**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра МО ЭВМ**

**Лабораторная работа № 1**

**"Примитивы OpenGL"**

|  |  |  |
| --- | --- | --- |
| Студент гр. 5383 |  | Допира В. Е. |
| Преподаватель |  | Герасимова Т.В. |

Санкт-Петербург

2018

**Лабораторная работа № 1**

**"Примитивы OpenGL"**

Цель: ознакомление с основными примитивами OpenGL.

Требования и рекомендации к выполнению задания

* проанализировать полученное задание, выделить информационные объекты и действия;
* разработать программу с использованием требуемых примитивов и атрибутов.

Задание

Разработать программу, реализующую представление определенного набора примитивов из имеющихся в OpenGL (GL\_POINT, GL\_LINES, GL\_LINE\_STRIP, GL\_LINE\_LOOP, GL\_TRIANGLES, GL\_TRIANGLE\_STRIP, GL\_TRIANGLE\_FAN, GL\_QUADS, GL\_QUAD\_STRIP, GL\_POLYGON). Разработанная на базе шаблона программа должна быть пополнена возможностями остановки интерактивно различных атрибутов примитивов рисования через вызов соответствующих элементов интерфейса пользователя.

Общие сведения

В лабораторной работе должны быть рассмотрены следующие примитивы:

GL\_POINTS – каждая вершина рассматривается как отдельная точка, параметры которой не зависят от параметров остальных заданных точек. При этом вершина n определяет точку n. Рисуется N точек (n – номер текущей вершины, N – общее число вершин).

Основой графики OpenGL являются вершины. Для их определения используется команда glVertex. Вызов команды определяется четырьмя координатами x, y, z и w. При этом вызов glVertex2\* устанавливает координаты x и y, координата z полагается равной 0, а w – 1. Вызов glVertex3\* устанавливает координаты x, y, z, а w равно 1.

GL\_LINES – каждая пара вершин рассматривается как независимый отрезок. Первые две вершины определяют первый отрезок, следующие две – второй отрезок и т.д., вершины (2n-1) и 2n определяют отрезок n. Всего рисуется N/2 линий. Если число вершин нечетно, то последняя просто игнорируется.

GL\_LINE\_STRIP – в этом режиме рисуется последовательность из одного или нескольких связанных отрезков. Первая вершина задает начало первого отрезка, а вторая – конец первого, который является также началом второго. В общем случае, вершина n (n > 1) определяет начало отрезка n и конец отрезка (n - 1). Всего рисуется (N - 1) отрезок.

GL\_LINE\_LOOP – осуществляется рисование замкнутой кривой линии. Первая вершина задает начало первого отрезка, а вторая – конец первого, который является также началом второго. В общем случае, вершина n (n > 1) определяет начало отрезка n и конец отрезка (n - 1). Первая вершина является концом последнего отрезка. Всего рисуется N отрезков.

GL\_TRIANGLES – каждая тройка вершин рассматривается как независимый треугольник. Вершины (3n-2), (3n-1), 3n (в таком порядке) определяют треугольник n. Если число вершин не кратно 3, то оставшиеся ( одна или две) вершины игнорируются. Всего рисуется N/3 треугольника.

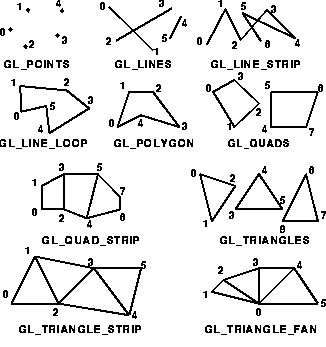
GL\_TRIANGLE\_STRIP - в этом режиме рисуется группа связанных треугольников, имеющих общую грань. Первые три вершины определяют первый треугольник, вторая, третья и четвертая – второй и т.д. для нечетного n вершины n, (n+1) и (n+2) определяют треугольник n. Для четного n треугольник определяют вершины (n+1), n и (n+2). Всего рисуется (N-2) треугольника.

GL\_TRIANGLE\_FAN - в этом режиме рисуется группа связанных треугольников, имеющих общие грани и одну общую вершину. Первые три вершины определяют первый треугольник, первая, третья и четвертая – второй и т.д. Всего рисуется (N-2) треугольника.

GL\_QUADS – каждая группа из четырех вершин рассматривается как независимый четырехугольник. Вершины (4n-3), (4n-2), (4n-1) и 4n определяют четырехугольник n. Если число вершин не кратно 4, то оставшиеся (одна, две или три) вершины игнорируются. Всего рисуется N/4 четырехугольника.

GL\_QUAD\_STRIP – рисуется группа четырехугольников, имеющих общую грань. Первая группа из четырех вершин задает первый четырехугольник. Третья, четвертая, пятая и шестая задают второй четырехугольник.

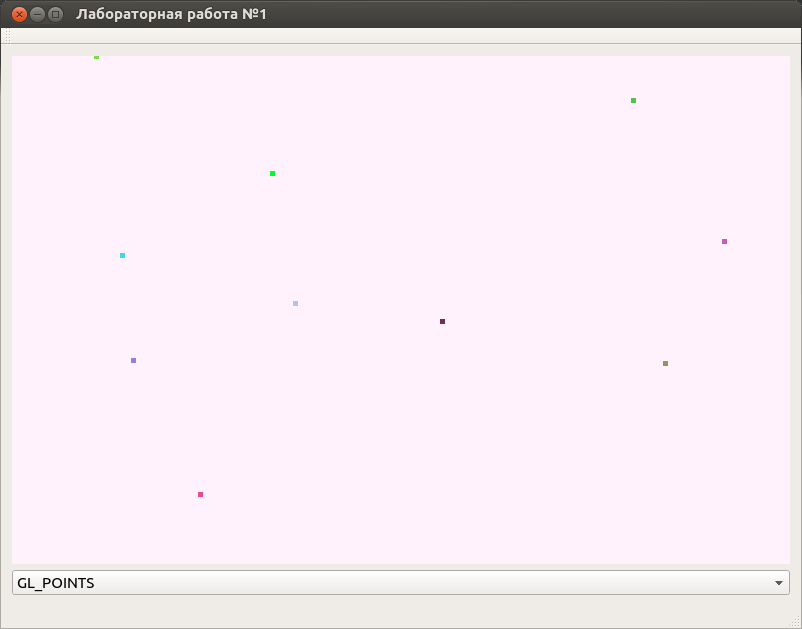
GL\_POLYGON – задет многоугольник. При этом число вершин равно числу вершин рисуемого многоугольника.



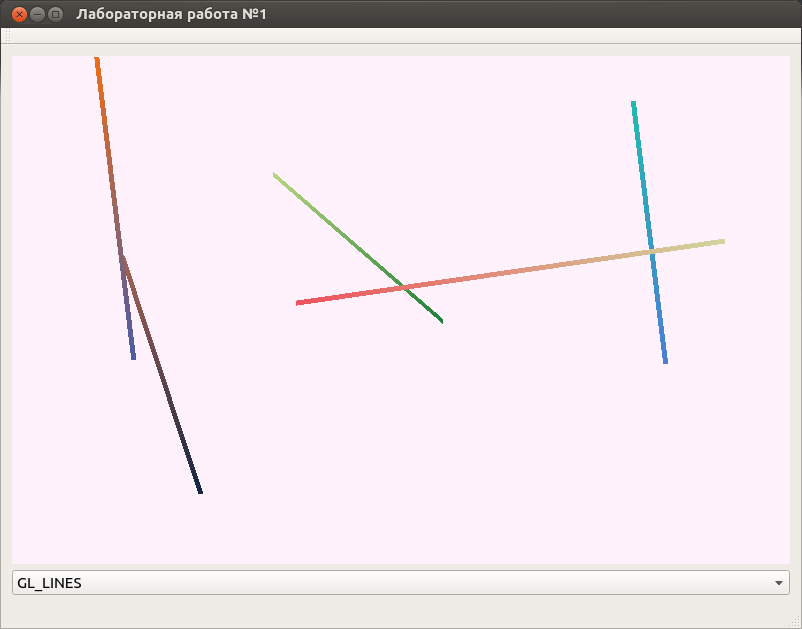
Тестирование

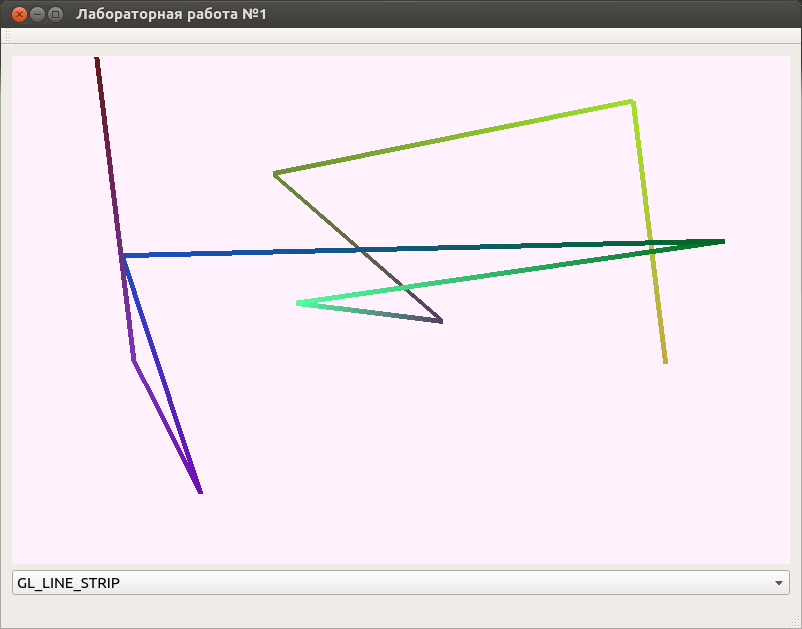
Работа выполнена в среде разработки Qt. Тестирование проводилось методом черного ящика. Результаты тестирования представлены на снимках экрана.

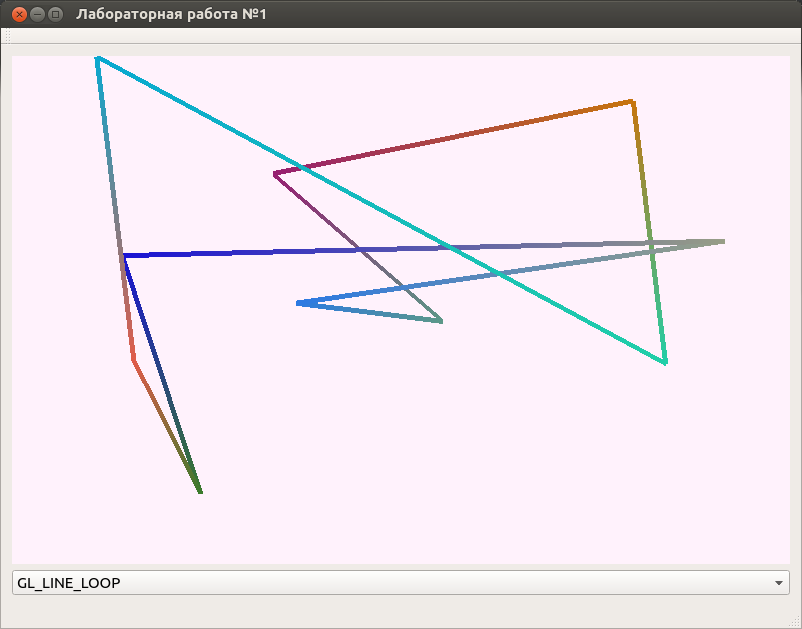
GL\_POINTS



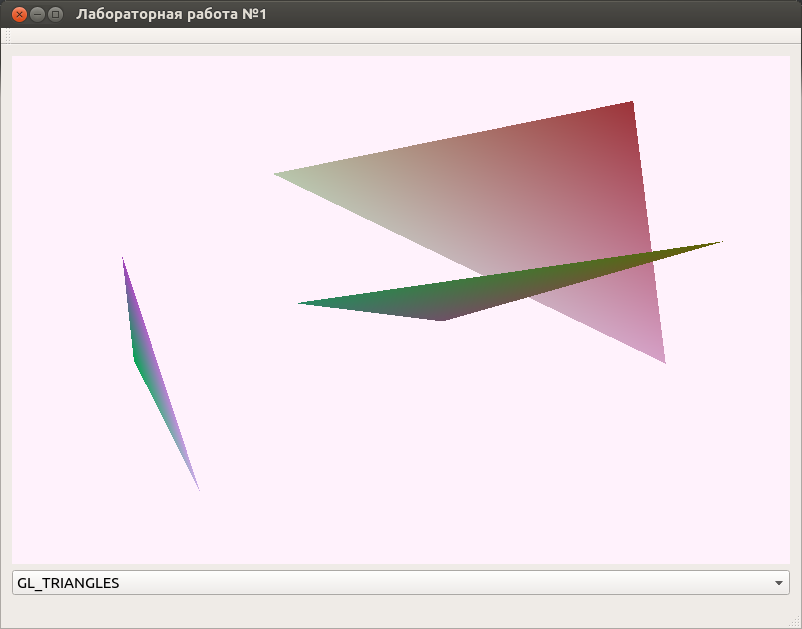
GL\_LINES

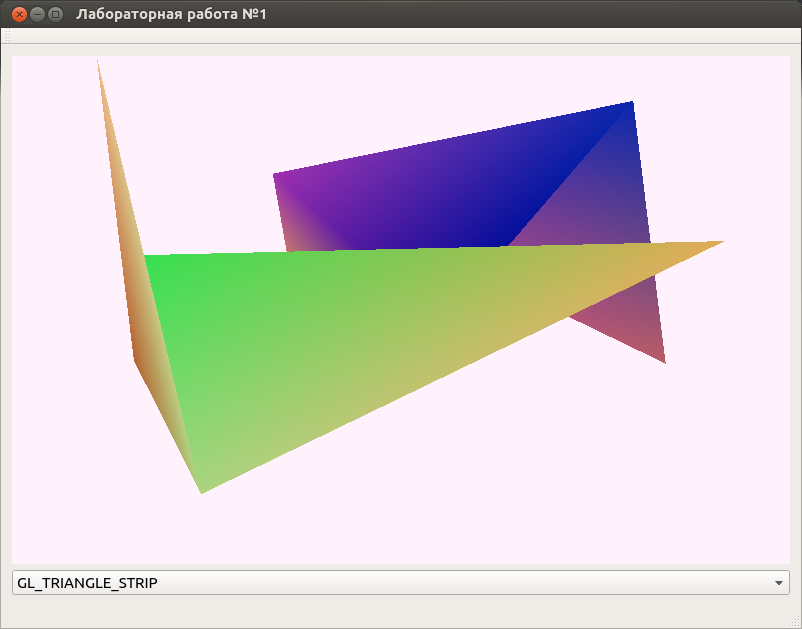
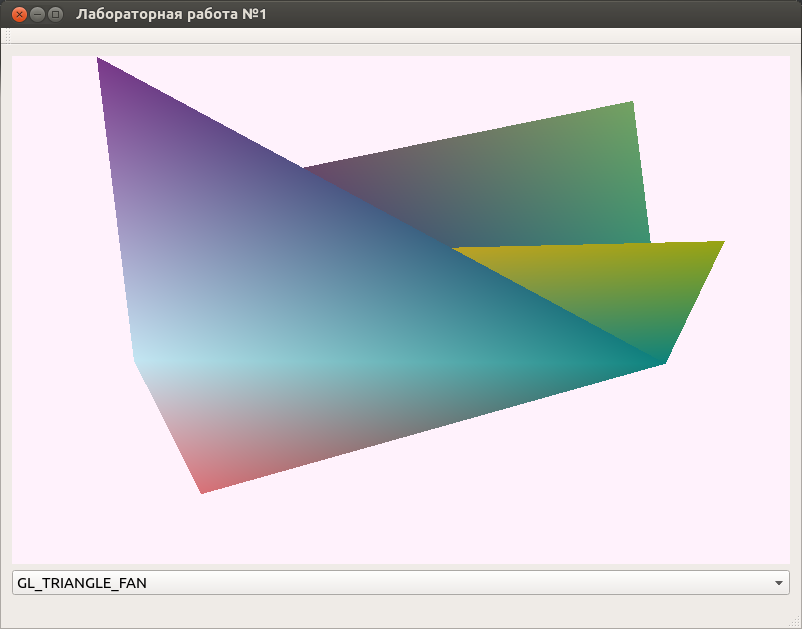
GL\_LINE\_STRIP

GL\_LINE\_LOOP

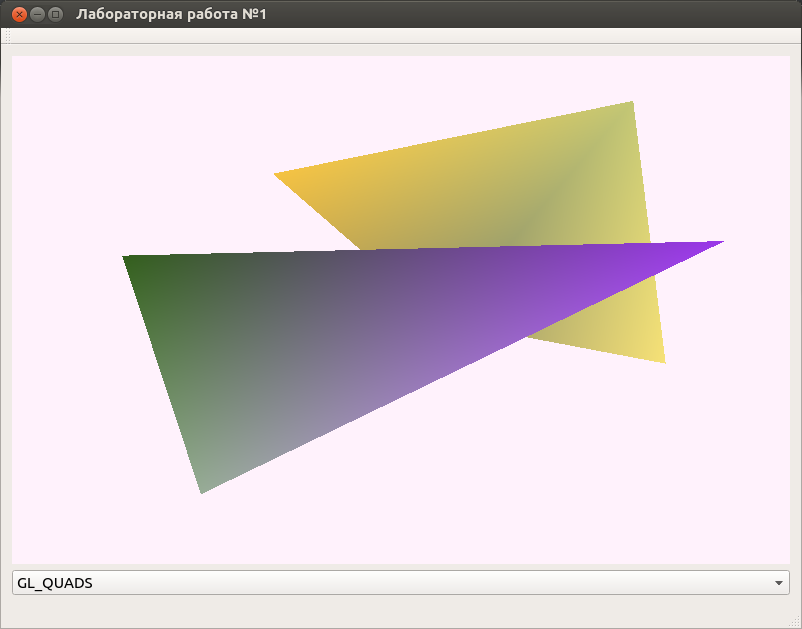
******

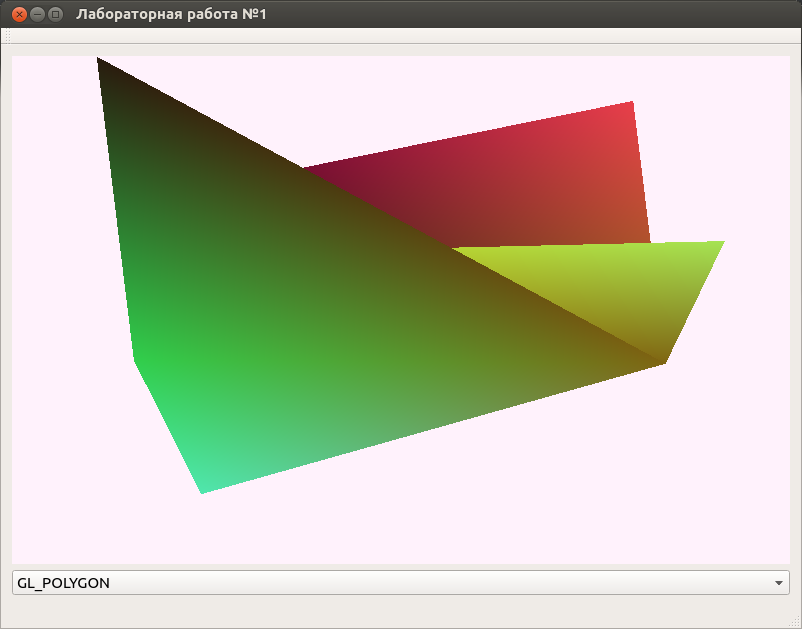
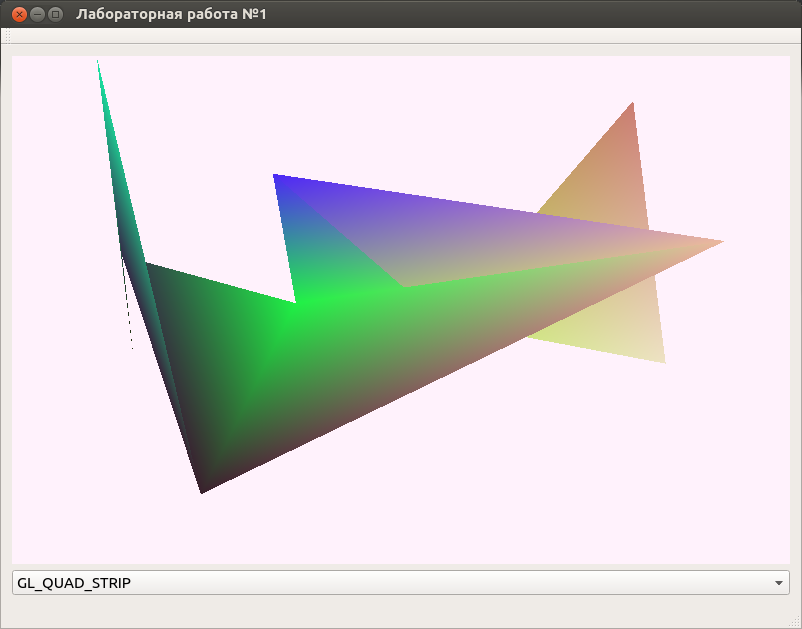
GL\_TRIANGLES

GL\_TRIANGLE\_STRIP

GL\_TRIANGLE\_FAN

GL\_QUADS

GL\_QUAD\_STRIP

GL\_POLYGON

Код программы

**gl\_widget.h**

#ifndef GL\_WINDGET\_H

#define GL\_WINDGET\_H

#include <QGLWidget>

struct Point3Df

{

double x;

double y;

double z;

};

class GL\_Widget: public QGLWidget

{

Q\_OBJECT

private:

int m\_primitiveIndex;

QList<Point3Df> m\_points;

public:

explicit GL\_Widget(QWidget \*parent = 0);

void initializeGL();

void paintGL();

void figuresGL();

void startDrawing();

public slots:

void setPrimitive(int p);

};

#endif // GL\_WINDGET\_H

**gl\_widget.cpp**

#include "gl\_widget.h"

GL\_Widget::**GL\_Widget**(QWidget \*parent):

QGLWidget(parent)

{

m\_primitiveIndex=0;

int pointsCount = 10;

double floor = -1;

double up = 1;

for(int i = 0; i < pointsCount; i++)

{

Point3Df tmp;

tmp.x = (double) (rand()) / RAND\_MAX \* (up - floor) + floor;

tmp.y = (double) (rand()) / RAND\_MAX \* (up - floor) + floor;

tmp.z = (double) (rand()) / RAND\_MAX \* (up - floor) + floor;

m\_points.push\_back(tmp);

}

}

void GL\_Widget::***initializeGL***(){

glClearColor(1, 0.95, 0.99, 0.93);

}

void GL\_Widget::***paintGL***(){

glClear(GL\_COLOR\_BUFFER\_BIT);

startDrawing();

}

void GL\_Widget::**figuresGL**(){

switch (m\_primitiveIndex) {

case 0:

{

glPointSize(5);

glBegin(GL\_POINTS);

for(auto point : m\_points)

{

glColor4f((double) (rand()) / RAND\_MAX,

(double) (rand()) / RAND\_MAX,

(double) (rand()) / RAND\_MAX,

(double) (rand()) / RAND\_MAX);

glVertex3f(point.x, point.y, point.z);

}

glEnd();

break;

};

case 1:

{

glLineWidth(5);

glBegin(GL\_LINES);

for(auto point : m\_points)

{

glColor4f((double) (rand()) / RAND\_MAX,

(double) (rand()) / RAND\_MAX,

(double) (rand()) / RAND\_MAX,

(double) (rand()) / RAND\_MAX);

glVertex3f(point.x, point.y, point.z);

}

glEnd();

break;

};

case 2:{

glLineWidth(5);

glBegin(GL\_LINE\_STRIP);

for(auto point : m\_points)

{

glColor4f((double) (rand()) / RAND\_MAX,

(double) (rand()) / RAND\_MAX,

(double) (rand()) / RAND\_MAX,

(double) (rand()) / RAND\_MAX);

glVertex3f(point.x, point.y, point.z);

}

glEnd();

break;

};

case 3:{

glLineWidth(5);

glBegin(GL\_LINE\_LOOP);

for(auto point : m\_points)

{

glColor4f((double) (rand()) / RAND\_MAX,

(double) (rand()) / RAND\_MAX,

(double) (rand()) / RAND\_MAX,

(double) (rand()) / RAND\_MAX);

glVertex3f(point.x, point.y, point.z);

}

glEnd();

break;

};

case 4:{

glLineWidth(5);

glBegin(GL\_TRIANGLES);

for(auto point : m\_points)

{

glColor4f((double) (rand()) / RAND\_MAX,

(double) (rand()) / RAND\_MAX,

(double) (rand()) / RAND\_MAX,

(double) (rand()) / RAND\_MAX);

glVertex3f(point.x, point.y, point.z);

}

glEnd();

break;

};

case 5:{

glLineWidth(5);

glBegin(GL\_TRIANGLE\_STRIP);

for(auto point : m\_points)

{

glColor4f((double) (rand()) / RAND\_MAX,

(double) (rand()) / RAND\_MAX,

(double) (rand()) / RAND\_MAX,

(double) (rand()) / RAND\_MAX);

glVertex3f(point.x, point.y, point.z);

}

glEnd();

break;

};

case 6:{

glBegin(GL\_TRIANGLE\_FAN);

glLineWidth(5);

for(auto point : m\_points)

{

glColor4f((double) (rand()) / RAND\_MAX,

(double) (rand()) / RAND\_MAX,

(double) (rand()) / RAND\_MAX,

(double) (rand()) / RAND\_MAX);

glVertex3f(point.x, point.y, point.z);

}

glEnd();

break;

};

case 7:{

glLineWidth(5);

glBegin(GL\_QUADS);

for(auto point : m\_points)

{

glColor4f((double) (rand()) / RAND\_MAX,

(double) (rand()) / RAND\_MAX,

(double) (rand()) / RAND\_MAX,

(double) (rand()) / RAND\_MAX);

glVertex3f(point.x, point.y, point.z);

}

glEnd();

break;

};

case 8:{

glLineWidth(5);

glBegin(GL\_QUAD\_STRIP);

for(auto point : m\_points)

{

glColor4f((double) (rand()) / RAND\_MAX,

(double) (rand()) / RAND\_MAX,

(double) (rand()) / RAND\_MAX,

(double) (rand()) / RAND\_MAX);

glVertex3f(point.x, point.y, point.z);

}

glEnd();

break;

};

case 9:{

glLineWidth(5);

glBegin(GL\_POLYGON);

for(auto point : m\_points)

{

glColor4f((double) (rand()) / RAND\_MAX,

(double) (rand()) / RAND\_MAX,

(double) (rand()) / RAND\_MAX,

(double) (rand()) / RAND\_MAX);

glVertex3f(point.x, point.y, point.z);

}

glEnd();

break;

};

default:

break;

}

}

void GL\_Widget::**startDrawing**(){

glViewport(0, 0, this->width(), this->height());

figuresGL();

}

void GL\_Widget::**setPrimitive**(int p){

this->m\_primitiveIndex = p;

this->*updateGL*();

}

**mainwindow.h**

**#ifndef MAINWINDOW\_H**

#define MAINWINDOW\_H

#include <QMainWindow>

namespace Ui

{

class MainWindow;

}

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

explicit MainWindow(QWidget \*parent = 0);

~***MainWindow***();

private:

Ui::MainWindow \*ui;

};

#endif // MAINWINDOW\_H

**mainwindow.cpp**

#include "mainwindow.h"

#include "ui\_mainwindow.h"

MainWindow::MainWindow(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::MainWindow)

{

ui->setupUi(this);

ui->taskComboBox->addItem("GL\_POINTS");

ui->taskComboBox->addItem("GL\_LINES");

ui->taskComboBox->addItem("GL\_LINE\_STRIP");

ui->taskComboBox->addItem("GL\_LINE\_LOOP");

ui->taskComboBox->addItem("GL\_TRIANGLES");

ui->taskComboBox->addItem("GL\_TRIANGLE\_STRIP");

ui->taskComboBox->addItem("GL\_TRIANGLE\_FAN");

ui->taskComboBox->addItem("GL\_QUADS");

ui->taskComboBox->addItem("GL\_QUAD\_STRIP");

ui->taskComboBox->addItem("GL\_POLYGON");

QObject::connect(ui->taskComboBox, SIGNAL(activated(int)), ui->glWidget, SLOT(setPrimitive(int)));

}

MainWindow::~MainWindow()

{

delete ui;

}

**main.cpp**

#include "mainwindow.h"

#include <QApplication>

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

{

QApplication a(argc, argv);

MainWindow w;

w.show();

return a.exec();

}

Вывод

В результате выполнения лабораторной работы разработана программа, создающая графические примитивы OpenGL. Программа работает корректно. При выполнении работы приобретены навыки работы с примитивами в графической библиотекой OpenGL.