Белгородский Государственный Технологический Университет им. В. Г. Шухова

Кафедра программного обеспечения вычислительной техники  
и автоматизированных систем

## Лабораторная работа №5 по теме: «Основы пространственной графики в OpenGL»

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

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

Белгород  
2015

**Цель работы:** получение навыков использования примитивов в пространстве и разработка приложения с использование OpenGL.

# Задание

Необходимо вывести на сцене простейшие графические примитивы (не менее 3). Реализовать анимации вращения, переноса, масштабирования. Сделать закраску примитивов. Поэкспериментировать с функциями для вывода сферы, и пр. стандартных объектов. Сделать выбор ортографической и перспективной проекции.

# 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

1. #include <glut.h>
3. **void** display(**void**);
4. **void** reshape(**int** w, **int** h);
5. **void** click(**int** option);
7. #define PERSPECTIVE 0
8. #define ORTHGRAPHIC 1
10. **int** projection = PERSPECTIVE;
12. // colors
13. **float** red[4] = {1, 0, 0, 1};
14. **float** semired[4] = {1, 0, 0, .5};
15. **float** green[4] = {0, 1, 0, 1};
16. **float** semigreen[4] = {0, 1, 0, .5};
17. **float** blue[4] = {0, 0, 1, 1};
18. **float** semiblue[4] = {0, 0, 1, .5};
19. **float** yellow[4] = {1, 1, 0, 1};
20. **float** semiyellow[4] = {1, 1, 0, .5};
21. **float** violet[4] = {1, 0, 1, 1};
22. **float** semiviolet[4] = {1, 0, 1, .5};
23. **float** cyan[4] = {0, 1, 1, 1};
24. **float** semicyan[4] = {0, 1, 1, .5};
26. // rotation angles
27. **float** rPyramid = 0,
28. rQuad = 0,
29. rPyramidSemi = 0;
31. // translate offsets
32. **float** tPyramidSemi = 0;
33. **int** tPyramidSemiMode = 0;
35. // speeds
36. **float** sPyramid = 0.4,
37. sQuad = 0.5,
38. sPyramidSemi = 0.6;
40. **int** main(**int** argc, **char** \*\*argv) {
41. glutInit(&argc, argv);
43. glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_DEPTH);
45. glutInitWindowSize(1000, 500);
46. glutInitWindowPosition(0, 0);
47. glutCreateWindow("Лабораторная работа №5");
48. glClearColor(1, .97, .86, 1);
50. glutDisplayFunc(display);
51. glutIdleFunc(display);
52. glutReshapeFunc(reshape);
54. **int** menu = glutCreateMenu(click);
55. glutAddMenuEntry("Перспективная проекция", PERSPECTIVE);
56. glutAddMenuEntry("Ортографическая проекция", ORTHGRAPHIC);
57. glutAttachMenu(GLUT\_LEFT\_BUTTON);
59. glutMainLoop();
60. }
62. **void** display(**void**) {
63. glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);
64. glEnable(GL\_DEPTH\_TEST);
65. glLoadIdentity();
67. **if** (projection == PERSPECTIVE) {
68. gluLookAt(0, 0, 10, 0, 0, 0, 0, 1, 0);
69. }
71. glPushMatrix();
72. glPushMatrix();
74. glTranslatef(-2.5, 0, 0);
75. glRotatef(rPyramid, 1, 0, 0);
76. glBegin(GL\_TRIANGLES);
77. // front
78. glColor4fv(red);
79. glVertex3f(0, 1, 0);
80. glColor4fv(cyan);
81. glVertex3f(1, -1, 1);
82. glColor4fv(yellow);
83. glVertex3f(-1, -1, 1);
85. // left
86. glColor4fv(red);
87. glVertex3f(0, 1, 0);
88. glColor4fv(yellow);
89. glVertex3f(-1, -1, 1);
90. glColor4fv(violet);
91. glVertex3f(0, -1, -1);
93. // right
94. glColor4fv(red);
95. glVertex3f(0, 1, 0);
96. glColor4fv(violet);
97. glVertex3f(0, -1, -1);
98. glColor4fv(cyan);
99. glVertex3f(1, -1, 1);
101. // bottom
102. glColor4fv(yellow);
103. glVertex3f(-1, -1, 1);
104. glColor4fv(cyan);
105. glVertex3f(1, -1, 1);
106. glColor4fv(violet);
107. glVertex3f(0, -1, -1);
108. glEnd();
110. glPopMatrix();
112. glTranslatef(1.5, 0, 0);
113. glRotatef(rQuad, 1, 1, 0);
114. glBegin(GL\_QUADS);
115. // front
116. glColor4fv(cyan);
117. glVertex3f(-1, -1, 1);
118. glVertex3f(1, -1, 1);
119. glVertex3f(1, 1, 1);
120. glVertex3f(-1, 1, 1);
122. // left
123. glColor4fv(red);
124. glVertex3f(1, -1, 1);
125. glVertex3f(1, -1, -1);
126. glVertex3f(1, 1, -1);
127. glVertex3f(1, 1, 1);
129. // back
130. glColor4fv(blue);
131. glVertex3f(1, -1, -1);
132. glVertex3f(-1, -1, -1);
133. glVertex3f(-1, 1, -1);
134. glVertex3f(1, 1, -1);
136. // right
137. glColor4fv(green);
138. glVertex3f(-1, -1, -1);
139. glVertex3f(-1, -1, 1);
140. glVertex3f(-1, 1, 1);
141. glVertex3f(-1, 1, -1);
143. // top
144. glColor4fv(violet);
145. glVertex3f(-1, -1, -1);
146. glVertex3f(1, -1, -1);
147. glVertex3f(1, -1, 1);
148. glVertex3f(-1, -1, 1);
150. // bottom
151. glColor4fv(yellow);
152. glVertex3f(-1, 1, 1);
153. glVertex3f(1, 1, 1);
154. glVertex3f(1, 1, -1);
155. glVertex3f(-1, 1, -1);
156. glEnd();
158. glPopMatrix();
160. glEnable(GL\_BLEND);
161. glDepthMask(GL\_FALSE);
162. glBlendFunc(GL\_DST\_COLOR, GL\_ONE\_MINUS\_SRC\_ALPHA);
164. glRotatef(rPyramidSemi, 0, 1, 0);
165. glTranslatef(0, 0, 5);
166. glRotatef(rPyramidSemi, 1, 1, 0);
167. glBegin(GL\_TRIANGLES);
168. // left
169. glColor4fv(semiblue);
170. glVertex3f(0, 1, 0);
171. glVertex3f(-1, -1, 1);
172. glVertex3f(0, -1, -1);
174. // right
175. glColor4fv(semigreen);
176. glVertex3f(0, 1, 0);
177. glVertex3f(0, -1, -1);
178. glVertex3f(1, -1, 1);
180. // bottom
181. glColor4fv(semired);
182. glVertex3f(-1, -1, 1);
183. glVertex3f(1, -1, 1);
184. glVertex3f(0, -1, -1);
186. // front
187. glColor4fv(semiyellow);
188. glVertex3f(0, 1, 0);
189. glVertex3f(1, -1, 1);
190. glVertex3f(-1, -1, 1);
191. glEnd();
193. glDepthMask(GL\_TRUE);
194. glDisable(GL\_BLEND);
196. rPyramidSemi += sPyramidSemi;
197. rPyramid += sPyramid;
198. rQuad += sQuad;
200. tPyramidSemi += (tPyramidSemiMode > 0) ? .01 : -.01;
202. **if** (tPyramidSemi > .5) {
203. tPyramidSemiMode = 0;
204. } **else** **if** (tPyramidSemi < -.5) {
205. tPyramidSemiMode = 1;
206. }
208. glutSwapBuffers();
209. }
211. **void** reshape(**int** w, **int** h) {
212. **if** (h == 0) h = 1;
214. **float** aspectRatio = (**float**)w / h;
216. glViewport(0, 0, w, h);
218. glMatrixMode(GL\_PROJECTION);
219. glLoadIdentity();
221. **if** (projection == PERSPECTIVE) {
222. gluPerspective(50, aspectRatio, .1, 100);
223. } **else** {
224. **float** scale = 5;
226. **if** (aspectRatio >= 1) {
227. glOrtho(-scale \* aspectRatio, scale \* aspectRatio, -scale, scale, -10, 10);
228. } **else** {
229. glOrtho(-scale, scale, -scale / aspectRatio, scale / aspectRatio, -10, 10);
230. }
231. }
233. glMatrixMode(GL\_MODELVIEW);
234. glLoadIdentity();
235. }
237. **void** click(**int** option) {
238. projection = option;
239. reshape(glutGet(GLUT\_WINDOW\_WIDTH), glutGet(GLUT\_WINDOW\_HEIGHT));
240. }

## Скриншоты

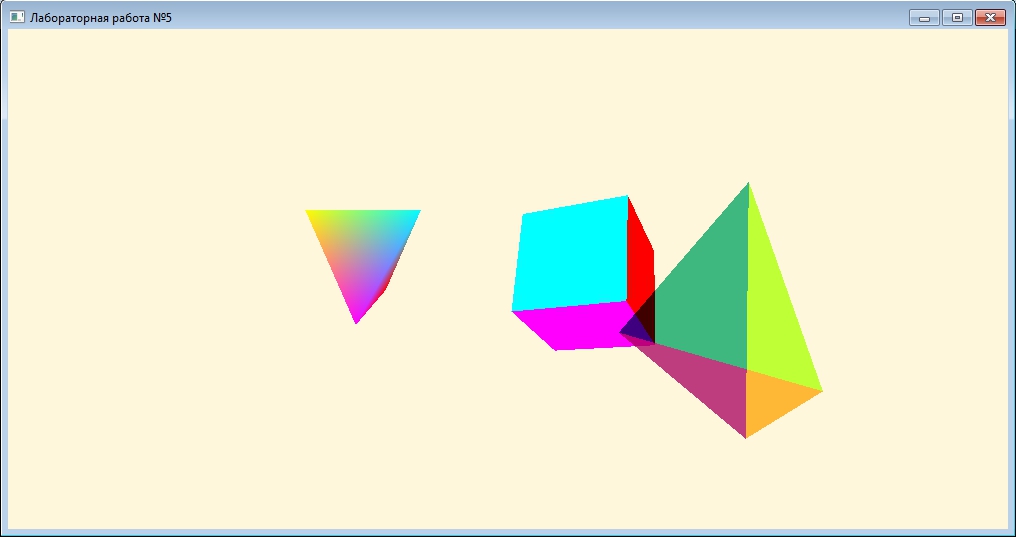


Рисунок 1. Рабочая программа