МИНОБРНАУКИ РОССИИ

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ

ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

«ЛЭТИ» ИМ. В. И. УЛЬЯНОВА (ЛЕНИНА)

Кафедра САПР

ОТЧЕТ

по лабораторной работе №5

по дисциплине «Компьютерная графика»

Тема: Исследование алгоритмов выявления видимости сложных сцен

Студенты гр. 5371 Бергер Э. Э.

Локкина О. С.

Уруков С. Д.

Преподаватель Матвеева И. В.

Санкт-Петербург

2017

**ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ**

*Вариант 5.1:*

Обеспечить реализацию алгоритма выявления видимых граней и ребер для одиночного выпуклого объемного тела. Добавлена возможность перемещения точки наблюдения относительно осей X, Y и Z.

**ХОД РАБОТЫ**

Использовалась библиотека OpenGL с языком программирования Python 3.6.x.

Видимость грани определяется пересечением ее плоскости отрезка, концами которого являются точка наблюдения и одна внутренняя точка фигуры.

Внутренняя точка берется как среднее арифметическое всех вершин по каждой координате.

Точка наблюдения первоначально задается в координате (6, 6, 6), далее изменяется с помощью клавиш q, w, e, a, s, d.

Для определения факта пересечения используется каноническая форма плоскости и параметрическая запись прямой.

Из уравнения прямой выводится уравнение вида где коэффициенты , , ,

Далее по известным нам координатам точки наблюдения и внутренней точки, а также коэффициентам a, b, c и d вычисляется значение t.

Чтобы плоскость пересекала отрезок t должен лежать в интервале от 0 до 1.

**ПРИМЕР ЗАПУСКА ПРОГРАММЫ**

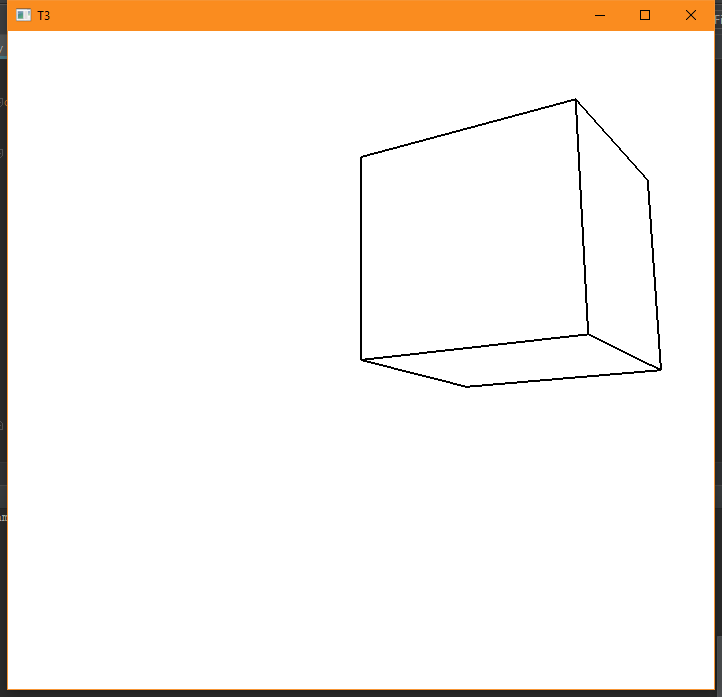
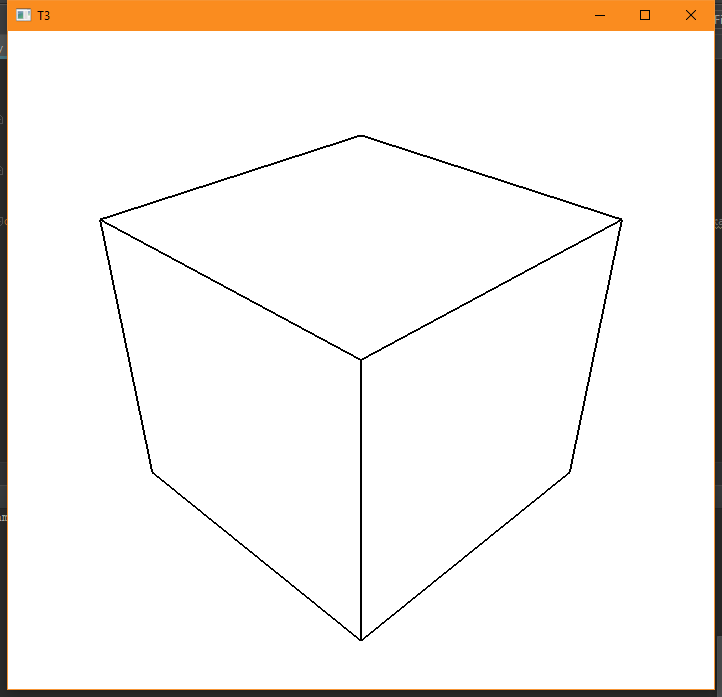
При запуске программы открывается окно. В окне отображается куб, у которого скрыты невидимые грани.

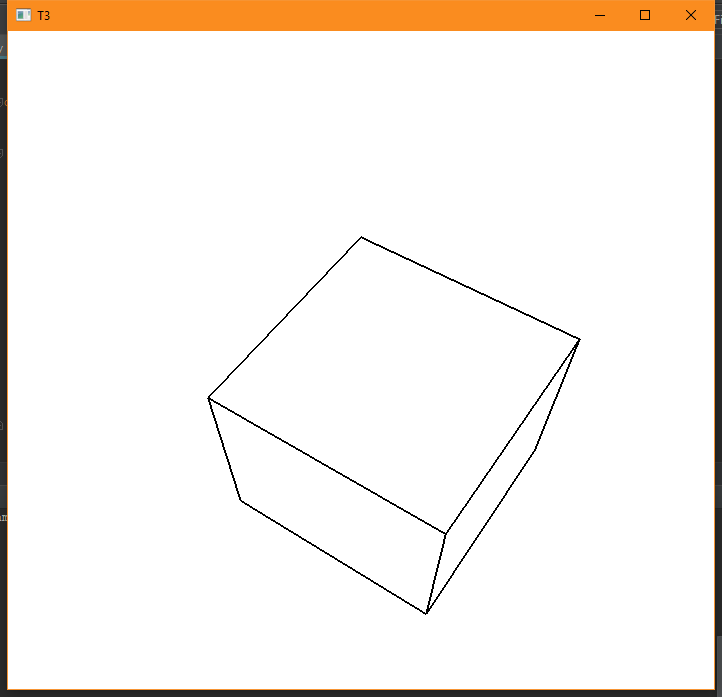
В оконе можно перемещать точку наблюдения относительно осей X, Y и Z с помощью следующих клавиш:

q-e – перемещение относительно Z

a-d – перемещение относительно X

w-s – перемещение относительно Y





**ВЫВОД**

В ходе выполнения лабораторной работы были получены навыки отрисовки объемных объектов с использованием алгоритма выявления видимости граней и ребер одиночного выпуклого тела.

**КОД ПРОГРАММЫ**

1. **from** OpenGL.GL **import** \*
2. **from** OpenGL.GLU **import** \*
3. **from** OpenGL.GLUT **import** \*
5. window = 0
7. faces = (((0, 0, 0), (0, 3, 0), (3, 3, 0), (3, 0, 0)),
8. ((0, 0, 0), (0, 3, 0), (0, 3, 3), (0, 0, 3)),
9. ((3, 0, 3), (3, 3, 3), (3, 3, 0), (3, 0, 0)),
10. ((0, 3, 3), (0, 3, 0), (3, 3, 0), (3, 3, 3)),
11. ((0, 0, 0), (3, 0, 0), (3, 0, 3), (0, 0, 3)),
12. ((0, 0, 3), (3, 0, 3), (3, 3, 3), (0, 3, 3)))
14. eye = (6, 6, 6)
16. point = [0, 0, 0]

19. **def** keyPressed(bkey, x, y):
20. **global** eye
21. **try**:
22. key = bkey.decode("utf-8")
23. **if** key == chr(27):
24. sys.exit()
25. **if** key == 'w':
26. eye = (eye[0], eye[1] + 1, eye[2])
27. **if** key == 's':
28. eye = (eye[0], eye[1] - 1, eye[2])
29. **if** key == 'd':
30. eye = (eye[0] + 1, eye[1], eye[2])
31. **if** key == 'a':
32. eye = (eye[0] - 1, eye[1], eye[2])
33. **if** key == 'q':
34. eye = (eye[0], eye[1], eye[2] - 1)
35. **if** key == 'e':
36. eye = (eye[0], eye[1], eye[2] + 1)
37. **except**:
38. **pass**
39. glutPostRedisplay()

42. **def** getCanonicalForm(pointOne, pointTwo, pointThree):  *# Просто считаю определитель для опеределения канонической формы плоскости*
43. x1 = pointOne[0]
44. y1 = pointOne[1]
45. z1 = pointOne[2]
47. x2 = pointTwo[0]
48. y2 = pointTwo[1]
49. z2 = pointTwo[2]
51. x3 = pointThree[0]
52. y3 = pointThree[1]
53. z3 = pointThree[2]
55. result = [0, 0, 0, 0]  *# ax+by+cz+d=0*
56. result[0] = (y2-y1)\*(z3-z1)-(z2-z1)\*(y3-y1)
57. result[1] = -((x2-x1)\*(z3-z1)-(z2-z1)\*(x3-x1))
58. result[2] = (x2-x1)\*(y3-y1)-(y2-y1)\*(x3-x1)
59. result[3] = -x1\*result[0] - y1\*result[1] - z1\*result[2]
61. **return** result

64. **def** getPointInside():
65. **global** point
66. count = 0
67. **for** face **in** faces:
68. count += len(face)
69. temp = list(zip(\*face))
70. **for** i **in** range(3):
71. point[i] += sum(temp[i])
72. point = [element/count **for** element **in** point]

75. **def** getT(plane, pointOne, pointTwo):
77. a = plane[0]
78. b = plane[1]
79. c = plane[2]
80. d = plane[3]
82. x1 = pointOne[0]
83. y1 = pointOne[1]
84. z1 = pointOne[2]
86. x2 = pointTwo[0]
87. y2 = pointTwo[1]
88. z2 = pointTwo[2]
90. t = (-d-a\*x1-b\*y1-c\*z1)/(a\*(x2-x1)+b\*(y2-y1)+c\*(z2-z1))
92. **return** t

95. **def** initFigure():
96. **global** faces, point, eye
97. glLineWidth(2)
98. **for** face **in** faces:
99. t = getT(getCanonicalForm(face[0], face[1], face[2]), point, eye)
100. **if** 0 <= t <= 1:
101. glBegin(GL\_LINE\_LOOP)
102. glColor3f(0, 0, 0)
103. **for** vertex **in** face:
104. glVertex3dv(vertex)
105. glEnd()

108. **def** initGL(Width, Height):
109. glClearColor(1.0, 1.0, 1.0, 1.0)
110. glClearDepth(1.0)
111. glDepthFunc(GL\_LESS)
112. glEnable(GL\_DEPTH\_TEST)
113. glShadeModel(GL\_SMOOTH)
114. glMatrixMode(GL\_PROJECTION)
115. glLoadIdentity()
116. gluPerspective(45.0, float(Width) / float(Height), 0.1, 100.0)
117. glMatrixMode(GL\_MODELVIEW)

120. **def** draw3DScene():
121. glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT)
123. glLoadIdentity()
125. gluLookAt(eye[0], eye[1], eye[2], 0, 0, 0, 0, 1, 0)
127. initFigure()
129. glutSwapBuffers()

132. **def** initWindow():
133. **global** window
135. glutInit(sys.argv)
136. glutInitDisplayMode(GLUT\_RGBA | GLUT\_DOUBLE | GLUT\_DEPTH)
137. glutInitWindowSize(800, 800)
138. glutInitWindowPosition(500, 100)
140. window = glutCreateWindow(b"Task5")
142. glutDisplayFunc(draw3DScene)
143. glutKeyboardFunc(keyPressed)
144. initGL(800, 800)

147. **def** main():
148. getPointInside()
149. initWindow()
150. glutMainLoop()

153. main()