Building Models

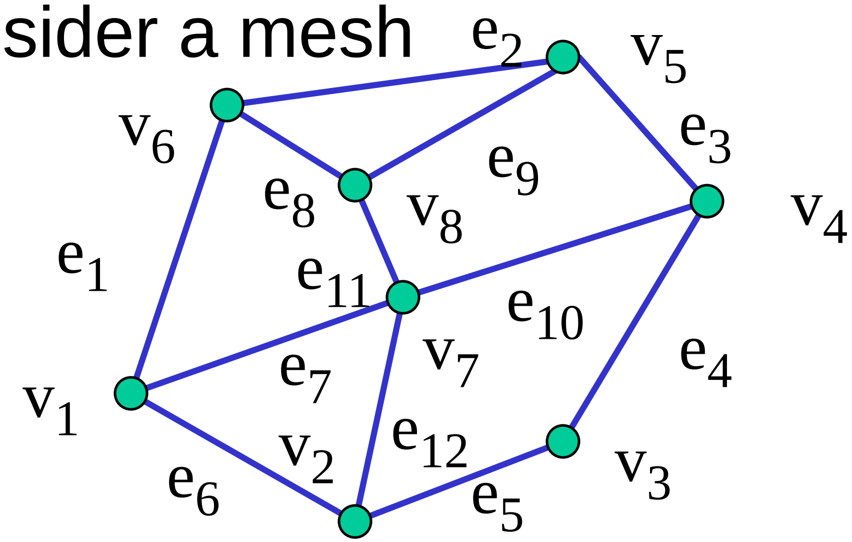
Ed Angel
Professor Emeritus of Computer Science
University of New Mexico

Objectives

- Introduce simple data structures for building polygonal models
    - Vertex lists
    - Edge lists
- Deprecated OpenGL vertex arrays

Representing a Mesh

- Consider a mesh



- There are 8 nodes and 12 edges
  - 5 interior polygons
  - 6 interior (shared) edges
- Each vertex has a location $v_i = (x_i\ y_i\ z_i)$

Simple Representation

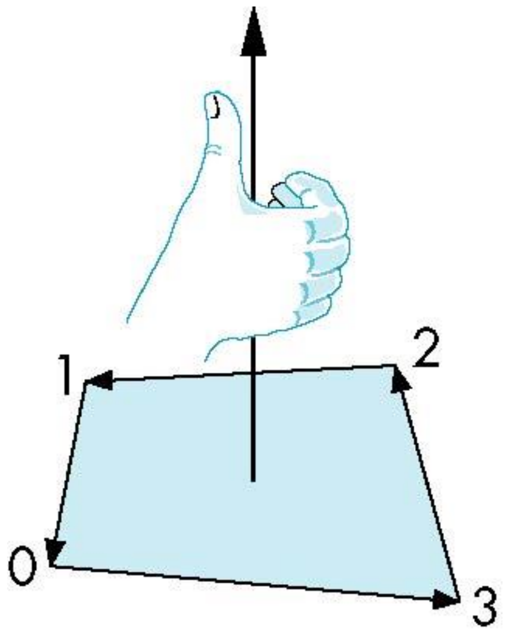
- Define each polygon by the geometric locations of its vertices
- Leads to OpenGL code such as

```
vertex[i] = vec3(x1, x1, x1);
vertex[i+1] = vec3(x6, x6, x6);
vertex[i+2] = vec3(x7, x7, x7);
i+=3;
```

- Inefficient and unstructured
  - Consider moving a vertex to a new location
  - Must search for all occurrences

Inward and Outward Facing Polygons

- The order $\{v_1, v_6, v_7\}$ and $\{v_6, v_7, v_1\}$ are equivalent in that the same polygon will be rendered by OpenGL but the order $\{v_1, v_7, v_6\}$ is different
- The first two describe *outwardly facing* polygons
- Use the *right-hand rule* = counter-clockwise encirclement of outward-pointing normal
- OpenGL can treat inward and outward facing polygons differently

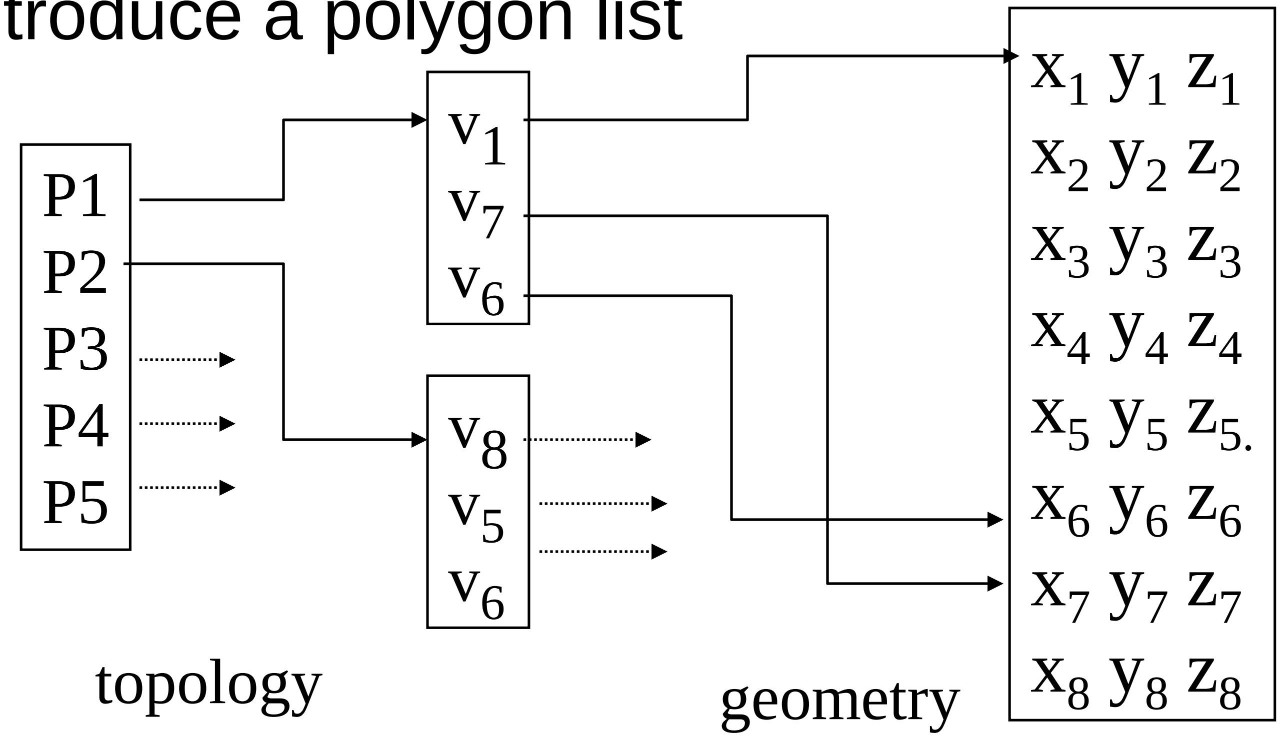


Geometry vs Topology

- Generally it is a good idea to look for data structures that separate the geometry from the topology
  - Geometry: locations of the vertices
  - Topology: organization of the vertices and edges
  - Example: a polygon is an ordered list of vertices with an edge connecting successive pairs of vertices and the last to the first
  - Topology holds even if geometry changes

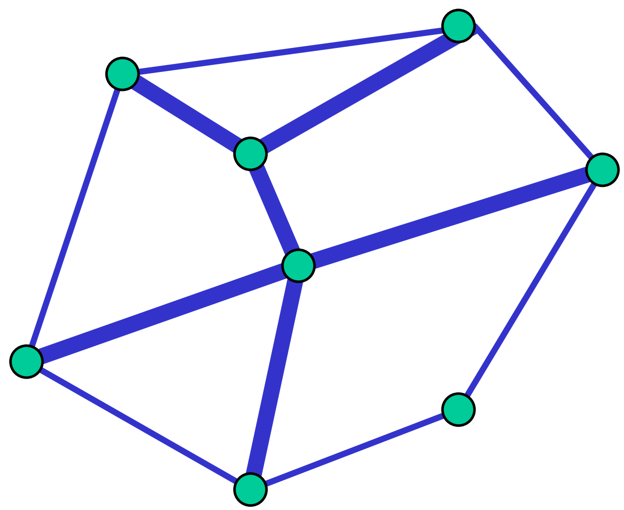
Vertex Lists

- Put the geometry in an array
- Use pointers from the vertices into this array
- Introduce a polygon list



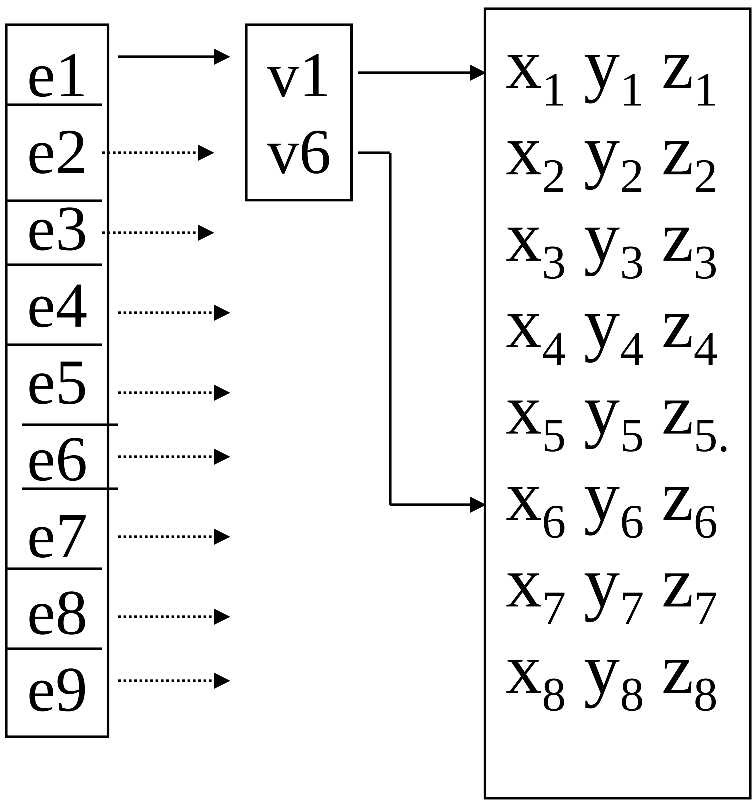
Shared Edges

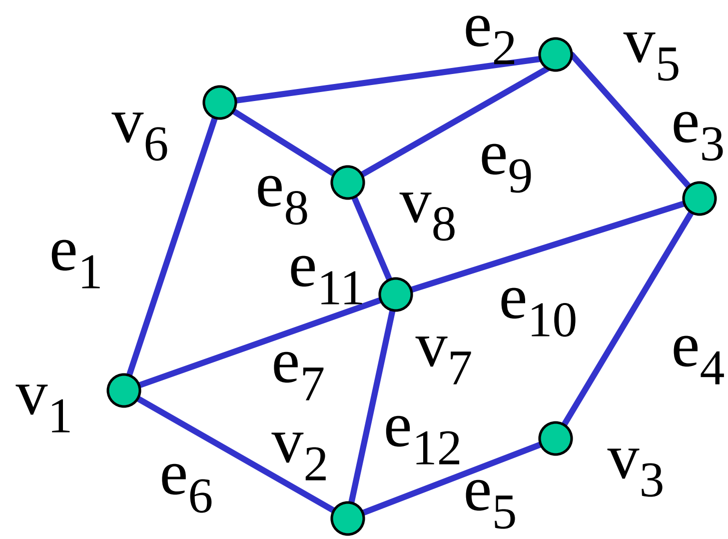
- Vertex lists will draw filled polygons correctly but if we draw the polygon by its edges, shared edges are drawn twice



- Can store mesh by *edge list*

Edge List





Note polygons are not represented

Modeling a Cube

Define global arrays for vertices and colors

```
typedef vec3 point3;
point3 vertices[] = {point3(-1.0,-1.0,-1.0),
  point3(1.0,-1.0,-1.0), point3(1.0,1.0,-1.0),
  point3(-1.0,1.0,-1.0), point3(-1.0,-1.0,1.0),
  point3(1.0,-1.0,1.0), point3(1.0,1.0,1.0),
  point3(-1.0,1.0,1.0)};

typedef vec3 color3;
color3 colors[] = {color3(0.0,0.0,0.0),
  color3(1.0,0.0,0.0), color3(1.0,1.0,0.0),
  color(0.0,1.0,0.0), color3(0.0,0.0,1.0),
  color3(1.0,0.0,1.0), color3(1.0,1.0,1.0),
  color3(0.0,1.0,1.0});
```

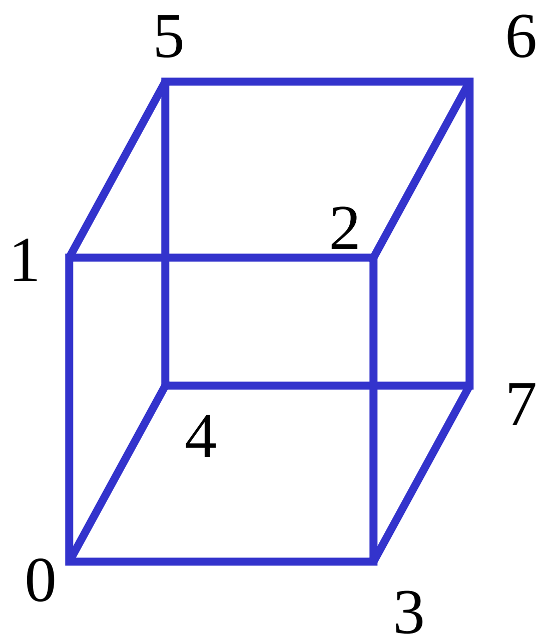
Drawing a triangle from a list of indices

Draw a triangle from a list of indices into the array **`vertices`** and assign a color to each index

```
void triangle(int a, int b, int c, int d)
{
   vcolors[i] = colors[d];
   position[i] = vertices[a];
   vcolors[i+1] = colors[d]);
   position[i+1] = vertices[a];
   vcolors[i+2] = colors[d];
   position[i+2] = vertices[a];
   i+=3;
 }
```

Draw cube from faces

```
void colorcube( )
{
    quad(0,3,2,1);
    quad(2,3,7,6);
    quad(0,4,7,3);
    quad(1,2,6,5);
    quad(4,5,6,7);
    quad(0,1,5,4);
}
```



Note that vertices are ordered so that we obtain correct outward facing normals

Efficiency

- The weakness of our approach is that we are building the model in the application and must do many function calls to draw the cube
- Drawing a cube by its faces in the most straight forward way. Used to require
  - 6 **`glBegin`**, 6 **`glEnd`**
  - 6 **`glColor`**
  - 24 **`glVertex`**
  - More if we use texture and lighting

Vertex Arrays

- OpenGL provided a facility called *vertex arrays* that allows us to store array data in the implementation
- Six types of arrays were supported initially
  - Vertices
  - Colors
  - Color indices
  - Normals
  - Texture coordinates
  - Edge flags
- Now vertex arrays can be used for any attributes

The University of New Mexico

Old Style Initialization

- Using the same color and vertex data, first we enable

```
glEnableClientState(GL_COLOR_ARRAY);
glEnableClientState(GL_VERTEX_ARRAY);
```

- Identify location of arrays

```
glVertexPointer(3, GL_FLOAT, 0, vertices);
```

data array

3d arrays   stored as floats   data contiguous

```
glColorPointer(3, GL_FLOAT, 0, colors);
```

Mapping indices to faces

- Form an array of face indices

```
GLubyte cubeIndices[24] = {0,3,2,1,2,3,7,6
    0,4,7,3,1,2,6,5,4,5,6,7,0,1,5,4};
```

- Each successive four indices describe a face of the cube
- Draw through `glDrawElements` which replaces all `glVertex` and `glColor` calls in the display callback

Drawing the cube

- Old Method:

```
glDrawElements(GL_QUADS, 24,
    GL_UNSIGNED_BYTE, cubeIndices);
```

Draws cube with 1 function call!!

- Problem is that although we avoid many function calls, data are still on client side
- Solution:
  - no immediate mode
  - Vertex buffer object
  - Use glDrawArrays

Rotating Cube

- Full example
- Model Colored Cube
- Use 3 button mouse to change direction of rotation
- Use idle function to increment angle of rotation

The University of New Mexico

Cube Vertices

```
// Vertices of a unit cube centered at origin
//  sides aligned with axes
point4 vertices[8] = {
    point4( -0.5, -0.5,  0.5, 1.0 ),
    point4( -0.5,  0.5,  0.5, 1.0 ),
    point4(  0.5,  0.5,  0.5, 1.0 ),
    point4(  0.5, -0.5,  0.5, 1.0 ),
    point4( -0.5, -0.5, -0.5, 1.0 ),
    point4( -0.5,  0.5, -0.5, 1.0 ),
    point4(  0.5,  0.5, -0.5, 1.0 ),
    point4(  0.5, -0.5, -0.5, 1.0 )
};
```

Colors

```
// RGBA colors
color4 vertex_colors[8] = {
    color4( 0.0, 0.0, 0.0, 1.0 ),  // black
    color4( 1.0, 0.0, 0.0, 1.0 ),  // red
    color4( 1.0, 1.0, 0.0, 1.0 ),  // yellow
    color4( 0.0, 1.0, 0.0, 1.0 ),  // green
    color4( 0.0, 0.0, 1.0, 1.0 ),  // blue
    color4( 1.0, 0.0, 1.0, 1.0 ),  // magenta
    color4( 1.0, 1.0, 1.0, 1.0 ),  // white
    color4( 0.0, 1.0, 1.0, 1.0 )   // cyan
};
```

Quad Function

```
// quad generates two triangles for each face and assigns colors
//    to the vertices
int Index = 0;
void quad( int a, int b, int c, int d )
{
    colors[Index] = vertex_colors[a]; points[Index] = vertices[a]; Index++;
    colors[Index] = vertex_colors[b]; points[Index] = vertices[b]; Index++;
    colors[Index] = vertex_colors[c]; points[Index] = vertices[c]; Index++;
    colors[Index] = vertex_colors[a]; points[Index] = vertices[a]; Index++;
    colors[Index] = vertex_colors[c]; points[Index] = vertices[c]; Index++;
    colors[Index] = vertex_colors[d]; points[Index] = vertices[d]; Index++;
}
```

Color Cube

```
// generate 12 triangles: 36 vertices and 36 colors
void
colorcube()
{
    quad( 1, 0, 3, 2 );
    quad( 2, 3, 7, 6 );
    quad( 3, 0, 4, 7 );
    quad( 6, 5, 1, 2 );
    quad( 4, 5, 6, 7 );
    quad( 5, 4, 0, 1 );
}
```

The University of New Mexico

Initialization I

```
void
init()
{
   colorcube();

   // Create a vertex array object

   GLuint vao;
   glGenVertexArrays ( 1, &vao );
   glBindVertexArray ( vao );
```

Initialization II

```
// Create and initialize a buffer object
    GLuint buffer;
    glGenBuffers( 1, &buffer );
    glBindBuffer( GL_ARRAY_BUFFER, buffer );
    glBufferData( GL_ARRAY_BUFFER, sizeof(points) +
       sizeof(colors), NULL, GL_STATIC_DRAW );
    glBufferSubData( GL_ARRAY_BUFFER, 0,
        sizeof(points), points );
    glBufferSubData( GL_ARRAY_BUFFER, sizeof(points),
        sizeof(colors), colors );
// Load shaders and use the resulting shader program
    GLuint program = InitShader( "vshader36.glsl", "fshader36.glsl" );
    glUseProgram( program );
```

The University of New Mexico

Initialization III

```
// set up vertex arrays
    GLuint vPosition = glGetAttribLocation( program, "vPosition" );
    glEnableVertexAttribArray( vPosition );
    glVertexAttribPointer( vPosition, 4, GL_FLOAT, GL_FALSE, 0,
                        BUFFER_OFFSET(0) );

    GLuint vColor = glGetAttribLocation( program, "vColor" );
    glEnableVertexAttribArray( vColor );
    glVertexAttribPointer( vColor, 4, GL_FLOAT, GL_FALSE, 0,
                        BUFFER_OFFSET(sizeof(points)) );

    theta = glGetUniformLocation( program, "theta" );
```

Display Callback

```
void
display( void )
{
    glClear( GL_COLOR_BUFFER_BIT
        |GL_DEPTH_BUFFER_BIT );

    glUniform3fv( theta, 1, Theta );
    glDrawArrays( GL_TRIANGLES, 0, NumVertices );

    glutSwapBuffers();
}
```

The University of New Mexico

Mouse Callback

```
void
mouse( int button, int state, int x, int y )
{
   if ( state == GLUT_DOWN ) {
      switch( button ) {
         case GLUT_LEFT_BUTTON:    Axis = Xaxis;  break;
         case GLUT_MIDDLE_BUTTON:  Axis = Yaxis;  break;
         case GLUT_RIGHT_BUTTON:   Axis = Zaxis;  break;
      }
   }
}
```

Idle Callback

```
void
idle( void )
{
   Theta[Axis] += 0.01;

   if ( Theta[Axis] > 360.0 ) {
      Theta[Axis] -= 360.0;
   }

   glutPostRedisplay();
}
```