A Room with a View

Tom Kelliher, CS 320

Apr. 5, 2013

1 Administrivia

Announcements

Assignment

Read 5.1–3.

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

Project day; 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.

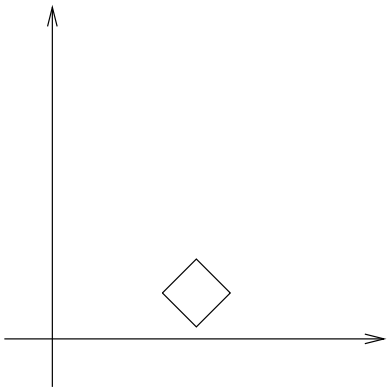
4 Coordinate Systems

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:

```
mv = Translate(...) * RotateX(...);
...
```

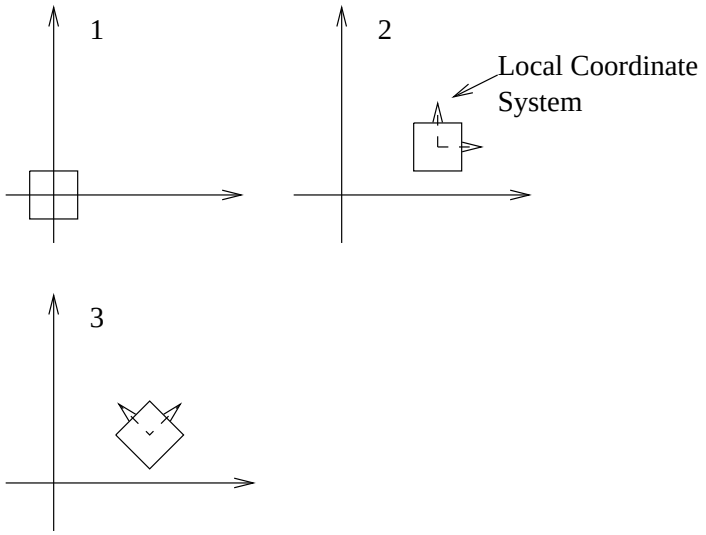
2. Local, movable coordinate system:

   (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:



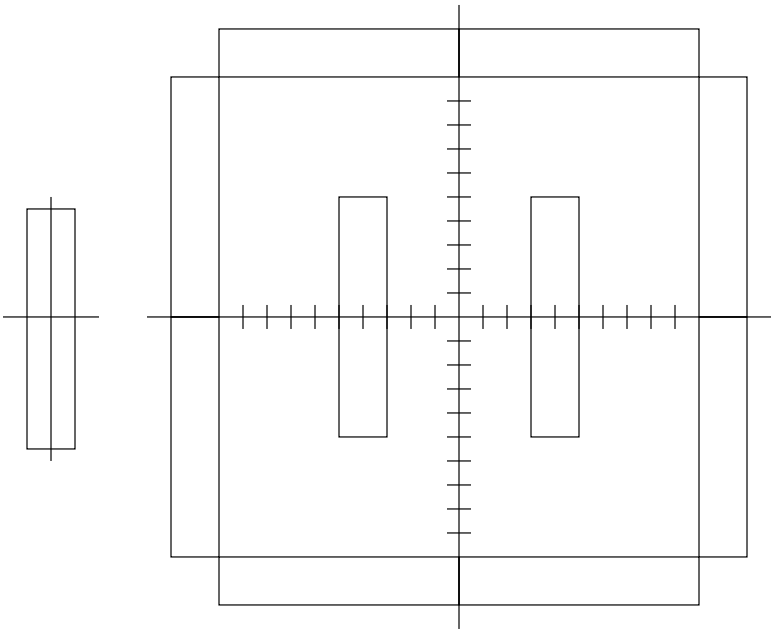
Translate, then rotate:

```
mv = Translate(...) * RotateX(...);
...
```

What's the difference???

4.1 Constructing a Room

Consider the transformations necessary for constructing:



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

Steps:

```
reset to world frame;
translate 4 left
draw;
translate 8 right;
draw;
reset to world frame;
translate 11 left, 5 down;

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

See `room()` for the real code.

5 colorCube() and quad()

A couple vertex lists:

```
Point Wall[] =
{ Point(-1.0, -5.0, 0.0), Point( 1.0, -5.0, 0.0),
  Point( 1.0,  5.0, 0.0), Point(-1.0,  5.0, 0.0),
  Point(-1.0, -5.0, 8.0), Point( 1.0, -5.0, 8.0),
  Point( 1.0,  5.0, 8.0), Point(-1.0,  5.0, 8.0)
};

Point Floor[] =
{ Point(-12.0, -12.0, 0.0), Point( 12.0, -12.0, 0.0),
  Point( 12.0,  12.0, 0.0), Point(-12.0,  12.0, 0.0)
};
```

Is there a general way for drawing?

`colorCube()`:

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

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

```
//
// generate 12 triangles: 36 vertices.

void colorcube(const Point vertices[])
{
    quad( vertices, 1, 0, 3, 2 );
    quad( vertices, 2, 3, 7, 6 );
    quad( vertices, 3, 0, 4, 7 );
    quad( vertices, 6, 5, 1, 2 );
    quad( vertices, 4, 5, 6, 7 );
    quad( vertices, 5, 4, 0, 1 );
}
```

`quad()`:

1. Drawing various quads.

```
// quad generates two triangles for each face.

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

6 Multiple Viewports with Multiple Projections

1. Detail from `init()` showing creation of drawn objects:

```
Index = 0;
colorcube(Wall);                // Dumps vertices into points.
oWall.vao = createVAO();        // Uses vertices from points.
oWall.numVertices = 36;
oWall.geometry = GL_TRIANGLES;
```

```
Index = 0;
quad(Floor, 0, 1, 2, 3);
oFloor.vao = createVAO();
oFloor.numVertices = 6;
oFloor.geometry = GL_TRIANGLES;

Index = 0;
quad(Viewer, 0, 1, 2, 3);
oViewer.vao = createVAO();
oViewer.numVertices = 6;
oViewer.geometry = GL_TRIANGLES;
```

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

   What about this case?

   Sketch of `display()`:

```
clear color and depth buffers;

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

draw room;
draw viewer;

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

// Prepare for the first-person view
set a perspective projection;
set viewport;

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

draw room;
draw ceiling;

swap color buffers;
```

3. Implementation of `display()`:

```
void display(void)
```

```
{
   mat4 p;    /* Projection matrix */

   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

   // Set-up to draw the map.

   p = Ortho(-13.0, 13.0, -13.0, 13.0, -10.0, 1.0);
   glUniformMatrix4fv(projection, 1, GL_TRUE, p);

   glViewport(0, 0, 100, 100);

   // Draw viewer.

   mv = Translate(viewerPosition[0], viewerPosition[1], 10.0);
   glUniformMatrix4fv(model_view, 1, GL_TRUE, mv);
   glBindVertexArray(oViewer.vao);
   glUniform3fv(vColor, 1, black);
   glDrawArrays(oViewer.geometry, 0, oViewer.numVertices);

   // Draw the room.

   mv = Translate(0.0, 0.0, 0.0);
   glUniformMatrix4fv(model_view, 1, GL_TRUE, mv);
   room();

   glFlush();

   // Set-up the do the first-person view.

   p = Perspective(90.0, 1.0, 0.1, 100.0);
   glUniformMatrix4fv(projection, 1, GL_TRUE, p);

   glViewport(100, 100, windowWidth - 100, windowHeight - 100);

   // Position the camera.
   mv = LookAt(vec4(viewerPosition[0], viewerPosition[1],
                    viewerPosition[2], 1.0),
               vec4(viewerPosition[0] + cos(viewerTheta),
                    viewerPosition[1] + sin(viewerTheta),
                    viewerPosition[2], 1.0),
               vec4(0.0, 0.0, 1.0, 1.0));
   glUniformMatrix4fv(model_view, 1, GL_TRUE, mv);

    room();
```

```
    // Draw ceiling.
    mv = mv * Translate(0.0, 0.0, 8.0);
    glUniformMatrix4fv(model_view, 1, GL_TRUE, mv);
    glBindVertexArray(oFloor.vao);
    glUniform3fv(vColor, 1, magenta);
    glDrawArrays(oFloor.geometry, 0, oFloor.numVertices);

    glutSwapBuffers();
}
```

4. room():

```
/*********************************************************************
 * room() --- draw the room.  Because of the separate needs of the map
 * and first-person views, it draws neither the ceiling nor the
 * viewer.
 *********************************************************************/

void room()
{
    mat4 lmv;   // Local copy of the global modelview matrix.

    // Draw floor.
    glBindVertexArray(oFloor.vao);
    glUniform3fv(vColor, 1, white);
    glDrawArrays(oFloor.geometry, 0, oFloor.numVertices);

    // Draw middle walls.
    lmv = mv * Translate(-4.0, 0.0, 0.0);
    glUniformMatrix4fv(model_view, 1, GL_TRUE, lmv);
    glBindVertexArray(oWall.vao);
    glUniform3fv(vColor, 1, red);
    glDrawArrays(oWall.geometry, 0, oWall.numVertices);
    lmv = lmv * Translate(8.0, 0.0, 0.0);
    glUniformMatrix4fv(model_view, 1, GL_TRUE, lmv);
    glDrawArrays(oWall.geometry, 0, oWall.numVertices);

    // Draw walls defining room boundaries.

    lmv = mv * Translate(-11.0, -5.0, 0.0);
    glUniformMatrix4fv(model_view, 1, GL_TRUE, lmv);

    for (int i = 0; i < 4; i++)
```

```
    {
        glUniform3fv(vColor, 1, (i == 1) ? green : blue);
        glDrawArrays(oWall.geometry, 0, oWall.numVertices);
        lmv = lmv * Translate(0.0, 10.0, 0.0);
        glUniformMatrix4fv(model_view, 1, GL_TRUE, lmv);
        glUniform3fv(vColor, 1, (i == 1) ? green : blue);
        glDrawArrays(oWall.geometry, 0, oWall.numVertices);
        lmv = lmv * Translate(6.0, 6.0, 0.0) * RotateZ(-90.0);
        glUniformMatrix4fv(model_view, 1, GL_TRUE, lmv);
    }
}
```

7 Lab Exercise

1. From class Web site, grab `roomviewLab.zip`.

2. Build and run. Note:

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

3. Add viewer rotation and fix left, right, forward, backward motion. Use `A` and `S` keys for rotation.

4. Can you modify the program so that the first-person view is full-window, with the map view movable to any corner of the window?

5. Can you add collision detection?