Meshes and More

CMSC425.01 fall 2019

Administrivia

- Google form distributed for grading issues

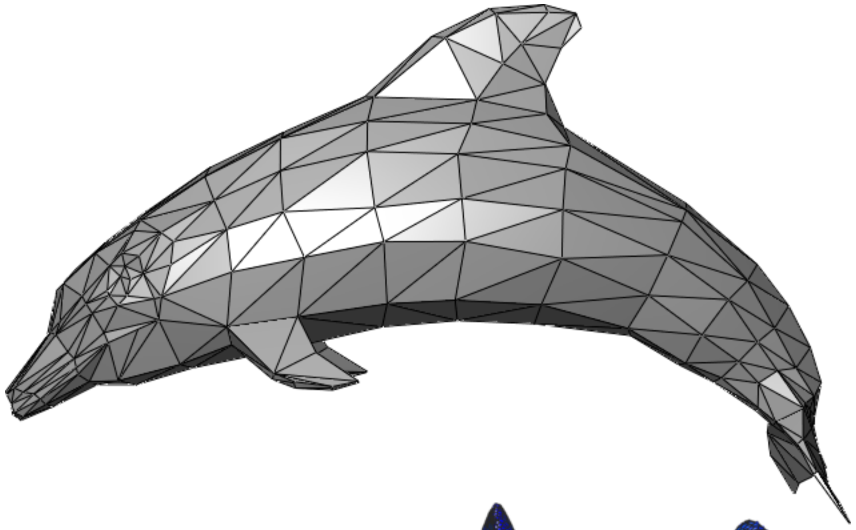
Today’s question

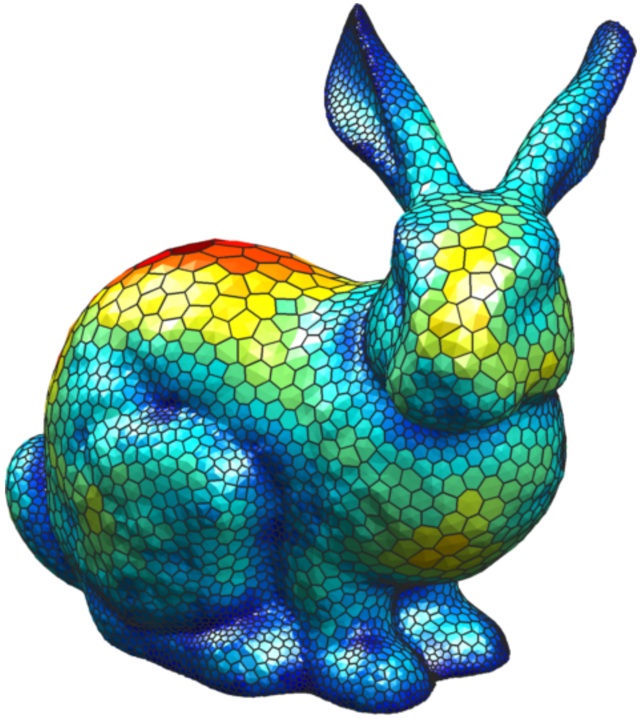
How to represent objects

Polygonal meshes

- Standard representation of 3D assets

- Questions:
  - What data and how stored?
  - How generate them?
  - How color and render them?





Data structure

- Geometric information
  - Vertices as 3D points
- Topology information
  - Relationships between vertices
  - Edges and faces

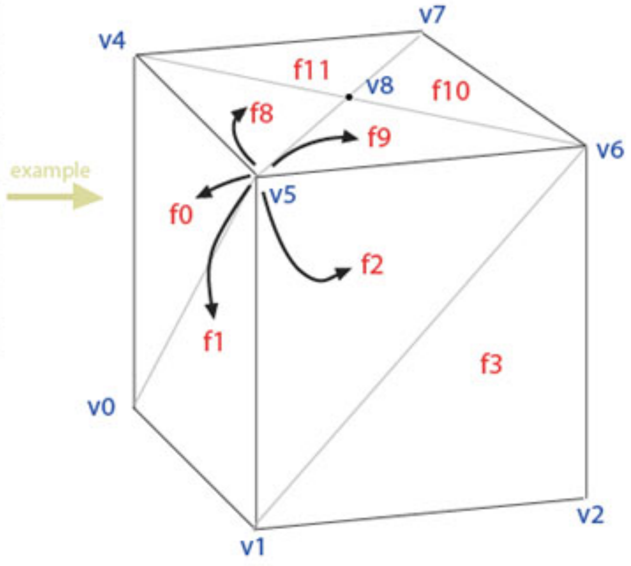
Face-Vertex Meshes

Face List

f0	v0 v4 v5
f1	v0 v5 v1
f2	v1 v5 v6
f3	v1 v6 v2
f4	v2 v6 v7
f5	v2 v7 v3
f6	v3 v7 v4
f7	v3 v4 v0
f8	v8 v5 v4
f9	v8 v6 v5
f10	v8 v7 v6
f11	v8 v4 v7
f12	v9 v5 v4
f13	v9 v6 v5
f14	v9 v7 v6
f15	v9 v4 v7

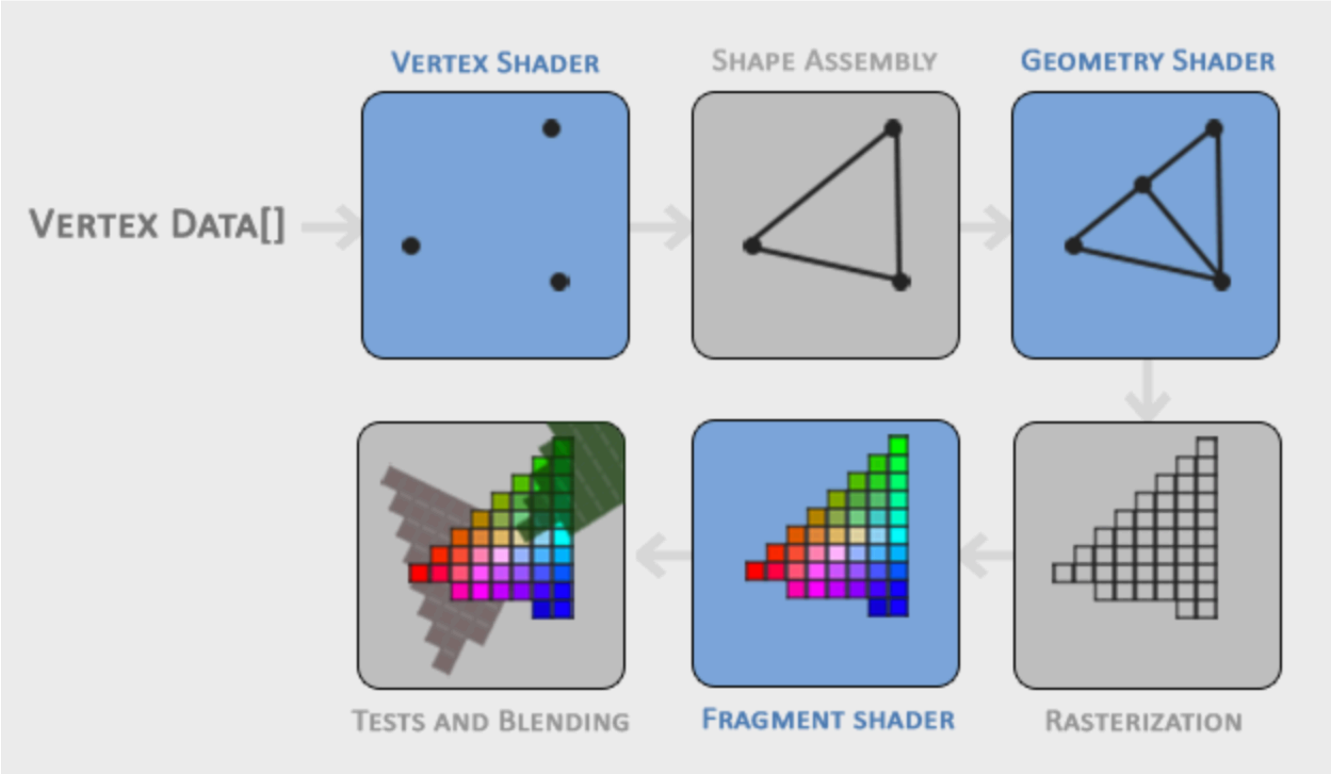
Vertex List

v0	0,0,0	f0 f1 f12 f15 f7
v1	1,0,0	f2 f3 f13 f12 f1
v2	1,1,0	f4 f5 f14 f13 f3
v3	0,1,0	f6 f7 f15 f14 f5
v4	0,0,1	f6 f7 f0 f8 f11
v5	1,0,1	f0 f1 f2 f9 f8
v6	1,1,1	f2 f3 f4 f10 f9
v7	0,1,1	f4 f5 f6 f11 f10
v8	.5,.5,0	f8 f9 f10 f11
v9	.5,.5,1	f12 13 14 15



Vertex and fragment shaders

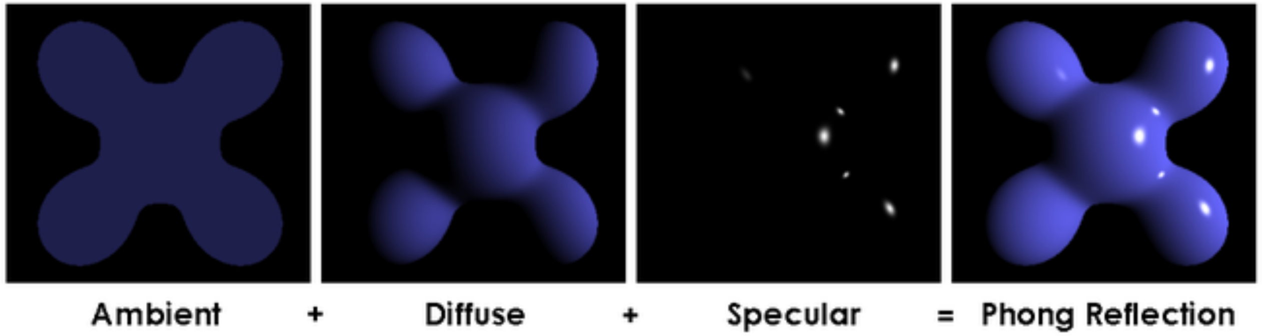
- Mapping triangle to screen
- Map and color vertices
  - Vertex shaders in 3D
- Assemble into fragments
- Render fragments
  - Fragment shaders in 2D

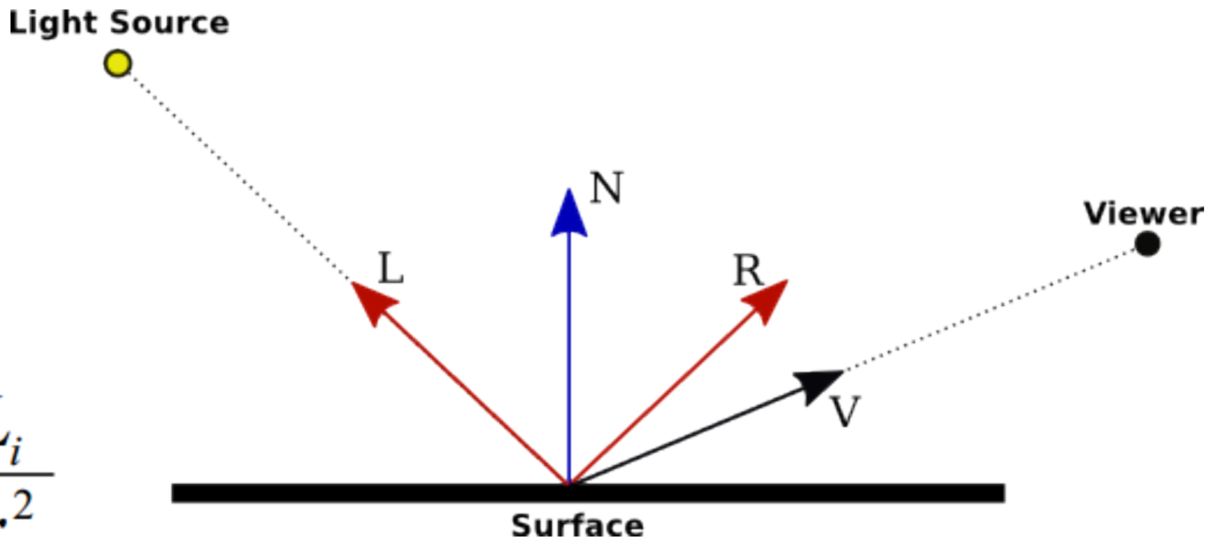


Normals and shading – shading equation

- Light equation
  - k terms – color of object
  - L terms – color of light
- Ambient term - ka La
  - Constant at all positions
- Diffuse term - kd (n • l)
  - Related to light direction
- Specular term - $(v \bullet r)^q$
  - Related to light, viewer direction

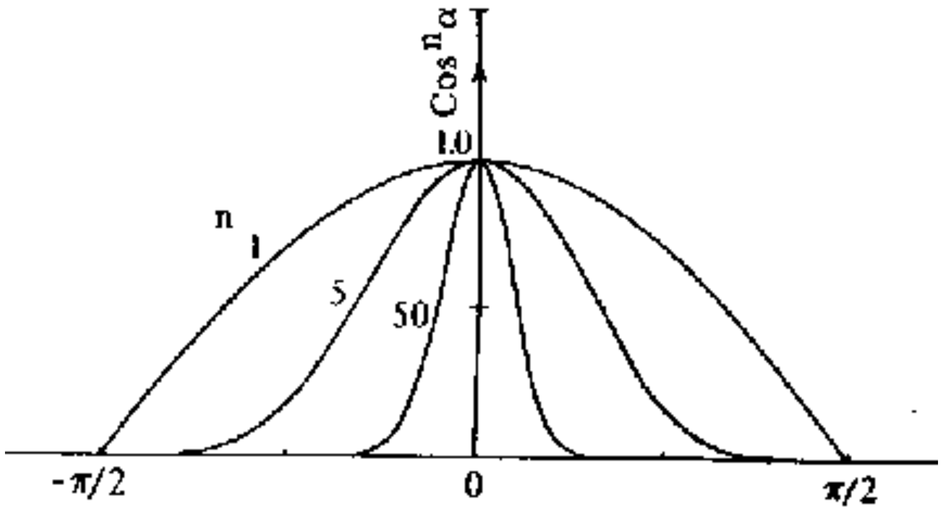
$$L_o = k_a L_a + \left(k_d (\mathbf{n} \cdot \mathbf{l}) + k_s (\mathbf{v} \cdot \mathbf{r})^q \right) \frac{L_i}{r^2}$$



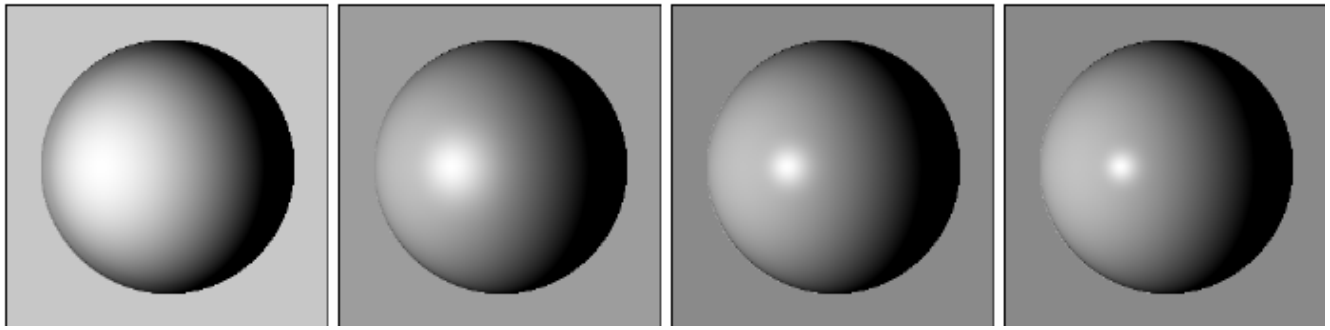


Phong exponent

- Powers of cos $(v \bullet r)^q$
  - v and r normalized
- Tightness of specular highlights

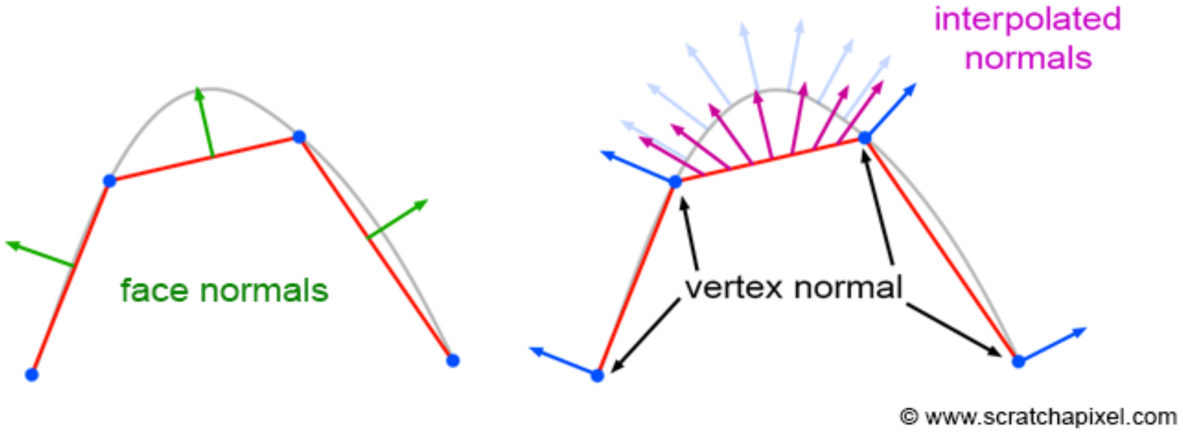


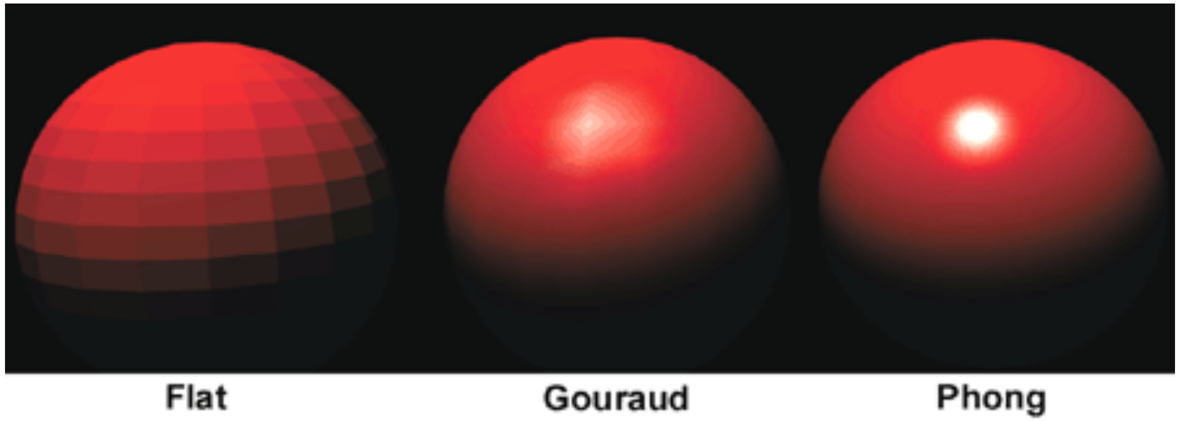
- Shininess of object



Normals and shading

- Face normal
  - One per face
- Vertex normal
  - One per vertex. More accurate
- Interpolation
  - Gouraud: Shade at vertices, interpolate
  - Phong: Interpolate normals, shade



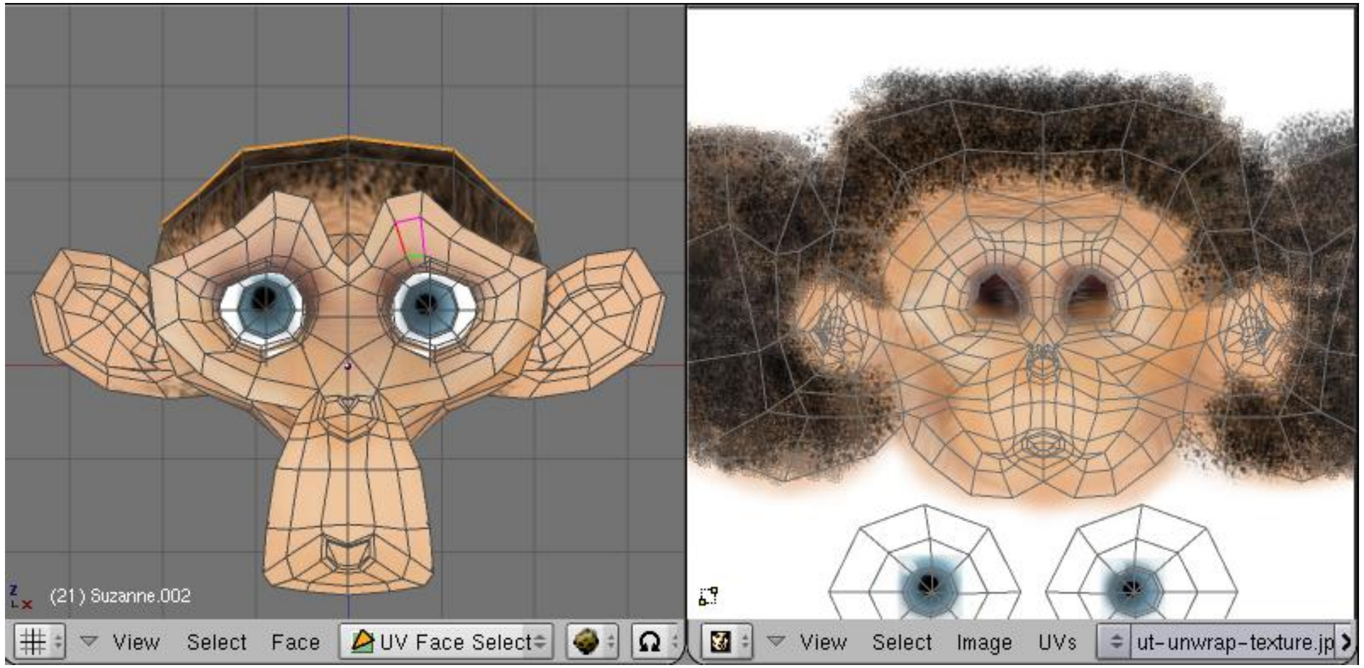


Texture mapping

- Vary color across figure
- ka, kd and ks terms

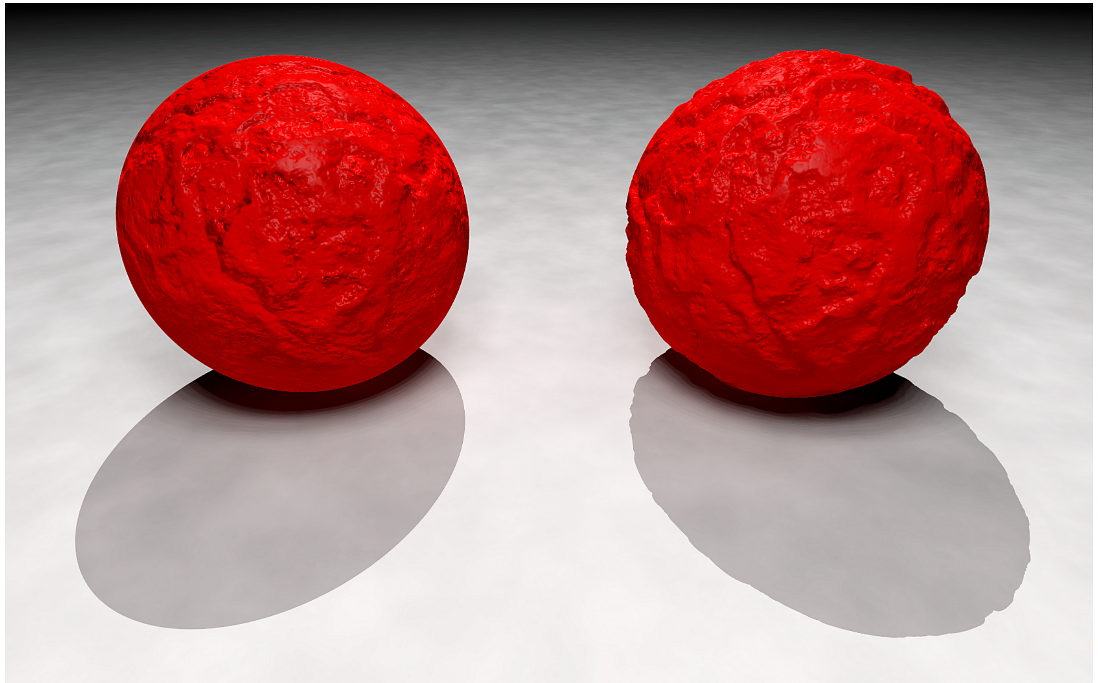
- Interpolate position inside polygon to get color

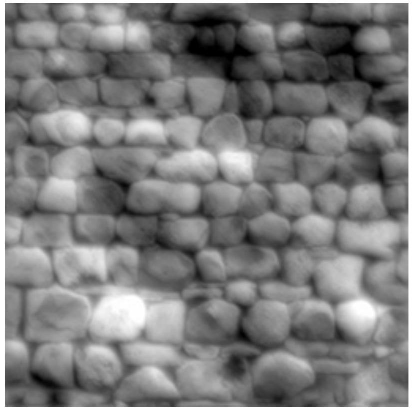
- Not trivial!
- Mapping complex

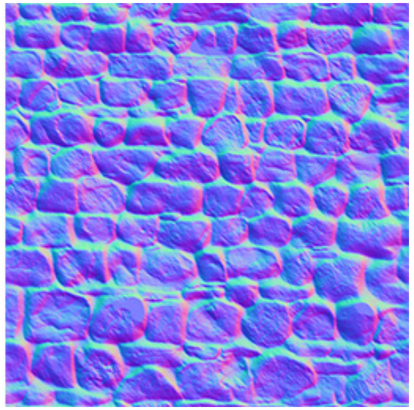


Bump mapping

- "Texture" map of
  - Perturbed normals (on right)
  - Perturbed height (on left)

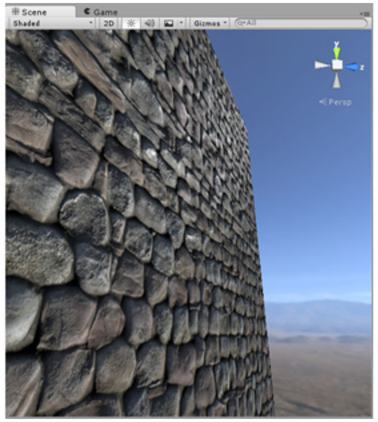


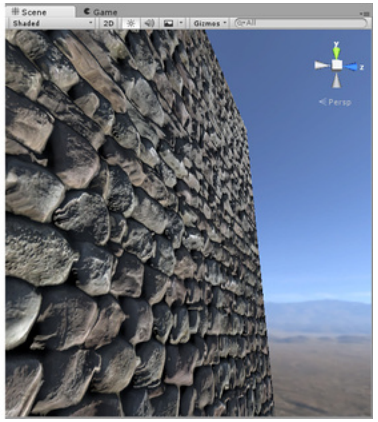


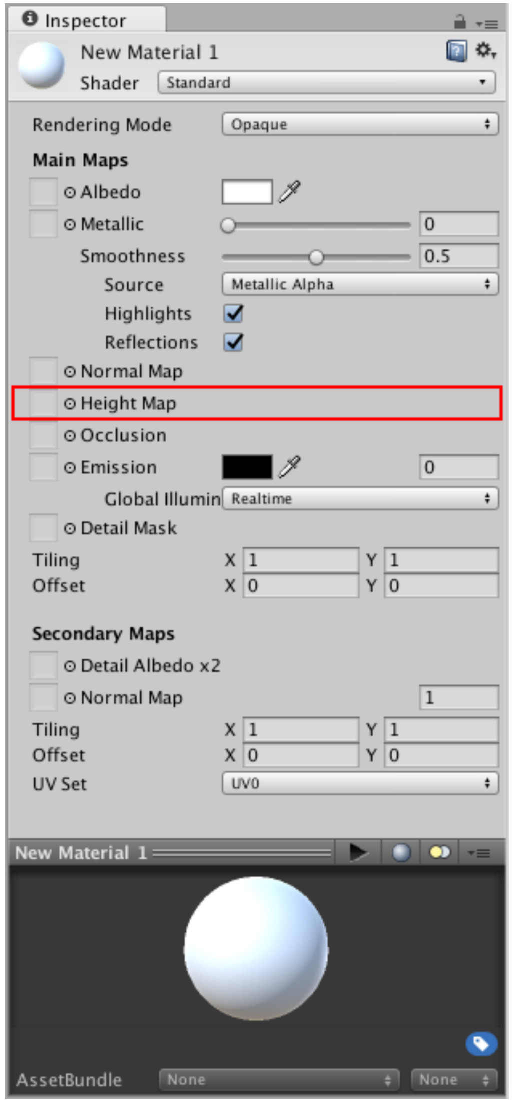


Summary – full polygon mesh asset

- Mesh can have vertices, faces, edges plus normals
- Material shader can have
  - Color (albedo)
  - Phong coefficient q
  - Normal map
  - Texture map
  - Bump map
  - Height map







WebGL example

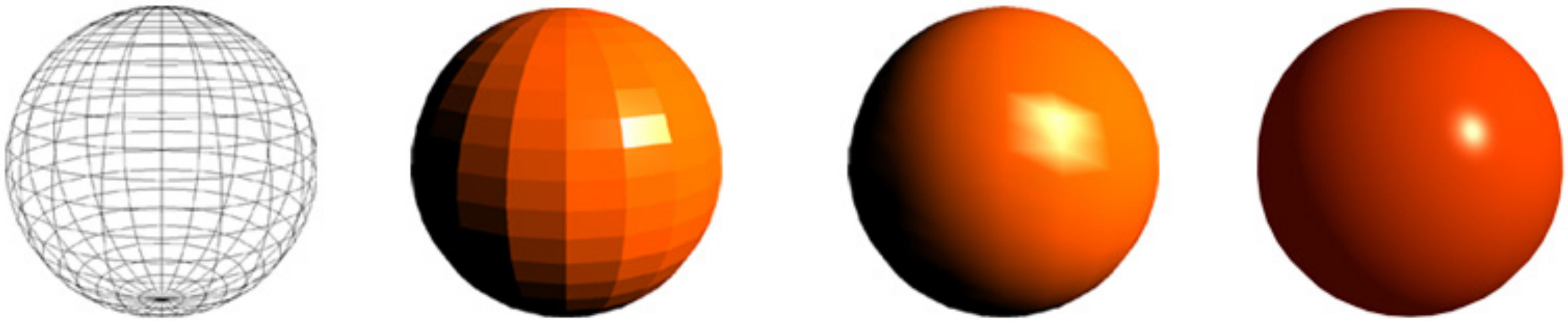
- Based on a http://multivis.net/lecture/phong.html | WebGL applet by Prof. Thorsten Thormählen. Modified by Johannes Kehrer

- http://www.cs.toronto.edu/~jacobson/phong-demo/

Rending mesh – throw data at screen

- Wireframe
- Flat shading
- Gouraud – shading at vertices
- Phong – shading at pixels

Occlusion – Z-buffer



(a) (b) (c) (d)

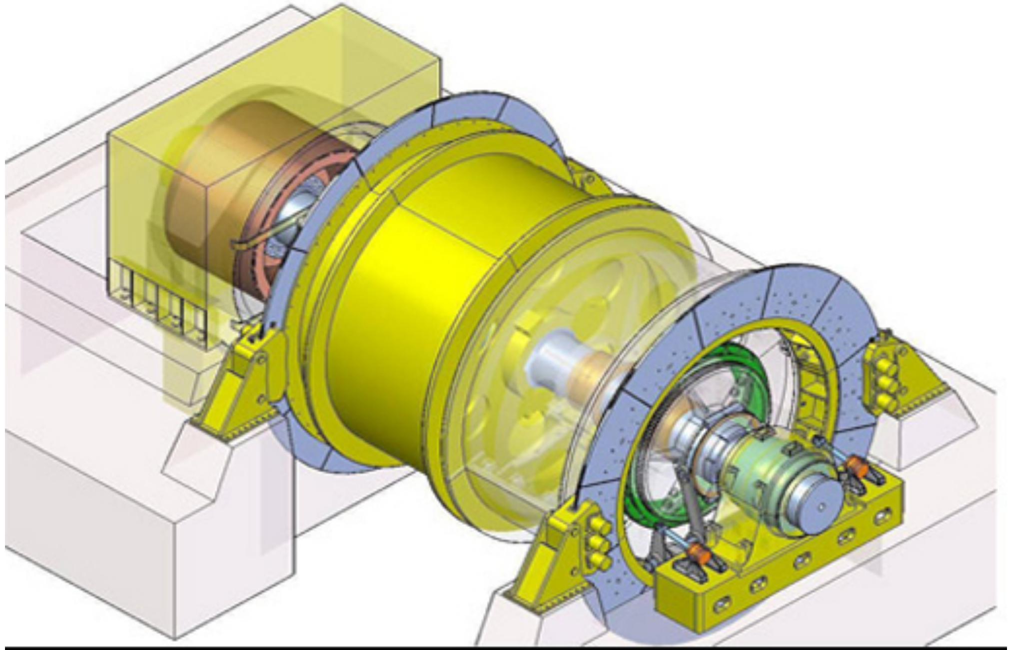
How create 3D asset?

- Model by hand
- Model by procedure
- Model by scanning

- Mix all three
  - By hand control B-spline surface procedure
  - Take pictures for texture map, bump map

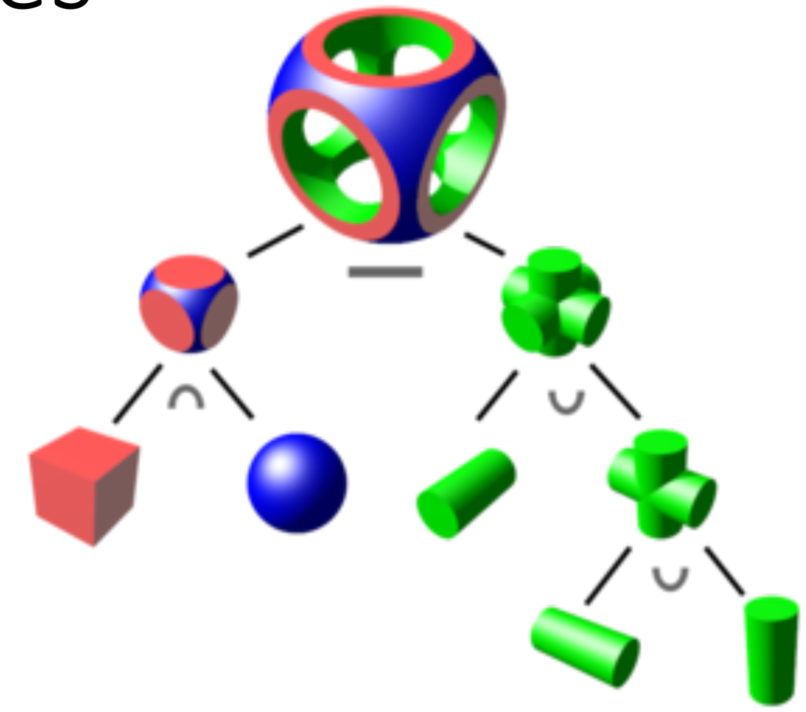
Constructive Solid Geometry (CSG)

- Volume based
- Supports physical and simulation of objects
- Heavily used in industry for precision and flexibility
- Can output polygonal mesh for Unity asset



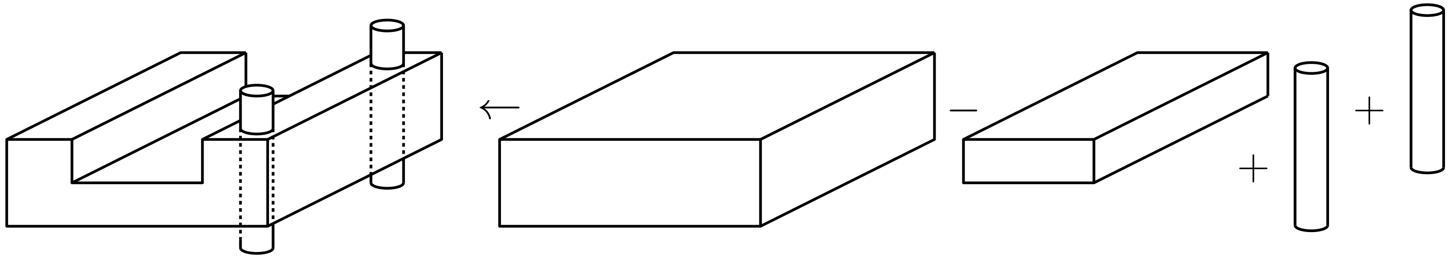
THIS IS A CAPTION FOR THE IMAGE

Boolean operations on primitives



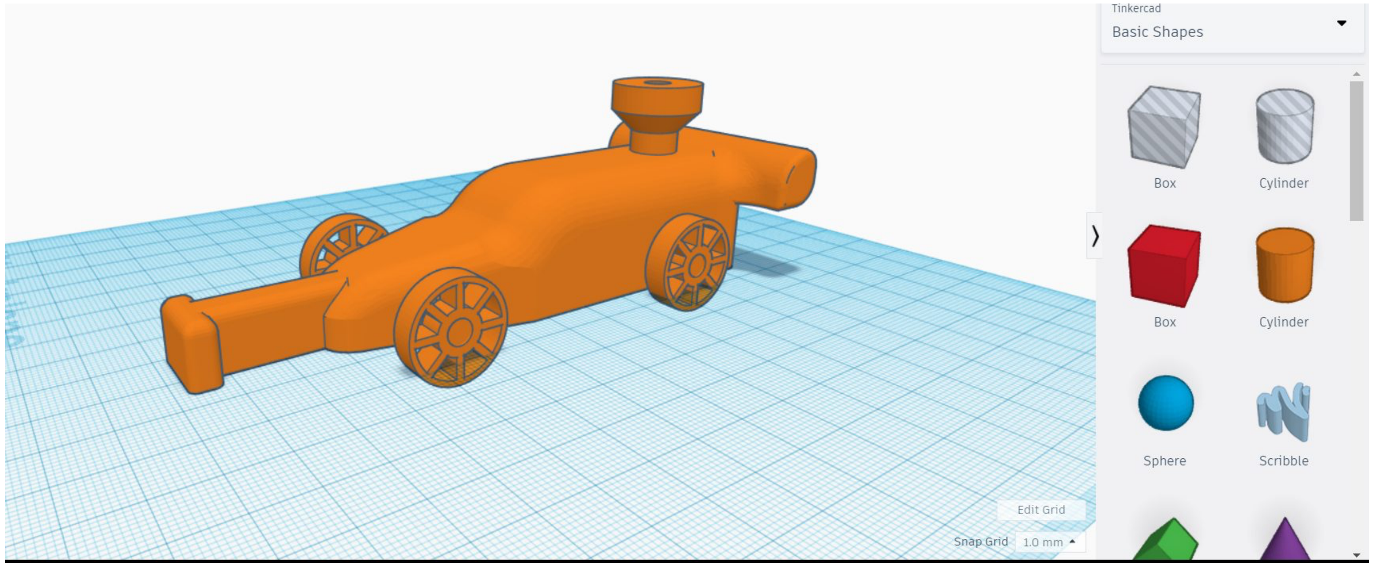
- Union
- Intersection
- Difference
- (and scaling)

- Rectangular blocks
- Spheres
- Cylinders



Easy CSG intro: Tinkercad

- https://www.tinkercad.com

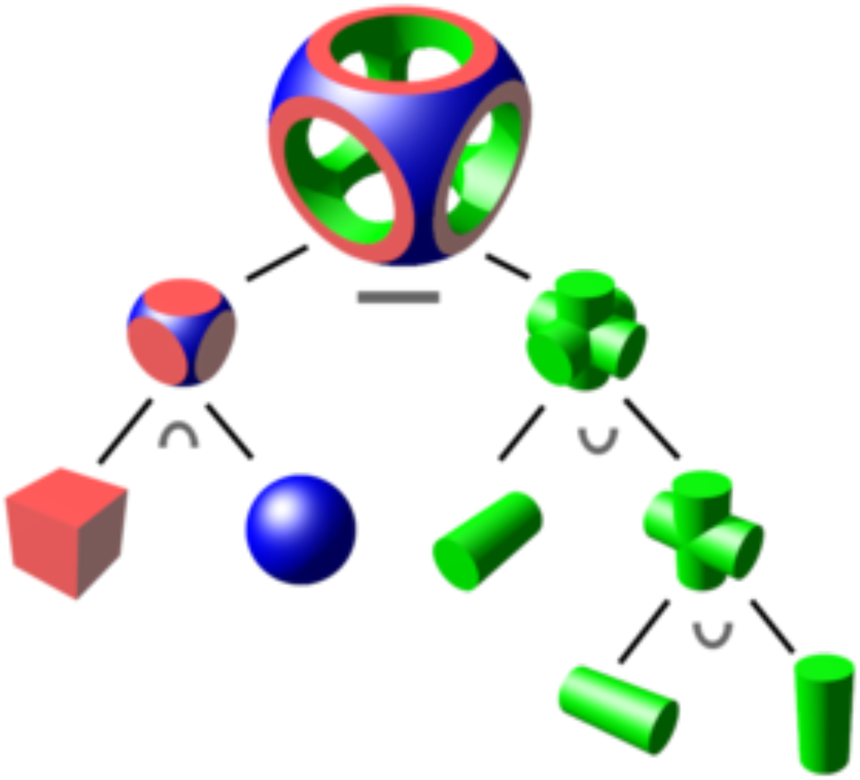


- Free
- Easy
- Online tutorials
- Can add own procedural object code in Javascript!

CSG tree

- Unevaluated CSG object represented as tree

- How determine if point is inside object?



CSG tree

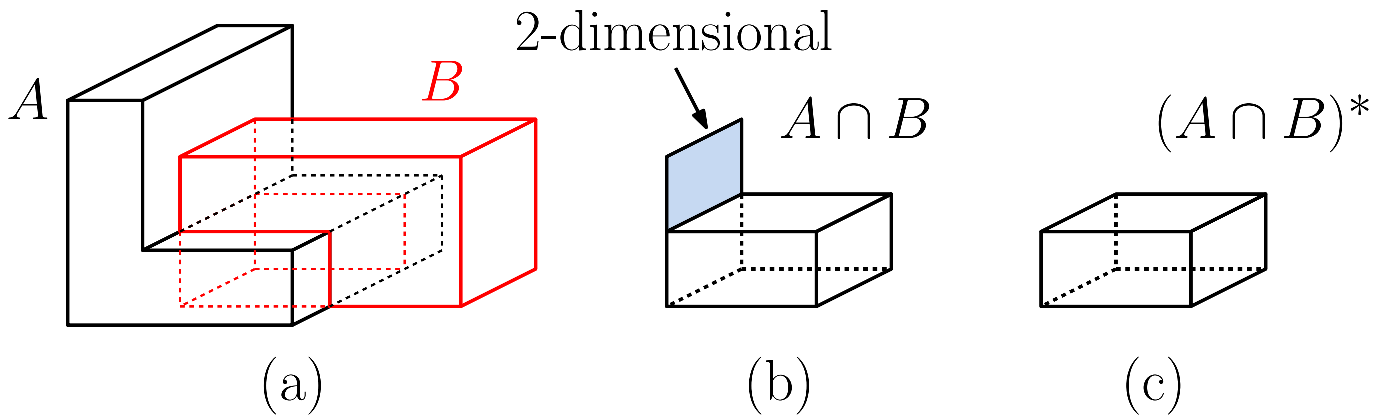
- Recursive procedure

Membership Test for CSG Tree

```
bool isMember(Point p, CSGnode u) {
    if (u.isLeaf)
        return u.primitiveMemberTest(p);
    else if (u.isUnion)
        return isMember(p, u.left) ||  isMember(p, u.right);
    else if (u.isIntersect)
        return isMember(p, u.left) &&  isMember(p, u.right);
    else if (u.isDifference)
        return isMember(p, u.left) && !isMember(p, u.right);
}
```

CSG problems: boundary issues

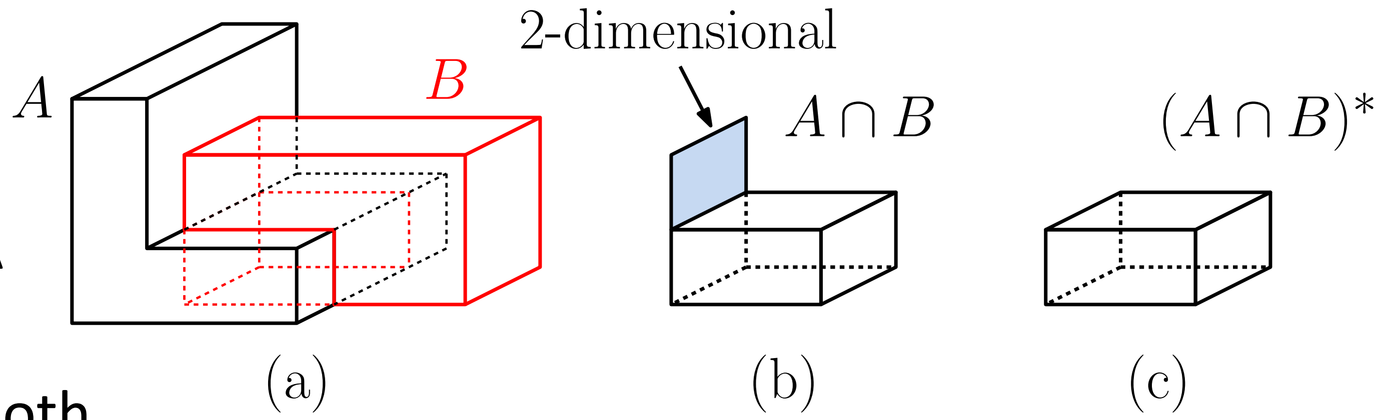
- Operation produces 2d glitch
- ??



CSG problems: boundary issues

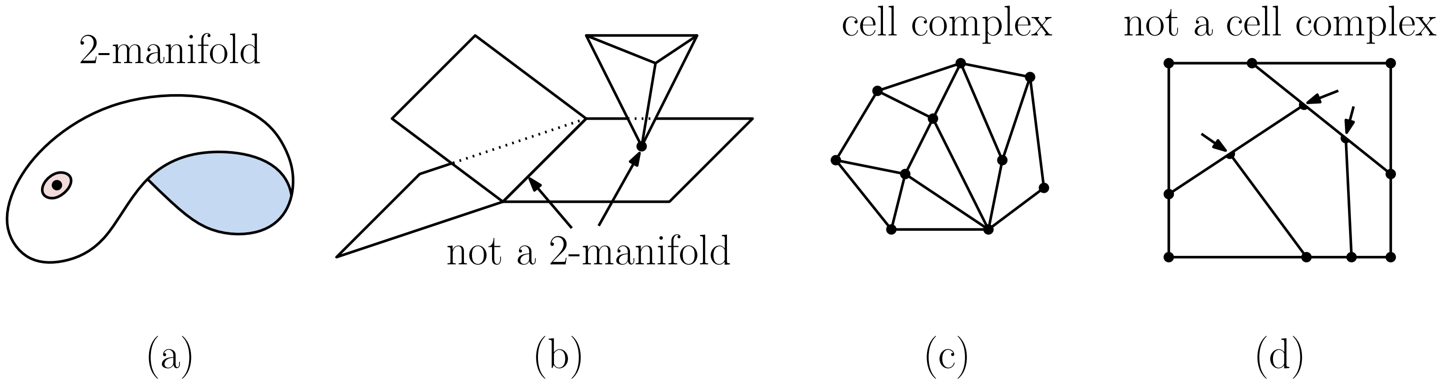
- Operation produces 2d glitch
- Definitions
  - Interior int(A) – surrounded by A
  - Exterior ext(A) – no A adjacent
  - Boundary bnd(A) – adjacent to both
  - Closure(A) = int(A) union bnd(A)
- A* = closure(interior(A))

- A op* B = closure(int(A op B))



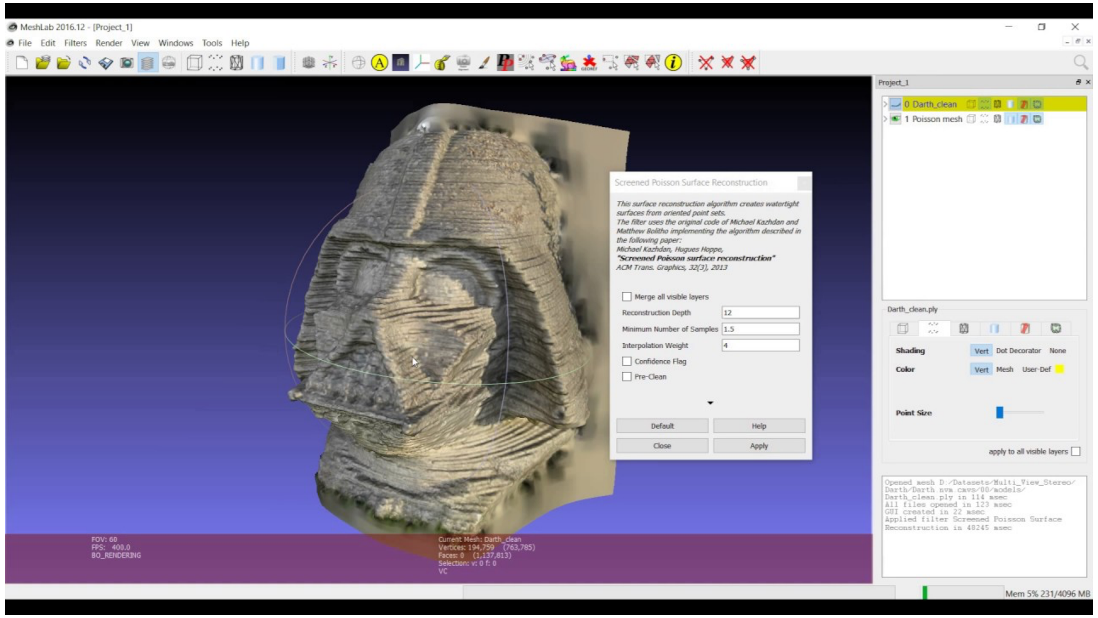
Polygonal meshes

- Represents boundary of object
- 2D manifold
  - Neighborhood of vertex is 2d
- Constraints:
  - No t-junctions
  - Only 2 faces/edge
  - No points inside polygon



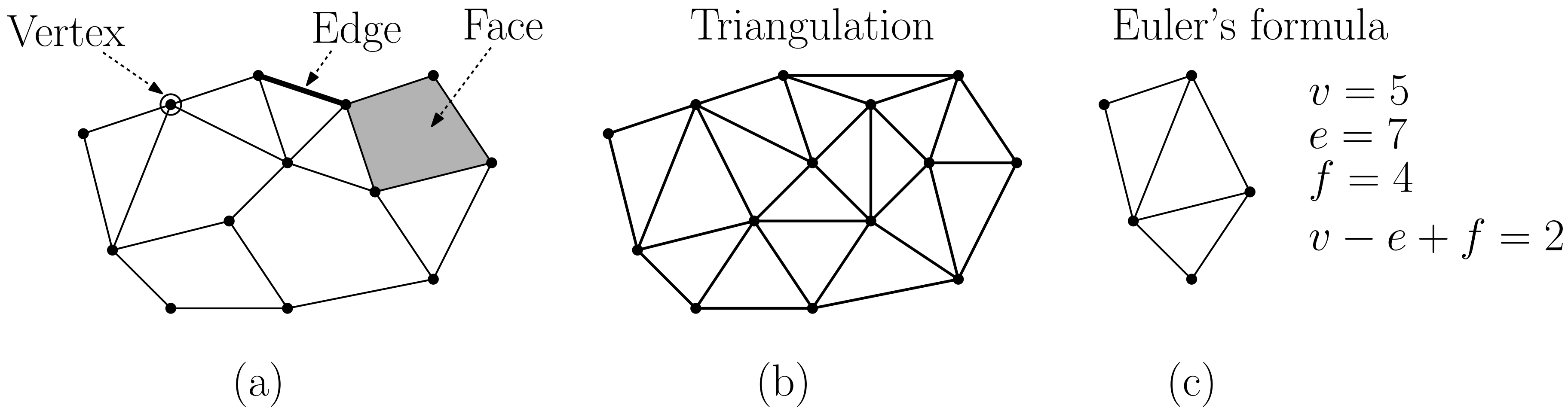
Meshlab

- Polygonal mesh editor
- Free
- View, edit, clean up meshes
- Many sophisticated algorithms



Meshes as planar graphs

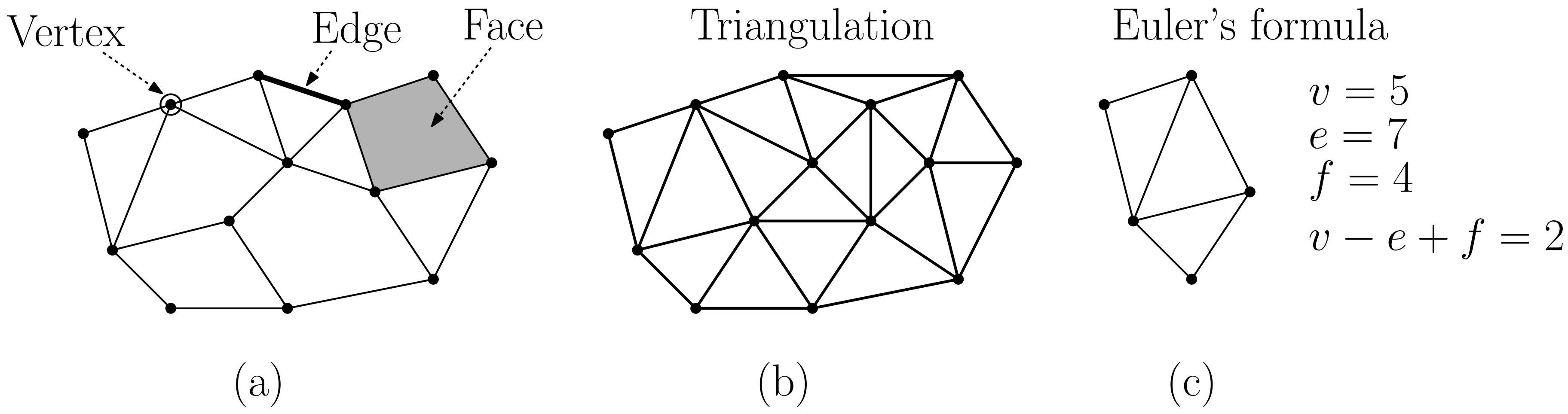
- Euler's formula
- $v - e + f = 2$



Meshes as planar graphs

- Euler's formula
- $v - e + f = 2$

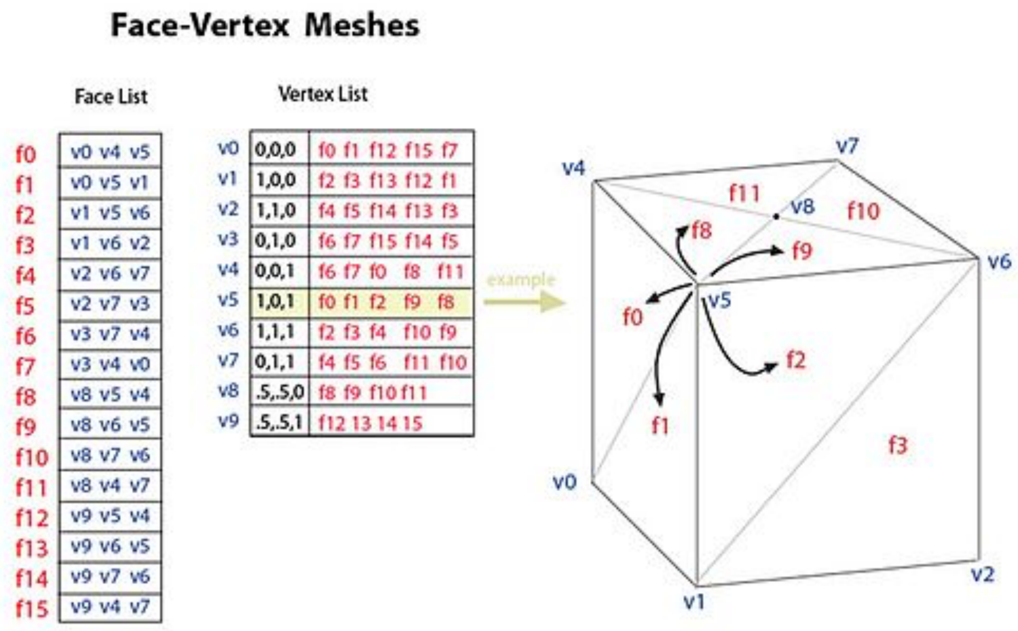
- Gives upper bounds on # of edges and faces



Data structure again

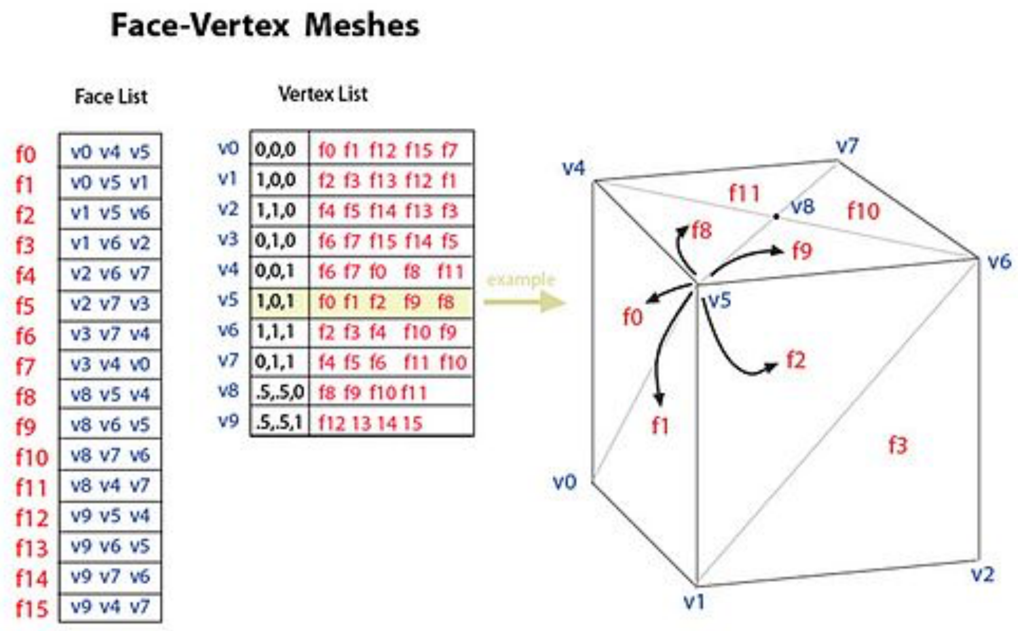
- Face—vertex representation

- What can you find easily?



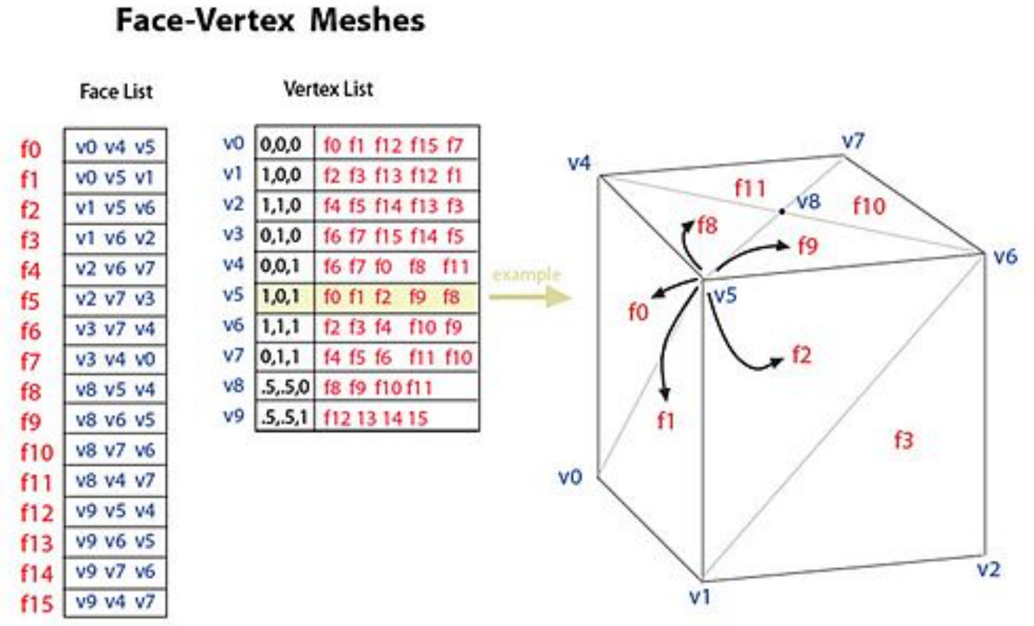
Data structure again

- Face—vertex representation
- What can you find easily?
    - Traverse vertices on face
    - Traverse faces from vertex
- What's hard to find?



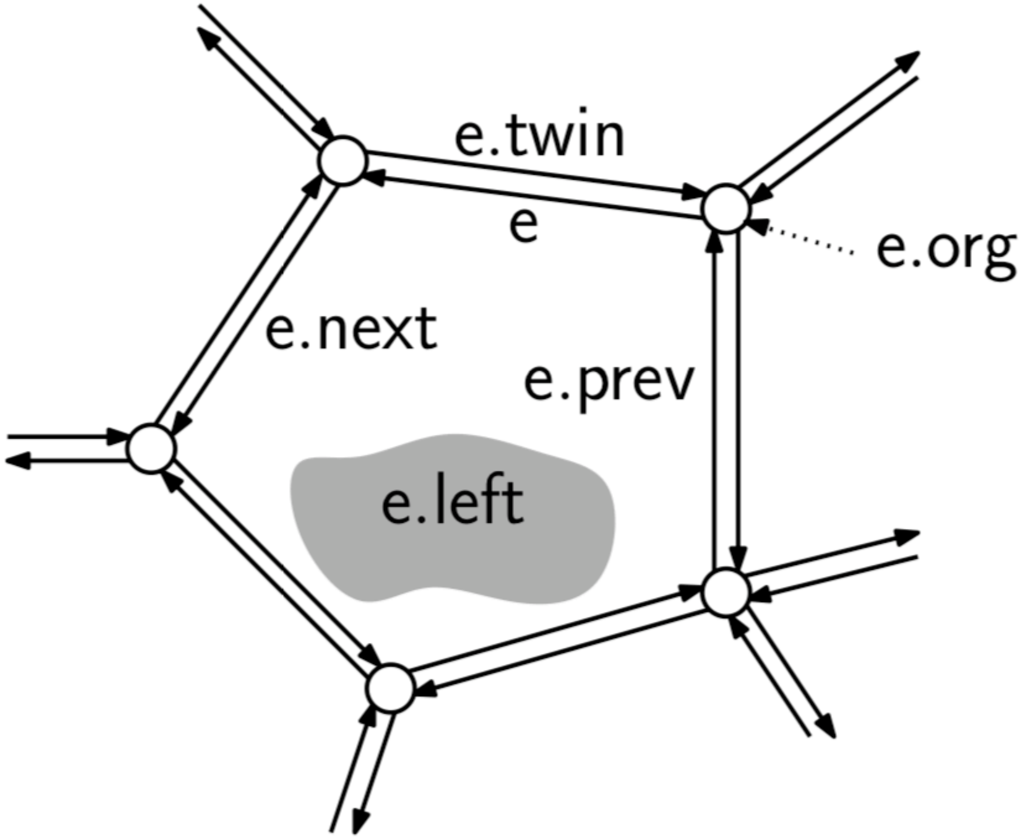
Data structure again

- Face—vertex representation
- What can you find easily?
  - Traverse vertices on face
  - Traverse faces from vertex
- What’s hard to find?
  - Adjacent faces?
  - Traverse vertices nearby systematically



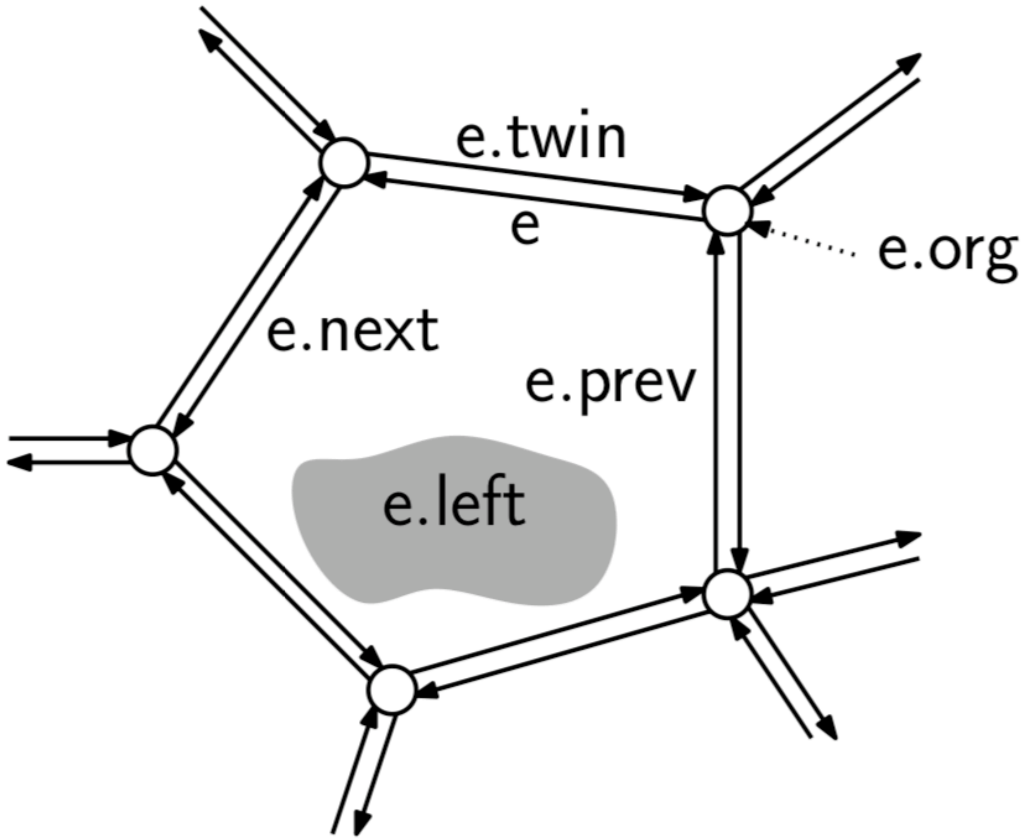
Winged edge representations

- DECL - doubly-connected edge list
- Stores directed half-edges
- Flexible, supports easier updates



Winged edge representations

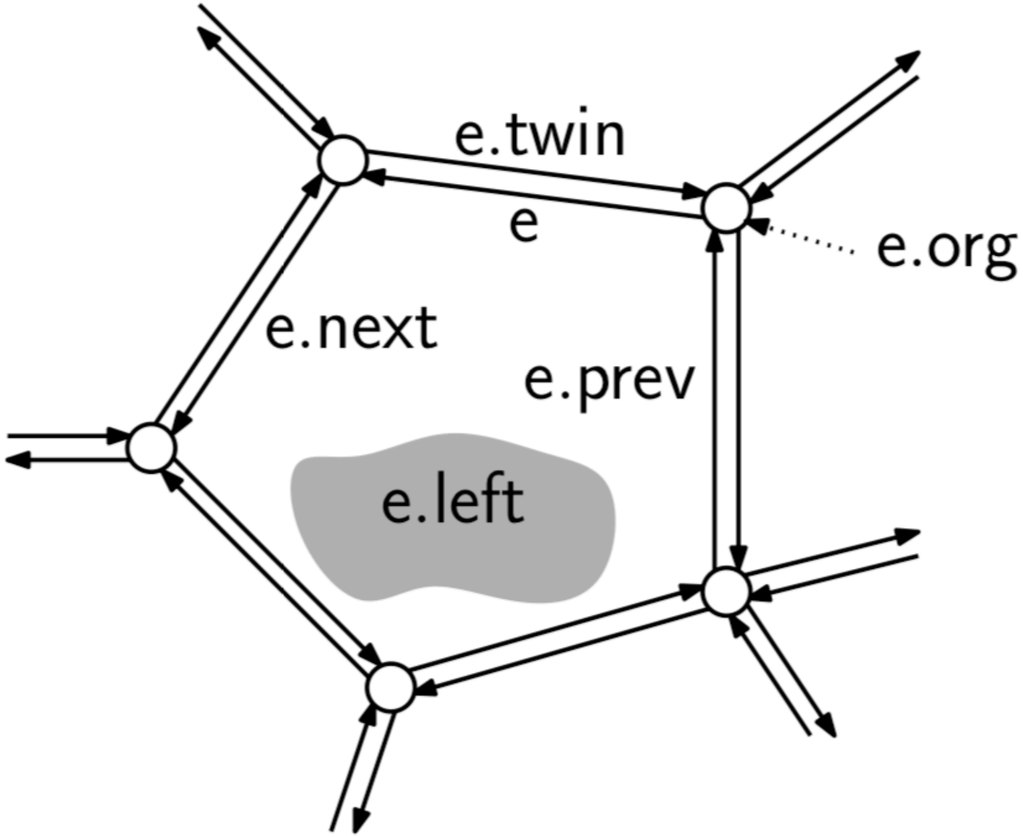
- Vertex v has coordinates plus one link to incident edge
- Face f has link to one half edge
- Edge (origin u, destination v) has
- *e.org*: e’s origin
- *e.twin*: e’s opposite twin half-edge
- *e.left*: the face on e’s left side
- *e.next*: the next half-edge after e in counterclockwise order about e’s left face
- *e.prev*: the previous half-edge to e in counterclockwise order about e’s left face (that is, the next edge in clockwise order).



Winged edge representations

- What is …

- e.dest: e’s destination vertex

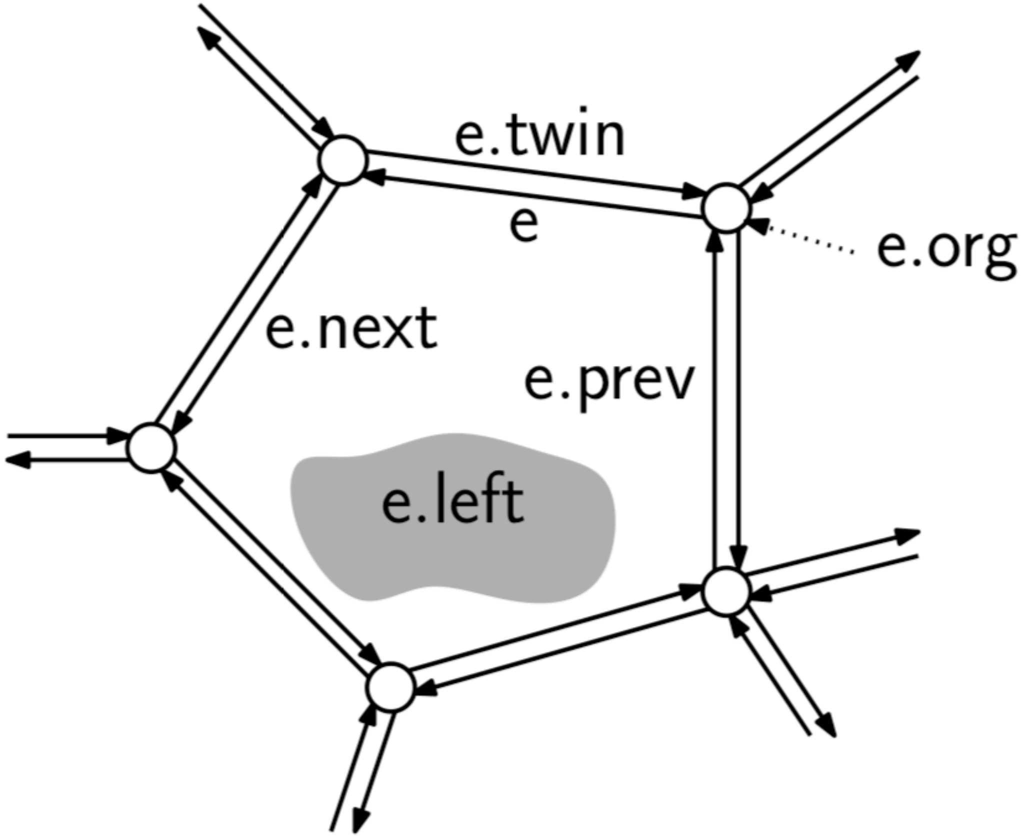


Winged edge representations

- What is …

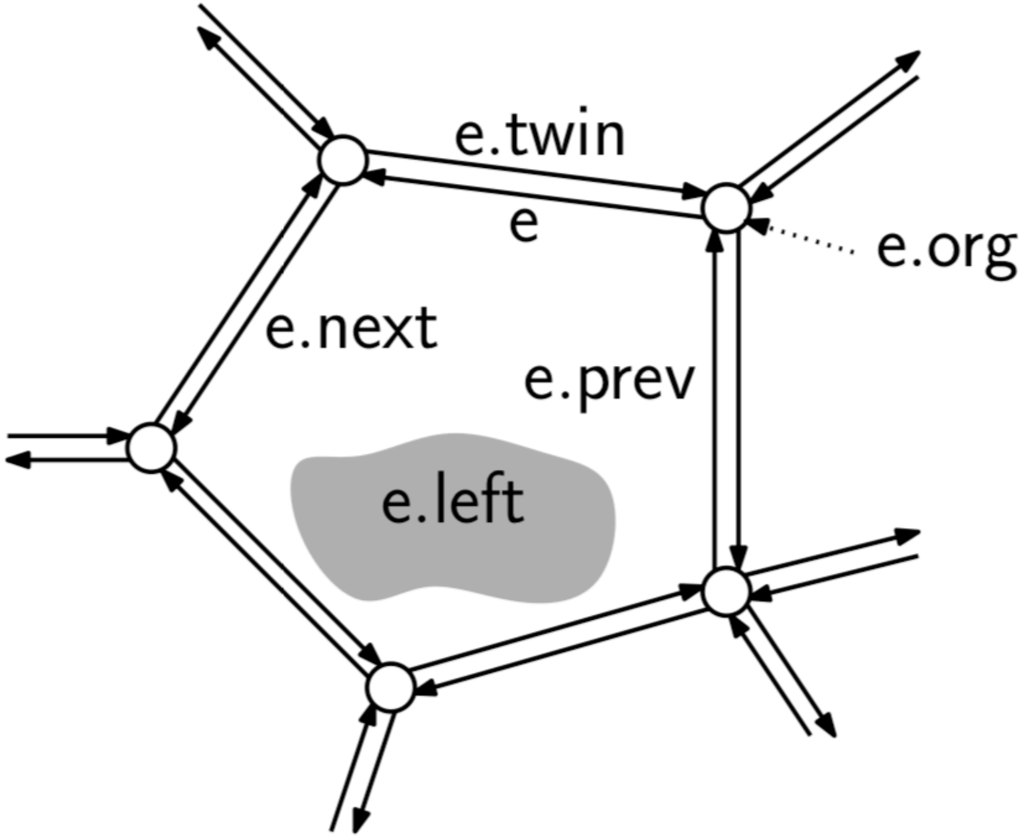
- e.dest: e’s destination vertex

  e.dest ← e.twin.org



Winged edge representations

- What is …
- e.right: the face on e's right side

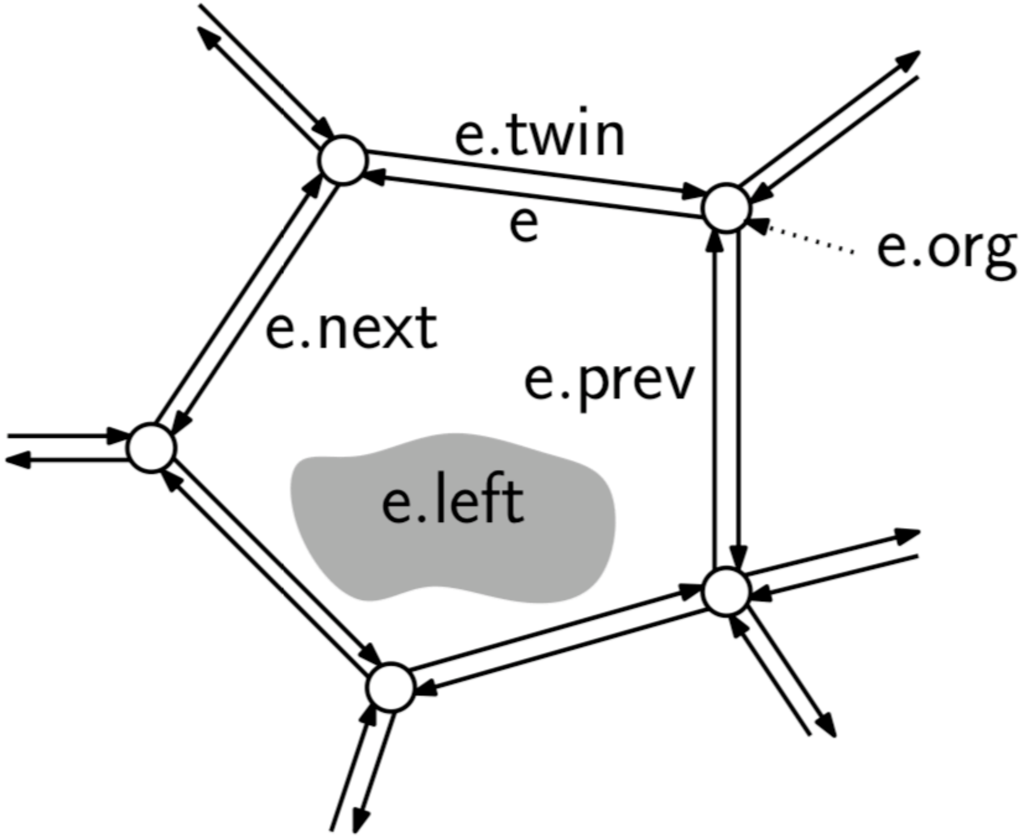


Winged edge representations

- What is …

- e.right: the face on e's right side

  e.right ← e.twin.left

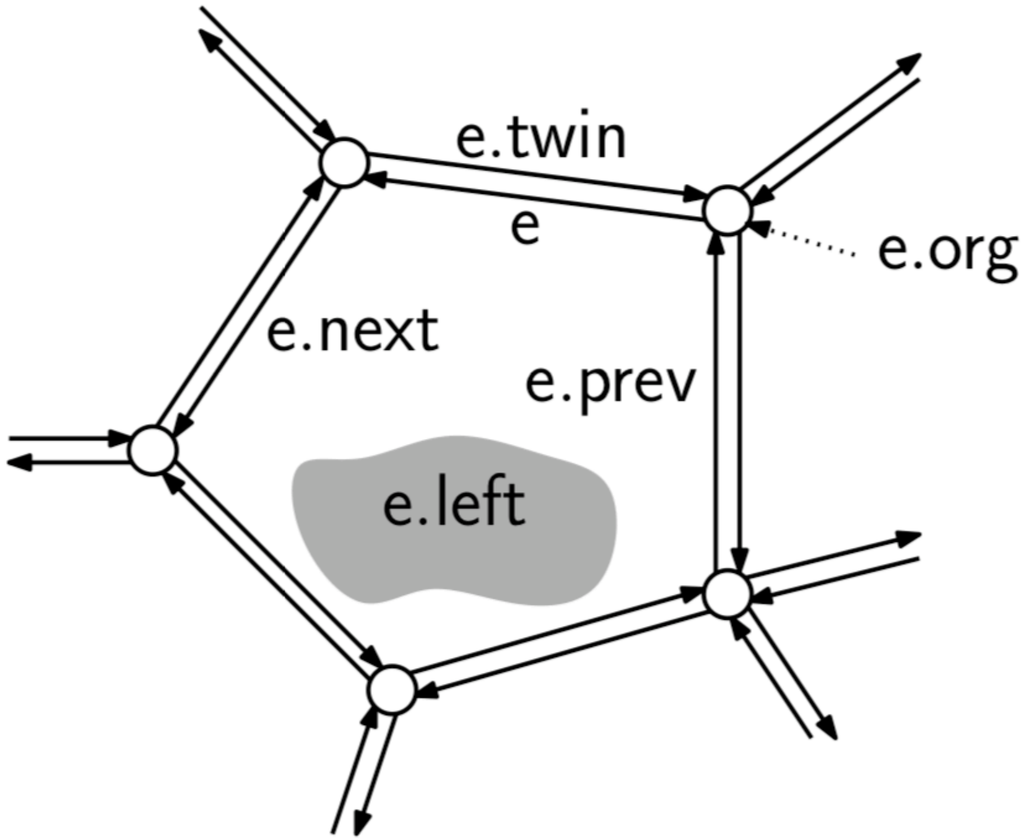


Winged edge representations

- What is …

- e.onext: the next half-edge that shares e’s origin that comes after e in clock-wise order

e.onext ← e.prev.twin

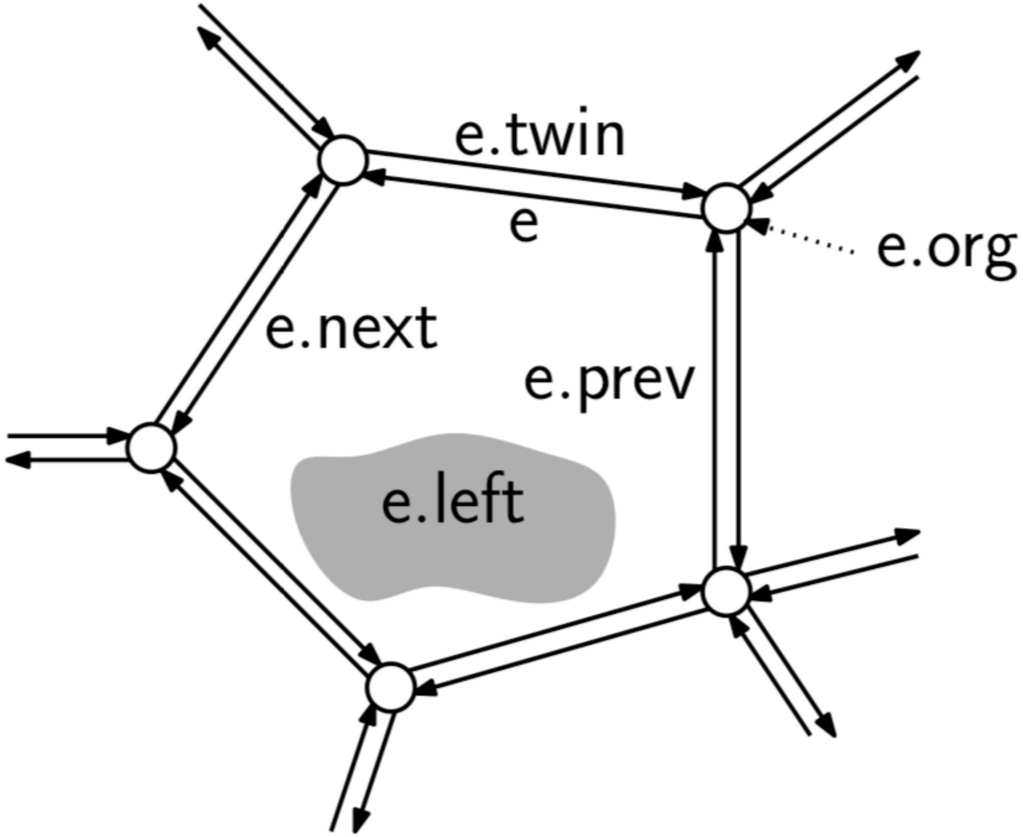


Winged edge representations

- What is …

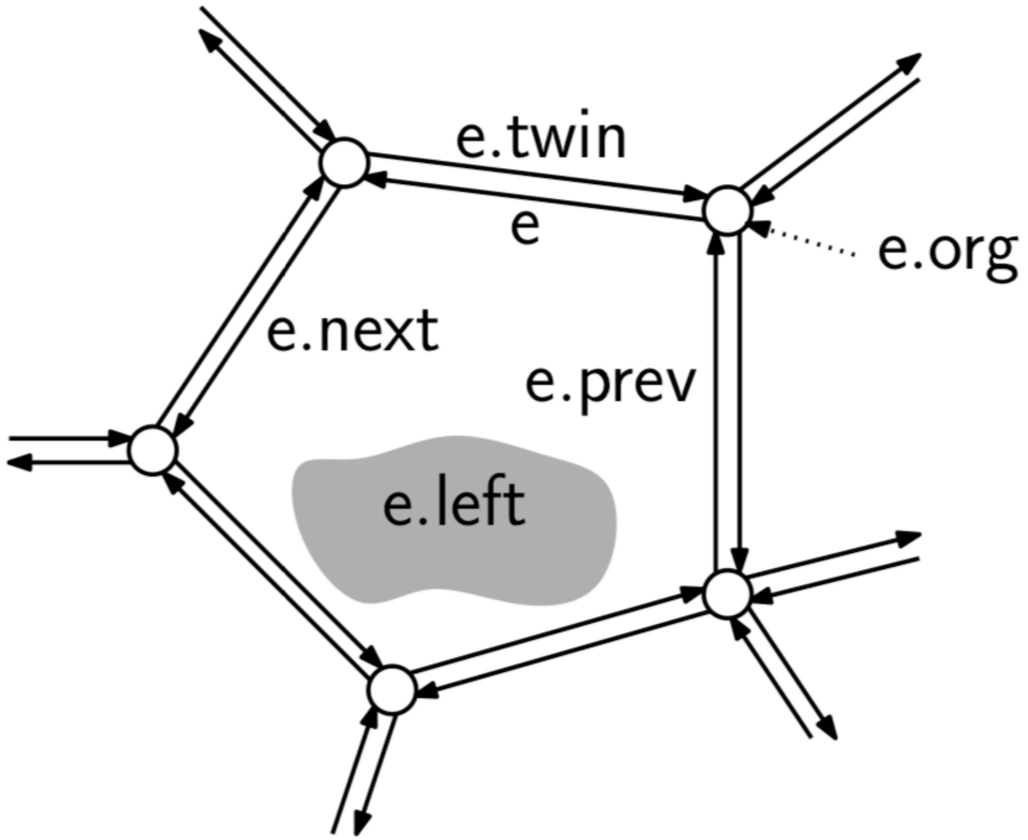
- the previous half-edge that shares e's origin that comes before e in clockwise order

e.oprev ← e.twin.next



Winged edge representations

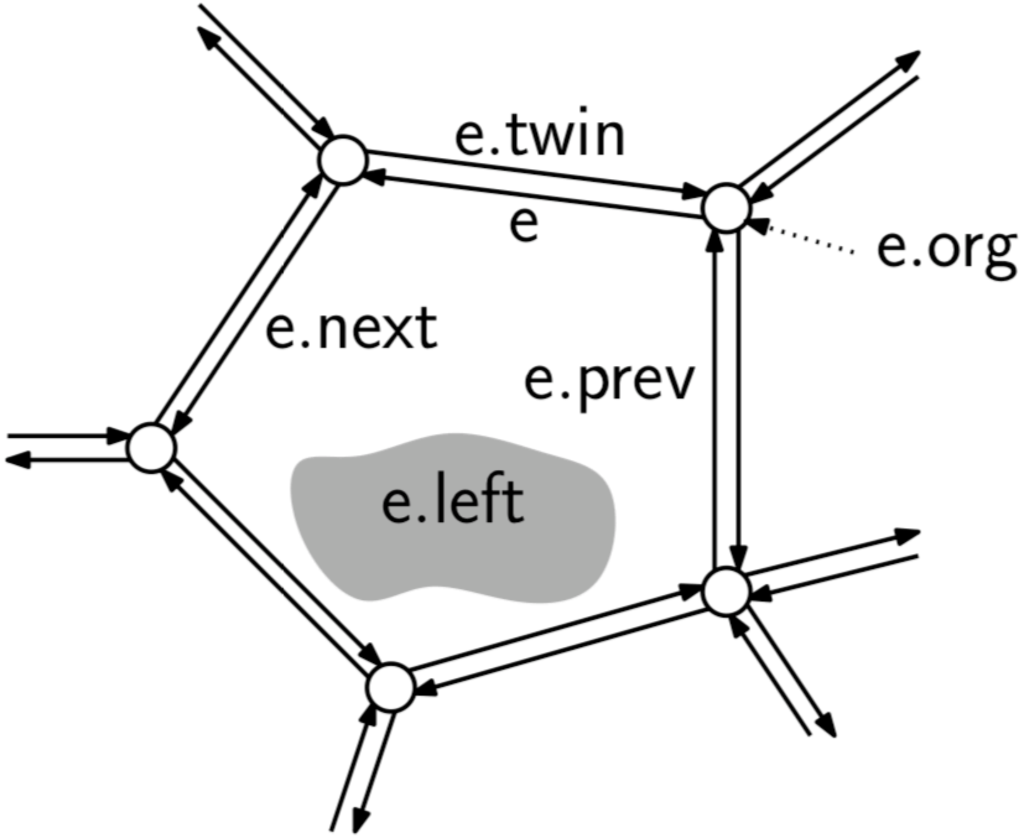
- Question: how traverse f in ccw order?



Winged edge representations

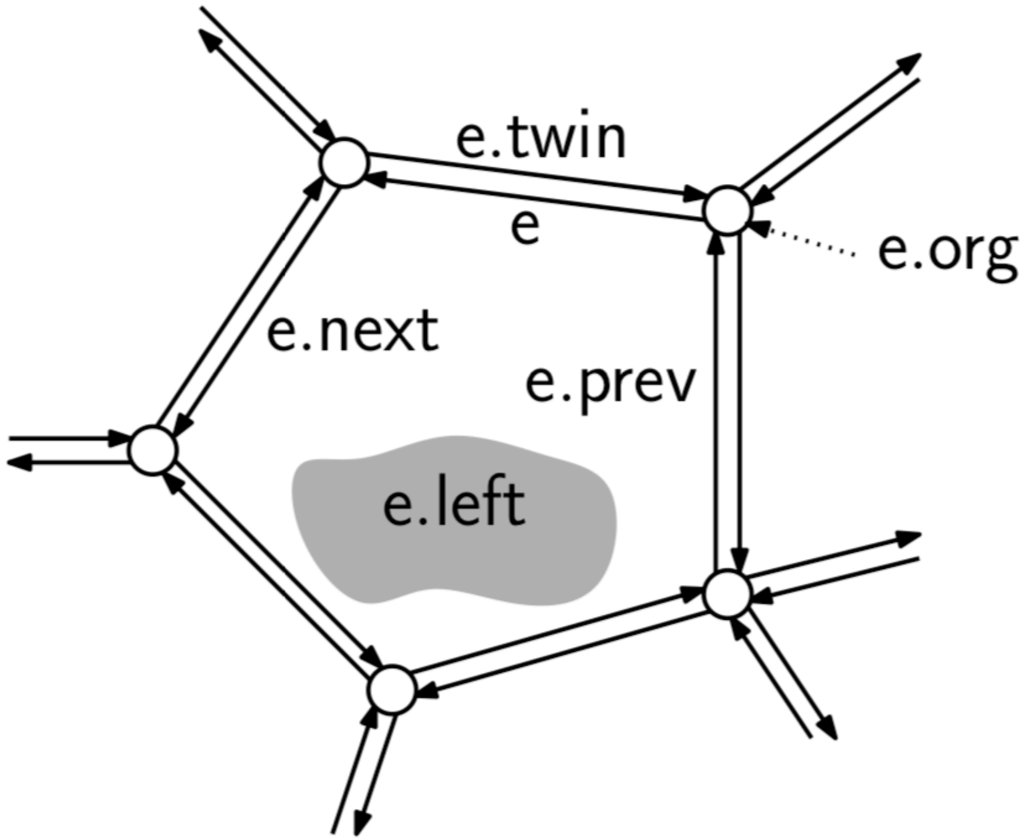
- Question: how traverse f in ccw order?

```
faceVerticesCCW(Face f) {
    Edge start = f.incident;
    Edge e = start;
    do {
        output e.org;
        e = e.next;
    } while (e != start);
}
```



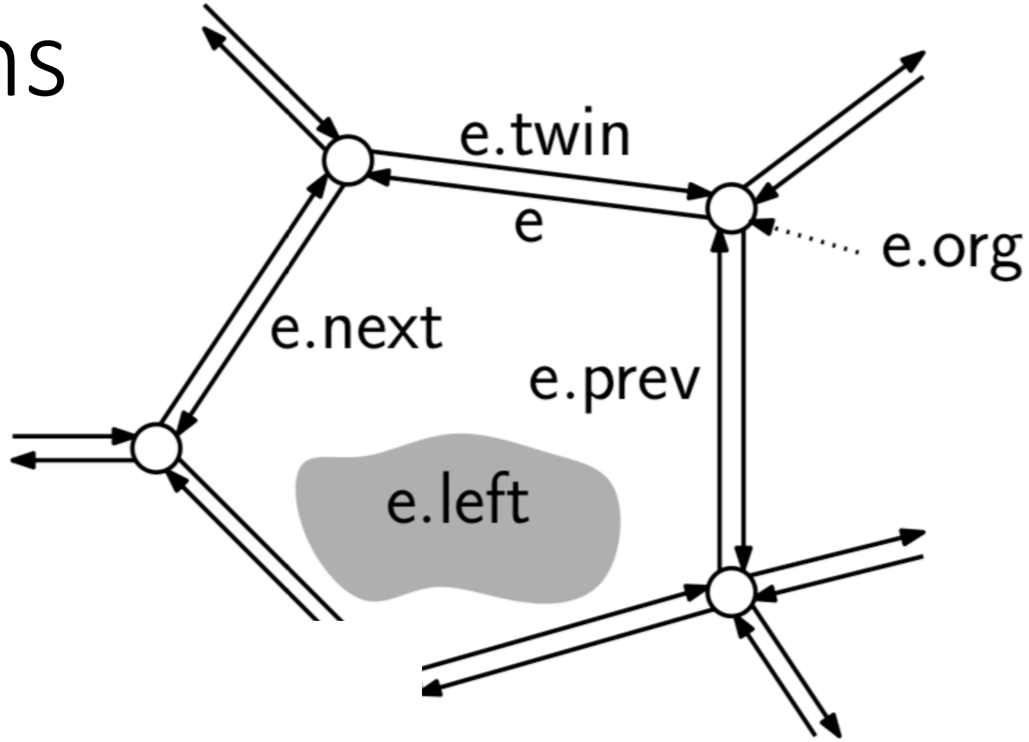
Winged edge representations

- Question: how traverse all vertices that are neighbors of v in cw order?



Winged edge representations

- Question: how traverse all vertices that are neighbors of v in cw order?




```
vertexNeighborsCW(Vertex v) {
    Edge start = v.incident;
    Edge e = start;
    do {
        output e.dest; // formally: output e.twin.org
        e = e.oprev; // formally: e = e.twin.next
    } while (e != start);
}
```

In class exercise

Given vertex v in a cell complex of a 2-manifold, the *link* of v is defined to be the edges that bound the faces that are incident to v, excluding the edges that are incident to v itself. Present a procedure (in pseudocode) that, given a vertex v of a DCEL, returns a list L consisting of the half edges of v's link ordered counterclockwise about v. For example, in the figure below, a possible output would be $\langle e_1, \ldots, e_{11} \rangle$. (Any cyclic permutation would be correct.)