Modeling a Colored Cube

Tom Kelliher, CS 320

Mar. 25, 2011

1 Administrivia

Announcements

Monday will be a project day.

Assignment

What you should be reading: 4.1–4.9, Appendices B and C as necessary.

From Last Time

Outline

Project lab.

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

Coming Up

More on linear algebra basis of transformations.

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.
- Note that the reshape function maintains the aspect ratio:

```
void myReshape(int w, int h)
{
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    if (w <= h)
        glOrtho(-2.0, 2.0, -2.0 * (GLfloat) h / (GLfloat) w,
            2.0 * (GLfloat) h / (GLfloat) w, -10.0, 10.0);
    else
        glOrtho(-2.0 * (GLfloat) w / (GLfloat) h,
            2.0 * (GLfloat) w / (GLfloat) h, -2.0, 2.0, -10.0, 10.0);
    glMatrixMode(GL_MODELVIEW);
}
```

(Note use of `glOrtho`.)

3.1 Representation

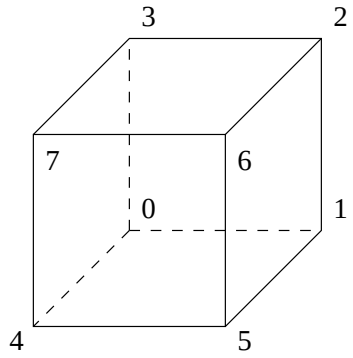
There's a hard way and an easy way to do this. Which way is this?

```
GLfloat vertices[][3] = {{-1.0,-1.0,-1.0},{1.0,-1.0,-1.0},
{1.0,1.0,-1.0}, {-1.0,1.0,-1.0}, {-1.0,-1.0,1.0},
{1.0,-1.0,1.0}, {1.0,1.0,1.0}, {-1.0,1.0,1.0}};

GLfloat colors[][3] = {{0.0,0.0,0.0},{1.0,0.0,0.0},
{1.0,1.0,0.0}, {0.0,1.0,0.0}, {0.0,0.0,1.0},
{1.0,0.0,1.0}, {1.0,1.0,1.0}, {0.0,1.0,1.0}};
```

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 colorcube(void)
{

/* map vertices to faces */

   polygon(0,3,2,1);
   polygon(2,3,7,6);
   polygon(0,4,7,3);
```

```
   polygon(1,2,6,5);
   polygon(4,5,6,7);
   polygon(0,1,5,4);
}


void polygon(int a, int b, int c , int d)
{

/* draw a polygon via list of vertices */

   glBegin(GL_POLYGON);
      glColor3fv(colors[a]);
      glVertex3fv(vertices[a]);
      glColor3fv(colors[b]);
      glVertex3fv(vertices[b]);
      glColor3fv(colors[c]);
      glVertex3fv(vertices[c]);
      glColor3fv(colors[d]);
      glVertex3fv(vertices[d]);
   glEnd();
}
```

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?

Why the normals in the program code?

5. Display function:

```
void display(void)
{
/* display callback, clear frame buffer and z buffer,
   rotate cube and draw, swap buffers */

   glLoadIdentity();
   glRotatef(theta[0], 1.0, 0.0, 0.0);
   glRotatef(theta[1], 0.0, 1.0, 0.0);
   glRotatef(theta[2], 0.0, 0.0, 1.0);

   colorcube();
```

```
    glutSwapBuffers();
}
```

3.2 Rotating One and Two Faces

Been here, done this. **Demo** with `Cube`. 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?

```
glLoadIdentity();
glRotatef(theta[0], 1.0, 0.0, 0.0);   /* Rotate about x axis. */
glRotatef(theta[2], 0.0, 0.0, 1.0);   /* Rotate about z axis. *.

glLoadIdentity();
glRotatef(theta[2], 0.0, 0.0, 1.0);   /* Rotate about z axis. *.
glRotatef(theta[0], 1.0, 0.0, 0.0);   /* Rotate about x axis. */
```

Can you describe the results?