|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | **Министерство науки и высшего образования Российской Федерации**  Калужский филиал  федерального государственного бюджетного  образовательного учреждения высшего образования  ***«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»***  ***(КФ МГТУ им. Н.Э. Баумана)*** |

**ФАКУЛЬТЕТ** ***ИУК «Информатика и управление»***

**КАФЕДРА** \_\_***ИУК4 «Программное обеспечение ЭВМ, информационные технологии»***

**ЛАБОРАТОРНАЯ РАБОТА №2**

**«Основные примитивы OpenGL»**

**ДИСЦИПЛИНА: «Компьютерная графика»**

|  |  |  |
| --- | --- | --- |
| Выполнил: студент гр. ИУК4-42Б | | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ( Карельский М.К. )  (Подпись) |
| Проверил: | | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ( Глебов С.А. )  (Подпись) |
| Дата сдачи (защиты):  Результаты сдачи (защиты): | | |
|  | - Балльная оценка:  - Оценка: | |

Калуга, 2022

**Цель:** формирование практических навыков по работе с графическими примитивами OpenGL, а также применения к ним эффектов средствами машины состояний и использования проверки глубины.

**Задачи:**

* научиться устанавливать размеры наблюдаемого объема
* изучить параметры функции glVertex
* изучить основные параметры функции glBegin
* сформировать понимание особенности использования функции glEnable с конкретными геометрическими примитивами
* выяснить основы построения сплошных объектов.

**Задание:**

1. Для каждого приложения установить размеры наблюдаемого объема.
2. Для **Листинга 1** задать с использованием тригонометрических преобразований (функций sin и cos) произвольное множество точек (не менее 30 точек) в трехмерном пространстве. Изучить принципы вращения экрана с помощью клавиш курсора и способы подключения меню к программе.
3. Для **Листинга 2** задать произвольное множество точек разного размера и цвета в трехмерном пространстве.
4. Для **Листинга 3** изобразить не менее 10 линий с разными параметрами (цвет, толщина, начертание по шаблону) в трехмерном пространстве.
5. Для **Листинга 4** выполнить аппроксимацию произвольной кривой линии посредством коротких прямых. На 3 примерах наглядно продемонстрировать точность аппроксимации. По возможности использовать меню.
6. Для **Листинга 5**, используя GL\_LINE\_STRIP и GL\_LINE\_LOOP нарисовать трехмерный объект из ломанных линий содержащих не менее 15 точек. Точки задавать, используя цикл.
7. Для **Листинга 6** используя GL\_TRIANGLES вывести на экран произвольную трехмерную геометрическую фигуру, состоящую из треугольников. Понимать принципы обхода точек треугольника и полигонов и знать на что он влияет.
8. Для **Листинга 7** продемонстрировать работу директив GL\_TRIANGLE\_STRIP и GL\_TRIANGLE\_FAN.
9. Для **Листинга 8** для произвольно заданной фигуры составленной из не менее чем 6 треугольников продемонстрировать работу функции glShadeModel.
10. Для **Листинга 9** на наглядном примере продемонстрировать работу проверки глубины для технологии удаления скрытых поверхностей. На наглядном примере продемонстрировать работу технологии отбора задних гране.

**Листинг 1:**

#include "glew.h" // System and OpenGL Stuff

#include "glut.h"

#include <math.h>

namespace Listing\_1

{

// Определяется константа со значением “пи”

#define GL\_PI 3.1415f

// Значения углов

static GLfloat xRot = 0.0f;

static GLfloat yRot = 0.0f;

// Вызывается для рисования сцены

void RenderScene(void)

{

GLfloat x, y, z, angle; // Здесь хранятся координаты и углы

// Окно очищается текущим цветом очистки

glClear(GL\_COLOR\_BUFFER\_BIT);

// Записываем состояние матрицы и выполняем поворот

glPushMatrix();

glRotatef(xRot, 1.0f, 0.0f, 0.0f);

glRotatef(yRot, 0.0f, 1.0f, 0.0f);

// Вызываем один раз для всех оставшихся точек

glBegin(GL\_POINTS);

z = -75.0f;

for (angle = 0.0f; angle <= (2.0f \* GL\_PI) \* 10.0f; angle += 0.1f)

{

x = 25.0f \* sin(angle);

y = 25.0f \* cos(angle);

// Задаем точку и немного смещаем значение z

glVertex3f(x, y, z);

z += 0.25f;

}

// Рисуем точки

glEnd();

// Восстанавливаем преобразования

glPopMatrix();

// Очищаем стек команд преобразования

glutSwapBuffers();

}

// Функция выполняет необходимую инициализацию

// в контексте визуализации

void SetupRC()

{

// Черный фон

glClearColor(1.0f, 1.0f, 1.0f, 1.0f);

// Цвет рисования выбирается фиолетовым

glColor3f(1.0f, 0.0f, 1.0f);

}

void SpecialKeys(int key, int x, int y)

{

if (key == GLUT\_KEY\_UP)

xRot -= 5.0f;

if (key == GLUT\_KEY\_DOWN)

xRot += 5.0f;

if (key == GLUT\_KEY\_LEFT)

yRot -= 5.0f;

if (key == GLUT\_KEY\_RIGHT)

yRot += 5.0f;

if (key > 356.0f)

xRot = 0.0f;

if (key < -1.0f)

xRot = 355.0f;

if (key > 356.0f)

yRot = 0.0f;

if (key < -1.0f)

yRot = 355.0f;

// Обновляем окно

glutPostRedisplay();

}

void ChangeSize(int w, int h)

{

GLfloat nRange = 100.0f;

// Предотвращает деление на нуль

if (h == 0)

h = 1;

// Устанавливаем размеры поля просмотра равными размерам окна

glViewport(0, 0, w, h);

// Устанавливаем перспективную систему координат

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

// Устанавливает объем отсечения с помощью отсекающих

// плоскостей (левая, правая, нижняя, верхняя, ближняя, дальняя)

if (w <= h)

glOrtho(-nRange, nRange, -nRange \* h / w, nRange \* h / w, -nRange, nRange);

else

glOrtho(-nRange \* w / h, nRange \* w / h, -nRange, nRange, -nRange, nRange);

// Обновляется стек матриц проекции модели

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

}

int main(int argc, char\* argv[])

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB | GLUT\_DEPTH);

glutCreateWindow("Listing 1");

glutReshapeFunc(ChangeSize);

glutSpecialFunc(SpecialKeys);

glutDisplayFunc(RenderScene);

SetupRC();

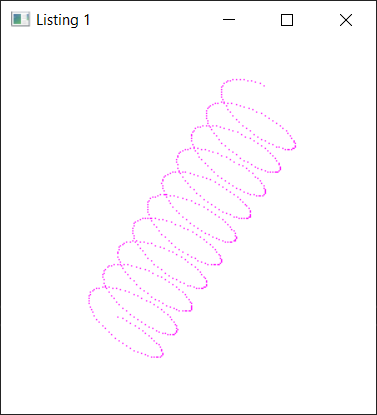
glutMainLoop();

return 0;

}

}

**Результат листинга 1:**



**Рисунок 1.** Результат листинга 1

**Листинг 2:**

...

void RenderScene(void)

{

GLfloat x, y, z, angle; // Место хранения координат и углов

GLfloat sizes[2]; //Запоминаем диапазон размеров поддерживаемых точек

GLfloat step; // Запоминаем поддержи\_ваемый инкремент размеров точек

GLfloat curSize; // Записываем размер текущих точек

GLfloat minSize; // Записываем минимальный размер точки

GLfloat maxSize; // Записываем максимальный размер точки

GLfloat blueColor = 1.0f;

GLfloat colorStep = 0.005f;

// Очищаем окно текущим цветом очистки

glClear(GL\_COLOR\_BUFFER\_BIT);

// Записывается состояние матрицы и выполняются повороты

glPushMatrix();

glRotatef(xRot, 1.0f, 0.0f, 0.0f);

glRotatef(yRot, 0.0f, 1.0f, 0.0f);

// Получаем диапазон размеров поддерживаемых точек и размер шага

glGetFloatv(GL\_POINT\_SIZE\_RANGE, sizes);

glGetFloatv(GL\_POINT\_SIZE\_GRANULARITY, &step);

// Задаем исходный размер точки

curSize = sizes[0];

minSize = sizes[0];

maxSize = sizes[1];

short multiplier = 3;

// Задаем начальную координату z

z = -75.0f;

// Циклический проход по окружности три раза

for (angle = 0.0f; angle <= (2.0f \* 3.1415f) \* 5.0f; angle += 3.1415f \* 0.1f)

{

// Расчет значений x и у точек окружности

x = 25.0f \* sin(angle);

y = 25.0f \* cos(angle);

// Задаем размер точки перед указанием при\_митива

glPointSize(curSize);

glColor3f(1.0f, 0.0f, blueColor);

// Рисуем точку

glBegin(GL\_POINTS);

glVertex3f(x, y, z);

glEnd();

// Увеличиваем значение z и размер точки

z += 1.5f;

if (curSize >= maxSize / 5)

multiplier = -3;

else if (curSize <= minSize)

multiplier = 3;

curSize += multiplier \* step;

blueColor -= colorStep;

}

// Восстанавливаем преобразования

glPopMatrix();

// Очищаем стек команд рисования

glutSwapBuffers();

}

...

**Результат листинга 2:**



**Рисунок 2.** Результат листинга 2

**Листинг 3:**

...

void RenderScene(void)

{

// Вызывается один раз для всех оставшихся точек

GLfloat x, y, z, angle; // Здесь хранятся координаты и углы

GLfloat fSizes[2]; // Метрики диапазона ширины линий

GLfloat fCurrSize; // Запись текущего состояния

GLfloat blueColor = 1.0f;

GLfloat colorStep = 0.04f;

GLint factor = 1; // Множитель фактуры

GLushort pattern = 0xFF00; // Шаблон фактуры

GLushort patternStep = 1;

// Очищаем окно текущим цветом очистки

glClear(GL\_COLOR\_BUFFER\_BIT);

glPushMatrix();

glRotatef(xRot, 1.0f, 0.0f, 0.0f);

glRotatef(yRot, 0.0f, 1.0f, 0.0f);

glGetFloatv(GL\_LINE\_WIDTH\_RANGE, fSizes);

fCurrSize = fSizes[0];

// Активизируется фактура

glEnable(GL\_LINE\_STIPPLE);

// Пошаговый проход оси у по 20 единиц за раз

for (y = -90.0f; y < 90.0f; y += 10.0f)

{

// Задается ширина линии

glLineWidth(fCurrSize);

glColor3f(1.0f, 0.0f, blueColor);

// Обновляется множитель повтора и шаблон

glLineStipple(factor, pattern);

// Рисуется линия

glBegin(GL\_LINES);

glVertex2f(-80.0f, y);

glVertex2f(80.0f, y);

glEnd();

// Увеличивается ширина линии

fCurrSize += 0.5f;

pattern += patternStep;

blueColor -= colorStep;

}

// Восстанавливаем преобразования

glPopMatrix();

// Очищаем стек команд рисования

glutSwapBuffers();

}

...

**Результат листинга 3:**



**Рисунок 3.** Результат листинга 3

**Листинг 4:**

...

bool isPointsVisible = true;

bool isLinesVisible = true;

// Вызывается для рисования сцены

void RenderScene(void)

{

GLfloat x, y, angle; // Здесь хранятся координаты и углы

// Окно очищается текущим цветом очистки

glClear(GL\_COLOR\_BUFFER\_BIT);

// Записываем состояние матрицы и выполняем поворот

glPushMatrix();

glRotatef(xRot, 1.0f, 0.0f, 0.0f);

glRotatef(yRot, 0.0f, 1.0f, 0.0f);

if (isLinesVisible)

{

glBegin(GL\_LINE\_STRIP);

for (angle = 0.0f; angle <= 2.0f \* GL\_PI; angle += GL\_PI \* 0.1f)

{

x = 25.0f \* (-GL\_PI + angle);

y = 25.0f \* sin(angle);

// Задаем точку и немного смещаем значение z

glVertex2f(x, y);

}

glEnd();

}

if (isPointsVisible)

{

glPointSize(5.0f);

// Вызываем один раз для всех оставшихся точек

glBegin(GL\_POINTS);

for (angle = 0.0f; angle <= 2.0f \* GL\_PI; angle += GL\_PI \* 0.1f)

{

x = 25.0f \* (-GL\_PI + angle);

y = 25.0f \* sin(angle);

// Задаем точку и немного смещаем значение z

glVertex2f(x, y);

}

// Рисуем точки

glEnd();

}

// Восстанавливаем преобразования

glPopMatrix();

// Очищаем стек команд преобразования

glutSwapBuffers();

}

...

void processMenuEvents(int option) {

switch (option) {

case 1:

isLinesVisible = !isLinesVisible;

break;

case 2:

isPointsVisible = !isPointsVisible;

break;

}

RenderScene();

}

int main(int argc, char\* argv[])

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB | GLUT\_DEPTH);

glutCreateWindow("Listing 4");

glutReshapeFunc(ChangeSize);

glutSpecialFunc(SpecialKeys);

glutDisplayFunc(RenderScene);

SetupRC();

glutCreateMenu(processMenuEvents);

//добавить записи в нашем меню

glutAddMenuEntry("Change lines visibility", 1);

glutAddMenuEntry("Change points visibility", 2);

// прикрепить меню правой кнопки

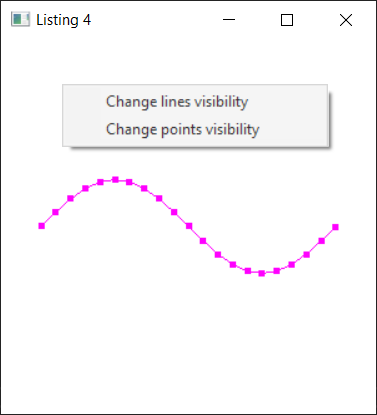
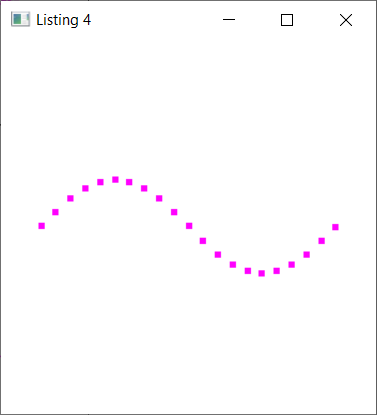
glutAttachMenu(GLUT\_RIGHT\_BUTTON);

glutMainLoop();

return 0;

}

**Результат листинга 4:**

**Рисунок 4.** Результат листинга 4

**Листинг 5:**

...

void RenderScene(void)

{

GLfloat x, y, z, angle; // Здесь хранятся координаты и углы

// Окно очищается текущим цветом очистки

glClear(GL\_COLOR\_BUFFER\_BIT);

// Записываем состояние матрицы и выполняем поворот

glPushMatrix();

glRotatef(xRot, 1.0f, 0.0f, 0.0f);

glRotatef(yRot, 0.0f, 1.0f, 0.0f);

// Вызываем один раз для всех оставшихся точек

glBegin(GL\_LINE\_LOOP);

for (angle=0.0f; angle<=2.0f\*GL\_PI; angle+=GL\_PI\*0.1f)

{

x = 50.0f \* sin(angle);

y = 50.0f \* cos(angle);

glVertex2f(x, y);

}

glEnd();

x = 0;

glBegin(GL\_LINE\_STRIP);

for (angle = 0.0f; angle <= GL\_PI; angle += GL\_PI \* 0.2f)

{

z = 50.0f \* sin(angle);

y = 50.0f \* cos(angle);

glVertex3f(x, y, z);

}

glEnd();

y = 0;

glBegin(GL\_LINE\_STRIP);

for (angle = 0.0f; angle <= GL\_PI; angle += GL\_PI \* 0.2f)

{

z = 50.0f \* sin(angle);

x = 50.0f \* cos(angle);

glVertex3f(x, y, z);

}

glEnd();

// Восстанавливаем преобразования

glPopMatrix();

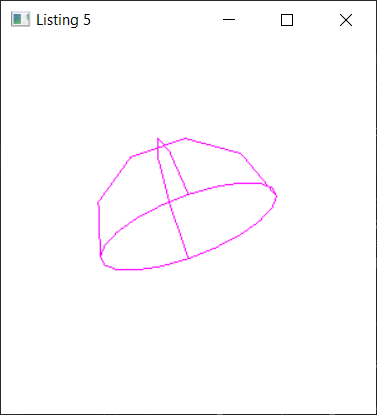
// Очищаем стек команд преобразования

glutSwapBuffers();

}

...

**Результат листинга 5:**



**Рисунок 5.** Результат листинга 5

**Листинг 6:**

...

void RenderScene(void)

{

GLfloat x, y, z, angle; // Здесь хранятся координаты и углы

// Окно очищается текущим цветом очистки

glClear(GL\_COLOR\_BUFFER\_BIT);

glEnable(GL\_CULL\_FACE);

// Записываем состояние матрицы и выполняем поворот

glPushMatrix();

glRotatef(xRot, 1.0f, 0.0f, 0.0f);

glRotatef(yRot, 0.0f, 1.0f, 0.0f);

glColor3f(0.0f, 0.0f, 0.0f);

glBegin(GL\_TRIANGLES);

glVertex2f(0.0f, 40.0f);

glVertex2f(30.0f, -20.0f);

glVertex2f(-30.0f, -20.0f);

glEnd();

glColor3f(1.0f, 0.0f, 0.0f);

glBegin(GL\_TRIANGLES);

glVertex3f(0.0f, 0.0f, 60.0f);

glVertex2f(30.0f, -20.0f);

glVertex2f(0.0f, 40.0f);

glEnd();

glColor3f(0.0f, 1.0f, 0.0f);

glBegin(GL\_TRIANGLES);

glVertex3f(0.0f, 0.0f, 60.0f);

glVertex2f(0.0f, 40.0f);

glVertex2f(-30.0f, -20.0f);

glEnd();

glColor3f(0.0f, 0.0f, 1.0f);

glBegin(GL\_TRIANGLES);

glVertex3f(0.0f, 0.0f, 60.0f);

glVertex2f(-30.0f, -20.0f);

glVertex2f(30.0f, -20.0f);

glEnd();

// Восстанавливаем преобразования

glPopMatrix();

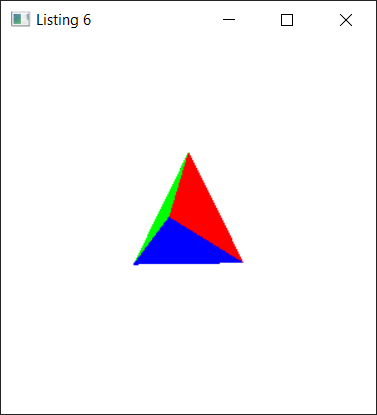
// Очищаем стек команд преобразования

glutSwapBuffers();

}

...

**Результат листинга 6:**



**Рисунок 6.** Результат листинга 6

**Листинг 7:**

...

void RenderScene(void)

{

GLfloat x, y, z, angle; // Здесь хранятся координаты и углы

// Окно очищается текущим цветом очистки

glClear(GL\_COLOR\_BUFFER\_BIT);

// Записываем состояние матрицы и выполняем поворот

glPushMatrix();

glRotatef(xRot, 1.0f, 0.0f, 0.0f);

glRotatef(yRot, 0.0f, 1.0f, 0.0f);

glEnable(GL\_CULL\_FACE);

glBegin(GL\_TRIANGLE\_STRIP);

glColor3f(1.0f, 0.0f, 1.0f);

glVertex2f(40.0f, 40.0f);

glVertex2f(40.0f, -40.0f);

glVertex2f(-40.0f, 40.0f);

glColor3f(1.0f, 1.0f, 0.0f);

glVertex2f(-40.0f, -40.0f);

glEnd();

glBegin(GL\_TRIANGLE\_FAN);

glColor3f(0.0f, 1.0f, 1.0f);

glVertex3f(0.0f, 0.0f, 40.0f);

glVertex2f(40.0f, -40.0f);

glColor3f(1.0f, 0.0f, 0.0f);

glVertex2f(40.0f, 40.0f);

glColor3f(0.0f, 1.0f, 0.0f);

glVertex2f(-40.0f, 40.0f);

glColor3f(0.0f, 0.0f, 1.0f);

glVertex2f(-40.0f, -40.0f);

glColor3f(0.0f, 1.0f, 1.0f);

glVertex2f(40.0f, -40.0f);

glEnd();

// Восстанавливаем преобразования

glPopMatrix();

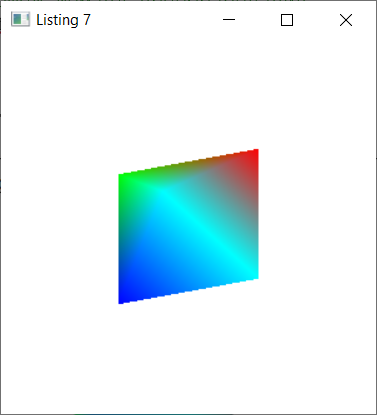
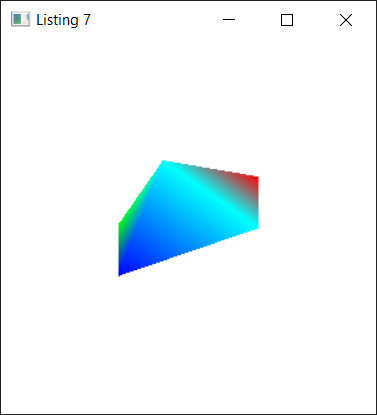
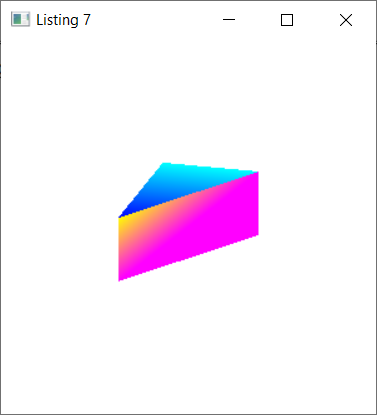
// Очищаем стек команд преобразования

glutSwapBuffers();

}

...

**Результат листинга 7:**

**Рисунок 7.** Результат листинга 7

**Листинг 8:**

...

bool flatMode = false;

// Вызывается для рисования сцены

void RenderScene(void)

{

GLfloat x, y, z, angle; // Здесь хранятся координаты и углы

// Окно очищается текущим цветом очистки

glClear(GL\_COLOR\_BUFFER\_BIT);

// Записываем состояние матрицы и выполняем поворот

glPushMatrix();

glRotatef(xRot, 1.0f, 0.0f, 0.0f);

glRotatef(yRot, 0.0f, 1.0f, 0.0f);

glEnable(GL\_CULL\_FACE);

glBegin(GL\_TRIANGLE\_STRIP);

glColor3f(1.0f, 0.0f, 1.0f);

glVertex2f(40.0f, 40.0f);

glVertex2f(40.0f, -40.0f);

glVertex2f(-40.0f, 40.0f);

glColor3f(1.0f, 1.0f, 0.0f);

glVertex2f(-40.0f, -40.0f);

glEnd();

glBegin(GL\_TRIANGLE\_FAN);

glColor3f(0.0f, 1.0f, 1.0f);

glVertex3f(0.0f, 0.0f, 40.0f);

glVertex2f(40.0f, -40.0f);

glColor3f(1.0f, 0.0f, 0.0f);

glVertex2f(40.0f, 40.0f);

glColor3f(0.0f, 1.0f, 0.0f);

glVertex2f(-40.0f, 40.0f);

glColor3f(0.0f, 0.0f, 1.0f);

glVertex2f(-40.0f, -40.0f);

glColor3f(0.0f, 1.0f, 1.0f);

glVertex2f(40.0f, -40.0f);

glEnd();

// Восстанавливаем преобразования

glPopMatrix();

// Очищаем стек команд преобразования

glutSwapBuffers();

}

...

void processMenuEvents(int option)

{

if (flatMode)

glShadeModel(GL\_SMOOTH);

else

glShadeModel(GL\_FLAT);

flatMode = !flatMode;

RenderScene();

}

int main(int argc, char\* argv[])

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB | GLUT\_DEPTH);

glutCreateWindow("Listing 8");

glutReshapeFunc(ChangeSize);

glutSpecialFunc(SpecialKeys);

glutDisplayFunc(RenderScene);

SetupRC();

glutCreateMenu(processMenuEvents);

//добавить записи в нашем меню

glutAddMenuEntry("Change shade model", 1);

// прикрепить меню правой кнопки

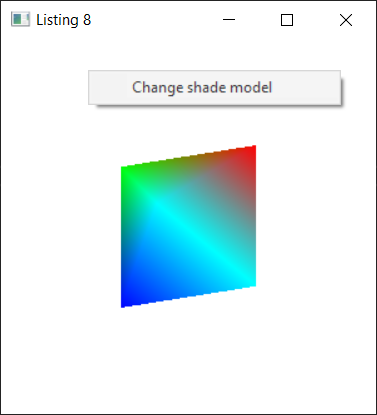
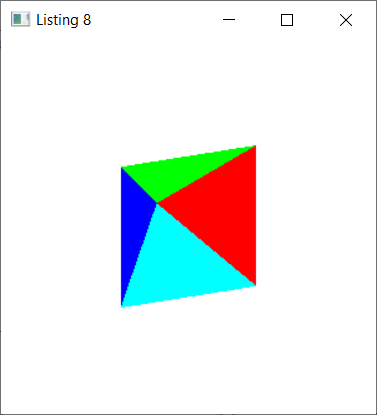
glutAttachMenu(GLUT\_RIGHT\_BUTTON);

glutMainLoop();

return 0;

}

**Результат листинга 8:**

**Рисунок 8.** Результат листинга 8

**Листинг 9:**

...

bool depthCheck = false;

bool culling = false;

bool showBottom = true;

// Вызывается для рисования сцены

void RenderScene(void)

{

GLfloat x, y, z, angle; // Здесь хранятся координаты и углы

// Окно очищается текущим цветом очистки

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

// Записываем состояние матрицы и выполняем поворот

glPushMatrix();

glRotatef(xRot, 1.0f, 0.0f, 0.0f);

glRotatef(yRot, 0.0f, 1.0f, 0.0f);

if (showBottom)

{

glBegin(GL\_TRIANGLE\_STRIP);

glColor3f(1.0f, 0.0f, 1.0f);

glVertex2f(40.0f, 40.0f);

glVertex2f(40.0f, -40.0f);

glVertex2f(-40.0f, 40.0f);

glColor3f(1.0f, 1.0f, 0.0f);

glVertex2f(-40.0f, -40.0f);

glEnd();

}

glBegin(GL\_TRIANGLE\_FAN);

glColor3f(0.0f, 1.0f, 1.0f);

glVertex3f(0.0f, 0.0f, 40.0f);

glVertex2f(40.0f, -40.0f);

glColor3f(1.0f, 0.0f, 0.0f);

glVertex2f(40.0f, 40.0f);

glColor3f(0.0f, 1.0f, 0.0f);

glVertex2f(-40.0f, 40.0f);

glColor3f(0.0f, 0.0f, 1.0f);

glVertex2f(-40.0f, -40.0f);

glColor3f(0.0f, 1.0f, 1.0f);

glVertex2f(40.0f, -40.0f);

glEnd();

// Восстанавливаем преобразования

glPopMatrix();

// Очищаем стек команд преобразования

glutSwapBuffers();

}

...

void processMenuEvents(int option)

{

switch (option) {

case 1:

if (depthCheck)

glDisable(GL\_DEPTH\_TEST);

else

glEnable(GL\_DEPTH\_TEST);

depthCheck = !depthCheck;

break;

case 2:

if (culling)

glDisable(GL\_CULL\_FACE);

else

glEnable(GL\_CULL\_FACE);

culling = !culling;

break;

case 3:

showBottom = !showBottom;

break;

}

RenderScene();

}

int main(int argc, char\* argv[])

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB | GLUT\_DEPTH);

glutCreateWindow("Listing 9");

glutReshapeFunc(ChangeSize);

glutSpecialFunc(SpecialKeys);

glutDisplayFunc(RenderScene);

SetupRC();

glutCreateMenu(processMenuEvents);

//добавить записи в нашем меню

glutAddMenuEntry("Change depth checking mode", 1);

glutAddMenuEntry("Change culling mode", 2);

glutAddMenuEntry("Change bottom visibility", 3);

// прикрепить меню правой кнопки

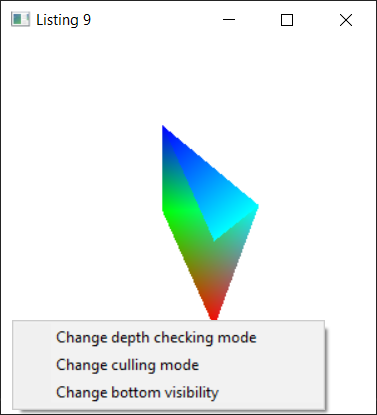
glutAttachMenu(GLUT\_RIGHT\_BUTTON);

glutMainLoop();

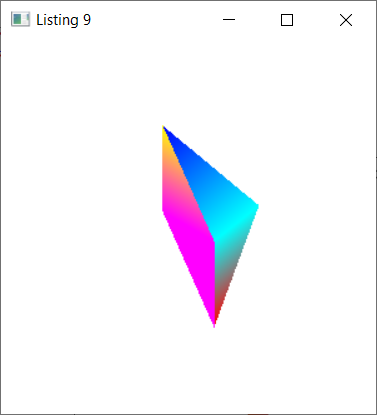
return 0;

}

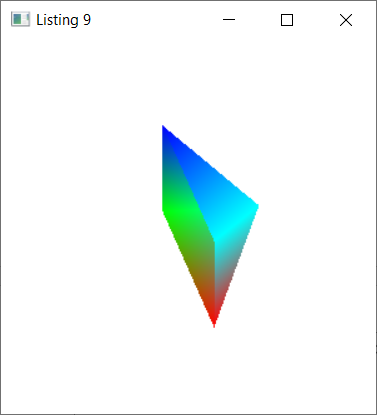
**Результат листинга 9:**



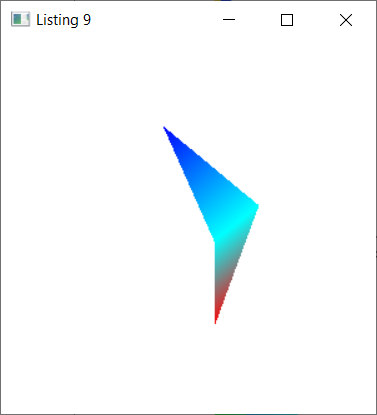
**Рисунок 9.1.** Результат листинга 9: без проверки глубины и отбора, с нижней гранью



**Рисунок 9.2.** Результат листинга 9: без отбора, с проверкой глубины и нижней гранью



**Рисунок 9.3.** Результат листинга 9: без отбора и нижней грани, с проверкой глубины



**Рисунок 9.4.** Результат листинга 9: без нижней грани, с проверкой глубины и отбором

**Вывод:** в ходе выполнения лабораторной работы были получены практические навыки работы с графическими примитивами разными способами: точками (изменение размера, цвета), прямыми (изменение толщины, фактуры, цвета) и полигонами (изменение цвета, способа заполнения цветом); создания меню, реализации вращения сцены, проверки глубины и отбора задних граней.