// lights.cpp : Defines the entry point for the console applic  
   
 #include<windows.h>  
 #include<GL/glut.h>  
 #include<stdlib.h>  
 #include<math.h>  
 #include<conio.h>  
 #include<stdio.h>  
 #include <iostream>  
 #include <iomanip>  
   
 using namespace std;  
   
 /\* This program demonstrates rendering a three dimensional cube in OpenGL. The program renders the cube in solid form  
 using the function Enable(GL\_DEPTH\_TEST) to activate the z-buffer to hide hidden surfaces. The  
 surfaces of the square are rendered by glBegin(GL\_POLYGON). This program also uses back face culling.  
   
 The program defines the cube as shiny plastic having reflective properties with the following statements  
   
 glEnable(GL\_COLOR\_MATERIAL);  
 glColorMaterial(GL\_FRONT,GL\_AMBIENT\_AND\_DIFFUSE);  
 glMaterialfv(GL\_FRONT,GL\_SPECULAR,specref);  
 glMateriali(GL\_FRONT,GL\_SHININESS,128);  
   
 Further a spotlight is set up and activated with this code  
 //focused spotlight with only 10 degrees one way  
 glLightf(GL\_LIGHT0,GL\_SPOT\_CUTOFF,10.0);  
 glLightf(GL\_LIGHT0,GL\_SPOT\_EXPONENT,15.0);  
 // point the light back to the origin  
 glLightfv(GL\_LIGHT0,GL\_SPOT\_DIRECTION,spotdir);  
 //enable the light  
 glEnable(GL\_LIGHT0);  
 //\*\*\*\*\*\*\*\*\*\*\* Global values\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 /\* These values are global because the timing call back functions will only take certain parameters  
 hence their needs to be global variables to communicate with these functions \*/  
 float static theta = 0.0, theta2 = 0;;//global angular value for rotation   
 float scale1 = 1.0;//global scaling value for square  
 float dx = 0.0, dy = 0.0, dz = 0.0;//global movement value for dx and dy/  
   
   
 void init(void);//this is a function to initialize the window clear color  
 void RenderScene(void);//this is a function to draw a square in an opened window  
 void loadicon(float[][4], float[][4], float[][4], float[][3], float[][3], float[], float[], float[]);  
 void calcNormals(float, float, float, float, float, float, float, float, float, float, float, float);  
 /\* loads the square icon \*/  
 void drawicon(float[][4], float[][4], float[][4], float[][3], float[][3], float[], float[], float[]);/\*  
 draws the icon \*/  
   
 void settrans2(void);/\* sets the translation matrix for the square  
 transformation matrix for desired scale, rotation,new pos\*/  
   
 /\*performs the transformation on the icon pattern \*/  
   
   
   
 void SetupRC(void);//sets up the clear color  
 void TimerFunction(int);//this call back function is call each 30 ms and changes the location,scale and rotation  
 // of the square.  
   
 //Main Program  
   
 int main(int argc, char\*\* argv)  
 {//set up window title  
   
   
 char header[] = "Lights and Cube by Joe Student";  
  
   
   
 glutInit(&argc, argv);  
 // Set up the display mode with a double buffer and a depth buffer and RGB colors  
 glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB | GLUT\_DEPTH);  
 SetupRC();  
 //Initialize window size and position  
 glutInitWindowSize(560, 440);  
 glutInitWindowPosition(140, 20);  
 // Open and Label Window   
 glutCreateWindow(header);  
 glutDisplayFunc(RenderScene);  
 glutTimerFunc(500, TimerFunction, 1);  
 glutMainLoop();  
   
 return 0;  
 }  
 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*RenderScene Function\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 void RenderScene(void)  
 {  
 float xdel = 0.25;  
   
 float x[6][4], y[6][4], z[6][4], fcolor[6][3], nvector[6][3], lx[2], ly[2], lz[2]; /\* these variables hold the  
 pattern for the square icon. Note that x,y,z hold the cube and lx,ly.lz hold a line  
 through the cube \*/  
 // set up light parameters  
 float ambientlight[] = { 1.0,1.0,1.0,1.0 };//strong white ambient light  
 float diffuselight[] = { 1.0,1.0,1.0,1.0 };//diffuse lighting  
 float specular[] = { 1.0,1.0,1.0,0.0 };//specular lighting  
 float lightpos[] = { -2.0,4.0,4.0,1.0 };//SEE CAUTIONARY NOTE BELOW FOR COORDINATE SYSTEM  
 float specref[] = { 1.0,1.0,1.0,1.0 };//set the reflectance of the material all is plastic  
 float spotdir[] = { 2.0,-4.0,-4.0 };//shine spot down on cube the light must shine toward the origin  
   
 //clear the window with the current background color  
 cout << "in renderscene" << endl;  
 glMatrixMode(GL\_PROJECTION);  
 glLoadIdentity();  
 //set the viewport to the window dimensions  
 glViewport(0, 0, 540, 440);  
 //Establish the clipping volume in user coordinates  
 glOrtho(-7.0, 7.0, -7.0, 7.0, -7.0, 7.0);  
 loadicon(x, y, z, fcolor, nvector, lx, ly, lz);  
 /\* draw the cube and line \*/  
   
 glEnable(GL\_DEPTH\_TEST);  
 //enable lighting  
 glEnable(GL\_LIGHTING);  
 glEnable(GL\_CULL\_FACE);  
 glFrontFace(GL\_CCW);  
 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*CAUTION DANGER WILL SMITH!!!! DANGER!!!\*\*\*\*\*\*\*\*\*\*\*\*  
   
 YOU MUST SWITCH TO MODELVIEW MATRIX MODE BEFORE YOU ENABLE THE LIGHT AND YOU MUST  
 THE REAL PROBLEM HERE SEEMS TO BE THE Angle of width of the spotlight beam described in  
 glLightf(GL\_LIGHT0,GL\_SPOT\_CUTOFF,20.0); values of 20 to 30 work best here. Values less than  
 20 seem to make the light too focused. Remember you must make the light wide enough to cover your object  
 else the polygon will not light. Finally pure colors of red, green or blue do not seem to reflect and  
 have a sepctular effect. Some mixture of these colors i.e.(0.5,0.4,0.3) will produce that spectular shine or flash  
 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*IGNORE THESE AT YOUR OWN RISK\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
 glMatrixMode(GL\_MODELVIEW);  
 glLoadIdentity();  
 // set light position, ambient, diffuse and specular strength  
 glLightfv(GL\_LIGHT0, GL\_POSITION, lightpos);  
 glLightfv(GL\_LIGHT0, GL\_AMBIENT, ambientlight);  
 glLightfv(GL\_LIGHT0, GL\_DIFFUSE, diffuselight);  
 glLightfv(GL\_LIGHT0, GL\_SPECULAR, specular);  
 //focused spotlight with only 10 degrees one way  
 glLightf(GL\_LIGHT0, GL\_SPOT\_CUTOFF, 60.0);  
 glLightf(GL\_LIGHT0, GL\_SPOT\_EXPONENT, 15.0);  
 // point the light back to the origin  
 glLightfv(GL\_LIGHT0, GL\_SPOT\_DIRECTION, spotdir);  
 //enable the light  
 glEnable(GL\_LIGHT0);  
   
   
 //now define the material properties  
 glEnable(GL\_COLOR\_MATERIAL);  
 glColorMaterial(GL\_FRONT, GL\_AMBIENT\_AND\_DIFFUSE);  
 glMaterialfv(GL\_FRONT, GL\_SPECULAR, specref);  
 glMateriali(GL\_FRONT, GL\_SHININESS, 128);  
 glClearColor(0.5, 0.5, 0.5, 1.0);  
 // Clear the window and the z buffer with the background color  
 glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);  
 settrans2();  
 //now draw the square  
 drawicon(x, y, z, fcolor, nvector, lx, ly, lz);  
 glFlush();  
   
   
 glEnd();  
   
 glutSwapBuffers();  
   
   
 return;  
   
 };//end of render scene  
 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Load Icon Function\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 void loadicon(float x[][4], float y[][4], float z[][4], float fcolor[][3], float nvector[][3], float xl[], float yl[], float zl[])  
 /\* this procedure loads a square icon \*/  
 {/\* load front face\*/  
 float ui, uj, uk;  
 x[0][0] = -1.0; y[0][0] = 1.0; z[0][0] = 1.0;  
 x[0][1] = -1.0; y[0][1] = -1.0; z[0][1] = 1.0;  
 x[0][2] = 1.0; y[0][2] = -1.0; z[0][2] = 1.0;  
 x[0][3] = 1.0; y[0][3] = 1.0; z[0][3] = 1.0;  
 /\* load the color on the front face red\*/  
 fcolor[0][0] = 0.250; fcolor[0][1] = 0.0; fcolor[0][2] = 0.25;  
 /\* load the normal to this face \*/  
 // nvector[0][0] = 0.0; nvector[0][1] = 0.0; nvector[0][2] = 1.0;  
 //// NOTE WHEN YOU calcNormals THE VECTORS MUST BE LOADED SO THAT RIGHT HAND RULE APPLIES ie windings must match  
 calcNormals(x[0][1], y[0][1], z[0][1], x[0][2], y[0][2], z[0][2], x[0][0], y[0][0], z[0][0], ui, uj, uk);  
 nvector[0][0] = ui; nvector[0][1] = uj; nvector[0][2] = uk;  
   
 /\* load the right side (x) face\*/  
 x[1][0] = 1.0; y[1][0] = 1.0; z[1][0] = 1.0;  
 x[1][1] = 1.0; y[1][1] = -1.0; z[1][1] = 1.0;  
 x[1][2] = 1.0; y[1][2] = -1.0; z[1][2] = -1.0;  
 x[1][3] = 1.0; y[1][3] = 1.0; z[1][3] = -1.0;  
 /\* load the color on the right side face green \*/  
 fcolor[1][0] =0.0;fcolor[1][1] = 1.0; fcolor[1][2] = 0.0;  
 // load the normal to this face pos x axis  
 nvector[1][0] = 1.0; nvector[1][1]=0.0; nvector[1][2] = 0.0;  
   
 /\* load the back side face \*/  
 x[2][0] = 1.0; y[2][0] = 1.0; z[2][0] = -1.0;  
 x[2][1] = 1.0; y[2][1] = -1.0; z[2][1] = -1.0;  
 x[2][2] = -1.0; y[2][2] = -1.0; z[2][2] = -1.0;  
 x[2][3] = -1.0; y[2][3] = 1.0; z[2][3] = -1.0;  
 /\*load the color on the back side blue \*/  
 fcolor[2][0]=0.30; fcolor[2][1] = 0.30; fcolor[2][2] = 0.30;  
 // load the normal to this face neg z axis  
 nvector[2][0] = 0.0; nvector[2][1] = 0.0;nvector[2][2]=-1.0;  
   
   
 /\* load the left side x face \*/  
 x[3][0] = -1.0; y[3][0] = 1.0; z[3][0] = 1.0;  
 x[3][1] = -1.0; y[3][1] = 1.0; z[3][1] = -1.0;  
 x[3][2] = -1.0; y[3][2] = -1.0; z[3][2] = -1.0;  
 x[3][3] = -1.0; y[3][3] = -1.0; z[3][3] = 1.0;  
 /\* load the color on the back side white \*/  
 fcolor[3][0] = 0.3; fcolor[3][1] = 0.0; fcolor[3][2] = 0.0;  
 // load the normal to this face neg x axis  
 nvector[3][0] = -1.0; nvector[3][1] = 0.0;nvector[3][2]=0.0;  
 calcNormals(x[3][1], y[3][1], z[3][1], x[3][0], y[3][0], z[3][0], x[3][2], y[3][2], z[3][2], ui, uj, uk);  
 //nvector[3][0] = ui; nvector[3][1] = uj; nvector[3][2] =uk;  
   
 /\*loat the top side\*/  
 x[4][0] = 1.0; y[4][0] = 1.0; z[4][0] = 1.0;  
 x[4][1] = 1.0; y[4][1] = 1.0; z[4][1] = -1.0;  
 x[4][2] = -1.0; y[4][2] = 1.0; z[4][2] = -1.0;  
 x[4][3] = -1.0; y[4][3] = 1.0; z[4][3] = 1.0;  
 /\* load the color on the top black \*/  
 fcolor[4][0] = 0.5; fcolor[4][1] = 0.5; fcolor[4][2] = 0.0;  
 // load the normal to this face pos y axis  
 nvector[4][0] = 0.0; nvector[4][1] = 1.0; nvector[4][2] = 0.0;  
   
   
 /\*load the bottom side \*/  
 x[5][0] = 1.0; y[5][0] = -1.0; z[5][0] = 1.0;  
 x[5][1] = -1.0; y[5][1] = -1.0; z[5][1] = 1.0;  
 x[5][2] = -1.0; y[5][2] = -1.0; z[5][2] = -1.0;  
 x[5][3] = 1.0; y[5][3] = -1.0; z[5][3] = -1.0;  
 /\* load the color on bottom yellow \*/  
 fcolor[5][0] = 0.0; fcolor[5][1] = 0.5; fcolor[5][2] = 0.5;  
 // load the normal to this face neg y axis  
 nvector[5][0] = 0.0; nvector[5][1] = -1.0; nvector[5][2] = 0.0;  
   
   
 /\*load the line \*/  
 xl[0] = 0.0; yl[0] = 3.0; zl[0] = 0.0;  
 xl[1] = 0.0; yl[1] = -3.0; zl[1] = 0.0;  
   
 return;  
 } /\* end of load icon \*/  
 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*function calcNormal\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
 void calcNormals(float x1, float y1, float z1, float x2, float y2, float z2, float x3, float y3, float z3, float ui, float uj, float uk)  
 {/\* this function calculates a normal vector to P1(x,y,z)-P2(x,y,z) and P1(x,y,z)-P3(x,y,z) \*/  
 float a1, a2, a3, b1, b2, b3, ix, jy, kz, ul;  
 a1 = x2 - x1;  
 a2 = y2 - y1;  
 a3 = z2 - z1;  
 b1 = x3 - x1;  
 b2 = y3 - y1;  
 b3 = z3 - z1;  
 ix = (a2\*b3 - a3\*b2);  
 jy = (a3\*b1 -a1\*b3);  
 kz = (a1\*b2 - a2\*b1);  
   
 // now calculate the values of the unit vector  
 ul = sqrtf(ix\*ix + jy\*jy + kz\*kz);  
 ui = ix / ul;  
 uj = jy / ul;  
 uk = kz / ul;  
 return;  
 }  
 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* function drawicon \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
   
 void drawicon(float x[][4], float y[][4], float z[][4], float fcolor[][3], float nvector[][3], float xl[], float yl[], float zl[])  
 {  
 /\* this function draws the cube at the transformed position \*/  
 int i, face;  
   
 for (face = 0; face <= 5; face++)  
 {// render each face of the cube  
 // note we are doing color tracking on the material color  
 glColor3f(fcolor[face][0], fcolor[face][1], fcolor[face][2]);  
   
 glBegin(GL\_POLYGON);  
 glNormal3f(nvector[face][0], nvector[face][1], nvector[face][2]);  
   
 for (i = 0; i <= 3; i++)  
 glVertex3f(x[face][i], y[face][i], z[face][i]);  
  
 glEnd();  
 }  
 glColor3f(10.0, 0.5, 0.5);  
 //render the line through the cube  
 glBegin(GL\_LINES);  
 glVertex3f(xl[0], yl[0], zl[0]);  
 glVertex3f(xl[1], yl[1], zl[1]);  
 glEnd();  
 glFlush();  
   
 return;  
 }//end of draw icon  
   
   
   
   
   
   
 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* function settrans2 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
 void settrans2(void)  
   
 //Sets the translation matrix for the cube  
 {  
 cout << "in settrans2" << endl;  
 glMatrixMode(GL\_MODELVIEW);  
 glLoadIdentity();  
 glTranslatef(dx, dy, dz);  
 glRotatef(theta, 0.0, 1.0, 0.0);// note that the angle theta is in degrees, not radians  
 glRotatef(theta2, 1.0, 1.0, 1.0);  
 return;  
   
 }  
   
   
 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Function SetupRC\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 // Setup the rendering state  
 void SetupRC(void)  
 {// this function sets the clear color of an open window and clears the open window  
 // Set clear color to green  
 glClearColor(0.0, 0.0, 1.0, 1.0);  
   
 return;  
 }//end of SetupRC  
   
 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Functioner Timer\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
 void TimerFunction(int value)  
 //this call back function is call each 30 ms and changes the location,scale and rotation  
 // of the square.  
 {  
 theta += 2.0;  
 theta2 += 5.0;  
   
 if (theta >= 720.0)theta = 0.0;  
   
   
 // Redraw the scene with new coordinates  
 glutPostRedisplay();  
 glutTimerFunc(33, TimerFunction, 1);  
 }