polygon meshes

polygon meshes representation

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

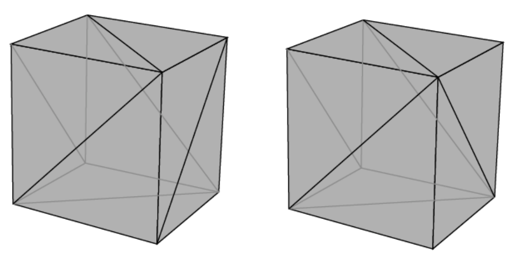
- compact
- efficient for rendering
  - fast enumeration of all faces
- efficient for geometry algorithms
  - finding adjacency (what is close to what)

vertices, edges, faces

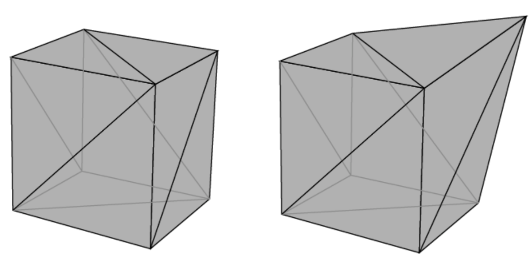
- fundamental entities
  - n_v vertices
  - n_e edges
  - n_f faces
  - simple closed surface: $n_v - n_e + n_f = 2$
- fundamental properties:
  - topology: how faces are connected
  - geometry: where faces are in space
  - separate issues
    - algorithms mostly care about topology

topology vs. geometry

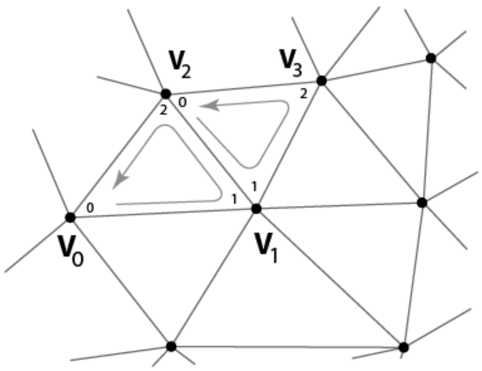
- same geometry
  different topology



- same topology
  different geometry



triangles



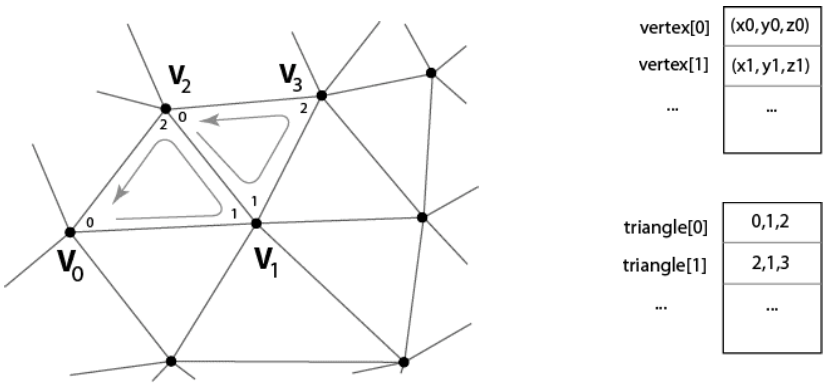
triangle[0]	(x0, y0, z0)	(x1, y1, z1)	(x2, y2, z2)
triangle[1]	(x1, y1, z1)	(x2, y2, z2)	(x3, y3, z3)
...	...	...	...

triangles

- array of vertex data
  - vertex[n_f][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



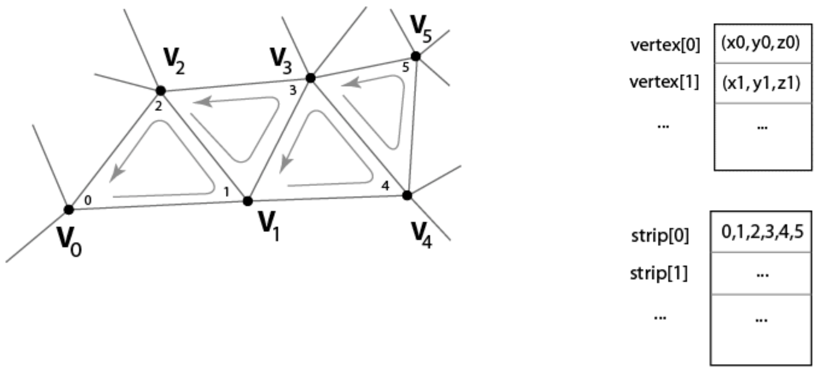
indexed triangles

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

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

triangles strips

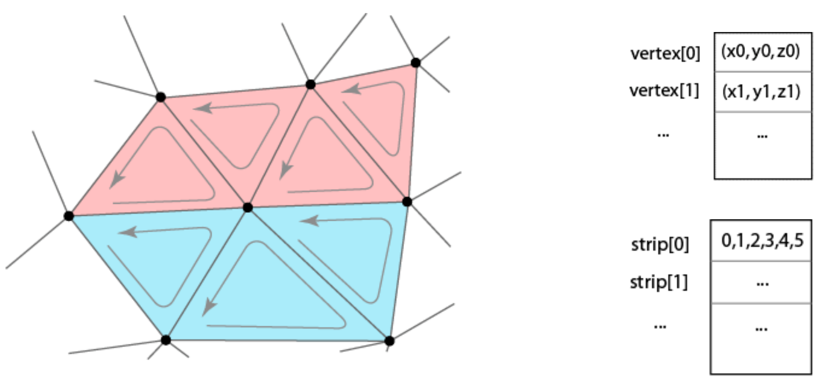
- since triangle share edges, let every triangle reuse the last one's vertices



 © 2008 fabio pellacini • 9

triangles strips

- requires multiple strips for general case



 © 2008 fabio pellacini • 10

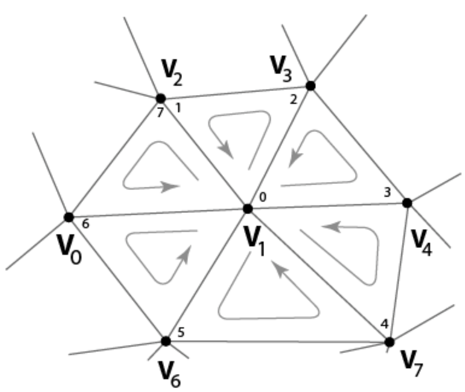
triangle strips

- 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

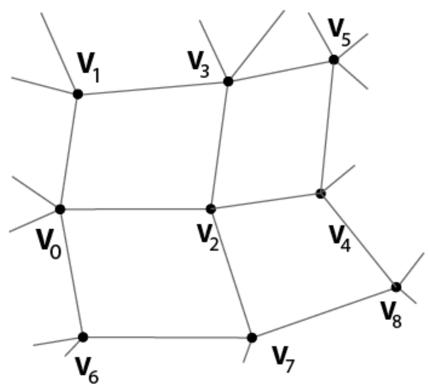


vertex[0]	(x0, y0, z0)
vertex[1]	(x1, y1, z1)
...	...

fan[0]	1,2,2,3,4,7,6,0,2
fan[1]	...
...	...

quad meshes

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



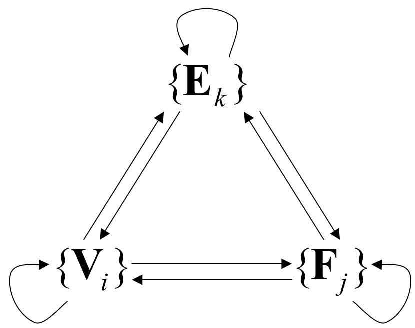
adjacency queries

- 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

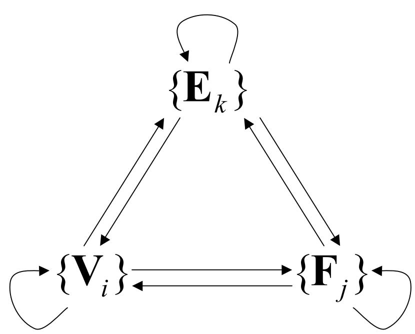
- store all vertex, edge, face adjacency
  - efficient adjacency queries, O(1)
  - extra storage



[based on Finkelstein 2004]

partial adjacency lists

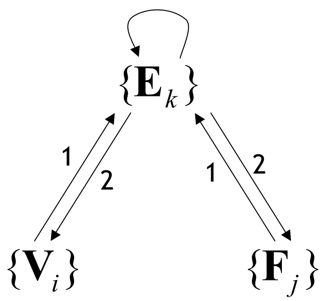
- store some vertex, edge, face adjacency
  - goal: efficient adjacency queries
  - goal: less storage



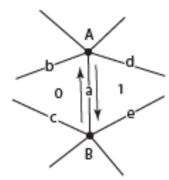
[based on Finkelstein 2004]

winged edge

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



[based on Finkelstein 2004]



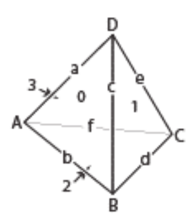
edge	vertex 1	vertex 2	face left	face right	pred left	succ left	pred right	succ right
a	B	A	0	1	c	b	d	e

[Shirley]

© 2008 fabio pellacini

winged edge

- tetrahedron example



vertex	edge
A	a
B	d
C	d
D	e

face	edge
0	a
1	c
2	d
3	a

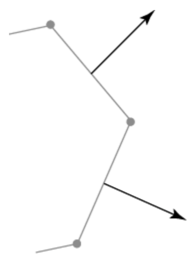
edge	vertex 1	vertex 2	face left	face right	pred left	succ left	pred right	succ right
a	A	D	3	0	f	e	c	b
b	A	B	0	2	a	c	d	f
c	B	D	0	1	b	a	e	d
d	B	C	1	2	c	e	f	b
e	C	D	1	3	d	c	a	f
f	C	A	3	2	e	a	b	d

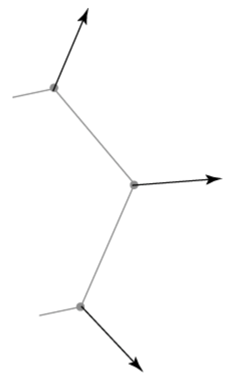
[Shirley]

© 2008 fabio pellacini

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.