Triangle Meshes

CS451

Prof. Jarek Rossignac

College of Computing

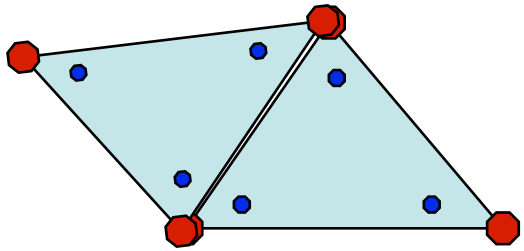
Georgia Institute of Technology

Lecture Objectives

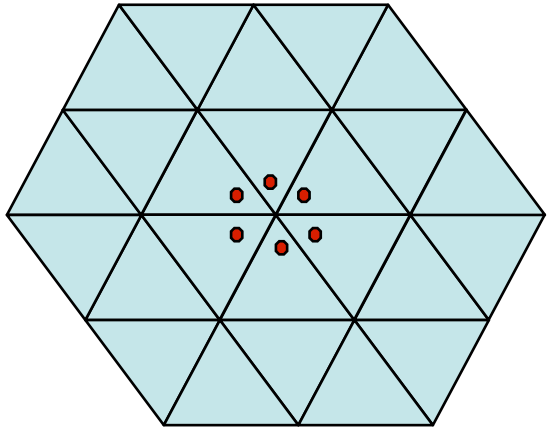
- Learn how to **triangulate** an unstructured set of points in 3D
- Learn the **terminology**: Incidence, orientation, corner…
- Learn how to represent a simple triangle mesh using a **Corner Table** data structure
- Learn how to **build** a **Corner Table** from a Face/Vertex index file
- Learn how to implement and use the primary **operators** for traversing the mesh
- Learn how to **traverse** the mesh to estimate **surface normals** at vertices and to identify the shells
- Learn the formula for computing the **genus** of each shell
- Understand the **topological limitations** of the Corner Table and how to use it for representing meshes with holes

Representation as **independent** triangles

- For each triangle:
  - Store the location of its 3 vertices



	vertex 1			vertex 2			vertex 3	
Triangle 1	x	y	z	x	y	z	x	y	z
Triangle 2	x	y	z	x	y	z	x	y	z
Triangle 3	x	y	z	x	y	z	x	y	z



- Each vertex is repeated 6 times (on average)
- Expensive to identify an adjacent triangle
  - Not suited for traversing a mesh

Representing **vertices + incidence**

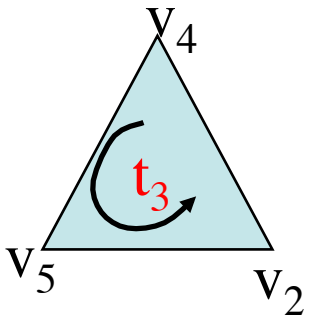
- **Samples:** Location of vertices + attributes (color, mass)
- Triangle/vertex **incidence:** specifies the indices of the 3 vertices of each triangle
  - Eliminates vertex repetition
  - But still does **not** support a direct access to neighboring triangles (**adjacency**)

Samples (vertices):

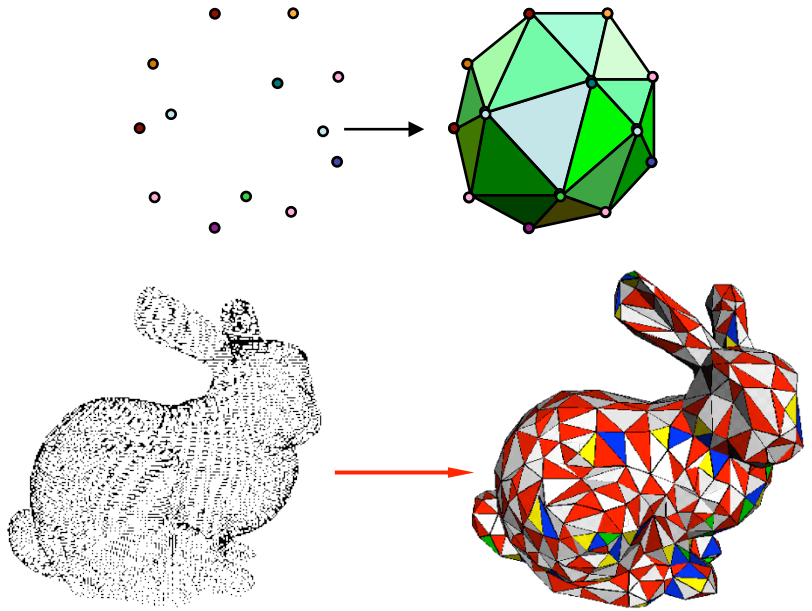
vertex 1	x	y	z	c
vertex 2	x	y	z	c
vertex 3	x	y	z	c
. . .	. . .			

Triangle/vertex incidence:

Triangle 1	1	2	3
Triangle 2	3	2	4
Triangle 3	4	5	2
Triangle 4	7	5	6
Triangle 5	6	5	8
Triangle 6	8	5	1
. . .	. . .		



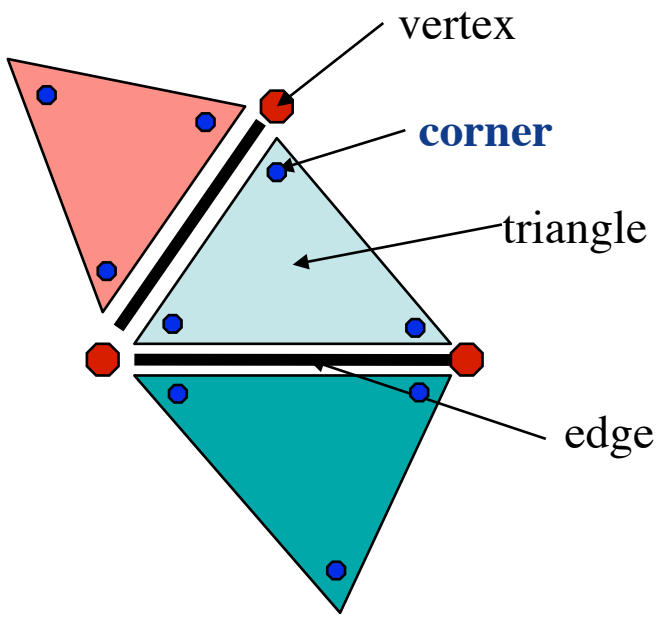
Order of vertex references defines outward direction (triangle orientation)



Corners, incidence and **adjacency**

Triangle/vertex incidence: identifies **corners**

- **Corner:** Abstract **association** of a **triangle** with a **vertex** (vertex-use)
  - A triangle has 3 corners
  - On average, 6 corners share a triangle



Triangle/triangle adjacency: Identifies neighboring triangles

- ✓ Neighboring triangles share a common edge
- ✓ Adjacency may be computed from the incidence
- ✓ Adjacency is convenient to **accelerate traversal** of triangulated surface
  - Walk from one triangle to an adjacent one
  - Estimate surface normals at vertices

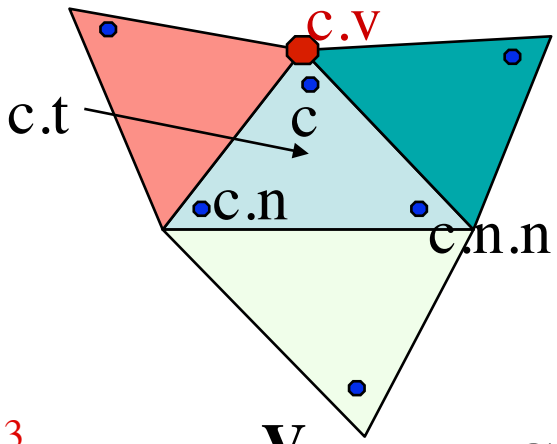
We will use the **Corner Table (V and O)** to represent incidence and adjacency

Representing the incidence as the **V table**

- Integer IDs for **vertices** (0, 1, 2… V-1) & **triangles** (0, 1, 2…T-1)

- **V-table:**
  - Identifies the **vertex ID** c.v for each **corner** c
  - The 3 corners of a triangle are **consecutive**
    - Triangle **number**: c.t = c DIV 3
  - Corners order for a triangle respects **orientation**
    - Cyclic order in which corners are listed
    - Next corner **around triangle**: c.n = 3 c.t + (c+1)MOD 3
    - Previous corner: c.p=c.n.n

- **Samples stored in geometry table (G):**
  - Location of vertex v is denoted v.g
    - Location of vertex of corner c is denoted c.v.g
      - Implementation as arrays: G[V[c]]
  - G tables list coordinates for vertex v
    - v.g = (v.g.x, v.g.y, v.g.z)
    - or use short cut: (v.x, v.y, v.z)



V

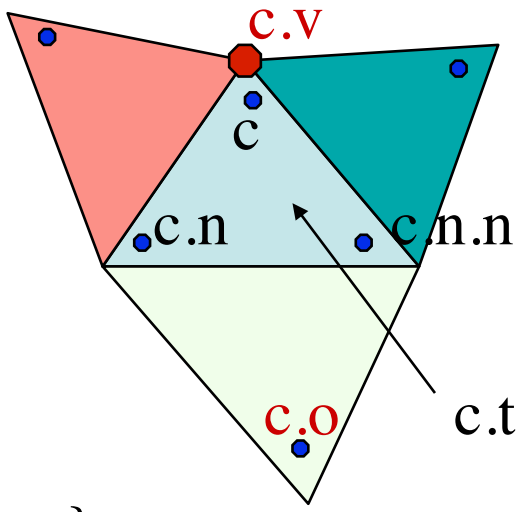
Triangle 0	1
Triangle 0	2
Triangle 0	3
Triangle 1	2
Triangle 1	1
Triangle 1	4
Triangle 2	1

G

vertex 1	x	y	z
vertex 2	x	y	z
vertex 3	x	y	z
vertex 4	x	y	z

Representing **adjacency** with the O table

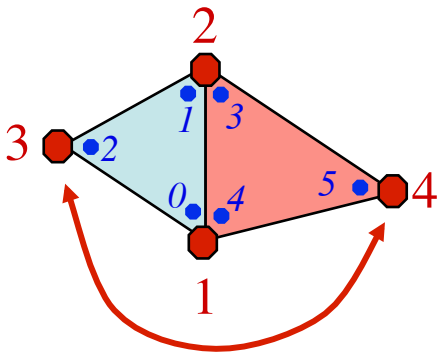
- For each corner **c** store:
  - **c.v** : integer reference to an entry in the G table
    - Content of V[c] in the V table
  - **c.o** : integer id of the opposite corner
    - Content of O[c] in the O table
- Computing the O table from V



For each corner a do For each corner b do

if (a.n.v==b.p.v && a.p.v==a.n.v) { O[a]:=b; O[b]:=a } ;

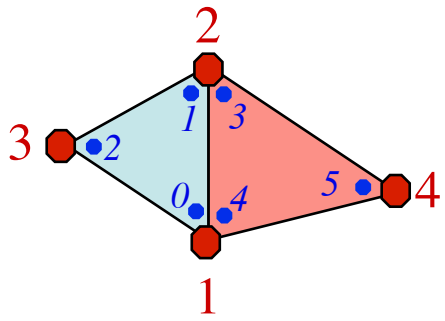
	V	O
Triangle 0 corner 0	1	7
Triangle 0 corner 1	2	8
Triangle 0 corner 2	3	5
Triangle 1 corner 3	2	9
Triangle 1 corner 4	1	6
Triangle 1 corner 5	4	2



	x	y	z
vertex 1			
vertex 2			
vertex 3			
vertex 4			

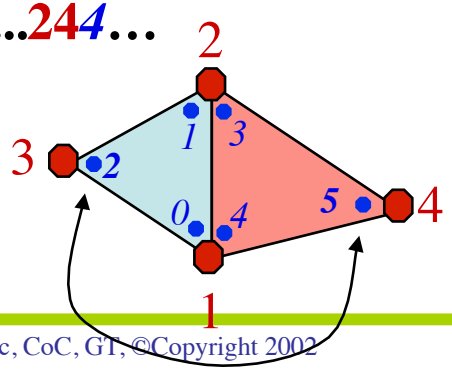
A faster computation of the O table

1. **List all of triplets {min(c.n.v, c.p.v), max(c.n.v, c.p.v), *c*}**
   - **230, 131, 122, 143, 244, 125, …**



	v	o	a
Triangle 1 corner 0	1		a
Triangle 1 corner 1	2		b
Triangle 1 corner 2	3		c
Triangle 2 corner 3	2		c
Triangle 2 corner 4	1		d
Triangle 2 corner 5	4		e

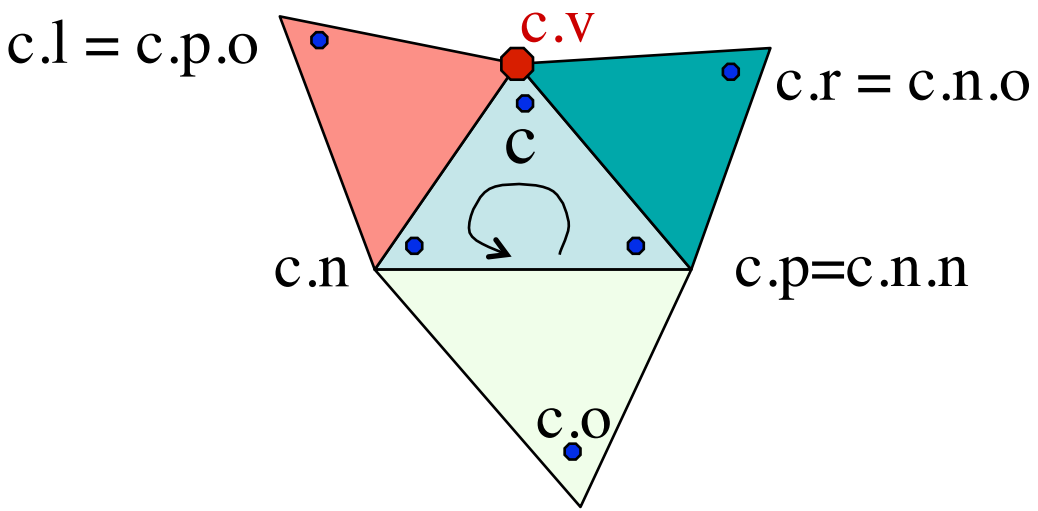
2. **Bucket-sort the triplets:**
   - **122, 125 …131… 143 …230…244 …**
3. **Pair-up consecutive entries 2k and 2k+1**
   - **(122, 125)…131… 143…230…244…**
4. **Their corners are opposite**
   - **(122,125)…131…143…230…244…**



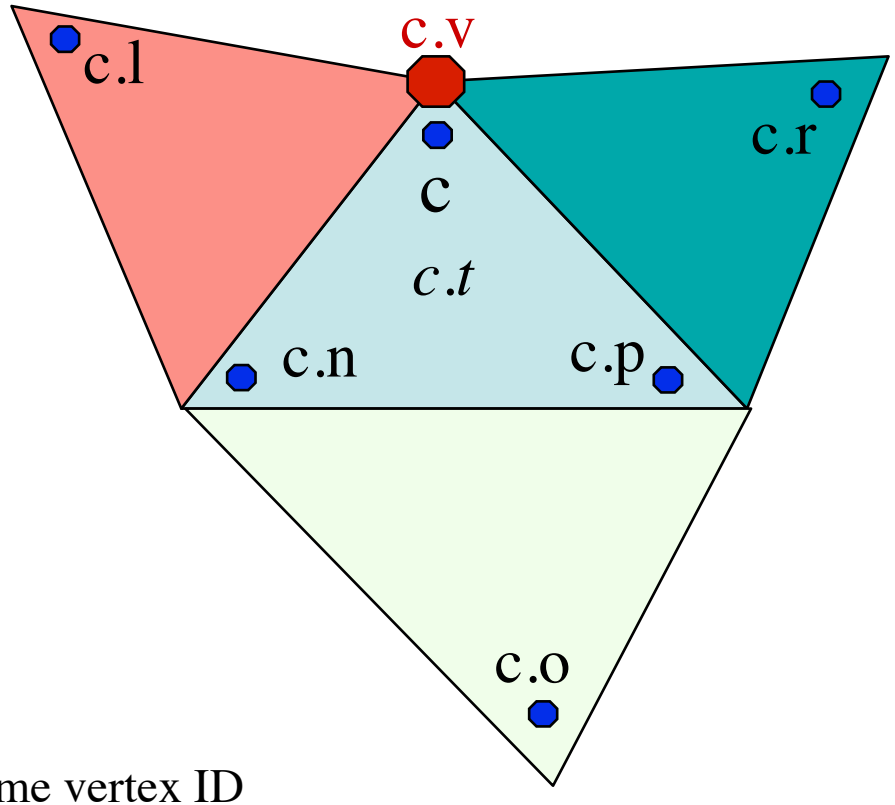
	v	o	a
Triangle 1 corner 0	1		a
Triangle 1 corner 1	2		b
Triangle 1 corner 2	3	5	c
Triangle 2 corner 3	2		c
Triangle 2 corner 4	1		d
Triangle 2 corner 5	4	2	e

Accessing left and right neighbors

- Direct access to opposite corners of right and left neighbors
  - **c.r = c.n.o**
  - **c.l = c.p.o**



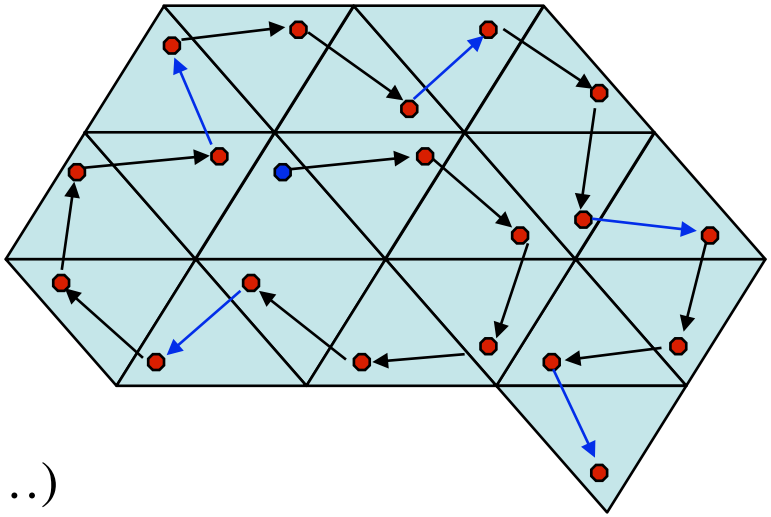
Summary notation

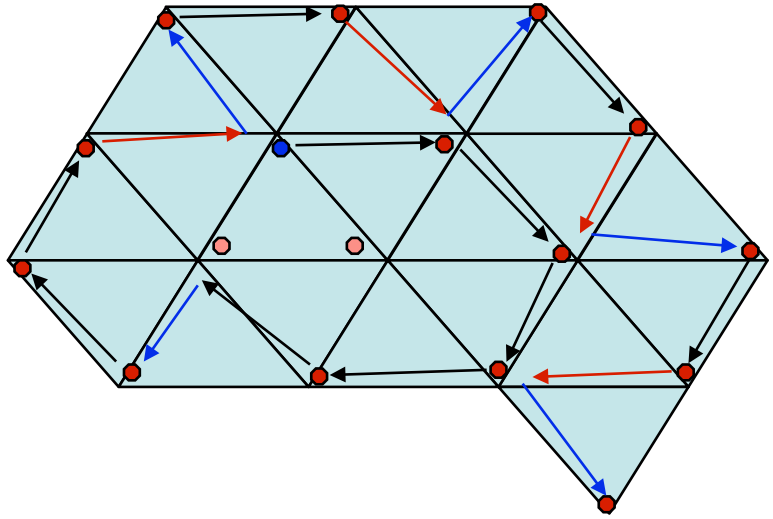


- Given corner c
  - c.v is the integer ID of its **vertex**
    - On average, 6 corners have the same vertex ID
  - c.v.g is the 3D point where c.v is located (**geometry**)
    - Must use .g for vector operations. Ex: c.n.v.g – c.v.g is vector along edge

Using adjacency table for T-mesh traversal

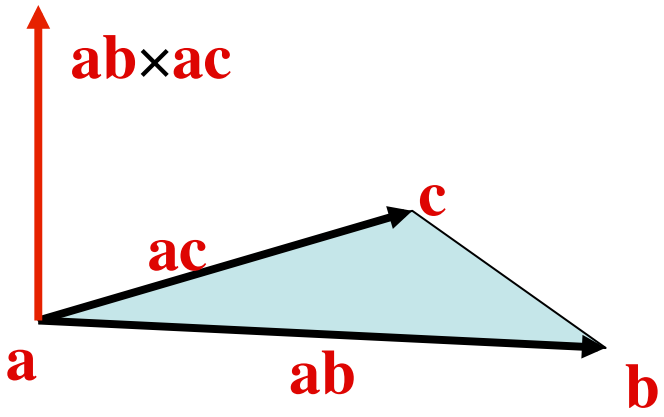
- **Visit T-mesh**
  - Mark triangles as you visit them
  - Start with any corner c and call Visit(c)
  - Visit(c)
    - mark c.t;
    - IF NOT marked(c.r.t) THEN visit(c.r);
    - IF NOT marked(c.l.t) THEN visit(c.l);
- **Label vertices** (for example as 1, 2, 3 …)
  - Label vertices with consecutive integers
  - Label(c.n.v); Label(c.n.n.v); Visit(c);
  - Visit(c)
    - IF NOT labeled(c.v) THEN Label(c.v);
    - mark c.t;
    - IF NOT marked(c.r.t) THEN visit(c.r);
    - IF NOT marked(c.l.t) THEN visit(c.l);

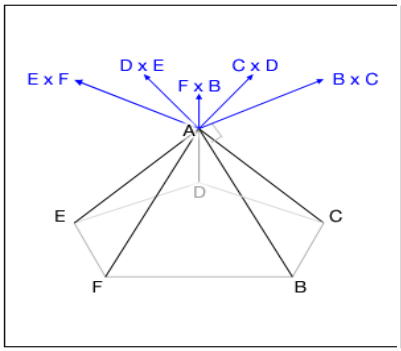


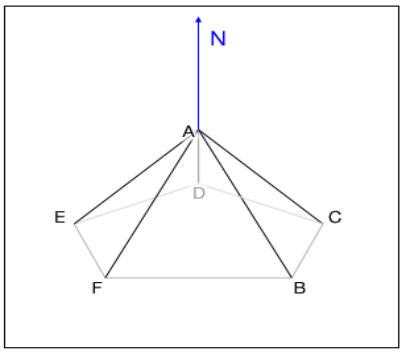


Estimating a vertex normal

- At vertex **a** having **b**, **c**, **d**, **e**, **f** as **neighbors**
- **N=ab×ac+ac×ad+ad×ae+ae×af+af×ab**
  - The notation **U×V** is the **cross product** of the two vectors
  - The notation **ac** is the vector between **a** and **c**. In other words: **ac=c–a**
  - Note that **N** is independent of the position of vertex **a**
    - **(b–a)×(c–a)+(b–a)×(c–a)+… = b×c+a×a–b×a–a×c + c×d+a×a–c×a–a×d +… –a×b +…**
    - **a×a = 0** , **–a×c** and **–c×a** cancel out, same for all other cross-products containing **a**
    - We are left with = **b×c + c×d** + … which does not depend on **a**
- Then **divide N** by its norm to make it a unit vector

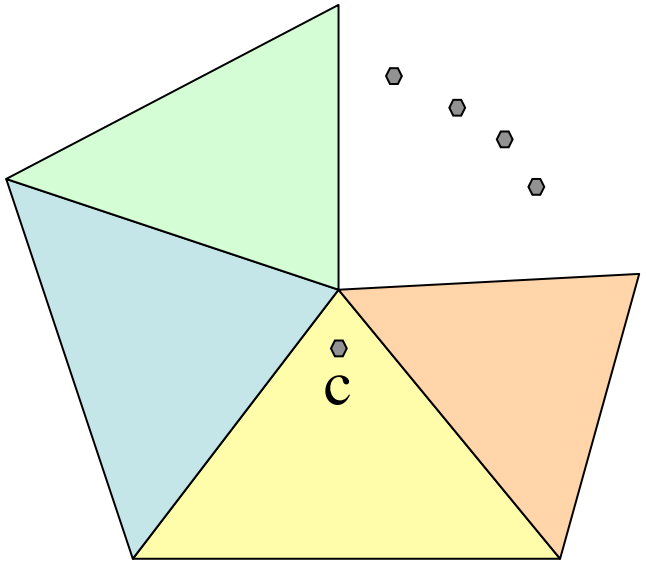






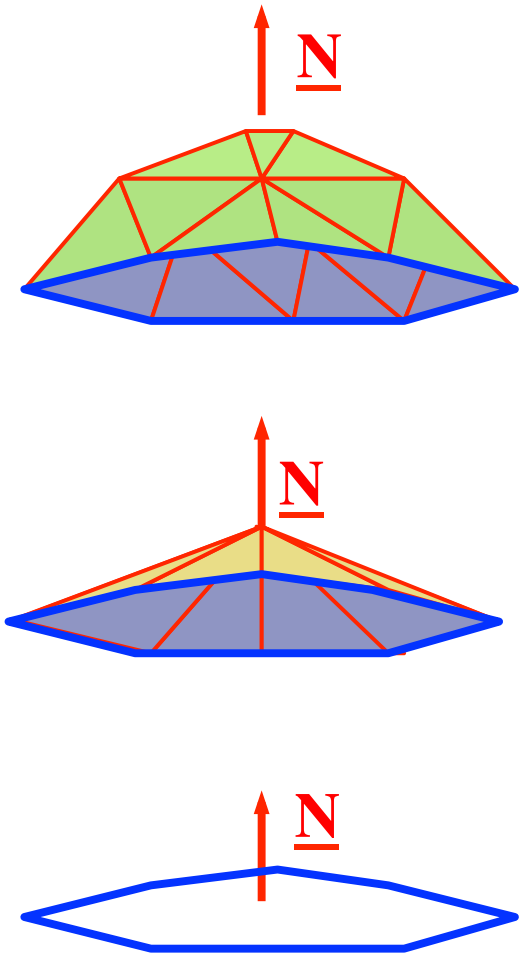
Exercise

- Given corner c, estimate the surface normal **n** at vertex c.v
  - Use the sum of cross-products approach proposed above to compute **N**
  - Then normalize it to get **n**
- Remember that **n** is independent of the position c.v.g of c.v



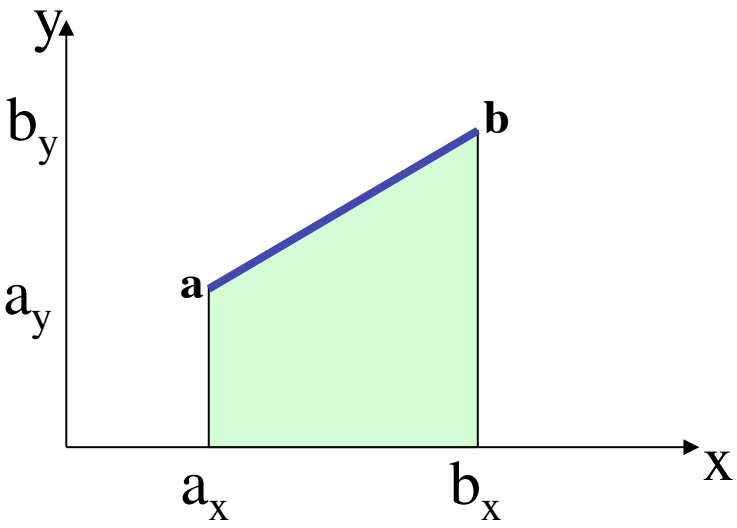
Computing the normal to a patch

- Consider a non-planar **patch** of triangles in 3D
- Consider its border (blue curve)
  - Border edges have a single incident triangle
- Estimate the normal **N** to the patch as the sum of the normals to its triangles weighted by the triangle areas (**ab**×**ac**).
- The result is independent on the position of the internal vertices (proof?)
- It only depends on the border.
- We could make a triangle-fan with some point **o** and compute **N**
  - Sum **oa**×**ob** for all border edges (**a**,**b**)
- Or we can use the projections of the border edges on in the x, y, and z directions…

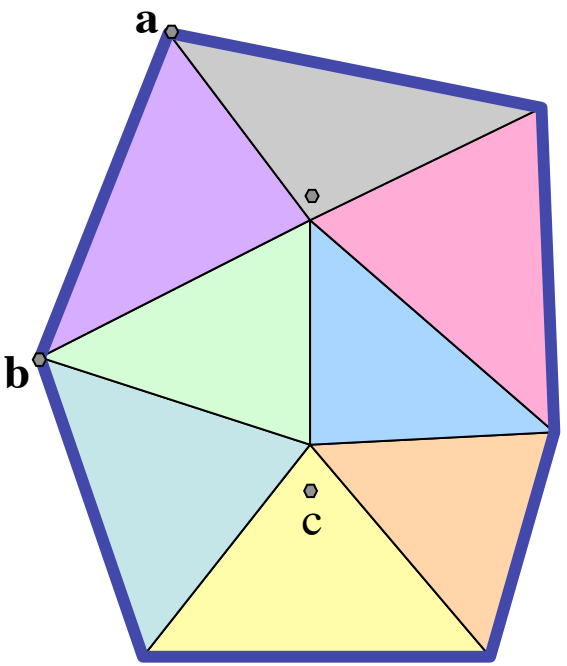


Faster computation of the normal N to a patch

- **N** may be computed from the projections of the border edges (**a**,**b**) onto the 3 principal planes:
- Compute signed areas of "shadows" of the border loop on the YZ, ZX, and XY planes
  - $\mathbf{N}_z$:= the sum of signed areas of 2D trapozoids under the projection of (**a**,**b**), for each border edge (**a**,**b**).
  - Same for $\mathbf{N}_x$ and $\mathbf{N}_y$



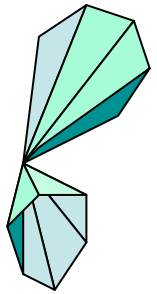
$$\text{area}(\mathbf{a},\mathbf{b}) := (a_y + b_y)(b_x - a_x)$$



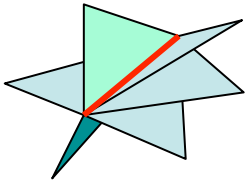
Assume T-mesh is an **orientable manifold**

- A set of triangles forms a **manifold** mesh when:
  - The 3 **corners** of a triangle refer to **different vertices** (no zero area triangles)
  - Each **edge bounds** exactly **2 triangles**
  - **The** star **of each vertex v** forms a single cone (connected if we remove **v**)
    - **Star** = union of edges and triangles incident upon the vertex

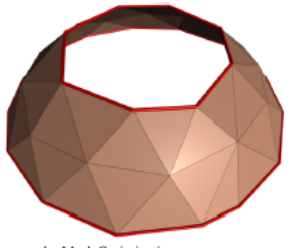
Non-manifold vertex



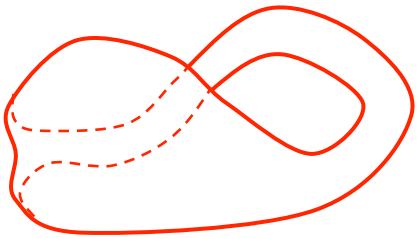
Non-manifold edge



border edges



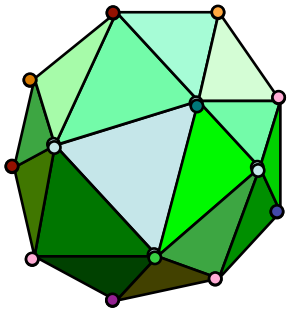
- A manifold triangles mesh is **orientable** when:
  - Triangle can be oriented consistently

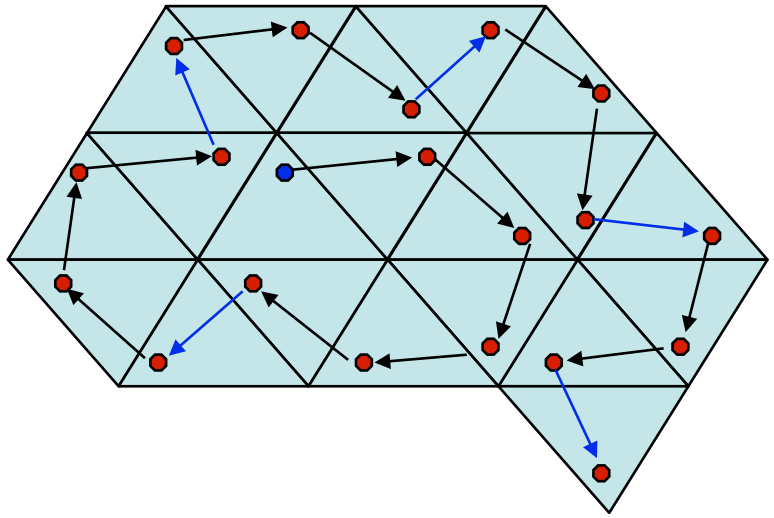


Klein bottle

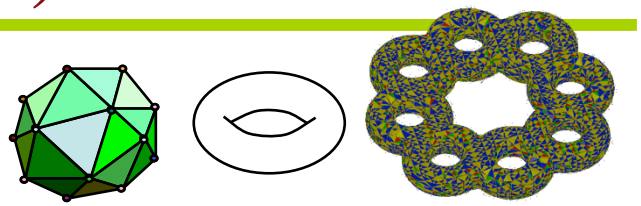
Shells: connected portions of T-meshes

- All triangles of a **shell** form a **connected** set
  - Two adjacent triangles are connected (through their common edge)
  - Connectivity is a transitive relation (can identify a mesh by invading it)
- **To identify a new shell**
  - Pick a new "color" (ID) and a virgin triangle
  - Use the Visit(c) procedure to reach all of the triangle of the shell and paint them

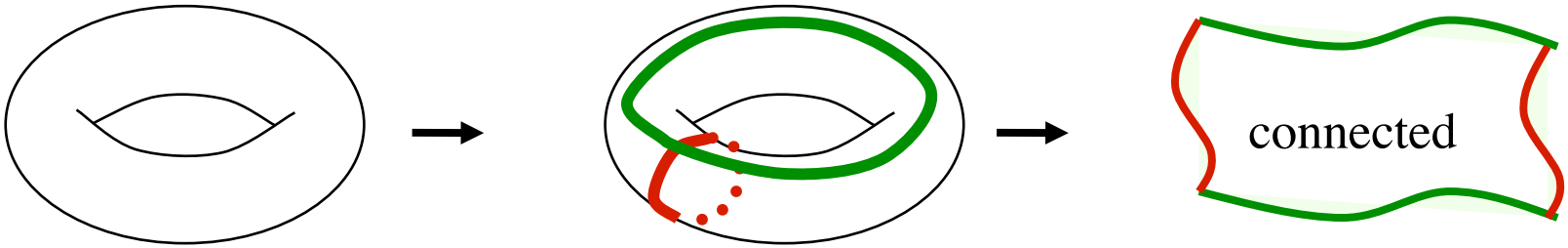




Genus (number of handles) in a shell



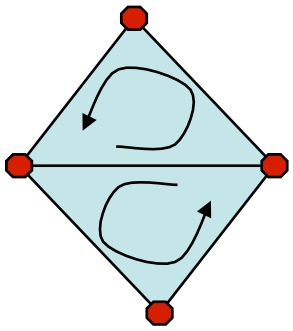
- **Handles** correspond to **through-holes**
  - A sphere has zero handles, a torus has one
- The **number H of handles** is called the **genus** of the shell
  - A handle cannot be identified as a particular set of triangles
- A T-mesh has **k handles** if and only if you can **remove at most 2k edge-loops without disconnecting** the mesh

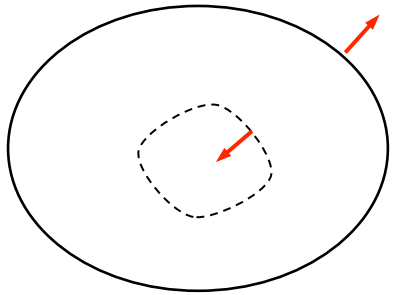


- Genus of a shell may be computed using: **H = T/4 – V/2 + 1**
  - Remember as **T = 2V – 4 + 4H**
- In a zero-genus mesh, **T = 2V – 4** (Euler-Poincare formula)

Solids and cavities

- A **solid** (here restricted to be a connected manifold polyhedron) may be represented by its **boundary**, which may be composed of one or more manifold shells
  - One shell defines the external boundary
  - The other shells define the boundaries of internal cavities (**holes**)
- All the shells of a solid can be consistently **oriented**
  - It you were a bug sitting on the **outward side** of an oriented triangle you would have to turn counterclockwise (with respect to your up vector normal to the triangle) to look at the vertices in the order in which their IDs are stored in the V table
  - The outward side of each triangle must be adjacent to the exterior of the solid.





Examples of questions for tests

- Define incidence, adjacency, corner, shell, solid, genus
- Difference between handle (through-hole) and hole (cavity)
- Explain the content of a corner table
- Provide the implementation of the corner operators: c.v, c.o, c.t, c.n, c.p, c.r, c.l
- How can we identify the corner opposite to c using the V table?
- Explain how to build a Corner Table from a list of triangles?
- How to identify the shells of a mesh represented by a corner table?
- How to compute the genus (number fo handles) of each shell?
- Can we represent solid by its bounding triangles (not-oriented)
- How to test whether a vertex lies inside a solid
- How to compute the volume of a solid

Questions to think about

- How to pick the proper outward orientation for a triangle
- How to consistently orient the triangles of a shell
- How to test whether a point P is inside a shell S
- How to identify the shells that bound a solid
- How to identify the solids (and their bounding shells) from a corner table that represents all the triangles
- How to orient the shells bounding a solid
- How to identify the non-manifold vertices of a shell
- How to test whether a shell is free from self-intersections
- How to test whether two shells intersect one another
- What if the triangles do not form a water-tight shell