polygon meshes representation

• which representation is good?
  – often triangles/quads only – will work on triangles

• compact
• efficient for rendering
  – fast enumeration of all faces
• efficient for geometry algorithms
  – finding adjacency (what is close to what)
vertices, edges, faces

- fundamental entities
  - $n_v$ vertices
  - $n_e$ edges
  - $n_f$ faces
  - simple closed surface: $n_v - n_e + n_f = 2$

- fundamental properties:
  - topology: how faces are connected
  - geometry: where faces are in space
  - separate issues
    - algorithms mostly care about topology

topology vs. geometry

- same geometry
different topology

- same topology
different geometry
triangles

- array of vertex data
  - vertex[n][3]
  - vertex stores position and optional data (normal, uvs)
  - ~72 bytes per triangle with vertex position only

- redundant
- adjacency is not well defined
  - floating point errors in comparing vertices
indexed triangles

- array of vertex data
  - vertex[\(n_v\)]
  - 12 bytes per vertex with position only
- array of vertex indices (3 per triangle)
  - int[\(n_t\)][3], often flattened in a single array
  - 24 bytes per triangle
- total storage: \(\sim 36\) bytes (50% memory)

- topology/geometry stored separately/explicitly
  - adjacency queries are well defined
triangles strips

• since triangle share edges, let every triangle reuse the last one’s vertices

| vertex[0] | (x0,y0,z0) |
| vertex[1] | (x1,y1,z1) |
| ... | ... |
| strip[0] | 0,1,2,3,4,5 |
| strip[1] | ... |
| ... | ... |

triangles strips

• requires multiple strips for general case

| vertex[0] | (x0,y0,z0) |
| vertex[1] | (x1,y1,z1) |
| ... | ... |
| strip[0] | 0,1,2,3,4,5 |
| strip[1] | ... |
| ... | ... |
**triangle strips**

- array of vertex data
  - `vertex[n_v]`
  - 12 bytes per vertex with position only
- array of lists of vertex indices
  - `int[n_f][varyingLength]`
- for long lists saves about 1/3 index memory

**triangle fans**

- idea similar to triangle strips
- different arrangement

![Diagram of triangle fans](image)
quad meshes

• similar options as for storing triangles
  – flat quads
  – indexed quad meshes
  – quad strips, no fans

adjacency queries

• example queries
  – given a face, find adjacent faces
  – given an edge, find faces that share it
  – given a vertex, find faces that share it

• previous data structures
  – inefficient adjacency queries, $O(n)$
adjacency lists

- store all vertex, edge, face adjacency
  - efficient adjacency queries, $O(1)$
  - extra storage

[diagram]

Based on Finkelstein 2004

partial adjacency lists

- store some vertex, edge, face adjacency
  - goal: efficient adjacency queries
  - goal: less storage

[diagram]

Based on Finkelstein 2004
winged edge

• adjacency stored in edges
  – all adjacency in O(1)
  – little extra storage

\[ \begin{align*}
&\{E_k\} \\
&\{V_i\} \quad \{F_j\}
\end{align*} \]

[based on Finkelstein 2004]

[Shirley]

winged edge

• tetrahedron example

\[ \begin{align*}
\text{vertex} & \quad \text{edge} \\
A & \quad a \\
B & \quad d \\
C & \quad d \\
D & \quad a
\end{align*} \]

\[ \begin{align*}
\text{face} & \quad \text{edge} \\
0 & \quad a \\
1 & \quad c \\
2 & \quad d \\
3 & \quad a
\end{align*} \]

\[ \begin{align*}
\text{edge} & \quad \text{vertex 1} \quad \text{vertex 2} \quad \text{face left} \quad \text{face right} \quad \text{pred left} \quad \text{pred right} \quad \text{succ left} \quad \text{succ right} \\
2 & \quad A \quad D \quad 3 \quad 0 \quad f \quad o \quad c \quad b \\
1 & \quad A \quad B \quad 0 \quad 2 \quad a \quad c \quad d \quad f \\
0 & \quad B \quad D \quad 0 \quad 1 \quad b \quad a \quad e \quad d \\
1 & \quad B \quad C \quad 1 \quad 2 \quad c \quad o \quad f \quad b \\
0 & \quad C \quad D \quad 1 \quad 3 \quad d \quad c \quad a \quad f \\
3 & \quad C \quad A \quad 3 \quad 2 \quad e \quad a \quad b \quad d
\end{align*} \]

[Shirley]
defining normals

• face normal
  – geometrically correct
  – not for smooth surfaces

• vertex normals
  – geometrically "inconsistent"
  – for smooth surface approx.

---

defining normals

• face normal
  – geometrically correct
  – not for smooth surfaces

• vertex normals
  – geometrically "inconsistent"
  – for smooth surface approx.