Building Models

CS 432 Interactive Computer Graphics
Prof. David E. Breen
Department of Computer Science

Objectives

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

Representation 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 WebGL code such as
  
  ```javascript
  vertex.push(vec3(x1, y1, z1));
  vertex.push(vec3(x6, y6, z6));
  vertex.push(vec3(x7, y7, z7));
  ```

- 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 WebGL 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
- WebGL 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

Face/Edge/Vertex List

The Rotating Cube
Objectives

- Put everything together to display rotating cube
- Two methods of display
  - by arrays
  - by elements

Modeling a Cube

Define global array for vertices

```javascript
var vertices = [
  vec3(-0.5, -0.5, 0.5),
  vec3(-0.5, 0.5, 0.5),
  vec3(0.5, -0.5, 0.5),
  vec3(-0.5, -0.5, -0.5),
  vec3(0.5, 0.5, -0.5),
  vec3(0.5, -0.5, -0.5)
];
```

Colors

Define global array for colors

```javascript
var vertexColors = [
  [ 1.0, 0.0, 0.0, 1.0 ], // red
  [ 1.0, 0.0, 1.0, 1.0 ], // magenta
  [ 1.0, 1.0, 1.0, 1.0 ], // white
  [ 1.0, 0.0, 0.0, 1.0 ], // yellow
  [ 0.0, 0.0, 0.0, 1.0 ], // black
  [ 0.0, 1.0, 1.0, 1.0 ], // blue
  [ 0.0, 1.0, 0.0, 1.0 ], // cyan
  [ 0.0, 0.0, 0.0, 1.0 ]   // green
];
```

Draw cube from faces

```javascript
function 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
Each quad generates two triangles

Initialization

```javascript
var canvas, gl;
var numVertices  = 36;
var points = [];
var colors = [];
window.onload = function init()
{
  canvas = document.getElementById( "glCanvas" );
  gl = WebGLUtils.setupWebGL( canvas );
  colorCube();
  gl.viewport( 0, 0, canvas.width, canvas.height );
  gl.clearColor( 1.0, 1.0, 1.0, 1.0 );
  gl.enable(gl.DEPTH_TEST);
  // rest of initialization and html file
  // same as previous examples
}
```

The quad Function

Put position and color data for two triangles from a list of indices into the array vertices

```javascript
function quad(a, b, c, d)
{
  var indices = [ a, b, c, a, c, d ];
  for ( var i = 0; i < indices.length; ++i ) {
    points.push( vertices[indices[i]]);
    colors.push( vertexColors[indices[i]] );
  }
  // for solid colored faces use
  //colors.push(vertexColors[a]);
}
```
Render Function

```javascript
function render()
{
    gl.clear( gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
    gl.drawArrays( gl.TRIANGLES, 0, numVertices );
    requestAnimFrame( render );
}
```

Mapping indices to faces

```javascript
var indices = [
    1,0,3,
    3,2,1,
    2,3,7,
    7,6,2,
    3,0,4,
    4,7,3,
    6,5,1,
    1,2,6,
    4,5,6,
    6,7,4,
    5,4,0,
    0,1,5
];
```

Rendering by Elements

- Just send vertices and vertexColors, then indices
- No redundant data transferred
  - More efficient

```javascript
function render()
{
    gl.clear( gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
    gl.drawArrays( gl.TRIANGLES, 0, numVertices );
    requestAnimFrame( render );
}
```

Adding Buttons for Rotation

```javascript
var xAxis = 0;
var yAxis = 1;
var zAxis = 2;
var axis = 0;
var theta = [ 0, 0, 0 ];
var thetaLoc;

document.getElementById( "xButton" ).onclick = function () {axis = xAxis;};
document.getElementById( "yButton" ).onclick = function () {axis = yAxis;};
document.getElementById( "zButton" ).onclick = function () {axis = zAxis;};
```

Rendering Function

```javascript
function render()
{
    gl.clear( gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
    gl.uniform3fv(thetaLoc, theta);
    gl.drawArrays( gl.TRIANGLES, 0, numVertices );
    requestAnimFrame( render );
}
```
WebGL matrix code

• Remember that matrices are column major order in GLSL

In OpenGL we had to transpose your matrices when sending them to the shaders!

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

• flatten() now does it for you!

```gl.UniformMatrix4fv(matrix_loc, false, flatten(model_view));```

Transforming Each Vertex

```attribute vec4 vPosition, vColor;
varying 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) → (1,1,1)
  • All geometry in box is parallel-projected into the z=0 plane!
  • Then rendered

Go to Assignment 4

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
• Render function draws cube
• Vertex shader applies transformation matrix to vertices

Classical Viewing
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

Classical Projections

- 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

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
Perspective Projection

Object

Projector

COP

Projection plane

Parallel Projection

Object

DOP

Projector

Projection plane

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

front

side

top

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
- Trilinear
- 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

History of Linear Perspective

- Renaissance artists
  - Alberti (1435)
  - Della Francesca (1470)
  - Da Vinci (1490)
  - Pélerin (1505)
  - Dürer (1525)

Dürer: Measurement Instruction with Compass and Straight Edge

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)

One-Point Perspective

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

Two-Point Perspective

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

Three-Point Perspective

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

Perspective Projection (Titanic)

- 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)

Advantages and Disadvantages
Taxonomy of Planar Geometric Projections

- Planar geometric projections
  - Parallel
  - Perspective
    - Multiview
      - Axonometric
        - 1 point
        - 2 point
        - 3 point
    - Orthographic
    - Isometric
    - Dimetric
    - Trimetric