**Московский авиационный институт**

**(Национальный исследовательский университет)**

Институт: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Компьютерная графика»

**Лабораторная работа № 6**

Тема: Создание шейдерных анимационных эффектов в OpenGL

Студент: Айрапетова Евгения Ашотовна

Группа: 80-306

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2021

1. **Постановка задачи**

Для поверхности, созданной в л.р. №4-5, обеспечить выполнение следующего шейдерного эффекта:

Вариант задания:

Анимация. Координата X изменяется по закону X= X\*cos(t).

1. **Описание программы**

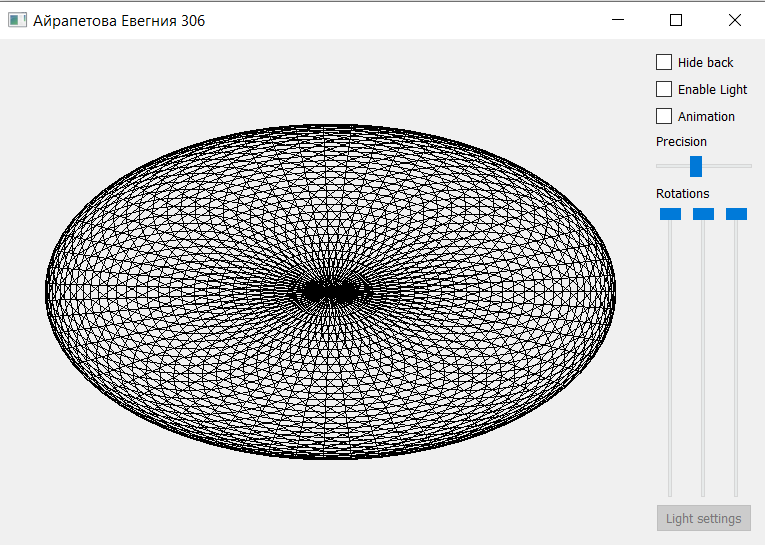
Для решения задачи я использовала фреймворк Qt и язык С++. Были реализованы функции для искажения фигуры по заданному закону и обновление ее через определенный промежуток времени. Для работы с временем использован QTimer.

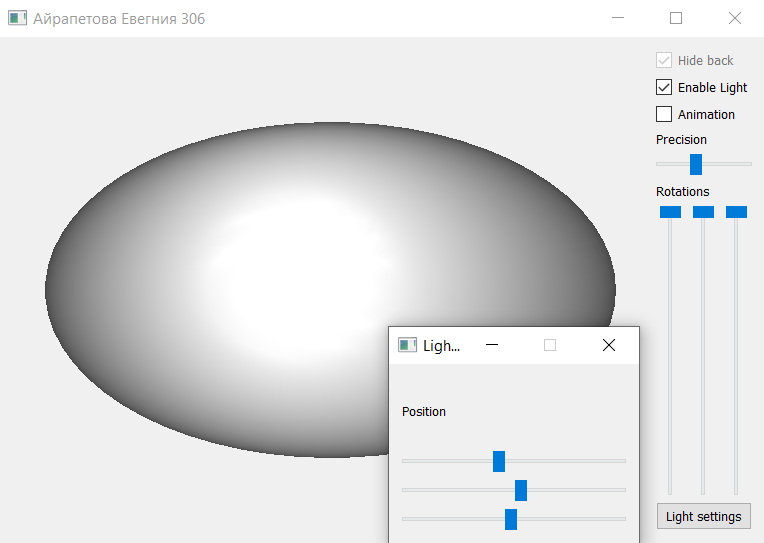
Также были реализованы классы для вычисления вершин эллипсоида, класс для отображения освещения и самого виджета.

Изменение параметров эллипсоида реализовано в классе ViewWidget. Можно изменить параметр аппроксимации (бегунок Precision), а также выбрать режим просмотра: каркасный или режим поверхности (чек-бокс Enable Light). Также во втором режиме доступно изменение освещения фигуры.

Изменение параметров освещения описано в классе LightSettingsWidget. Можно передвигать источник освещения по X, Y, Z. В диалоговом окне изменение производится за счет изменения значений на трех бегунках соответственно. Реализовано вращение эллипсоида с помощью трех бегунков справа в окне, вращение происходит в направлениях по X, Y, Z соответственно.

1. **Тесты**





1. **Листинг программы**

main.cpp:

//Айрапетова

#include "glwidget.h"

#include <QApplication>

#include <viewwidget.h>

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

{

QApplication a(argc, argv);

ViewWidget\* view = new ViewWidget;

view->setWindowTitle("Айрапетова Евгения 306");

view->show();

return a.exec();

}

glabstractpainting.h:

#ifndef GLABSTRACTPAINTING\_H

#define GLABSTRACTPAINTING\_H

#include<QtOpenGL>

const GLenum FRAME = GL\_LINE;

const GLenum MATERIAL = GL\_FILL;

class GLAbstractPainting : public QObject

{

private:

GLenum mode = FRAME;

QColor colorFrame = Qt::black;

QColor colorMaterial = Qt::yellow;

public:

void setFrameColor(QColor c);

QColor getFrameColor() const;

void setMaterialColor(QColor c);

QColor getMaterialColor() const;

void setMode(GLenum m);

QColor getColor();

GLenum getMode() const;

GLAbstractPainting();

void addPolygon(const QVector3D& f, const QVector3D& s, const QVector3D& t);

void addPolygon(const QVector3D& f, const QVector3D& s, const QVector3D& t, QColor c);

virtual void paint() = 0;

virtual void animate(int){};

virtual ~GLAbstractPainting();

};

#endif // GLABSTRACPAINTING\_H

gllight.h:

#ifndef GLLIGHT\_H

#define GLLIGHT\_H

#include <QColor>

#include <QVector4D>

class GLLight

{

public:

QColor diffuse = Qt::black;

QColor specular = Qt::black;

QColor ambient = Qt::black;

QVector4D direction = {0.0,0.0,0.0,1.0};

GLLight();

};

#endif // GLLIGHT\_H

glwidget.h

#ifndef GLWIDGET\_H

#define GLWIDGET\_H

#define GL\_GLEXT\_PROTOTYPES

#include<QtOpenGL>

#include"glfigureengine.h"

class GlWidget : public QGLWidget {

public:

GLFigureEngine Engine;

GlWidget(QWidget\* parent = nullptr);

void initializeGL() override;

void resizeGL(int nWidth, int nHeight) override;

void paintGL() override;

void scene();

void read(int a);

void cuboid();

};

#endif // GLWIDGET\_H

glwidget.cpp

#include "glwidget.h"

GlWidget::GlWidget(QWidget\* parent) : QGLWidget(parent) {

setMinimumHeight(480);

setMinimumWidth(480);

}

void GlWidget::initializeGL(){

qglClearColor(QColor(240,240,240));

glEnable(GL\_DEPTH\_TEST); // устанавливает режим проверки глубины пикселей

// glShadeModel(GL\_FLAT); // отключает режим сглаживания цветов

}

void GlWidget::resizeGL(int nW, int nH){

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

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

glViewport(0, 0, (GLint)nW, (GLint)nH);

}

void GlWidget::paintGL(){

Engine.setProjection(width(), height());

Engine.paint();

}

glabstractpainting.cpp:

#include "glabstractpainting.h"

GLAbstractPainting::GLAbstractPainting()

{

}

GLAbstractPainting::~GLAbstractPainting()

{

}

void GLAbstractPainting::setMode(GLenum m){

mode = m;

}

GLenum GLAbstractPainting::getMode() const{

return mode;

}

void GLAbstractPainting::setFrameColor(QColor c){

colorFrame = c;

}

QColor GLAbstractPainting::getFrameColor() const{

return colorFrame;

}

void GLAbstractPainting::setMaterialColor(QColor c){

colorMaterial = c;

}

QColor GLAbstractPainting::getMaterialColor() const{

return colorMaterial;

}

void GLAbstractPainting::addPolygon(const QVector3D& f, const QVector3D& s, const QVector3D& t){

auto c = getColor();

glBegin(GL\_POLYGON);

glColor3f(c.redF(), c.greenF(), c.blueF());

glVertex3f( f.x(), f.y(), f.z());

glVertex3f( s.x(), s.y(), s.z());

glVertex3f( t.x(), t.y(), t.z());

glEnd();

}

void addPolygon(const QVector3D& f, const QVector3D& s, const QVector3D& t, QColor c){

glBegin(GL\_POLYGON);

glColor3f(c.redF(), c.greenF(), c.blueF());

glVertex3f( f.x(), f.y(), f.z());

glVertex3f( s.x(), s.y(), s.z());

glVertex3f( t.x(), t.y(), t.z());

glEnd();

}

QColor GLAbstractPainting::getColor(){

return getMode() == FRAME ? getFrameColor() : getMaterialColor();

}

glfigureengine.h:

#ifndef GLFIGUREENGINE\_H

#define GLFIGUREENGINE\_H

#include<QVector>

#include<QQueue>

#include<QtOpenGL>

#include "glabstractpainting.h"

#include "gllight.h"

#include <QObject>

#include <QTimer>

class GLFigureEngine : public QObject

{

Q\_OBJECT

private:

float rotateX = 0.0;

float rotateY = 0.0;

float rotateZ = 0.0;

float scale = 1.0;

bool isIzomteric = false;

bool isOrtho = true;

bool isHiddingBack = false;

bool isLightEnab = false;

bool isAnimate = false;

QVector<GLAbstractPainting\*> paintings;

QQueue<GLenum> enableMods;

QQueue<GLenum> disableMods;

void applyTransform() const;

void applyLight() const;

void disableBooleanProj();

void prePaintSettings(GLAbstractPainting\* figure);

public:

QTimer timer;

GLFigureEngine();

GLLight light;

void paint();

void addFigure(GLAbstractPainting\* f);

void deleteFigures();

~GLFigureEngine();

void setRotation(float x, float y, float z);

float getRotationX() const;

float getRotationY() const;

float getRotationZ() const;

void setRotationX(float a);

void setRotationY(float a);

void setRotationZ(float a);

void setIzomtericProjection();

void setOrthoProjection();

bool isIzomtericProjection();

bool isOrthoProjection();

bool isHidingBack() const;

void setHidingBack(bool b);

void setProjection(int w, int h);

bool isLightEnabled() const;

void setLightEnabled(bool b);

void setAnimation(bool s);

bool Animation();

unsigned long long tick = 0;

public slots:

signals:

void rePaint();

};

#endif // GLFIGUREENGINE\_H

glfigureengine.cpp

#include "glfigureengine.h"

GLFigureEngine::GLFigureEngine()

{

timer.start(100);

}

void GLFigureEngine::paint(){

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

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

applyTransform();

applyLight();

for(auto\* painting : paintings){

if(!isAnimate){ //!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

if(painting->getMode() == MATERIAL){

prePaintSettings(painting);

painting->paint();

}

if(painting->getMode() == FRAME){

if(isHidingBack()){

glLineWidth(1.5);

prePaintSettings(painting);

painting->paint();

painting->setMode(MATERIAL);

painting->setMaterialColor(QColor(240,240,240));

prePaintSettings(painting);

painting->paint();

painting->setMode(FRAME);

} else{

glLineWidth(1);

prePaintSettings(painting);

painting->paint();

}

}

} else {

if(painting->getMode() == MATERIAL){

prePaintSettings(painting);

painting->animate(tick); //!!!!!!!!!!!!

}

if(painting->getMode() == FRAME){

if(isHidingBack()){

glLineWidth(1.5);

prePaintSettings(painting);

painting->animate(tick);

painting->setMode(MATERIAL);

painting->setMaterialColor(QColor(240,240,240));

prePaintSettings(painting);

painting->animate(tick); //!!!!!!!!!!!!!

painting->setMode(FRAME);

} else{

glLineWidth(1);

prePaintSettings(painting);

painting->animate(tick); //!!!!!!!!!!!!!!

}

}

++tick; //!!!!!!!!!!!!!!!!!!!!

}

}

}

void GLFigureEngine::addFigure(GLAbstractPainting\* f){

paintings.push\_back(f);

}

void GLFigureEngine::deleteFigures(){

paintings.clear();

}

GLFigureEngine::~GLFigureEngine(){

}

void GLFigureEngine::setRotation(float x, float y, float z){

setRotationX(x);

setRotationY(y);

setRotationZ(z);

}

float GLFigureEngine::getRotationX() const{

return rotateX;

}

float GLFigureEngine::getRotationY() const{

return rotateY;

}

float GLFigureEngine::getRotationZ() const{

return rotateZ;

}

void GLFigureEngine::setRotationX(float a){

rotateX = a;

}

void GLFigureEngine::setRotationY(float a){

rotateY = a;

}

void GLFigureEngine::setRotationZ(float a){

rotateZ = a;

}

void GLFigureEngine::applyTransform() const{

glRotatef(rotateX, 1.0,0.0,0.0);

glRotatef(rotateY, 0.0,1.0,0.0);

glRotatef(rotateZ, 0.0,0.0,1.0);

glScalef(scale, scale, scale);

}

void GLFigureEngine::disableBooleanProj(){

isIzomteric = isOrtho = false;

}

void GLFigureEngine::setIzomtericProjection(){

disableBooleanProj();

isIzomteric = true;

}

void GLFigureEngine::setOrthoProjection(){

disableBooleanProj();

isOrtho = true;

}

bool GLFigureEngine::isIzomtericProjection(){

return isIzomteric;

}

bool GLFigureEngine::isOrthoProjection(){

return isOrtho;

}

void GLFigureEngine::setProjection(int w, int h){

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

float res = (float)w/(float)h;

float hK = (res > 1 ? res : 1);

float vK = (res < 1 ? 1/res : 1);

if(isOrthoProjection())

glOrtho(-1.0\*hK, 1.0\*hK, -1.0\*vK, 1.0\*vK, -10.0, 1.0);

if(isIzomtericProjection()){

glFrustum(-1.0\*hK, 1.0\*hK, -1.0\*vK, 1.0\*vK, 8, 10.0);

glTranslatef(0.0,0.0,-9.0);

}

}

void GLFigureEngine::prePaintSettings(GLAbstractPainting\* figure){

glPolygonMode(GL\_FRONT\_AND\_BACK, figure->getMode());

}

bool GLFigureEngine::isHidingBack() const{

return isHiddingBack;

}

void GLFigureEngine::setHidingBack(bool b){

if(isHiddingBack != b){

isHiddingBack = b;

}

}

bool GLFigureEngine::isLightEnabled() const{

return isLightEnab;

}

void GLFigureEngine::setLightEnabled(bool b){

isLightEnab = b;

}

void GLFigureEngine::applyLight() const{

if(isLightEnab){

glEnable(GL\_LIGHTING);

glLightModelf(GL\_LIGHT\_MODEL\_TWO\_SIDE, GL\_TRUE);

glEnable(GL\_NORMALIZE);

float red = light.diffuse.redF();

float green = light.diffuse.greenF();

float blue = light.diffuse.blueF();

float light0\_diffuse[] = {red, green, blue};

red = light.specular.redF();

green = light.specular.greenF();

blue = light.specular.blueF();

float light0\_specular[] = {red, green, blue};

float light0\_direction[] = {light.direction.x(), light.direction.y(), light.direction.z(), 1.0};

red = light.ambient.redF();

green = light.ambient.greenF();

blue = light.ambient.blueF();

float light0\_ambient[] = {red, green, blue};

glEnable(GL\_LIGHT0);

glLightfv(GL\_LIGHT0, GL\_DIFFUSE, light0\_diffuse);

glLightfv(GL\_LIGHT0, GL\_SPECULAR, light0\_specular);

glLightfv(GL\_LIGHT0, GL\_POSITION, light0\_direction);

glLightfv(GL\_LIGHT0, GL\_AMBIENT, light0\_ambient);

GLfloat white[] = { 1.0f, 1.0f, 1.0f, 1.0f };

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_SPECULAR, white);

glMaterialf(GL\_FRONT\_AND\_BACK, GL\_SHININESS, 100.0f);

}

else{

glDisable(GL\_LIGHTING);

}

}

void GLFigureEngine::setAnimation(bool s){ //!

if(isAnimate != s){

isAnimate = s;

if(isAnimate){

timer.start(20);

}

else{

timer.stop();

}

}

}

bool GLFigureEngine::Animation(){

return isAnimate;

}

**Вывод**

В процессе выполнения данной лабораторной работы я вспомнила, как работать с библиотекой QTimer, а также узнала, как реализовать анимацию с помощью OpenGL.

**ЛИТЕРАТУРА**

Документация по Qt [Электронный ресурс]: https://doc.qt.io (дата обращения 24.12.2021).