Introducing Assignment 2:
Mesh Processing & Half Edges
Agenda

● 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
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?
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?

<table>
<thead>
<tr>
<th>Half Edge</th>
<th>Vertex</th>
<th>Face</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vertex</td>
<td>Position</td>
<td>Half-Edge</td>
</tr>
<tr>
<td>Opposite Half-Edge</td>
<td>Outgoing</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Half-Edge</td>
<td></td>
</tr>
<tr>
<td>Face</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Next Half-Edge</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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

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

- Faces:
- Half-edges:
- Vertices:
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?
How do we get one-ring neighbors of a vertex?

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

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

```cpp
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()
```
Traversing (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()!

```javascript
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()
```
Traversals (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)
Traversing (Laplacian Smoothing)

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

```java
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);  \Rightarrow  avg_pos.add(w*he.vertex.pos);
num_neigh  \Rightarrow  total_w
```

- Notes:
  - $p_i$ = center vert
  - Iterate over all neighboring $p_j$
  - $p_i$, $p_j$ will share two faces
  - $\alpha_{ij}$, $\beta_{ij}$ 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 all the primitives
  - `splitEdgeMakeVert()`
  - `joinEdgeKillVert()`
  - `splitFaceMakeEdge()`
  - `joinFaceKillEdge()`
Data Structure Modification (splitEdge)

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

\texttt{splitEdgeMakeVert(v1,v2,factor):}
splitEdgeMakeVert(v1, v2, factor):

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

\[
v3 = \text{addVertex}(v1\text{.pos}.\text{lerp}(v2\text{.pos}, \text{factor}));
\]

\[
\begin{align*}
\text{he3} &= \text{addHalfEdge}(v3, v2, f1); \\
\text{he4} &= \text{addHalfEdge}(v3, v1, f2); \\
\text{he1}.\text{vertex} &= v3; \\
\text{he2}.\text{vertex} &= v2;
\end{align*}
\]

\[
\begin{align*}
\text{he1}.\text{next} &= \text{he3}; \\
\text{he2}.\text{next} &= \text{he4}; \\
\text{he3}.\text{next} &= \text{he1}_\text{next}; \\
\text{he4}.\text{next} &= \text{he2}_\text{next}; \\
\text{he1}.\text{opposite} &= \text{he4}; \\
\text{he4}.\text{opposite} &= \text{he1}; \\
\text{he2}.\text{opposite} &= \text{he3}; \\
\text{he3}.\text{opposite} &= \text{he2};
\end{align*}
\]
Data Structure Modification (splitFace)

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

\[
\text{splitFaceMakeEdge}(f, v1, v2, \text{vertOnF}, \text{switchFaces})
\]

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

\[
\begin{align*}
  f2 &= \text{addFace}(); \\
  \text{he5} &= \text{addHalfEdge}(v1,v2,f1); \\
  \text{he6} &= \text{addHalfEdge}(v2,v1,f2); \\
  \text{he5}.\text{opposite} &= \text{he6}; \\
  \text{he6}.\text{opposite} &= \text{he5}; \\
  \text{he5}.\text{next} &= \text{he2}; \\
  \text{he3}.\text{next} &= \text{he5}; \\
  \text{he1}.\text{next} &= \text{he6}; \\
  \text{he6}.\text{next} &= \text{he4}; \\
  \text{f1}.\text{halfedge} &= \text{he5}; \\
  \text{f2}.\text{halfedge} &= \text{he6};
\end{align*}
\]
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)

Split Edge Make Vert

Split Face Make Edge

SplitEdgeMakeVert

Split Face Make Edge