```
1  //veryNewestSoilTest
2  #include<windows.h>
3  #include<GL/glut.h>
4  #include<stdlib.h>
5  #include<math.h>
6  #include<conio.h>
7  #include<stdio.h>
8  #include <iostream>
9  #include <iomanip>
10 #include <gl/glut.h>
11 #include"External Libraries\SOIL2\include\SOIL2.h"
12 using namespace std;
13
14 /* This program demonstrates rendering a three dimensional trapazoid in OpenGL.
       The program renders the trapazoid in solid form
15 using the function Enable(GL_DEPTH_TEST) to activate the z-buffer to hide
     hidden surfaces.  The
16 surfaces of the square are rendered by glBegin(GL_POLYGON).  This program also
     uses back face culling. Finally this program
17 also considers texture on the square.  It is in fact "squares on squares".
18
19 The program creates two textures with a call for to SOIL and these textures
20
21
22 //*********** Global values************************************************
23 /* These values are global because the timing call back functions will only
     take certain parameters
24 hence their needs to be global variables to communicate with these functions */
25 float static theta = 0.0, theta2 = 0;;//global angular value for rotation
26 float scale1 = 1.0;//global scaling value for square
27 float dx = 0.0, dy = 0.0, dz = 0.0;//global movement value for dx and dy/
28 int nocolors = 1;//This is a switch to allow the trapazoid to be colored panels
      or white panels.  If the panels are white, the textures show up better
29
30
31
32 void init(void);//this is a function to initialize the window clear color
33 void RenderScene(void);//this is a function to draw a square in an opened
     window
34 void loadicon(float[][4], float[][4], float[][4], float[][3], float[][3], float
     [], float[], float[]);
35
36 /*   loads the square icon              */
37 void drawicon(float[][4], float[][4], float[][4], float[][3], float[][3], float
     [], float[], float[]);/*
38                                                                              
                            draws the icon                                   */
39 void drawlightsource(float[]);// draws the position of the light source
40
```

```
41 void settrans2(void);/* sets the translation matrix for the square
42                        transformation matrix for desired scale, rotation,new
                         pos*/
43
44                        /*performs the transformation on the icon pattern     */
45
46
47
48 void SetupRC(void);//sets up the clear color
49 void TimerFunction(int);//this call back function is call each 30 ms and
     changes the location,scale and rotation
50 GLuint textures[3];// This array will hold the two OpenGL Texture objects.  The
    call to SOIL creates these texture objects.
51
52
53                      //Main Program
54
55 int main(int argc, char* *argv)
56 {//set up window title
57
58
59     char header[] = "Square by Joe Student";
60
61
62
63     glutInit(&argc, argv);
64     // Set up the display mode with a double buffer and a depth buffer and RGB
        colors
65     glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
66     SetupRC();
67     //Initialize window size and position
68     glutInitWindowSize(560, 440);
69     glutInitWindowPosition(140, 20);
70     //  Open and Label Window
71     glutCreateWindow(header);
72     //Now create a couple of texture objects
73     //
74     textures[0] = SOIL_load_OGL_texture("butterfly.png", SOIL_LOAD_AUTO,
        SOIL_CREATE_NEW_ID, SOIL_FLAG_POWER_OF_TWO | SOIL_FLAG_INVERT_Y);
75     if (!textures[0])
76     {
77         printf("soil failed to load butterfly.png texture\n");
78         exit(0);
79     }
80     textures[1] = SOIL_load_OGL_texture("chrysanthemum.jpg", SOIL_LOAD_AUTO,
        SOIL_CREATE_NEW_ID, SOIL_FLAG_POWER_OF_TWO | SOIL_FLAG_INVERT_Y);
81     if (!textures[1])
82     {
83         printf("soil failed to load chrysanthemum.jpg texture\n");
```

```
84          exit(0);
85      }
86
87          textures[2] = SOIL_load_OGL_texture("Christmas_tree_ball_Icon_256.bmp",
               SOIL_LOAD_AUTO, SOIL_CREATE_NEW_ID, SOIL_FLAG_POWER_OF_TWO |
              SOIL_FLAG_INVERT_Y);
88      if (!textures[2])
89      {
90          printf("soil failed to load Christmas_tree_ball_Icon_256.bmp texture
              \n");
91          exit(0);
92      }
93      //Enable the texture state
94      glEnable(GL_TEXTURE_2D);
95
96
97
98      // clamp the image in the s direction and in the t direction
99
100     glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP);
101     glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP);
102     // Interpolate to the nearest pixel for color outside of image
103     glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
104     glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
105
106
107     glutDisplayFunc(RenderScene);
108     glutTimerFunc(500, TimerFunction, 1);
109     //Now draw the scene
110
111     glutMainLoop();
112
113     return 0;
114 }
115 //************************RenderScene Function************************
116 void RenderScene(void)
117 {
118     float xdel = 0.25;
119
120     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
121
              pattern for the square icon.  Note that x,y,z hold the cube and
    lx,ly.lz hold a line
122
              through the cube            */
123
              // set up light parameters
124     float ambientlight[] = { 1.0,1.0,1.0,1.0 };//strong white ambient light
```

```
125     float diffuselight[] = { 1.0,1.0,1.0,1.0 };//diffuse lighting
126     float specular[] = { 1.0,1.0,1.0,1.0 };//specular lighting
127     float lightpos[] = { 2.0,4.0,4.0,1.0 };//SEE CAUTIONARY NOTE BELOW FOR
          COORDINATE SYSTEM
128     float specref[] = { 1.0,1.0,1.0,1.0 };//set the reflectance of the material
           all is plastic
129     float spotdir[] = { -2.0,-4.0,-4.0 };//shine spot down on cube the light
          must shine toward the origin
130
131                                           //clear the window with the current
                    background color
132     cout << "in renderscene" << endl;
133
134
135     glMatrixMode(GL_PROJECTION);
136     glLoadIdentity();
137     //set the viewport to the window dimensions
138     glViewport(0, 0, 540, 440);
139     //Establish the clipping volume in user coordinates
140     glOrtho(-7.0, 7.0, -7.0, 7.0, 10.0, -10.0);
141
142
143     loadicon(x, y, z, fcolor, nvector, lx, ly, lz);
144     /*     draw the cube and line      */
145
146     glEnable(GL_DEPTH_TEST);
147
148     glEnable(GL_LIGHTING);
149     glEnable(GL_CULL_FACE);
150     glFrontFace(GL_CCW);
151
152
153     /*****************************CAUTION DANGER WILL SMITH!!!! DANGER!!!
          ************
154
155     YOU MUST SWITCH TO MODELVIEW MATRIX MODE BEFORE YOU ENABLE THE LIGHT AND
          YOU MUST LOAD A NEW IDENTITY
156     IDENTITY MATRIX.  IF YOU DO NOT DO THIS AND YOU MOVE THE ICON LATER.  THE
          LIGHT WILL FOLLOW T
157     THE ICON.  ALSO NOTE THAT THE COORDINATE SYSTEM FOR.
158     POSITIVE X IS TO THE Right, POSITIVE Y IS UP AND POSITIVE Z IS TOWARD THE
          VIEWER OUT OF THE
159     SCREEN
160
161     ****************************IGNORE THESE AT YOUR OWN
          RISK*****************************/
162     glMatrixMode(GL_MODELVIEW);
163     glLoadIdentity();
164     // set light position, ambient, diffuse and specular strength
```

```
165     glLightfv(GL_LIGHT0, GL_POSITION, lightpos);
166     glLightfv(GL_LIGHT0, GL_AMBIENT, ambientlight);
167     glLightfv(GL_LIGHT0, GL_DIFFUSE, diffuselight);
168     glLightfv(GL_LIGHT0, GL_SPECULAR, specular);
169     //focused spotlight with only 10 degrees one way
170     glLightf(GL_LIGHT0, GL_SPOT_CUTOFF, 40.0);
171     glLightf(GL_LIGHT0, GL_SPOT_EXPONENT, 15.0);
172     // point the light back to the origin
173     glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION, spotdir);
174     //enable the light
175     glEnable(GL_LIGHT0);
176     //enable lighting
177     glEnable(GL_LIGHTING);
178
179     //now define the material properties
180     glEnable(GL_COLOR_MATERIAL);
181     glColorMaterial(GL_FRONT, GL_AMBIENT_AND_DIFFUSE);
182     glMaterialfv(GL_FRONT, GL_SPECULAR, specref);
183     glMateriali(GL_FRONT, GL_SHININESS, 128);
184     glClearColor(0.5, 0.5, 0.5, 1.0);
185     // Clear the window and the z buffer with the background color
186     glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
187
188
189     settrans2();
190     //now draw the square
191     drawicon(x, y, z, fcolor, nvector, lx, ly, lz);
192     //now draw the light source
193     drawlightsource(lightpos);
194     glMatrixMode(GL_MODELVIEW);
195     glLoadIdentity();
196
197
198
199
200     glEnd();
201
202     glutSwapBuffers();
203
204
205     return;
206
207 };//end of render scene
208   //*****************************Load Icon                                ↵
        Function***********************************
209 void loadicon(float x[][4], float y[][4], float z[][4], float fcolor[][3],     ↵
      float nvector[][3], float xl[], float yl[], float zl[])
210 /*   this procedure loads a trapzoidal icon            */
211 {/* load front face*/
```

```
212     x[0][0] = -1.0; y[0][0] = 1.0; z[0][0] = 1.0;
213     x[0][1] = -2.0; y[0][1] = -1.0; z[0][1] = 1.0;
214     x[0][2] = 2.0; y[0][2] = -1.0; z[0][2] = 1.0;
215     x[0][3] = 1.0; y[0][3] = 1.0; z[0][3] = 1.0;
216     /* load the color on the front face red*/
217     fcolor[0][0] = 1.0; fcolor[0][1] = 0.0; fcolor[0][2] = 0.0;
218     /* load the normal to this face */
219     nvector[0][0] = 0.0; nvector[0][1] = 0.0; nvector[0][2] = 1.0;
220
221     /* load the right side (x) face*/
222     x[1][0] = 1.0; y[1][0] = 1.0; z[1][0] = 1.0;
223     x[1][1] = 2.0; y[1][1] = -1.0; z[1][1] = 1.0;
224     x[1][2] = 2.0; y[1][2] = -1.0; z[1][2] = -1.0;
225     x[1][3] = 1.0; y[1][3] = 1.0; z[1][3] = -1.0;
226     /* load the color on the right side face green */
227     fcolor[1][0] = 0.0; fcolor[1][1] = 1.0; fcolor[1][2] = 0.0;
228     // load the normal to this face pos x axis
229     nvector[1][0] = 1.0; nvector[1][1] = 0.0; nvector[1][2] = 0.0;
230
231     /* load the back side face */
232     x[2][0] = 1.0; y[2][0] = 1.0; z[2][0] = -1.0;
233     x[2][1] = 2.0; y[2][1] = -1.0; z[2][1] = -1.0;
234     x[2][2] = -2.0; y[2][2] = -1.0; z[2][2] = -1.0;
235     x[2][3] = -1.0; y[2][3] = 1.0; z[2][3] = -1.0;
236     /*load the color on the back side blue */
237     fcolor[2][0] = 0.0; fcolor[2][1] = 0.0; fcolor[2][2] = 1.0;
238     // load the normal to this face neg z axis
239     nvector[2][0] = 0.0; nvector[2][1] = 0.0; nvector[2][2] = -1.0;
240
241
242     /* load the left side x face */
243     x[3][0] = -1.0; y[3][0] = 1.0; z[3][0] = 1.0;
244     x[3][1] = -1.0; y[3][1] = 1.0; z[3][1] = -1.0;
245     x[3][2] = -2.0; y[3][2] = -1.0; z[3][2] = -1.0;
246     x[3][3] = -2.0; y[3][3] = -1.0; z[3][3] = 1.0;
247     /* load the color on the back side white */
248     fcolor[3][0] = 1.0; fcolor[3][1] = 1.0; fcolor[3][2] = 1.0;
249     // load the normal to this face neg x axis
250     nvector[3][0] = -1.0; nvector[3][1] = 0.0; nvector[3][2] = 0.0;
251
252     /*loat the top side*/
253     x[4][0] = 1.0; y[4][0] = 1.0; z[4][0] = 1.0;
254     x[4][1] = 1.0; y[4][1] = 1.0; z[4][1] = -1.0;
255     x[4][2] = -1.0; y[4][2] = 1.0; z[4][2] = -1.0;
256     x[4][3] = -1.0; y[4][3] = 1.0; z[4][3] = 1.0;
257     /* load the color on the top  black */
258     fcolor[4][0] = 0.5; fcolor[4][1] = 0.5; fcolor[4][2] = 0.0;
259     // load the normal to this face pos y axis
260     nvector[4][0] = 0.0; nvector[4][1] = 1.0; nvector[4][2] = 0.0;
```

```
261
262
263     /*load the bottom side */
264     x[5][0] = 2.0; y[5][0] = -1.0; z[5][0] = 1.0;
265     x[5][1] = -2.0; y[5][1] = -1.0; z[5][1] = 1.0;
266     x[5][2] = -2.0; y[5][2] = -1.0; z[5][2] = -1.0;
267     x[5][3] = 2.0; y[5][3] = -1.0; z[5][3] = -1.0;
268     /* load the color on bottom yellow */
269     fcolor[5][0] = 0.0; fcolor[5][1] = 0.5; fcolor[5][2] = 0.5;
270     // load the normal to this face neg y axis
271     nvector[5][0] = 0.0; nvector[5][1] = -1.0; nvector[5][2] = 0.0;
272
273
274     /*load the line  */
275     xl[0] = 0.0; yl[0] = 3.0; zl[0] = 0.0;
276     xl[1] = 0.0; yl[1] = -3.0; zl[1] = 0.0;
277
278     return;
279 }    /*     end of load icon       */
280      /************************ function drawicon  ***************************/
281
282 void drawicon(float x[][4], float y[][4], float z[][4], float fcolor[][3],
      float nvector[][3], float xl[], float yl[], float zl[])
283 {
284     /*     this function draws the cube at the transformed position
          */
285     //float s1[4]={0.0,0.0,1.0,1.0},t1[4]={0.0,1.0,1.0,0.0};
286     float s[6][4], t[6][4];    int i, face;
287
288     // load face 0 the red face by hand.
289     s[0][0] = 0.25; t[0][0] = 1.0;
290     s[0][1] = 0.0; t[0][1] = 0.0;
291     s[0][2] = 1.0; t[0][2] = 0.0;
292     s[0][3] = 0.75; t[0][3] = 1.0;
293     // now map every other texture on the 4 conrners of the figure.
294     //loading face 1
295     s[1][0] = 0.0; t[1][0] = 0.0;
296     s[1][1] = 0.0; t[1][1] = 1.0;
297     s[1][2] = 1.0; t[1][2] = 1.0;
298     s[1][3] = 1.0; t[1][3] = 0.0;
299     // loading face 2
300     s[2][0] = 0.0; t[2][0] = 0.0;
301     s[2][1] = 0.25; t[2][1] = 1.0;
302     s[2][2] = 0.75; t[2][2] = 1.0;
303     s[2][3] = 1.0; t[2][3] = 0.0;
304
305     //loading face 3
306     s[3][0] = 0.0; t[3][0] = 0.0;
307     s[3][1] = 0.0; t[3][1] = 1.0;
```

```
308      s[3][2] = 1.0; t[3][2] = 1.0;
309      s[3][3] = 1.0; t[3][3] = 0.0;
310      //loading face 4
311      s[4][0] = 0.0; t[4][0] = 0.0;
312      s[4][1] = 0.0; t[4][1] = 1.0;
313      s[4][2] = 1.0; t[4][2] = 1.0;
314      s[4][3] = 1.0; t[4][3] = 0.0;
315      //loading face 5
316      s[5][0] = 0.0; t[5][0] = 0.0;
317      s[5][1] = 0.0; t[5][1] = 1.0;
318      s[5][2] = 1.0; t[5][2] = 1.0;
319      s[5][3] = 1.0; t[5][3] = 0.0;
320
321
322
323      for (face = 0; face <= 5; face++)
324      {// render each face of the cube
325       // Decide which texture we want bound to this face.
326          if (face <= 1)glBindTexture(GL_TEXTURE_2D, textures[0]);
327          else if((face>=2)&&(face<=3)) glBindTexture(GL_TEXTURE_2D, textures
              [1]);
328          else glBindTexture(GL_TEXTURE_2D, textures[2]);
329
330          if (nocolors == 1) glColor3f(1.0, 1.0, 1.0);
331          else
332              glColor3f(fcolor[face][0], fcolor[face][1], fcolor[face][2]);
333
334          glBegin(GL_POLYGON);
335          glNormal3f(nvector[face][0], nvector[face][1], nvector[face][2]);
336
337          for (i = 0; i <= 3; i++)
338          {//Place the texture coordinate on the surface of the cube clamp it on
              this vertex corner.  Note as we move around the
339           // s[i] varies from 0.0 to 1.0 and t[i] varies from 0.0 to 1.0
340
341              glTexCoord2f(s[face][i], t[face][i]);
342              glVertex3f(x[face][i], y[face][i], z[face][i]);
343          }
344          glEnd();
345      }//end of textured face build.
346
347
348       //render the line through the cube
349      glBegin(GL_LINES);
350      glVertex3f(xl[0], yl[0], zl[0]);
351      glVertex3f(xl[1], yl[1], zl[1]);
352      glEnd();
353
354      return;
```

```
355 }//end of draw icon
356
357 void drawlightsource(float lightxyz[])
358 {// this function draws a light at the position held in the array lightxyz
359     glMatrixMode(GL_MODELVIEW);
360     glLoadIdentity();
361     glTranslatef(lightxyz[0], lightxyz[1], lightxyz[2]);
362     glColor3f(1.0, 1.0, 1.0);//white light at this position
363     glutSolidSphere(0.5, 10, 10);
364
365
366     return;
367 }//end of drawlightsource
368
369
370
371
372
373
374  /*************************  function settrans2  **********************/
375 void settrans2(void)
376
377 //Sets the translation matrix for the cube
378 {
379     cout << "in settrans2" << endl;
380     glMatrixMode(GL_MODELVIEW);
381     glLoadIdentity();
382     glTranslatef(dx, dy, dz);
383     glRotatef(theta, 0.0, 1.0, 0.0);// note that the angle theta is in degrees,
        not radians
384     glRotatef(theta2, 1.0, 1.0, 1.0);
385     return;
386
387 }
388
389
390 //***************************Function
      SetupRC***********************************
391 // Setup the rendering state
392 void SetupRC(void)
393 {// this function sets the clear color of an open window and clears the open
      window
394  // Set clear color to green
395     glClearColor(0.0, 0.0, 1.0, 1.0);
396
397     return;
398 }//end of SetupRC
399
400  /****************************** 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);
}

```