Triangle meshes

Carlos Andújar

April 2014

Face array

- OK for “triangle soups”. Exemple: STL
- Vertex duplication
- No explicit connectivity information

Triangles		
x_{11} y_{11} z_{11}	x_{12} y_{12} z_{12}	x_{13} y_{13} z_{13}
x_{21} y_{21} z_{21}	x_{22} y_{22} z_{22}	x_{23} y_{23} z_{23}
...	...	...
x_{F1} y_{F1} z_{F1}	x_{F2} y_{F2} z_{F2}	x_{F3} y_{F3} z_{F3}

Face array + vertex array

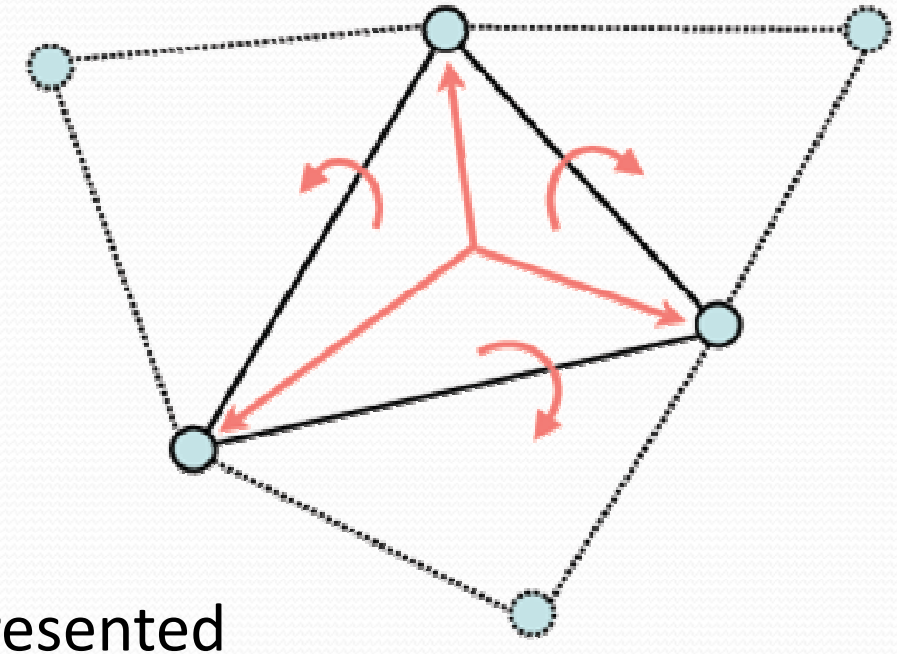
- Example: format OBJ, OFF, PLY
- No vertex duplication
- No neighbor info

Vertices
x_1 y_1 z_1
...
x_V y_V z_V

Triangles
v_{11} v_{12} v_{13}
...
...
...
...
v_{F1} v_{F2} v_{F3}

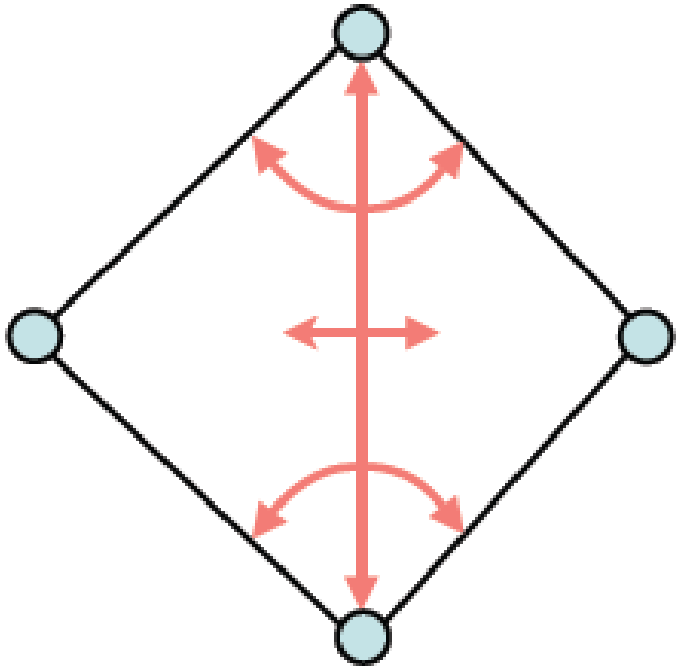
Face-based connectivity

- Vertex:
  - Position (x,y,z)
  - 1 Face(ref)
- **Face**:
  - 3 vertices (refs)
  - 3 faces (refs)
- Edges are not explicitly represented



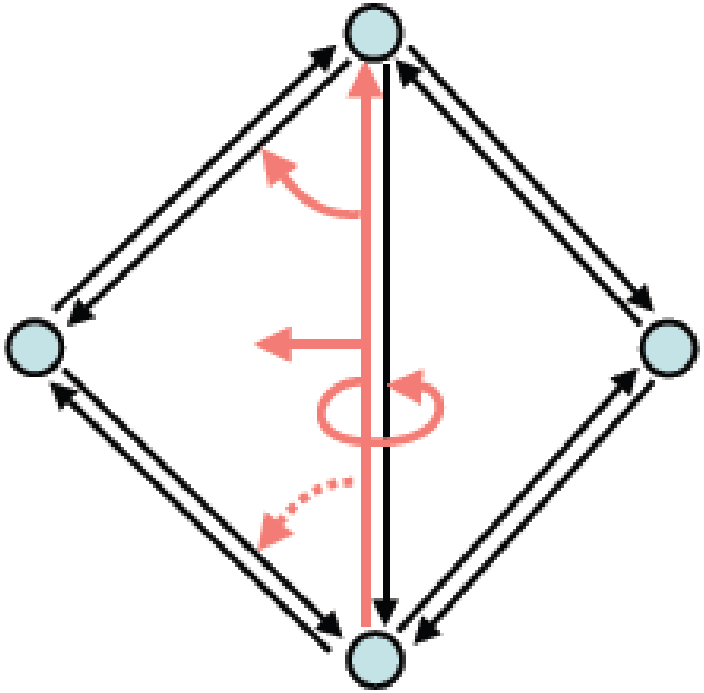
Edge-based connectivity

- Vertex:
  - position
  - 1 edge
- **Edge**:
  - 2 vertices
  - 2 faces
  - 4 edges
- Face:
  - 1 edge



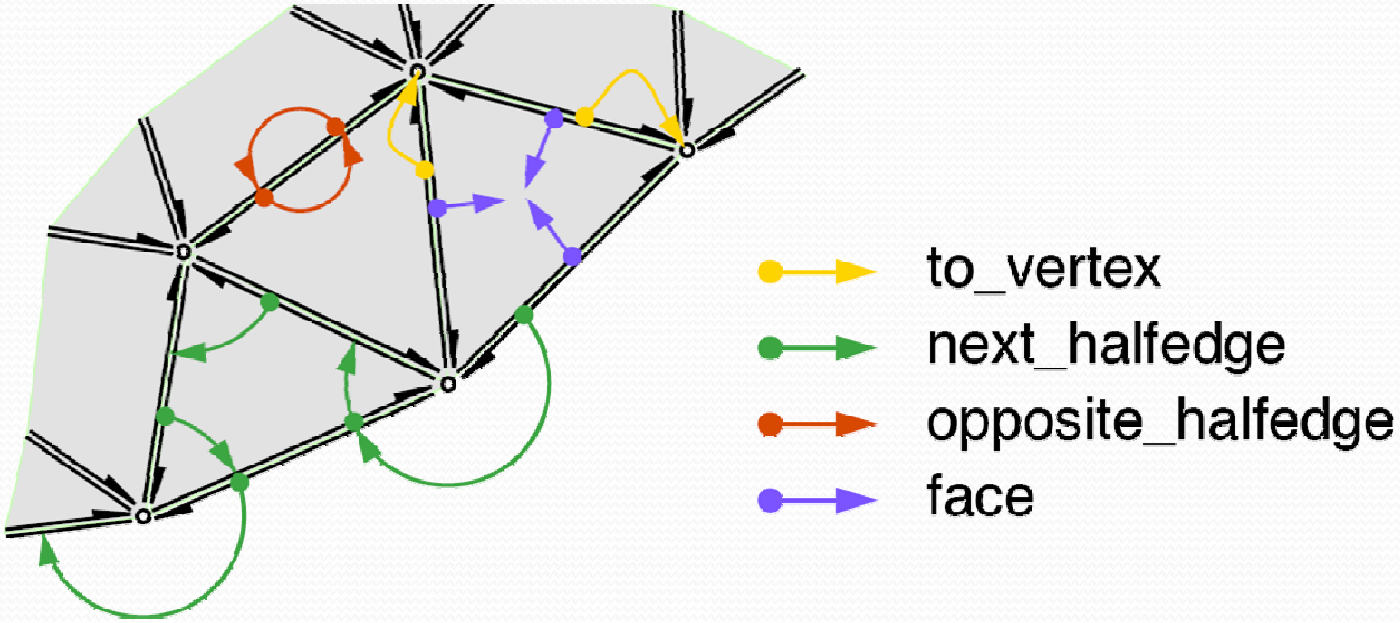
Half-edge connectivity

- Vertex:
  - position
  - 1 half-edge (outgoing)
- **Half-edge**:
  - 1 vertex (endpoint)
  - 1 face
  - 2 or 3 half-edges
    next, opposite, previous
- Face:
  - 1 half-edge



Half-edge connectivity

- **Half-edge**:
  - *to_vertex:* vertex pointed to by the half-edge
  - *next_halfedge: next half-edge* of the face (or the boundary)
  - *opposite_halfedge*
  - *Face:* face the half-edge belongs to (null if boundary)



Half-edge connectivity

Halfedge data structure

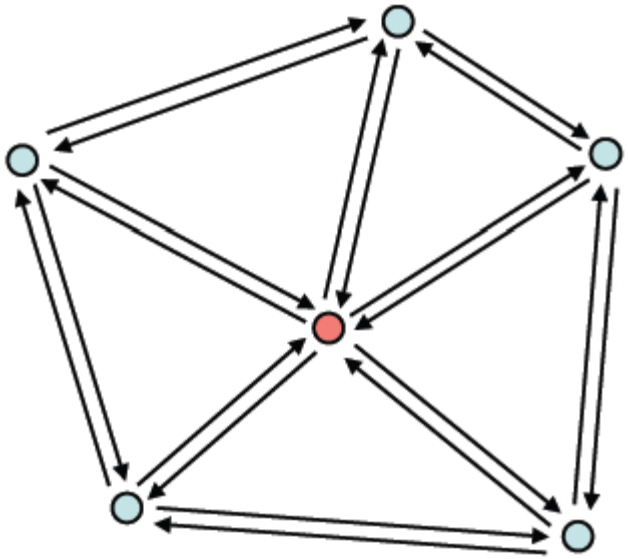
```
struct Halfedge {
  HalfedgeRef     next_halfedge;
  HalfedgeRef     opposite_halfedge;
  FaceRef         face;
  VertexRef       to_vertex;
};
```

```
struct Face {
  HalfedgeRef     halfedge;
};

struct Vertex {
  HalfedgeRef     outgoing_halfedge;
};
```

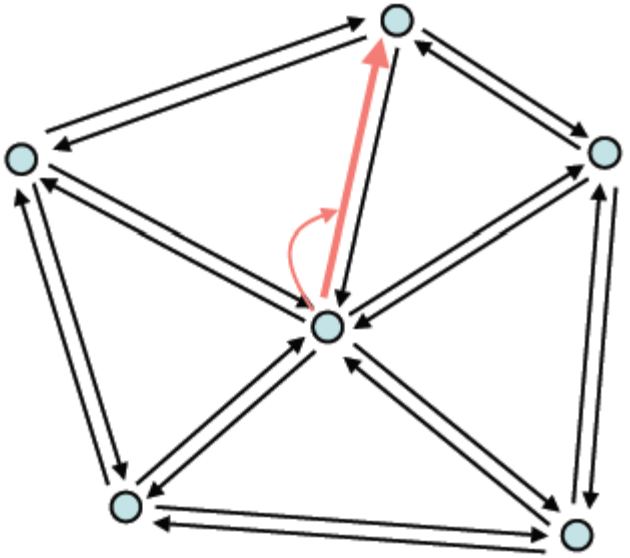
Traversing the 1-ring

1. Start at vertex



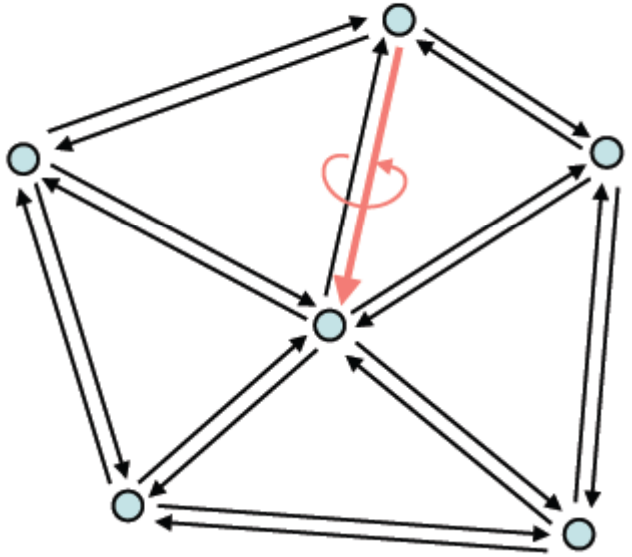
Traversing the 1-ring (CW)

1. Start at vertex
2. Outgoing halfedge



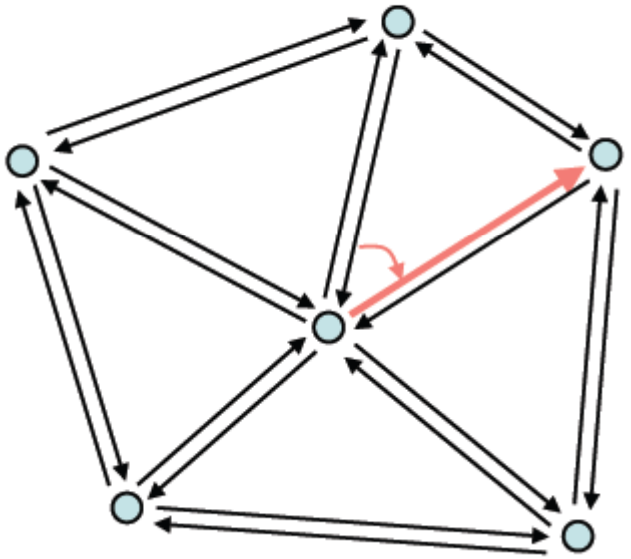
Traversing the 1-ring

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



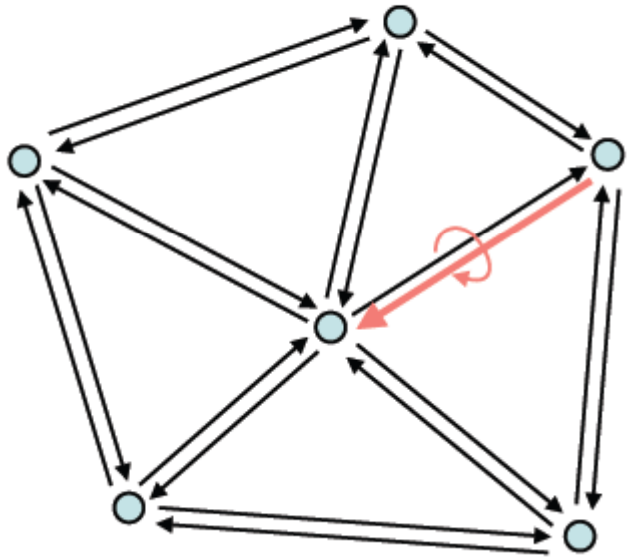
Traversing the 1-ring

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



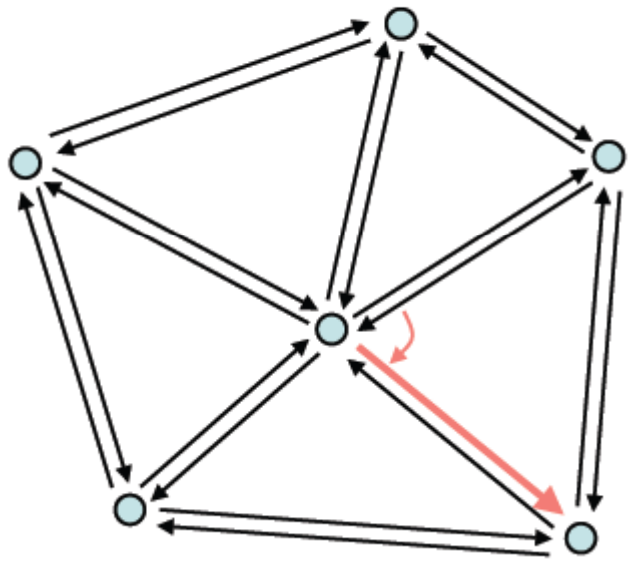
Traversing the 1-ring

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

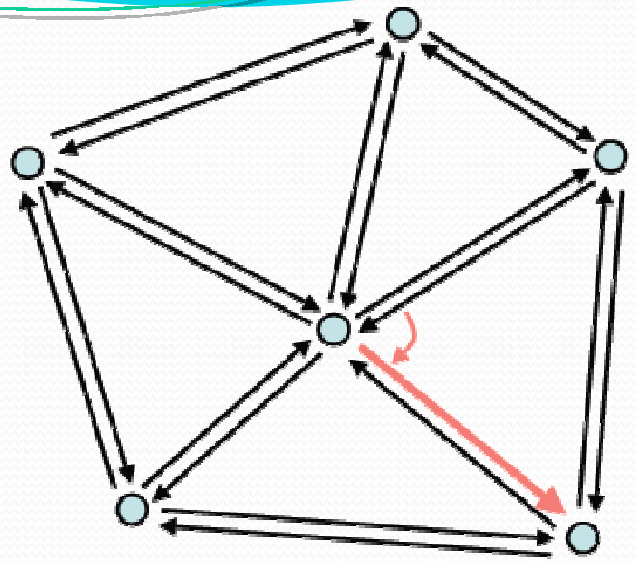


Traversing the 1-ring

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



Traversing the 1-ring



```
enumerate_1_ring(Vertex * center)
{
    HalfedgeRef h = outgoing_halfedge(center);
    HalfedgeRef hstop = h;
    do {
        VertexRef v = to_vertex(h);
        // do something with v
        h = next_halfedge(opposite_halfedge(h));
    } while ( h != hstop );
}
```

Half-edge connectivity

Implementation

- One **face array**, one **half-edge array**, and one **vertex array**
- References = indices (better than pointers)
  - Easier memory handling
  - A single index for all the attibutes of an element
- Each pair of half-edges are stored in consecutive positions in the array
  - Opposite_halfedge is implicit (we save 24 B/v → 96 B/v)
    - h **mod** 2 = 0 → opp:=h+1
      h **mod** 2 = 1 → opp:= h-1
  - Attributes can be attached to edges and half-edges

Half-edge connectivity

Summary of *half-edge data structure*:

- Suitable for any two-manifold (with/without boundary)
- Not limited to triangular faces (OK for arbitrary polygons)
- Easy to associate dynamic attributes to the different elements: per-face, per-vertex, per-edge and per-corner attributes

Mesh APIs

Important topics

- **Access**: how are elements accessed? How are neighboring elements enumerated?
- **Modification**: how can we add/delete elements? How can we guarantee mesh consistency after the updates?
- **Dynamic attribs**: How can we associate attributes to the different elements?
- **Input/output:** how can we generate the model from a vertex array + face array?

Mesh APIs

- CGAL
  - `www.cgal.org`
  - Computational geometry
  - Free for non-commercial use
- OpenMesh
  - `www.openmesh.org`
  - Mesh processing
  - Free, LGPL licence

OpenMesh - access

Access:

- All elements (faces, edges, halfedges and vertices) represented explicitly.
- Access through:
  - Handles (~ index)
  - Iterators (eg all vertices of the mesh)
  - Circular iterators (*circulators*): eg V:{F}

OpenMesh - access

Example: compute centroid of the neighbors of a vertex:

```
for ( VertexIter vi = mymesh.vertices_begin(); vi != mymesh.vertices_end(); ++vi )
{
    int cnt = 0;
    Point cog(0,0,0);
    // A VertexVertexIter is a circulator that enumerates the 1-ring of a vertex
    for ( VertexVertexIter vvi = mymesh.vv_iter(vi); vvi; ++vvi )
    {
        cnt += 1;
        cog += mymesh.point(vvi);
    }
    cog /= cnt;
    // Now cog equals the center of gravity of vi's neighbors
}
```

OpenMesh - editing

Example of mesh editing:

```
TriangleMesh mymesh;
// Add three vertices
VertexHandle v0 = mymesh.add_vertex( Point( 0, 0, 0 ) );
VertexHandle v1 = mymesh.add_vertex( Point( 0, 1, 0 ) );
VertexHandle v2 = mymesh.add_vertex( Point( 3, 0, 2 ) );
// Add the triangle
FaceHandle f = mymesh.add_face( v0, v1, v2 );
// Delete a face
mymesh.delete_face( f );
```

OpenMesh - attributes

Adding attributes at runtime:

```
// Add a property to an edge
EdgePropertyHandle<float > weight;
mymesh.add_property(weight);

// Setting a value
for ( EdgeIter e= mymesh.edges.begin(); e != mymesh.edges.end(); ++e )
    mymesh.property(weight, e) = value;

// Getting a value
for ( EdgeIter e = mymesh.edges.begin(); e != mymesh.edges.end(); ++e )
    // do something with mymesh.property( weight, e )

// Remove the property
mymesh.remove property(weight);
```

OpenMesh – input/output

```
TriangleMesh mymesh;
read_mesh( mymesh, “filename.obj" );
(...)
write_mesh( mymesh, “filename.stl" );
```

References

- S. Campagna, L. Kobbelt, H.-P. Seidel, Directed Edges - A Scalable Representation For Triangle Meshes , ACM Journal of Graphics Tools 3 (4), 1998.

- Lutz Kettner, Using Generic Programming for Designing a Data Structure for Polyhedral Surfaces, in Proc. 14th Annual ACM Symp. on Computational Geometry, 1998.

- http://openmesh.org