Indexed Mesh Data Structures

1. The Euler Formula

The Euler Formula states the following relationship for the elements of a closed and connected surface mesh:

\[ V - E + F = 2(1 - G) \]

- \( V \) is the number of vertices
- \( E \) is the number of edges
- \( F \) is the number of faces
- \( G \) is the genus of the surface (how holes/handles it has)

Show that for a triangle mesh with no holes we have \( F \approx 2V \). Hint: each face has 3 edges and each edge is shared by 2 faces.

If mesh is like a sphere, \( G = 0 \)

\[
\begin{align*}
V - E + F &= 2 \\
V - \frac{3F}{2} + F &= 2 \\
V &= 2 + \frac{1}{2} F \\
2V &\approx F
\end{align*}
\]

2. Memory Requirements

Using the fact that \( F \approx 2V \), compare the storage requirements for an indexed face mesh and a triangle soup (in WebGL this corresponds to using `gl.drawElements` versus `gl.drawArrays`). Assume the mesh has \( V \) vertices and a number requires 4 bytes of space. Derive functions for the number of bytes the mesh will require as a function of \( V \).

**Soup:** \( F \) \( (3) \) \( (3) \) \( (4) \) \( \frac{F}{6} \) \( \text{v's per v} \) \( \text{words per v} \) \( \text{bytes per word} \)

\[ = F \frac{36}{2} = 72 \frac{V}{2} \text{ bytes} \]

**Indexed:** \( V \) \( (3) \) \( (4) \) \( \approx 12V \) for vertices

\( F \) \( (3) \) \( (4) \) \( \approx 24V \) for faces

\[ \approx 36 \text{ bytes} \]
3. Generate an Indexed Tessellated Quad

Write pseudo-code to generate an indexed mesh representation of a
tessellated quad. Let the number of vertices along each side be \(N+1\). Assume that the corners of the quad are \((-1,-1,0)\) and \((1,1,0)\). Your
code should produce a set of coordinates for the vertices and a set of
elements that index into the vertex set.

\[
\begin{align*}
\Delta x &= \Delta y = \frac{2}{N} \\
\text{for } i = 0, j < N, i++ \\
\text{for } j = 0, j < N, j++ \\
\text{// j = column, i = row} \\
x &= 1 - j \times \Delta x \\
y &= 1 - i \times \Delta y \\
\text{z = 0} \\
\text{push_vertex} (x, y, z)
\end{align*}
\]

4. Computing Normal Vectors

Write pseudo-code to efficiently compute per-vertex normals
averaged from the face normals incident upon the vertex. Assume the
mesh is in an indexed representation.

\[
\begin{align*}
F &= [v_1, v_2, \ldots] \\
V &= [v_1, v_2, \ldots] \\
N &= [\langle 0, 0, 0 \rangle, \langle 0, 0, 0 \rangle, \ldots] \\
\text{size of } N &= 3 \times \text{size of } V \\
\text{for each face } f \in F \\
\text{nrm} &= \frac{V[f[i]] - V[f[0]] \times V[f[2]] - V[f[0]]}{\|V[f[i]] - V[f[0]] \times V[f[2]] - V[f[0]]\|} \\
\text{for } i = 0, j < 3, j++ \\
N[f[i]] &= \text{nrm} \\
\text{for each } n \in N \\
\text{normalize}(n)
\end{align*}
\]
\[ \text{for } i = 0, j < N, i++ \]
\[ \text{for } j = 0, j < N, j++ \]
\[ \text{vid}1 = i \cdot (N+1) + j \]
\[ \text{vid}2 = \text{vid}1 + 1 \]
\[ \text{vid}3 = \text{vid}1 + (N+1) \]
\[ \text{push}_\text{tr}.(\text{vid}1, \text{vid}2, \text{vid}3) \]
\[ \text{vid}4 = \text{vid}3 + 1 \]
\[ \text{push}_\text{tr}.(\text{vid}2, \text{vid}4, \text{vid}3) \]