* Objects in graphics (and many other areas) are stored as triangles

* independent

  * easy to render
  * hard to do real processing with
  * need to know our neighbors

* Better \( \rightarrow \) store vertices

  * triangles are vertex indices

* Store
  \( \text{vector \langle \mathrm{vec}3 \rangle \text{ pts}} \)
  \( \text{vector \langle \mathrm{vec}3 \rangle \text{ tris}} \)

\[ 3n + 3v \]
\[ \approx 4.5n \]
about half the storage

* good representation for
many general uses

\( \frac{v}{n^2} \)

\( n \approx 2v \)

\( \text{six tris/vertex} \)
\( \text{three vert/tri} \)
**Manifold**

1. Every edge has **two** triangles

2. Every vertex has a full loop of triangles

**Manifold with boundary**

1. Edges have 1 or 2 triangles

2. Every vertex has one edge-connected set of tris

**Orientation**

Triangle vertices listed in consistent order (usually CCW)

*Note that the direction on each side of an edge is reversed*

\[(A \, C \, B) \neq (C \, B \, A) \neq (B \, A \, C)\]

\[\text{so } (A \, C \, B), (B \, C \, D) \text{ is ok here.}\]

\[(A \, B \, C) \, (B \, C \, D) \text{ is bad}\]

Oriented *may not be possible if not manifold or non-orientable surface (Moebius strip)*
* Strips

- Most of the benefit even for small strips
- Compress storage

\( n \) triangles \( \rightarrow \) \( n+2 \) points \( \approx n \)

(instead of \( 3n \))

\[ \rightarrow \left(1 + \frac{3}{2}\right)n \approx 2.5n \]

* Fans

- Same storage size as strip, but rather limited generally to no more than 6 tri.

* OpenGL supports both

* Mostly strip only for storage, rendering
find neighbor

<table>
<thead>
<tr>
<th>T</th>
<th>E</th>
<th>V</th>
</tr>
</thead>
<tbody>
<tr>
<td>TT</td>
<td>TE</td>
<td>TV</td>
</tr>
<tr>
<td></td>
<td>EE</td>
<td>EV</td>
</tr>
<tr>
<td>VT</td>
<td>VE</td>
<td>VV</td>
</tr>
</tbody>
</table>

* do you need edges?
* which queries do you need?
* constant time lookup

Can store these things directly:

Tri:
- vertex[3] ← index or pointer to vertex
- edge[3]

Edge:
- Vertex[2]
- Tri[2]

* array of Tri, Edge, Vertex

* overkill

Vertex:
- Tri[t]
- Edge[e]

← amazing - dynamic size
* only store adj triangles
* each vertex store arbitrary triangle

Tri:
  vertex [x] → TV
  tri [y] → IT
  vertex
  any tri

* TT in CCW order

* no edge

* TT → std
* TV → std
* VT → walk pointers

eq:

0 → C
first tri

C → 105 → HOB → D

D → 012 → ECI → E

E → 302 → JAD → A

A → 034 → BEF → B

B → 050 → AGC → C → stop (started on C)

* for VV, output next vertex in triangle while doing this traversal
Half-edge structure

half-edge:
  half-edge next, prev, pair i
  face cell j

face:
  face
  half-edge e j

vertex:
  half-edge e j

can answer all queries efficiently

e j: V T

O \rightarrow BA \rightarrow BC \rightarrow CB \rightarrow C
  \downarrow cell
  \downarrow B

CB \rightarrow CD \rightarrow DC \rightarrow DE \rightarrow ED \rightarrow EF...
  \downarrow cell
  \downarrow O

EF \rightarrow FE \rightarrow FA \rightarrow AF \rightarrow AB \rightarrow BA \rightarrow step
  \downarrow cell
  \downarrow E

* walk around a vertex
A \rightarrow AB \rightarrow AF \rightarrow AG \rightarrow \boxed{AB} \text{ step}

\downarrow \text{pair} \downarrow \text{pair} \downarrow \text{pair}

BA \rightarrow FA \rightarrow GA

\downarrow \text{cell} \downarrow \text{cell} \downarrow \text{cell}

B \rightarrow F \rightarrow G

* half-edge works well with any size polygons, does not require triangles

* Since half-edges come in pairs, put pairs together, pair index can be computed. 0, 1, 2, 3, etc.

* Can also do center splitting and merging, etc. Not too difficult to update structure, but must be very careful about ordering and filling in all pointers.