6. Shape Modeling with Polygonal Meshes

**Polygonal meshes:** polygons (faces) and their outward normals

- Solid
- thin skin

Sometimes each vertex is assigned a **vertex normal**. Why?

How should a cube be represented?

- One needs both **geometric** and **topological** information
Consider the following approach:

![Diagram of Face Table, Index Table, Vertex Table, and Normal Table]

**Advantages?**

**Disadvantages?**
**Data type:**

class VertexID{
  public:
    int vertIndex; // index in vertex list
    int normIndex; // index in normal list
};

class Face{
  public:
    int nVerts; // number of vertices in the face
    VertexID * vert; // list of vertex/normal indices
    Face() {nVerts = 0; vert = NULL;} // constructor
    ~Face() {delete[] vert; nVerts = 0;} // destructor
};

class Mesh{
  private:
    int numVerts; // number of vertices in mesh
    Point3* pt; // array of 3D vertices
    int numNormals; // number of normal vectors for mesh
    Vector3 *norm; // array of normals
    int numFaces; // number of faces in the mesh
    Face* face; // array of face data
  public:
    Mesh(); // constructor
    ~Mesh(); // destructor
};
Computing Normal Vectors

Standard approach: two problems

\[ \mathbf{m} = ( \mathbf{V}_3 - \mathbf{V}_2 ) \times ( \mathbf{V}_1 - \mathbf{V}_2 ) \]

Better approach: (Martin Newell)

\[
\begin{align*}
m_x &= \sum_{i=0}^{N-1} (y_i - y_{next(i)})(z_i + z_{next(i)}) \\
m_y &= \sum_{i=0}^{N-1} (z_i - z_{next(i)})(x_i + x_{next(i)}) \\
m_z &= \sum_{i=0}^{N-1} (x_i - x_{next(i)})(y_i + y_{next(i)})
\end{align*}
\]
Why?

1. The components of $\mathbf{m}$ are proportional to $A_x$, $A_y$ and $A_z$, respectively, where

$$A_x = \text{area of the orthographic projection of the polygon onto the } x = 0 \text{ plane}$$

... 

2. Compute the vector $(A_x, A_y, A_z)$ and normalize it to unit length.
Advantages:

1. Each vertex (normal) is stored just once, memory space is saved

2. Coordinates of a vertex (normal) can be easily changed

Disadvantages:

1. Not easy to find faces which share an edge

2. Shared face edges will be drawn twice when the faces are displayed (wireframe)

?? Can you think of a way to overcome these problems?
A slightly different approach

The cube is still defined by 6 outward-pointing faces, but each face is now defined by 4 edges in counterclockwise order, instead of 4 vertices

Face $A: e_0 e_3 e_2 e_1$

Face $B: e_{11} e_6 e_{10} e_2$
<table>
<thead>
<tr>
<th>Object</th>
<th>Cube</th>
</tr>
</thead>
<tbody>
<tr>
<td>Face Table</td>
<td># of edges</td>
</tr>
<tr>
<td>Index Table</td>
<td>0  3  2  1  11  6  10  2</td>
</tr>
<tr>
<td>Edge Table</td>
<td>e_0  e_1  e_2  ···  e_{11}</td>
</tr>
<tr>
<td>Vertex Table</td>
<td>v_0  v_1  v_2  ···  v_7</td>
</tr>
</tbody>
</table>

The diagram shows a cube object with corresponding tables for faces, vertices, and edges.
Can you think of any disadvantages of this approach?

How should the above approach be modified to accommodate the information of vertex normals?

Is it easy for this approach to determine which edges are incident to a vertex?
Objects with Holes

Extension of page 115’s approach:

<table>
<thead>
<tr>
<th>Face Table</th>
<th>Index Table</th>
<th>Vertex Table</th>
<th>Normal Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 1</td>
<td>1 1 1</td>
<td>1 a</td>
<td>( \mathbf{n}_0 )</td>
</tr>
<tr>
<td></td>
<td>4 1</td>
<td>2 b</td>
<td>( \mathbf{n}_1 )</td>
</tr>
<tr>
<td></td>
<td>8 1</td>
<td>3 c</td>
<td>( \mathbf{n}_2 )</td>
</tr>
<tr>
<td></td>
<td>5 1</td>
<td>4 d</td>
<td>( \mathbf{n}_3 )</td>
</tr>
<tr>
<td></td>
<td>13 1</td>
<td>5 e</td>
<td>( \mathbf{n}_4 )</td>
</tr>
<tr>
<td></td>
<td>14 1</td>
<td>6 f</td>
<td></td>
</tr>
<tr>
<td></td>
<td>12 1</td>
<td>7 g</td>
<td></td>
</tr>
<tr>
<td></td>
<td>9 1</td>
<td>8 h</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 2</td>
<td>9 i</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 2</td>
<td>10 j</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3 2</td>
<td>11 k</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4 2</td>
<td>12 l</td>
<td></td>
</tr>
<tr>
<td></td>
<td>. .</td>
<td>13 m</td>
<td></td>
</tr>
<tr>
<td></td>
<td>. .</td>
<td>14 n</td>
<td></td>
</tr>
</tbody>
</table>

No of loops

External loop
Internal loop
void Mesh::draw()
{
    for ( int f = 0; f < numFaces; f++ )
    {
        glBegin ( GL_POLYGON ) ;

        for (int v = 0; v < face[f].nVerts; v++) {
            int in = face[f].vert[v].normIndex ;
            // index of this normal

            int iv = face[f].vert[v].vertIndex ;
            // index of this vertex

            glNormal3f( norm[in].x, norm[in].y, norm[in].z );

            glVertex3f ( pt[iv].x, pt[iv].y, pt[iv].z );
        }
        glEnd ( );
    }
}
Polyhedra:

- Faces are planar polygons

Euler’s validation rules:

1. \( V - E + F = 2 \)

2. \( V - E + F = 2 - 2G + H \)

where

- \( V = \) number of vertices;
- \( E = \) number of edges;
- \( F = \) number of faces;
- \( S = \) number of disconnected components in a collection of solids;
- \( G = \) number of holes through the solid;
- \( H = \) number of disconnected interior rings;
Example:

\[ v = 16 \]
\[ E = 24 \]
\[ F = 11 \]
\[ G = 0 \]
\[ H = 1 \]
Mesh Approximations to Smooth Objects

Build meshes to approximate smooth shapes with two concerns in mind:

1. Accuracy
2. Efficiency
Accuracy:

\[ \leq \varepsilon \]

Efficiency:

- Efficient mesh construction process
- Including computation of the vertices, normals at the vertices, and construction of the faces
- Prefer parametric representation
Parametric Representation:

\[ S(u, v) = (X(u, v), Y(u, v), Z(u, v)) \]

\[ \alpha \leq u, v \leq \beta \]

Vertices and Normals are usually computed for points on \( u \)-contours and \( v \)-contours

\[ \mathbf{n} = \frac{\partial S}{\partial u} \times \frac{\partial S}{\partial v} \]
Implicit representations are converted to parametric representations first.

**Sphere:**

\[ F(x, y, z) = x^2 + y^2 + z^2 - 1 = 0 \]

\[ S(u, v) = (\cos(v)\cos(u), \cos(v)\sin(u), \sin(v)) \]

\[ 0 \leq u \leq 2\pi, \quad -\pi/2 \leq v \leq \pi/2 \]
How should vertex table, normal table and face table be constructed?

If possible, use **forward differencing** to evaluate $v_1, v_2, v_3, ...$ and partial derivatives at these points.

Vertices of $F_i$ are:  ???
If the parameter space $[0, 1] \times [0, 1]$ is uniformly divided into $m$ subintervals and $n$ subintervals in $u$ and $v$ directions, respectively,

and if

$$i = jn + k ; \quad 0 \leq k \leq m-1, \ 0 \leq k \leq n-1$$

then vertices of $F_i$ are:

$$V_{j(n+1)+k}, \quad V_{j(n+1)+k+1},$$

$$V_{(j+1)(n+1)+k}, \quad V_{(j+1)(n+1)+k+1}$$