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 ИСТОЧНИКОМ СВЕТА И СВОЙСТВАМИ ИСТОЧНИКОВ И МАТЕРИАЛОВ» |
| 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](#_Toc55663083)

[2. Листинг программы 3](#_Toc55663084)

[3. Выводы 21](#_Toc55663085)

1. Задание

Включить 3 источника света, задать отражающие свойства поверхностей, положение и цвет источников света. Цвет поверхностей сделать одинаковым, а отражающие свойства – разными. Вывести несколько объемных объектов. Источники света должны иметь различный цвет. Необходимо организовать вращение сцены, управляемое с клавиатуры (сцена с объектами объекты вращается, источники неподвижны).

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");

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

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

Q\_PROPERTY(float specStrength READ specStrength WRITE setSpecStrength NOTIFY specStrengthChanged)

Q\_PROPERTY(float specIntensity READ specIntensity WRITE setSpecIntensity NOTIFY specIntensityChanged)

Q\_PROPERTY(GLLight\* light\_1 READ light\_1 NOTIFY light\_1Changed)

Q\_PROPERTY(GLLight\* light\_2 READ light\_2 NOTIFY light\_2Changed)

Q\_PROPERTY(GLLight\* light\_3 READ light\_3 NOTIFY light\_3Changed)

Q\_PROPERTY(QColor objectColor READ objectColor WRITE setObjectColor NOTIFY objectColorChanged)

*struct* **LightUniforms** {

GLint modelLoc;

GLint viewLoc;

GLint projLoc;

GLint colorLoc;

};

*struct* **MainUniforms** {

GLint modelLoc;

GLint viewLoc;

GLint projLoc;

GLint objectColorLoc;

GLint lightPosLoc\_1;

GLint lightPosLoc\_2;

GLint lightPosLoc\_3;

GLint viewPosLoc;

GLint scepStrengthLoc;

GLint scepIntensityLoc;

GLint lightColorLoc\_1;

GLint lightColorLoc\_2;

GLint lightColorLoc\_3;

GLint lightCount;

};

*public*:

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

~***QuickItem***();

float **specStrength**() *const*;

float **specIntensity**() *const*;

GLLight \***light\_1**() *const*;

GLLight \***light\_2**() *const*;

GLLight \***light\_3**() *const*;

QColor **objectColor**() *const*;

*protected*:

*virtual* void ***keyPressEvent***(QKeyEvent \*event) Q\_DECL\_OVERRIDE;

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

*public* slots:

void **setSpecStrength**(float specStrength);

void **setSpecIntensity**(float specIntensity);

void **setObjectColor**(QColor objectColor);

signals:

void **specStrengthChanged**(float specStrength);

void **specIntensityChanged**(float specIntensity);

void **light\_1Changed**(GLLight\* light\_1);

void **light\_2Changed**(GLLight\* light\_2);

void **light\_3Changed**(GLLight\* light\_3);

void **objectColorChanged**(QColor objectColor);

*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;

QOpenGLShaderProgram \*m\_lightProgram;

*//* *QOpenGLTexture* *\*text1;*

*//* *QOpenGLTexture* *\*text;*

GLCamera \*m\_camera;

GLLight \*m\_light\_1;

GLLight \*m\_light\_2;

GLLight \*m\_light\_3;

QOpenGLFunctions\_3\_3\_Core m\_glFuncts;

bool m\_isOpenGLInitialized;

GLuint m\_VBO, m\_VAO;

GLuint m\_firstLightVAO, m\_secLightVAO, m\_thirdLightVAO;

GLfloat m\_specStrength;

GLfloat m\_specIntensity;

MainUniforms m\_mainUn;

LightUniforms m\_lightUn;

QColor m\_objectColor;

GLfloat m\_verts[216]{

-0.5f, -0.5f, -0.5f, 0.0f, 0.0f, -1.0f,

0.5f, -0.5f, -0.5f, 0.0f, 0.0f, -1.0f,

0.5f, 0.5f, -0.5f, 0.0f, 0.0f, -1.0f,

0.5f, 0.5f, -0.5f, 0.0f, 0.0f, -1.0f,

-0.5f, 0.5f, -0.5f, 0.0f, 0.0f, -1.0f,

-0.5f, -0.5f, -0.5f, 0.0f, 0.0f, -1.0f,

-0.5f, -0.5f, 0.5f, 0.0f, 0.0f, 1.0f,

0.5f, -0.5f, 0.5f, 0.0f, 0.0f, 1.0f,

0.5f, 0.5f, 0.5f, 0.0f, 0.0f, 1.0f,

0.5f, 0.5f, 0.5f, 0.0f, 0.0f, 1.0f,

-0.5f, 0.5f, 0.5f, 0.0f, 0.0f, 1.0f,

-0.5f, -0.5f, 0.5f, 0.0f, 0.0f, 1.0f,

-0.5f, 0.5f, 0.5f, -1.0f, 0.0f, 0.0f,

-0.5f, 0.5f, -0.5f, -1.0f, 0.0f, 0.0f,

-0.5f, -0.5f, -0.5f, -1.0f, 0.0f, 0.0f,

-0.5f, -0.5f, -0.5f, -1.0f, 0.0f, 0.0f,

-0.5f, -0.5f, 0.5f, -1.0f, 0.0f, 0.0f,

-0.5f, 0.5f, 0.5f, -1.0f, 0.0f, 0.0f,

0.5f, 0.5f, 0.5f, 1.0f, 0.0f, 0.0f,

0.5f, 0.5f, -0.5f, 1.0f, 0.0f, 0.0f,

0.5f, -0.5f, -0.5f, 1.0f, 0.0f, 0.0f,

0.5f, -0.5f, -0.5f, 1.0f, 0.0f, 0.0f,

0.5f, -0.5f, 0.5f, 1.0f, 0.0f, 0.0f,

0.5f, 0.5f, 0.5f, 1.0f, 0.0f, 0.0f,

-0.5f, -0.5f, -0.5f, 0.0f, -1.0f, 0.0f,

0.5f, -0.5f, -0.5f, 0.0f, -1.0f, 0.0f,

0.5f, -0.5f, 0.5f, 0.0f, -1.0f, 0.0f,

0.5f, -0.5f, 0.5f, 0.0f, -1.0f, 0.0f,

-0.5f, -0.5f, 0.5f, 0.0f, -1.0f, 0.0f,

-0.5f, -0.5f, -0.5f, 0.0f, -1.0f, 0.0f,

-0.5f, 0.5f, -0.5f, 0.0f, 1.0f, 0.0f,

0.5f, 0.5f, -0.5f, 0.0f, 1.0f, 0.0f,

0.5f, 0.5f, 0.5f, 0.0f, 1.0f, 0.0f,

0.5f, 0.5f, 0.5f, 0.0f, 1.0f, 0.0f,

-0.5f, 0.5f, 0.5f, 0.0f, 1.0f, 0.0f,

-0.5f, 0.5f, -0.5f, 0.0f, 1.0f, 0.0f

};

QVector3D objPos[3] = {

QVector3D( 1.5f, 0.0f, 0.0f),

QVector3D( 0.0f, 1.5f, 0.0f),

QVector3D( 1.5f, 0.0f, 1.5f),

};

};

#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\_lightProgram(*nullptr*)

, m\_camera(*new* GLCamera)

, m\_light\_1(*new* GLLight(0.0f, 0.0f, 4.0f, QColor("red")))

, m\_light\_2(*new* GLLight(-2.0f, 0.0f, 4.0f, QColor("green")))

, m\_light\_3(*new* GLLight(2.0f, 0.0f, 4.0f, QColor("blue")))

, m\_isOpenGLInitialized(*false*)

, m\_specStrength(32)

, m\_specIntensity(0.5f)

, m\_objectColor(200,200,200)

{

setFlag(QQuickItem::*ItemHasContents*);

setFocus(*true*);

connect(*this*, &QuickItem::specStrengthChanged, *this*, &QuickItem::update);

connect(*this*, &QuickItem::specIntensityChanged, *this*, &QuickItem::update);

connect(m\_light\_1, &GLLight::positionChanged, *this*, &QuickItem::update);

connect(m\_light\_2, &GLLight::positionChanged, *this*, &QuickItem::update);

connect(m\_light\_3, &GLLight::positionChanged, *this*, &QuickItem::update);

connect(m\_light\_1, &GLLight::colorChanged, *this*, &QuickItem::update);

connect(m\_light\_2, &GLLight::colorChanged, *this*, &QuickItem::update);

connect(m\_light\_3, &GLLight::colorChanged, *this*, &QuickItem::update);

connect(*this*, &QuickItem::objectColorChanged, *this*, &QuickItem::update);

}

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

{

}

float QuickItem::**specStrength**() *const*

{

*return* m\_specStrength;

}

float QuickItem::**specIntensity**() *const*

{

*return* m\_specIntensity;

}

GLLight \*QuickItem::**light\_1**() *const*

{

*return* m\_light\_1;

}

GLLight \*QuickItem::**light\_2**() *const*

{

*return* m\_light\_2;

}

GLLight \*QuickItem::**light\_3**() *const*

{

*return* m\_light\_3;

}

QColor QuickItem::**objectColor**() *const*

{

*return* m\_objectColor;

}

void QuickItem::***keyPressEvent***(QKeyEvent \*event)

{

*switch* (event->key()) {

*case* Qt::*Key\_W*:

m\_camera->rotate(-3, m\_camera->right().normalized());

*break*;

*case* Qt::*Key\_D*:

m\_camera->rotate(3, m\_camera->up().normalized());

*break*;

*case* Qt::*Key\_S*:

m\_camera->rotate(3, m\_camera->right().normalized());

*break*;

*case* Qt::*Key\_A*:

m\_camera->rotate(-3, m\_camera->up().normalized());

*break*;

*case* Qt::*Key\_Q*:

m\_camera->rotate(3, m\_camera->target());

*break*;

*case* Qt::*Key\_E*:

m\_camera->rotate(-3, m\_camera->target());

*break*;

*default*:

*return*;

}

update();

}

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::**setSpecStrength**(float specStrength)

{

*if* (qFuzzyCompare(m\_specStrength, specStrength))

*return*;

m\_specStrength = specStrength;

emit specStrengthChanged(m\_specStrength);

}

void QuickItem::**setSpecIntensity**(float specIntensity)

{

*if* (qFuzzyCompare(m\_specIntensity, specIntensity))

*return*;

m\_specIntensity = specIntensity;

emit specIntensityChanged(m\_specIntensity);

}

void QuickItem::**setObjectColor**(QColor objectColor)

{

*if* (m\_objectColor == objectColor)

*return*;

m\_objectColor = objectColor;

emit objectColorChanged(m\_objectColor);

}

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

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

m\_lightProgram->setUniformValue(m\_mainUn.objectColorLoc, m\_objectColor.redF(), m\_objectColor.greenF(), m\_objectColor.blueF());

m\_lightProgram->setUniformValue(m\_mainUn.lightColorLoc\_1, m\_light\_1->colorAsVector());

m\_lightProgram->setUniformValue(m\_mainUn.lightColorLoc\_2, m\_light\_2->colorAsVector());

m\_lightProgram->setUniformValue(m\_mainUn.lightColorLoc\_3, m\_light\_3->colorAsVector());

m\_program->setUniformValue(m\_mainUn.viewPosLoc, m\_camera->eye());

m\_program->setUniformValue(m\_mainUn.lightPosLoc\_1, m\_light\_1->position());

m\_program->setUniformValue(m\_mainUn.lightPosLoc\_2, m\_light\_2->position());

m\_program->setUniformValue(m\_mainUn.lightPosLoc\_3, m\_light\_3->position());

m\_program->release();

m\_lightProgram->bind();

m\_lightProgram->setUniformValue(m\_lightUn.modelLoc, m\_light\_1->model());

m\_lightProgram->setUniformValue(m\_lightUn.viewLoc, m\_camera->view());

m\_lightProgram->setUniformValue(m\_lightUn.projLoc, m\_camera->projection());

m\_lightProgram->setUniformValue(m\_lightUn.colorLoc, m\_light\_1->colorAsVector());

m\_lightProgram->release();

m\_lightProgram->bind();

m\_glFuncts.glBindVertexArray(m\_firstLightVAO);

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

m\_glFuncts.glBindVertexArray(0);

m\_lightProgram->release();

m\_lightProgram->bind();

m\_lightProgram->setUniformValue(m\_lightUn.modelLoc, m\_light\_2->model());

m\_lightProgram->setUniformValue(m\_lightUn.viewLoc, m\_camera->view());

m\_lightProgram->setUniformValue(m\_lightUn.projLoc, m\_camera->projection());

m\_lightProgram->setUniformValue(m\_lightUn.colorLoc, m\_light\_2->colorAsVector());

m\_lightProgram->release();

m\_lightProgram->bind();

m\_glFuncts.glBindVertexArray(m\_secLightVAO);

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

m\_glFuncts.glBindVertexArray(0);

m\_lightProgram->release();

m\_lightProgram->bind();

m\_lightProgram->setUniformValue(m\_lightUn.modelLoc, m\_light\_3->model());

m\_lightProgram->setUniformValue(m\_lightUn.viewLoc, m\_camera->view());

m\_lightProgram->setUniformValue(m\_lightUn.projLoc, m\_camera->projection());

m\_lightProgram->setUniformValue(m\_lightUn.colorLoc, m\_light\_3->colorAsVector());

m\_lightProgram->release();

m\_lightProgram->bind();

m\_glFuncts.glBindVertexArray(m\_thirdLightVAO);

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

m\_glFuncts.glBindVertexArray(0);

m\_lightProgram->release();

m\_program->bind();

m\_program->release();

*for*(int i = 0; i < 3; ++i) {

QMatrix4x4 mat = m\_camera->model();

mat.translate(objPos[i]);

mat.rotate(20.0f \* i, QVector3D(1.0f, 0.3f, 0.5f));

GLfloat specInt = 0.0f;

GLfloat specStr = 0.0f;

*switch*(i) {

*case* 0:

specInt = 0.5f;

specStr = 32.0f;

*break*;

*case* 1:

specInt = 1.0f;

specStr = 128.0f;

*break*;

*case* 2:

specInt = 1.5f;

specStr = 256.0f;

*break*;

}

setSpecStrength(specStr);

setSpecIntensity(specInt);

m\_program->bind();

m\_program->setUniformValue(m\_mainUn.scepStrengthLoc, m\_specStrength);

m\_program->setUniformValue(m\_mainUn.scepIntensityLoc, m\_specIntensity);

m\_lightProgram->setUniformValue(m\_mainUn.modelLoc, mat);

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*();

*if*(m\_lightProgram)

*return*;

m\_lightProgram = *new* QOpenGLShaderProgram();

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

*if*(!m\_lightVert.compileSourceFile(":/lightVert.vert"))

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

m\_lightProgram->addShader(&m\_lightVert);

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

*if*(!m\_lightFrag.compileSourceFile(":/lightFrag.frag"))

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

m\_lightProgram->addShader(&m\_lightFrag);

m\_lightProgram->*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\_mainUn.objectColorLoc = m\_program->uniformLocation("objectColor");

m\_mainUn.lightColorLoc\_1 = m\_program->uniformLocation("lights[0].color");

m\_mainUn.lightColorLoc\_2 = m\_program->uniformLocation("lights[1].color");

m\_mainUn.lightColorLoc\_3 = m\_program->uniformLocation("lights[2].color");

m\_mainUn.lightPosLoc\_1 = m\_program->uniformLocation("lights[0].position");

m\_mainUn.lightPosLoc\_2 = m\_program->uniformLocation("lights[1].position");

m\_mainUn.lightPosLoc\_3 = m\_program->uniformLocation("lights[2].position");

m\_mainUn.viewPosLoc = m\_program->uniformLocation("viewPos");

m\_mainUn.scepStrengthLoc = m\_program->uniformLocation("specStrength");

m\_mainUn.scepIntensityLoc = m\_program->uniformLocation("specIntensity");

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

m\_program->release();

m\_lightProgram->bind();

m\_lightUn.modelLoc = m\_lightProgram->uniformLocation("model");

m\_lightUn.viewLoc = m\_lightProgram->uniformLocation("view");

m\_lightUn.projLoc = m\_lightProgram->uniformLocation("projection");

m\_lightUn.colorLoc = m\_lightProgram->uniformLocation("lightColor");

m\_lightProgram->release();

}

void QuickItem::**initOpenGLState**()

{

*if*(m\_isOpenGLInitialized)

*return*;

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

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.glGenVertexArrays(1, &m\_firstLightVAO);

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

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

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, 6 \* *sizeof*(GLfloat), (GLvoid\*)0);

m\_glFuncts.glEnableVertexAttribArray(0);

m\_glFuncts.glVertexAttribPointer(1, 3, GL\_FLOAT, GL\_FALSE, 6 \* *sizeof*(GLfloat), (GLvoid\*)(3 \* *sizeof*(GLfloat)));

m\_glFuncts.glBindBuffer(GL\_ARRAY\_BUFFER, 0);

m\_glFuncts.glEnableVertexAttribArray(1);

m\_glFuncts.glBindVertexArray(0);

m\_glFuncts.glBindVertexArray(m\_firstLightVAO);

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, 6 \* *sizeof*(GLfloat), (GLvoid\*)0);

m\_glFuncts.glEnableVertexAttribArray(0);

m\_glFuncts.glBindBuffer(GL\_ARRAY\_BUFFER, 0);

m\_glFuncts.glBindVertexArray(0);

m\_glFuncts.glBindVertexArray(m\_secLightVAO);

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, 6 \* *sizeof*(GLfloat), (GLvoid\*)0);

m\_glFuncts.glEnableVertexAttribArray(0);

m\_glFuncts.glBindBuffer(GL\_ARRAY\_BUFFER, 0);

m\_glFuncts.glBindVertexArray(0);

m\_glFuncts.glBindVertexArray(m\_thirdLightVAO);

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, 6 \* *sizeof*(GLfloat), (GLvoid\*)0);

m\_glFuncts.glEnableVertexAttribArray(0);

m\_glFuncts.glBindBuffer(GL\_ARRAY\_BUFFER, 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 gllight.h – содержит вспомогательный класс, инкапсулирующий методы управления источниками света

#ifndef GLLIGHT\_H

#define GLLIGHT\_H

#include <QObject>

#include <QVector3D>

#include <QMatrix4x4>

#include <QColor>

*class* **GLLight** : *public* QObject

{

Q\_OBJECT

Q\_PROPERTY(QColor color READ color WRITE setColor NOTIFY colorChanged)

*public*:

**GLLight**(float x = 0.0f, float y = 0.0f, float z = 0.0f,

QColor color = QColor("white"),

QObject \*parent = *nullptr*);

QVector3D **position**() *const*;

QMatrix4x4 **model**() *const*;

QVector3D **colorAsVector**() *const*;

QColor **color**() *const*;

*public* slots:

void **setPosition**(float x, float y, float z);

void **setColor**(QColor color);

signals:

void **positionChanged**();

void **colorChanged**(QColor color);

*private*:

QVector3D m\_position;

QMatrix4x4 m\_model;

QColor m\_color;

};

#endif *//* *GLLIGHT\_H*

Файл gllight.cpp – содержит реализацию методов класса GLLight

#include "gllight.h"

GLLight::**GLLight**(float x, float y, float z,

QColor color, QObject \*parent) :

QObject(parent)

, m\_model()

, m\_color(color)

{

setPosition(x, y, z);

}

QVector3D GLLight::**position**() *const*

{

*return* m\_position;

}

void GLLight::**setPosition**(float x, float y, float z)

{

m\_position = QVector3D(x, y, z);

QMatrix4x4 newMat;

m\_model = newMat;

m\_model.translate(m\_position);

m\_model.scale(0.2);

emit positionChanged();

}

void GLLight::**setColor**(QColor color)

{

*if* (m\_color == color)

*return*;

m\_color = color;

emit colorChanged(m\_color);

}

QMatrix4x4 GLLight::**model**() *const*

{

*return* m\_model;

}

QVector3D GLLight::**colorAsVector**() *const*

{

*return* QVector3D(m\_color.redF(), m\_color.greenF(), m\_color.blueF());

}

QColor GLLight::**color**() *const*

{

*return* m\_color;

}

Файл mainVert.vert – вершинный шейдер, накладываемый на объект

#version 330

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

layout (location = 1) *in* *vec3* normal;

*out* *vec3* Normal;

*out* *vec3* FragPos;

*uniform* *mat4* model;

*uniform* *mat4* view;

*uniform* *mat4* projection;

*void* main()

{

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

FragPos = *vec3*(model \* *vec4*(position, 1.0f));

Normal = *vec3*(model \* *vec4*(normal, 1.0f));

}

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

#version 330

*out* *vec4* color;

*in* *vec3* Normal;

*in* *vec3* FragPos;

*struct* light {

*vec3* color;

*vec3* position;

};

*uniform* light[3] lights;

*uniform* *vec3* objectColor;

*uniform* *vec3* viewPos;

*uniform* *float* specStrength;

*uniform* *float* specIntensity;

*uniform* *int* lightCount;

*vec3* calcColor(light defLight) {

*//Ambient*

*float* ambientStrength = 0.3f;

*vec3* ambient = ambientStrength \* defLight.color;

*//Deffuse*

*vec3* norm = normalize(Normal);

*vec3* lightDir = normalize(defLight.position - FragPos);

*float* diffAngle = max(dot(norm, lightDir), 0.0f);

*vec3* diffuse = diffAngle \* defLight.color;

*//Specular*

*vec3* viewDir = normalize(viewPos - FragPos);

*vec3* reflectDir = reflect(-lightDir, norm);

*float* specAngle = pow(max(dot(viewDir, reflectDir), 0.0), specStrength);

*vec3* specular = specIntensity \* specAngle \* defLight.color;

*//Result*

*return* (ambient + diffuse + specular);

}

*void* main(*void*)

{

*vec3* result;

*for*(*int* i = 0; i < 3; ++i) {

result += calcColor(lights[i]);

}

result \*= objectColor;

color = *vec4*(result, 1.0f);

}

Файл lightVert.vert – вершинный шейдер, накладываемый на источник света

#version 330

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

layout (location = 1) *in* *vec3* normal;

*uniform* *mat4* model;

*uniform* *mat4* view;

*uniform* *mat4* projection;

*void* main()

{

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

}

Файл lightFrag.frag - фрагментный шейдер, накладываемый на источник света

#version 330 core

*out* *vec4* color;

*uniform* *vec3* lightColor;

*void* main()

{

color = *vec4*(lightColor, 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 объекты. В программе реализованы несколько источников разного света и управление 3D объектами с помощью клавиатуры. При вращении объектов можно видеть отличия отражающей способности отрисованных объектов (рис 1.).

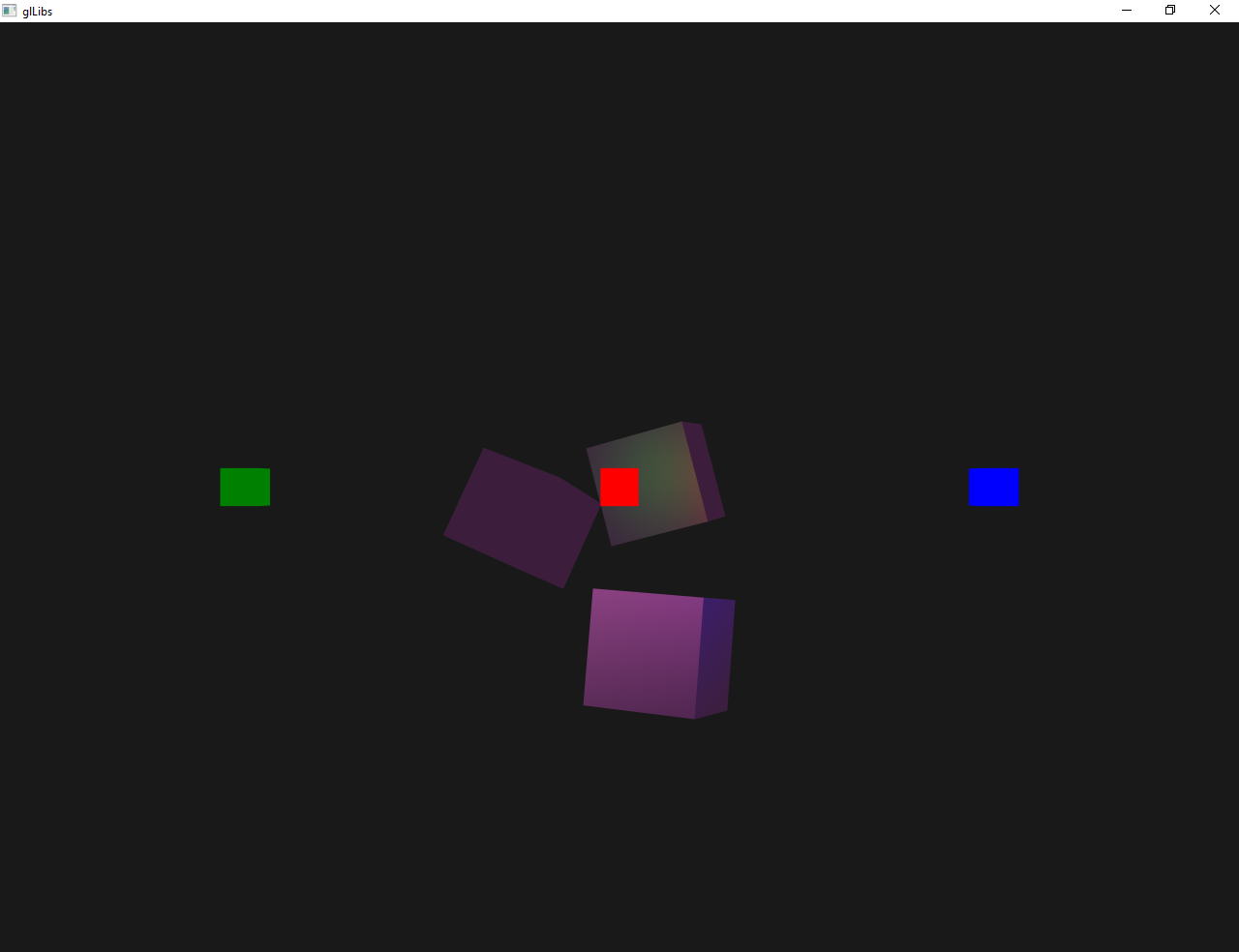
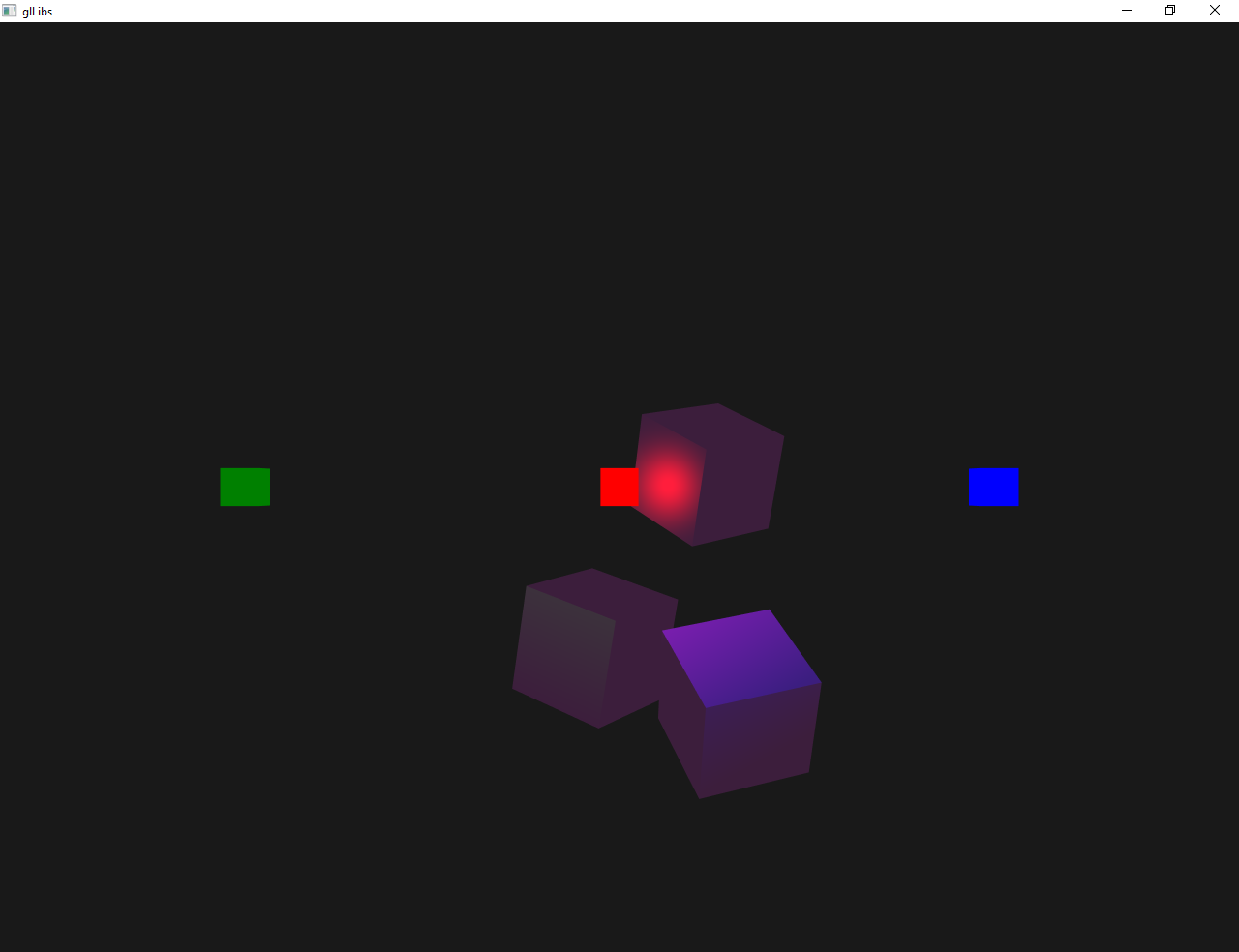
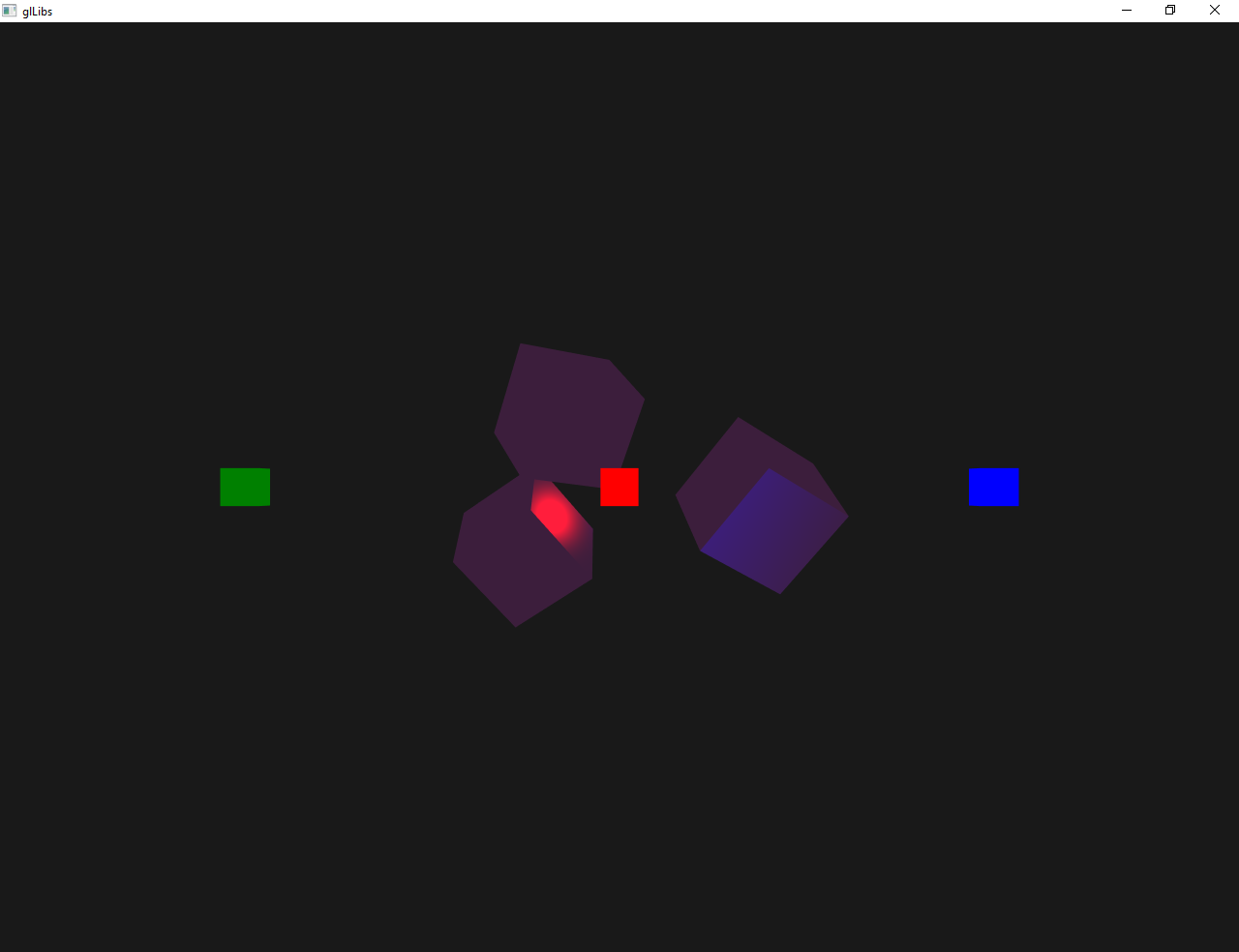


Рис. 1. Результат работы программы