Introducing Assignment 2: Mesh Processing & Half Edges

COS 426: Computer Graphics (Spring 2020)

Sahan Paliskara, Zheng Shi, Will Sweeny

Agenda

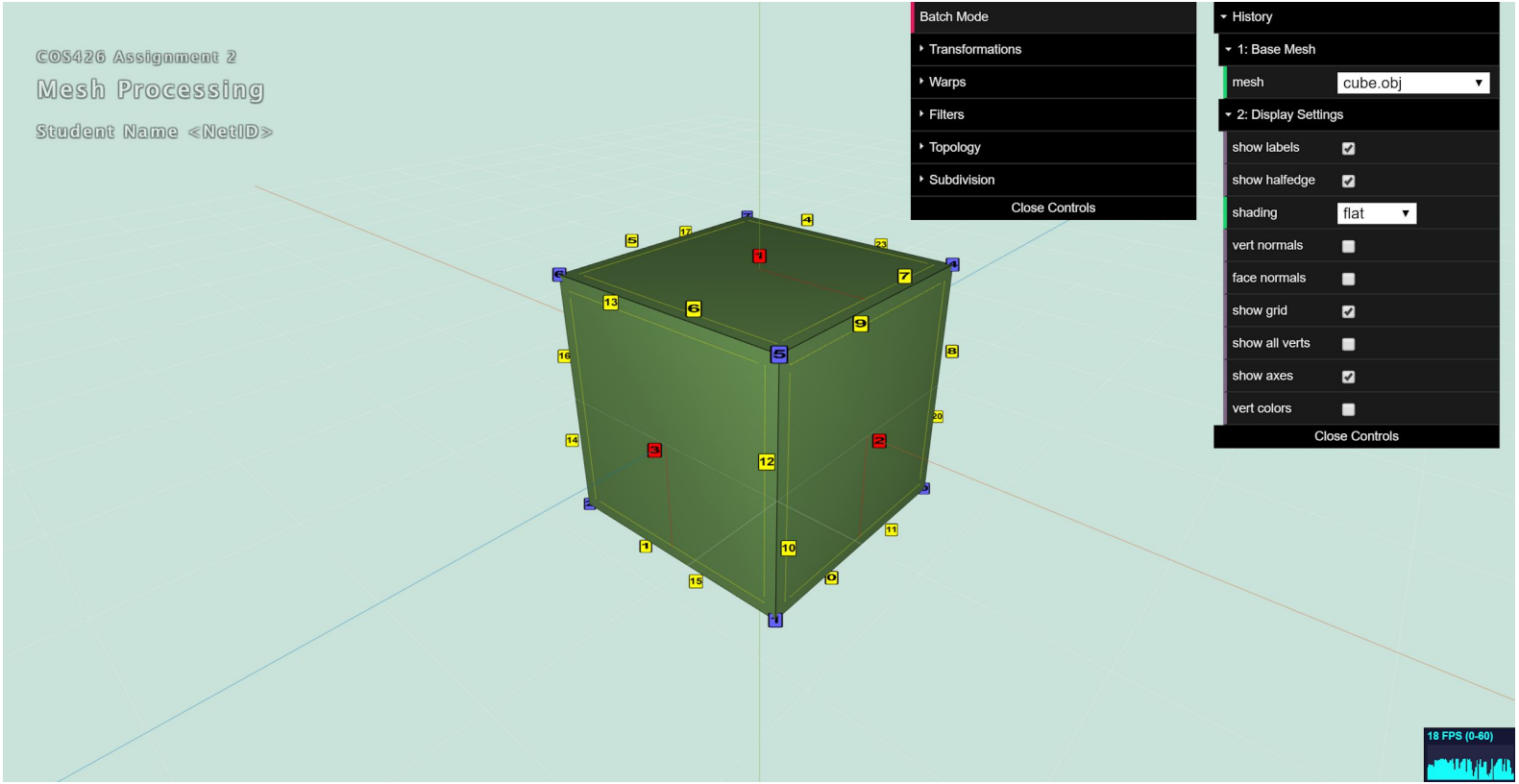
- Please fill out A1 Feedback Form!
- Change in OH Schedule
- Brief overview of A2
- Half-edge data structure
  - Definition
  - Traversal
  - Modification

Setup

Same as in A0 and A1:

- Run “`python3 -m http.server`” (or similar) inside the assignment directory
- Open “`http://localhost:8000`” in web browser

GUI


COS426 Assignment 2
Mesh Processing
Student Name <NetID>
Batch Mode
Transformations
Warps
Filters
Topology
Subdivision
Close Controls
History
1: Base Mesh
mesh
cube.obj
2: Display Settings
show labels
show halfedge
shading
flat
vert normals
face normals
show grid
show all verts
show axes
vert colors
Close Controls
18 FPS (0-60)

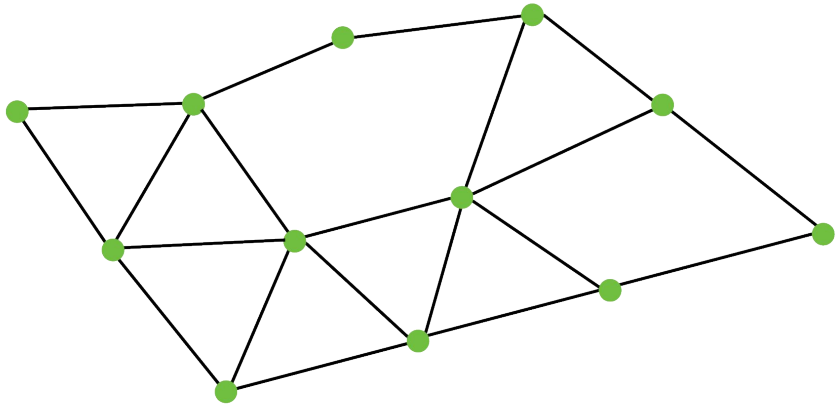
Tips for Three.js and A2

- For A2 you will be using the Three.js library
  - Simple and efficient primitives for working in 3D
- You should **read the docs!**
  - Vector3
  - Euler (for rotations)
- Modularity is your friend!
  - You will be writing helper functions. Use them!

Meshes vs. Images

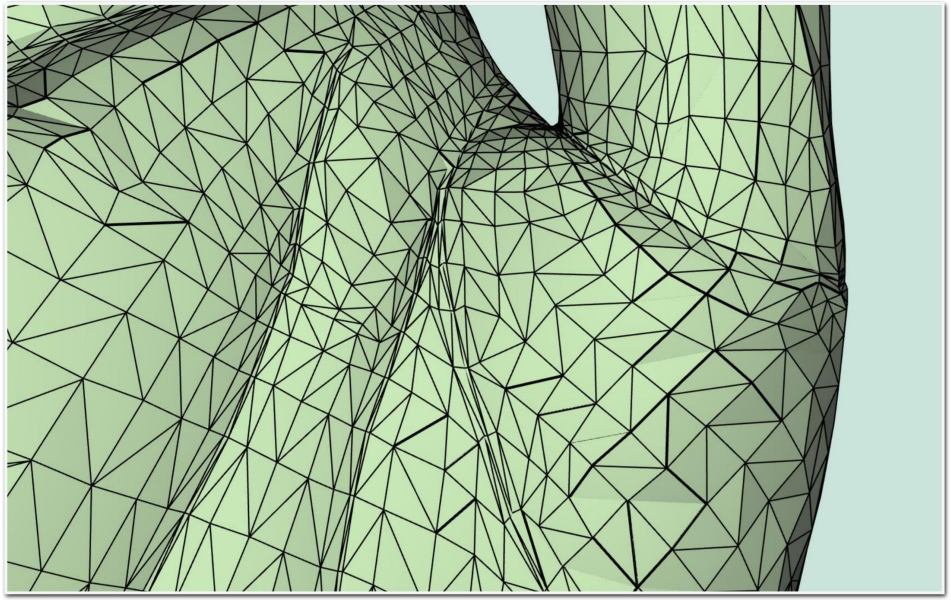
- Images have implicit adjacency information
  - Window around a pixel
  - Easy to express local operations
    - (e.g. convolution)

- What about meshes?
  - How to apply smoothing?



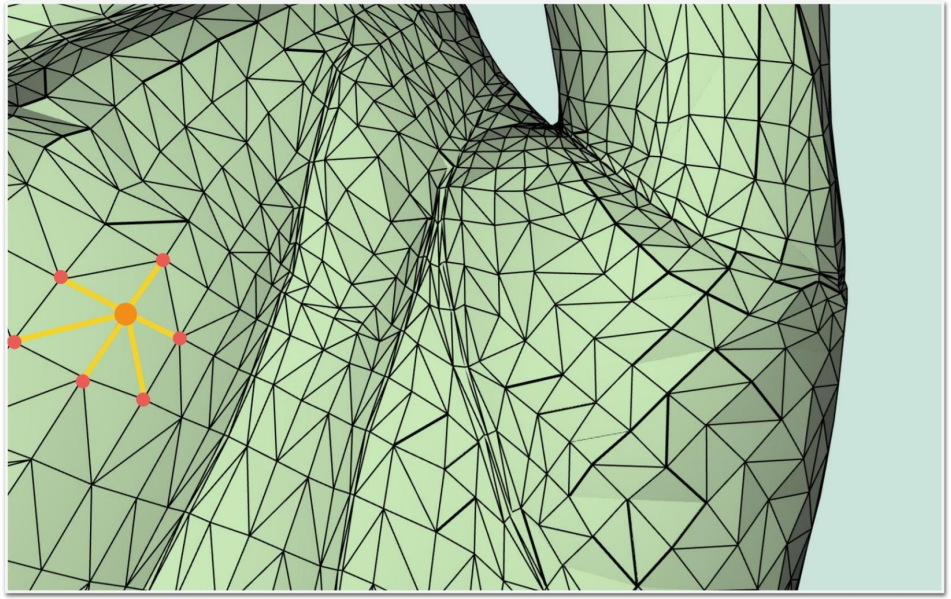
Meshes

- Meshes can be quite dense



Meshes

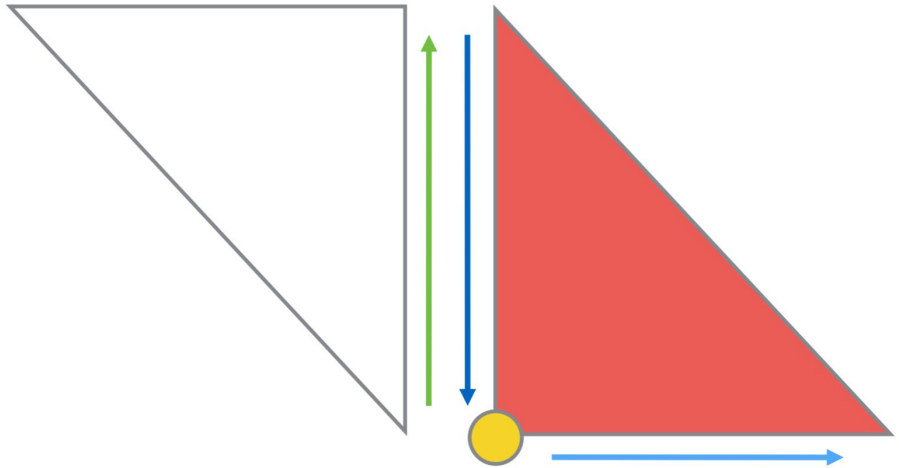
- How can we efficiently access adjacency information?



One - Ring Neighborhood

What is a Half-Edge?

- Imagine splitting each edge in two
  - Each half gets one of the edge's faces
  - Each face, vertex, and half-edge stores some state
  - Conceptually very similar to doubly linked list



Half-Edge: What State is Stored?

Half Edge	Vertex	Face
Vertex	Position	Half-Edge
Opposite Half-Edge	Outgoing Half-Edge	
Face		
Next Half-Edge		

Half-Edge Data Structure

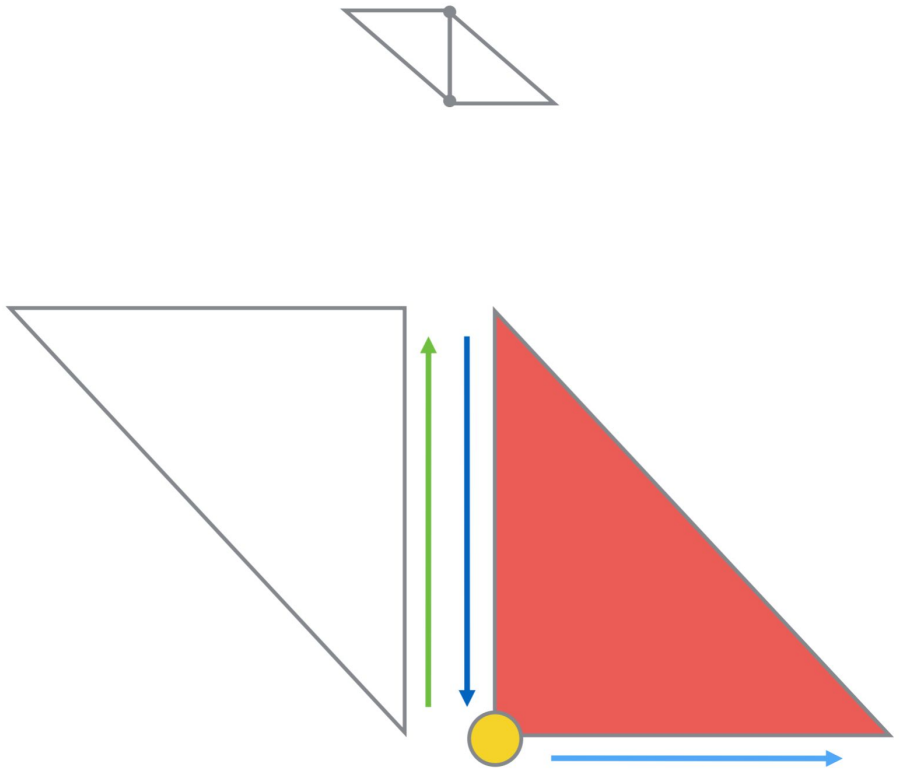
Half Edge

Vertex

Opposite Half-Edge

Face

Next Half-Edge

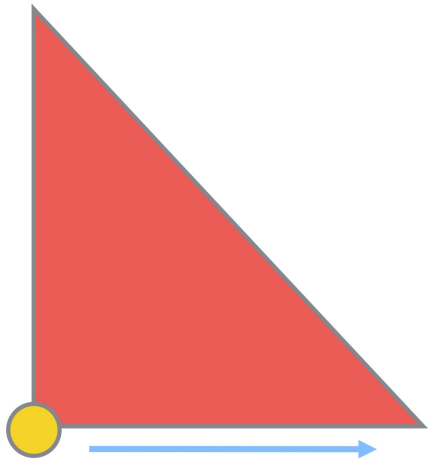


Half-Edge Data Structure

Vertex

Location

Outgoing Half-Edge

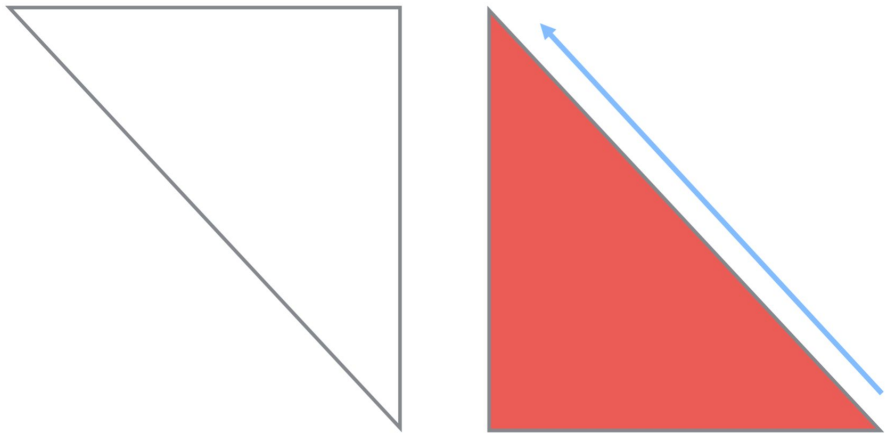


Q: Which half-edge to choose?
A: Pick one arbitrarily

Half-Edge Data Structure

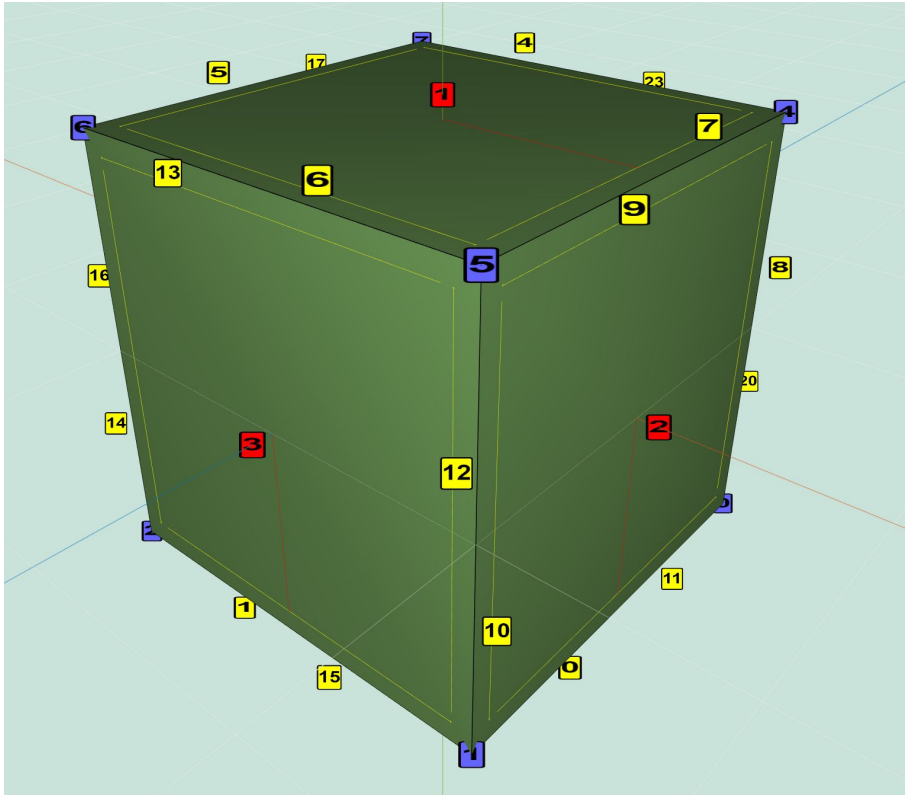
Face

Half-Edge



Q: Which half-edge to choose?
A: Pick one arbitrarily

Half-Edge Visualization



- Faces: **id**
- Half-edges: **id**
- Vertices: **id**

Features

Transformations
- Translation
- Rotation
- Scale

Traversal
- **Various edge/vertex/face helpers**

Analysis
- Face Area
- Per-vertex Normals
- Average Edge Lengths

Warps
- Twist
- Inflate
- Wacky

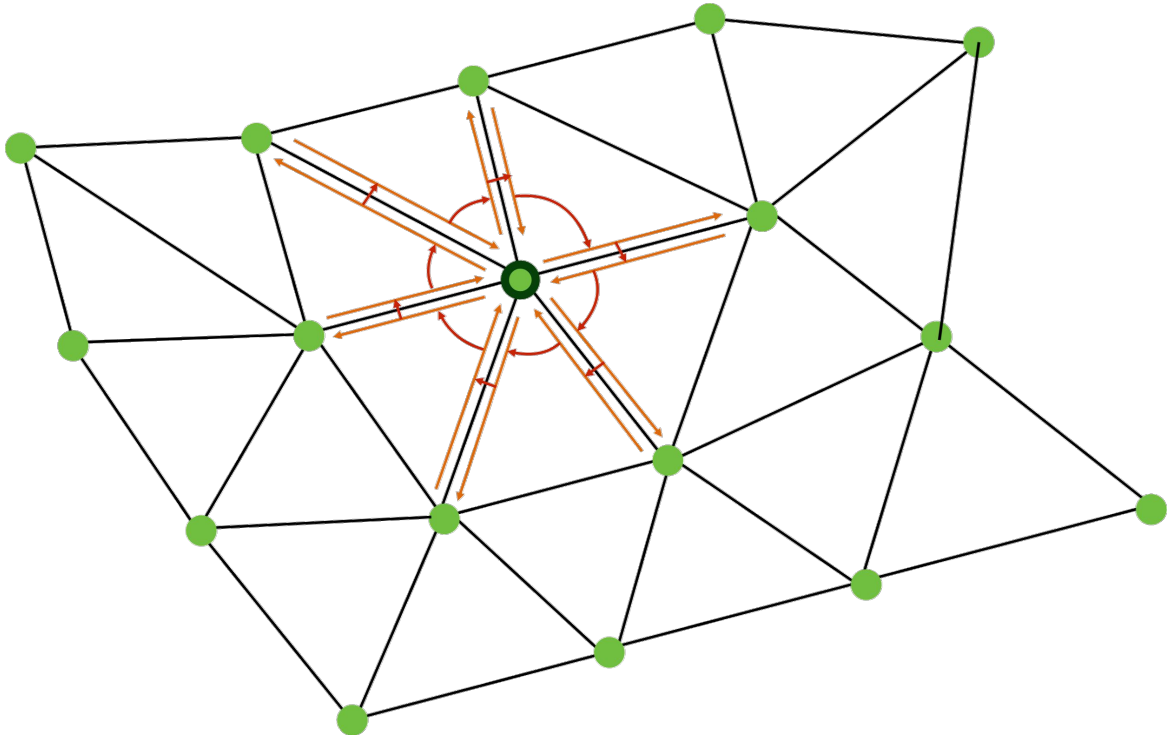
Filters
- Noise
- Smoothing
- Sharpening
- Curvature

Topology
- Triangulate
- Truncate
- Extrude
- Split Long Edges

Subdivision
- Triangle Topology
- Loop
- Quad Topology
- Catmull-Clark

Traversal (Vertices on vertex)

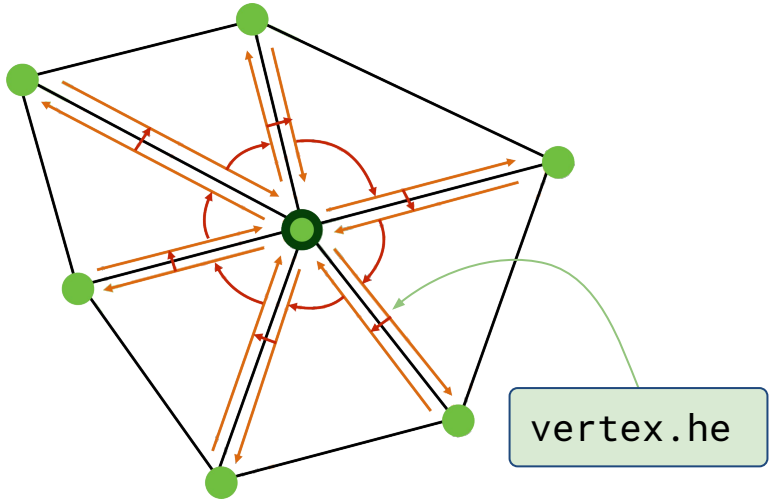
- How do we get one-ring neighbors of a vertex?



Traversal (Vertices on vertex)

- How do we get one-ring neighbors of a vertex?

```
original_he = vertex.he;
he = original_he;
do {
    // some calculations
    he = he.opposite.next
} while (he != original_he)
```



Traversal (Vertex Normals)

- Vertex Normals are defined as a weighted average of the normals of adjacent faces (weighted by face area)
- How would you compute vertex normals given face normals and areas?

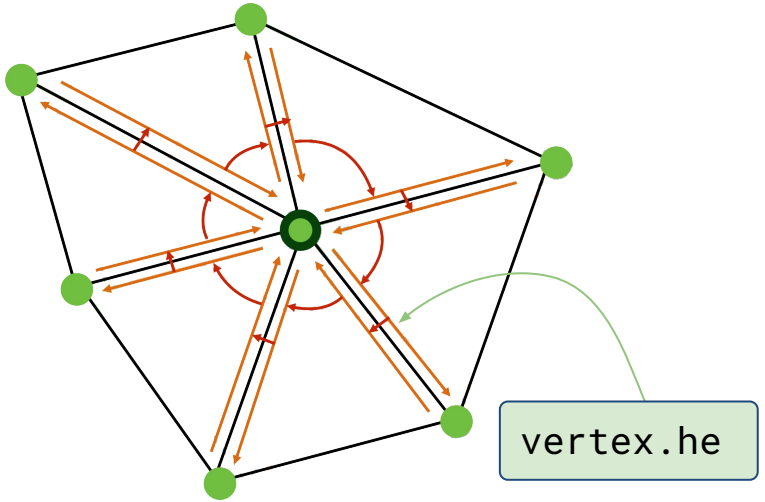
Half Edge

Vertex

Opposite Half-Edge

Face

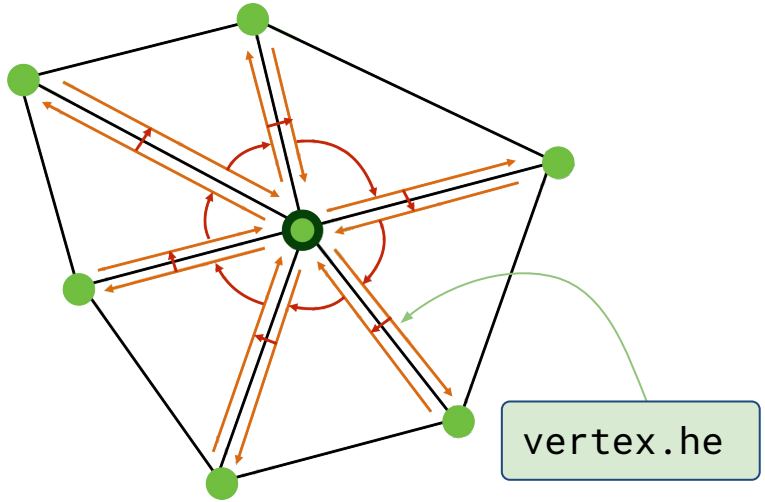
Next Half-Edge



Traversal (Vertex Normals)

- Vertex Normals are defined as a weighted average of the normals of adjacent faces (weighted by face area)
- How would you compute vertex normals given face normals and areas?

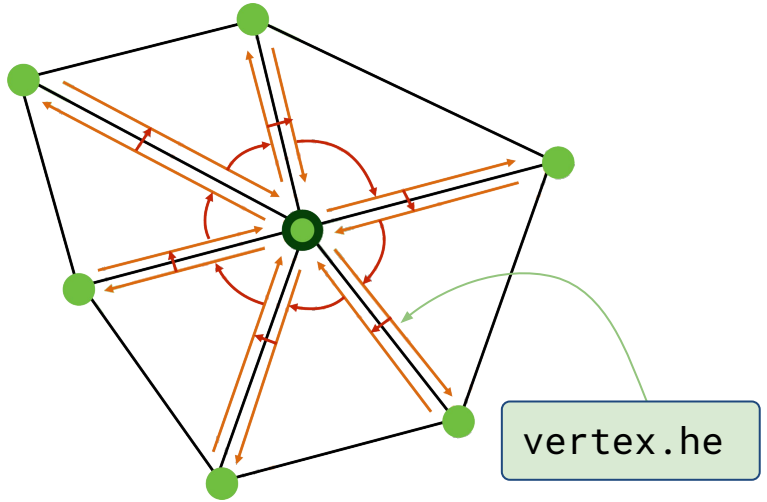
```
original_he = vertex.he;
he = original_he;
do {
    // some calculations
    he = he.opposite.next
} while (he != original_he)
```



Traversal (Vertex Normals)

- Vertex Normals are defined as a weighted average of the normals of adjacent faces (weighted by face area)
- How would you compute vertex normals given face normals and areas?

```
original_he = vertex.he;
he = original_he;
v_normal.set(0,0,0);
do {
    f_normal = he.face.normal;
    area = he.face.area;
    v_normal.add(f_normal*area);
    he = he.opposite.next
} while (he != original_he)
v_normal.normalize()
```

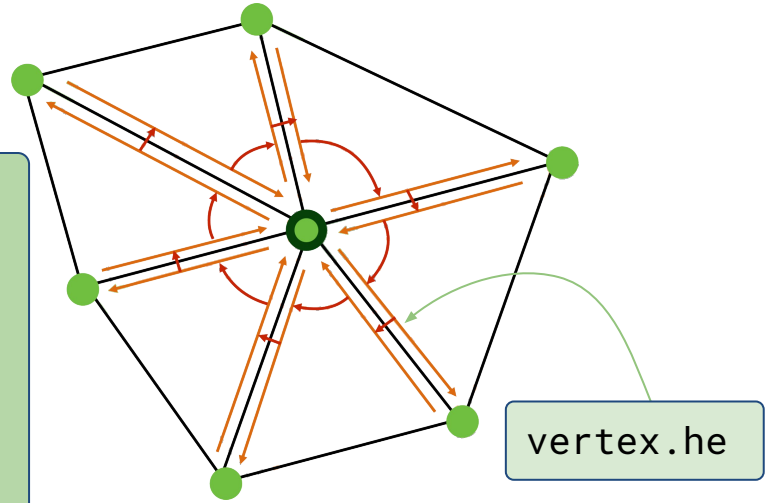


Traversal (Vertex Normals)

- Vertex Normals are defined as a weighted average of the normals of adjacent faces (weighted by face area)
- How would you compute vertex normals given face normals and areas?

Easier way: use facesOnVertex()!

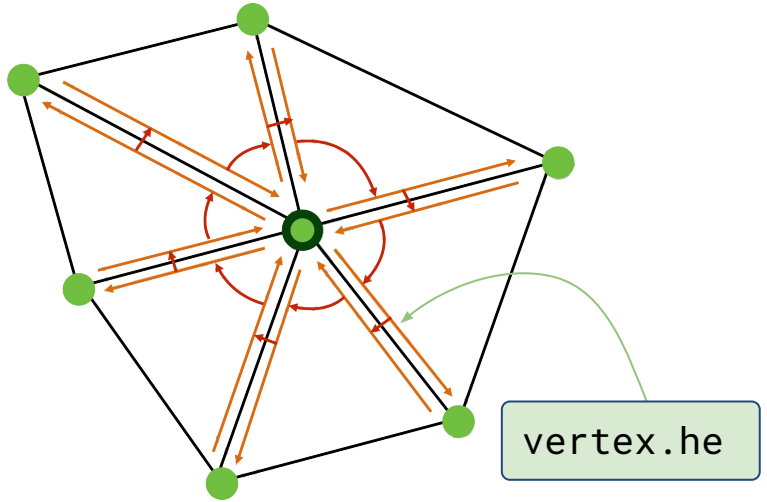
```
fs = mesh.facesOnVertex(v);
v_normal.set(0,0,0);
for (let f of fs) {
    v_normal.add(f.normal * f.area);
}
v_normal.normalize()
```



Traversal (Laplacian Smoothing)

- Similarly, in uniform Laplacian smoothing, each vertex moves towards the average of it and its neighbors.

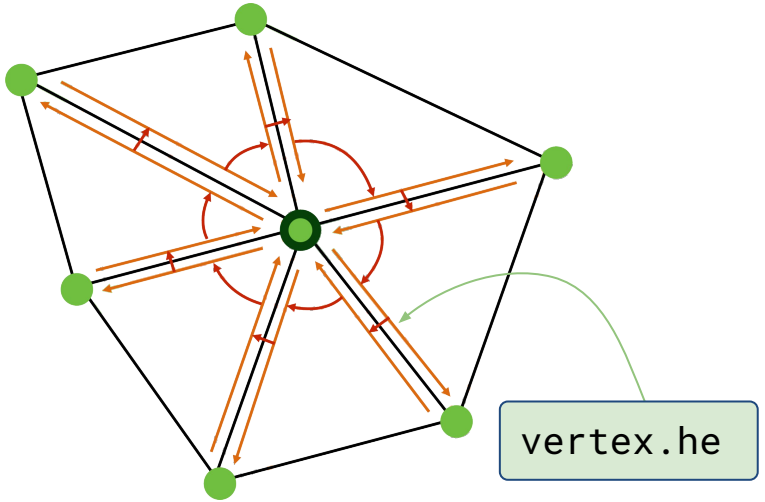
```
original_he = vertex.he;
he = original_he;
do {
    // some calculations
    he = he.opposite.next
} while (he != original_he)
```



Traversal (Laplacian Smoothing)

- Similarly, in uniform Laplacian smoothing, each vertex moves towards the average of it and its neighbors.

```
original_he = vertex.he;
he = original_he;
avg_pos.set(0,0,0);
do {
    avg_pos.add(he.vertex.pos);
    he = he.opposite.next
} while (he != original_he)
avg_pos.add(-vertex*num_neigh);
new_pos = vertex + avg_pos * delta;
```



Traversal (Laplacian Smoothing)

- Some tips for uniform Laplacian smoothing:
  - You can use `verticesOnVertex()` to simplify your code!
  - Be careful not to modify your mesh before you've computed offsets for all vertices!
    - (Similar to filters in A1 that modified the image)

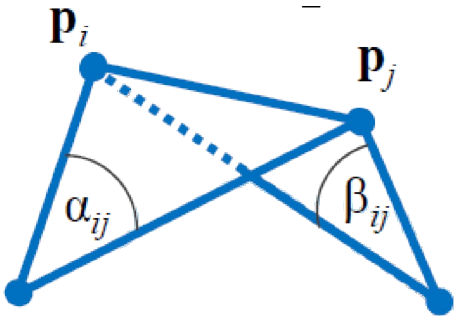
Traversal (Cotan Laplacian Smoothing)

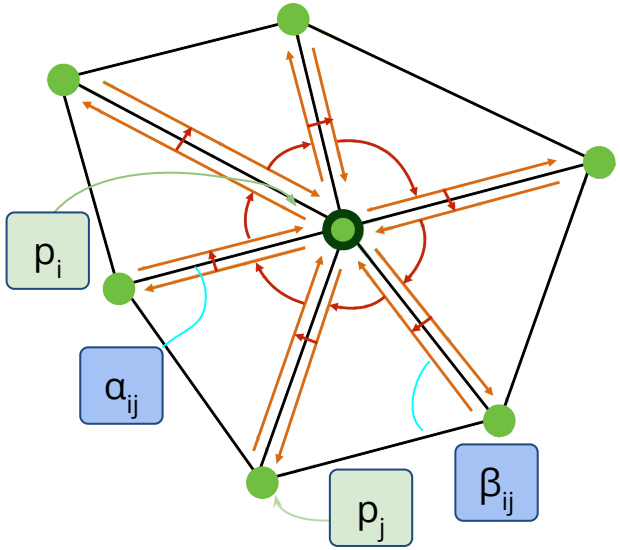
- Cotangent Laplacian smoothing

```
avg_pos.add(he.vertex.pos); ⇒ avg_pos.add(w*he.vertex.pos);
num_neigh                   ⇒ total_w
```

- Notes:
  - p_i = center vert
  - Iterate over all neighboring p_j
  - p_i , p_j will share two faces
  - α_{ij} , β_{ij} are the far angles on these faces

$$w = \frac{\cot(\alpha_{ij}) + \cot(\beta_{ij})}{2}$$





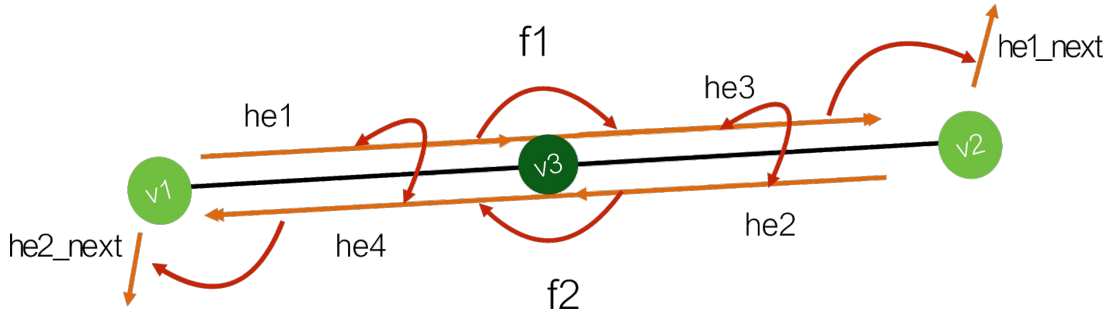
Data Structure Modification

- Take a look meshUtils.js for the all the primitives
  - splitEdgeMakeVert()
  - joinEdgeKillVert()
  - splitFaceMakeEdge()
  - joinFaceKillEdge()

Data Structure Modification (splitEdge)

How to add new vertices to an existing half-edge data structure?

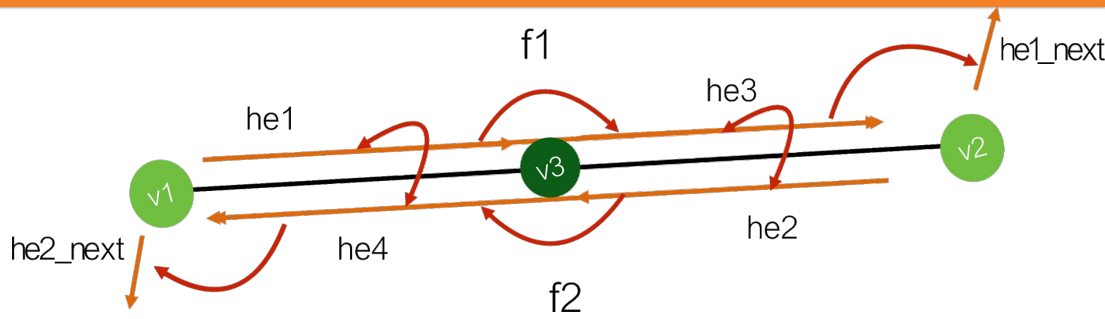
splitEdgeMakeVert(v1,v2,factor):



Data Structure Modification (splitEdge)

splitEdgeMakeVert(v1,v2,factor):

- addVertex
- addHalfEdge
- he.vertex, he.opposite




```
v3 = addVertex(v1.pos.lerp(v2.pos, factor));
```

`he3 = addHalfEdge(v3,v2,f1);` `he4 = addHalfEdge(v3,v1,f2);`	`he1.vertex = v3;` `he2.vertex = v2;`

`he1.next = he3;`
`he2.next = he4;`
`he3.next = he1_next;`
`he4.next = he2_next;`
	`he1.opposite = he4;`
	`he4.opposite = he1;`
	`he2.opposite = he3;`
	`he3.opposite = he2;`

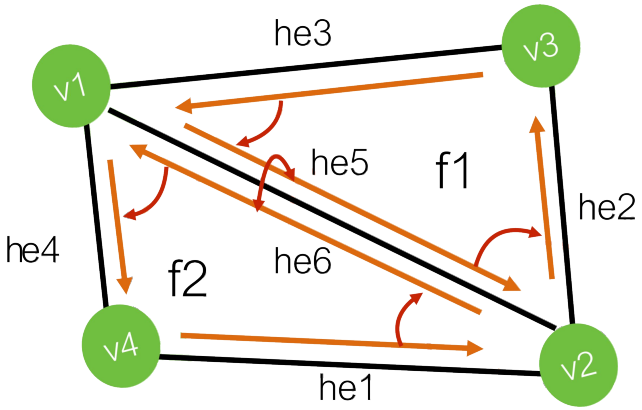
Data Structure Modification (splitFace)

How to add new edges to an existing half-edge data structure?

splitFaceMakeEdge(f, v1, v2, vertOnF, switchFaces)

```
f2 = addFace();

he5 = addHalfEdge(v1,v2,f1);
he6 = addHalfEdge(v2,v1,f2);

he5.opposite = he6;
he6.opposite = he5;
```

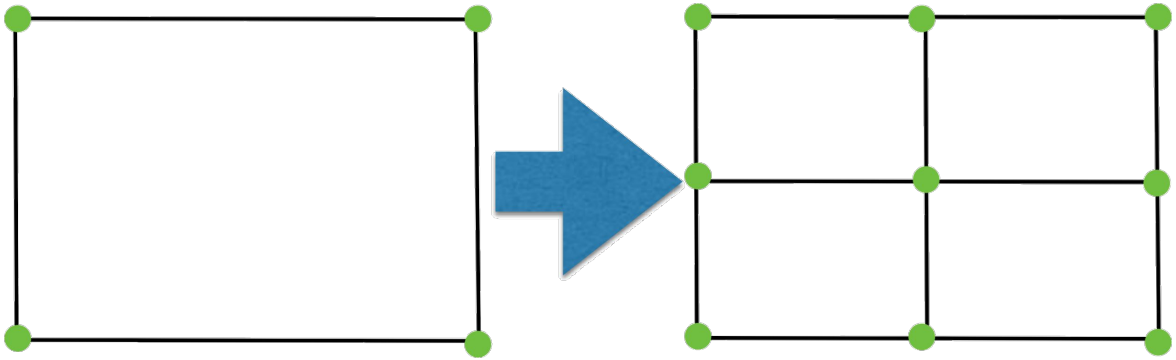
```
he5.next = he2;
he3.next = he5;
he1.next = he6;
he6.next = he4;

f1.halfedge = he5;
f2.halfedge = he6;
```



- Optional args: (for advanced filters, like Extrude)
  - vertOnF: if provided, this vert will still be on the original face
  - switchFaces: if true, vertOnF is placed on the new face instead

Data Structure Modification (subdividing)

- How would you go about subdividing a quad face?
  - You're given *split edge* and *split face*
  - Just use those - guaranteed validity of mesh after use!



Data Structure Modification(subdividing)