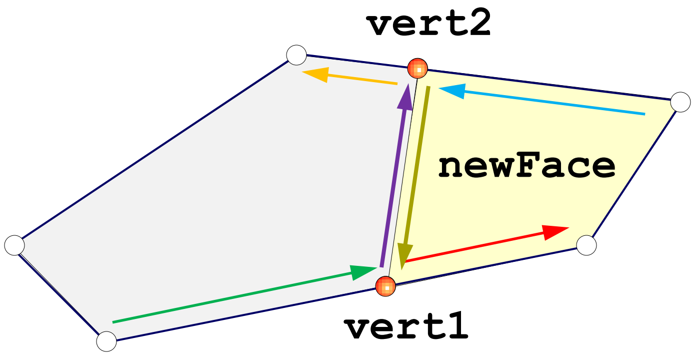


Simple Operations

Split a face

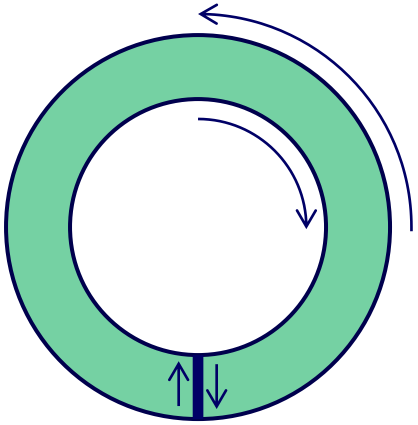
```
newFace = new HalfEdgeFace
newFace.halfEdge = edge2;

HalfEdge *curEdge = edge2;
do {
  curEdge->face = newFace;
  curEdge = curEdge->next;
} while (curEdge != edge2);
```



Triangulation Demo
Part 3

Related Operations



- Cut off an ear/triangle
  - Exactly same as split face
  - Apply recursively to triangulate face
- Insert auxiliary edge
  - Connect inner and outer loops to support holes in faces

What Else Can We Do?

- Split a mesh in half with a cutting plane
  - Step 1: Split edges that cross the plane
  - Step 2: Split faces that share two split edges
  - …

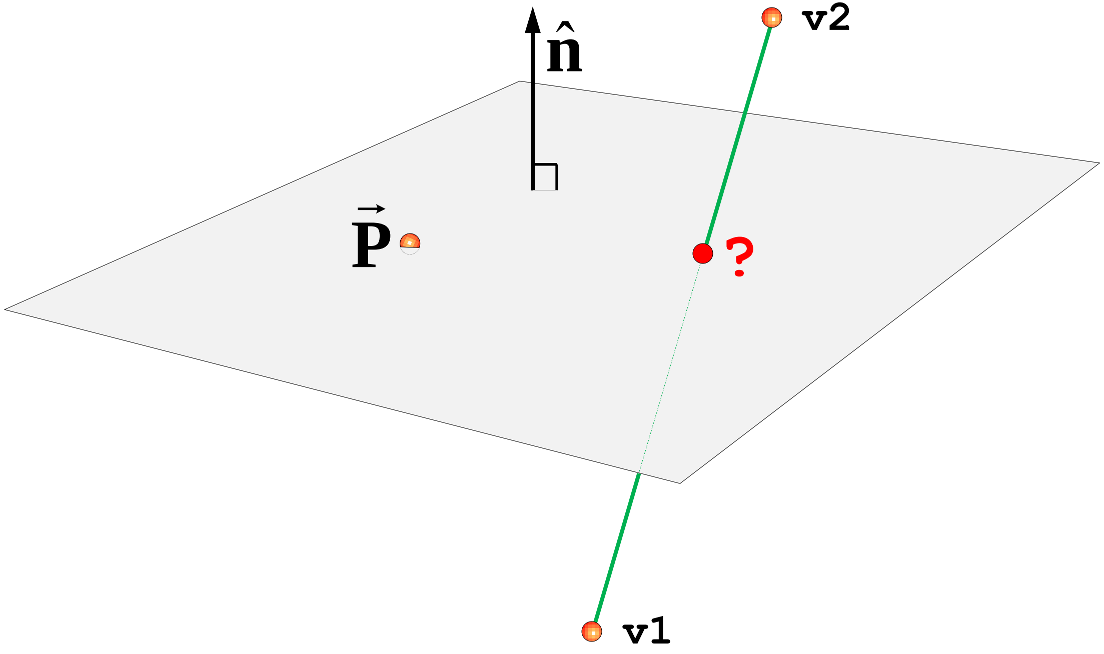
Intersection of edge and plane

- Plane equation

$$\hat{\mathbf{n}} \cdot \vec{\mathbf{P}} = d$$

- Line Equation

$$\vec{\mathbf{P}}_{line} = \vec{v}_1 + t\left(\vec{v}_2 - \vec{v}_1\right)$$



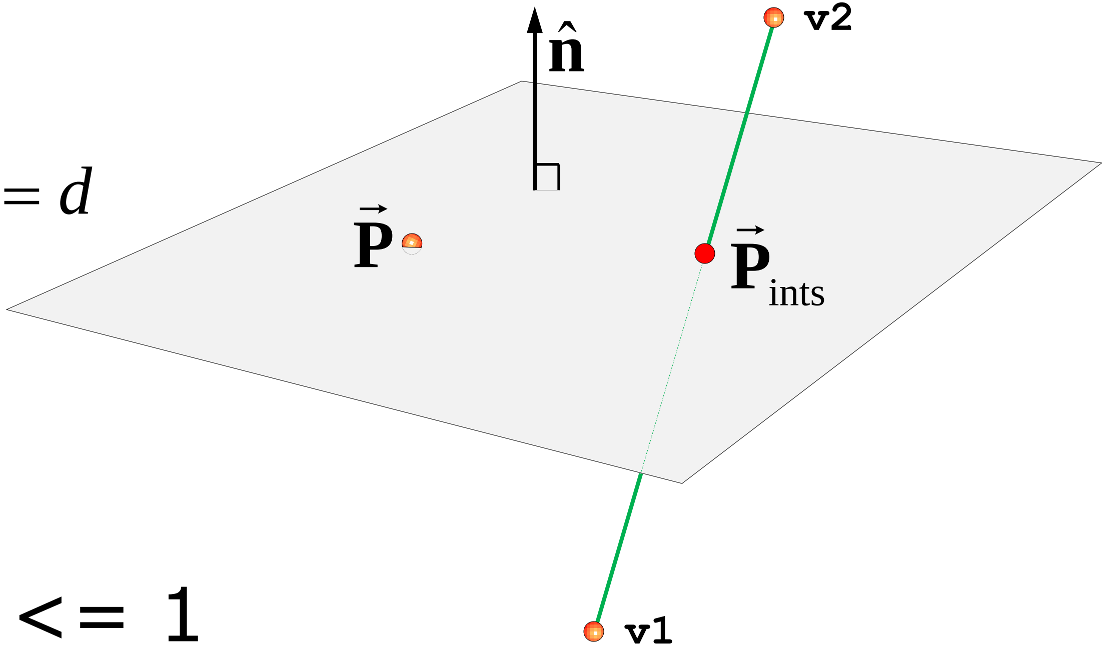
Intersection of edge and plane

- Solve for t

$$\hat{\mathbf{n}} \cdot \vec{v}_1 + t_{\text{ints}} \vec{\mathbf{n}} \cdot \left(\vec{v}_2 - \vec{v}_1 \right) = d$$

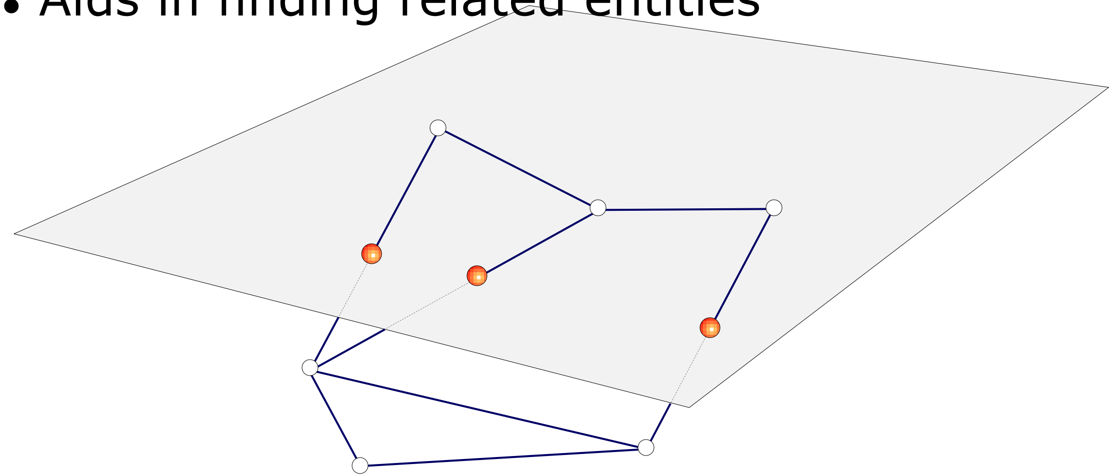
$$t_{\text{ints}} = \frac{\left(d - \hat{\mathbf{n}} \cdot \vec{v}_1 \right)}{\vec{\mathbf{n}} \cdot \left(\vec{v}_2 - \vec{v}_1 \right)}$$

- If t >= 0 and t <= 1
  - Edge touches plane



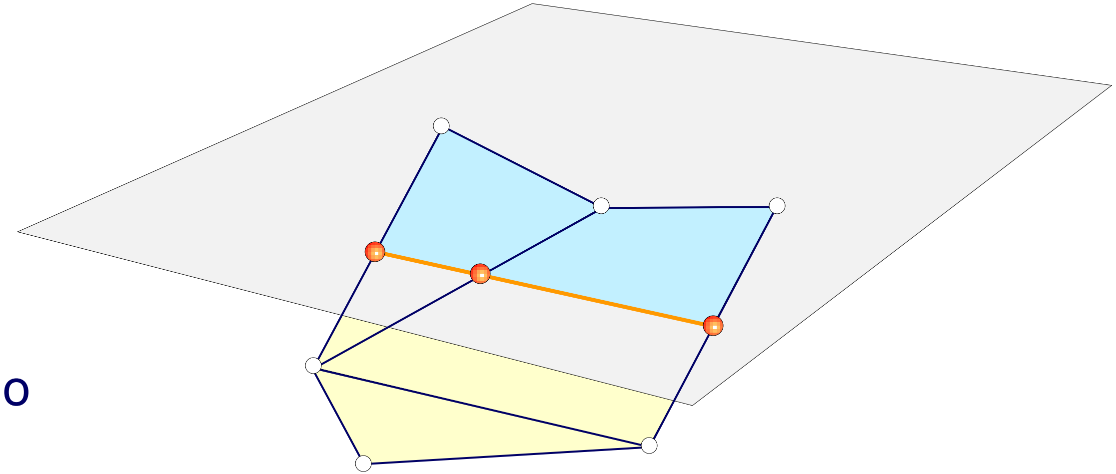
What Can We Do with a B-rep Mesh?

- Split a mesh in half with a cutting plane
  - Step 1: Split edges that cross the plane
    - Use marker variables to tag affected geometry
      - Aids in finding related entities



What Can We Do with a B-rep Mesh?

- Split a mesh in half with a cutting plane
  - Step 2: Split faces that share two split edges

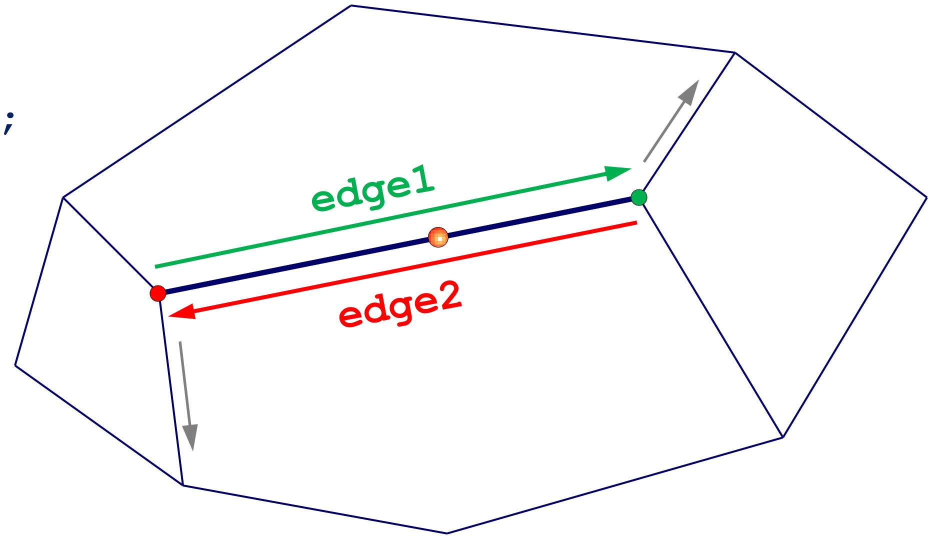


Edge Split Demo
Part 1

Simple Operations

Split an edge

```
HalfEdge edge1;
HalfEdge edge2 = edge1.opposite;
```

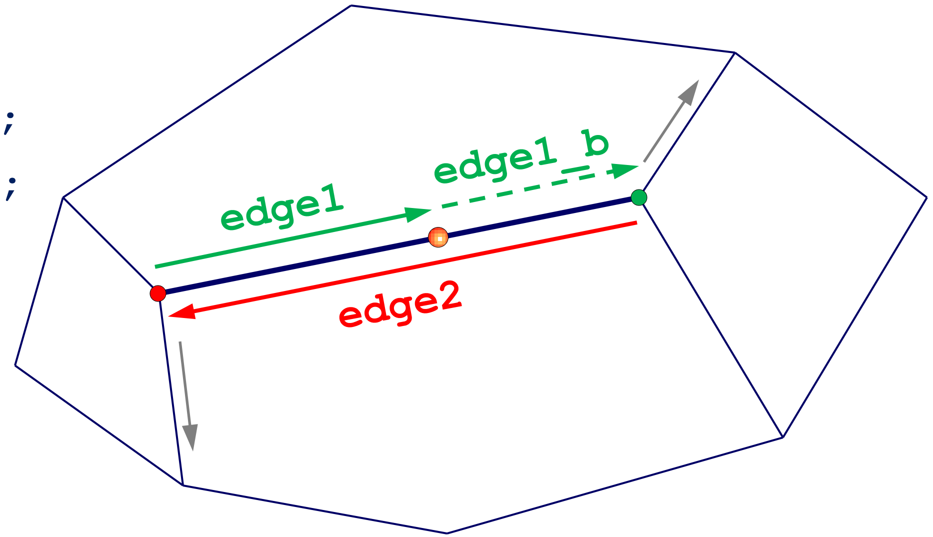


Simple Operations

Split an edge

```
HalfEdge edge1;
HalfEdge edge2 = edge1.opposite;
HalfEdge edge1_b = new HalfEdge;

edge1_b.EndPt = edge1.EndPt;
edge1_b.face = edge1.face;
edge1_b.next = edge1.next;
edge1.EndPt = splitVert;
edge1.next = edge1_b;
edge1_b.EndPt.halfEdge = edge1_b;
```

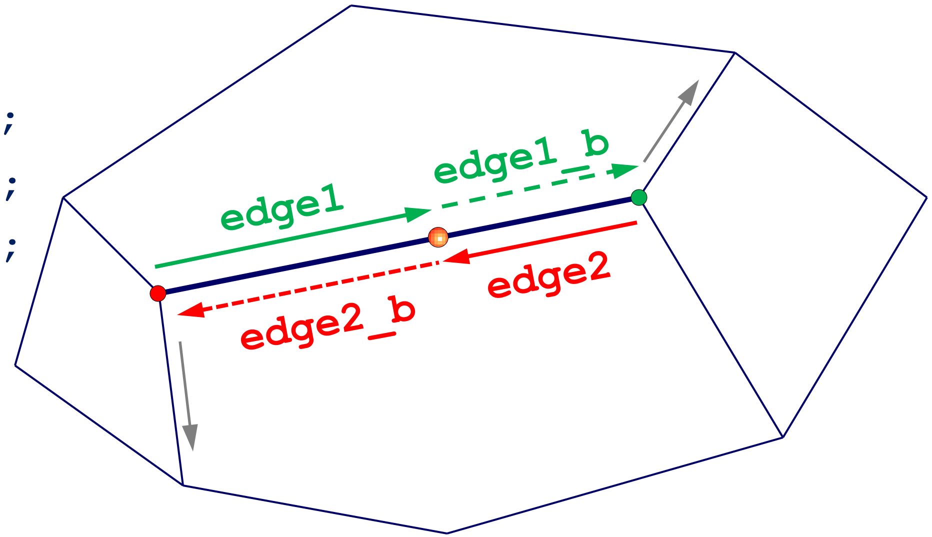


Simple Operations

Split an edge

```
HalfEdge edge1;
HalfEdge edge2 = edge1.opposite;
HalfEdge edge1_b = new HalfEdge;
HalfEdge edge2_b = new HalfEdge;

edge2_b.EndPt = edge2.EndPt;
edge2_b.face = edge2.face;
edge2_b.next = edge2.next;
edge2.EndPt = splitVert;
edge2.next = edge2_b;
edge2_b.EndPt.halfEdge = edge2_b;
```

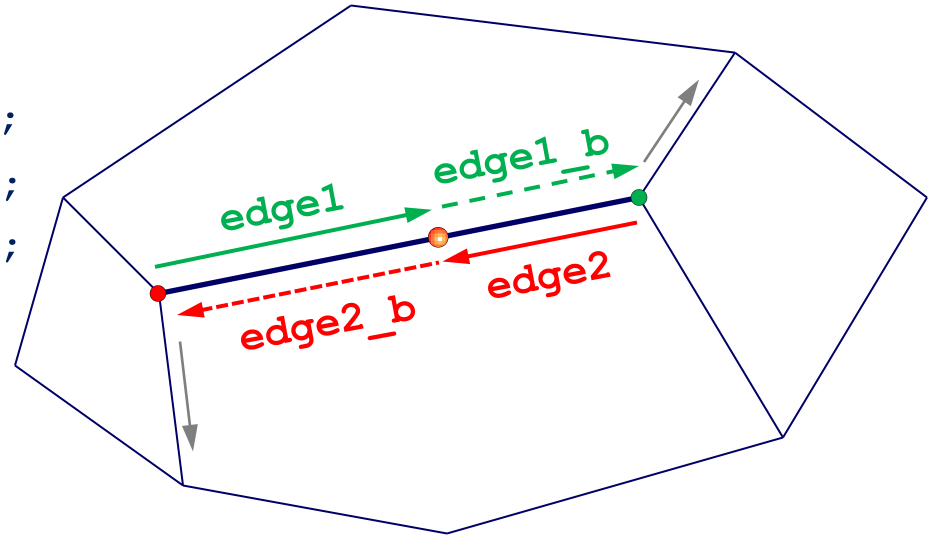


Simple Operations

Split an edge

```
HalfEdge edge1;
HalfEdge edge2 = edge1.opposite;
HalfEdge edge1_b = new HalfEdge;
HalfEdge edge2_b = new HalfEdge;

edge2_b.opposite = edge1;
edge2.opposite = edge1_b;
edge1_b.opposite = edge2;
edge1.opposite = edge2_b;
splitVert.halfEdge = edge1;
```



Other Operations

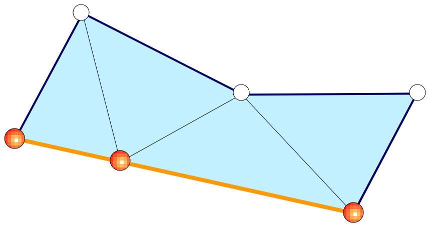
- Remove face(s)
  - Delete HalfEdgeFaces and any related topology that is unused elsewhere
  - Take care to properly RE-connect half edges/verts that are not on open boundary

Other Operations

- Unhook face(s)
  - Same as remove faces but copies removed face and related to another object

What Else Can We Do?

- Split a mesh in half with a cutting plane
  - Step 3: Remove or unhook faces on one side
  - Step 4: Find and cover open boundary loops
  - Step 5: Triangulate the remaining faces

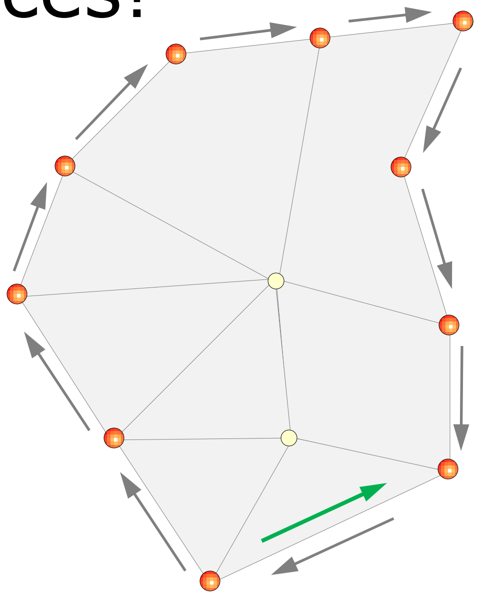


Edge Split Demo
Part 2

Pop Quiz!

Find the open boundary vertices!

```
IndexList Boundary;
Boundary =
        FindVertexLoop(startEdge->opposite);
```



(But what if the boundary isn't connected properly?)

*See speaker notes below slide for an important consideration!

Simple Traversals

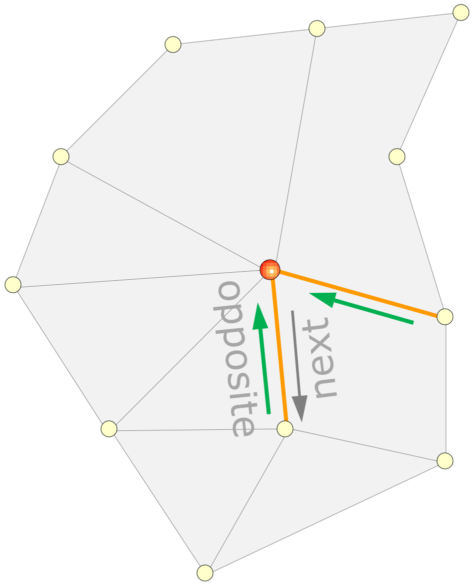
Find edges around a vertex

```
EdgeList FindEdgeRing(HalfEdgeVert *vert)
{
  EdgeList ring;
  HalfEdge *curEdge = vert->halfEdge;
  do {
    ring.push_back(curEdge);
    curEdge = curEdge->next->opposite;
  } while (curEdge != vert->halfEdge);
  return ring;
};
```

Simple Traversals

Find edges around a vertex

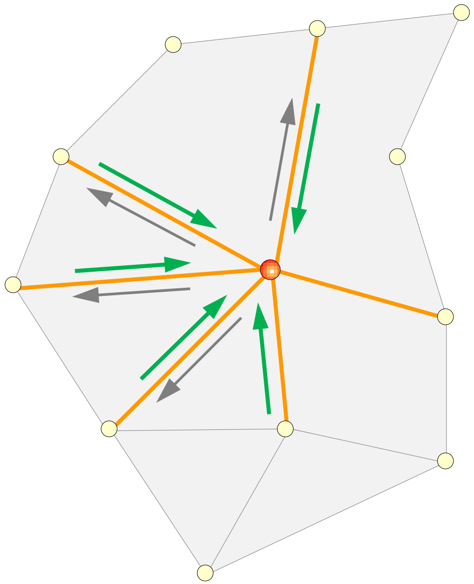
```
EdgeList FindEdgeRing(HalfEdgeVert *vert)
{
  EdgeList ring;
  HalfEdge *curEdge = vert->halfEdge;
  do {
    ring.push_back(curEdge);
    curEdge = curEdge->next->opposite;
  } while (curEdge != vert->halfEdge);
  return ring;
};
```



Simple Traversals

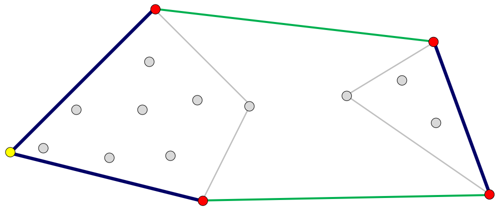
Find edges around a vertex

```
EdgeList FindEdgeRing(HalfEdgeVert *vert)
{
  EdgeList ring;
  HalfEdge *curEdge = vert->halfEdge;
  do {
    ring.push_back(curEdge);
    curEdge = curEdge->next->opposite;
  } while (curEdge != vert->halfEdge);
  return ring;
};
```



What Else Can We Do?

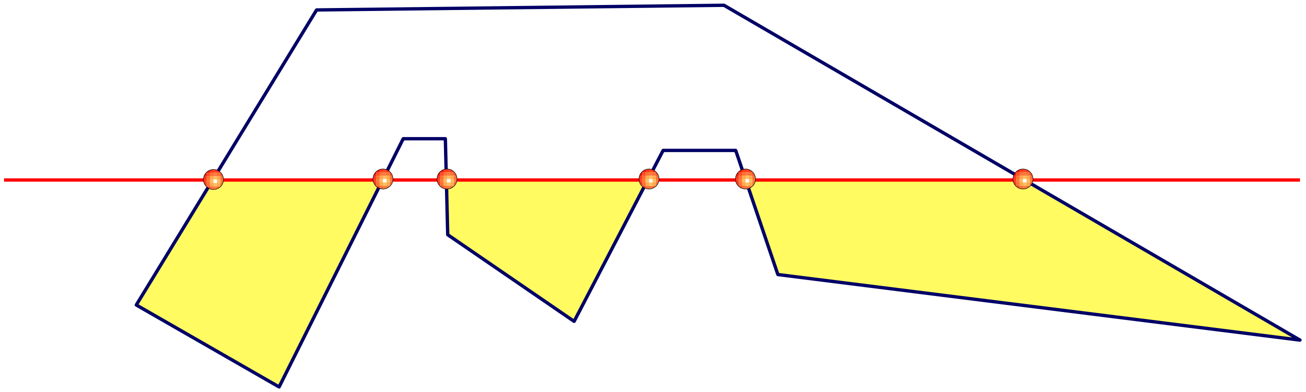
- Generate a convex hull mesh
  - Divide and conquer method is fast
    - O(n log n)



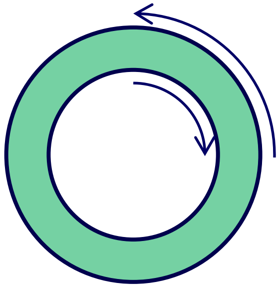
- Role of the half edge data structure
  - Remove interior faces/edges during stitch phase
  - Create new faces between boundary loops to perform the stitch

What can go wrong?

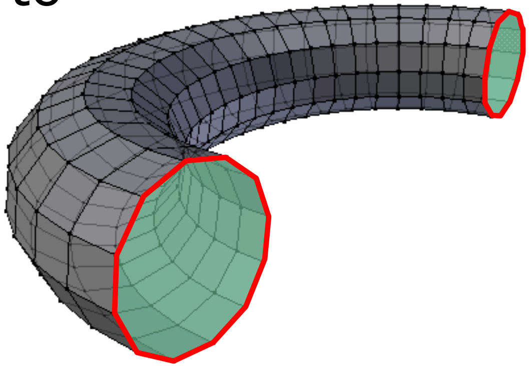
- Be careful when clipping concave face
  - Clipping against a plane can generate multiple loops
  - User marker flags to tag start and stop points
  - Recursively traverse to find ears to clip



What can go wrong?

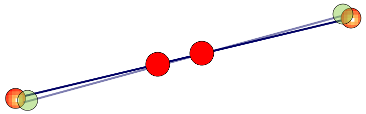


- Some scenarios produce multiple loops
  - Holes in a face
  - Requires additional triangulation logic
  - Nested loops: auxiliary edge to convert to simple polygon
  - Multiple un-nested loops: locate and triangulate each loop separately



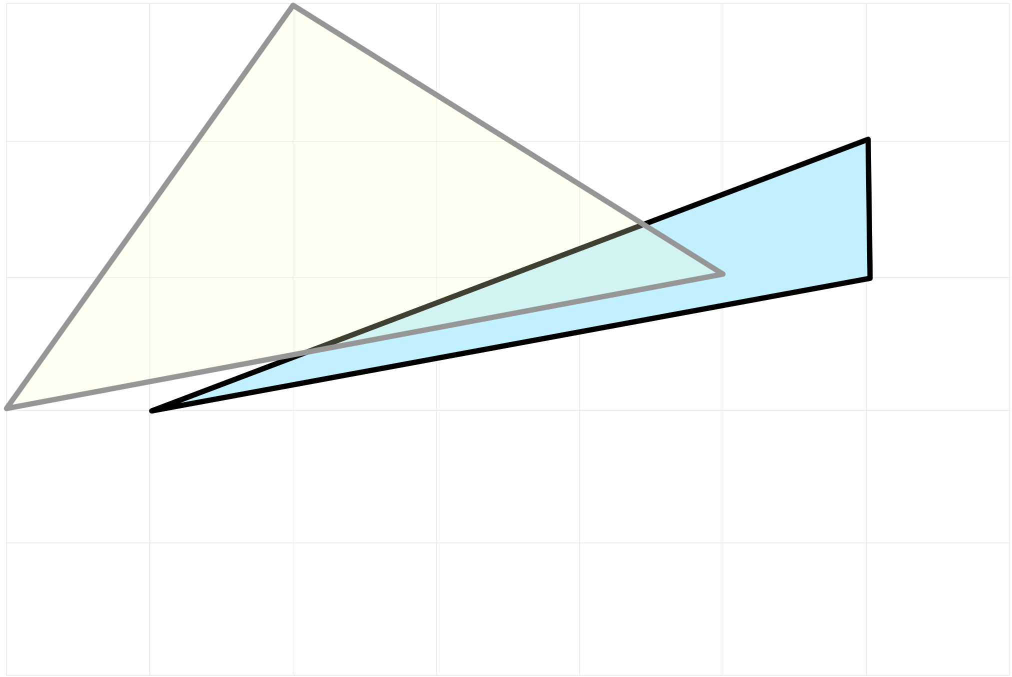
*See speaker notes below slide for an important consideration!

What can go wrong?

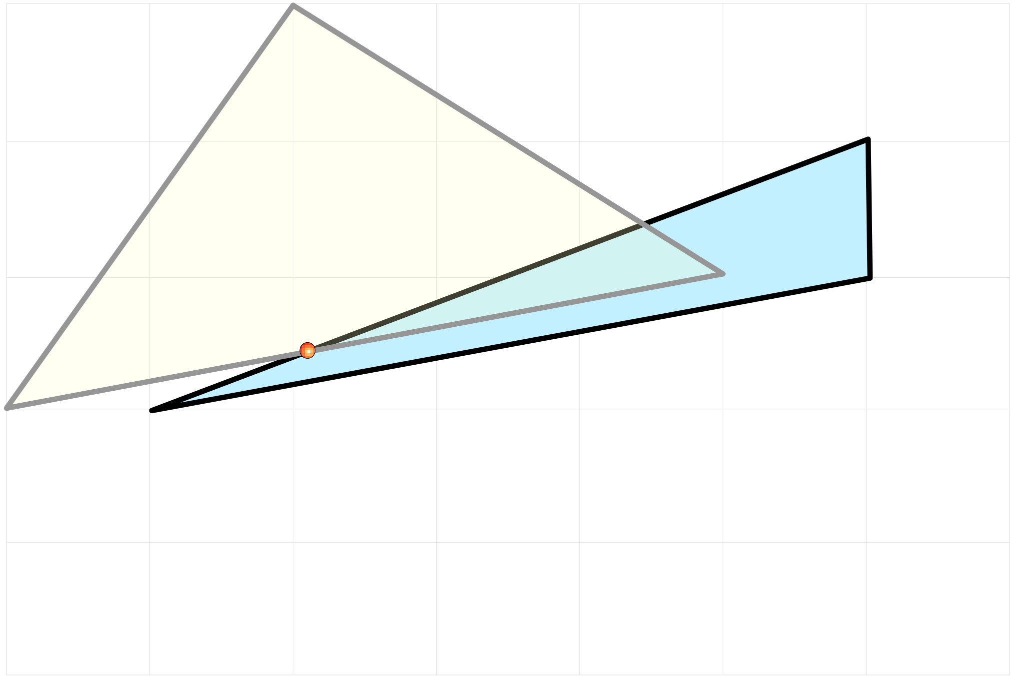


- Tolerance issues
  - Edges not quite collinear
    - Location of intersection point is highly sensitive
  - Points nearly collocated
    - Possible creation of very short/degenerate edges
  - On which side of an edge is a point?/Which face does a ray intersect?

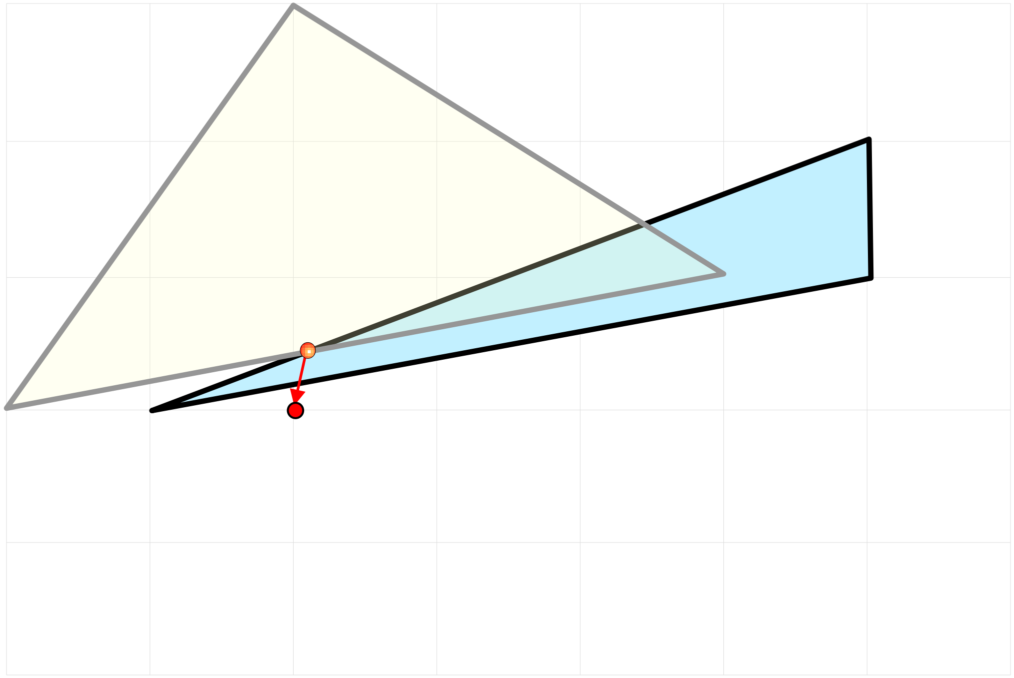
Orientation Inversion



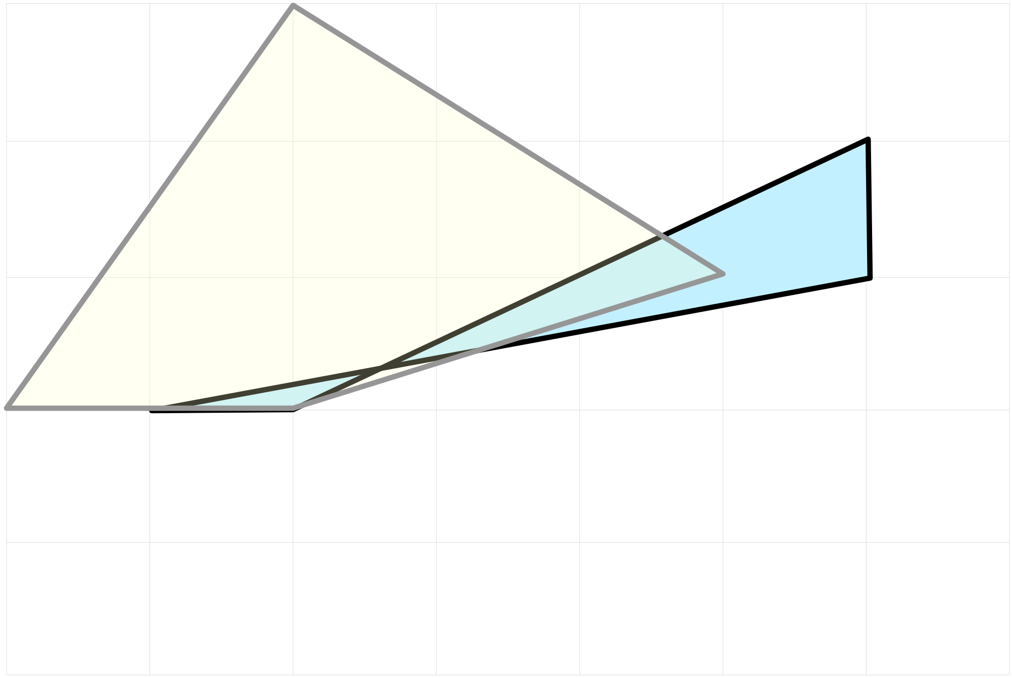
Orientation Inversion



Orientation Inversion

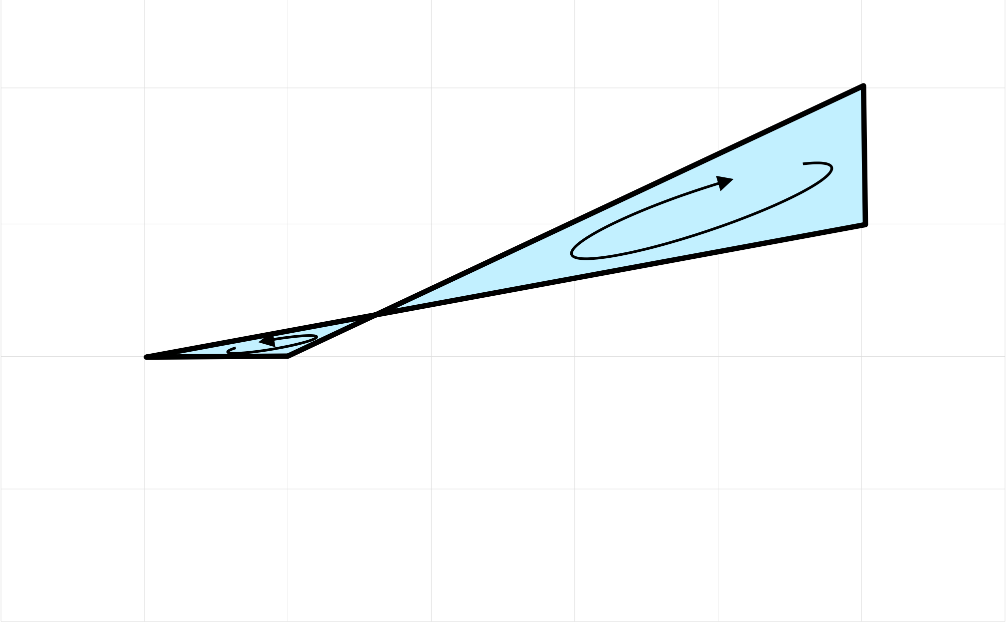


Orientation Inversion

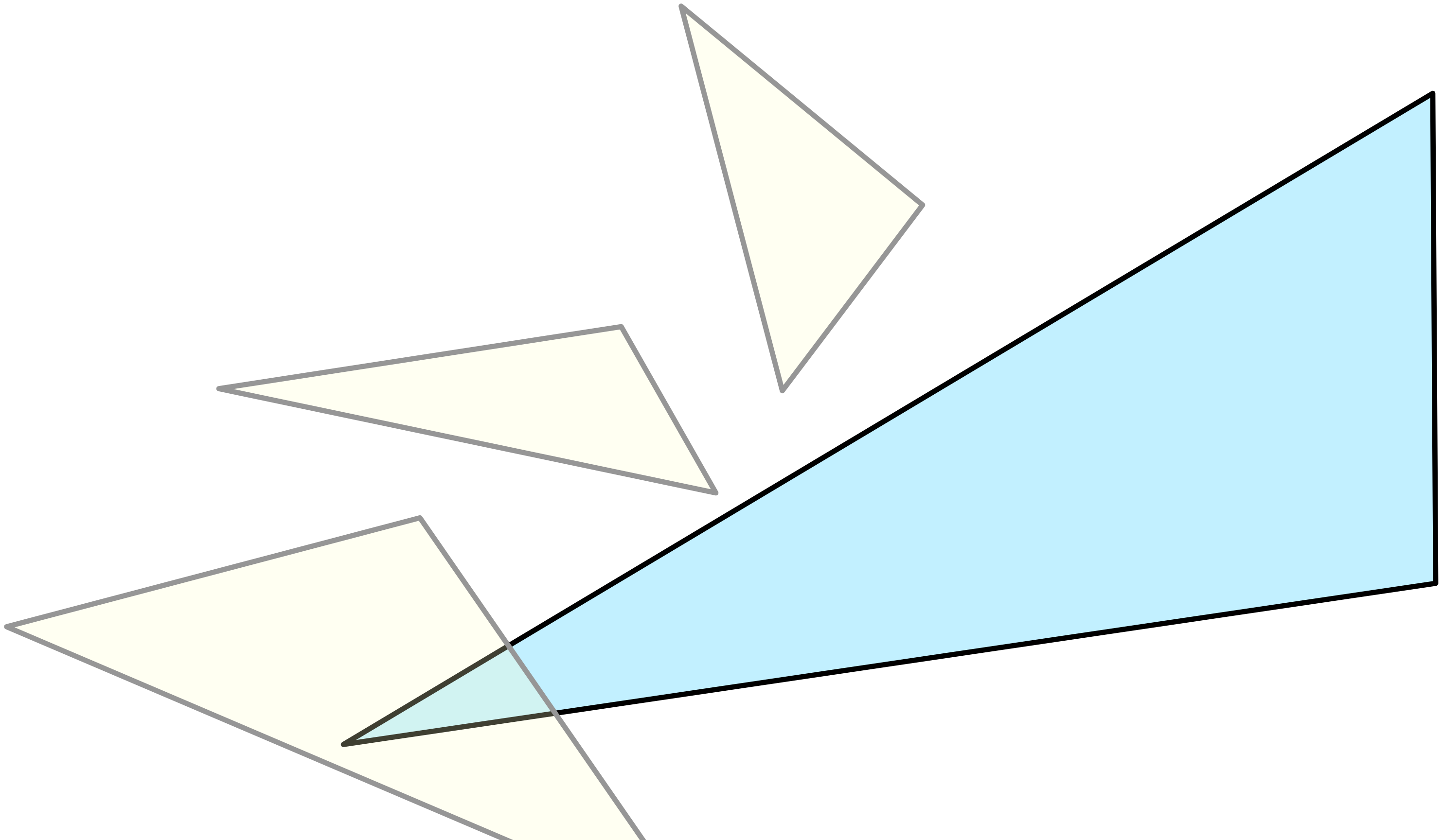


Orientation Inversion

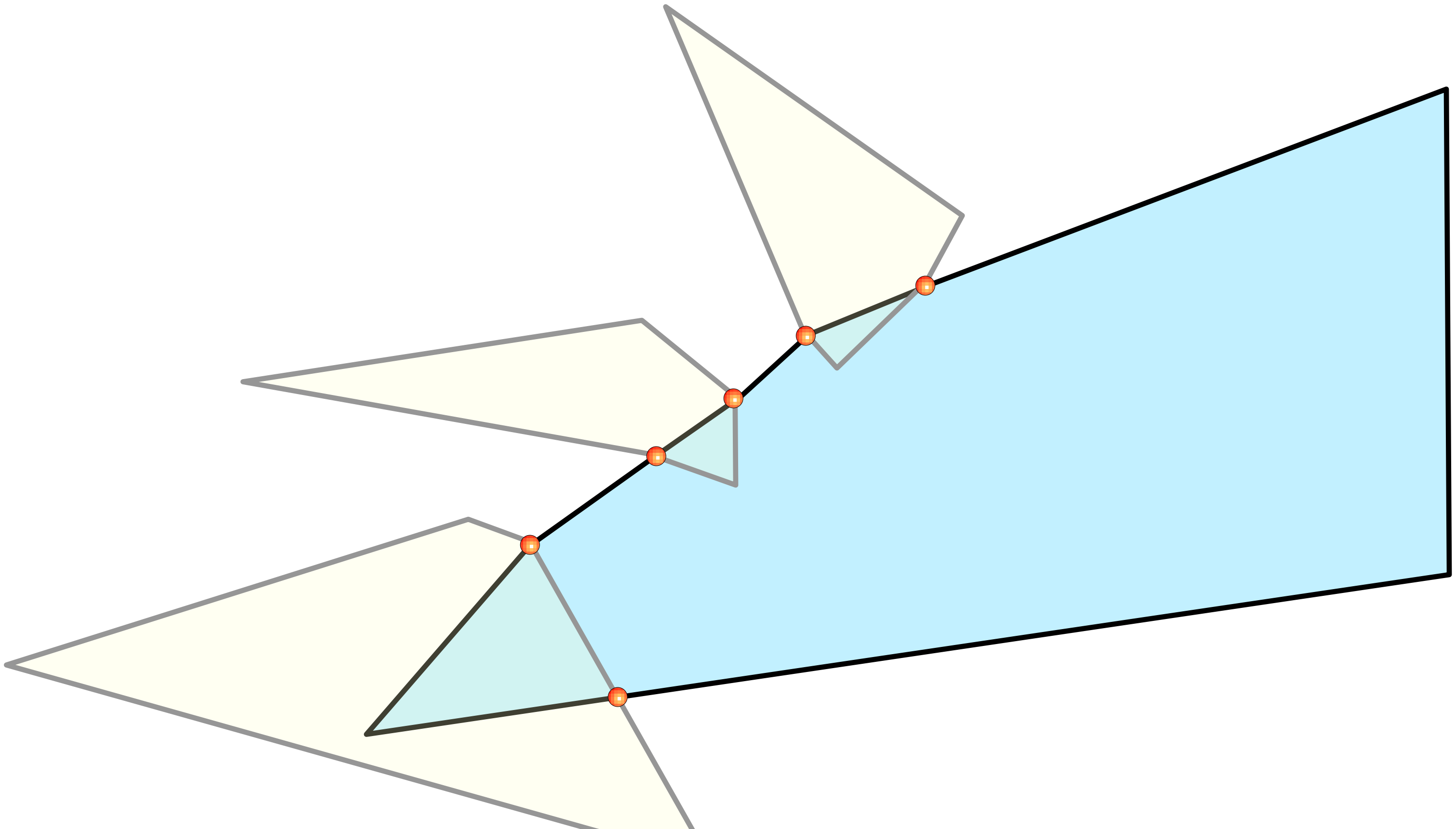
Polygon is no longer simple (it self-intersects) and no longer has a consistent orientation



Cascades of Extraneous Intersections

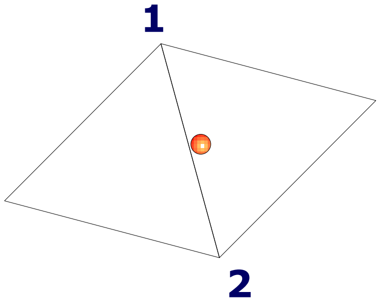


Cascades of Extraneous Intersections

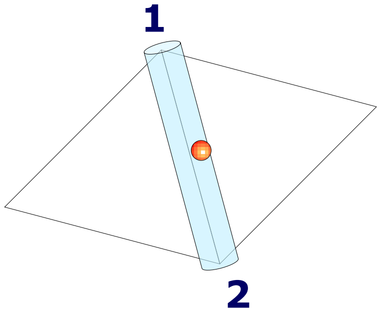


Tolerant Geometry

- Treat edges and points as thick primitives
  - Assign a radius to be used in intersection and proximity testing



Is point on edge?
On left side?
On right side?



Ambiguous answer depends on:
- Edge from 1->2 or 2->1
- Location

Point is **ON** the edge

References and Resources

- Sample code for half edge data structure
  - http://www.essentialmath.com
- These slides
  - See http://www.gdcvault.com after GDC
- References
  - http://www.cs.cornell.edu/courses/cs4620/2010fa/lectures/05meshes.pdf (Shirley & Marschner)
  - http://www.cgafaq.info/wiki/Half_edge_general
    - Nice discussion of invariants
  - http://people.csail.mit.edu/indyk/6.838-old/handouts/lec4.pdf
    - Polygon triangulation

References and Resources

- More references
  - Tolerant geometry and precision issues
    - Christer Ericson, *Real-time Collision Detection*
    - Jonathan Shewchuk's, "Adaptive Precision Floating-Point Arithmetic and Fast Robust Predicates for Computational Geometry"
    - John Hobby, "Practical Segment Intersection with Finite Precision Output" (snap rounding)

Questions?