Building Models

CS 537 Interactive Computer Graphics
Prof. David E. Breen
Department of Computer Science

Objectives

• Introduce simple data structures for building polygonal models
  - Vertex lists
  - Edge lists

Representations of 3D Transformations

• Z axis represents depth
• Right Handed System
  - When looking “down” at the origin, positive rotation is CCW
• Left Handed System
  - When looking “down”, positive rotation is in CW
  - More natural interpretation for displays, big z means “far”

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
  ```c
  vertex[i] = vec3(x1, y1, z1);
  vertex[i+1] = vec3(x6, y6, z6);
  vertex[i+2] = vec3(x7, y7, z7);
  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 \( \{v_1, v_2, v_3\} \) 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

Rotating Cube

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

Draw cube from faces

```c
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);
}
```

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

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( "vshdrcube.glsl", "fshdrcube.glsl" );
glUseProgram( program );
Initialization III

```c
// 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)));

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

Display Callback

```c
void
display( void )
{
    glClear( GL_COLOR_BUFFER_BIT |
             GL_DEPTH_BUFFER_BIT );
    glUniform3fv(thetaLoc, 1, theta);
    glDrawArrays(GL_TRIANGLES, 0, NumVertices);
    glutSwapBuffers();
}
```

OpenGL code

- Remember that matrices are column major order in GLSL, so …

```c
transpose your matrices when sending them to the shaders!

```glUniformMatrix4fv(matrix_loc, 1, GL_TRUE, model_view);
```

Keyboard Callback

```c
void
key( int key, int x, int y )
{
    switch( key ) {
    case GLUT_KEY_LEFT:    axis = Xaxis;  break;
    case GLUT_KEY_UP:      axis = Yaxis;  break;
    case GLUT_KEY_RIGHT:   axis = Zaxis;  break;
    case GLUT_KEY_DOWN:    axis = Yaxis;  break;
    case 'q':              exit(0); break;
    case 'r':              randomize(); break;
    default:               return; break;
    }
    glutPostRedisplay();
}
```

Mouse Callback

```c
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

```c
void
idle( void )
{
    theta[axis] += 0.01;
    if ( theta[axis] > 360.0 ) {
        theta[axis] -= 360.0;
    }
    glutPostRedisplay();
}
```

Transforming Each Vertex

```c
in vec4 vPosition, vColor;
out vec4 color;
uniform mat4 rot;

```void
main()
{
    gl_Position = rot * vPosition;
    color = vColor;
}
```
OpenGL Default View Volume

- The default viewing volume is a box centered at the origin with sides of length 2
- \((-1,-1,-1) \rightarrow (1,1,1)\)
- All geometry in box is parallel-projected into the z=0 plane!
- Then rendered

Assignment 4 Suggestions

- Define cube geometry and color in `init()`
- Keyboard callback
  - Figures out how to change transformation values
  - Calculates new transformation matrix, that includes scale, rotation and translation, and sends it the GPU via a uniform variable
  - Calls `glutPostRedisplay()`
- Display function draws cube
- Vertex shader applies transformation matrix to vertices

Objectives

- Introduce the classical views
- Compare and contrast image formation by computer with how images have been formed by architects, artists, and engineers
- Learn the benefits and drawbacks of each type of view

Classical Viewing

- Viewing requires three basic elements
  - One or more objects
  - A viewer with a projection surface
  - Projectors that go from the object(s) to the projection surface
- Classical views are based on the relationship among these elements
  - The viewer picks up the object and orients it how she would like to see it
- Each object is assumed to be constructed from flat principal faces
  - Buildings, polyhedra, manufactured objects
Planar Geometric Projections

- Standard projections project onto a plane
- Projectors are lines that either
  - converge at a center of projection
  - are parallel
- Such projections preserve lines
  - but not necessarily angles
- Nonplanar projections are needed for applications such as map construction

Classical Projections

- Front elevation
- Elevation oblique
- Plan oblique
- Isometric
- One-point perspective
- Three-point perspective

Perspective vs Parallel

- Computer graphics treats all projections the same and implements them with a single pipeline
- Classical viewing developed different techniques for drawing each type of projection
- Fundamental distinction is between parallel and perspective viewing even though mathematically parallel viewing is the limit of perspective viewing

Taxonomy of Planar Geometric Projections

- Parallel vs Perspective
- Multiview, Axonometric, Orthographic
- Isometric, Dimetric, Trimetric
- 1 point, 2 point, 3 point

Perspective Projection

Parallel Projection
Orthographic Projection

Projectors are orthogonal to projection surface

Multiview Orthographic Projection

- Projection plane parallel to principal face
- Usually form front, top, side views
- Isometric (not multiview orthographic view)

In CAD and architecture, we often display three multiviews plus isometric.

Advantages and Disadvantages

- Preserves both distances and angles
  - Shapes preserved
  - Can be used for measurements
    - Building plans
    - Manuals
- Cannot see what object really looks like
  - Because many surfaces hidden from view
  - Often we add the isometric

Axonometric Projections

- Allow projection plane to move relative to object
- Classify by how many angles of a corner of a projected cube are the same
  - None: trimetric
  - Two: dimetric
  - Three: isometric

Types of Axonometric Projections

- Dimetric
- Tretric
- Isometric

Advantages and Disadvantages

- Lines are scaled (foreshortened) but can find scaling factors
- Lines preserved but angles are not
  - Projection of a circle in a plane not parallel to the projection plane is an ellipse
- Can see three principal faces of a box-like object
- Some optical illusions possible
- Parallel lines appear to diverge
- Does not look real because far objects are scaled the same as near objects
- Used in CAD applications
Oblique Projection

Arbitrary relationship between projectors and projection plane

Advantages and Disadvantages

• Can pick the angles to emphasize a particular face
  - Architecture: plan oblique, elevation oblique
• Angles in faces parallel to projection plane are preserved while we can still see "around" side
• In physical world, cannot create with simple camera; possible with bellows camera or special lens (architectural)

Perspective Projection

Projectors converge at center of projection

Vanishing Points

• Parallel lines (not parallel to the projection plan) on the object converge at a single point in the projection (the vanishing point)
• Drawing simple perspectives by hand uses these vanishing point(s)

Three-Point Perspective

• No principal face parallel to projection plane
• Three vanishing points for cube

Two-Point Perspective

• One principal direction parallel to projection plane
• Two vanishing points for cube
One-Point Perspective

• One principal face parallel to projection plane
• One vanishing point for cube

Advantages and Disadvantages

• Objects further from viewer are projected smaller than the same sized objects closer to the viewer (*diminution*)
  - Looks realistic
• Equal distances along a line are not projected into equal distances (*nonuniform foreshortening*)
• Angles preserved only in planes parallel to the projection plane
• More difficult to construct by hand than parallel projections (but not more difficult by computer)

Taxonomy of Planar Geometric Projections

- planar geometric projections
  - parallel
    - perspective
      - multiview
        - axonometric
        - orthographic
      - isometric
      - dimetric
      - trimetric
  - 1 point
  - 2 point
  - 3 point