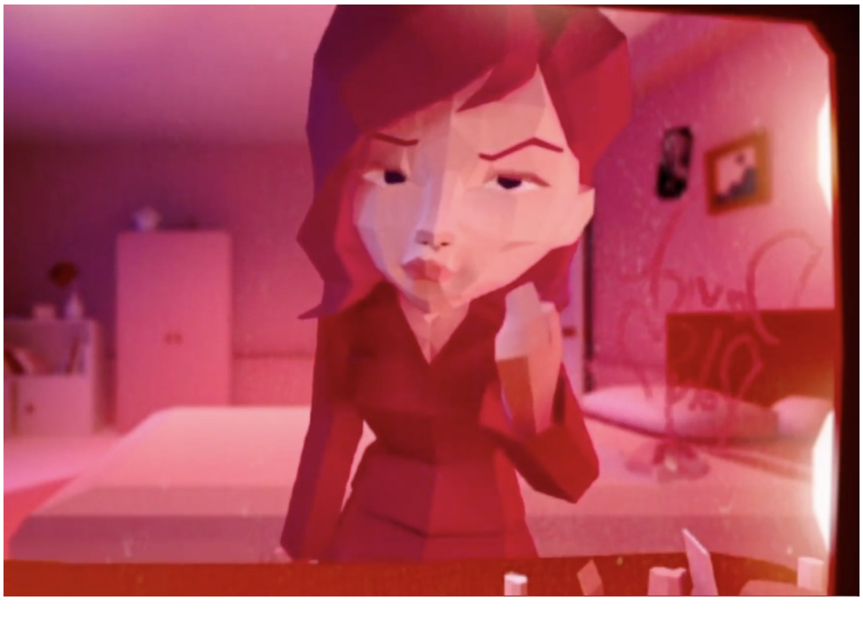
Adjacency Data Structures

includes material from Justin Legakis

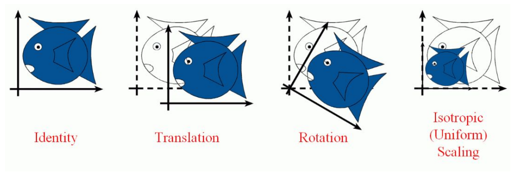
Cubic Tragedy

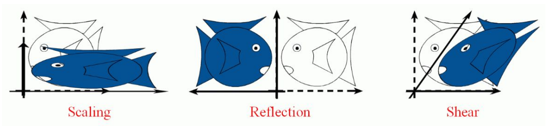


Ming-Yuan Chuan & Chun-Wang Sun, SIGGRAPH 2005

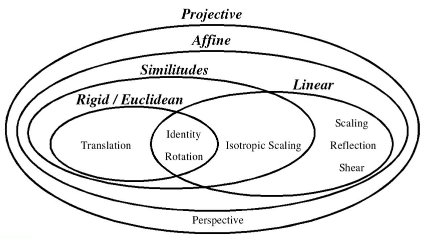
Last Time?

- Simple Transformations

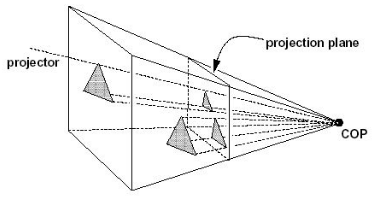




- Classes of Transformations
- Representation
  - homogeneous coordinates
- Composition
  - not commutative
- Orthographic & Perspective Projections

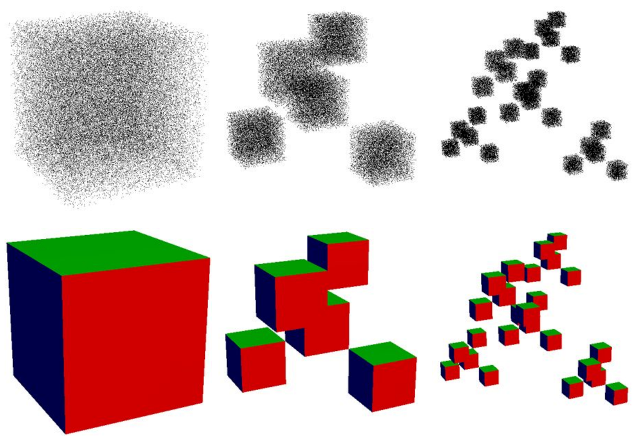


$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$



Homework 0: OpenGL Warmup

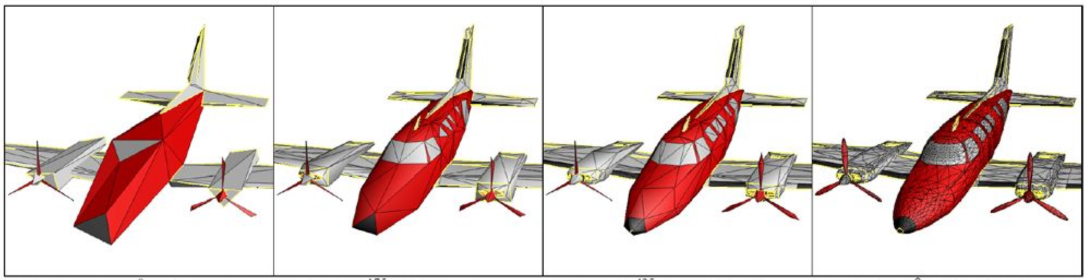
- Get familiar with:
  - C++ environment
  - OpenGL/Metal
  - Transformations
  - simple Vector & Matrix classes
  - CMake
- Have Fun!
- Due ASAP…



Today

- Reading: “Progressive Meshes”
- Surface Definitions
- Simple Data Structures
- Fixed Storage Data Structures
- Fixed Computation Data Structures
- Reading for Tuesday & HW1

Progressive Meshes


(a) Base mesh M^0 (150 faces) (b) Mesh M^{175} (500 faces) (c) Mesh M^{425} (1,000 faces) (d) Original $\hat{M}=M^n$ (13,546 faces)

Hugues Hoppe, SIGGRAPH 1996

Progressive Meshes

- Mesh Simplification
  - vertex split / edge collapse
  - geometry & discrete/scalar attributes
  - priority queue
- Level of Detail
  - geomorphs
- Progressive Transmission
- Mesh Compression
- Selective Refinement
  - view dependent

Selective Refinement

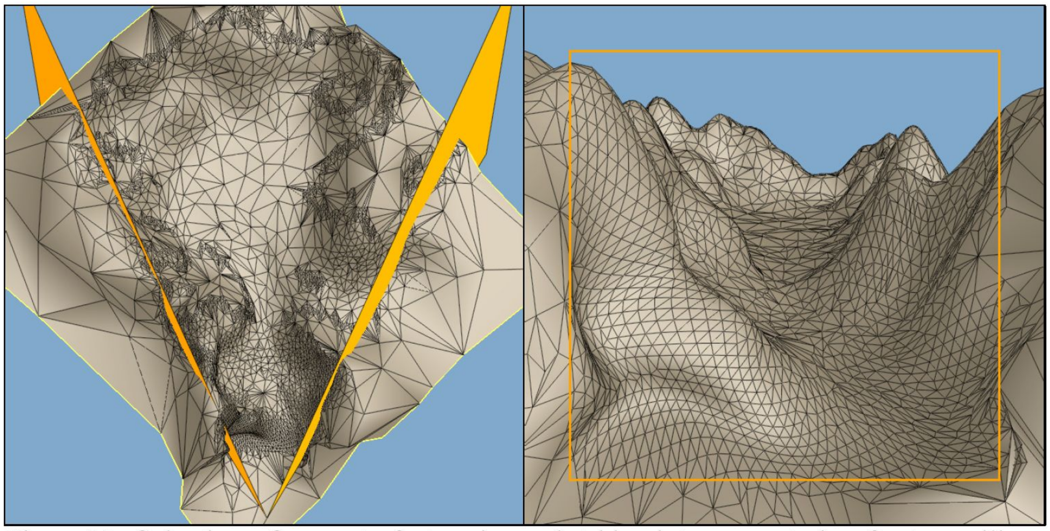
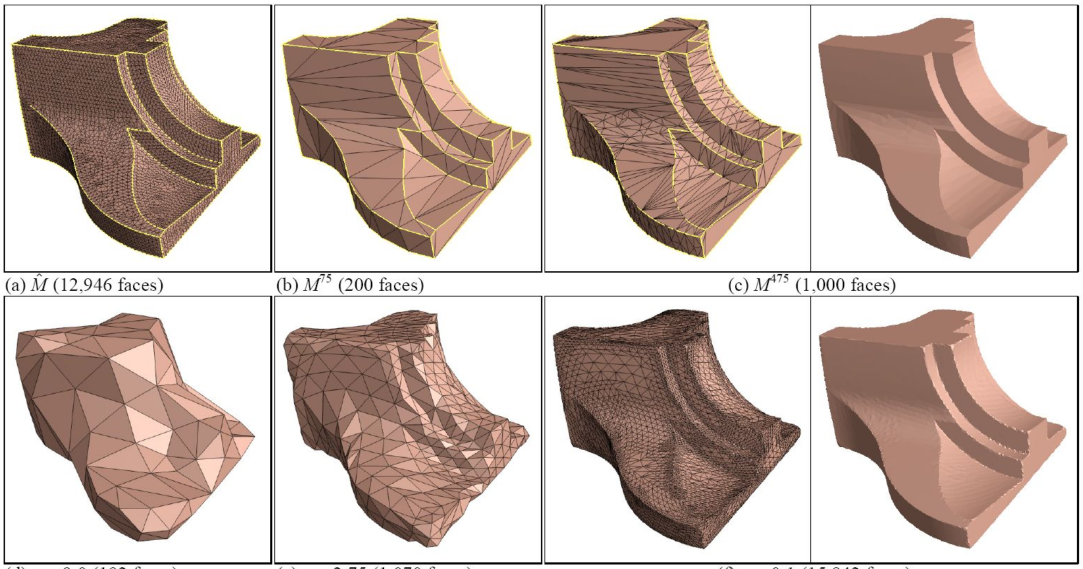


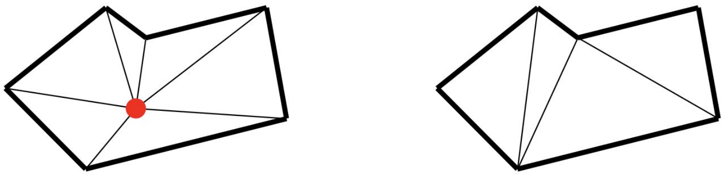
Figure 10: Selective refinement of a terrain mesh taking into account view frustum, silhouette regions, and projected screen size of faces (7,438 faces).

Preserving Discontinuity Curves


(a) $\hat{M}$ (12,946 faces) (b) M^{75} (200 faces) (c) M^{475} (1,000 faces)
(d) $\epsilon = 9.0$ (192 faces) (e) $\epsilon = 2.75$ (1,070 faces) (f) $\epsilon = 0.1$ (15,842 faces)
Figure 12: Approximations of a mesh $\hat{M}$ using (b–c) the PM representation, and (d–f) the MRA scheme of Eck et al. [7]. As demonstrated, MRA cannot recover $\hat{M}$ exactly, cannot deal effectively with surface creases, and produces approximating meshes of inferior quality.

Other Simplification Strategies

- Remove a vertex & surrounding triangles, re-triangulate the hole



- Merge Nearby Vertices
  - will likely change the topology…

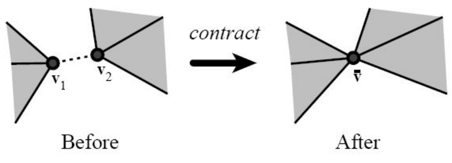


Figure 2: **Non-edge contraction.** When non-edge pairs are contracted, unconnected sections of the model are joined. The dashed line indicates the two vertices being contracted together.

from Garland & Heckbert, “Surface Simplification Using Quadric Error Metrics” SIGGRAPH 1997

When to Preserve Topology?

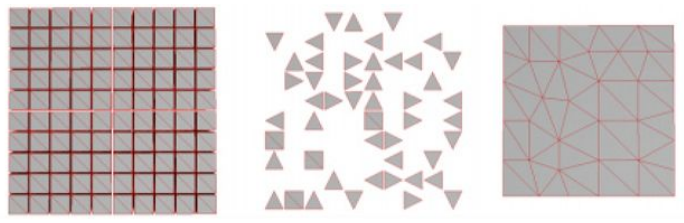


Figure 3: On the left is a regular grid of 100 closely spaced cubes. In the middle, an approximation built using only edge contractions demonstrates unacceptable fragmentation. On the right, the result of using more general pair contractions to achieve aggregation is an approximation much closer to the original.

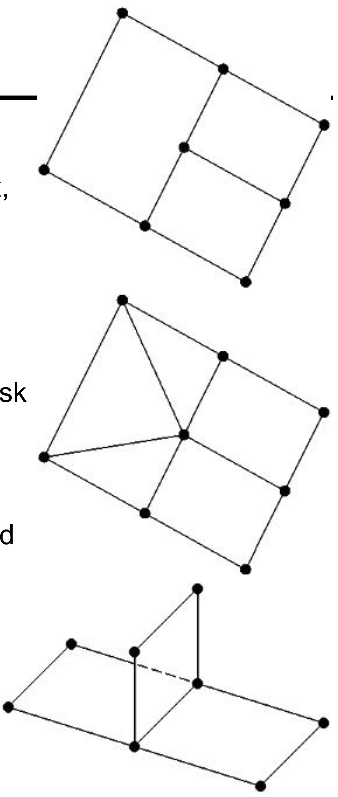
from Garland & Heckbert, "Surface Simplification Using Quadric Error Metrics" SIGGRAPH 1997

Today

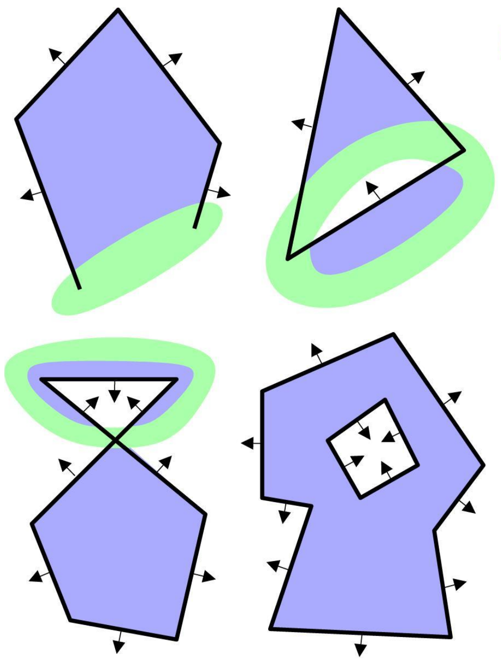
- Reading: "Progressive Meshes"
- Surface Definitions
  - Well-Formed Surfaces
  - Orientable Surfaces
  - Computational Complexity
- Simple Data Structures
- Fixed Storage Data Structures
- Fixed Computation Data Structures
- Reading for Tuesday & HW1

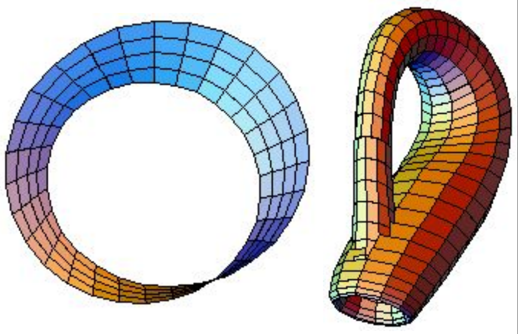
Well-Formed Surfaces

- Components Intersect "Properly"
  - Any pair of Faces are: disjoint, share single Vertex, *or* share 2 Vertices and the Edge joining them
  - Every edge is incident to exactly 2 vertices
  - Every edge is incident to exactly 2 faces
- Local Topology is "Proper"
  - Neighborhood of a vertex is *homeomorphic* to a disk (permits stretching and bending, but not tearing)
  - Also called a 2-manifold
  - If boundaries are allowed, points on the boundary are homeomorphic to a half-disk, called a "manifold with boundaries"
- Global Topology is "Proper"
  - Connected, Closed, & Bounded

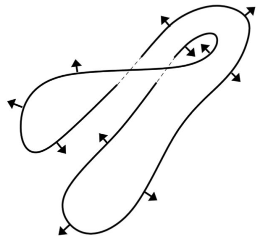


Orientable Surfaces?





from mathworld.wolfram.com

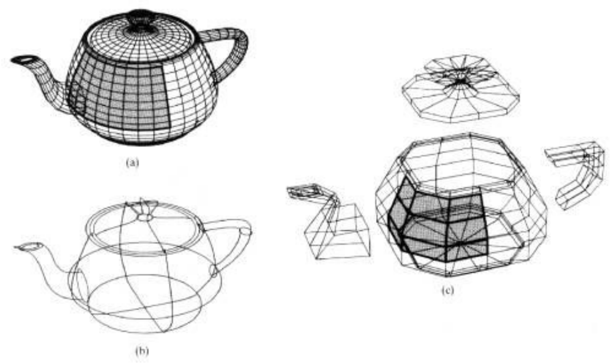


Closed Surfaces and Refraction

- Original Teapot model is not "watertight":

  intersecting surfaces at spout & handle, no bottom,
  a hole at the spout tip, a gap between lid & base

- Requires repair before ray tracing with refraction



Henrik Wann Jensen

Computational Complexity

- Adjacent Element Access Time
  - linear, constant time average case, or constant time?
  - requires loops/recursion/if ?
- Memory
  - variable size arrays or constant size?
- Maintenance
  - ease of editing
  - ensuring consistency

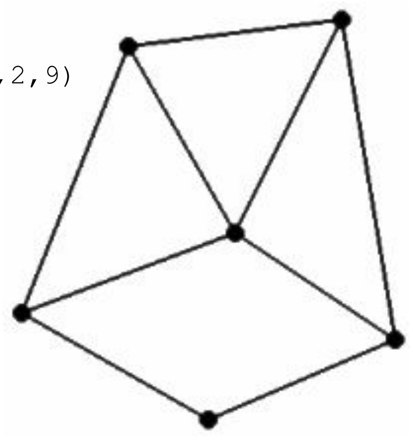
Questions?

Today

- Reading: “Progressive Meshes”
- Surface Definitions
- Simple Data Structures
  - List of Polygons
  - List of Edges
  - List of Unique Vertices & Indexed Faces:
  - Simple Adjacency Data Structure
- Fixed Storage Data Structures
- Fixed Computation Data Structures
- Reading for Tuesday & HW1

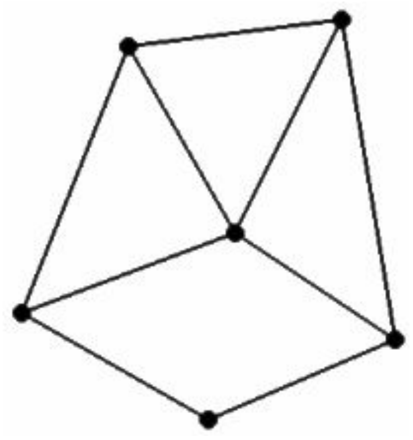
List of Polygons:

```
(3,-2,5), (3,6,2), (-6,2,4)
(2,2,4), (0,-1,-2), (9,4,0), (4,2,9)
(1,2,-2), (8,8,7), (-4,-5,1)
(-8,2,7), (-2,3,9), (1,2,-7)
```



List of Edges:

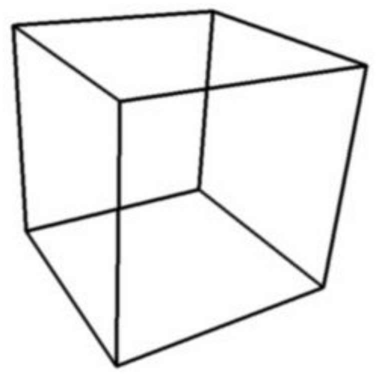
```
(3,6,2), (-6,2,4)
(2,2,4), (0,-1,-2)
(9,4,0), (4,2,9)
(8,8,7), (-4,-5,1)
(-8,2,7), (1,2,-7)
(3,0,-3), (-7,4,-3)
(9,4,0), (4,2,9)
(3,6,2), (-6,2,4)
(-3,0,-4), (7,-3,-4)
```



List of Unique Vertices & Indexed Faces:

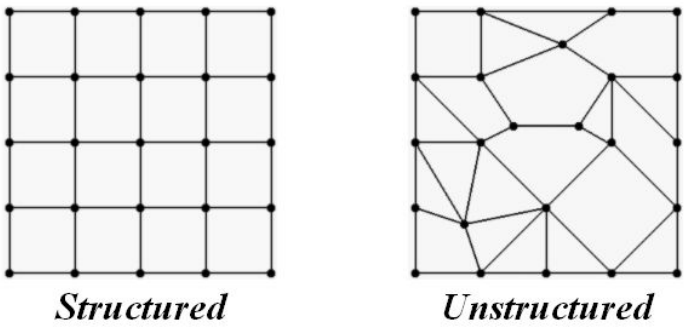
Vertices:
```
(-1, -1, -1)
(-1, -1, 1)
(-1, 1, -1)
(-1, 1, 1)
(1, -1, -1)
(1, -1, 1)
(1, 1, -1)
(1, 1, 1)
```

Faces:
```
1 2 4 3
5 7 8 6
1 5 6 2
3 4 8 7
1 3 7 5
2 6 8 4
```



Problems with Simple Data Structures

- No Adjacency Information
- Linear-time Searches



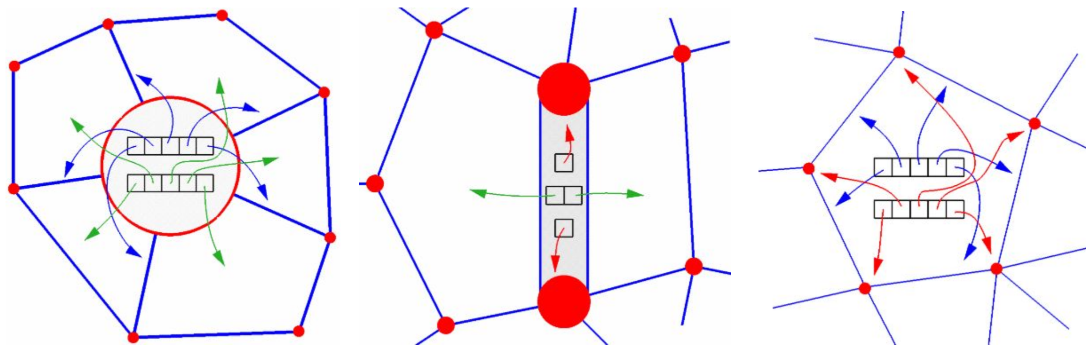
- Adjacency is implicit for structured meshes, but what do we do for unstructured meshes?

Mesh Data

- So, in addition to:
  - Geometric Information (position)
  - Attribute Information (color, texture, temperature, population density, etc.)
- Let's store:
  - Topological Information (adjacency, connectivity)

Simple Adjacency

- Each element (vertex, edge, and face) has a list of pointers to all incident elements
- Queries depend only on local complexity of mesh
- Data structures do not have fixed size
- Slow! Big! Too much work to maintain!



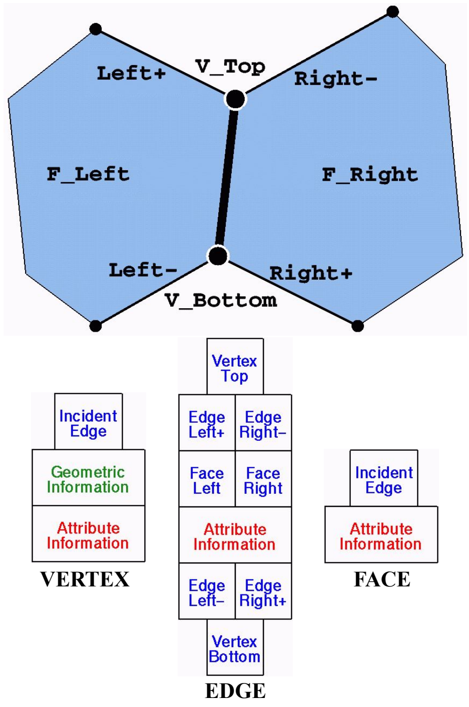
Questions?

Today

- Reading: “Progressive Meshes”
- Surface Definitions
- Simple Data Structures
- Fixed Storage Data Structures
  - Winged Edge (Baumgart, 1975)
- Fixed Computation Data Structures
- Reading for Tuesday & HW1

Winged Edge (Baumgart, 1975)

- Each edge stores pointers to 4 Adjacent Edges, 2 Face & 2 Vertex neighbors
- Vertices and Faces have a single pointer to one incident Edge
- Data Structure Size? Fixed
- How do we gather all faces surrounding one vertex? Messy, because there is no consistent way to order pointers

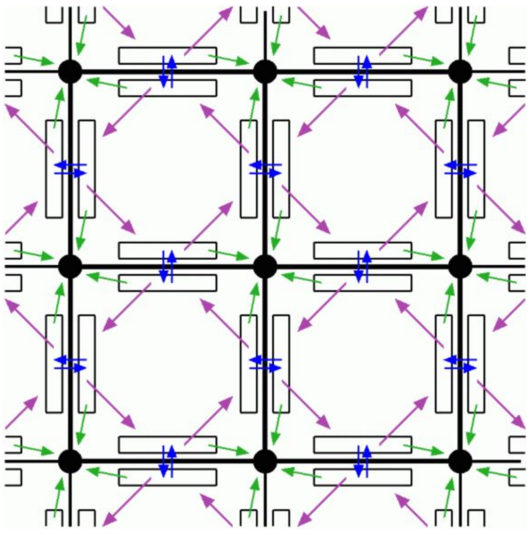


Today

- Reading: “Progressive Meshes”
- Surface Definitions
- Simple Data Structures
- Fixed Storage Data Structures
- Fixed Computation Data Structure
  - HalfEdge (Eastman, 1982)
  - SplitEdge
  - Corner
  - QuadEdge (Guibas and Stolfi, 1985)
  - FacetEdge (Dobkin and Laszlo, 1987)
- Reading for Tuesday & HW1

HalfEdge (Eastman, 1982)

- Every edge is represented by two directed HalfEdge structures
- Each HalfEdge stores:
  - **vertex** at end of directed edge
  - **symmetric** half edge
  - **face** to left of edge
  - **next** points to the HalfEdge counter-clockwise around face on left
- Orientation is essential, but can be done consistently!



HalfEdge (Eastman, 1982)

- Starting at a half edge, how do we find:

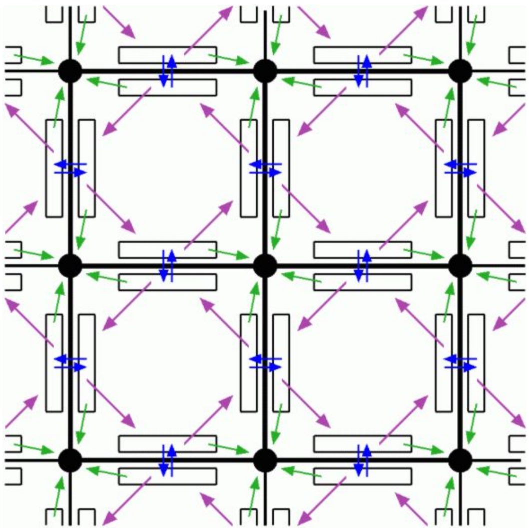
  the other vertex of the edge?

  the other face of the edge?

  the clockwise edge around the face at the left?

  all the edges surrounding the face at the left?

  all the faces surrounding the vertex?



HalfEdge (Eastman, 1982)

- Loop around a Face:

```
HalfEdgeMesh::FaceLoop(HalfEdge *HE) {
   HalfEdge *loop = HE;
   do {
      loop = loop->Next;
   } while (loop != HE);
}
```

- Loop around a Vertex:

```
HalfEdgeMesh::VertexLoop(HalfEdge *HE) {
   HalfEdge *loop = HE;
   do {
      loop = loop->Next->Sym;
   } while (loop != HE);
}
```

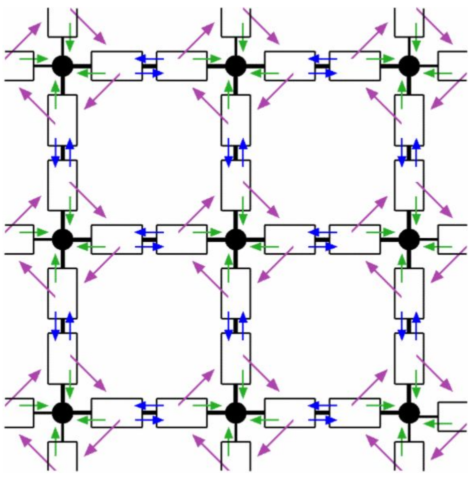
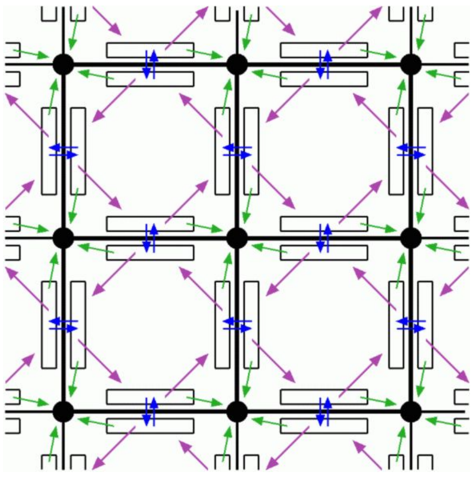
HalfEdge (Eastman, 1982)

- Data Structure Size?

  Fixed
- Data:
  - geometric information stored at Vertices
  - attribute information in Vertices, HalfEdges, and/or Faces
  - topological information in HalfEdges only!
- Orientable surfaces only (no Mobius Strips!)
- Local consistency everywhere implies global consistency
- Time Complexity?

  linear in the amount of information gathered

SplitEdge Data Structure:

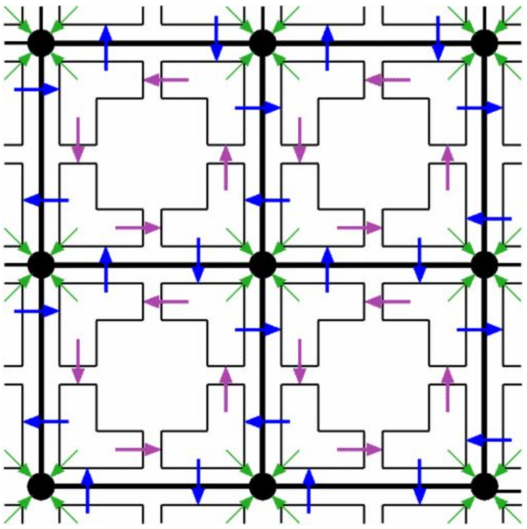
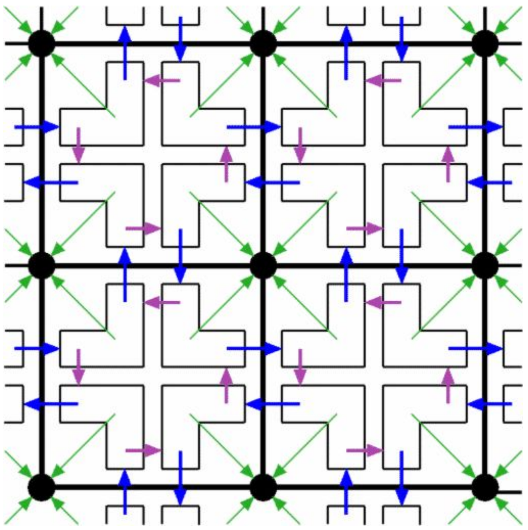



- HalfEdge and SplitEdge are dual structures!

```
SplitEdgeMesh::FaceLoop() = HalfEdgeMesh::VertexLoop()
SplitEdgeMesh::VertexLoop() = HalfEdgeMesh::FaceLoop()
```

Corner Data Structure:

- The Corner data structure is its own dual!

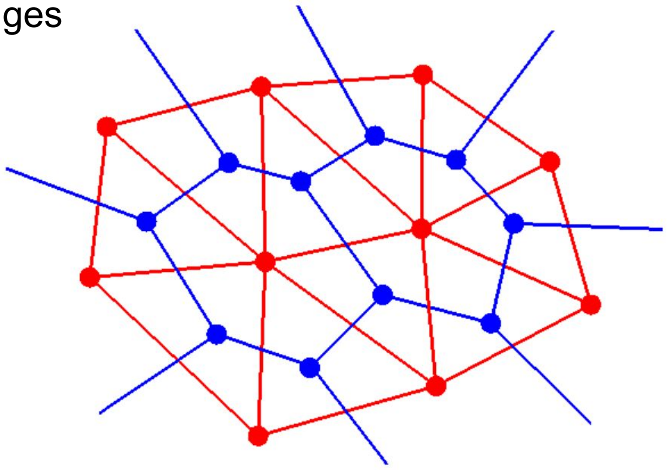
Questions?

Today

- Reading: “Progressive Meshes”
- Surface Definitions
- Simple Data Structures
- Fixed Storage Data Structures
- Fixed Computation Data Structure
  - HalfEdge (Eastman, 1982)
  - SplitEdge
  - Corner
  - QuadEdge (Guibas and Stolfi, 1985)
  - FacetEdge (Dobkin and Laszlo, 1987)
- Reading for Tuesday & HW1

QuadEdge (Guibas and Stolfi, 1985)

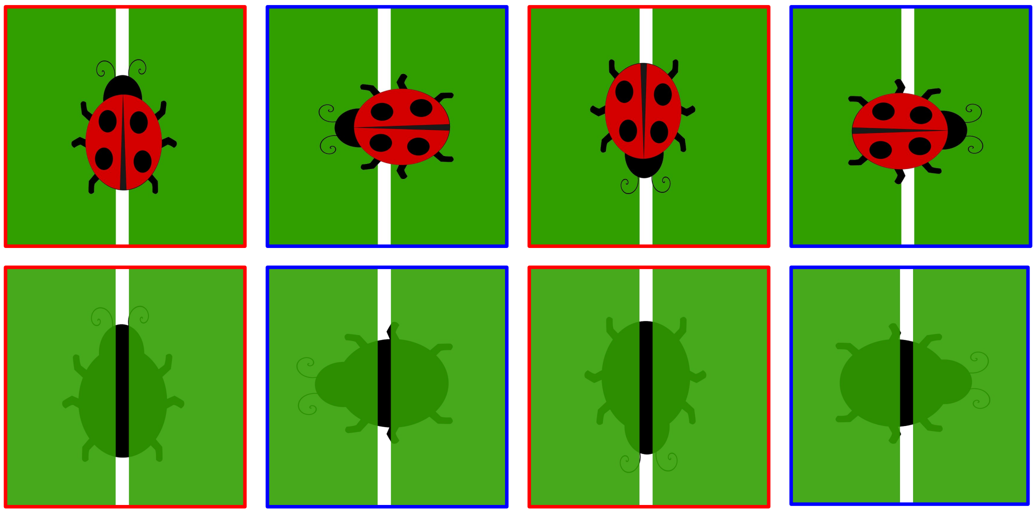
- Consider the Mesh and its *Dual* simultaneously
  - Vertices and Faces switch roles, we just re-label them
  - Edges remain Edges
- Classic dual mesh example:
  - Delaunay triangulation*
  - Voronoi diagram*

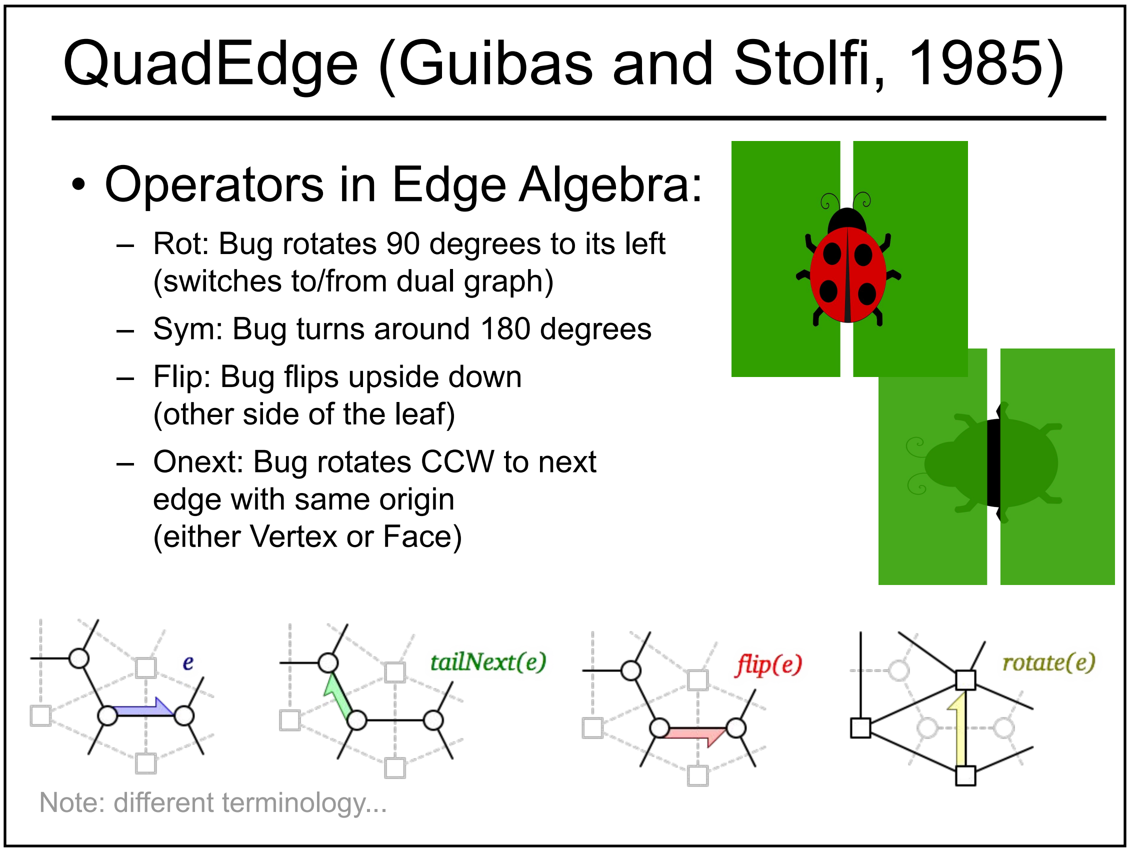


* has other special properties

QuadEdge (Guibas and Stolfi, 1985)

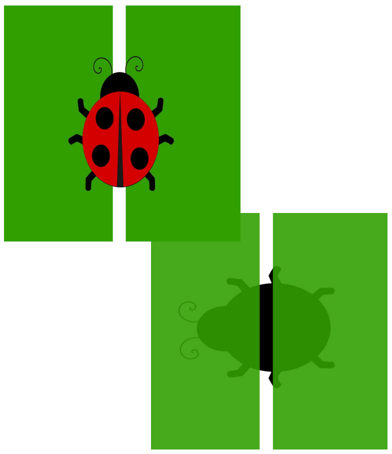
- Eight ways to look at each edge
  - Four ways to look at primal edge
  - Four ways to look at dual edge

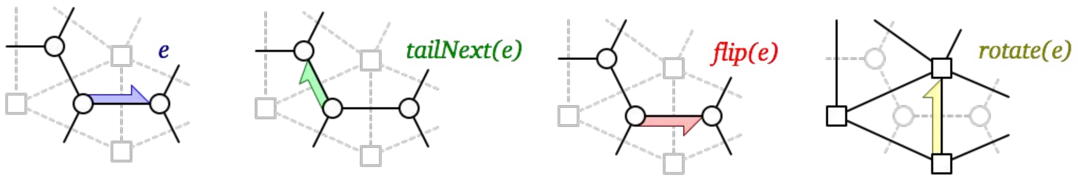




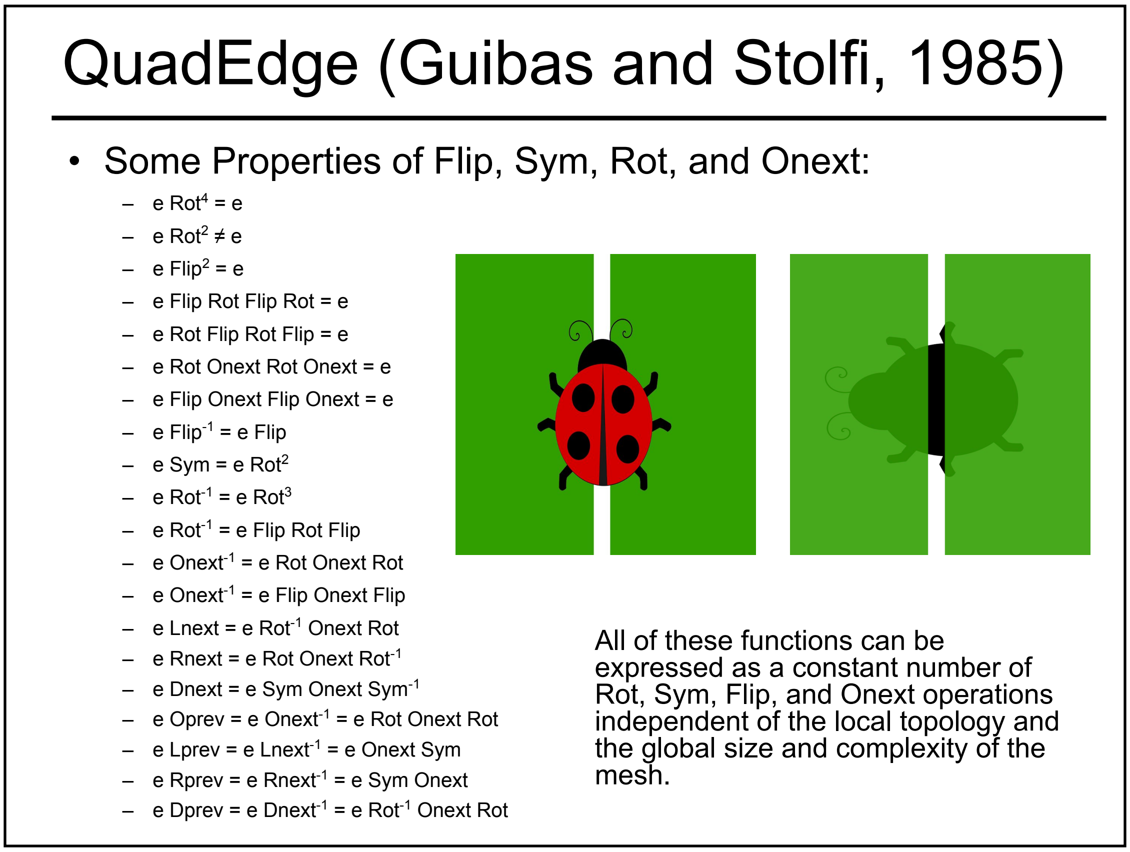
QuadEdge (Guibas and Stolfi, 1985)

- Operators in Edge Algebra:
  - Rot: Bug rotates 90 degrees to its left (switches to/from dual graph)
  - Sym: Bug turns around 180 degrees
  - Flip: Bug flips upside down (other side of the leaf)
  - Onext: Bug rotates CCW to next edge with same origin (either Vertex or Face)



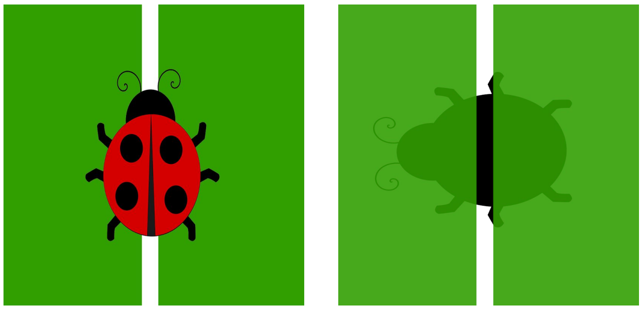


Note: different terminology...



QuadEdge (Guibas and Stolfi, 1985)

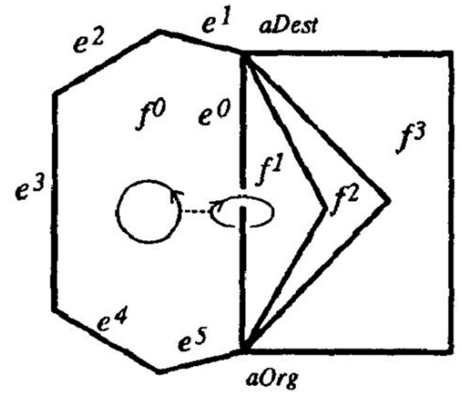
- Some Properties of Flip, Sym, Rot, and Onext:
  - e Rot^4 = e
  - e Rot^2 ≠ e
  - e Flip^2 = e
  - e Flip Rot Flip Rot = e
  - e Rot Flip Rot Flip = e
  - e Rot Onext Rot Onext = e
  - e Flip Onext Flip Onext = e
  - e Flip^{-1} = e Flip
  - e Sym = e Rot^2
  - e Rot^{-1} = e Rot^3
  - e Rot^{-1} = e Flip Rot Flip
  - e Onext^{-1} = e Rot Onext Rot
  - e Onext^{-1} = e Flip Onext Flip
  - e Lnext = e Rot^{-1} Onext Rot
  - e Rnext = e Rot Onext Rot^{-1}
  - e Dnext = e Sym Onext Sym^{-1}
  - e Oprev = e Onext^{-1} = e Rot Onext Rot
  - e Lprev = e Lnext^{-1} = e Onext Sym
  - e Rprev = e Rnext^{-1} = e Sym Onext
  - e Dprev = e Dnext^{-1} = e Rot^{-1} Onext Rot

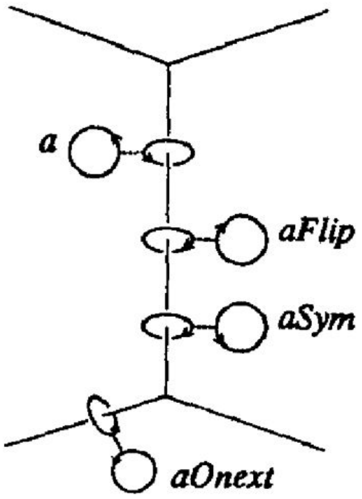


All of these functions can be expressed as a constant number of Rot, Sym, Flip, and Onext operations independent of the local topology and the global size and complexity of the mesh.

FacetEdge (Dobkin and Laszlo, 1987)

- QuadEdge (2D, surface) → FacetEdge (3D, volume)
- Faces → Polyhedra / Cells
- Edge → Polygon & Edge pair





Questions?

Today

- Reading: “Progressive Meshes”
- Surface Definitions
- Simple Data Structures
- Fixed Storage Data Structures
- Fixed Computation Data Structures
- Reading for Tuesday & HW1

Reading for Tuesday

Four new volunteers to be “Discussants”?

- "Teddy: A Sketching Interface for 3D Freeform Design", Igarashi et al., SIGGRAPH 1999

How do we represent objects that don’t have flat polygonal faces & sharp corners? What are the right tools to design/construct digital models of blobby, round, or soft things? What makes a user interface intuitive, quick, and easy-to-use for beginners?

Homework 1 - *will be posted soon!*