A Room with a View

Tom Kelliher, CS 320

Apr. 17, 2009

1 Administrivia

Announcements

Assignment

Read 6.1–3.

From Last Time

3-D projections, Movement in 3-D, Problems with 3-D movement.

Outline

1. roomView.
2. Model construction.
3. Another way of thinking about coordinate systems and transformations.
4. Walk-through of `roomView.c`. Multiple viewports and projections.
5. Lab exercise.

Coming Up

Light.

2 Demonstration of `roomView.c`

3 Model Construction

Consider building a room.

Two approaches:

1. Pre-fab:
   (a) A few object types.
   (b) Each is built at the origin.
   (c) Transformed into place.
2. Stick-built:
   (a) Exactly what is needed is built.
   (b) Built in final location.

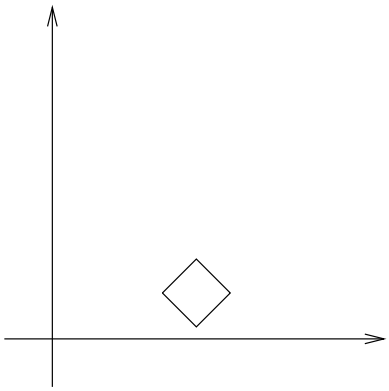
4 Coordinate Systems

Again, two approaches:

1. Global, fixed coordinate system:

(a) All transformations relative to global coordinate system.

(b) Transformation order, code order reversed.



Rotate, then translate:

```
glLoadIdentity();
glTranslatef(...);
glRotatef(...);
// Render polygon.
```

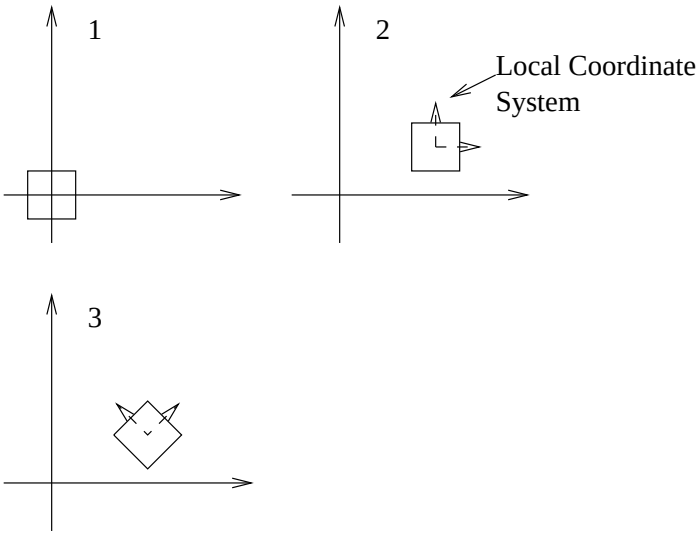
2. Local, movable coordinate system:

   (a) Imagine a local coordinate system fixed to the object.

   (b) Initially, local system is identical to global system.

   (c) Transformations are applied to the local system.

   (d) Transformation order, code order the same:

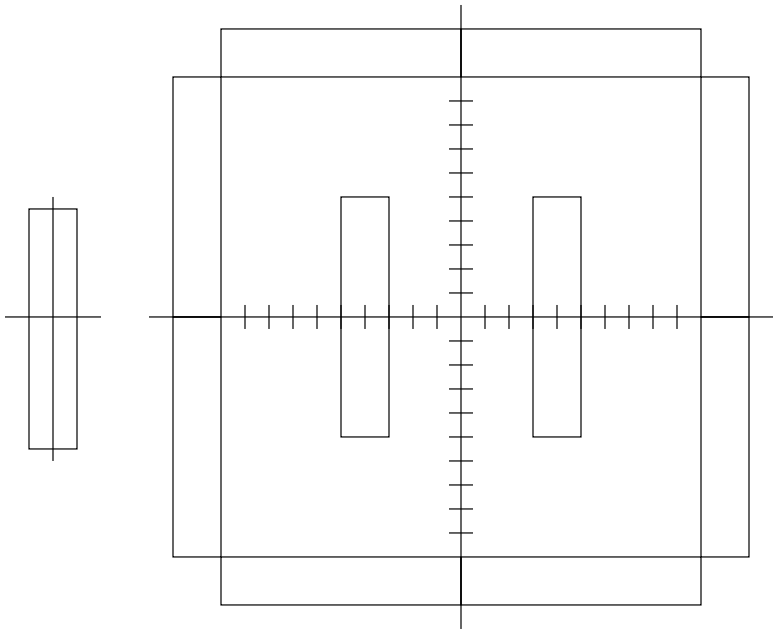


Translate, then rotate:

```
glLoadIdentity();
glTranslatef(...);
glRotatef(...);
// Render polygon.
```

4.1 Constructing a Room

Consider the transformations necessary for constructing:



when what you have to start with is the one block.

Steps:

```
load identity matrix;
push matrix;
move 4 left and render;
move 8 right and render;
pop matrix;
push matrix;
move 11 left, 5 down;

// Repeat the following 4 times.
render;
move 10 up and render;
move 6 right and 6 up;
rotate -90 and render;
move 10 up and render;   // Don't forget that we also rotated the
                         // local coordinate system!

pop matrix;
```

See `room()` for the real code.

Pushes, pops must balance. Used so that we can get back to a previous position.

5 colorCube() and polygon()

A couple vertex lists:

```
GLfloat wall[][3] =
{ {-1.0, -5.0, 0.0}, { 1.0, -5.0, 0.0},
  { 1.0,  5.0, 0.0}, {-1.0,  5.0, 0.0},
  {-1.0, -5.0, 8.0}, { 1.0, -5.0, 8.0},
  { 1.0,  5.0, 8.0}, {-1.0,  5.0, 8.0}
};

GLfloat Floor[][3] =
{ {-12.0, -12.0, 0.0}, { 12.0, -12.0, 0.0},
  { 12.0,  12.0, 0.0}, {-12.0,  12.0, 0.0}
};
```

Is there a general way for rendering?

`colorCube()`:

1. "Canonical" cube vertex list.
2. Rendering various cubes (vertex lists).
3. Color vectors.

```
// colorCube renders a cube.
//    cube: the vertex list specifying the cube.
//    color: the color vector.
//
// Assumptions regarding the vertex list:
// Index   Vertex
//   0     Lower left vertex of back face.
//   1     Lower right vertex of back face.
//   2     Upper right vertex of back face.
//   3     Upper left vertex of back face.
// 4--7    Similar for front face.
// (Assumes we are looking at the origin from the +z axis with the +y axis
//  being "up."
```

```
//

void colorCube(GLfloat cube[][3], GLfloat *color)
{
   polygon(cube,0,3,2,1,color);
   polygon(cube,2,3,7,6,color);
   polygon(cube,0,4,7,3,color);
   polygon(cube,1,2,6,5,color);
   polygon(cube,4,5,6,7,color);
   polygon(cube,0,1,5,4,color);
}
```

`polygon()`:

1. Rendering various polygons.

```
//
// polygon renders a polygon:
//    polygon: name of the vertex list.
//    a, b, c, d: indices of the vertex list vertices to be rendered.
//    color: the color vector.
//

void polygon(GLfloat polygon[][3], int a, int b, int c , int d,
             GLfloat *color)
{
   glPushAttrib(GL_ALL_ATTRIB_BITS);
   glBegin(GL_POLYGON);
      glColor3fv(color);
      glVertex3fv(polygon[a]);
      glVertex3fv(polygon[b]);
      glVertex3fv(polygon[c]);
      glVertex3fv(polygon[d]);
   glEnd();
   glPopAttrib();
}
```

6 Multiple Viewports with Multiple Projections

Ordinarily, the projection mode is set-up in `reshape()`.

What about this case?

Sketch of `display()`:

```
clear color and depth buffers;

// Prepare for the overhead view.
set an orthographic projection;
set viewport;

render room;
render viewer;

flush all polygons;   // Ensure that none are "hanging" around.

// Prepare for the immersed view
set a perspective projection;
set viewport;

position the camera;   // Remember: viewer transformation, then
                       // model transformations.

render room;
render ceiling;

swap color buffers;
```

7 Lab Exercise

1. From class Web site, grab `roomViewLab.c`.

2. Build and run. Note:

   (a) Use arrow keys to increment/decrement x/y to move.

   (b) After about 16 moves, ceiling and walls are lost. Why?

3. Insert the missing pop matrix.

4. Add viewer rotation and fix left, right, forward, backward motion.

5. Can you add collision detection?