МИНИСТЕРСТВА НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ

УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»

(БГТУ им В. Г. Шухова)

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

Лабораторная работа №6

Дисциплина: Компьютерная графика

Тема: «Знакомство с библиотекой OpenGL»

Выполнил: ст. группы ПВ-31

Донцов А.А.

Проверил: Осипов О. В.

Белгород, 2019

**Цель работы**

Изучение функций OpenGL для построения трёхмерных объектов в среде Qt Creator

**Задание**

1. Изучить функции OpenGL для задания формата вершин, цветов, векторов нормалей; выполнения матричных преобразований.

2. Выбрать и согласовать с преподавателем предметную область для дальнейшего выполнения лабораторных работ. Примеры предметных областей:

2.1. Сцена для игры (настольная игра, лего, кубики или др.);

2.2. Трёхмерные модели реальных объектов различной величины;

2.3. Простейшая трёхмерная компьютерная игра (симулятор гонок, стратегия или другой жанр);

2.4. Моделирование физического процесса (поверхность воды, распространение волн, столкновение упругих шаров);

2.5. Визуализация научных вычислений (графики в трёхмерном пространстве, тепловые карты);

2.6. Геометрические задачи в OpenGL, например:

2.6.1. Сортировка прозрачных объектов по глубине,

2.6.2. Нахождение пересечения селектирующего луча с объектами на сцене,

2.6.3. Отсечение невидимых объектов с использованием пирамиды видимости;

2.7. Изучение физического движка Havok Physics.

3. Разработать алгоритм и написать программу для создания трёхмерной сцены, содержащей графические объекты, в соответствии с выбранной предметной областью.

4. Разрешается написание одной сложной программы в группе из нескольких человек (не более трёх) по согласованию с преподавателем. Программа в данном случае должна быть разбита на модули, каждый из которых разрабатывает отдельный программист. Например, один программист создаёт эффекты (освещение, текстурирование), второй занимается видовыми преобразованиями и проектированием, третий – создаёт модели.

**Выполнение работы**

**Main.cpp**

#include <QApplication>

#include "widget.h"

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

{

QGuiApplication::setAttribute(Qt::AA\_EnableHighDpiScaling);

QApplication a(argc, argv);

Widget w;

w.showFullScreen();

w.show();

return a.exec();

}

**Widget.h**

#ifndef WIDGET\_H

#define WIDGET\_H

#include <QWidget>

#include <QtGui>

#include <QGLWidget>

#include <gl/glu.h>

class Widget : public QGLWidget

{

public:

Widget(QWidget \*parent = 0);

float angle = 0.0;

float lx = 0.0f, lz = 1.0f, ly = 0.0f;

float x = 0.0f, y = 0.0f, z = 0.0f;

void initializeGL();

void resizeGL(int nWidth, int nHeight);

void paintGL();

GLuint textures[5];

void drawRoom();

void drawTree();

void LoadTexture();

QPointF previousPoint;

qreal dw, dh;

qreal step = 0.1f;

void keyPressEvent(QKeyEvent \*event);

void mouseMoveEvent(QMouseEvent \*event);

void mousePressEvent(QMouseEvent \*event);

qreal light\_y;

int type;

protected:

int id\_timer;

void timerEvent(QTimerEvent \*event);

};

#endif // WIDGET\_H**Widget.cpp**

#include "widget.h"

Widget::Widget(QWidget \*parent)

: QGLWidget(parent)

{

resize(600, 600);

id\_timer=startTimer(25);

//setMouseTracking(true);

}

void Widget::*initializeGL*()

{

glClearColor(1, 1, 1, 1); // заполняем экран белым цветом

LoadTexture();

glEnable(GL\_DEPTH\_TEST); // задаем глубину проверки пикселей

glShadeModel(GL\_SMOOTH); // убираем режим сглаживания цветов

glEnable(GL\_CULL\_FACE); // говорим, что будем строить только внешние поверхности

glPolygonMode(GL\_FRONT\_AND\_BACK,GL\_FILL); // фигуры будут закрашены с обеих сторон

}

void Widget::*resizeGL*(int nWidth, int nHeight)

{

glViewport(0, 0, nWidth, nHeight); // установка точки обзора

glMatrixMode(GL\_PROJECTION); // установка режима матрицы

glLoadIdentity(); // загрузка матрицы

gluPerspective( 90.0f, (GLfloat)nWidth/(GLfloat)nHeight, 0.001f, 1000.0f );

}

void Widget::*paintGL*() // рисование

{

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT); // очистка экрана

glMatrixMode(GL\_MODELVIEW); // задаем модельно-видовую матрицу

glLoadIdentity(); // загрузка единичную матрицу

//glRotatef(dw, 0, 1, 0);

if (x >= 1)

x = 0.99;

else if (x <= -1)

x = -0.99;

if (z >= 2)

z = 1.99;

else if (z <= -1)

z = -0.99;

gluLookAt(x, y, z,

x+lx, y+ly, z+lz,

0.0f, 1.0f, 0.0f );

drawRoom();

drawTree();

}

void Widget::drawRoom()

{

//glEnable(GL\_TEXTURE\_2D);

glTranslatef(0.0f,0.0f,-1.0f);

glColor3f(0, 0, 0);

// Плато

glBegin(GL\_QUADS);

glVertex3f(-10, -0.51, -10);

glVertex3f(-10, -0.51, 10);

glVertex3f(10, -0.51, 10);

glVertex3f(10, -0.51, -10);

glEnd();

glColor3f(0, 0, 1); // задаем цвет

// Стены без окна

//glBindTexture(GL\_TEXTURE\_2D, textures[0]);

glBegin(GL\_QUADS);

glTexCoord2d(0, 0); glVertex3f(-1, 0.5, 0.0);

glTexCoord2d(0, 1); glVertex3f(-1, 0.5, 3);

glTexCoord2d(1, 1); glVertex3f(-1, -0.5, 3);

glTexCoord2d(1, 0); glVertex3f(-1, -0.5, 0.0);

glEnd();

glBegin(GL\_QUADS);

glVertex3f(1, 0.5, 3);

glVertex3f(1, 0.5, 0);

glVertex3f(1, -0.5, 0);

glVertex3f(1, -0.5, 3);

glEnd();

glBegin(GL\_QUADS);

glVertex3f(1, 0.5, 3);

glVertex3f(1, -0.5, 3);

glVertex3f(-1, -0.5, 3);

glVertex3f(-1, 0.5, 3);

glEnd();

// Стена с окном

glBegin(GL\_QUADS);

glTexCoord2d(1, 1); glVertex3f(1, 0.5, 0);

glTexCoord2d(0, 1); glVertex3f(0.5, 0.3, 0);

glTexCoord2d(0, 0); glVertex3f(0.5, -0.3, 0);

glTexCoord2d(1, 0); glVertex3f(1, -0.5, 0);

glEnd();

glBegin(GL\_QUADS);

glVertex3f(-1, 0.5, 0);

glVertex3f(-1, -0.5, 0);

glVertex3f(-0.5, -0.3, 0);

glVertex3f(-0.5, 0.3, 0);

glEnd();

glBegin(GL\_QUADS);

glVertex3f(-1, 0.5, 0);

glVertex3f(-0.5, 0.3, 0);

glVertex3f(0.5, 0.3, 0);

glVertex3f(1, 0.5, 0);

glEnd();

glBegin(GL\_QUADS);

glVertex3f(-1, -0.5, 0);

glVertex3f(1, -0.5, 0);

glVertex3f(0.5, -0.3, 0);

glVertex3f(-0.5, -0.3, 0);

glEnd();

// Пол

glColor3f(1, 0, 0); // задаем цвет

glBegin(GL\_QUADS);

glVertex3f(-1, -0.5, 0);

glVertex3f(-1, -0.5, 3);

glVertex3f(1, -0.5, 3);

glVertex3f(1, -0.5, 0);

glEnd();

// Потолок

glColor3f(1, 0, 1); // задаем цвет

glBegin(GL\_QUADS);

glVertex3f(-1, 0.5, 0);

glVertex3f(1, 0.5, 0);

glVertex3f(1, 0.5, 3);

glVertex3f(-1, 0.5, 3);

glEnd();

// Картина

glColor3f(1, 1, 1); // задаем цвет

glBegin(GL\_QUADS);

glVertex3f(0, 0.4, 2.99);

glVertex3f(0, 0, 2.99);

glVertex3f(-0.32, 0, 2.99);

glVertex3f(-0.32, 0.4, 2.99);

glEnd();

}

void Widget::drawTree()

{

// Верхий уровень

glColor3f(0.44, 0.66, 0.5);

glBegin(GL\_TRIANGLES);

glVertex3f(0.5+0.45\*0.5, -0.5, 2.5 + 0.45\*sqrt(3)\*0.5);

glVertex3f(0.5, -0.5, 2.5);

glVertex3f(0.5, 0.45, 2.5);

glEnd();

glBegin(GL\_TRIANGLES);

glVertex3f(0.5-0.45, -0.5, 2.5);

glVertex3f(0.5, 0.45, 2.5);

glVertex3f(0.5, -0.5, 2.5);

glEnd();

glBegin(GL\_TRIANGLES);

glVertex3f(0.5+0.45\*0.5, -0.5, 2.5 - 0.45\*sqrt(3)\*0.5);

glVertex3f(0.5, -0.5, 2.5);

glVertex3f(0.5, 0.45, 2.5);

glEnd();

glBegin(GL\_TRIANGLES);

glVertex3f(0.5+0.45\*0.5, -0.5, 2.5 + 0.45\*sqrt(3)\*0.5);

glVertex3f(0.5, 0.45, 2.5);

glVertex3f(0.5, -0.5, 2.5);

glEnd();

glBegin(GL\_TRIANGLES);

glVertex3f(0.5-0.45, -0.5, 2.5);

glVertex3f(0.5, -0.5, 2.5);

glVertex3f(0.5, 0.45, 2.5);

glEnd();

glBegin(GL\_TRIANGLES);

glVertex3f(0.5+0.45\*0.5, -0.5, 2.5 - 0.45\*sqrt(3)\*0.5);

glVertex3f(0.5, 0.45, 2.5);

glVertex3f(0.5, -0.5, 2.5);

glEnd();

// Средний уровень

glColor3f(0.32, 0.54, 0.39);

glBegin(GL\_TRIANGLES);

glVertex3f(0.5+0.45\*sqrt(3)\*0.5, -0.5, 2.5 + 0.45\*0.5);

glVertex3f(0.5, -0.5, 2.5);

glVertex3f(0.5, 0.45, 2.5);

glEnd();

glBegin(GL\_TRIANGLES);

glVertex3f(0.5-0.45\*sqrt(3)\*0.5, -0.5, 2.5 + 0.45\*0.5);

glVertex3f(0.5, 0.45, 2.5);

glVertex3f(0.5, -0.5, 2.5);

glEnd();

glBegin(GL\_TRIANGLES);

glVertex3f(0.5, -0.5, 2.5 - 0.45);

glVertex3f(0.5, -0.5, 2.5);

glVertex3f(0.5, 0.45, 2.5);

glEnd();

glBegin(GL\_TRIANGLES);

glVertex3f(0.5+0.45\*sqrt(3)\*0.5, -0.5, 2.5 + 0.45\*0.5);

glVertex3f(0.5, 0.45, 2.5);

glVertex3f(0.5, -0.5, 2.5);

glEnd();

glBegin(GL\_TRIANGLES);

glVertex3f(0.5-0.45\*sqrt(3)\*0.5, -0.5, 2.5 + 0.45\*0.5);

glVertex3f(0.5, -0.5, 2.5);

glVertex3f(0.5, 0.45, 2.5);

glEnd();

glBegin(GL\_TRIANGLES);

glVertex3f(0.5, -0.5, 2.5 - 0.45);

glVertex3f(0.5, 0.45, 2.5);

glVertex3f(0.5, -0.5, 2.5);

glEnd();

// Нижний уровень

glColor3f(0.24, 0.4, 0.29);

glBegin(GL\_TRIANGLES);

glVertex3f(0.5+0.45, -0.5, 2.5);

glVertex3f(0.5, -0.5, 2.5);

glVertex3f(0.5, 0.45, 2.5);

glEnd();

glBegin(GL\_TRIANGLES);

glVertex3f(0.5-0.45\*0.5, -0.5, 2.5 + 0.45\*sqrt(3)\*0.5);

glVertex3f(0.5, 0.45, 2.5);

glVertex3f(0.5, -0.5, 2.5);

glEnd();

glBegin(GL\_TRIANGLES);

glVertex3f(0.5-0.45\*0.5, -0.5, 2.5 - 0.45\*sqrt(3)\*0.5);

glVertex3f(0.5, 0.45, 2.5);

glVertex3f(0.5, -0.5, 2.5);

glEnd();

glBegin(GL\_TRIANGLES);

glVertex3f(0.5+0.45, -0.5, 2.5);

glVertex3f(0.5, 0.45, 2.5);

glVertex3f(0.5, -0.5, 2.5);

glEnd();

glBegin(GL\_TRIANGLES);

glVertex3f(0.5-0.45\*0.5, -0.5, 2.5 + 0.45\*sqrt(3)\*0.5);

glVertex3f(0.5, -0.5, 2.5);

glVertex3f(0.5, 0.45, 2.5);

glEnd();

glBegin(GL\_TRIANGLES);

glVertex3f(0.5-0.45\*0.5, -0.5, 2.5 - 0.45\*sqrt(3)\*0.5);

glVertex3f(0.5, -0.5, 2.5);

glVertex3f(0.5, 0.45, 2.5);

glEnd();

}

void Widget::LoadTexture()

{

glGenTextures(5, textures);

QImage walls;

walls.load(QDir::currentPath() + "\\walls.jpg");

walls = QGLWidget::convertToGLFormat(walls);

glBindTexture(GL\_TEXTURE\_2D, textures[0]);

glTexImage2D(GL\_TEXTURE\_2D, 0, 3, (GLsizei)walls.width(), (GLsizei)walls.height(), 0, GL\_RGBA, GL\_UNSIGNED\_BYTE, walls.bits());

glTexEnvi(GL\_TEXTURE\_ENV, GL\_TEXTURE\_ENV\_MODE, GL\_MODULATE);

glTexEnvi(GL\_TEXTURE\_ENV, GL\_TEXTURE\_ENV\_MODE, GL\_DECAL);

glTexParameterf(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR);

glTexParameterf(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR);

QImage el;

el.load(QDir::currentPath() + "\\el.jpg");

el = QGLWidget::convertToGLFormat(el);

glBindTexture(GL\_TEXTURE\_2D, textures[1]);

glTexImage2D(GL\_TEXTURE\_2D, 0, 3, (GLsizei)el.width(), (GLsizei)el.height(), 0, GL\_RGBA, GL\_UNSIGNED\_BYTE, el.bits());

glTexEnvi(GL\_TEXTURE\_ENV, GL\_TEXTURE\_ENV\_MODE, GL\_MODULATE);

glTexEnvi(GL\_TEXTURE\_ENV, GL\_TEXTURE\_ENV\_MODE, GL\_DECAL);

glTexParameterf(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR);

glTexParameterf(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR);

QImage pol;

pol.load(QDir::currentPath() + "\\pol.jpg");

pol = QGLWidget::convertToGLFormat(pol);

glBindTexture(GL\_TEXTURE\_2D, textures[2]);

glTexImage2D(GL\_TEXTURE\_2D, 0, 3, (GLsizei)pol.width(), (GLsizei)pol.height(), 0, GL\_RGBA, GL\_UNSIGNED\_BYTE, pol.bits());

glTexEnvi(GL\_TEXTURE\_ENV, GL\_TEXTURE\_ENV\_MODE, GL\_MODULATE);

glTexEnvi(GL\_TEXTURE\_ENV, GL\_TEXTURE\_ENV\_MODE, GL\_DECAL);

glTexParameterf(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR);

glTexParameterf(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR);

QImage pot;

pot.load(QDir::currentPath() + "\\potolok.jpg");

pot = QGLWidget::convertToGLFormat(pot);

glBindTexture(GL\_TEXTURE\_2D, textures[3]);

glTexImage2D(GL\_TEXTURE\_2D, 0, 3, (GLsizei)pot.width(), (GLsizei)pot.height(), 0, GL\_RGBA, GL\_UNSIGNED\_BYTE, pot.bits());

glTexEnvi(GL\_TEXTURE\_ENV, GL\_TEXTURE\_ENV\_MODE, GL\_MODULATE);

glTexEnvi(GL\_TEXTURE\_ENV, GL\_TEXTURE\_ENV\_MODE, GL\_DECAL);

glTexParameterf(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR);

glTexParameterf(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR);

QImage pic;

pic.load(QDir::currentPath() + "\\new\_year.jpg");

pic = QGLWidget::convertToGLFormat(pic);

glBindTexture(GL\_TEXTURE\_2D, textures[4]);

glTexImage2D(GL\_TEXTURE\_2D, 0, 3, (GLsizei)pic.width(), (GLsizei)pic.height(), 0, GL\_RGBA, GL\_UNSIGNED\_BYTE, pic.bits());

glTexEnvi(GL\_TEXTURE\_ENV, GL\_TEXTURE\_ENV\_MODE, GL\_MODULATE);

glTexEnvi(GL\_TEXTURE\_ENV, GL\_TEXTURE\_ENV\_MODE, GL\_DECAL);

glTexParameterf(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR);

glTexParameterf(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR);}

void Widget::*keyPressEvent*(QKeyEvent \*event) {

switch (event->key()) {

case Qt::Key\_W:

x += lx \* step;

z += lz \* step;

break;

case Qt::Key\_S:

x -= lx \* step;

z -= lz \* step;

break;

}

//*updateGL*(); // обновление изображения

}

void Widget::*mousePressEvent*(QMouseEvent \*event)

{

previousPoint = event->windowPos();

}

void Widget::*mouseMoveEvent*(QMouseEvent \*event)

{

dh += (event->y() - previousPoint.y()) / 1000;

dw += (event->x() - previousPoint.x()) / 1000;

lx = -sin(dw);

ly = -tan(dh);

lz = +cos(dw);

//Обновление данных о предыдущей координате

previousPoint = event->windowPos();

//*updateGL*(); // обновление изображения

}

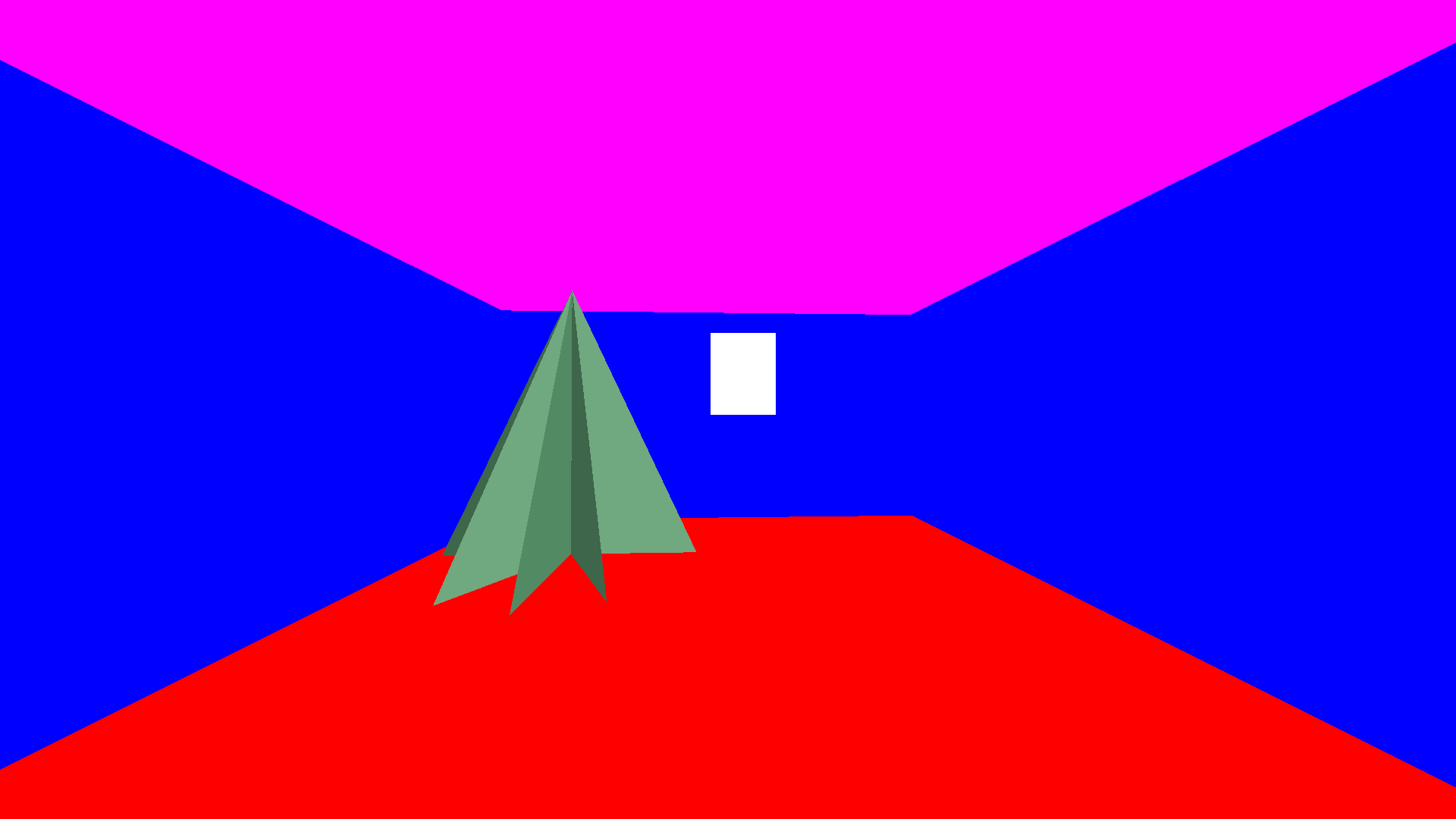
void Widget::timerEvent(QTimerEvent \*event)

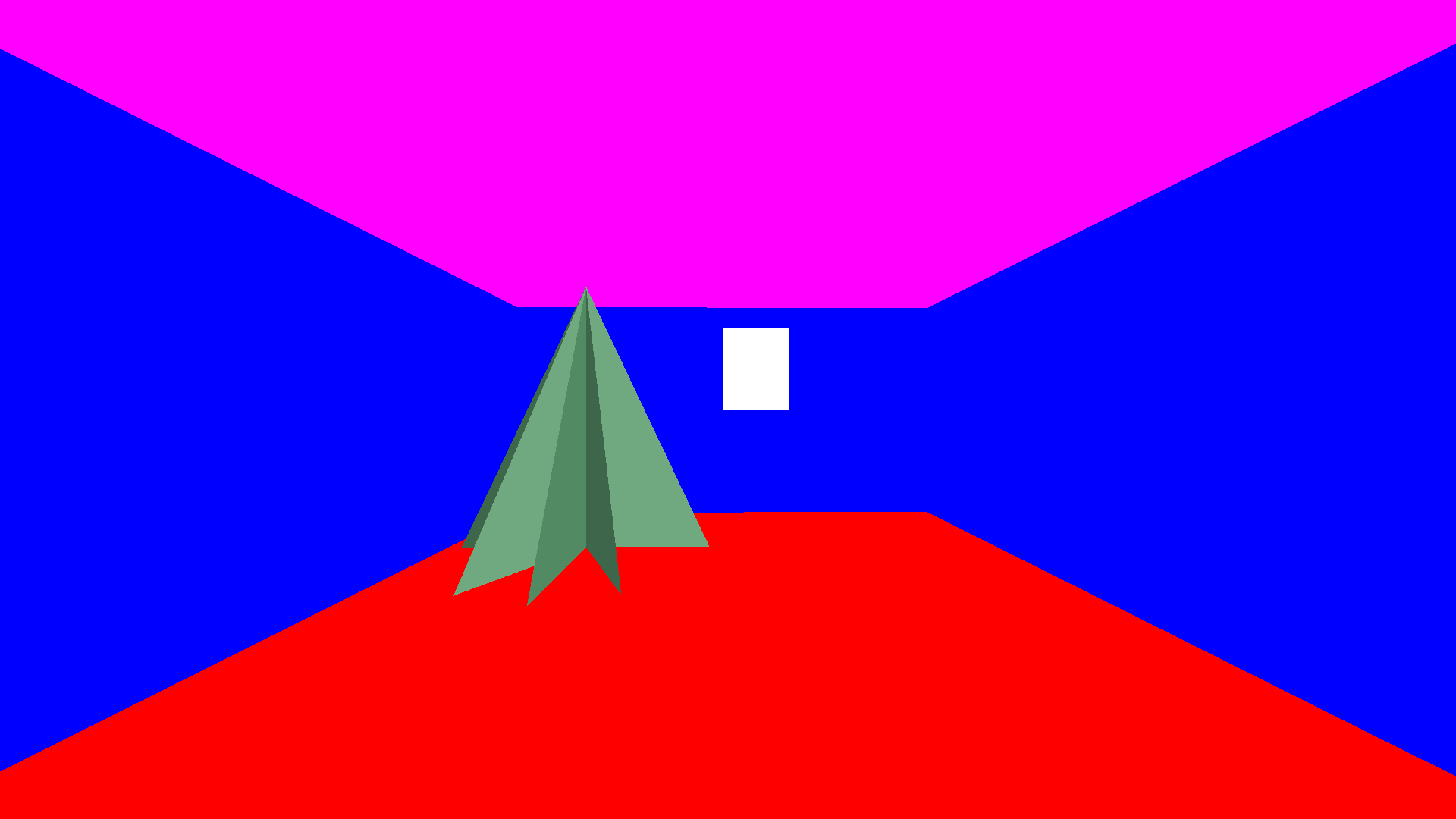
{

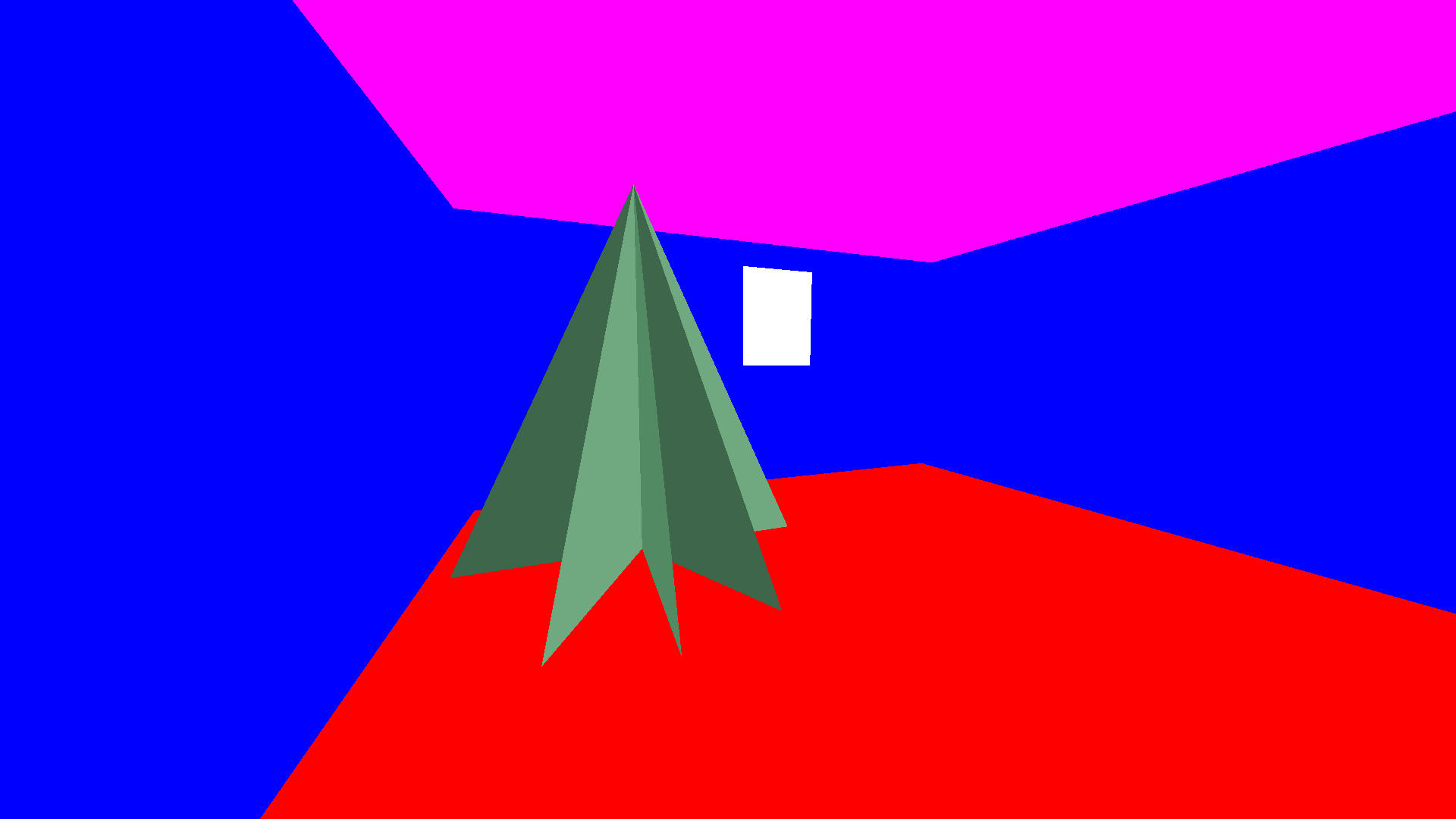
updateGL();

}

**Скриншот результата работы программы**:







**Вывод**: выполнение данной лабораторной работы помогло изучить функции OpenGL для построения трёхмерных объектов в среде Qt Creator.