**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

Лабораторная работа № 5

дисциплина “Компьютерная графика”

по теме “ Алгоритмы удаления невидимых поверхностей ”

Выполнил: ст. группы ПВ-31  
Ковалев Павел Александрович

Проверил: Осипов О.В.

**Белгород 2020**

**Вариант 6**

**Цель работы:** изучить алгоритмы удаления невидимых поверхностей и создать программу для визуализации объёмной трёхмерной модели с закрашенными гранями. Порядок выполнения работы

Разработать алгоритм и составить программу для построения на экране трёхмерной модели с закрашенными гранями в соответствии с номером варианта лабораторной работы №4.

Требования к программе

1. В программе по центру окна должна отображаться только центральная проекция фигуры. Грани фигуры должны быть закрашены и отсортированы по глубине, т.е. пользователь должен видеть передние грани. Должна быть предусмотрена возможность задавать параметры центральной проекции: угол обзора или удалённость точки схода.
2. Пользователь должен иметь возможность поворачивать и перемещать фигуру вдоль оси с использованием мыши. Поворот фигуры лучше выполнять при обработке события mouseMoveEvent при зажатой кнопке мыши, перенос – при обработке собятия wheelEvent. Фигура должна поворачиваться «вслед» за мышью. Т.е., если мышь перемещается влево-вправо, то фигура должна поворачиваться вокруг вертикальной оси; если вверх-вниз, то вокруг горизонтальной оси.

В программе должна быть предусмотрена возможность ввода пользователем исходных данных (из правой колонки таблицы №1 лабораторной работы №4).

**Ход работы**

**Приложения:**

Содержимое файла mainwindow.h

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <QPainter>

#include <QMouseEvent>

#include "object.h"

#include <QMatrix4x4>

QT\_BEGIN\_NAMESPACE

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

QT\_END\_NAMESPACE

*class* **MainWindow** : *public* QMainWindow

{

Q\_OBJECT

*public*:

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

~***MainWindow***();

void ***paintEvent***(QPaintEvent \* event);

void **drawObject**(QPainter \* painter);

*private*:

Ui::MainWindow \*ui;

bool debugFlag;

Cubes obj;

Cubes translatedObject;

int range, view;

double globalScale, centralViewPoint;

QMatrix4x4 rotationMatrix, M1, M\_t;

QPointF pos, paintingPolygonPos;

void ***mouseMoveEvent***(QMouseEvent \* event);

*static* bool **polygonComp**(CustomPolygon a, CustomPolygon b);

};

#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*);

obj = Cubes(1);

translatedObject = obj;

pos = QPointF(0, 0);

}

MainWindow::~***MainWindow***(){

*delete* ui;

}

QColor **getRandomColor**(){

*return* QColor(rand() % 255, rand() % 255, rand() % 255);

}

void MainWindow::***paintEvent***(QPaintEvent \*e){

QPainter painter(*this*);

drawObject(&painter);

}

void MainWindow::**drawObject**(QPainter \* painter){

QVector3D center = obj.getCenter();

float scale = 35;

QVector<QPolygonF> polygons;

QPolygonF polygon;

translatedObject = obj;

QVector4D v;

QMatrix4x4 M, M1, M\_t;

M1 = {1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1};

M.translate(-center);

M\_t.scale(scale, -scale, scale);

M = M.transposed() \* M\_t.transposed();

M = M \* rotationMatrix;

M\_t.setToIdentity();

M = M \* M\_t.transposed();

M\_t.setToIdentity();

M\_t.translate(ui->br->geometry().center().x(), ui->br->geometry().center().y(), 0);

M = M \* M\_t.transposed();

*for* (int i = 0; i < translatedObject.obj.size(); i++)

*for*(int j = 0; j < translatedObject.obj[i].getPolygon().size(); j++){

v.setX(translatedObject.obj[i].getPolygon()[j].x());

v.setY(translatedObject.obj[i].getPolygon()[j].y());

v.setZ(translatedObject.obj[i].getPolygon()[j].z());

v.setW(1);

v = v \* M;

QVector3D tmpPolygon;

tmpPolygon.setX(v.x() / v.w());

tmpPolygon.setY(v.y() / v.w());

tmpPolygon.setZ(v.z() / v.w());

translatedObject.obj[i].setVector(tmpPolygon, j);

}

std::stable\_sort(translatedObject.obj.begin(), translatedObject.obj.end(), polygonComp);

M.setToIdentity();

M.translate(-1 \* ui->br->geometry().center().x(), -1 \* ui->br->geometry().center().y(), 0);

M = M.transposed() \* M1.transposed();

M\_t.setToIdentity();

M\_t.translate(ui->br->geometry().center().x(), ui->br->geometry().center().y(), 0);

M = M \* M\_t.transposed();

*for* (int i = 0; i < translatedObject.obj.size(); i++){

*for*(int j = 0; j < translatedObject.obj[i].getPolygon().size(); j++){

v.setX(translatedObject.obj[i].getPolygon().at(j).x());

v.setY(translatedObject.obj[i].getPolygon().at(j).y());

v.setZ(translatedObject.obj[i].getPolygon().at(j).z());

v.setW(1);

v = v \* M;

polygon << QPointF(v.x()/v.w(), v.y()/v.w());

}

polygons << polygon;

polygon.clear();

}

QColor red = getRandomColor();

*for* (int i = polygons.size() - 1; i >= 0; i--){

*if* (polygons[i].containsPoint(paintingPolygonPos, Qt::FillRule::*OddEvenFill*)

&& !translatedObject.getObject()[i].isColored()){

qDebug() << "!!!" << i ;

paintingPolygonPos = QPointF(0, 0);

obj.obj[translatedObject.obj[i].getNumber()].setColor(red);

qDebug() << translatedObject.obj[i].getNumber() << i ;

}

}

*for* (int i = 0; i < translatedObject.obj.size(); i++){

painter->setBrush(translatedObject.obj[i].getColor());

painter->drawPolygon(polygons[i]);

}

}

bool MainWindow::**polygonComp**(CustomPolygon a, CustomPolygon b){

*return* a.getDepth() <= b.getDepth();

}

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

double k = 5;

QPointF dp = event->pos() - pos;

pos = event->pos();

rotationMatrix.rotate(- dp.x() / k, QVector3D(0, 1, 0));

rotationMatrix.rotate(dp.y() / k, QVector3D(1, 0, 0));

repaint();

}

Содержимое файла mainwindow.ui

<?xml version="1.0" encoding="UTF-8"?>

<ui version="4.0">

<class>MainWindow</class>

<widget class="QMainWindow" name="MainWindow">

<property name="geometry">

<rect>

<x>0</x>

<y>0</y>

<width>800</width>

<height>600</height>

</rect>

</property>

<property name="windowTitle">

<string>MainWindow</string>

</property>

<widget class="QWidget" name="br"/>

</widget>

<resources/>

<connections/>

</ui>

Содержимое файла object.h

#ifndef OBJECT\_H

#define OBJECT\_H

#include <QVector3D>

#include <QtMath>

#include <custompolygon.h>

*class* **Object**{

*public*:

QVector <CustomPolygon> obj;

**Object**() {}

QVector<CustomPolygon> **getObject**();

QVector3D **getCenter**();

QVector3D **getSize**();

void **setPolygon**(CustomPolygon p, int pos);

void **setPolygonColor**(int i, QColor color);

};

*class* **Cubes** : *public* Object{

*public*:

**Cubes**();

**Cubes**(int range);

};

*class* **Star** : *public* Object{

*public*:

**Star**();

};

#endif *//* *OBJECT\_H*

Содержимое файла object.cpp

#include "object.h"

QVector<CustomPolygon> Object::**getObject**(){

*return* obj;

}

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

float x0 = 0, y0 = 0, z0 = 0, x1, y1, z1;

float n1, n = obj.size();

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

x1 = 0;

y1 = 0;

z1 = 0;

n1 = obj[i].getPolygon().size();*//.at(i).size();*

*for* (int j = 0; j < obj[i].getPolygon().size(); j++){

x1 += obj[i].getPolygon().at(j).x();

y1 += obj[i].getPolygon().at(j).y();

z1 += obj[i].getPolygon().at(j).z();

}

x0 += x1/n1;

y0 += y1/n1;

z0 += z1/n1;

}

*return* QVector3D(x0/n, y0/n, z0/n);

}

QVector3D Object::**getSize**(){

float min\_x = obj[0].getPolygon().at(0).x(),

max\_x = min\_x,

min\_y = obj[0].getPolygon().at(0).y(),

max\_y = min\_y,

min\_z = obj[0].getPolygon().at(0).z(),

max\_z = min\_z;

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

*for* (int j = 0; j < obj[i].getPolygon().size(); j++){

*if* (obj[i].getPolygon().at(j).x() < min\_x) min\_x = obj[i].getPolygon().at(j).x();

*if* (obj[i].getPolygon().at(j).x() > max\_x) max\_x = obj[i].getPolygon().at(j).x();

*if* (obj[i].getPolygon().at(j).y() < min\_y) min\_y = obj[i].getPolygon().at(j).y();

*if* (obj[i].getPolygon().at(j).y() > max\_y) max\_y = obj[i].getPolygon().at(j).y();

*if* (obj[i].getPolygon().at(j).z() < min\_z) min\_z = obj[i].getPolygon().at(j).z();

*if* (obj[i].getPolygon().at(j).z() > max\_z) max\_z = obj[i].getPolygon().at(j).z();

}

}

*return* QVector3D(max\_x - min\_x, max\_y - min\_y, max\_z - min\_z);

}

void Object::**setPolygon**(CustomPolygon p, int pos){

obj[pos] = p;

}

Cubes::**Cubes**(){

Cubes(1);

}

Cubes::**Cubes**(int range){

QVector3D A (0,0,0), B (4,0,0), C (4,4,0), D (0,4,0),

A1(0,0,4), B1(4,0,4), C1(4,4,4), D1(0,4,4);

QVector<QVector3D> tmp;

tmp << A << B << C << D; obj << tmp; tmp.clear();

tmp << B << C << C1 << B1; obj << tmp; tmp.clear();

tmp << B1 << C1 << D1 << A1; obj << tmp; tmp.clear();

tmp << A1 << D1 << D << A; obj << tmp; tmp.clear();

tmp << A << A1 << B1 << B; obj << tmp; tmp.clear();

tmp << D << D1 << C1 << C; obj << tmp; tmp.clear();

A = QVector3D(4, 3, 3); B = QVector3D(4, 3, 1);

C = QVector3D(5 + range, 3, 1); D = QVector3D(5 + range, 3, 3);

A1 = QVector3D(4, 1, 3); B1 = QVector3D(4, 1, 1);

C1 = QVector3D(5 + range, 1, 1); D1 = QVector3D(5 + range, 1, 3);

tmp << A << B << C << D; obj << tmp; tmp.clear();

tmp << B << C << C1 << B1; obj << tmp; tmp.clear();

tmp << B1 << C1 << D1 << A1; obj << tmp; tmp.clear();

tmp << A1 << D1 << D << A; obj << tmp; tmp.clear();

tmp << A << A1 << B1 << B; obj << tmp; tmp.clear();

tmp << D << D1 << C1 << C; obj << tmp; tmp.clear();

A = QVector3D(5 + range, -1, -1); B = QVector3D(9 + range, -1, -1);

C = QVector3D(9 + range, 5, -1); D = QVector3D(5 + range, 5, -1);

A1 = QVector3D(5 + range, -1, 5); B1 = QVector3D(9 + range, -1, 5);

C1 = QVector3D(9 + range, 5, 5); D1 = QVector3D(5 + range, 5, 5);

tmp << A << B << C << D; obj << tmp; tmp.clear();

tmp << B << C << C1 << B1; obj << tmp; tmp.clear();

tmp << B1 << C1 << D1 << A1; obj << tmp; tmp.clear();

tmp << A1 << D1 << D << A; obj << tmp; tmp.clear();

tmp << A << A1 << B1 << B; obj << tmp; tmp.clear();

tmp << D << D1 << C1 << C; obj << tmp; tmp.clear();

A = QVector3D(9 + range, 3, 3); B = QVector3D(9 + range, 3, 1);

C = QVector3D(10 + range \* 2, 3, 1); D = QVector3D(10 + range \* 2, 3, 3);

A1 = QVector3D(9 + range, 1, 3); B1 = QVector3D(9 + range, 1, 1);

C1 = QVector3D(10 + range \* 2, 1, 1); D1 = QVector3D(10 + range \* 2, 1, 3);

tmp << A << B << C << D; obj << tmp; tmp.clear();

tmp << B << C << C1 << B1; obj << tmp; tmp.clear();

tmp << B1 << C1 << D1 << A1; obj << tmp; tmp.clear();

tmp << A1 << D1 << D << A; obj << tmp; tmp.clear();

tmp << A << A1 << B1 << B; obj << tmp; tmp.clear();

tmp << D << D1 << C1 << C; obj << tmp; tmp.clear();

A = QVector3D(10 + range \* 2, 0, 0); B = QVector3D(14 + range \* 2, 0, 0);

C = QVector3D(14 + range \* 2, 4, 0); D = QVector3D(10 + range \* 2, 4, 0);

A1 = QVector3D(10 + range \* 2, 0, 4); B1 = QVector3D(14 + range \* 2, 0, 4);

C1 = QVector3D(14 + range \* 2, 4, 4); D1 = QVector3D(10 + range \* 2, 4, 4);

tmp << A << B << C << D; obj << tmp; tmp.clear();

tmp << B << C << C1 << B1; obj << tmp; tmp.clear();

tmp << B1 << C1 << D1 << A1; obj << tmp; tmp.clear();

tmp << A1 << D1 << D << A; obj << tmp; tmp.clear();

tmp << A << A1 << B1 << B; obj << tmp; tmp.clear();

tmp << D << D1 << C1 << C; obj << tmp; tmp.clear();

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

obj[i].setNumber(i);

}

void Object::**setPolygonColor**(int i, QColor color){

obj[i].setColor(color);

obj[i].setClicked();

}

Содержимое файла custompolygon.h

#ifndef CUSTOMPOLYGON\_H

#define CUSTOMPOLYGON\_H

#include <QVector>

#include <QVector3D>

#include <QColor>

*class* **CustomPolygon**{

QVector <QVector3D> polygon;

QColor color;

float depth;

bool isClicked;

int number;

*public*:

**CustomPolygon**();

**CustomPolygon**(QVector<QVector3D>);

QVector <QVector3D> **getPolygon**();

float **getDepth**();

QColor **getColor**();

void **setVector**(QVector3D v, int pos);

void **setColor**(QColor);

void **setClicked**();

void **setNumber**(int);

int **getNumber**();

int **size**();

bool **isColored**(){*return* isClicked;}

QVector3D *operator* [](int i);

};

#endif *//* *CUSTOMPOLYGON\_H*

Содержимое файла custompolygon.cpp

#include "custompolygon.h"

CustomPolygon::**CustomPolygon**(){

}

CustomPolygon::**CustomPolygon**(QVector<QVector3D> newPolygon){

polygon = newPolygon;

depth = 0;

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

depth += newPolygon[i].z();

}

color = Qt::*red*;

color.setAlpha(150);

isClicked = *false*;

}

QVector <QVector3D> CustomPolygon::**getPolygon**(){

*return* polygon;

}

float CustomPolygon::**getDepth**(){

depth = 0;

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

depth += polygon[i].z();

}

*return* depth;

}

void CustomPolygon::**setVector**(QVector3D v, int pos){

polygon[pos] = v;

}

QColor CustomPolygon::**getColor**(){

*return* color;

}

void CustomPolygon::**setColor**(QColor c){

color = c;

}

int CustomPolygon::**size**(){

*return* polygon.size();

}

QVector3D CustomPolygon::*operator* [] (int i){

*return* polygon[i];

}

void CustomPolygon::**setClicked**(){

isClicked = *true*;

}

void CustomPolygon::**setNumber**(int n){

number = n;

}

int CustomPolygon::**getNumber**(){

*return* number;

}

**Результат работы программы:**

