**МИНИСТЕРСТВА НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**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

Тема: «Алгоритм сортировки граней»

Выполнила: ст. группы ПВ-31

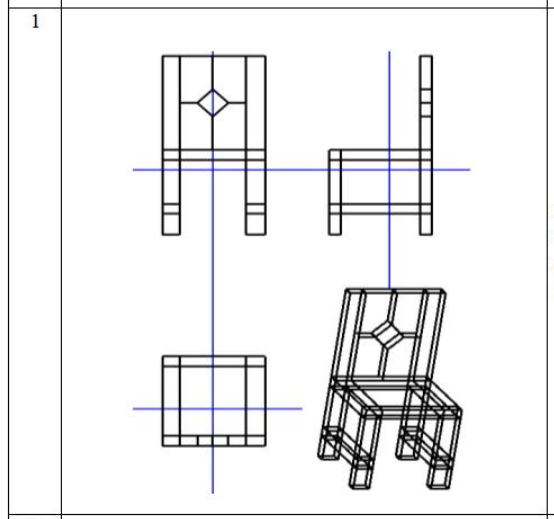
Донцов А.А.

Проверил: Осипов О. В.

Белгород 2019

**Цель работы:** реализация простейшего алгоритма сортировки многоугольников по глубине   
Задания для выполнения к работе

**Вариант 1**



Main

#include "mainwindow.h"

#include <QApplication>

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

{

QApplication a(*argc*, argv);

MainWindow w;

w.show();

return a.exec();

}

Figure.h

#ifndef FIGURE\_H

#define FIGURE\_H

#include <mainwindow.h>

#include <QVector>

#include <QVector3D>

#include <QMatrix4x4>

#include <map>

#include <iterator>

#include "matr\_pr.h"

using namespace std;

class **MyPoligon**{

public:

QPolygonF a;

int color;

**MyPoligon**(QPolygonF a1, int color1):a(a1) {

color = color1;

}

};

class **Figure**{

public:

int color;

//вектор точек

QVector<QVector3D> points;

//вектор цветов

QVector<int> colors;

//Вектор полигонов для фигуры

map<double, MyPoligon> polygones;

//Вектор количества точек в полигонах

QVector<int> sizepolygones;

//Вектор точек для полигонов фигуры

QVector <vector<int>> dotsPolygones;

//Вектор координат Z

QVector <double> zpVector;

//Вектор цветов для каждого полигона

QVector<QColor> vectorColor;

//Вектор состояний - нажат, не нажат

QVector <bool> isPressedVector;

**Figure**();

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

void **addPolygon**(int n, QVector3D \*arr, int color);

void **rewritePolygon**();

void **pushPolygon**();

~**Figure**() {};

};

#endif // FIGURE\_H

Figure.cpp

#include "figure.h"

Figure::**Figure**()

{

//сиденье

QVector3D a[4];

a[0].setX(-2); a[0].setY(0.25); a[0].setZ(-2);

a[1].setX(-2); a[1].setY(-0.25); a[1].setZ(-2);

a[2].setX(2); a[2].setY(-0.25); a[2].setZ(-2);

a[3].setX(2); a[3].setY(0.25); a[3].setZ(-2);

addPolygon(4, a, 0);

a[0].setX(-2); a[0].setY(0.25); a[0].setZ(2);

a[1].setX(-2); a[1].setY(-0.25); a[1].setZ(2);

a[2].setX(2); a[2].setY(-0.25); a[2].setZ(2);

a[3].setX(2); a[3].setY(0.25); a[3].setZ(2);

addPolygon(4, a, 1);

a[0].setX(-2); a[0].setY(0.25); a[0].setZ(-2);

a[1].setX(-2); a[1].setY(-0.25); a[1].setZ(-2);

a[3].setX(-2); a[3].setY(0.25); a[3].setZ(2);

a[2].setX(-2); a[2].setY(-0.25); a[2].setZ(2);

addPolygon(4, a, 2);

a[0].setX(2); a[0].setY(-0.25); a[0].setZ(-2);

a[1].setX(2); a[1].setY(0.25); a[1].setZ(-2);

a[3].setX(2); a[3].setY(-0.25); a[3].setZ(2);

a[2].setX(2); a[2].setY(0.25); a[2].setZ(2);

addPolygon(4, a, 0);

a[0].setX(-2); a[0].setY(-0.25); a[0].setZ(-2);

a[1].setX(2); a[1].setY(-0.25); a[1].setZ(-2);

a[2].setX(2); a[2].setY(-0.25); a[2].setZ(2);

a[3].setX(-2); a[3].setY(-0.25); a[3].setZ(2);

addPolygon(4, a, 1);

a[0].setX(-2); a[0].setY(0.25); a[0].setZ(-2);

a[1].setX(2); a[1].setY(0.25); a[1].setZ(-2);

a[2].setX(2); a[2].setY(0.25); a[2].setZ(2);

a[3].setX(-2); a[3].setY(0.25); a[3].setZ(2);

addPolygon(4, a, 2);

//ножки

//1

a[0].setX(-2); a[0].setY(0.25); a[0].setZ(2);

a[1].setX(-2); a[1].setY(3.5); a[1].setZ(2);

a[2].setX(-2); a[2].setY(3.5); a[2].setZ(1.5);

a[3].setX(-2); a[3].setY(0.25); a[3].setZ(1.5);

addPolygon(4, a, 0);

a[0].setX(-1.5); a[0].setY(0.25); a[0].setZ(2);

a[1].setX(-1.5); a[1].setY(3.5); a[1].setZ(2);

a[2].setX(-1.5); a[2].setY(3.5); a[2].setZ(1.5);

a[3].setX(-1.5); a[3].setY(0.25); a[3].setZ(1.5);

addPolygon(4, a, 1);

a[0].setX(-2); a[0].setY(0.25); a[0].setZ(2);

a[1].setX(-2); a[1].setY(3.5); a[1].setZ(2);

a[2].setX(-1.5); a[2].setY(3.5); a[2].setZ(2);

a[3].setX(-1.5); a[3].setY(0.25); a[3].setZ(2);

addPolygon(4, a, 2);

a[0].setX(-2); a[0].setY(0.25); a[0].setZ(1.5);

a[1].setX(-2); a[1].setY(3.5); a[1].setZ(1.5);

a[2].setX(-1.5); a[2].setY(3.5); a[2].setZ(1.5);

a[3].setX(-1.5); a[3].setY(0.25); a[3].setZ(1.5);

addPolygon(4, a, 0);

a[0].setX(-2); a[0].setY(3.5); a[0].setZ(2);

a[1].setX(-1.5); a[1].setY(3.5); a[1].setZ(2);

a[2].setX(-1.5); a[2].setY(3.5); a[2].setZ(1.5);

a[3].setX(-2); a[3].setY(3.5); a[3].setZ(1.5);

addPolygon(4, a, 1);

//2

a[0].setX(2); a[0].setY(0.25); a[0].setZ(2);

a[1].setX(2); a[1].setY(3.5); a[1].setZ(2);

a[2].setX(2); a[2].setY(3.5); a[2].setZ(1.5);

a[3].setX(2); a[3].setY(0.25); a[3].setZ(1.5);

addPolygon(4, a, 0);

a[0].setX(1.5); a[0].setY(0.25); a[0].setZ(2);

a[1].setX(1.5); a[1].setY(3.5); a[1].setZ(2);

a[2].setX(1.5); a[2].setY(3.5); a[2].setZ(1.5);

a[3].setX(1.5); a[3].setY(0.25); a[3].setZ(1.5);

addPolygon(4, a, 1);

a[0].setX(2); a[0].setY(0.25); a[0].setZ(2);

a[1].setX(2); a[1].setY(3.5); a[1].setZ(2);

a[2].setX(1.5); a[2].setY(3.5); a[2].setZ(2);

a[3].setX(1.5); a[3].setY(0.25); a[3].setZ(2);

addPolygon(4, a, 2);

a[0].setX(2); a[0].setY(0.25); a[0].setZ(1.5);

a[1].setX(2); a[1].setY(3.5); a[1].setZ(1.5);

a[2].setX(1.5); a[2].setY(3.5); a[2].setZ(1.5);

a[3].setX(1.5); a[3].setY(0.25); a[3].setZ(1.5);

addPolygon(4, a, 0);

a[0].setX(2); a[0].setY(3.5); a[0].setZ(2);

a[1].setX(1.5); a[1].setY(3.5); a[1].setZ(2);

a[2].setX(1.5); a[2].setY(3.5); a[2].setZ(1.5);

a[3].setX(2); a[3].setY(3.5); a[3].setZ(1.5);

addPolygon(4, a, 1);

//3

a[0].setX(2); a[0].setY(0.25); a[0].setZ(-2);

a[1].setX(2); a[1].setY(3.5); a[1].setZ(-2);

a[2].setX(2); a[2].setY(3.5); a[2].setZ(-1.5);

a[3].setX(2); a[3].setY(0.25); a[3].setZ(-1.5);

addPolygon(4, a, 0);

a[0].setX(1.5); a[0].setY(0.25); a[0].setZ(-2);

a[1].setX(1.5); a[1].setY(3.5); a[1].setZ(-2);

a[2].setX(1.5); a[2].setY(3.5); a[2].setZ(-1.5);

a[3].setX(1.5); a[3].setY(0.25); a[3].setZ(-1.5);

addPolygon(4, a, 1);

a[0].setX(2); a[0].setY(0.25); a[0].setZ(-2);

a[1].setX(2); a[1].setY(3.5); a[1].setZ(-2);

a[2].setX(1.5); a[2].setY(3.5); a[2].setZ(-2);

a[3].setX(1.5); a[3].setY(0.25); a[3].setZ(-2);

addPolygon(4, a, 2);

a[0].setX(2); a[0].setY(0.25); a[0].setZ(-1.5);

a[1].setX(2); a[1].setY(3.5); a[1].setZ(-1.5);

a[2].setX(1.5); a[2].setY(3.5); a[2].setZ(-1.5);

a[3].setX(1.5); a[3].setY(0.25); a[3].setZ(-1.5);

addPolygon(4, a, 0);

a[0].setX(2); a[0].setY(3.5); a[0].setZ(-2);

a[1].setX(1.5); a[1].setY(3.5); a[1].setZ(-2);

a[2].setX(1.5); a[2].setY(3.5); a[2].setZ(-1.5);

a[3].setX(2); a[3].setY(3.5); a[3].setZ(-1.5);

addPolygon(4, a, 1);

//4

//1

a[0].setX(-2); a[0].setY(0.25); a[0].setZ(-2);

a[1].setX(-2); a[1].setY(3.5); a[1].setZ(-2);

a[2].setX(-2); a[2].setY(3.5); a[2].setZ(-1.5);

a[3].setX(-2); a[3].setY(0.25); a[3].setZ(-1.5);

addPolygon(4, a, 2);

a[0].setX(-1.5); a[0].setY(0.25); a[0].setZ(-2);

a[1].setX(-1.5); a[1].setY(3.5); a[1].setZ(-2);

a[2].setX(-1.5); a[2].setY(3.5); a[2].setZ(-1.5);

a[3].setX(-1.5); a[3].setY(0.25); a[3].setZ(-1.5);

addPolygon(4, a, 0);

a[0].setX(-2); a[0].setY(0.25); a[0].setZ(-2);

a[1].setX(-2); a[1].setY(3.5); a[1].setZ(-2);

a[2].setX(-1.5); a[2].setY(3.5); a[2].setZ(-2);

a[3].setX(-1.5); a[3].setY(0.25); a[3].setZ(-2);

addPolygon(4, a, 1);

a[0].setX(-2); a[0].setY(0.25); a[0].setZ(-1.5);

a[1].setX(-2); a[1].setY(3.5); a[1].setZ(-1.5);

a[2].setX(-1.5); a[2].setY(3.5); a[2].setZ(-1.5);

a[3].setX(-1.5); a[3].setY(0.25); a[3].setZ(-1.5);

addPolygon(4, a, 2);

a[0].setX(-2); a[0].setY(3.5); a[0].setZ(-2);

a[1].setX(-1.5); a[1].setY(3.5); a[1].setZ(-2);

a[2].setX(-1.5); a[2].setY(3.5); a[2].setZ(-1.5);

a[3].setX(-2); a[3].setY(3.5); a[3].setZ(-1.5);

addPolygon(4, a, 0);

//спинка

a[0].setX(-2); a[0].setY(-0.25); a[0].setZ(-2);

a[1].setX(-2); a[1].setY(-3.5); a[1].setZ(-2);

a[2].setX(2); a[2].setY(-3.5); a[2].setZ(-2);

a[3].setX(2); a[3].setY(-0.25); a[3].setZ(-2);

addPolygon(4, a, 0);

a[0].setX(-2); a[0].setY(-0.25); a[0].setZ(-1.75);

a[1].setX(-2); a[1].setY(-3.5); a[1].setZ(-1.75);

a[2].setX(2); a[2].setY(-3.5); a[2].setZ(-1.75);

a[3].setX(2); a[3].setY(-0.25); a[3].setZ(-1.75);

addPolygon(4, a, 1);

a[0].setX(-2); a[0].setY(-0.25); a[0].setZ(-1.75);

a[1].setX(-2); a[1].setY(-3.5); a[1].setZ(-1.75);

a[2].setX(-2); a[2].setY(-3.5); a[2].setZ(-2);

a[3].setX(-2); a[3].setY(-0.25); a[3].setZ(-2);

addPolygon(4, a, 2);

a[0].setX(2); a[0].setY(-0.25); a[0].setZ(-1.75);

a[1].setX(2); a[1].setY(-3.5); a[1].setZ(-1.75);

a[2].setX(2); a[2].setY(-3.5); a[2].setZ(-2);

a[3].setX(2); a[3].setY(-0.25); a[3].setZ(-2);

addPolygon(4, a, 0);

a[0].setX(2); a[0].setY(-3.5); a[0].setZ(-2);

a[1].setX(2); a[1].setY(-3.5); a[1].setZ(-1.75);

a[2].setX(-2); a[2].setY(-3.5); a[2].setZ(-1.75);

a[3].setX(-2); a[3].setY(-3.5); a[3].setZ(-2);

addPolygon(4, a, 1);

}

void Figure::**addPolygon**(int n, QVector3D \*arr, int color){

sizepolygones.push\_back(n);

QPolygonF a;

double z = 0;

for(int i = 0; i < n; i++){

points.push\_back(arr[i]);

a << QPointF(arr[i].x(), arr[i].y());

z += arr[i].z();

}

colors.push\_back(color);

polygones.insert(make\_pair(*z*, MyPoligon(a,color)));

}

void Figure::**rewritePolygon**() {

polygones.clear();

int index = 0;

double z = 0;

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

z = 0;

QPolygonF a;

for(int j = 0; j < sizepolygones[i]; j++){

a << QPointF(points[index].x(), points[index].y());

z += points[index].z();

index++;

}

polygones.insert(make\_pair(*z*, MyPoligon(a,colors[i])));

}

}

void Figure::**draw**(QPainter \*painter)

{

painter->setPen(QPen(Qt::black, 2, Qt::SolidLine, Qt::RoundCap));

map<double,MyPoligon>::iterator it;

it = polygones.begin();

while(it != polygones.end()){

if(it->second.color == 0)

painter->setBrush(QColor(0, 255, 0, 150));

if(it->second.color == 1)

painter->setBrush(QColor(255, 0, 0, 150));

if(it->second.color == 2)

painter->setBrush(QColor(0, 0, 255, 150));

painter->drawPolygon(it->second.a);

it++;

}

}

Mainwindow.h

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <QMouseEvent>

#include <QPainter>

#include "figure.h"

#include "matr\_pr.h"

QT\_BEGIN\_NAMESPACE

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

QT\_END\_NAMESPACE

class **MainWindow** : public QMainWindow

{

Q\_OBJECT

private slots:

void **newProection\_pressed**();

public:

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

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

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

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

QMatrix4x4 R1;

int wheelCoeficient;

void ***wheelEvent***(QWheelEvent\* events);

~***MainWindow***();

private:

Ui::MainWindow \*ui;

//Повороты по осям

float xAxisRotation;

float yAxisRotation;

int numberProjection;

//Координаты предыдущей точки курсора мыши

QPointF previousPoint;

};

#endif // MAINWINDOW\_H

Mainwindow.cpp

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include <math.h>

MainWindow::**MainWindow**(QWidget \*parent)

: QMainWindow(parent)

, ui(new Ui::MainWindow)

{

setMinimumWidth(320);

numberProjection = 0;

xAxisRotation = 0;

yAxisRotation = 0;

wheelCoeficient = 0;

R1 = QMatrix4x4();

ui->setupUi(this);

}

void MainWindow::***paintEvent***(QPaintEvent \*event)

{

QPainter painter(this);

painter.setPen((QPen(Qt::black, 2, Qt::SolidLine, Qt::FlatCap)));

double scale = 1;

if (width()/2 < height())

scale = width()/10.0;

else

scale = height()/10.0;

Figure fg;

QMatrix4x4 matr;

matr.translate(width() / 2, height() / 2, 0);

matr = matr \* Matrix::prj\_c(scale\*15);

matr.translate(0, 0, wheelCoeficient);

matr = matr \* R1.transposed();

matr.scale(scale, scale, scale);

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

{

QVector3D V(fg.points[i].x(), fg.points[i].y(), fg.points[i].z());

V = matr \* V;

fg.points[i] = QVector3D(V.x(), V.y(), V.z());

}

fg.rewritePolygon();

fg.draw(&painter);

}

void MainWindow::**newProection\_pressed**()

{

repaint();

}

//Нажатие мыши

void MainWindow::***mousePressEvent***(QMouseEvent \*event)

{

previousPoint = event->windowPos();

}

//Смещение мыши относительно нажатия

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

{

float dh = (event->y() - previousPoint.y());

float dw = (event->x() - previousPoint.x());

R1.rotate(-dw, QVector3D(0,1,0));

R1.rotate(dh, QVector3D(1,0,0));

//Обновление данных о предыдущей координате

previousPoint = event->windowPos();

repaint();

}

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

int wheelCoeficientPlus = event -> delta() / 4;

wheelCoeficient += wheelCoeficientPlus;

repaint();

}

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

{

delete ui;

}

Matr\_pr.h

#ifndef MATRIX\_H

#define MATRIX\_H

#include <math.h>

#include <QMatrix4x4>

class **Matrix** {

public:

static QMatrix4x4 **prj**(int t);

static QMatrix4x4 **prj\_c**(double h);

};

#endif // MATRIX\_H

Matr\_pr.cpp

#include "matr\_pr.h"

#include "stdio.h"

QMatrix4x4 Matrix::**prj\_c**(double h)

{

return QMatrix4x4 (1, 0, 0, 0,

0, 1, 0, 0,

0, 0, 1, 0,

0, 0, -1/h, 1);

}

QMatrix4x4 Matrix::**prj**(int t)

{

QMatrix4x4 M;

// 1 тип - Ортографическая проекция с z = 0

if (t == 1)

M = QMatrix4x4(1, 0, 0, 0,

0, 1, 0, 0,

0, 0, 0, 0,

0, 0, 0, 1);

/\*2 тип - Триметрическая проекция (поворот на 30 вокруг Y

поворот на 45 вокруг X и проецирование на Z = 0)\*/

if (t == 2)

M =

QMatrix4x4(sqrt(3)/2, 0, 1/2, 0,

sqrt(2)/4, sqrt(2)/2, -sqrt(6)/4, 0,

0, 0, 0, 0,

0, 0, 0, 1);

//3 тип - Косоугольная Свободная , т.к угол = pi/4

if (t == 3)

M = QMatrix4x4(1, 0, sqrt(2)/2, 0,

0, 1, sqrt(2)/2, 0,

0, 0, 0, 0,

0, 0, 0, 1);

//4 тип - Косоугольная Кабинетная , т.к угол = 1/2 \* pi/4

if (t == 4)

M = QMatrix4x4(1, 0, sqrt(2)/4, 0,

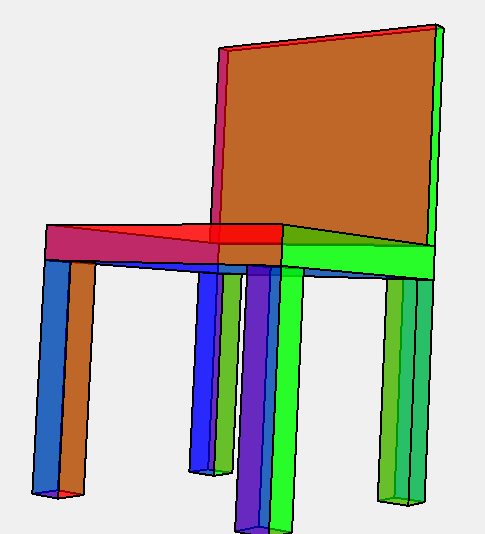
0, 1, sqrt(2)/4, 0,

0, 0, 0, 0,

0, 0, 0, 1);

return M;

}

я