VGP353 – Week 7

- Agenda:
  - Quiz #3
  - Generating shadow volume geometry
    - Side-trip into mesh data structures
    - Using a *real* mesh data structure to generate the shadow volume geometry
  - Assignments...

Shadow Volume Geometry

- Generating shadow volume geometry directly from raw vertex data is *hard*
  - Clearly some data structure is needed to make the work easier
- What features must this data structure have?

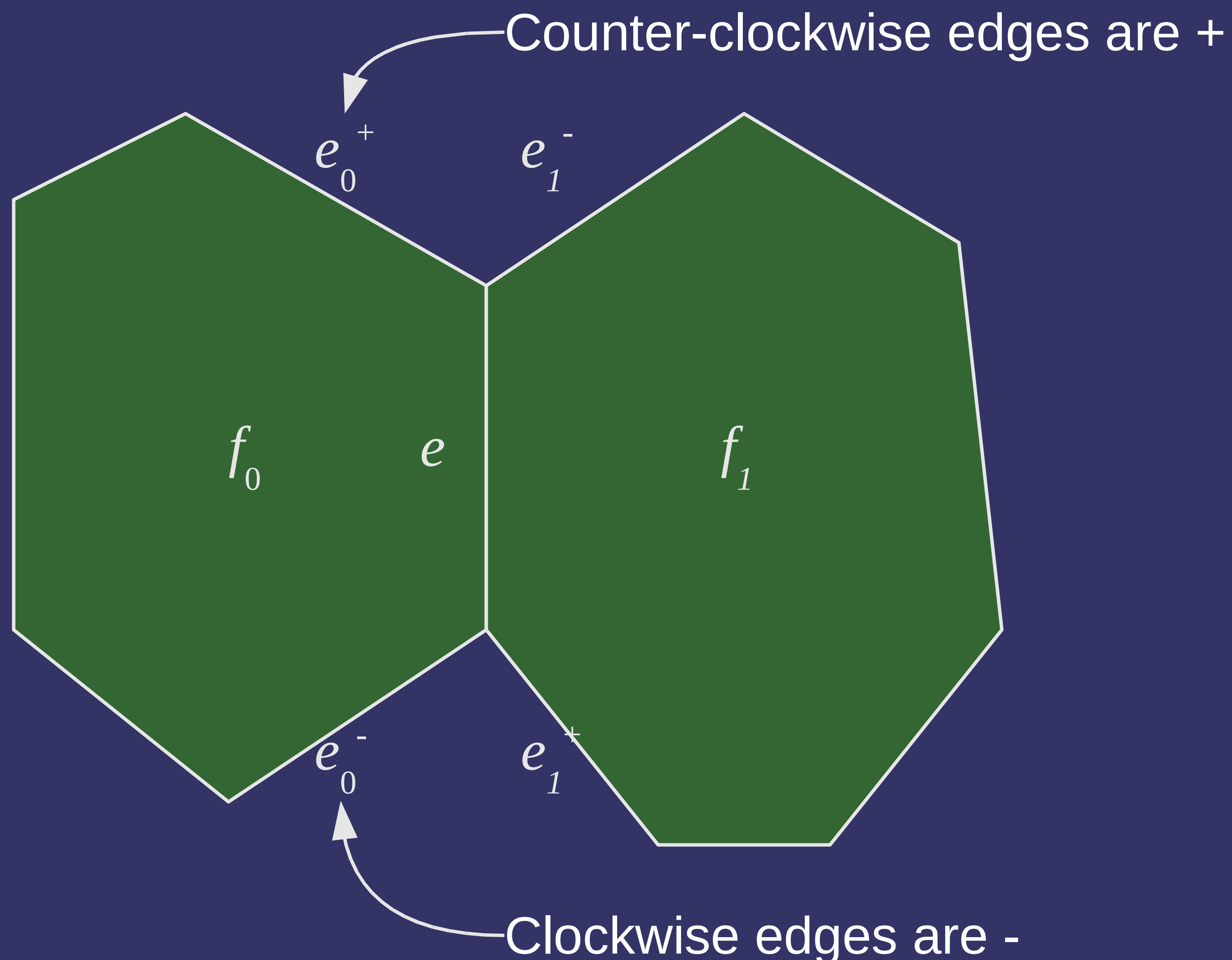
Shadow Volume Geometry

- Generating shadow volume geometry directly from raw vertex data is *hard*
  - Clearly some data structure is needed to make the work easier
- What features must this data structure have?
  - Iterate over each edge in the mesh *exactly once*
  - Access to each polygon sharing an edge
  - Access to neighboring edges in each polygon
    - This is so that normals can be calculated
- Does such a magical data structure exist?

Winged-Edge Mesh

- The *original* mesh structure to store connectivity information
- As the name implies, the focus is the *edge*
  - Each vertex stores a pointer to one of the edges "radiating" from it
  - Each polygon stores a pointer to one of its edges
  - Each edge has 8 pointers:
    - Pointers to each of its vertices (2)
    - Pointers to each of its polygons (2)
    - Pointers to each of its connecting edges (4)

Winged-Edge Mesh



13-May-2008

Winged-Edge Mesh

- Desirable mesh representation properties:
  - Ease of manipulation: adding and removing data should not be too expensive
  - Scalability: May want to trade data size for performance per the needs of the application

Winged-Edge Mesh

- Desirable mesh representation properties:
  - Ease of manipulation: adding and removing data should not be too expensive
  - Scalability: May want to trade data size for performance per the needs of the application

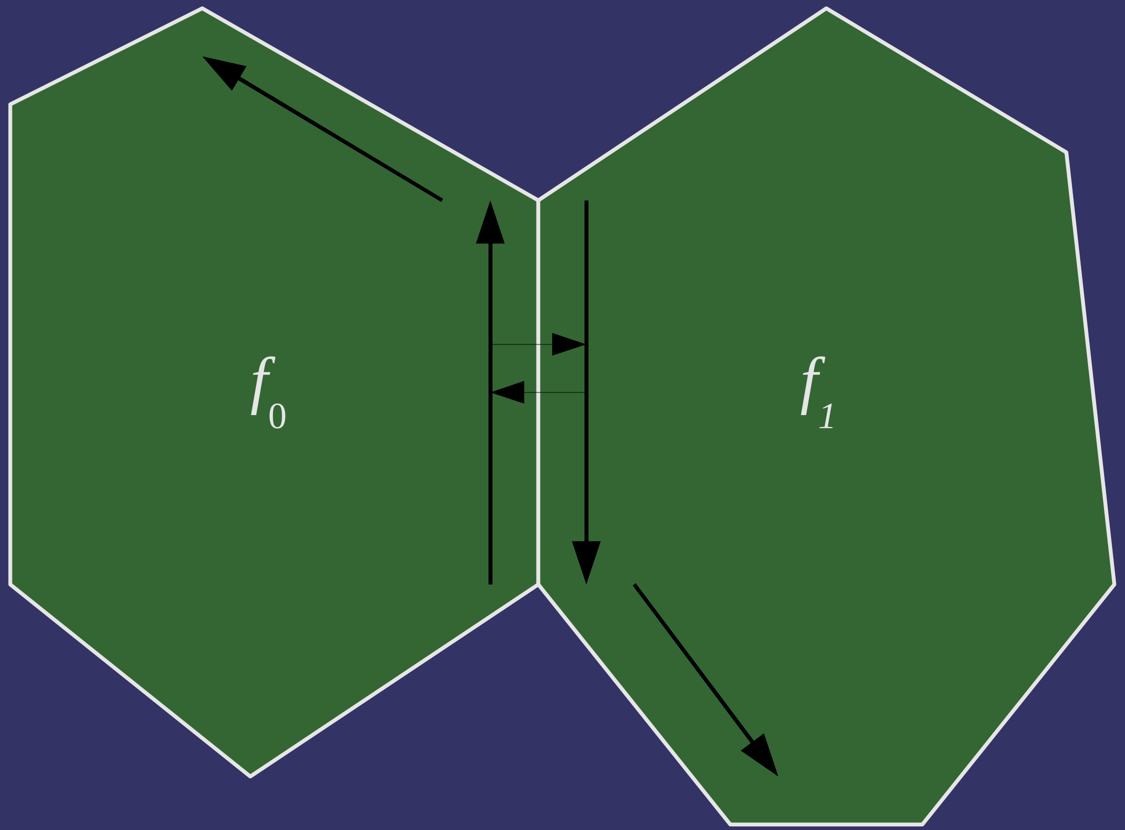
- Several common types of updates on WE meshes are *really* complicated to implement correctly

- Base winged-edge lacks the ability to iterate over the edges
- Base winged-edge has a *lot* of extra pointers that we will never use

Half-Edge Mesh

- Slight modification of winged-edge mesh:
  - Half-edge (HE) structures replace (full) edges
  - Each HE stores 4 pointers:
    - Pointer to starting vertex (1)
    - Pointer to polygon (1)
    - Pointer to counter-clockwise neighbor HE on the same polygon (1)
    - The “opposite” HE (1)
      - I call this the *sibling edge*
      - Other references call it *symmetric edge* or *pair edge*

Half-Edge Mesh



Half-Edge Mesh

```
struct half_edge {
    // Pointer to next counter-clockwise edge on same
    // polygon
    struct half_edge *next_ccw;

    // Pointer to matching edge on different polygon
    struct half_edge *sibling;

    // Pointer to the owning polygon
    struct polygon *p;

    // Pointer to next edge in global mesh edge list
    struct half_edge *next;

    // Pointer to starting vertex
    struct vertex *v;
};
```

Half-Edge Mesh

- If each HE only stores one vertex pointer, how do we get the other end?

Half-Edge Mesh

- If each HE only stores one vertex pointer, how do we get the other end?
  - The sibling edge stores a pointer to the other vertex
  - `e->v` and `e->sibling->v` make up the complete edge

Half-Edge Mesh

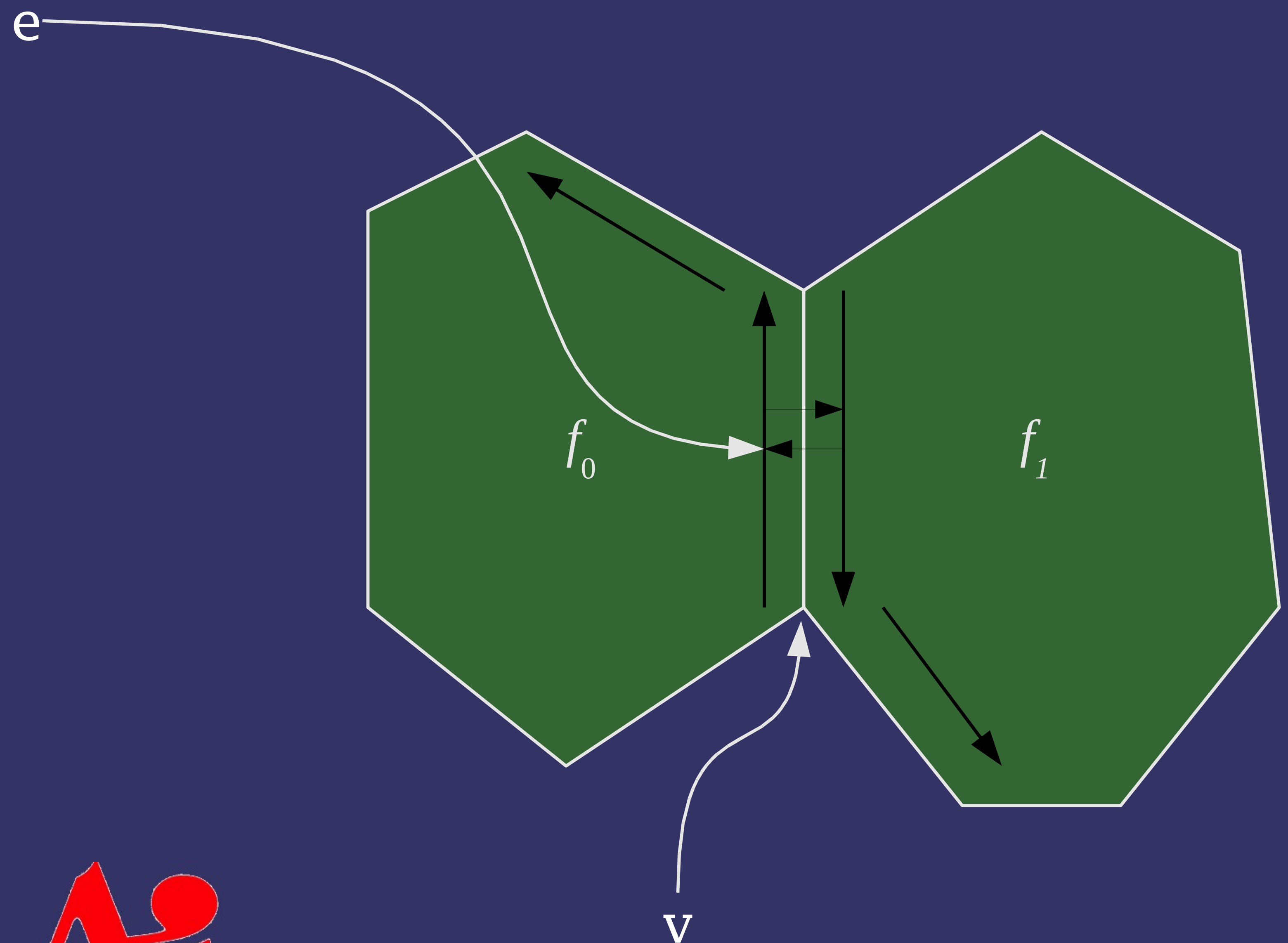
```
struct vertex {
    // Pointer an edge leaving this vertex
    struct half_edge *edge;

    // Pointer to position data for this vertex
    Vectormath::Aos::Vector4 *v;
};
```

Half-Edge Mesh

- Given a vertex structure, how can we iterate all the edges that share that vertex?

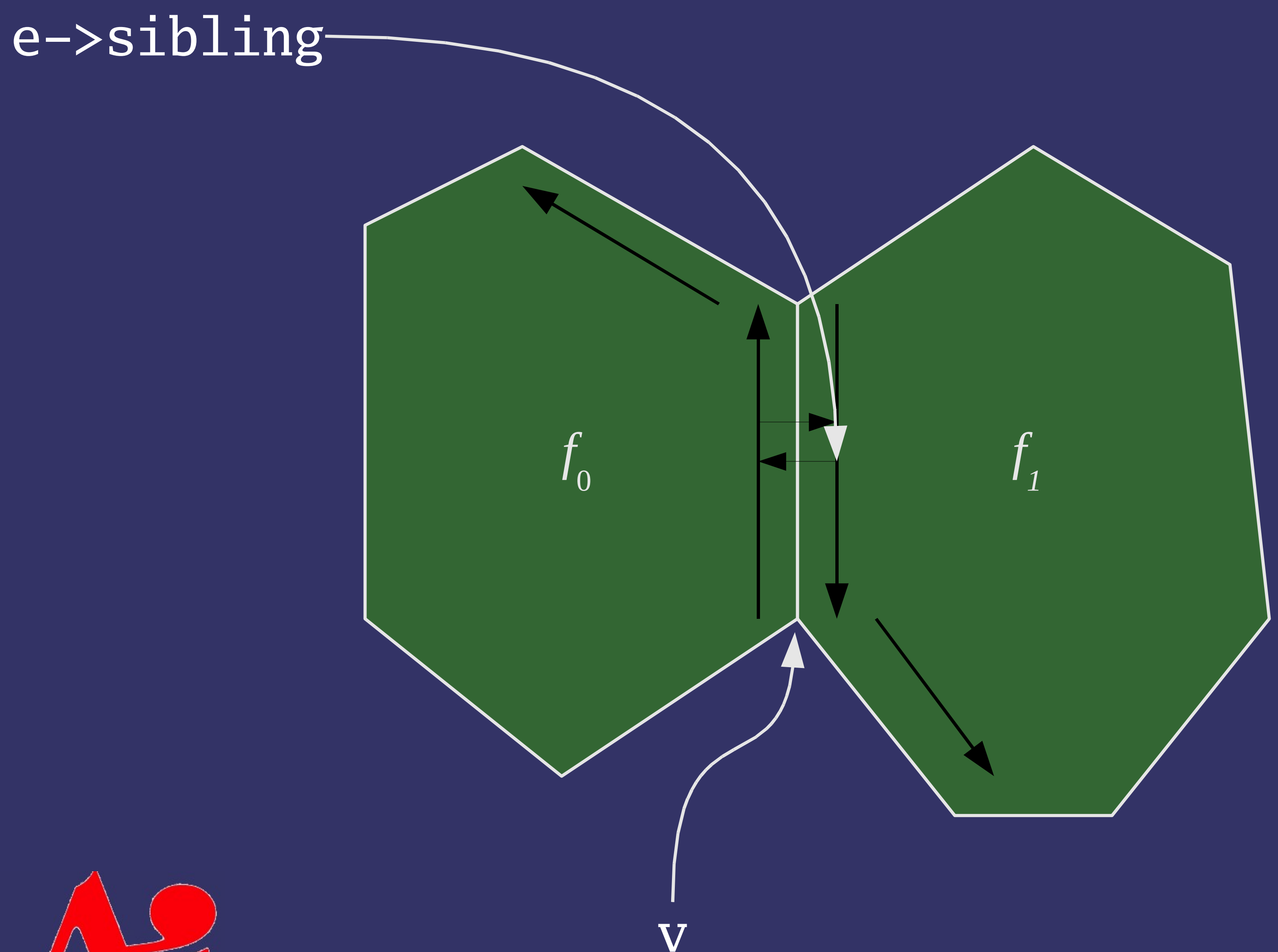
```
half_edge *e = v->edge;
do {
    // Do real work here.

    // Iterate to next edge
    e = e->sibling->next_ccw;
} while (e != v->edge);
```

Half-Edge Mesh



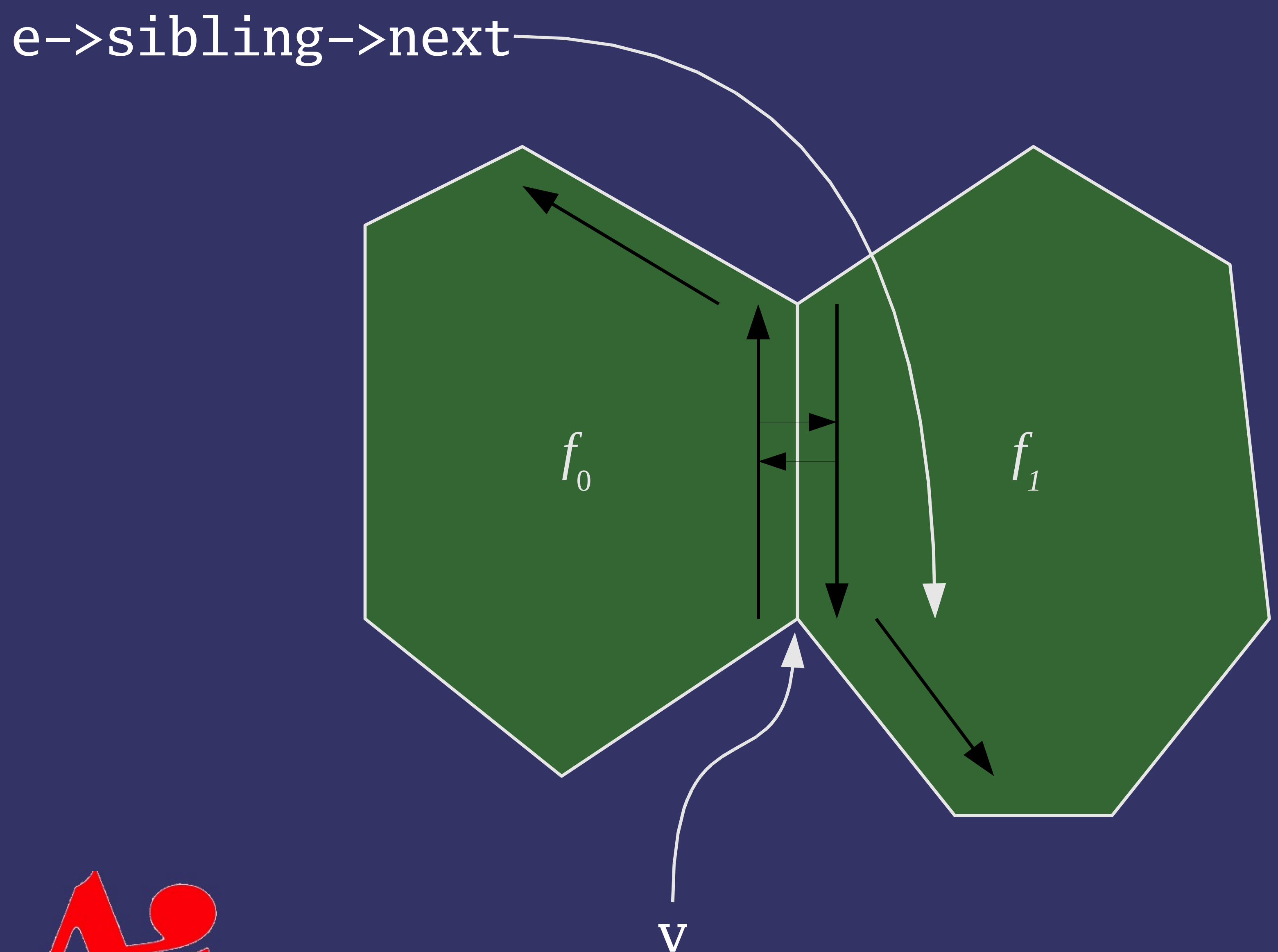
13-May-2008

Half-Edge Mesh



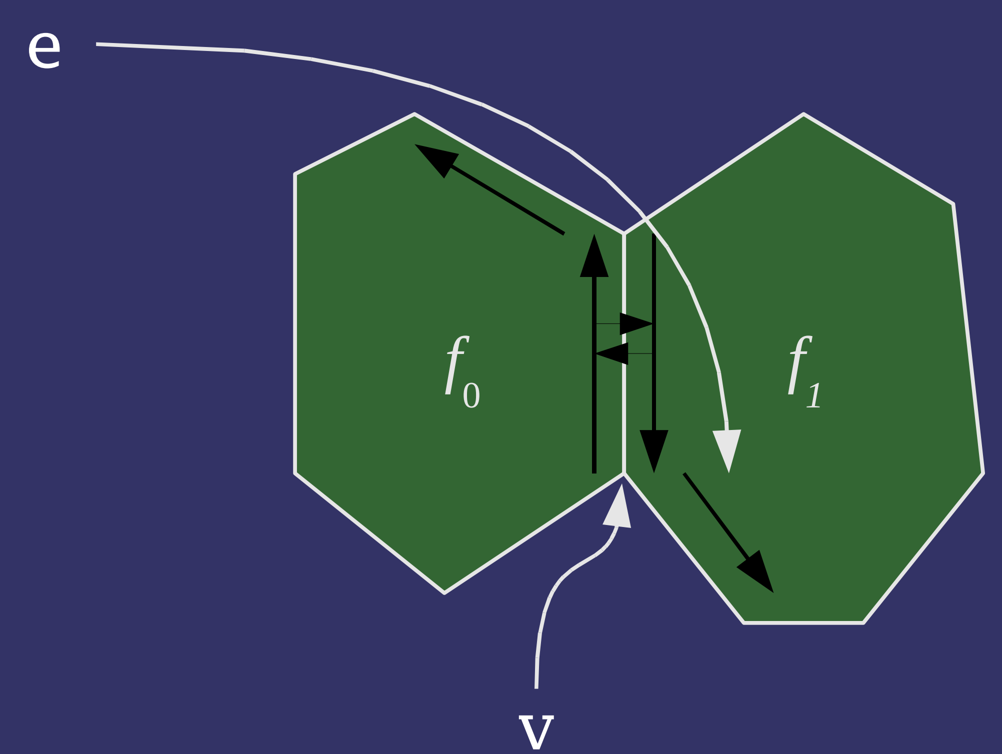
13-May-2008

Half-Edge Mesh



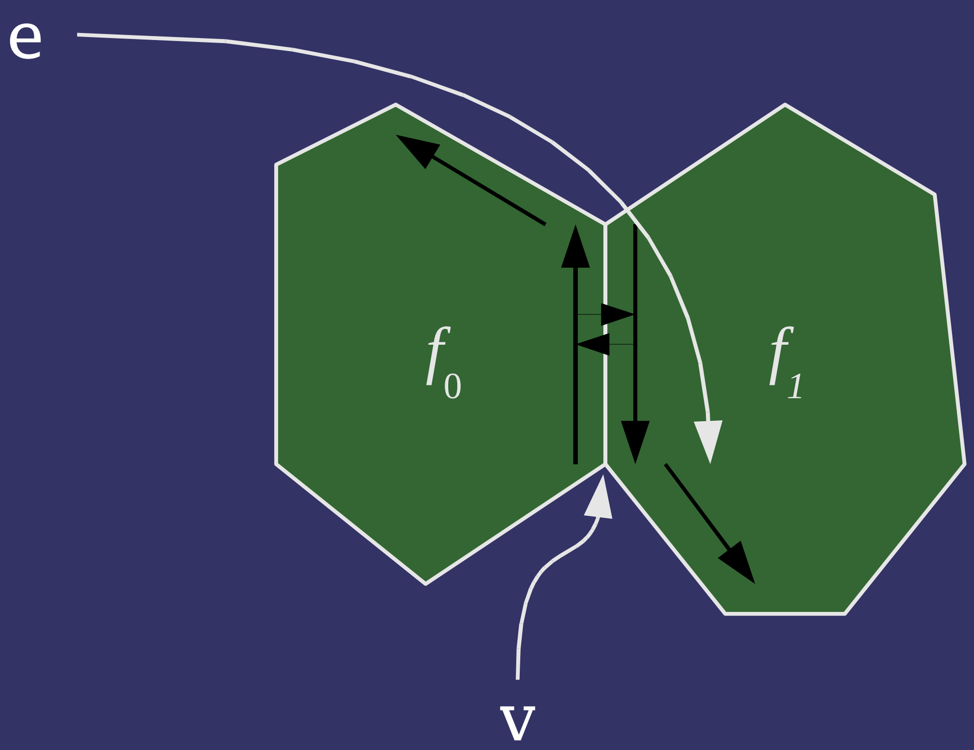
Half-Edge Mesh

- What's the problem?



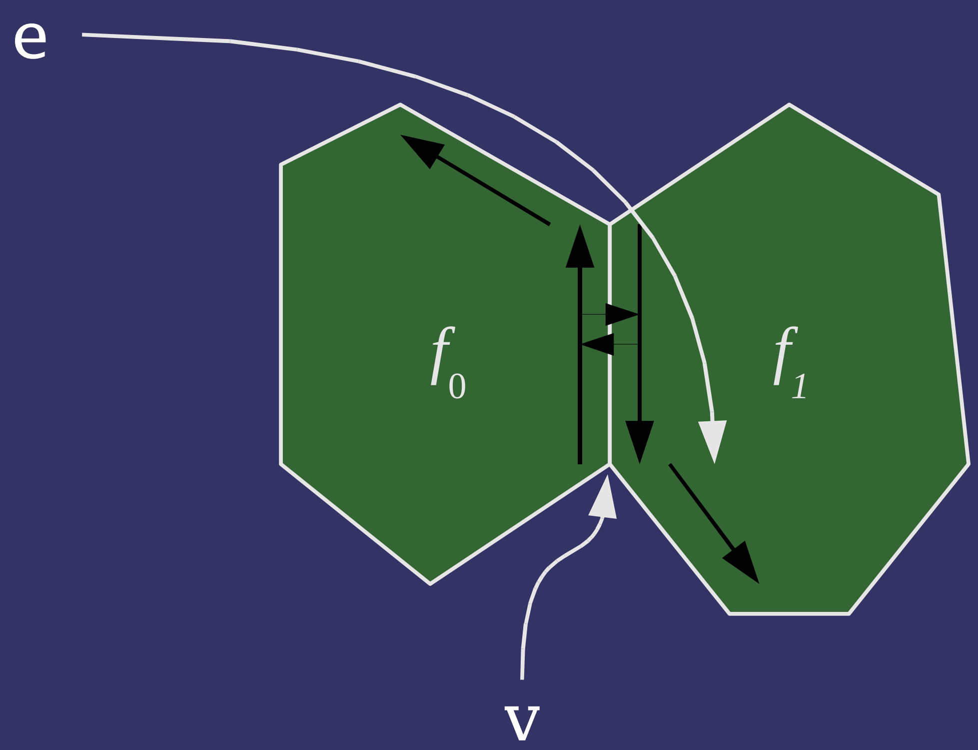
Half-Edge Mesh

- What's the problem?
  - The new e doesn't really have a sibling!
  - There are no pointers to follow to get the next edge



Half-Edge Mesh

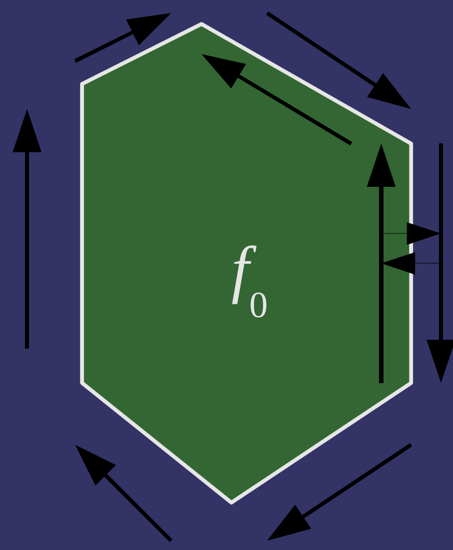
- How can we add new edges to the mesh and prevent this problem?



13-May-2008

Half-Edge Mesh

- How can we add new edges to the mesh and prevent this problem?
  - As new polygons are created, the sibling edges are linked in a “fake” CCW ring
    - The polygon pointers of these HEs is `NULL`
  - Adding *new* edges is a matter of updating all the linked lists



Half-Edge Mesh

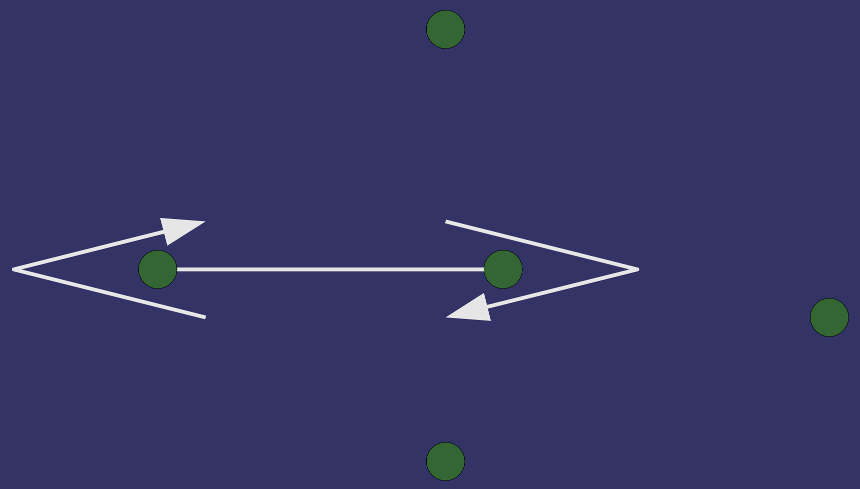
- To make the HE work, there are a few more primitives required
  - `create_edge(v0, v1)`: Create a new pair of HEs between `v0` and `v1`
  - `make_adjacent(a, b)`: Link `a` and `b` so that `a->next = b`
  - `add_polygon(edges, n)`: Create a new polygon from a list of existing edges

Half-Edge Mesh

- To create a *new* edge:
  - Allocate two HEs, link one to `v0` and the other to `v1`
  - Set both polygon pointers to `NULL`
  - Link both HEs as siblings
  - Link both HEs as each other's `next_ccw`
    - Tricky! This makes the bootstrap case work and fixes other issues in `make_adjacent`
  - Insert each edge in the “gap” in the vertex's edge list
    - Some HE where:
      - `e->sibling->v == v`
      - `e->p == NULL`
      - `e->next_ccw->v == v`

Half-Edge Mesh

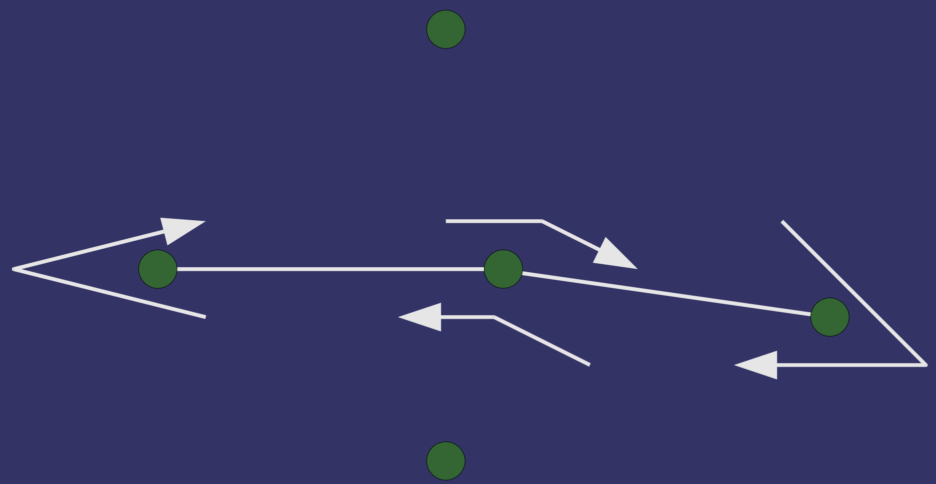
- Edges can be added in arbitrary order...



13-May-2008

Half-Edge Mesh

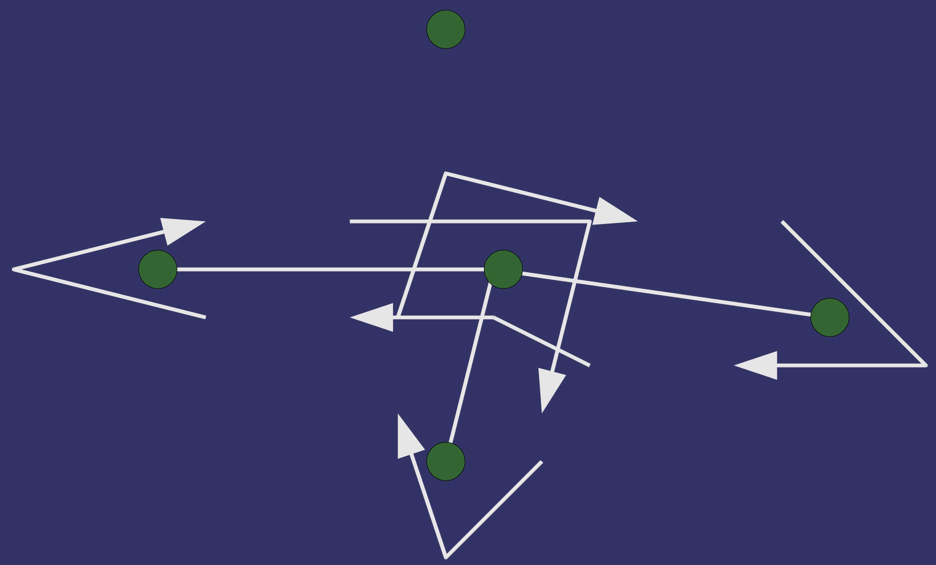
- Edges can be added in arbitrary order...



13-May-2008

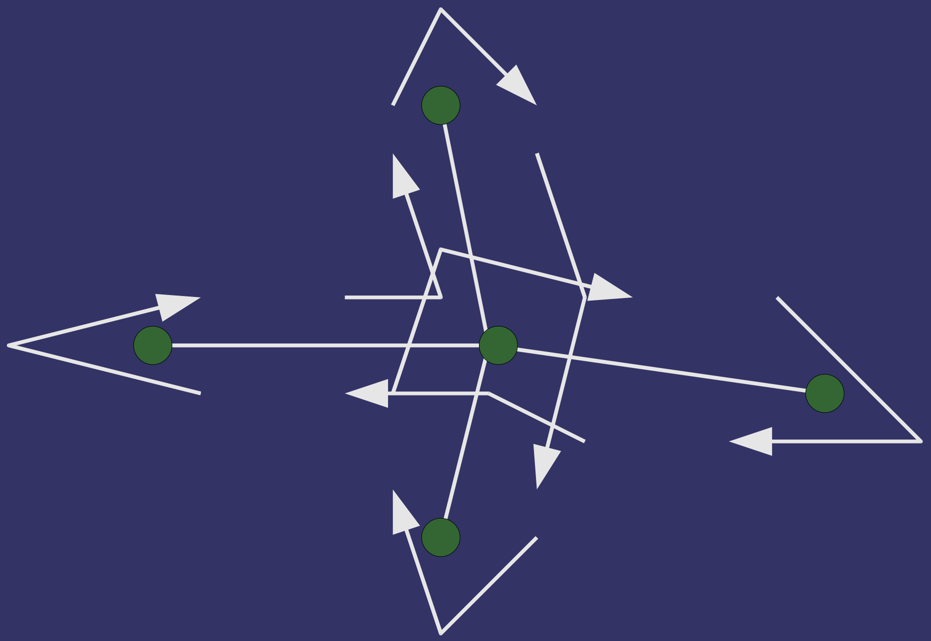
Half-Edge Mesh

- Edges can be added in arbitrary order...



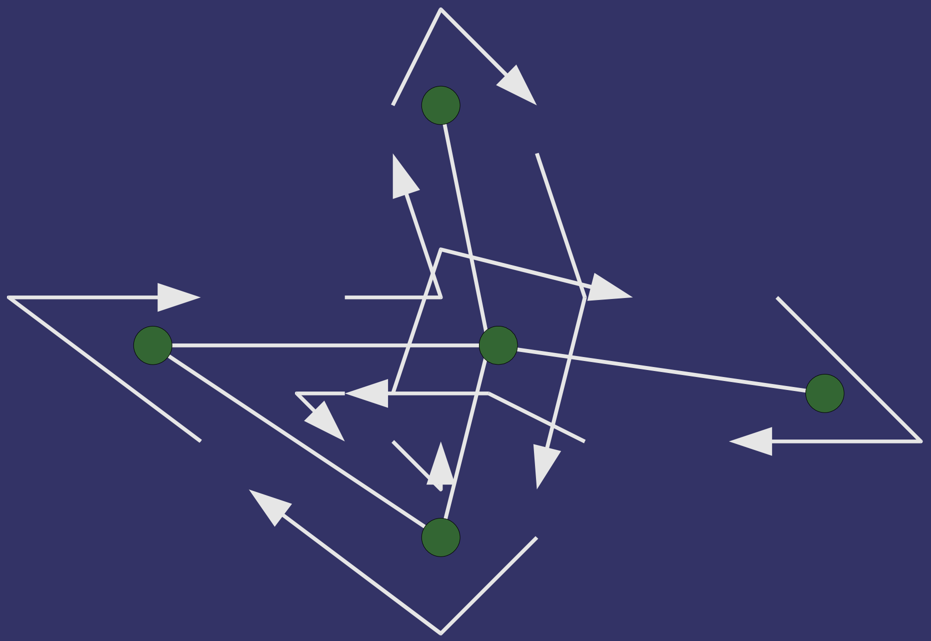
Half-Edge Mesh

- Edges can be added in arbitrary order...



Half-Edge Mesh

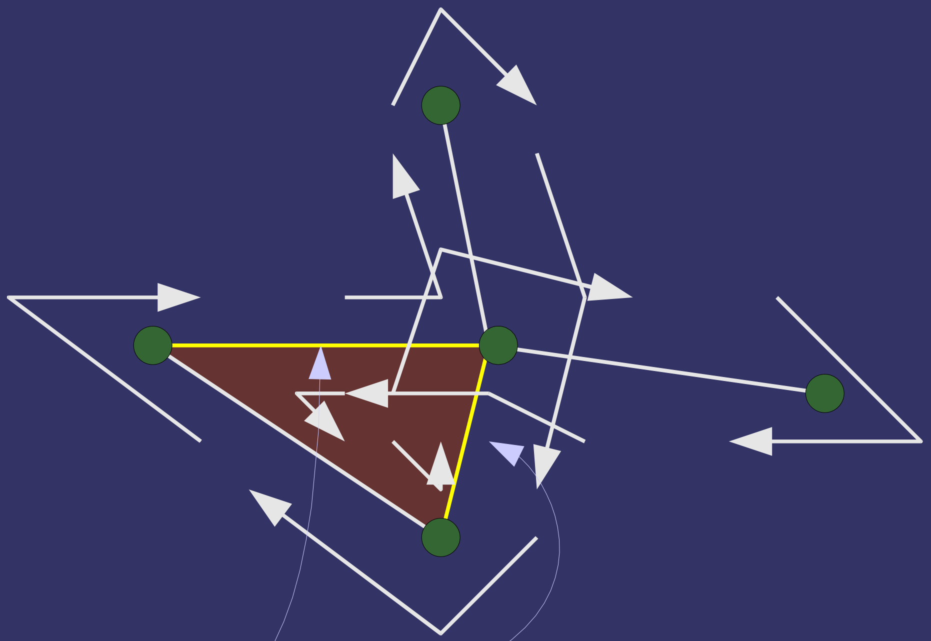
- Edges can be added in arbitrary order...



13-May-2008

Half-Edge Mesh

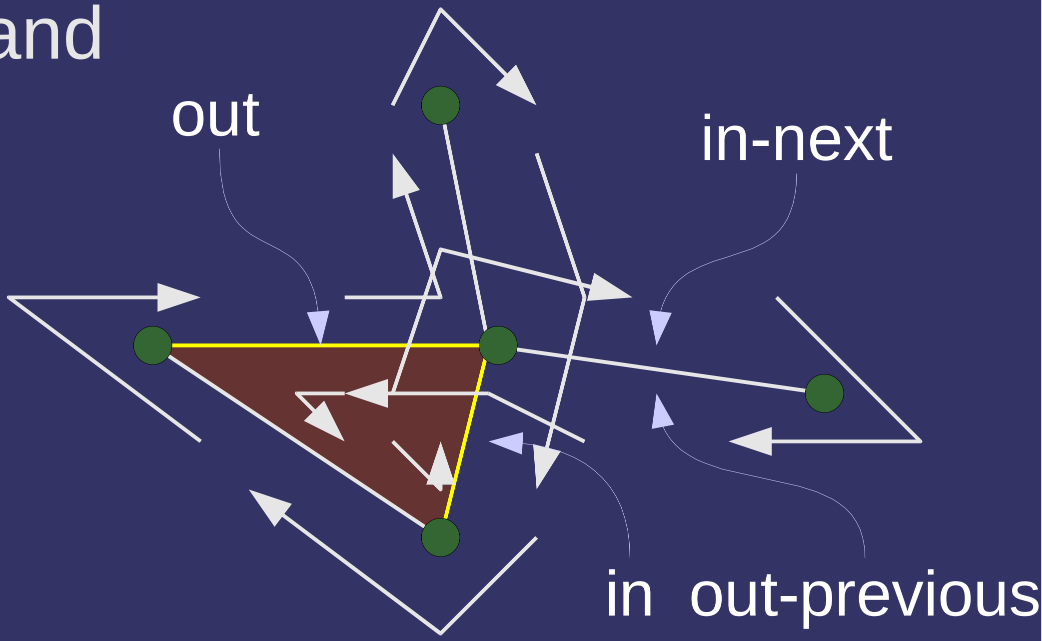
- Edges can be added in arbitrary order...
  - This causes problems when edges are formed into a polygon



These edges should be linked

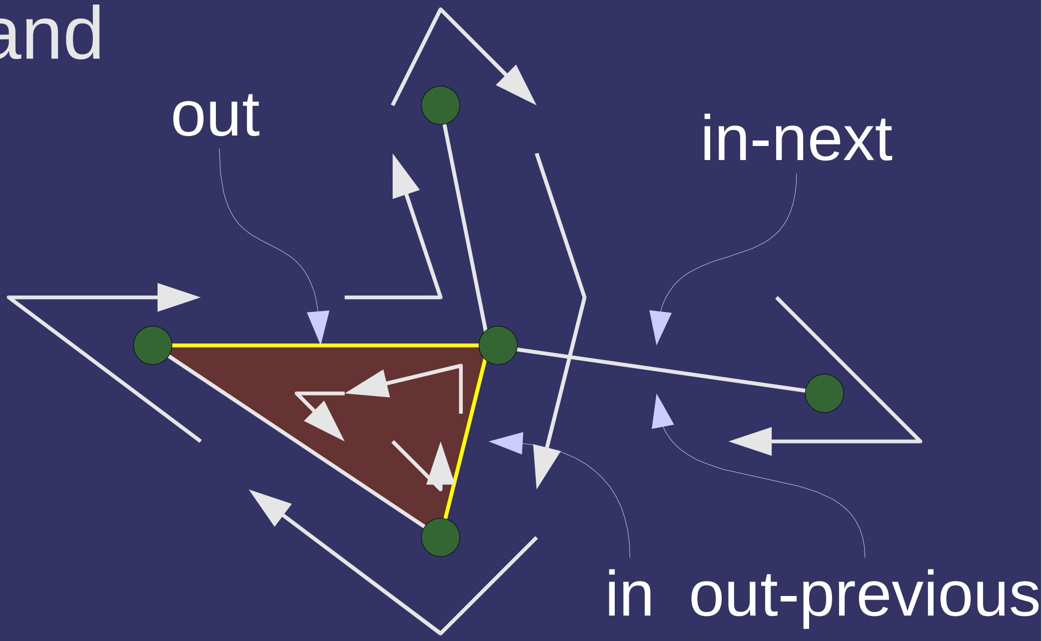
Half-Edge Mesh

- Relink the edges to create the correct relationships
  - Cut the links between *in* and *in-next*, and between *out* and *out-previous*



Half-Edge Mesh

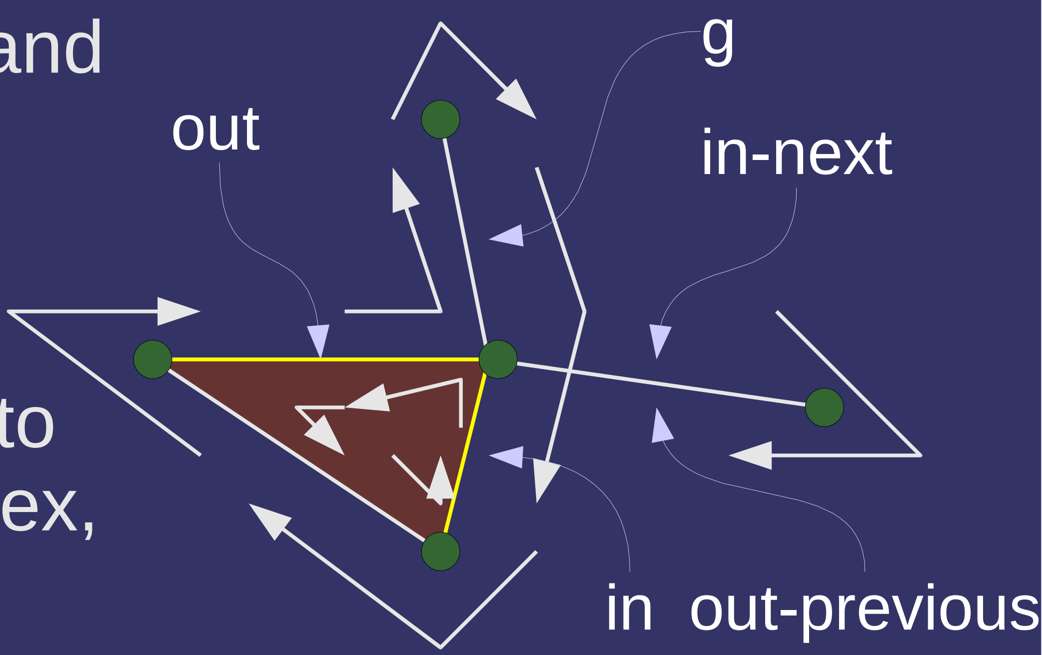
- Relink the edges to create the correct relationships
  - Cut the links between *in* and *in-next*, and between *out* and *out-previous*
  - Link *in* and *out*



13-May-2008

Half-Edge Mesh

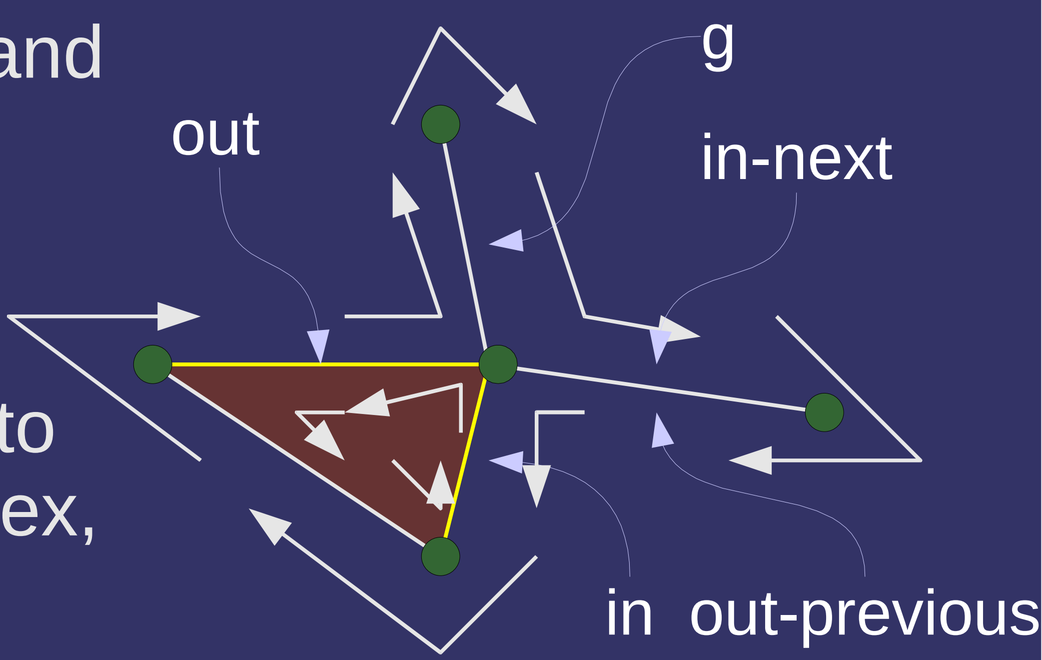
- Relink the edges to create the correct relationships
  - Cut the links between *in* and *in-next*, and between *out* and *out-previous*
  - Link *in* and *out*
  - Find a free edge going into *in* and *out's* common vertex, call it *g*
    - This edge must be between *out-sibling* and *in*



13-May-2008

Half-Edge Mesh

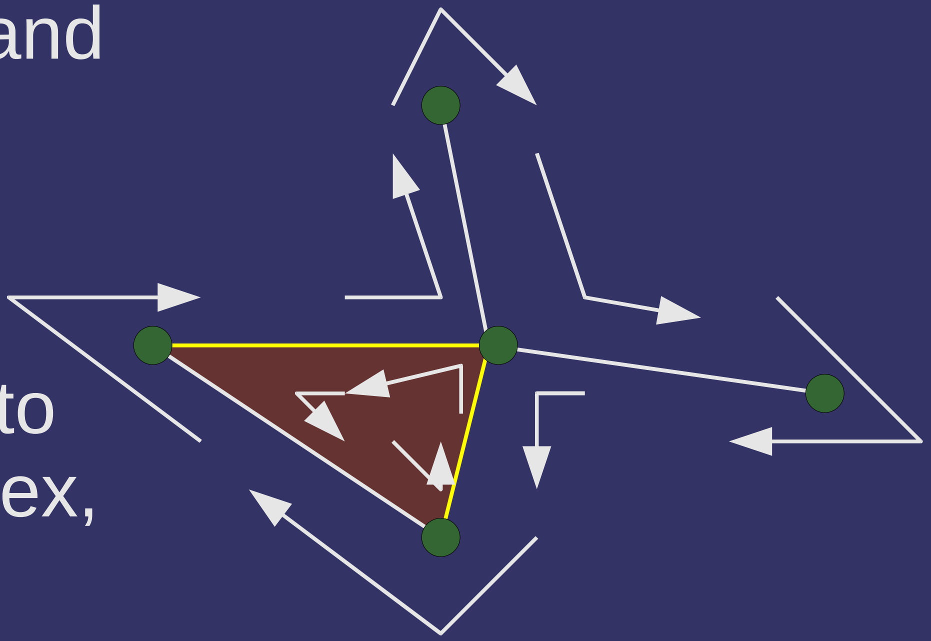
- Relink the edges to create the correct relationships
  - Cut the links between *in* and *in-next*, and between *out* and *out-previous*
  - Link *in* and *out*
  - Find a free edge going into *in* and *out's* common vertex, call it *g*
    - This edge must be between *out-sibling* and *in*
  - Link *g* to *in-next*
  - Link *out-previous* to *g-next*



13-May-2008

Half-Edge Mesh

- Relink the edges to create the correct relationships
  - Cut the links between *in* and *in-next*, and between *out* and *out-previous*
  - Link *in* and *out*
  - Find a free edge going into *in* and *out's* common vertex, call it *g*
    - This edge must be between *out-sibling* and *in*
  - Link *g* to *in-next*
  - Link *out-previous* to *g-next*



Half-Edge Mesh

- With these primitives, adding a new polygon is easy
  - For all edges, verify that the end point of one edge and the start point of the next edge is the same
  - For all edges, verify that the edge is not already associated with a polygon
  - For all edges, connect the edge to the next edge in the list
  - Allocate a new polygon object and connect all of the edges to it

References

Matt Pharr and Ken Schoemake, ed. *comp.graphics.algorithims FAQ.* Accessed 13 May 2008; available from http://cgafaq.info/wiki/Geometric_data_structures; Internet.

Shadow Volume Geometry

- Once we have a model stored half-edge or winged-edge data structure, how do we generate the shadow volume geometry?

Shadow Volume Geometry

- Once we have a model stored half-edge or winged-edge data structure, how do we generate the shadow volume geometry?
  - For each edge in the mesh:
    - If the either of the edge's polygon pointers is `NULL`, skip the edge
    - Calculate the normal of each polygon sharing the edge, call these n_0 and n_1
    - If n_0 and n_1 are equal, skip the edge
      - This happens if the surfaces are co-planar, and can *never* be on the silhouette
    - Emit a quad of (v_0, n_0), (v_1, n_0), (v_1, n_1), (v_0, n_1)

Fixing Object Geometry

- What about edges with `NULL` polygon pointers?

Fixing Object Geometry

- What about edges with `NULL` polygon pointers?
  - These represent *holes* in the model
    - The Stanford bunny model has several holes in the bottom
  - For each hole, the hole-edges form a ring

Fixing Object Geometry

- What about edges with `NULL` polygon pointers?
  - These represent *holes* in the model
    - The Stanford bunny model has several holes in the bottom
  - For each hole, the hole-edges form a ring
- What can we do with this?

Fixing Object Geometry

- What about edges with `NULL` polygon pointers?
  - These represent *holes* in the model
    - The Stanford bunny model has several holes in the bottom
  - For each hole, the hole-edges form a ring
- What can we do with this?
  - Walk the hole-edge ring and insert *new* edges between each pair of hole-edges
  - Each new edge will forms a triangle that fills part of the hole
  - Do this step *before* generating shadow volume geometry

Next week...

- Advanced shadow volume techniques:
  - Fixing z-pass and z-fail with ZP+
  - Hardware based optimizations:
    - Depth clamping
    - Depth bounds testing

Legal Statement

This work represents the view of the authors and does not necessarily represent the view of IBM or the Art Institute of Portland.

OpenGL is a trademark of Silicon Graphics, Inc. in the United States, other countries, or both.

Khronos and OpenGL ES are trademarks of the Khronos Group.

Other company, product, and service names may be trademarks or service marks of others.