**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИИ**  
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

Расчетно-графическое задание

Компьютерная графика

**Редактор .obj файлов**

Выполнил: ст. группы ВТ-31  
Ковалёв И. Д.

Проверил: Осипов О. В.

**Белгород 2019**

**Постановка задачи:**

Выбранная предметная область: чтение .obj файлов.

Программа должна считывать данные о 3D модели (координаты точек, текстурные координаты, вектора нормали) не зависимо от настроек импорта сторонних приложений, то есть читать в том числе нетриангулированые фигуры, фигуры, в которых нет текстурных координат, и / или координат вектора нормали.

Для выполнения постановленной задачи необходимо реализовать:

Класс ViewWidget, задачей которого является непосредственная отрисовка содержимого файла на экран.

Класс Object, задачей которого будет чтение, хранение и изменение информации об объекте.

Класс CustomPolygon, который будет представлять собой полигон нефиксированной длины и хранить в себе все необходимые данные для отрисовки.

**Содержимое файла MainWindow.h:**

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <QFileDialog>

#include <QOpenGLWidget>

#include <QtOpenGL>

#include <viewwidget.h>

QT\_BEGIN\_NAMESPACE

namespace **Ui** { class **MainWindow**; }

QT\_END\_NAMESPACE

class **MainWindow** : public QMainWindow

{

Q\_OBJECT

public:

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

~***MainWindow***();

private slots:

void **openFile**();

void **light1**();

void **light2**();

void **light3**();

private:

Ui::MainWindow \*ui;

QAction \* openFileAction, \* setupLight1Action, \* setupLight2Action, \* setupLight3Action;

QString filename;

bool l1Enabled, l2Enabled, l3Enabled;

};

#endif // MAINWINDOW\_H

**Содержимое файла MainWindow.cpp:**

#include "mainwindow.h"

#include "ui\_mainwindow.h"

MainWindow::MainWindow(QWidget \*parent)

: QMainWindow(parent), ui(new Ui::MainWindow){

ui->setupUi(this);

setWindowTitle("Это не баг, это фича!");

this->resize(1200, 700);

openFileAction = new QAction("Открыть", this);

connect(openFileAction, SIGNAL(triggered()), this, SLOT(openFile()));

ui->menu->addAction(openFileAction);

ui->view->loadObject(QString("D:/git/labs/cg/rgz/models/Звезда.obj"));

l1Enabled = true; l2Enabled = true; l3Enabled = true;

setupLight1Action = new QAction("Включить источник света 1", this);

setupLight2Action = new QAction("Включить источник света 2", this);

setupLight3Action = new QAction("Включить источник света 3", this);

ui->lightSettings->addAction(setupLight1Action);

ui->lightSettings->addAction(setupLight2Action);

ui->lightSettings->addAction(setupLight3Action);

connect(setupLight1Action, SIGNAL(triggered()), this, SLOT(light1()));

connect(setupLight2Action, SIGNAL(triggered()), this, SLOT(light2()));

connect(setupLight3Action, SIGNAL(triggered()), this, SLOT(light3()));

}

MainWindow::~MainWindow(){

delete ui;

}

void MainWindow::openFile(){

filename = QFileDialog::getOpenFileName(this, "", "\*.obj");

ui->view->loadObject(filename);

}

void MainWindow::light1(){

if (l1Enabled){

setupLight1Action->setText("Отключить источник света 1");

l1Enabled = false;

} else {

setupLight1Action->setText("Включить источник света 1");

l1Enabled = true;

}

ui->view->changeLights(!l1Enabled, 0);

}

void MainWindow::light2(){

if (l2Enabled){

setupLight2Action->setText("Отключить источник света 2");

l2Enabled = false;

} else {

setupLight2Action->setText("Включить источник света 2");

l2Enabled = true;

}

ui->view->changeLights(!l2Enabled, 1);

}

void MainWindow::light3(){

if (l3Enabled){

setupLight3Action->setText("Отключить источник света 3");

l3Enabled = false;

} else {

setupLight3Action->setText("Включить источник света 3");

l3Enabled = true;

}

ui->view->changeLights(!l3Enabled, 2);

}

**Содержимое файла ViewWidget.h:**

#ifndef VIEWWIDGET\_H

#define VIEWWIDGET\_H

#include <QOpenGLWidget>

#include <QtOpenGL>

#include <object.h>

#include <QMatrix4x4>

#include <GL/glu.h>

class ViewWidget : public QGLWidget{

Q\_OBJECT

private:

float zoffset = 3, xAngle = 90, yAngle = 0, globalScale;

QPoint pos;

QString filename;

Object currentObject;

QMatrix4x4 rotationMatrix;

public:

ViewWidget(QWidget \*parent = nullptr);

~ViewWidget(){};

bool isInitialized;

void loadObject(QString filename);

void changeLights(bool enable, int lightNum);

void setupLightning();

void resetProjection();

void resetModelView();

void changeRotationMatrix(float dx, float dy);

void initializeGL() override;

void resizeGL(int w, int h) override;

void paintGL() override;

void mouseMoveEvent(QMouseEvent\* event);

void wheelEvent(QWheelEvent\* event);

void keyPressEvent(QKeyEvent\* event);

void mousePressEvent(QMouseEvent \* event);

};

#endif // VIEWWIDGET\_H

**Содержимое файла Object.h:**

#ifndef OBJECT\_H

#define OBJECT\_H

#include <QVector>

#include <QVector3D>

#include <QtOpenGL>

#include <QOpenGLWidget>

class CustomPolygon{

QVector <QVector3D> positions;

QVector <QVector3D> normals;

QVector <QVector2D> texture;

GLenum drawMode;

public:

CustomPolygon(){};

CustomPolygon(QVector <QVector3D> positions);

CustomPolygon(QVector <QVector3D> positions, QVector <QVector3D> normals);

CustomPolygon(QVector <QVector3D> positions, QVector <QVector3D> normals, QVector <QVector2D> texture);

void pushPolygon(QVector3D position, QVector3D normal, QVector2D texture);

void draw();

void setDrawMode(GLenum mode);

void addPosition(QVector3D newPosition);

void addNormal(QVector3D newNormal);

void addTexture(QVector2D newTexture);

void translatePositions(QVector3D center);

QVector <QVector3D> getPositions() {return positions;}

};

class Object{

private:

QVector <CustomPolygon> p;

QVector <uint> indicies;

QVector3D center;

public:

Object();

~Object();

void init();

bool read(QString filename);

QVector <uint> getIndicies(){return indicies;}

QVector3D getCenter();

void translate();

void draw();

};

#endif // OBJECT\_H

**Содержимое файла ViewWidget.cpp:**

#include "viewwidget.h"

#include <GL/glu.h>

ViewWidget::ViewWidget(QWidget \*parent)

: QGLWidget(parent){

currentObject = Object();

isInitialized = false;

globalScale = 0.1f;

}

void ViewWidget::initializeGL(){

glEnable(GL\_DEPTH\_TEST);

glEnable(GL\_NORMALIZE);

glShadeModel(GL\_SMOOTH);

glEnable(GL\_MULTISAMPLE);

glClearColor(1.0f, 1.0f, 1.0f, 1.0f);

resetModelView();

resetProjection();

setupLightning();

}

void ViewWidget::resizeGL(int w, int h){

glViewport(0, 0, w, h);

resetProjection();

}

void ViewWidget::changeLights(bool enable, int lightNum){

switch (lightNum) {

case 0:

if (enable)

glEnable(GL\_LIGHT0);

else glDisable(GL\_LIGHT0);

break;

case 1:

if (enable)

glEnable(GL\_LIGHT1);

else glDisable(GL\_LIGHT1);

break;

case 2:

if (enable)

glEnable(GL\_LIGHT2);

else glDisable(GL\_LIGHT2);

break;

}

repaint();

}

void ViewWidget::paintGL(){

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

if (isInitialized){

currentObject.draw();

}

glPushMatrix();

glRotatef(xAngle \* 2, 1, 0, 0);

glRotatef(yAngle \* 2, 0, 1, 0);

glPopMatrix();

currentObject.draw();

}

void ViewWidget::resetProjection(){

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluPerspective(30.0, (float)width() / height(), 0.1, 20);

}

void ViewWidget::resetModelView(){

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

glTranslatef(0, 0, -zoffset);

glMultMatrixf(rotationMatrix.inverted().data());

glScalef(globalScale, globalScale, globalScale);

}

void ViewWidget::changeRotationMatrix(float dx, float dy){

rotationMatrix.rotate(-dx, 0, 1, 0);

rotationMatrix.rotate(-dy, 1, 0, 0);

}

void ViewWidget::mouseMoveEvent(QMouseEvent \* event){

QPoint newPos = event->pos();

float dx = newPos.x() - pos.x();

float dy = newPos.y() - pos.y();

changeRotationMatrix(dx / 2, dy / 2);

pos = newPos;

resetModelView();

repaint();

}

void ViewWidget::wheelEvent(QWheelEvent \* event){

zoffset -= event->delta() / 1000.0f;

resetProjection();

resetModelView();

repaint();

}

void ViewWidget::mousePressEvent(QMouseEvent \* event){

pos = event->pos();

repaint();

}

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

if (event->key() == Qt::Key\_Escape){

close();

}

}

void ViewWidget::loadObject(QString filename){

isInitialized = false;

if (currentObject.read(filename))

isInitialized = true;

}

**Содержимое файла Object.cpp:**

#include "object.h"

CustomPolygon::**CustomPolygon**(QVector <QVector3D> positions) : positions(positions){};

CustomPolygon::**CustomPolygon**(QVector <QVector3D> positions, QVector <QVector3D> normals)

: positions(positions), normals(normals) {};

CustomPolygon::**CustomPolygon**(QVector <QVector3D> positions, QVector <QVector3D> normals, QVector <QVector2D> texture)

: positions(positions), normals(normals), texture(texture) {};

void CustomPolygon::**draw**(){

glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_FILL);

glBegin(GL\_POLYGON);

if (normals.size() != 0)

glNormal3f(normals[0].x(), normals[0].y(), normals[0].z());

for (int i = 0; i < positions.size(); i++){

glVertex3f(positions[i].x(), positions[i].y(), positions[i].z());

}

glEnd();

}

void CustomPolygon::**setDrawMode**(GLenum mode){

drawMode = mode;

}

void CustomPolygon::**pushPolygon**(QVector3D position, QVector3D normal, QVector2D texture){

this->positions.push\_back(position);

this->normals.push\_back(normal);

this->texture.push\_back(texture);

}

void CustomPolygon::**addPosition**(QVector3D newPosition){

positions.push\_back(newPosition);

}

void CustomPolygon::**addNormal**(QVector3D newNormal){

normals.push\_back(newNormal);

}

void CustomPolygon::**addTexture**(QVector2D newTexture){

texture.push\_back(newTexture);

}

void CustomPolygon::**translatePositions**(QVector3D center){

for (int i = 0; i < positions.size(); i++){

positions[i].setX(positions[i].x() - center.x());

positions[i].setY(positions[i].y() - center.y());

positions[i].setZ(positions[i].z() - center.z());

}

}

Object::**Object**(){}

bool Object::**read**(QString filename){

p.clear();

indicies.clear();

QFile f(filename);

if (!f.exists()) {

qDebug() << "Такого файла нет!";

return false;

}

if (!f.*open*(QIODevice::ReadOnly)){

qDebug() << "Не могу открыть файл!";

return false;

}

QTextStream istream(&f);

QVector<QVector3D> \_v;

QVector<QVector2D> \_t;

QVector<QVector3D> \_n;

QString tmp;

QStringList list, indiciesList;

while (!istream.atEnd()){

tmp = istream.readLine();

list = tmp.split(" ");

if (list[0] == "v"){

\_v.append(QVector3D(list[1].toFloat(), list[2].toFloat(), list[3].toFloat()));

continue;

}

if (list[0] == "vt"){

\_t.append(QVector2D(list[1].toFloat(), list[2].toFloat()));

continue;

}

if (list[0] == "vn"){

\_n.append(QVector3D(list[1].toFloat(), list[2].toFloat(), list[3].toFloat()));

continue;

}

if (list[0] == "f"){

CustomPolygon newPolygon;

if (list[1].contains("/")){

for (int i = 1; i < list.size(); i++){

indiciesList = list[i].split("/");

newPolygon.addPosition(\_v[indiciesList[0].toLong() - 1]);

if (indiciesList[2].size() != 0){

newPolygon.addNormal(\_n[indiciesList[2].toLong()-1]);

}

if (indiciesList[1].size() != 0){

newPolygon.addTexture(\_t[indiciesList[1].toLong()-1]);

}

}

} else {

for (int i = 1; i < list.size(); i++){

if (list[i] != "")

newPolygon.addPosition(\_v[list[i].toLong() - 1]);

}

}

p.push\_back(newPolygon);

}

}

f.*close*();

translate();

return true;

}

void Object::**translate**(){

center = getCenter();

for (int i = 0; i < p.size(); i++){

p[i].translatePositions(center);

}

}

void Object::**draw**(){

for (int i = 0; i < p.size(); i++)

p[i].draw();

}

QVector3D Object::**getCenter**(){

QVector3D result(0, 0, 0);

int totalSize = 0;

for (int i = 0; i < p.size(); i++){

QVector <QVector3D> currentPositions = p[i].getPositions();

totalSize += currentPositions.size();

for (int j = 0; j < currentPositions.size(); j++){

result.setX(result.x() + currentPositions[j].x());

result.setY(result.y() + currentPositions[j].y());

result.setZ(result.z() + currentPositions[j].z());

}

}

result.setX(result.x() / totalSize);

result.setY(result.y() / totalSize);

result.setZ(result.z() / totalSize);

return result;

}

Object::~**Object**() {}

Результат работы программы:





