Modeling a Colored Cube

Tom Kelliher, CS 320

Mar. 29, 2013

1 Administrivia

Announcements

Assignment

4.1–4.5.

From Last Time

Project lab.

Outline

1. Rotating cube program.
2. Cube representation.
3. Depth buffering.
4. Non-Commutativity of rotations.

Coming Up

Viewer movement in OpenGL.

2 Prelude

If you want to rotate an object about its center, in what order do you apply the three transformations? Does the order matter?

3 A Rotating, Color-Interpolated Cube

- Assign a color to each vertex and see what happens.
- Note dimensions of cube and clipping volume.

3.1 Representation

There's a hard way and an easy way to do this. Which way is this?

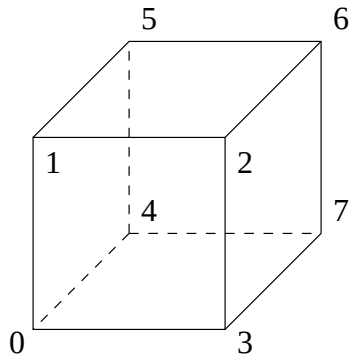
```
// Vertices of a unit cube centered at origin, sides aligned with axes
point4 vertices[8] = {
    point4( -0.5, -0.5,  0.5, 1.0 ),
    point4( -0.5,  0.5,  0.5, 1.0 ),
    point4(  0.5,  0.5,  0.5, 1.0 ),
    point4(  0.5, -0.5,  0.5, 1.0 ),
    point4( -0.5, -0.5, -0.5, 1.0 ),
    point4( -0.5,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

// 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
};
```

1. Coordinate system: +x to right, +y up, +z towards us. Right-hand system.

2. Vertex list and a numbering of the cube's vertices:



3. Color interpolation: bilinear interpolation.

   Let p be α the way from P_0 to P_1. p's color is:

   $$(1-\alpha)P_0 + \alpha P_1$$

   (for each color)

   What about points on interior of polygon?

4. Enumerating the vertices on each of the faces:

```
void quad( int a, int b, int c, int d )
{
    colors[Index] = vertex_colors[a]; points[Index++] = vertices[a];
    colors[Index] = vertex_colors[b]; points[Index++] = vertices[b];
    colors[Index] = vertex_colors[c]; points[Index++] = vertices[c];
    colors[Index] = vertex_colors[a]; points[Index++] = vertices[a];
    colors[Index] = vertex_colors[c]; points[Index++] = vertices[c];
    colors[Index] = vertex_colors[d]; points[Index++] = vertices[d];
```

```
}

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 );
}
```

Are we following the righthand rule for the outer side of each face? Does order of rendering faces matter?

Why represent a cube this way?

5. Display function:

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

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

    glutSwapBuffers();
}
```

3.2 Rotating One and Two Faces

Been here, done this. **Demo** with `CubeFaces`. One face:

1. The face is rotating about the origin.

2. Perspective is *not* maintained.

Two faces:

1. An unexpected result? Why?

2. Fixing it: the depth buffer and hidden surface removal. Idea: associate a z-value with each pixel in the frame buffer and only conditionally write new pixels.

3.3 Non-Commutativity of Rotations

Will these give the same result?

```
mv = RotateZ(Theta[Zaxis]) * RotateX(Theta[Xaxis]);
```

```
mv = RotateX(Theta[Xaxis]) * RotateZ(Theta[Zaxis]);
```

Can you describe the results?