МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ   
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

ИНСТИТУТ НЕПРЕРЫВНОГО И ДИСТАНЦИОННОГО ОБРАЗОВАНИЯ

|  |
| --- |
| КАФЕДРА компьютерных технологий и программной инженерии |

ОЦЕНКА

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ |
| «РАБОТА С ТРЕХМЕРНЫМИ ОБЪЕКТАМИ» |
| по дисциплине: Компьютерная графика |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. № | Z9431 |  |  |  | Андреев Д.И. |
|  | номер группы |  | подпись, дата |  | инициалы, фамилия |
| Студенческий билет № | 2019/3781 | |  |  |  |

Санкт-Петербург 2020

Оглавление

[1. Задание 3](#_Toc55759175)

[2. Листинг программы 3](#_Toc55759176)

[3. Выводы 12](#_Toc55759177)

1. Задание

Создать приложение, выводящее объемный объект средствами Open GL. Среда программирования – QT Creator c MinGW и Open GL .

1. Листинг программы

Файл main.cpp

#include <QGuiApplication>

#include <QSurfaceFormat>

#include "mainwindow.h"

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

{

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

QGuiApplication app(*argc*, argv);

MainWindow win;

win.open();

*return* app.exec();

}

Файл mainwindow.h – содержит класс главного окна программы

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QObject>

*class* QuickItem;

*class* QQuickView;

*class* QQmlEngine;

*class* MainWindow : *public* QObject

{

Q\_OBJECT

*public*:

*explicit* MainWindow(QObject \*parent = *nullptr*);

*public* *slots*:

void open();

*private*:

QQuickView \*m\_view;

QuickItem \*m\_item;

QQmlEngine \*m\_engine;

};

#endif *//* *MAINWINDOW\_H*

Файл mainwindow.cpp – содержит реализацию методов класса MainWindow

#include "mainwindow.h"

#include "quickitem.h"

#include "gllight.h"

#include <QQuickView>

#include <QQmlEngine>

#include <QGuiApplication>

MainWindow::**MainWindow**(QObject \*parent) :

QObject(parent)

, m\_engine(*new* QQmlEngine)

{

qmlRegisterType<QuickItem>("opengl\_lib", 1, 0, "QuickItem");

m\_view = *new* QQuickView(m\_engine, *nullptr*);

m\_view->setResizeMode(QQuickView::*SizeRootObjectToView*);

m\_view->setSource(QUrl("qrc:/main.qml"));

m\_item = m\_view->rootObject()->findChild<QuickItem \*>();

}

void MainWindow::**open**()

{

m\_view->show();

m\_item->update();

}

Файл quickitem.h – содержит класс 3D объекта

#ifndef QUICKITEM\_H

#define QUICKITEM\_H

#include <QQuickItem>

#include <QtGui/QOpenGLFunctions\_3\_3\_Core>

#include <QQuickWindow>

*class* **GLCamera**;

*class* **GLLight**;

*class* **QSGSimpleTextureNode**;

*class* **QOpenGLTexture**;

*class* **QOpenGLShaderProgram**;

*class* **QuickItem** : *public* QQuickItem

{

Q\_OBJECT

*struct* **MainUniforms** {

GLint modelLoc;

GLint viewLoc;

GLint projLoc;

};

*public*:

**QuickItem**(QQuickItem \*parent = *nullptr*);

~***QuickItem***();

*protected*:

*virtual* QSGNode \****updatePaintNode***(QSGNode \*pNode, UpdatePaintNodeData \*pData) Q\_DECL\_OVERRIDE;

*private*:

void **initFBO**(int width, int height, QSGSimpleTextureNode \*node);

void **doRender**(QSGSimpleTextureNode \*pTextureNode);

void **pushOpenGLMatrix**();

void **initShader**();

void **initOpenGLState**();

QOpenGLFramebufferObject \*m\_pSourceFbo;

QOpenGLFramebufferObject \*m\_pTargetFbo;

QOpenGLShaderProgram \*m\_program;

GLCamera \*m\_camera;

QOpenGLFunctions\_3\_3\_Core m\_glFuncts;

bool m\_isOpenGLInitialized;

GLuint m\_VBO, m\_VAO;

MainUniforms m\_mainUn;

GLfloat m\_verts[118]{

-0.5f, -0.5f, -0.5f,

0.5f, -0.5f, -0.5f,

0.5f, 0.5f, -0.5f,

0.5f, 0.5f, -0.5f,

-0.5f, 0.5f, -0.5f,

-0.5f, -0.5f, -0.5f,

-0.5f, -0.5f, 0.5f,

0.5f, -0.5f, 0.5f,

0.5f, 0.5f, 0.5f,

0.5f, 0.5f, 0.5f,

-0.5f, 0.5f, 0.5f,

-0.5f, -0.5f, 0.5f,

-0.5f, 0.5f, 0.5f,

-0.5f, 0.5f, -0.5f,

-0.5f, -0.5f, -0.5f,

-0.5f, -0.5f, -0.5f,

-0.5f, -0.5f, 0.5f,

-0.5f, 0.5f, 0.5f,

0.5f, 0.5f, 0.5f,

0.5f, 0.5f, -0.5f,

0.5f, -0.5f, -0.5f,

0.5f, -0.5f, -0.5f,

0.5f, -0.5f, 0.5f,

0.5f, 0.5f, 0.5f,

-0.5f, -0.5f, -0.5f,

0.5f, -0.5f, -0.5f,

0.5f, -0.5f, 0.5f,

0.5f, -0.5f, 0.5f,

-0.5f, -0.5f, 0.5f,

-0.5f, -0.5f, -0.5f,

-0.5f, 0.5f, -0.5f,

0.5f, 0.5f, -0.5f,

0.5f, 0.5f, 0.5f,

0.5f, 0.5f, 0.5f,

-0.5f, 0.5f, 0.5f,

-0.5f, 0.5f, -0.5f,

};

};

#endif *//* *QUICKITEM\_H*

Файл quickitem.cpp – содержит реализацию методов класса объекта (в т.ч. логику отрисовки)

#include "quickitem.h"

#include <QKeyEvent>

#include <QOpenGLFramebufferObject>

#include <QSGSimpleTextureNode>

#include <QOpenGLTexture>

#include "glcamera.h"

#include "gllight.h"

#include <cmath>

QuickItem::**QuickItem**(QQuickItem \*parent) :

QQuickItem(parent)

, m\_pSourceFbo(*nullptr*)

, m\_pTargetFbo(*nullptr*)

, m\_program(*nullptr*)

, m\_camera(*new* GLCamera)

, m\_isOpenGLInitialized(*false*)

{

setFlag(QQuickItem::*ItemHasContents*);

setFocus(*true*);

}

QuickItem::~***QuickItem***()

{

}

QSGNode \*QuickItem::***updatePaintNode***(QSGNode \*pNode, QQuickItem::UpdatePaintNodeData \*pData)

{

Q\_UNUSED(pData)

QSGSimpleTextureNode\* pTextureNode = *static\_cast*<QSGSimpleTextureNode\*>(pNode);

*if* (pTextureNode == NULL)

{

pTextureNode = *new* QSGSimpleTextureNode();

pTextureNode->setTexture(*this*->window()->createTextureFromId(0, QSize(0,0)));

}

doRender(pTextureNode);

*return* pTextureNode;

}

void QuickItem::**initFBO**(int width, int height, QSGSimpleTextureNode \*pTextureNode)

{

*delete* m\_pSourceFbo;

m\_pSourceFbo = *nullptr*;

*delete* m\_pTargetFbo;

m\_pTargetFbo = *nullptr*;

*if* ((width > 0) && (height > 0)) {

QOpenGLFramebufferObjectFormat sourceFormat;

sourceFormat.setAttachment(QOpenGLFramebufferObject::*Depth*);

sourceFormat.setSamples(16);

m\_pSourceFbo= *new* QOpenGLFramebufferObject(width, height, sourceFormat);

m\_pTargetFbo= *new* QOpenGLFramebufferObject(width, height);

}

*else* {

pTextureNode->setTexture(*this*->window()->createTextureFromId(0, QSize(0,0)));

pTextureNode->setRect(*this*->*boundingRect*());

}

}

void QuickItem::**doRender**(QSGSimpleTextureNode \*pTextureNode)

{

int width= *this*->width();

int height= *this*->height();

pTextureNode->setTextureCoordinatesTransform(QSGSimpleTextureNode::*MirrorVertically*);

initFBO(width, height, pTextureNode);

initShader();

*if* (m\_pTargetFbo && m\_pTargetFbo->isValid() && m\_pSourceFbo && m\_pSourceFbo->isValid())

{

initOpenGLState();

m\_pSourceFbo->bind();

m\_glFuncts.glClearColor(0.1f, 0.1f, 0.1f, 1.0f);

m\_glFuncts.glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT | GL\_STENCIL\_BUFFER\_BIT);

m\_program->bind();

m\_camera->setCam(m\_camera->eye(),

m\_camera->eye() + m\_camera->front(),

m\_camera->up());

m\_program->setUniformValue(m\_mainUn.modelLoc, m\_camera->model());

m\_program->setUniformValue(m\_mainUn.viewLoc, m\_camera->view());

m\_program->setUniformValue(m\_mainUn.projLoc, m\_camera->projection());

m\_program->release();

m\_program->bind();

m\_glFuncts.glBindVertexArray(m\_VAO);

m\_glFuncts.glDrawArrays(GL\_TRIANGLES, 0, 36);

m\_glFuncts.glBindVertexArray(0);

m\_program->release();

m\_pSourceFbo->release();

QRect rect(0, 0, width, height);

QOpenGLFramebufferObject::blitFramebuffer(m\_pTargetFbo, rect, m\_pSourceFbo, rect);

pTextureNode->setTexture(*this*->window()->createTextureFromId(m\_pSourceFbo->texture(), m\_pSourceFbo->size()));

pTextureNode->setRect(*this*->*boundingRect*());

*delete* m\_pSourceFbo;

m\_pSourceFbo= NULL;

}

*else*

{

*delete* m\_pSourceFbo;

m\_pSourceFbo= NULL;

*delete* m\_pTargetFbo;

m\_pTargetFbo= NULL;

}

}

void QuickItem::**initShader**()

{

*if*(m\_program)

*return*;

m\_program = *new* QOpenGLShaderProgram();

QOpenGLShader m\_vert(QOpenGLShader::*Vertex*);

*if*(!m\_vert.compileSourceFile(":/mainVert.vert"))

qFatal("QuickItem::initShader: m\_program: vertex shader did not compiled");

m\_program->addShader(&m\_vert);

QOpenGLShader m\_frag(QOpenGLShader::*Fragment*);

*if*(!m\_frag.compileSourceFile(":/mainFrag.frag"))

qFatal("QuickItem::initShader: m\_program: fragment shader did not compiled");

m\_program->addShader(&m\_frag);

m\_program->*link*();

m\_program->bind();

m\_mainUn.modelLoc = m\_program->uniformLocation("model");

m\_mainUn.viewLoc = m\_program->uniformLocation("view");

m\_mainUn.projLoc = m\_program->uniformLocation("projection");

m\_program->release();

}

void QuickItem::**initOpenGLState**()

{

*if*(m\_isOpenGLInitialized)

*return*;

m\_camera->projection(45.0f, *this*->width()/*this*->height(), 0.1f, 100.0f);

m\_camera->rotate(30, m\_camera->up());

m\_camera->rotate(10, m\_camera->right());

m\_glFuncts.*initializeOpenGLFunctions*();

m\_glFuncts.glViewport(0,0, *this*->width(), *this*->height());

m\_glFuncts.glGenBuffers(1, &m\_VBO);

m\_glFuncts.glGenVertexArrays(1, &m\_VAO);

m\_glFuncts.glBindVertexArray(m\_VAO);

m\_glFuncts.glBindBuffer(GL\_ARRAY\_BUFFER, m\_VBO);

m\_glFuncts.glBufferData(GL\_ARRAY\_BUFFER, *sizeof*(m\_verts), m\_verts, GL\_STATIC\_DRAW);

m\_glFuncts.glVertexAttribPointer(0, 3, GL\_FLOAT, GL\_FALSE, 3 \* *sizeof*(GLfloat), (GLvoid\*)0);

m\_glFuncts.glEnableVertexAttribArray(0);

m\_glFuncts.glBindVertexArray(0);

m\_glFuncts.glEnable(GL\_DEPTH\_TEST);

m\_isOpenGLInitialized = *true*;

}

Файл glcamera.h – содержит вспомогательный класс, инкапсулирующий управление методы камерой.

#ifndef GLCAMERA\_H

#define GLCAMERA\_H

#include <QVector3D>

#include <QMatrix4x4>

*class* GLCamera

{

*public*:

GLCamera();

QMatrix4x4 setCam(*const* QVector3D &eye,

*const* QVector3D &target,

*const* QVector3D &right);

QVector3D eye() *const*;

QVector3D front() *const*;

QVector3D target() *const*;

QMatrix4x4 view() *const*;

QMatrix4x4 model() *const*;

QMatrix4x4 projection() *const*;

QVector3D right() *const*;

QVector3D dir() *const*;

QVector3D up() *const*;

void setEye(*const* QVector3D &eye);

void setTarget(*const* QVector3D &target);

void setUp(*const* QVector3D &up);

void setFront(*const* QVector3D &front);

void projection(float verticalAngle, float aspectRatio, float nearPlane, float farPlane);

void translate(*const* QVector3D &vec);

void rotate(float angle, *const* QVector3D &vec);

void scale(float zoom);

*private*:

QVector3D m\_eye;

QVector3D m\_up;

QVector3D m\_front;

QMatrix4x4 m\_view;

QMatrix4x4 m\_model;

QMatrix4x4 m\_projection;

};

#endif *//* *GLCAMERA\_H*

Файл glcamera.cpp – содержит реализацию методов класс GLCamera

#include "glcamera.h"

GLCamera::**GLCamera**() :

m\_eye(0.0f,0.0f,5.0f)

, m\_up(0.0f, 1.0f, 0.0f)

, m\_front(0.0f, 0.0f, -1.0f)

, m\_view()

, m\_model()

, m\_projection()

{}

QMatrix4x4 GLCamera::**setCam**(*const* QVector3D &eyeVec,

*const* QVector3D &targetVec,

*const* QVector3D &upVec)

{

setEye(eyeVec);

setFront(targetVec - m\_eye);

setUp(upVec);

QMatrix4x4 camMat;

camMat.lookAt(eye(), target(), up());

camMat.translate(-eye());

m\_view = camMat;

*return* m\_view;

}

QVector3D GLCamera::**eye**() *const*

{

*return* m\_eye;

}

void GLCamera::**setEye**(*const* QVector3D &eye)

{

*if*(eye == m\_eye)

*return*;

m\_eye = eye;

}

QVector3D GLCamera::**target**() *const*

{

*return* m\_eye + m\_front;

}

QMatrix4x4 GLCamera::**view**() *const*

{

*return* m\_view;

}

QMatrix4x4 GLCamera::**model**() *const*

{

*return* m\_model;

}

QMatrix4x4 GLCamera::**projection**() *const*

{

*return* m\_projection;

}

QVector3D GLCamera::**up**() *const*

{

*return* QVector3D::crossProduct(dir(), right());

}

void GLCamera::**setUp**(*const* QVector3D &up)

{

*if*(up == m\_up)

*return*;

m\_up = up;

}

QVector3D GLCamera::**right**() *const*

{

*return* QVector3D::crossProduct(m\_up.normalized(), dir()).normalized();

}

QVector3D GLCamera::**dir**() *const*

{

*return* QVector3D(m\_eye - target()).normalized();

}

QVector3D GLCamera::**front**() *const*

{

*return* m\_front;

}

void GLCamera::**setFront**(*const* QVector3D &front)

{

*if*(front == m\_front)

*return*;

m\_front = front;

}

void GLCamera::**translate**(*const* QVector3D &vec)

{

m\_model.translate(vec);

}

void GLCamera::**rotate**(float angle, *const* QVector3D &vec)

{

m\_model.rotate(angle, vec);

}

void GLCamera::**scale**(float zoom)

{

m\_model.scale(zoom);

}

void GLCamera::**projection**(float verticalAngle, float aspectRatio, float nearPlane, float farPlane)

{

m\_projection.perspective(verticalAngle, aspectRatio, nearPlane, farPlane);

}

Файл mainVert.vert – вершинный шейдер, накладываемый на объект

#version 330

layout (location = 0) *in* *vec3* position;

*uniform* *mat4* model;

*uniform* *mat4* view;

*uniform* *mat4* projection;

*void* main()

{

gl\_Position = projection \* view \* model \* *vec4*(position, 1.0f);

}

Файл mainFrag.frag – фрагментный шейдер, накладываемый на объект (содержит расчет освещения)

#version 330

*out* *vec4* color;

*void* main(*void*)

{

color = *vec4*(0.5,0.7,0.0, 1.0f);

}

Файл main.qml – содержит описание отображаемых объектов на языке QML (для упрощения создания главного окна и 3D объектов

*import* QtQuick 2.11

*import* QtQuick.Window 2.11

*import* QtQuick.Controls 2.11

*import* opengl\_lib 1.0

Item {

width: 640

height: 480

visible: true

QuickItem {

id: *item*

anchors.fill: *parent*

}

}

1. Выводы

В ходе лабораторной работы была разработана программа, отрисовывающая 3D объект. Как можно видеть, так как в программе не реализован расчет света отрисованный 3D объект выглядит как 2D (рис 1.).

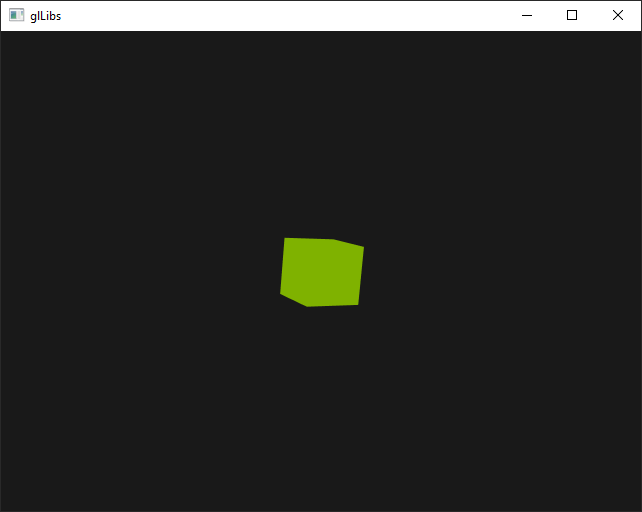


Рис. 1. Результат работы программы