Лабораторная работа 1. Вариант 1

Калабин Павел Павлович 5130904/20103

15 сентября 2025 г.

1 Постановка задачи

Целью работы является ознакомление с основами векторной графики и получение навыков работы с базовыми функциями графического API и трехмерными графическими примитивами. Требуется при помощи стандартных функций бибилиотеки (OpenGL/Vulkan или DirectX) изобразить указанные объекты и произвести необходимые преобразования.

1. Изобразить каркасный конус и каркасную сферу, расположенные на некотором расстоянии друг от друга.
2. Совместить центр основания конуса и центр сферы.
3. Изобразить тор и цилиндр. Размеры и местоположение примитивов задать самостоятельно.
4. Выполнить последовательно сначала поворот цилиндра вокруг оси X, а затем растяжение тора в 2 раза.

2 Ход работы

В качестве среды выполнения работы была выбрана библиотека OpenGL. Для выполнения работы были использованы примитивы из библиотеки OpenGL Utility (GLU) и OpenGL Utility Toolkit (GLUT).

2.1 Настройка OpenGL

Для корректной работы и отрисовки примитивов необходимо настроить OpenGL. Для этого при помощи стека матриц были созданы *матрица проекции (projection matrix)* и *видовая матрица (view matrix)*. Кратко работу со стеком матриц можно описать следующим образом:

1. Загружается матрица, с которой предполагается производить операции,

2. При помощи функций библиотеки эта матрица умножается справа на изменяющую матрицу,

3. На стеке остаётся преобразованная матрица с необходимыми нам характеристиками.

Подробнее работа со стеком матриц будет рассмотрена на примере построения сцены.

Матрица проекции отвечает за проекцию трёхмерного пространства на двумерное пространство экрана и за отсечение тех объектов, которые не находятся в поле зрения.

Существует несколько видов матриц проекции, например *ортографическая проекция*, такая проекция переносит объекты «как есть» без учета перспективы. Второй вид проекции это *проекция с перспективой*, она позволяет отобразить объекты с учетом их положения в пространстве так, как они бы выглядели при взгляде на них с позиции камер.

В работе была использована матрица проекции с перспективой из библиотеки GLU, которая имеет дополнительный параметр угла обзора (FOV).

Видовая матрица отвечает за преобразование мировых координат в пространство координат камеры, эта матрица как бы перемещает точку наблюдения в центр камеры.

В качестве такой матрицы была использована матрица LookAt, которую предоставляет функция `gluLookAt`.

2.2 Сцены

Была составлена сцена из синей сферы и фиолетового конуса (см. рис. 1), для этого использованы функции `glutWireCone` и `glutWireSphere`. Эти функции используют внутреннюю реализацию из библиотеки GLU, в частности реализация конуса представляет собой вызов функции отрисовки циллиндра с нулевым параметром радиуса верхней части, так что верхний радиус циллиндра вырождается в точку.

Для создания сцены необходимо использовать стек матриц, для представления каждого примитива в виде набора вершин и манипуляции этими объектами. При создании сцены на стек матриц заносится матрица, отвечающая за трансформации сцены в целом (например, вращение целой сцены). Далее по очередно заносятся матрицы, отвечающие за трансформации над объектами и описываются соотвествующие им примитивы.

2.3 Анимации

Для анимирования сцен введены параметры, например смещение сферы на некоторую позицию. Параметры сцен изменяются с течением времени в функциях `animateX`, которые определяют анимации для соответсвующей сцены.

2.4 Демонстрация работы программы

Далее на рисунках с 1 по 4 показана работа программы и вид сцены №1. На рисунках с 5 по 8 демонстрируется сцена №2.

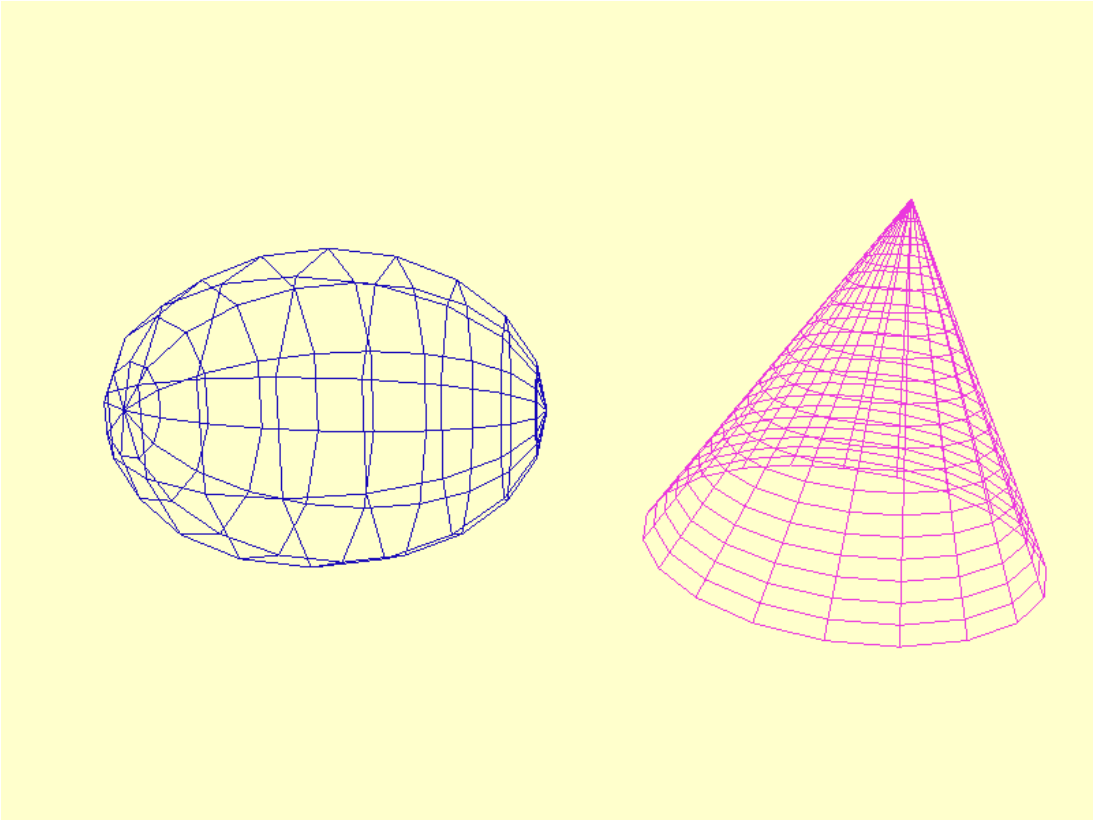

Рис. 1: Сцена 1. Начальное состояние

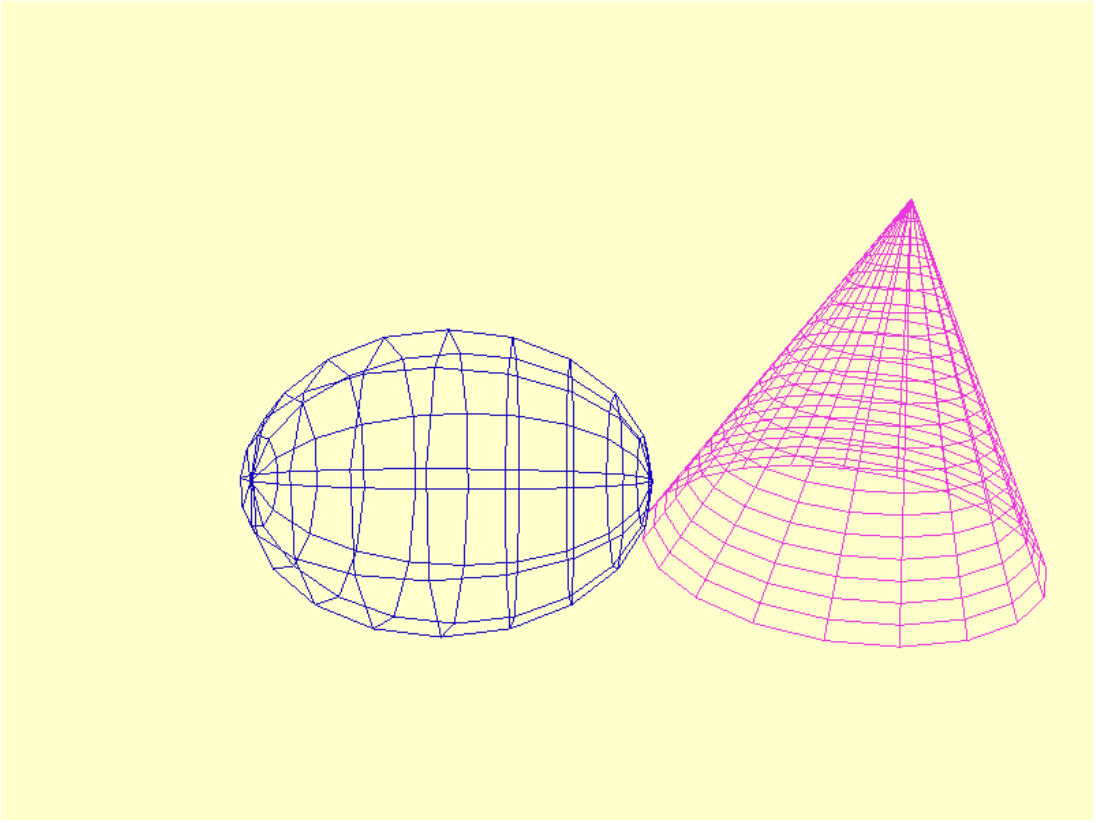

Рис. 2: Сцена 1. Перенос сферы

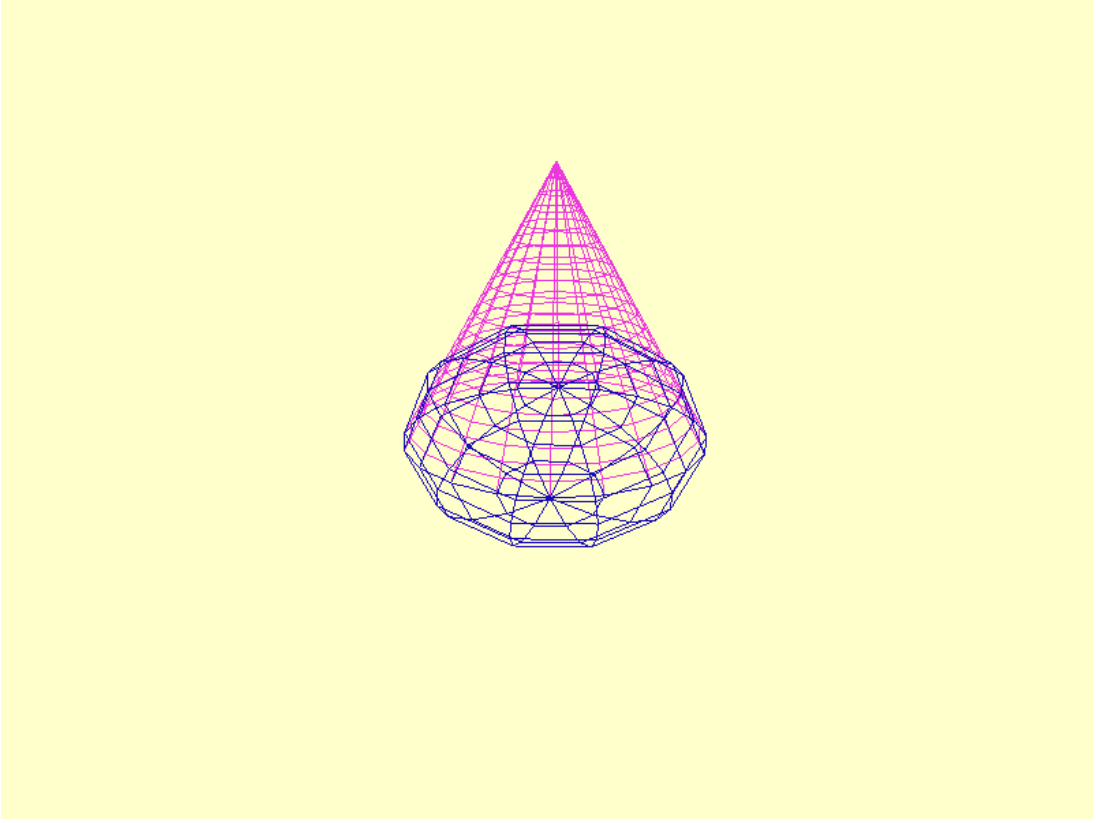

Рис. 3: Сцена 1. Перенос сферы

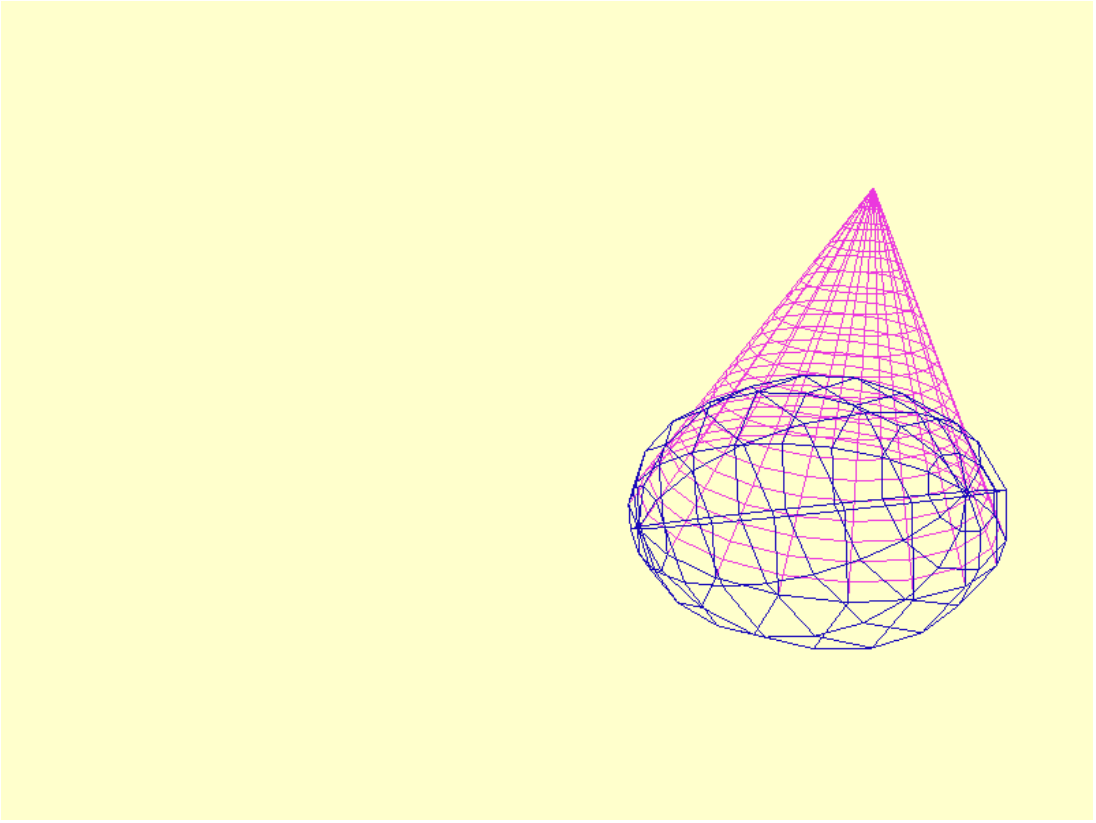

Рис. 4: Сцена 1. Перенос сферы

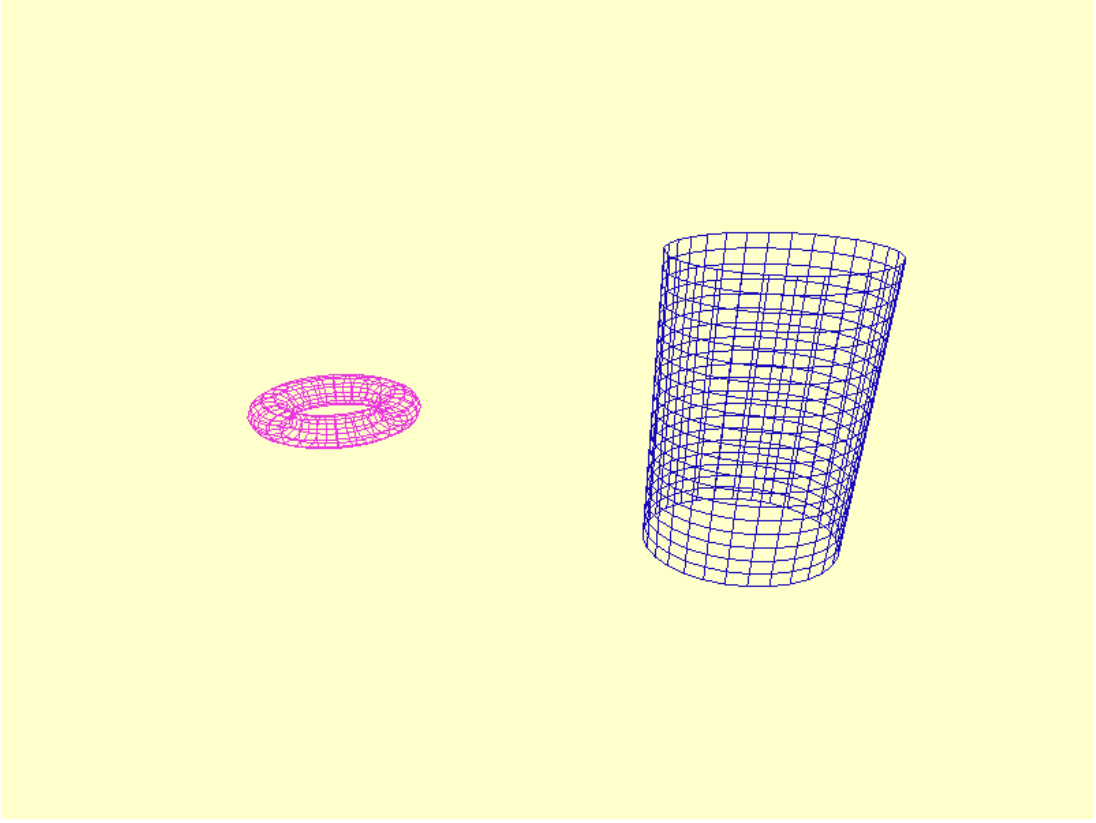

Рис. 5: Сцена 2. Начальное состояние

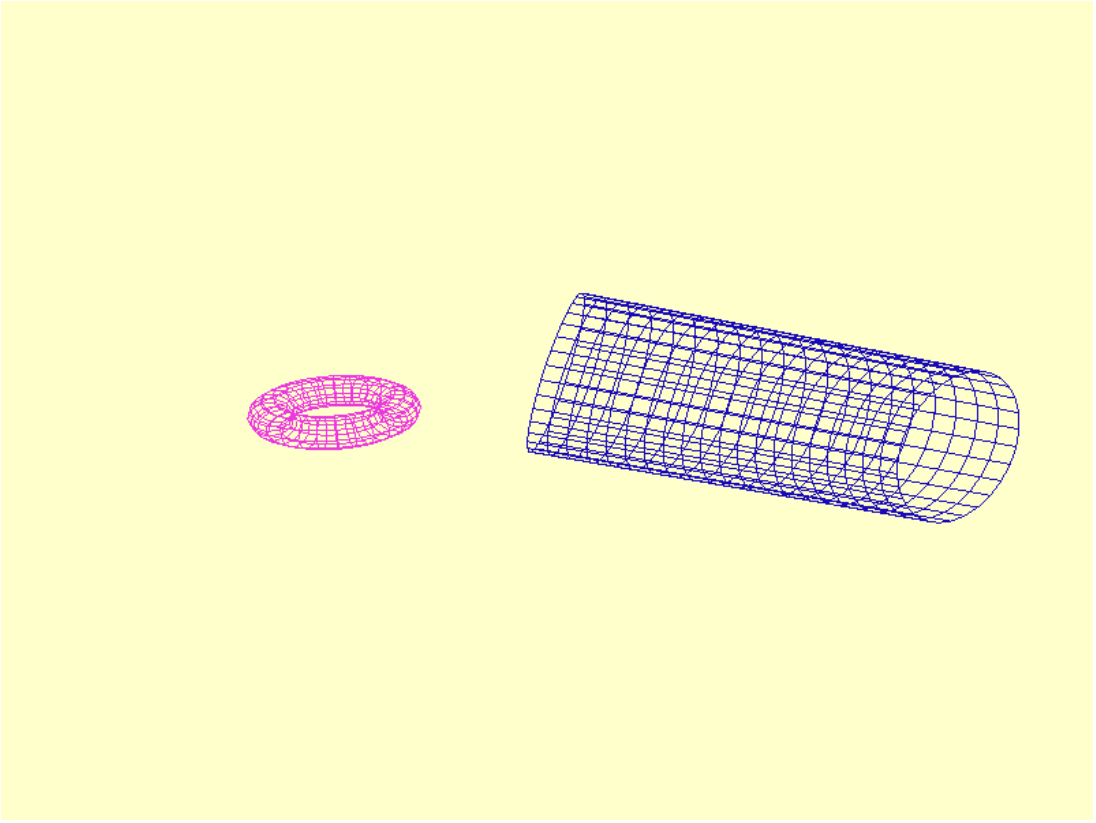

Рис. 6: Сцена 2. Вращение циллиндра

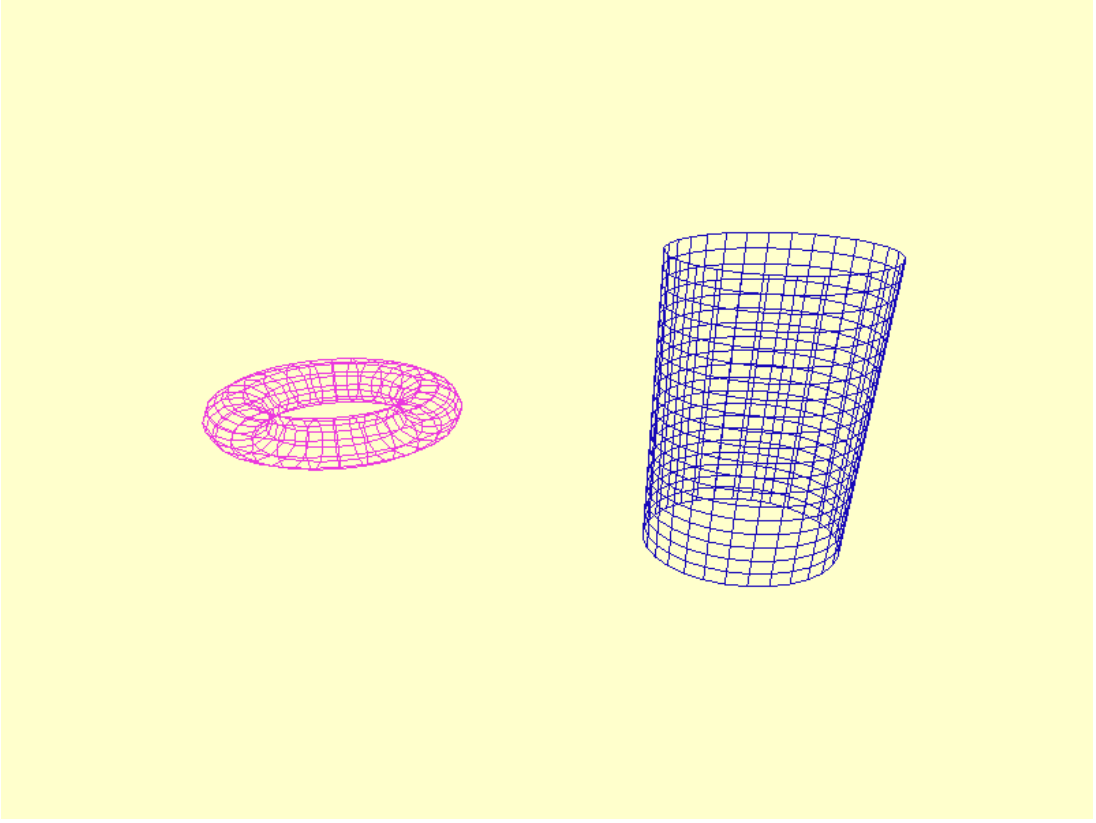

Рис. 7: Сцена 2. Начало увеличения тора в 2 раза

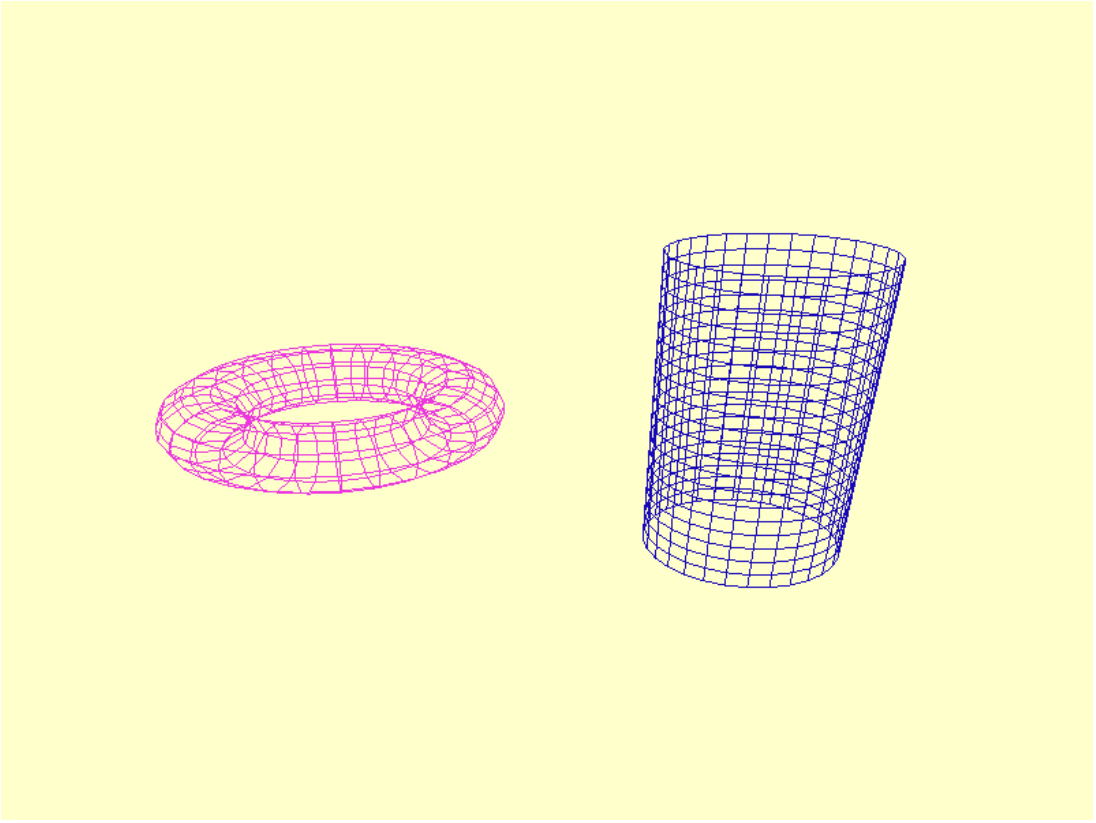

Рис. 8: Сцена 2. Тор увеличен в 2 раза

Приложение

Листинг 1: Исходный текст программы

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <time.h>
4
5  #include <GL/gl.h>
6  #include <GL/glut.h>
7
8  #define RGB(r, g, b) r/255.0, g/255.0, b/255.0
9
10 void animate1();
11 void animate2();
12 void scene1();
13 void scene2();
14 void display();
15 void reshape(int width, int height);
16 void keyboard_handler(unsigned char key, int x, int y);
17 void usage();
18 void screendump(const char *filename, short width, short height);
```

```
19
20 enum { scene_1, scene_2 };
21 int scene_number;
22
23 double x, y, z;
24 int angle1;
25 bool rotation;
26
27 double scale = 1;
28 double angle2;
29 int sub_angle_stage_1;
30
31 int frame = 0;
32 int scene_numher;
33
34 int scene1AnimationDuration = 500;
35 int scene2AnimationDuration = 200;
36
37 int main (int argc, char *argv[])
38 {
39     usage();
40     /* initialize GLUT, using any commandline parameters passed to the
41        program */
42     glutInit(&argc,argv);
43
44     /* setup the size, position, and display mode for new windows */
45     glutInitWindowSize(800,600);
46     glutInitWindowPosition(0,0);
47     glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE);
48
49     /* create and set up a window */
50     glutCreateWindow("LearnOpenGL");
51     glutDisplayFunc(display);
52     glutKeyboardFunc(keyboard_handler);
53     glutReshapeFunc(reshape);
54
55     glEnable(GL_DEPTH_TEST);
56
57     glMatrixMode(GL_PROJECTION);
58     glLoadIdentity();
59     gluPerspective(60, 1, 1, 10);
60
```

```
61      glMatrixMode(GL_MODELVIEW);
62      glLoadIdentity();
63      gluLookAt(4, 2, 0, 0, 0, 0, 0, 1, 0);
64
65      x = 0;
66      y = 0;
67      z = 1;
68      rotation = false;
69
70      /* tell GLUT to wait for events */
71      glutMainLoop();
72  }
73
74  void usage()
75  {
76      printf("Usage:\n");
77      printf(" q, Q, ESC -- Exit\n");
78      printf(" 1 -- switch to scene #1\n");
79      printf(" 2 -- switch to scene #2\n");
80      printf(" r, R -- toggle rotation\n");
81      printf(" s, S -- start animating current scene\n");
82  }
83
84  void reshape(int width, int height)
85  {
86      glViewport(0, 0, width, height);
87  }
88
89  void scene1()
90  {
91      glPushMatrix();
92
93      glRotatef(angle1, 0, 1, 0);
94
95      glPushMatrix();
96      glRotatef(-90, 1, 0, 0);
97      glTranslatef(0, 1.5, -1);
98      glColor3f(0.92, 0.22, 0.87);
99      glutWireCone(1, 2.3, 20, 20);
100     glPopMatrix();
101
102
```

```
103         glPushMatrix();
104         glTranslatef(0, y, z);
105         glColor3f(RGB(23, 8, 184));
106         glutWireSphere(1, 10, 10);
107         glPopMatrix();
108 
109         glPopMatrix();
110 }
111 
112 void scene2()
113 {
114         glPushMatrix();
115 
116         glRotatef(angle2, 0, 1, 0);
117 
118         glPushMatrix();
119         glRotatef(−90, 1, 0, 0);
120         glColor3f(0.92, 0.22, 0.87);
121         glTranslatef(0, −1, 0);
122         glScalef(0.5, 0.5, 0.5);
123         glScalef(scale, scale, scale);
124         glutWireTorus(0.2, 0.6, 10, 30);
125         glPopMatrix();
126 
127         glPushMatrix();
128 
129         glRotatef(−90, 1, 0, 0);
130         glColor3f(RGB(23, 8, 184));
131         glTranslatef(0, 1, 0);
132         glRotatef(sub_angle_stage_1, 1, 0, 0);
133         glTranslatef(0, 0, −1);
134 
135 
136         GLUquadricObj ∗quad = gluNewQuadric();
137         gluQuadricDrawStyle(quad, GLU_LINE);
138 
139         gluCylinder(quad, 0.5, 0.5, 2, 30, 20);
140         glPopMatrix();
141 
142         glPopMatrix();
143 }
144 
```

```
145 void animate1()
146 {
147     frame++;
148     x += .005;
149     y -= .005;
150     z -= .005;
151
152     if (rotation) {
153         angle1 += 1;
154         angle1 = angle1 > 360 ? 0 : angle1;
155     }
156
157     y = y < -1 ? -1 : y;
158     z = z < -1.5 ? -1.5 : z;
159
160     if (z == 2.5) {
161         glutIdleFunc(NULL);
162     }
163
164     glutPostRedisplay();
165 }
166
167 void animate2()
168 {
169     static bool stage1_complete = false;
170
171     if (rotation) {
172         angle2 += 1;
173         angle2 = angle2 > 360 ? 0 : angle2;
174     }
175     sub_angle_stage_1 += 1;
176     sub_angle_stage_1 = sub_angle_stage_1 > 180 ? 180 :
        ↪ sub_angle_stage_1;
177
178     if (sub_angle_stage_1 >= 180)
179         stage1_complete = true;
180
181     if (stage1_complete)
182     {
183         scale += 0.005;
184         scale = scale > 2.0 ? 2.0 : scale;
185     }
```

```
186     glutPostRedisplay();
187 }
188
189 void display ()
190 {
191     /* clear window */
192     glClearColor(1.0f, 1.0f, 0.8f, 1.0f);
193     glClear(GL_COLOR_BUFFER_BIT |
            ↪ GL_DEPTH_BUFFER_BIT);
194
195     /* future matrix manipulations should affect the modelview matrix */
196     glMatrixMode(GL_MODELVIEW);
197
198     if (scene_number == scene_1) {
199         scene1();
200     }
201     else if (scene_number == scene_2) {
202         scene2();
203     }
204
205     glFlush();
206
207     glutSwapBuffers();
208 }
209
210 void keyboard_handler(unsigned char key, int x, int y)
211 {
212     printf("%c %d %d\n", key, x, y);
213     if (key == 'q' || key == 'Q') {
214         glutDestroyWindow(1);
215     }
216     else if (key == 0x1b) {
217         glutDestroyWindow(1);
218     }
219     else if (key == '1') {
220         scene_number = scene_1;
221         glutPostRedisplay();
222     }
223     else if (key == '2') {
224         scene_number = scene_2;
225         glutPostRedisplay();
226     }
```

```
227     else if (key == 'R' || key == 'r') {
228         rotation = !rotation;
229     }
230     else if (key == 's' || key == 'S') {
231         if (scene_number == scene_1) {
232             glutIdleFunc(animate1);
233         } else if (scene_number == scene_2) {
234             glutIdleFunc(animate2);
235         }
236     }
237     else if (key == 'p' || key == 'P') {
238         char filename[512] = { 0 };
239         sprintf(filename, "%s_%ld.tga", "screenshot", time(NULL));
240         screendump(filename, 800, 600);
241     }
242 }
243 
244 /*
245  * Code by Paul Bourke
246  * From: http://www.paulbourke.net/dataformats/tga/
247  */
248 void write_tga_header(FILE *fp, short width, short height)
249 {
250     putc(0,fp);
251     putc(0,fp);
252     putc(2,fp); /* uncompressed RGB */
253     putc(0,fp); putc(0,fp);
254     putc(0,fp); putc(0,fp);
255     putc(0,fp);
256     putc(0,fp); putc(0,fp); /* X origin */
257     putc(0,fp); putc(0,fp); /* y origin */
258     putc((width & 0x00FF),fp);
259     putc((width & 0xFF00) / 256,fp);
260     putc((height & 0x00FF),fp);
261     putc((height & 0xFF00) / 256,fp);
262     putc(24,fp); /* 24 bit bitmap */
263     putc(0,fp);
264 }
265 
266 void screendump(const char *filename, short width, short height)
267 {
268     unsigned int size = width * height * 3;
```

```
269     unsigned char *pixels;
270     FILE *fp = fopen(filename, "w");
271     if (!fp) {
272         perror("fopen");
273         return;
274     }
275
276     write_tga_header(fp, width, height);
277
278     pixels = new unsigned char[size];
279     glReadPixels(0, 0, width, height, GL_BGR,
            ↪ GL_UNSIGNED_BYTE, pixels);
280
281     fwrite(pixels, size, 1, fp);
282     fclose(fp);
283
284     delete[] pixels;
285 }
```