Белгородский Государственный Технологический Университет им. В. Г. Шухова

Кафедра программного обеспечения вычислительной техники  
и автоматизированных систем

## РГЗ по компьютерной графике на тему: «Пространственная графика в OpenGL»

**Выполнил:**  
студент группы ПВ-41  
Адаменко И. И.

**Проверил:**  
старший преподаватель  
Осипов О. В.

Белгород  
2015

**Цель работы:** закрепление навыков использования примитивов в пространстве, использования текстур и разработка приложения с использованием OpenGL.

# Задание

Необходимо изобразить на сцене стол (с наложенной текстурой), на столе разместить какие-нибудь объекты по своему усмотрению, но обязательно нам должны быть несколько книг (с наложенными текстурами; разбросать или стопкой), шахматная доска и на ней несколько шахматных фигур (обязательно, чтобы фигуры были отдельно от доски). Необходимо также реализовать возможность поворота камеры на сцене. Источники света: точечный (несколько), направленный, фоновый.

Также сделать на сцене несколько полупрозрачных объектов, корректно расположенных на столе.

# Теоретическая часть

OpenGL (Open Graphic Library) — создана в 1992 году корпорацией Silicon Graphics. OpenGL — это универсальная аппаратно-независимая библиотека, которая поддерживает разнообразные 3D-объекты и конструкции, начиная с примитивов типа треугольника или линии и заканчивая сложными поверхностями. OpenGL — это всего лишь спецификация и все детали её реализации возлагаются на производителей аппаратного обеспечения.

В OpenGL есть механизм расширений, позволяющий добавлять в библиотеку какие-то функции, нереализованные базовой версией API. Начиная с версии 2.0 OpenGL получил встроенное дополнение под названием OpenGL Shader Language. Что позволило программистам выбирать между использованием жёстко прописанных функций и программируемыми шейдерами.

Важной особенностью OpenGL является кроссплатформенность. Она доступна на многих ОС, поэтому ядро OpenGL не поддерживает никаких функций для обработки окон.

В трехмерной графике все объекты представляются набором полигонов (многоугольников) соединенных в форме объекта. Каждую конечность полигона называют вершиной (vertex). Вершины — это ядро трёхмерной графики, поскольку они несут всю необходимую для представления информацию, такую как местоположение, цвет и данные текстуры. На вход поступают данные, описывающие трёхмерную сцену, а выходом является двухмерная растровая картинка.

Объект выводится в несколько этапов. Первый из них — триангуляция. Триангуляция — разбиение на треугольники. Треугольник является простейшим полигоном, вершины которого однозначно задают плоскость.

Любую область или поверхность можно гарантированно разбить на треугольники. Вычислительная сложность алгоритмов разбиения на треугольники существенно меньше, чем при использовании других полигонов.

Для треугольника легко определить три его ближайших соседа, имеющих с ним общие грани.

В OpenGL вывод треугольника на экран делается следующим образом:

1. glBegin(GL\_TRIANGLE);
2. glVertex3f(10.0f, 5.0f, 3.0f);
3. /\* 3 — трёхмерная графика, f — параметры типа float \*/
4. glVertex3f(..);
5. glVertex3f(..);
6. glEnd();

У вершины могут быть дополнительные свойства, помимо её местоположения. Например, вершине может быть назначен цвет или с ней может быть связана нормаль. OpenGL обладает значительной гибкостью и позволяет конструировать собственные форматы вершин. Другими словами, они позволяют нам указать, какая информация будет содержаться в данных вершинах. Для задания цвета используется функция glColor:

1. glColor3f(1.0f, 0.0f, 0.0f)
2. /\* цифры задают rgb [0..1] \*/

Пример рисования квадрата на плоскости x = 1, с указанием вектора нормали. Вектор имеет координаты (1; 0; 0):

1. glBegin(GL\_QUADS);
2. glNormal3f(1.0f, .0f, .0f);
3. glVertex3f(1.0f, .0f, .0f);
4. glVertex3f(1.0f, .0f, 1.0f);
5. glVertex3f(1.0f, 1.0f, 1.0f);
6. glVertex3f(1.0f, 1.0f, .0f);
7. glEnd();

В некоторых случаях, если накладывается текстура на объект, то необходимо указывать её координаты.

# Практическая часть

## Код программы

main.cpp — основной файл программы:

1. #include <Windows.h>
2. #include <GL/GLU.H>
3. #include <glut.h>
4. #include <math.h>
5. #include <stdio.h>
6. #include "shapes.h"
8. **void** display(**void**);
9. **void** reshape(**int** w, **int** h);
10. **void** keypress(unsigned **char** key, **int** x, **int** y);
11. **void** keypressSpecial(**int** key, **int** x, **int** y);
12. **void** keyunpress(**int** key, **int** x, **int** y);
13. **void** timer(**int** value);
14. **void** checkDeltas();
15. **void** light();
17. **float** angleCamera = 0,
19. xCameraView = 0,
20. yCameraView = 0,
21. zCameraView = -.5,
23. xCamera = 5,
24. yCamera = 1.3,
25. zCamera = 7,
27. dAngle = 0,
28. dPosition = 0,
30. cameraMoveSpeed = 3,
31. cameraUpDownSpeed = .3,
32. cameraLeftRightSpeed = .06;
34. **bool** light0 = **false**,
35. light1 = **false**,
36. light2 = **false**,
37. light3 = **false**;
39. **int** king,
40. pawn,
41. queen,
42. rook;
44. **int** main(**int** argc, **char** \*\*argv) {
45. glutInit(&argc, argv);
47. glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_DEPTH);
49. glutInitWindowSize(640, 480);
50. glutInitWindowPosition(0, 0);
51. glutCreateWindow("РГЗ");
53. glClearColor(0, 0, 0, 1);
54. glClearDepth(1);
56. glEnable(GL\_DEPTH\_TEST);
57. glDepthFunc(GL\_LEQUAL);
59. glEnable(GL\_BLEND);

62. glShadeModel(GL\_SMOOTH);
63. glHint(GL\_PERSPECTIVE\_CORRECTION\_HINT, GL\_NICEST);
65. glEnable(GL\_LIGHTING);
66. glLightModelf(GL\_LIGHT\_MODEL\_LOCAL\_VIEWER, GL\_TRUE);
67. glEnable(GL\_NORMALIZE);
69. king    = loadObject("king.obj");
70. pawn    = loadObject("pawn.obj");
71. queen   = loadObject("queen.obj");
72. rook    = loadObject("rook.obj");
74. glutDisplayFunc(display);
75. glutReshapeFunc(reshape);
76. glutIdleFunc(display);
77. glutKeyboardFunc(keypress);
78. glutSpecialFunc(keypressSpecial);
79. glutSpecialUpFunc(keyunpress);
80. glutTimerFunc(0, timer, 0);
82. glutMainLoop();
84. **return** 0;
85. }
87. **void** display(**void**) {
88. glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);
89. glMatrixMode(GL\_MODELVIEW);
90. glLoadIdentity();
92. checkDeltas();
94. gluLookAt(xCamera,
95. yCamera,
96. zCamera,
97. xCamera + xCameraView,
98. yCamera + yCameraView,
99. zCamera + zCameraView,
100. 0, 1, 0);
102. glPushMatrix();
103. renderTable();
104. glPopMatrix();
106. glPushMatrix();
107. renderBook(0, 0);
108. glPopMatrix();
110. glPushMatrix();
111. renderBook(25, 1);
112. glPopMatrix();
114. glPushMatrix();
115. renderChessBoard();
116. glPopMatrix();
118. glPushMatrix();
119. glTranslatef(6.21, .2, -2.43);
120. renderChess(pawn, **false**);
121. glPopMatrix();
123. glPushMatrix();
124. glTranslatef(6.66, .2, -2.665);
125. renderChess(queen, **false**);
126. glPopMatrix();
128. glPushMatrix();
129. glTranslatef(7.11, .2, -2.665);
130. renderChess(king, **false**);
131. glPopMatrix();
133. glPushMatrix();
134. glTranslatef(7.335, .2, -3.135);
135. renderChess(rook, **false**);
136. glPopMatrix();
138. glPushMatrix();
139. renderPyramid(1);
140. glPopMatrix();
142. glPushMatrix();
143. renderPyramid(2);
144. glPopMatrix();
146. glPushMatrix();
147. renderPyramid(3);
148. glPopMatrix();
150. glPushMatrix();
151. renderPyramid(4);
152. glPopMatrix();
154. glPushMatrix();
155. renderPyramid(5);
156. glPopMatrix();
158. glPushMatrix();
159. light();
160. glPopMatrix();
162. glutSwapBuffers();
163. }
165. **void** reshape(**int** w, **int** h) {
166. **if** (h == 0) h = 1;
167. **float** aspectRatio = (**float**)w / h;
169. glViewport(0, 0, w, h);
171. glMatrixMode(GL\_PROJECTION);
172. glLoadIdentity();
174. gluPerspective(45, aspectRatio, .1, 100);
176. glMatrixMode(GL\_MODELVIEW);
177. }
179. **void** timer(**int** value) {
180. glutPostRedisplay();
181. glutTimerFunc(15, timer, 0);
182. }
184. **void** keypress(unsigned **char** key, **int** x, **int** y) {
185. **switch** (key) {
186. **case** '1':
187. light0 = !light0;
188. **break**;
189. **case** '2':
190. light1 = !light1;
191. **break**;
192. **case** '3':
193. light2 = !light2;
194. **break**;
195. **case** '4':
196. light3 = !light3;
197. **break**;
198. }
199. }
201. **void** keypressSpecial(**int** key, **int** x, **int** y) {
202. **switch** (key) {
203. **case** GLUT\_KEY\_LEFT:
204. dAngle = -cameraLeftRightSpeed;
205. **break**;
206. **case** GLUT\_KEY\_RIGHT:
207. dAngle = cameraLeftRightSpeed;
208. **break**;
209. **case** GLUT\_KEY\_UP:
210. dPosition = cameraMoveSpeed;
211. **break**;
212. **case** GLUT\_KEY\_DOWN:
213. dPosition = -cameraMoveSpeed;
214. **break**;
215. **case** GLUT\_KEY\_PAGE\_UP:
216. yCameraView += cameraUpDownSpeed;
217. **break**;
218. **case** GLUT\_KEY\_PAGE\_DOWN:
219. yCameraView -= cameraUpDownSpeed;
220. **break**;
221. **case** GLUT\_KEY\_HOME:
222. yCamera += cameraUpDownSpeed;
223. **break**;
224. **case** GLUT\_KEY\_END:
225. yCamera -= cameraUpDownSpeed;
226. **break**;
227. }
228. }
230. **void** keyunpress(**int** key, **int** x, **int** y) {
231. **switch** (key) {
232. **case** GLUT\_KEY\_LEFT:
233. **case** GLUT\_KEY\_RIGHT:
234. dAngle = 0;
235. **break**;
236. **case** GLUT\_KEY\_UP:
237. **case** GLUT\_KEY\_DOWN:
238. dPosition = 0;
239. **break**;
240. }
241. }
243. **void** checkDeltas() {
244. **if** (dAngle) {
245. angleCamera += dAngle;
246. xCameraView = sin(angleCamera);
247. zCameraView = -cos(angleCamera);
248. }
250. **if** (dPosition) {
251. xCamera += dPosition \* xCameraView \* .1;
252. zCamera += dPosition \* zCameraView \* .1;
253. }
254. }
256. **void** light() {
257. **if** (light0) {
258. // направленный
259. **float** ambient0[] = {3, 3, 3, 1};
260. glLightModelfv(GL\_LIGHT\_MODEL\_AMBIENT, ambient0);
261. } **else** {
262. **float** ambient0[] = {0, 0, 0, 0};
263. glLightModelfv(GL\_LIGHT\_MODEL\_AMBIENT, ambient0);
264. }
266. **if** (light1) {
267. // точечный
268. **float** diffuse1[] = {1, 1, 1},
269. position1[] = {0, 0, 1, 1};
270. glEnable(GL\_LIGHT1);
271. glLightfv(GL\_LIGHT1, GL\_DIFFUSE, diffuse1);
272. glLightfv(GL\_LIGHT1, GL\_POSITION, position1);
273. } **else** {
274. glDisable(GL\_LIGHT1);
275. }
277. **if** (light2) {
278. // ещё один точечный
279. **float** diffuse2[] = {1, 1, 1},
280. position2[] = {1, 1, 3, 1};
282. glEnable(GL\_LIGHT2);
283. glLightfv(GL\_LIGHT2, GL\_DIFFUSE, diffuse2);
284. glLightfv(GL\_LIGHT2, GL\_POSITION, position2);
285. } **else** {
286. glDisable(GL\_LIGHT2);
287. }
289. **if** (light3) {
290. **float** diffuse4[] = {1, 1, 1},
291. position4[] = {1, 0, 6, 1},
292. spot\_direction4[] = {1, 0, -1};
294. glEnable(GL\_LIGHT3);
295. glLightfv(GL\_LIGHT3, GL\_DIFFUSE, diffuse4);
296. glLightfv(GL\_LIGHT3, GL\_POSITION, position4);
297. glLightf(GL\_LIGHT3, GL\_SPOT\_CUTOFF, 40);
298. glLightfv(GL\_LIGHT3, GL\_SPOT\_DIRECTION, spot\_direction4);
299. glLightf(GL\_LIGHT3, GL\_SPOT\_EXPONENT, 30);
300. } **else** {
301. glDisable(GL\_LIGHT3);
302. }
303. }

shapes.cpp — дополнительный файл, реализующий работу с моделями и фигурами:

1. #include <glut.h>
2. #include <stdio.h>
3. #include <stdlib.h>
4. #include <string.h>
5. #include <algorithm>
6. #include <string>
7. #include <iostream>
8. #include <vector>
9. #include <fstream>
10. #include <cstdio>
11. #include "shapes.h"
13. #ifndef GL\_BGR
14. #define GL\_BGR 0x80E0
15. #endif
17. // colors
18. **float** red[4] = {1, 0, 0, 1};
19. **float** semired[4] = {1, 0, 0, .5};
20. **float** green[4] = {0, 1, 0, 1};
21. **float** semigreen[4] = {0, 1, 0, .5};
22. **float** blue[4] = {0, 0, 1, 1};
23. **float** semiblue[4] = {0, 0, 1, .5};
24. **float** yellow[4] = {1, 1, 0, 1};
25. **float** semiyellow[4] = {1, 1, 0, .5};
26. **float** violet[4] = {1, 0, 1, 1};
27. **float** semiviolet[4] = {1, 0, 1, .5};
28. **float** cyan[4] = {0, 1, 1, 1};
29. **float** semicyan[4] = {0, 1, 1, .5};
31. **struct** coordinate {
32. **float** x, y, z;
33. coordinate(**float** a, **float** b, **float** c) : x(a), y(b), z(c) {};
34. };
36. **struct** face {
37. **int** facenum;
38. **bool** four;
39. **int** faces[4];
40. face(**int** facen, **int** f1, **int** f2, **int** f3) : facenum(facen) {
41. faces[0] = f1;
42. faces[1] = f2;
43. faces[2] = f3;
44. four = **false**;
45. }
47. face(**int** facen, **int** f1, **int** f2, **int** f3, **int** f4) : facenum(facen) {
48. faces[0] = f1;
49. faces[1] = f2;
50. faces[2] = f3;
51. faces[3] = f4;
52. four = **true**;
53. }
54. };
56. **void** renderTable() {
57. glEnable(GL\_TEXTURE\_2D);
59. **float** height = .5,
60. length = 10,
61. width = 5;
63. unsigned **int** texture = loadTexture("wood\_edge.bmp");
64. glBindTexture(GL\_TEXTURE\_2D, texture);
65. glBegin(GL\_QUADS);
66. // left
67. glTexCoord2d(0, 1);
68. glVertex3f(0, 0, 0);
69. glTexCoord2d(1, 1);
70. glVertex3f(0, -height, 0);
71. glTexCoord2d(1, 0);
72. glVertex3f(0, -height, -width);
73. glTexCoord2d(0, 0);
74. glVertex3f(0, 0, -width);
76. // front
77. glTexCoord2d(1, 0);
78. glVertex3f(0, 0, 0);
79. glTexCoord2d(1, 1);
80. glVertex3f(length, 0, 0);
81. glTexCoord2d(0, 1);
82. glVertex3f(length, -height, 0);
83. glTexCoord2d(0, 0);
84. glVertex3f(0, -height, 0);
86. // right
87. glTexCoord2d(1, 0);
88. glVertex3f(length, 0, 0);
89. glTexCoord2d(1, 1);
90. glVertex3f(length, 0, -width);
91. glTexCoord2d(0, 1);
92. glVertex3f(length, -height, -width);
93. glTexCoord2d(0, 0);
94. glVertex3f(length, -height, 0);
96. // back
97. glTexCoord2d(1, 1);
98. glVertex3f(length, 0, -width);
99. glTexCoord2d(0, 1);
100. glVertex3f(0, 0, -width);
101. glTexCoord2d(0, 0);
102. glVertex3f(0, -height, -width);
103. glTexCoord2d(1, 0);
104. glVertex3f(length, -height, -width);
105. glEnd();
106. glDeleteTextures(1, &texture);
108. texture = loadTexture("wood\_surface.bmp");
109. glBindTexture(GL\_TEXTURE\_2D, texture);
110. glBegin(GL\_QUADS);
111. // top
112. glTexCoord2d(0, 0);
113. glVertex3f(0, 0, 0);
114. glTexCoord2d(0, 1);
115. glVertex3f(0, 0, -width);
116. glTexCoord2d(1, 1);
117. glVertex3f(length, 0, -width);
118. glTexCoord2d(1, 0);
119. glVertex3f(length, 0, 0);
121. // bottom
122. glTexCoord2d(1, 0);
123. glVertex3f(0, -height, -width);
124. glTexCoord2d(1, 1);
125. glVertex3f(0, -height, 0);
126. glTexCoord2d(0, 1);
127. glVertex3f(length, -height, 0);
128. glTexCoord2d(0, 0);
129. glVertex3f(length, -height, -width);
130. glEnd();
131. glDeleteTextures(1, &texture);
132. }
134. **void** renderBook(**float** angle, unsigned **int** num) {
135. glEnable(GL\_TEXTURE\_2D);
137. **float** height = .3,
138. length = 3,
139. width = 1.5;
141. **if** (angle != 0) {
142. glTranslatef(length / 2, 0, width / 2);
143. glRotatef(angle, 0, 1, 0);
144. }
146. glTranslatef(1, height \* num, -1);
148. unsigned **int** textureFront = loadTexture("book\_front.bmp");
149. glBindTexture(GL\_TEXTURE\_2D, textureFront);
150. glBegin(GL\_QUADS);
151. glTexCoord2d(1, 0);
152. glVertex3f(0, height, 0);
153. glTexCoord2d(1, 1);
154. glVertex3f(0, height, -width);
155. glTexCoord2d(0, 1);
156. glVertex3f(length, height, -width);
157. glTexCoord2d(0, 0);
158. glVertex3f(length, height, 0);
159. glEnd();
161. unsigned **int** textureBack = loadTexture("book\_back.bmp");
162. glBindTexture(GL\_TEXTURE\_2D, textureBack);
163. glBegin(GL\_QUADS);
164. glTexCoord2d(0, 1);
165. glVertex3f(0, 0, 0);
166. glTexCoord2d(1, 1);
167. glVertex3f(length, 0, 0);
168. glTexCoord2d(0, 0);
169. glVertex3f(length, 0, -width);
170. glTexCoord2d(1, 0);
171. glVertex3f(0, 0, -width);
172. glEnd();
174. unsigned **int** textureSpine = loadTexture("book\_spine.bmp");
175. glBindTexture(GL\_TEXTURE\_2D, textureSpine);
176. glBegin(GL\_QUADS);
177. glTexCoord2d(0, 0);
178. glVertex3f(0, 0, 0);
179. glTexCoord2d(1, 0);
180. glVertex3f(0, height, 0);
181. glTexCoord2d(1, 1);
182. glVertex3f(length, height, 0);
183. glTexCoord2d(0, 1);
184. glVertex3f(length, 0, 0);
185. glEnd();
187. unsigned **int** textureSide = loadTexture("book\_side.bmp");
188. glBindTexture(GL\_TEXTURE\_2D, textureSide);
189. glBegin(GL\_QUADS);
190. glTexCoord2d(1, 0);
191. glVertex3f(length, 0, -width);
192. glTexCoord2d(1, 1);
193. glVertex3f(length, height, -width);
194. glTexCoord2d(0, 1);
195. glVertex3f(0, height, -width);
196. glTexCoord2d(0, 0);
197. glVertex3f(0, 0, -width);
198. glEnd();
200. unsigned **int** textureTopBottom = loadTexture("book\_top-bottom.bmp");
201. glBindTexture(GL\_TEXTURE\_2D, textureTopBottom);
202. glBegin(GL\_QUADS);
203. glTexCoord2d(0, 0);
204. glVertex3f(length, height, 0);
205. glTexCoord2d(1, 0);
206. glVertex3f(length, height, -width);
207. glTexCoord2d(1, 1);
208. glVertex3f(length, 0, -width);
209. glTexCoord2d(0, 1);
210. glVertex3f(length, 0, 0);
211. glEnd();
213. glBegin(GL\_QUADS);
214. glTexCoord2d(0, 0);
215. glVertex3f(0, 0, 0);
216. glTexCoord2d(1, 0);
217. glVertex3f(0, 0, -width);
218. glTexCoord2d(1, 1);
219. glVertex3f(0, height, -width);
220. glTexCoord2d(0, 1);
221. glVertex3f(0, height, 0);
222. glEnd();
224. glDeleteTextures(1, &textureFront);
225. glDeleteTextures(1, &textureBack);
226. glDeleteTextures(1, &textureSpine);
227. glDeleteTextures(1, &textureSide);
228. glDeleteTextures(1, &textureTopBottom);
229. }
231. **void** renderChessBoard() {
232. glEnable(GL\_TEXTURE\_2D);
234. **float** height = .2,
235. length = 2,
236. width = 2;
238. glTranslatef(6, height, -2);
240. unsigned **int** texture = loadTexture("wood\_edge.bmp");
241. glBindTexture(GL\_TEXTURE\_2D, texture);
242. glBegin(GL\_QUADS);
243. // left
244. glTexCoord2d(0, 1);
245. glVertex3f(0, 0, 0);
246. glTexCoord2d(1, 1);
247. glVertex3f(0, -height, 0);
248. glTexCoord2d(1, 0);
249. glVertex3f(0, -height, -width);
250. glTexCoord2d(0, 0);
251. glVertex3f(0, 0, -width);
253. // front
254. glTexCoord2d(1, 0);
255. glVertex3f(0, 0, 0);
256. glTexCoord2d(1, 1);
257. glVertex3f(length, 0, 0);
258. glTexCoord2d(0, 1);
259. glVertex3f(length, -height, 0);
260. glTexCoord2d(0, 0);
261. glVertex3f(0, -height, 0);
263. // right
264. glTexCoord2d(1, 0);
265. glVertex3f(length, 0, 0);
266. glTexCoord2d(1, 1);
267. glVertex3f(length, 0, -width);
268. glTexCoord2d(0, 1);
269. glVertex3f(length, -height, -width);
270. glTexCoord2d(0, 0);
271. glVertex3f(length, -height, 0);
273. // back
274. glTexCoord2d(1, 1);
275. glVertex3f(length, 0, -width);
276. glTexCoord2d(0, 1);
277. glVertex3f(0, 0, -width);
278. glTexCoord2d(0, 0);
279. glVertex3f(0, -height, -width);
280. glTexCoord2d(1, 0);
281. glVertex3f(length, -height, -width);
282. glEnd();
283. glDeleteTextures(1, &texture);
285. texture = loadTexture("chessboard.bmp");
286. glBindTexture(GL\_TEXTURE\_2D, texture);
287. glBegin(GL\_QUADS);
288. // top
289. glTexCoord2d(0, 0);
290. glVertex3f(0, 0, 0);
291. glTexCoord2d(0, 1);
292. glVertex3f(0, 0, -width);
293. glTexCoord2d(1, 1);
294. glVertex3f(length, 0, -width);
295. glTexCoord2d(1, 0);
296. glVertex3f(length, 0, 0);
297. glEnd();
298. glDeleteTextures(1, &texture);
299. }
301. **void** renderChess(**int** chess, **bool** type) {
302. **float** scale = 0.06;
304. **if** (type) {
305. glRotatef(180, 0, 1, 0);
306. }
308. glScalef(scale, scale, scale);
309. glCallList(chess);
310. }
312. **void** renderPyramid(**int** num) {
313. glDisable(GL\_LIGHTING);
314. glEnable(GL\_BLEND);
315. glDepthMask(GL\_FALSE);
316. glBlendFunc(GL\_DST\_COLOR, GL\_ONE\_MINUS\_SRC\_ALPHA);
318. **float** size = .3;
320. glTranslatef(2 + (num - 1) \* size \* 2, size + .1, -4);
322. glBegin(GL\_TRIANGLES);
323. // front
324. glColor4fv(semicyan);
325. glVertex3f(0, size, 0);
326. glVertex3f(size, -size, size);
327. glVertex3f(-size, -size, size);
329. // left
330. glColor4fv(semiblue);
331. glVertex3f(0, size, 0);
332. glVertex3f(-size, -size, size);
333. glVertex3f(0, -size, -size);
335. // right
336. glColor4fv(semigreen);
337. glVertex3f(0, size, 0);
338. glVertex3f(0, -size, -size);
339. glVertex3f(size, -size, size);
341. // bottom
342. glColor4fv(semired);
343. glVertex3f(-size, -size, size);
344. glVertex3f(size, -size, size);
345. glVertex3f(0, -size, -size);
346. glEnd();
348. glDepthMask(GL\_TRUE);
349. glDisable(GL\_BLEND);
350. glEnable(GL\_LIGHTING);
351. }
353. unsigned **int** loadTexture(**char** \*filename) {
354. unsigned **char** header[54];
355. unsigned **int** position, width, height, size;
356. unsigned **char** \*data;
357. **char** \*filepath = (**char** \*)malloc(255);
359. strcpy(filepath, filename);
360. prepend(filepath, "textures/");
362. **FILE** \*file = fopen(filepath, "rb");
363. free(filepath);
365. fread(header, 1, 54, file);
367. position = \*(**int**\*)&header[10];
368. width = \*(**int**\*)&header[18];
369. height = \*(**int**\*)&header[22];
370. size = \*(**int**\*)&header[34];
372. **if** (size == 0) size = width \* height \* 3;
373. **if** (position == 0) position = 54;
375. data = (unsigned **char** \*)malloc(size);
377. fread(data, 1, size, file);
379. fclose(file);
381. unsigned **int** texture;
383. glGenTextures(1, &texture);
384. glBindTexture(GL\_TEXTURE\_2D, texture);
385. glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGB, width, height, 0, GL\_BGR, GL\_UNSIGNED\_BYTE, data);
386. glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_NEAREST);
387. glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_NEAREST);
389. free(data);
391. **return** texture;
392. }
394. **int** loadObject(**char**\* filename) {
395. std::vector<std::string\*> coord;
396. std::vector<coordinate\*> vertex;
397. std::vector<face\*> faces;
398. std::vector<coordinate\*> normals;
399. **char** \*filepath = (**char** \*)malloc(255);
401. strcpy(filepath, filename);
402. prepend(filepath, "objects/");
404. std::ifstream in(filepath);
406. **if** (!in.is\_open()) {
407. std::cout << "Nor opened" << std::endl;
408. **return** -1;
409. }
411. **char** buf[300];
412. **while** (in.getline(buf, 300)) {
413. coord.push\_back(**new** std::string(buf));
414. }
416. **for** (**int** i = 0; i < coord.size(); i++) {
417. **if** (coord[i]->c\_str()[0] == '#') {
418. **continue**;
419. } **else** **if** (coord[i]->c\_str()[0] == 'v' && coord[i]->c\_str()[1] == ' ') {
420. **float** tmpx, tmpy, tmpz;
421. sscanf(coord[i]->c\_str(), "v %f %f %f", &tmpx, &tmpy, &tmpz);
422. vertex.push\_back(**new** coordinate(tmpx, tmpy, tmpz));
423. } **else** **if** (coord[i]->c\_str()[0] == 'v' && coord[i]->c\_str()[1] == 'n') {
424. **float** tmpx, tmpy, tmpz;
425. sscanf(coord[i]->c\_str(), "vn %f %f %f", &tmpx, &tmpy, &tmpz);
426. normals.push\_back(**new** coordinate(tmpx, tmpy, tmpz));
427. } **else** **if**(coord[i]->c\_str()[0] == 'f') {
428. **int** a, b, c, d, e;
429. **if** (count(coord[i]->begin(), coord[i]->end(), ' ') == 3) {
430. sscanf(coord[i]->c\_str(), "f %d//%d %d//%d %d//%d", &a, &b, &c, &b, &d, &b);
431. faces.push\_back(**new** face(b, a, c, d));
432. } **else** {
433. sscanf(coord[i]->c\_str(), "f %d//%d %d//%d %d//%d %d//%d", &a, &b, &c, &b, &d, &b, &e, &b);
434. faces.push\_back(**new** face(b, a, c, d, e));
435. }
436. }
437. }
439. **int** num = glGenLists(1);
440. glNewList(num, GL\_COMPILE);
441. **for** (**int** i = 0; i < faces.size(); i++) {
442. **if**(faces[i]->four) {
443. glBegin(GL\_QUADS);
444. glNormal3f(normals[faces[i]->facenum-1]->x,
445. normals[faces[i]->facenum-1]->y,
446. normals[faces[i]->facenum-1]->z);
447. glVertex3f(vertex[faces[i]->faces[0]-1]->x,
448. vertex[faces[i]->faces[0]-1]->y,
449. vertex[faces[i]->faces[0]-1]->z);
450. glVertex3f(vertex[faces[i]->faces[1]-1]->x,
451. vertex[faces[i]->faces[1]-1]->y,
452. vertex[faces[i]->faces[1]-1]->z);
453. glVertex3f(vertex[faces[i]->faces[2]-1]->x,
454. vertex[faces[i]->faces[2]-1]->y,
455. vertex[faces[i]->faces[2]-1]->z);
456. glVertex3f(vertex[faces[i]->faces[3]-1]->x,
457. vertex[faces[i]->faces[3]-1]->y,
458. vertex[faces[i]->faces[3]-1]->z);
459. glEnd();
460. } **else** {
461. glBegin(GL\_TRIANGLES);
462. glNormal3f(normals[faces[i]->facenum-1]->x,
463. normals[faces[i]->facenum-1]->y,
464. normals[faces[i]->facenum-1]->z);
465. glVertex3f(vertex[faces[i]->faces[0]-1]->x,
466. vertex[faces[i]->faces[0]-1]->y,
467. vertex[faces[i]->faces[0]-1]->z);
468. glVertex3f(vertex[faces[i]->faces[1]-1]->x,
469. vertex[faces[i]->faces[1]-1]->y,
470. vertex[faces[i]->faces[1]-1]->z);
471. glVertex3f(vertex[faces[i]->faces[2]-1]->x,
472. vertex[faces[i]->faces[2]-1]->y,
473. vertex[faces[i]->faces[2]-1]->z);
474. glEnd();
475. }
476. }
478. glEndList();
480. **for** (**int** i = 0; i < coord.size(); i++) **delete** coord[i];
481. **for** (**int** i = 0; i < faces.size(); i++) **delete** faces[i];
482. **for** (**int** i = 0; i < normals.size(); i++) **delete** normals[i];
483. **for** (**int** i = 0; i < vertex.size(); i++) **delete** vertex[i];
485. **return** num;
486. }
488. **void** prepend(**char**\* s, **const** **char**\* t)
489. {
490. **size\_t** len = strlen(t);
491. **size\_t** i;
493. memmove(s + len, s, strlen(s) + 1);
495. **for** (i = 0; i < len; ++i) {
496. s[i] = t[i];
497. }
498. }

## Скриншоты

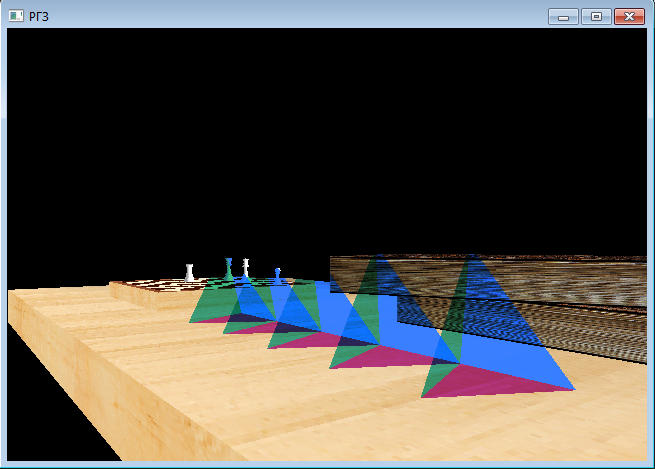


Рисунок 1. Рабочая программа