

МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

Лабораторная работа № 1
"Примитивы OpenGL"

Студент гр. 5383

Лянгузов А.А.

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 . Рисуются 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 $(N - 1)$ отрезков.

`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.

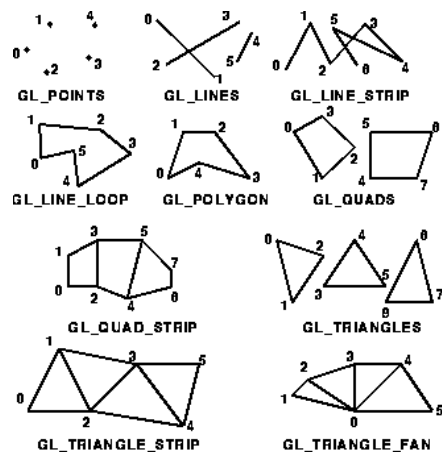
`GL_TRIANGLE_STRIP` - в этом режиме рисуется группа связанных треугольников, имеющих общую грань. Первые три вершины определяют первый треугольник, вторая, третья и четвертая – второй и т.д. для нечетного n вершины n , $(n+1)$ и $(n+2)$ определяют треугольник n . Для четного n треугольник определяют 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.

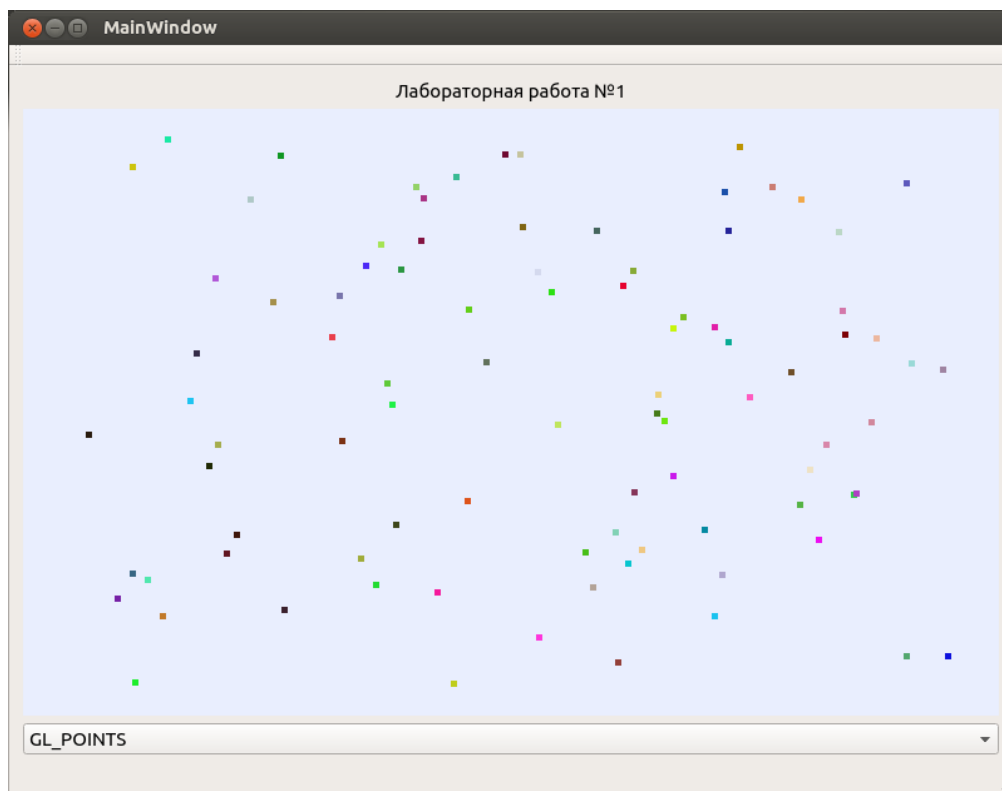
GL_POLYGON – задает многоугольник. При этом число вершин равно числу вершин рисуемого многоугольника.



Тестирование

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

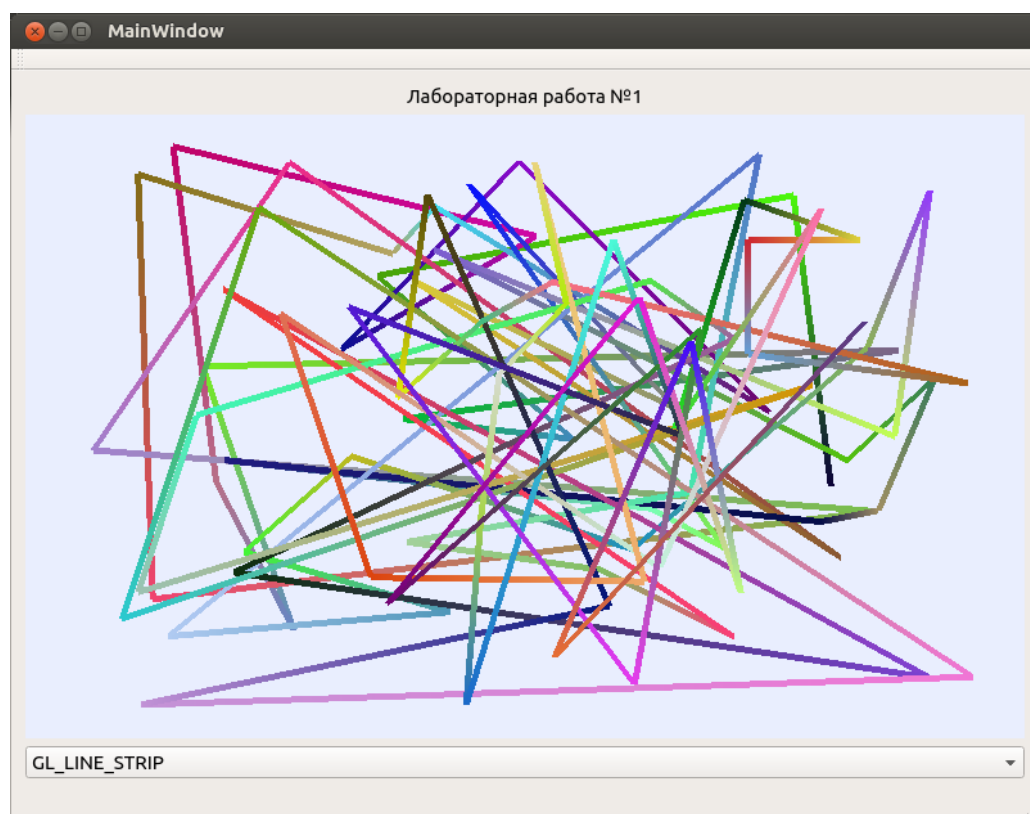
GL_POINTS



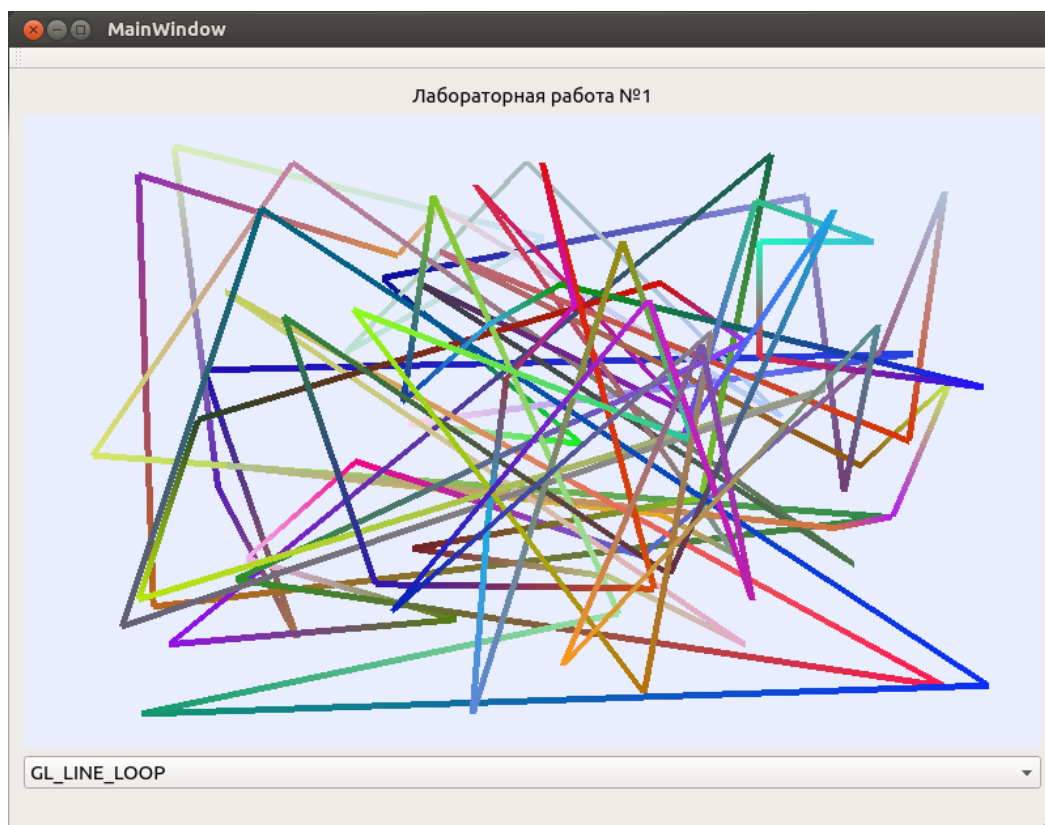
GL_LINES



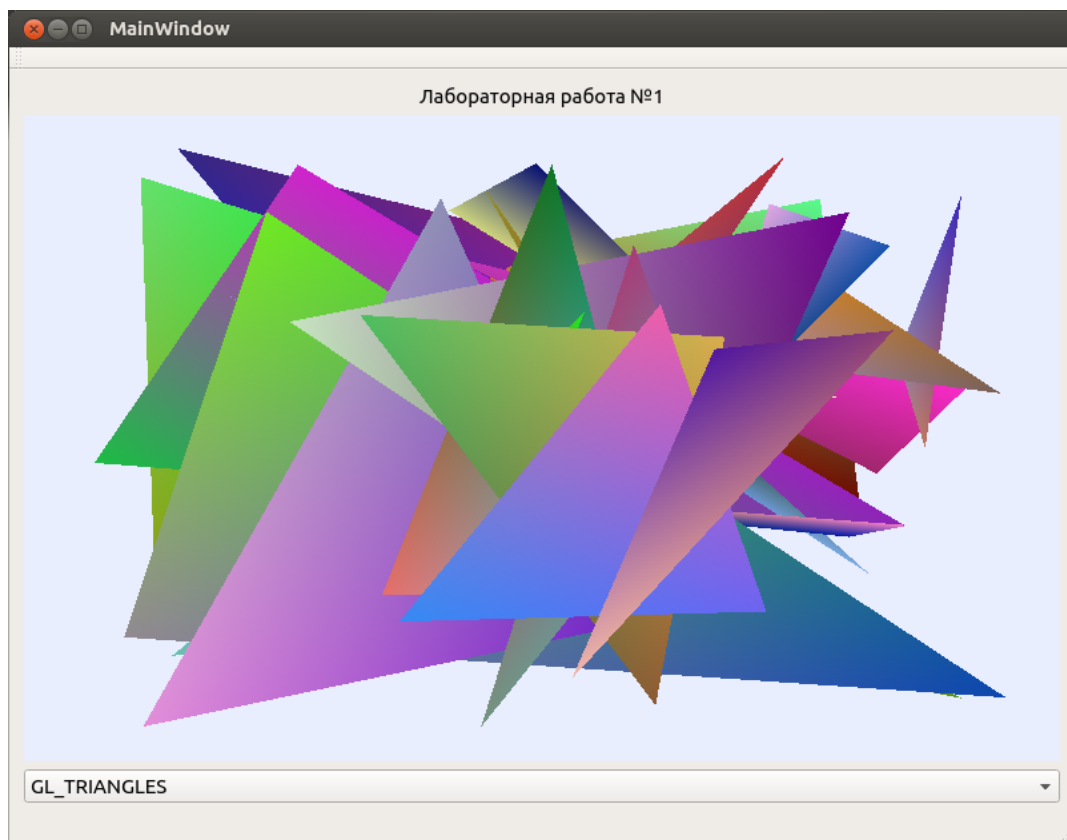
GL_LINE_STRIP



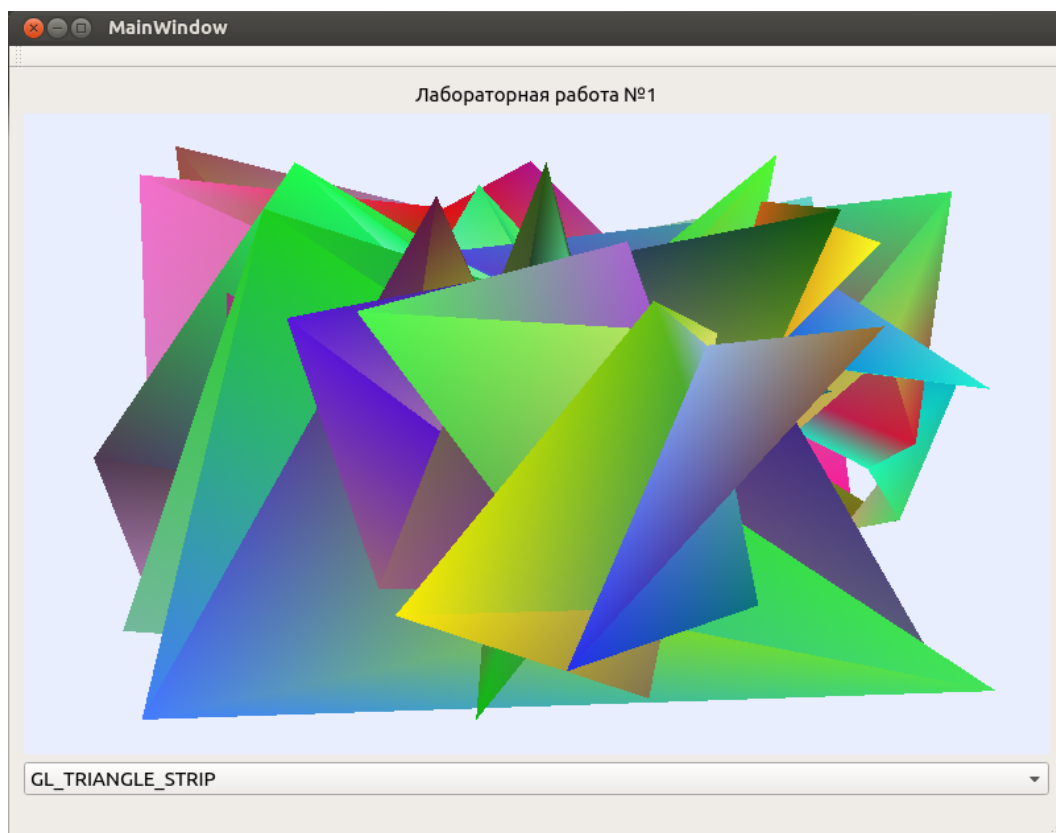
GL_LINE_LOOP



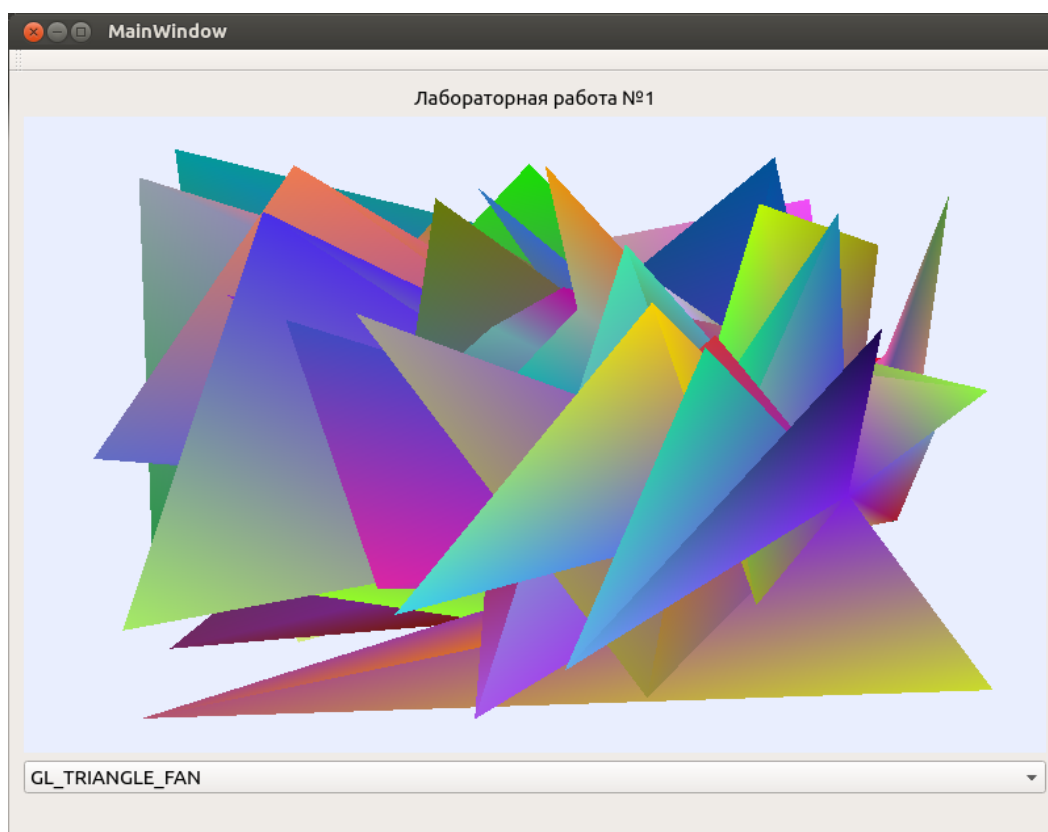
GL_TRIANGLES



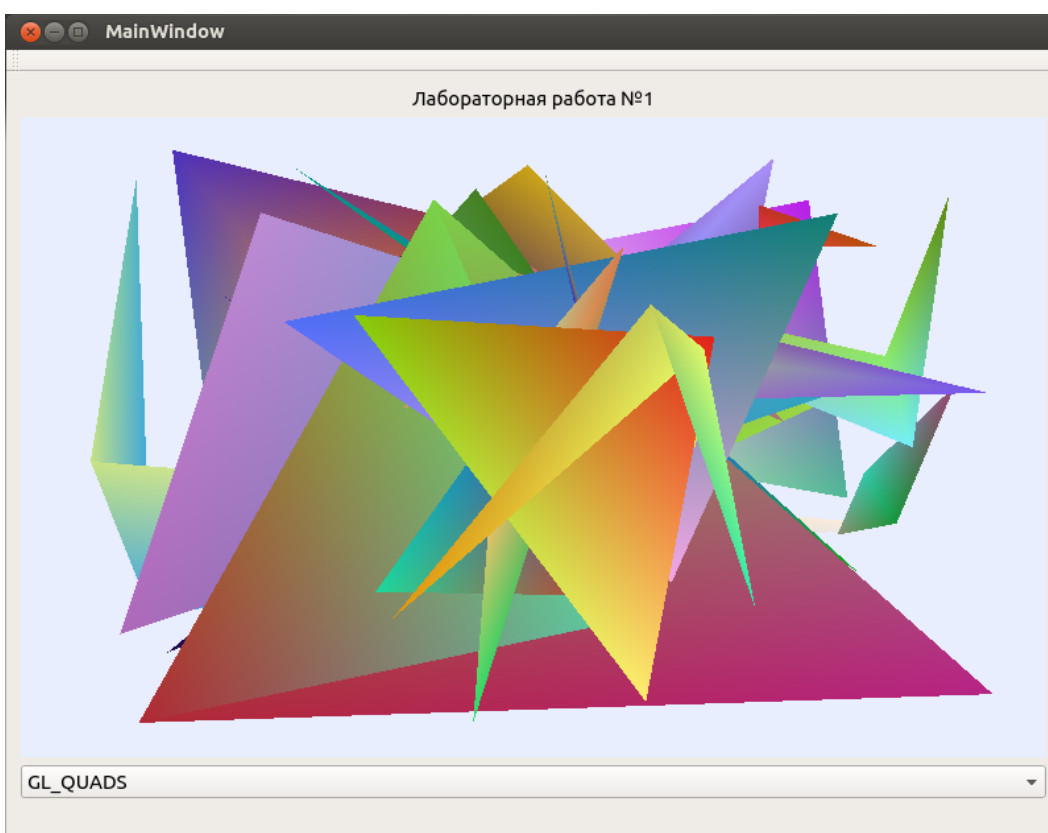
GL_TRIANGLE_STRIP



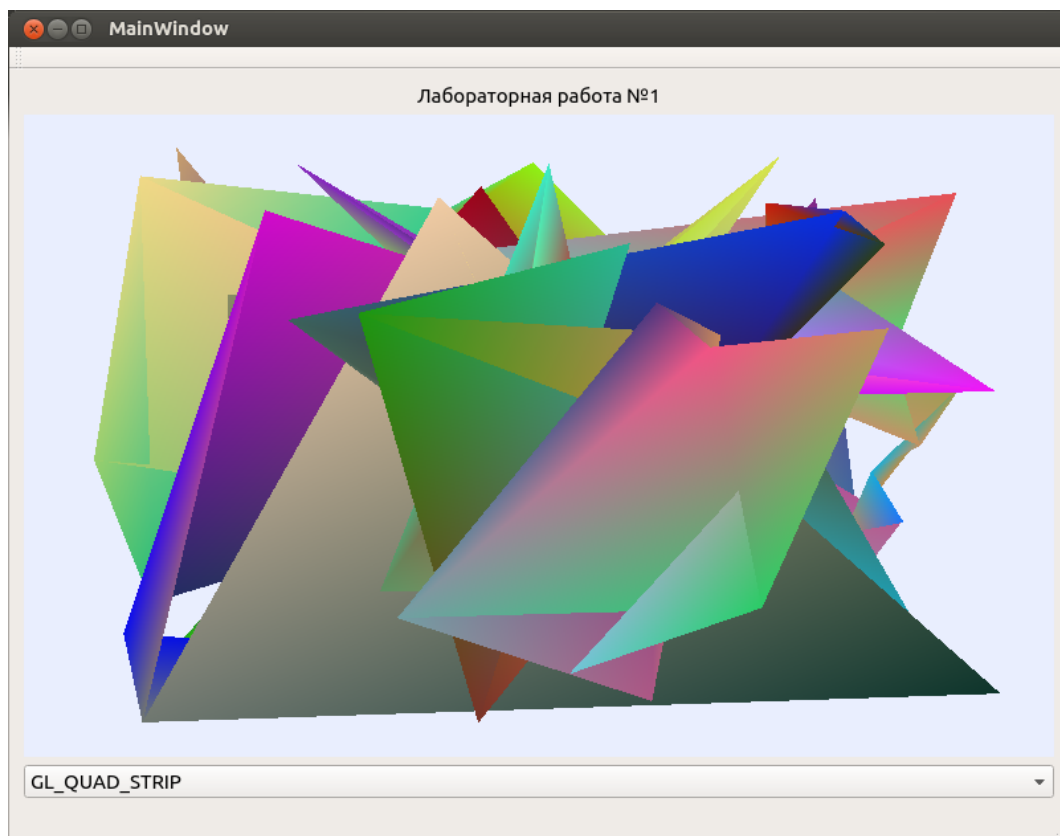
GL_TRIANGLE_FAN



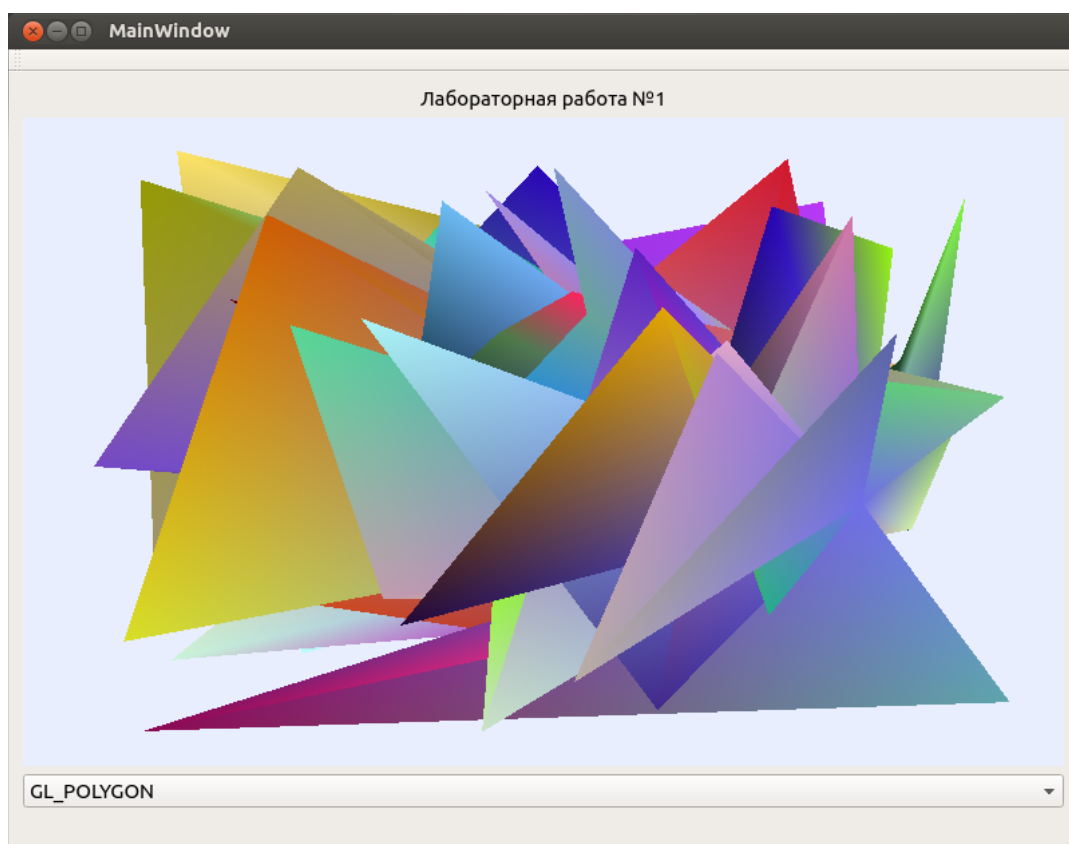
GL_QUADS



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;
    //setGeometry(20, 20, 550, 500);
    int pointsCount = 90;
    double floor = -0.9;
    double up = 0.9;
    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(0.913, 0.933, 0.996, 1);
}
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 ppoint : 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->typesComboBox->addItem("GL_POINTS");
    ui->typesComboBox->addItem("GL_LINES");
    ui->typesComboBox->addItem("GL_LINE_STRIP");
    ui->typesComboBox->addItem("GL_LINE_LOOP");
    ui->typesComboBox->addItem("GL_TRIANGLES");
    ui->typesComboBox->addItem("GL_TRIANGLE_STRIP");
    ui->typesComboBox->addItem("GL_TRIANGLE_FAN");
    ui->typesComboBox->addItem("GL_QUADS");
    ui->typesComboBox->addItem("GL_QUAD_STRIP");
    ui->typesComboBox->addItem("GL_POLYGON");
    QObject::connect(ui->typesComboBox, SIGNAL(activated(int)), ui->fieldGLWidget, SLOT(setPrimitive(int)));
}
MainWindow::~MainWindow()
{
    delete ui;
}

```

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.