Лабораторная работа 1. Вариант 1

Калабин Павел Павлович 5130904/20103

2025-09-15

# 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 Х, а затем растяжение тора в 2 раза.

# 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).

## 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.

## Сцены

Была составлена сцена из синей сферы и фиолетового конуса (см. рис. [1](#scene11_figure)), 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.

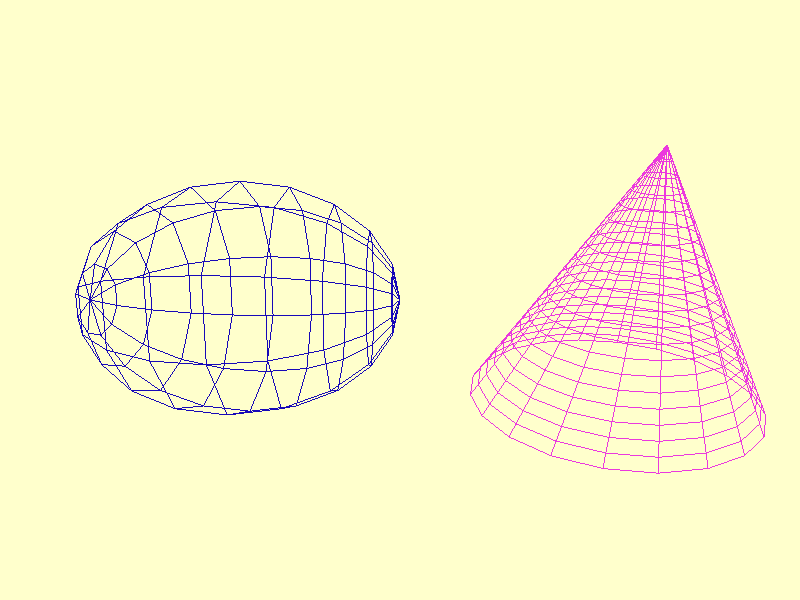
Для создания сцены необходимо использовать стек матриц, для представления каждого примитива в виде набора вершин и манипуляции этими объектами. При создании сцены на стек матриц заносится матрица, отвечающая за трансформации сцены в целом (например, вращение целой сцены). Далее по очередно заносятся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

## Анимации

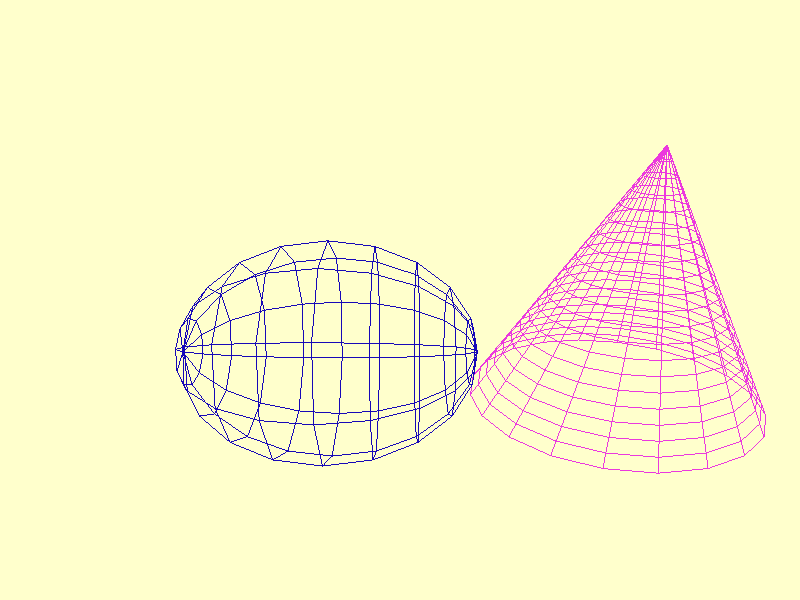
Для анимирования сцен введены параметры, например смещение сферы на некоторую позицию. Параметры сцен изменяются с течением времени в функциях animateX, которые определяют анимации для соответсвующей сцены.

## Демонстрация работы программы

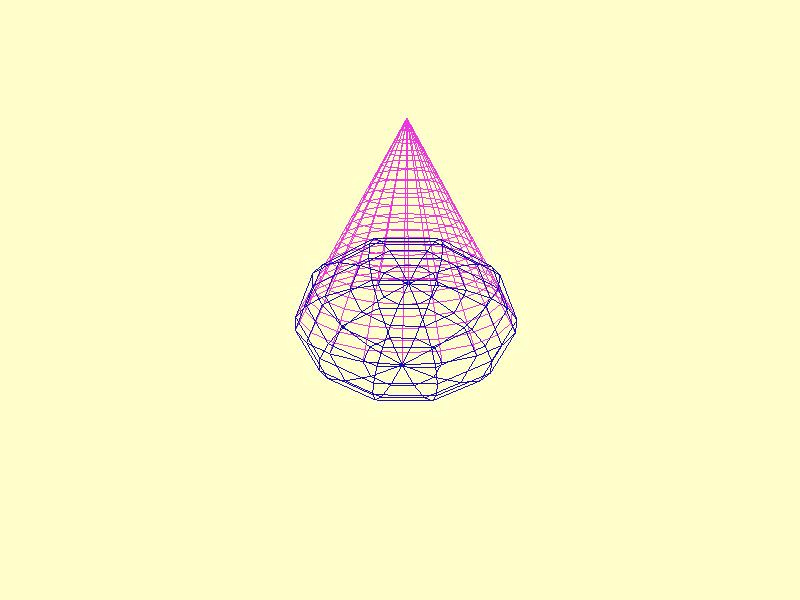
Далее на рисунках с  [1](#scene11_figure) по  [4](#scene14_figure) показана работа программы и вид сцены №1. На рисунках с  [5](#scene21_figure) по  [8](#scene24_figure) демонстрируется сцена №2.



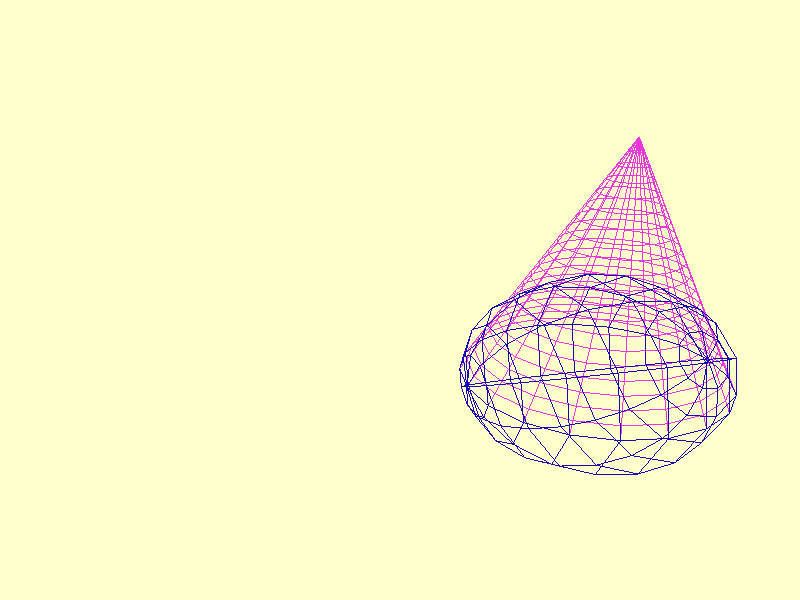
Сцена 1. Начальное состояние



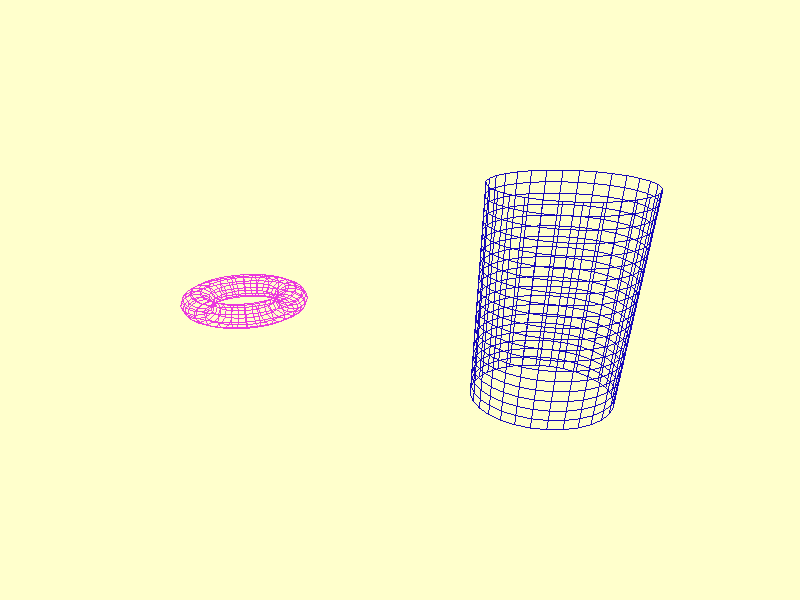
Сцена 1. Перенос сферы



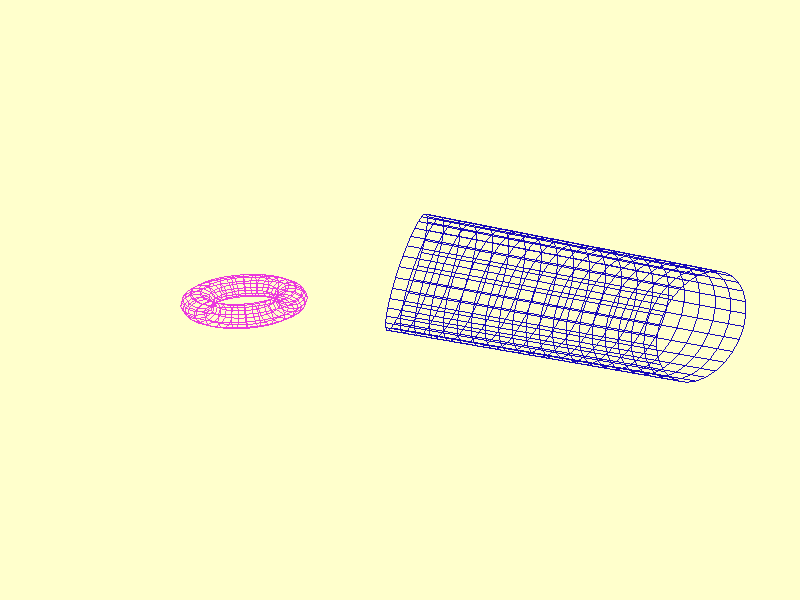
Сцена 1. Перенос сферы



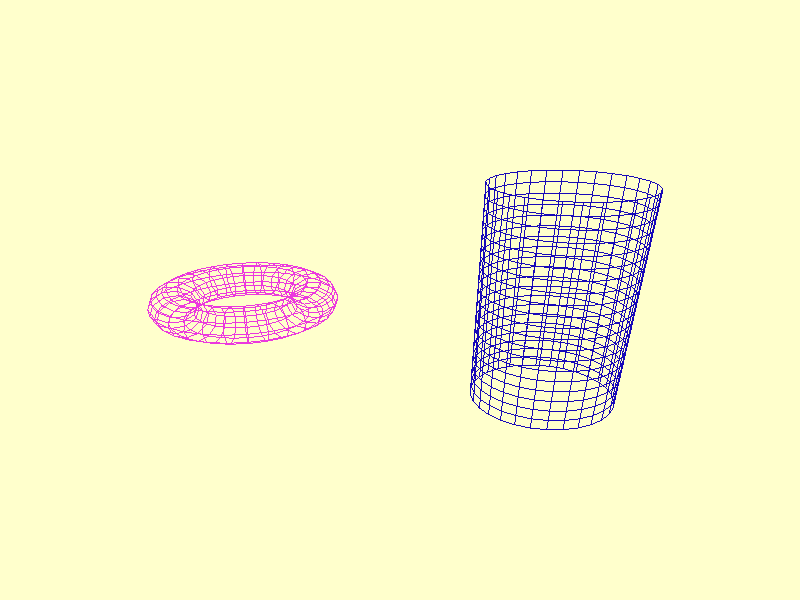
Сцена 1. Перенос сферы



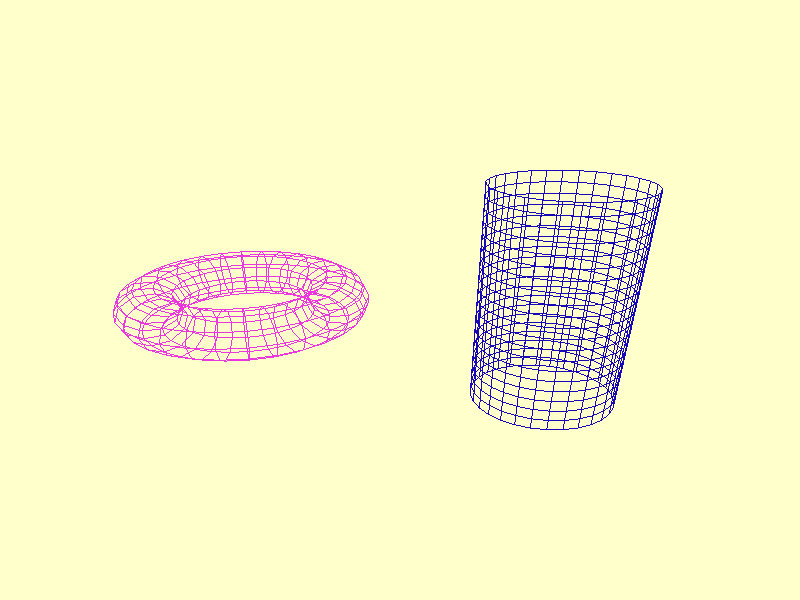
Сцена 2. Начальное состояние



Сцена 2. Вращение циллиндра



Сцена 2. Начало увеличения тора в 2 раза



Сцена 2. Тор увеличен в 2 раза

# Приложение

#include <stdio.h>  
#include <stdlib.h>  
#include <time.h>  
  
#include <GL/gl.h>  
#include <GL/glut.h>  
  
#define RGB(r, g, b) r/255.0, g/255.0, b/255.0  
  
void animate1();  
void animate2();  
void scene1();  
void scene2();  
void display();  
void reshape(int width, int height);  
void keyboard\_handler(unsigned char key, int x, int y);  
void usage();  
void screendump(const char \*filename, short width, short height);  
  
enum { scene\_1, scene\_2 };  
int scene\_number;  
  
double x, y, z;  
int angle1;  
bool rotation;  
  
double scale = 1;   
double angle2;  
int sub\_angle\_stage\_1;  
  
int frame = 0;  
int scene\_numher;  
  
int scene1AnimationDuration = 500;  
int scene2AnimationDuration = 200;  
  
int main (int argc, char \*argv[])   
{  
 usage();  
 /\* initialize GLUT, using any commandline parameters passed to the   
 program \*/  
 glutInit(&argc,argv);  
  
 /\* setup the size, position, and display mode for new windows \*/  
 glutInitWindowSize(800,600);  
 glutInitWindowPosition(0,0);  
 glutInitDisplayMode(GLUT\_RGB | GLUT\_DOUBLE);  
  
 /\* create and set up a window \*/  
 glutCreateWindow("LearnOpenGL");  
 glutDisplayFunc(display);  
 glutKeyboardFunc(keyboard\_handler);  
 glutReshapeFunc(reshape);  
  
 glEnable(GL\_DEPTH\_TEST);  
  
 glMatrixMode(GL\_PROJECTION);  
 glLoadIdentity();  
 gluPerspective(60, 1, 1, 10);  
  
 glMatrixMode(GL\_MODELVIEW);  
 glLoadIdentity();  
 gluLookAt(4, 2, 0, 0, 0, 0, 0, 1, 0);  
  
 x = 0;  
 y = 0;  
 z = 1;  
 rotation = false;  
   
 /\* tell GLUT to wait for events \*/  
 glutMainLoop();  
}  
  
void usage()  
{  
 printf("Usage:\n");  
 printf(" q, Q, ESC -- Exit\n");  
 printf(" 1 -- switch to scene #1\n");  
 printf(" 2 -- switch to scene #2\n");  
 printf(" r, R -- toggle rotation\n");  
 printf(" s, S -- start animating current scene\n");  
}  
  
void reshape(int width, int height)  
{  
 glViewport(0, 0, width, height);  
}  
  
void scene1()   
{  
 glPushMatrix();  
  
 glRotatef(angle1, 0, 1, 0);  
  
 glPushMatrix();  
 glRotatef(-90, 1, 0, 0);  
 glTranslatef(0, 1.5, -1);  
 glColor3f(0.92, 0.22, 0.87);  
 glutWireCone(1, 2.3, 20, 20);  
 glPopMatrix();  
  
  
 glPushMatrix();  
 glTranslatef(0, y, z);  
 glColor3f(RGB(23, 8, 184));  
 glutWireSphere(1, 10, 10);  
 glPopMatrix();  
  
 glPopMatrix();  
}  
  
void scene2()   
{  
 glPushMatrix();  
  
 glRotatef(angle2, 0, 1, 0);  
  
 glPushMatrix();  
 glRotatef(-90, 1, 0, 0);  
 glColor3f(0.92, 0.22, 0.87);  
 glTranslatef(0, -1, 0);  
 glScalef(0.5, 0.5, 0.5);  
 glScalef(scale, scale, scale);  
 glutWireTorus(0.2, 0.6, 10, 30);  
 glPopMatrix();  
  
 glPushMatrix();  
  
 glRotatef(-90, 1, 0, 0);  
 glColor3f(RGB(23, 8, 184));  
 glTranslatef(0, 1, 0);  
 glRotatef(sub\_angle\_stage\_1, 1, 0, 0);  
 glTranslatef(0, 0, -1);  
  
  
 GLUquadricObj \*quad = gluNewQuadric();  
 gluQuadricDrawStyle(quad, GLU\_LINE);  
  
 gluCylinder(quad, 0.5, 0.5, 2, 30, 20);  
 glPopMatrix();  
  
 glPopMatrix();  
}  
  
void animate1()   
{  
 frame++;  
 x += .005;  
 y -= .005;  
 z -= .005;  
   
 if (rotation) {  
 angle1 += 1;  
 angle1 = angle1 > 360 ? 0 : angle1;  
 }  
  
 y = y < -1 ? -1 : y;  
 z = z < -1.5 ? -1.5 : z;  
  
 if (z == 2.5) {  
 glutIdleFunc(NULL);  
 }  
  
 glutPostRedisplay();  
}  
  
void animate2()   
{  
 static bool stage1\_complete = false;  
  
 if (rotation) {  
 angle2 += 1;  
 angle2 = angle2 > 360 ? 0 : angle2;  
 }  
 sub\_angle\_stage\_1 += 1;  
 sub\_angle\_stage\_1 = sub\_angle\_stage\_1 > 180 ? 180 : sub\_angle\_stage\_1;  
  
 if (sub\_angle\_stage\_1 >= 180)  
 stage1\_complete = true;  
  
 if (stage1\_complete)   
 {  
 scale += 0.005;  
 scale = scale > 2.0 ? 2.0 : scale;  
 }  
 glutPostRedisplay();  
}  
  
void display ()   
{  
 /\* clear window \*/  
 glClearColor(1.0f, 1.0f, 0.8f, 1.0f);  
 glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);  
  
 /\* future matrix manipulations should affect the modelview matrix \*/  
 glMatrixMode(GL\_MODELVIEW);  
  
 if (scene\_number == scene\_1) {  
 scene1();  
 }  
 else if (scene\_number == scene\_2) {  
 scene2();  
 }  
  
 glFlush();  
  
 glutSwapBuffers();  
}  
  
void keyboard\_handler(unsigned char key, int x, int y)  
{  
 printf("%c %d %d\n", key, x, y);  
 if (key == 'q' || key == 'Q') {  
 glutDestroyWindow(1);  
 }  
 else if (key == 0x1b) {  
 glutDestroyWindow(1);  
 }  
 else if (key == '1') {  
 scene\_number = scene\_1;  
 glutPostRedisplay();  
 }   
 else if (key == '2') {  
 scene\_number = scene\_2;  
 glutPostRedisplay();  
 }   
 else if (key == 'R' || key == 'r') {  
 rotation = !rotation;  
 }   
 else if (key == 's' || key == 'S') {  
 if (scene\_number == scene\_1) {  
 glutIdleFunc(animate1);  
 } else if (scene\_number == scene\_2) {  
 glutIdleFunc(animate2);  
 }  
 }  
 else if (key == 'p' || key == 'P') {  
 char filename[512] = { 0 };  
 sprintf(filename, "%s\_%ld.tga", "screenshot", time(NULL));  
 screendump(filename, 800, 600);  
 }  
}  
  
/\*  
 \* Code by Paul Bourke  
 \* From: http://www.paulbourke.net/dataformats/tga/  
 \*/  
void write\_tga\_header(FILE \*fp, short width, short height)  
{  
 putc(0,fp);  
 putc(0,fp);  
 putc(2,fp); /\* uncompressed RGB \*/  
 putc(0,fp); putc(0,fp);  
 putc(0,fp); putc(0,fp);  
 putc(0,fp);  
 putc(0,fp); putc(0,fp); /\* X origin \*/  
 putc(0,fp); putc(0,fp); /\* y origin \*/  
 putc((width & 0x00FF),fp);  
 putc((width & 0xFF00) / 256,fp);  
 putc((height & 0x00FF),fp);  
 putc((height & 0xFF00) / 256,fp);  
 putc(24,fp); /\* 24 bit bitmap \*/  
 putc(0,fp);  
}  
  
void screendump(const char \*filename, short width, short height)  
{  
 unsigned int size = width \* height \* 3;  
 unsigned char \*pixels;   
 FILE \*fp = fopen(filename, "w");  
 if (!fp) {  
 perror("fopen");  
 return;  
 }  
  
 write\_tga\_header(fp, width, height);  
  
 pixels = new unsigned char[size];  
 glReadPixels(0, 0, width, height, GL\_BGR, GL\_UNSIGNED\_BYTE, pixels);  
  
 fwrite(pixels, size, 1, fp);  
 fclose(fp);  
  
 delete[] pixels;  
}