Day11 example

// This program is a flyby around the RGB color cube.  One intersting note

// is that because the cube is a convex polyhedron and it is the only thing

// in the scene, we can render it using backface culling only. i.e., there

// is no need for a depth buffer.

#ifdef \_\_APPLE\_CC\_\_

#include <GLUT/glut.h>

#else

#include <GL/glut.h>

#endif

#include <cmath>

// The cube has opposite corners at (0,0,0) and (1,1,1), which are black and

// white respectively.  The x-axis is the red gradient, the y-axis is the

// green gradient, and the z-axis is the blue gradient.  The cube's position

// and colors are fixed.

namespace Cube {

const int NUM\_VERTICES = 8;

const int NUM\_FACES = 6;

GLint vertices[NUM\_VERTICES][3] = {

  {0, 0, 0}, {0, 0, 1}, {0, 1, 0}, {0, 1, 1},

  {1, 0, 0}, {1, 0, 1}, {1, 1, 0}, {1, 1, 1}};

GLint faces[NUM\_FACES][4] = {

  {1, 5, 7, 3}, {5, 4, 6, 7}, {4, 0, 2, 6},

  {3, 7, 6, 2}, {0, 1, 3, 2}, {0, 4, 5, 1}};

GLfloat vertexColors[NUM\_VERTICES][3] = {

  {0.0, 1.0, 0.0}, {0.0, 1.0, 0.0}, {0.0, 1.0, 0.0}, {0.0, 1.0, 0.0},

  {1.0, 0.0, 0.0}, {1.0, 0.0, 0.0}, {1.0, 0.0, 0.0}, {1.0, 0.0, 0.0}};

void draw() {

  glBegin(GL\_QUADS);

  for (int i = 0; i < NUM\_FACES; i++) {

    for (int j = 0; j < 4; j++) {

      glColor4fv((GLfloat\*)&vertexColors[faces[i][j]]);

      glVertex3iv((GLint\*)&vertices[faces[i][j]]);

    }

  }

  glEnd();

}

}

// Display and Animation. To draw we just clear the window and draw the cube.

// Because our main window is double buffered we have to swap the buffers to

// make the drawing visible. Animation is achieved by successively moving our

// camera and drawing. The function nextAnimationFrame() moves the camera to

// the next point and draws. The way that we get animation in OpenGL is to

// register nextFrame as the idle function; this is done in main().

void display() {

  glClear(GL\_COLOR\_BUFFER\_BIT);

  Cube::draw();

  glFlush();

  glutSwapBuffers();

}

// We'll be flying around the cube by moving the camera along the orbit of the

// curve u->(8\*cos(u), 7\*cos(u)-1, 4\*cos(u/3)+2).  We keep the camera looking

// at the center of the cube (0.5, 0.5, 0.5) and vary the up vector to achieve

// a weird tumbling effect.

void timer(int v) {

  static GLfloat u = 0.0;

  u += 0.01;

  glLoadIdentity();

  gluLookAt(8\*cos(u), 7\*cos(u)-1, 4\*cos(u/3)+2, .5, .5, .5, cos(u), 1, 0);

  glutPostRedisplay();

  glutTimerFunc(1000/60.0, timer, v);

}

// When the window is reshaped we have to recompute the camera settings to

// match the new window shape.  Set the viewport to (0,0)-(w,h).  Set the

// camera to have a 60 degree vertical field of view, aspect ratio w/h, near

// clipping plane distance 0.5 and far clipping plane distance 40.

void reshape(int w, int h) {

  glViewport(0, 0, w, h);

  glMatrixMode(GL\_PROJECTION);

  glLoadIdentity();

  gluPerspective(60.0, GLfloat(w) / GLfloat(h), 0.5, 40.0);

  glMatrixMode(GL\_MODELVIEW);

}

// Application specific initialization:  The only thing we really need to do

// is enable back face culling because the only thing in the scene is a cube

// which is a convex polyhedron.

void init() {

  glEnable(GL\_CULL\_FACE);

  glCullFace(GL\_BACK);

}

// The usual main for a GLUT application.

int main(int argc, char\*\* argv) {

  glutInit(&argc, argv);

  glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB);

  glutInitWindowSize(500, 500);

  glutCreateWindow("The RGB Color Cube");

  glutReshapeFunc(reshape);

  glutTimerFunc(100, timer, 0);

  glutDisplayFunc(display);

  init();

  glutMainLoop();

}

Output:

