МИНИСТЕРСТВО 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)

Лабораторная работа №5
дисциплина «Компьютерная графика »
по теме «Алгоритм удаления невидимых граней»

Выполнил: студент группы ВТ-31 Макаров Д.С.
Проверил: Осипов О.В.

Белгород 2019

Лабораторная работа №5

«Алгоритм удаления невидимых граней»

Цель работы:Реализация простейшего алгоритма сортировки многоугольников по глубине..

Вариант 9

Порядок выполнения работы

1. Реализовать алгоритм сортировки выпуклых многоугольников по глубине Z, где Z - центр тяжести многоугольника (при помощи qStableSort).

2. На экран вывести только 1 проекцию модели (центральную или ортографическую). Пользователь должен иметь возможность поворачивать фигуру, с использованием мыши или клавиш, удалять и приближать фигуру.

3. Фигура должна состоять из набора небольших граней. Грани должны быть соизмеримы.

4. При выводе изображения предоставить возможность пользователю видеть грани прозрачными. Разработать алгоритм и написать программу для создания трёхмерной сцены, содержащей графические объекты, в соответствии с выбранной предметной областью.

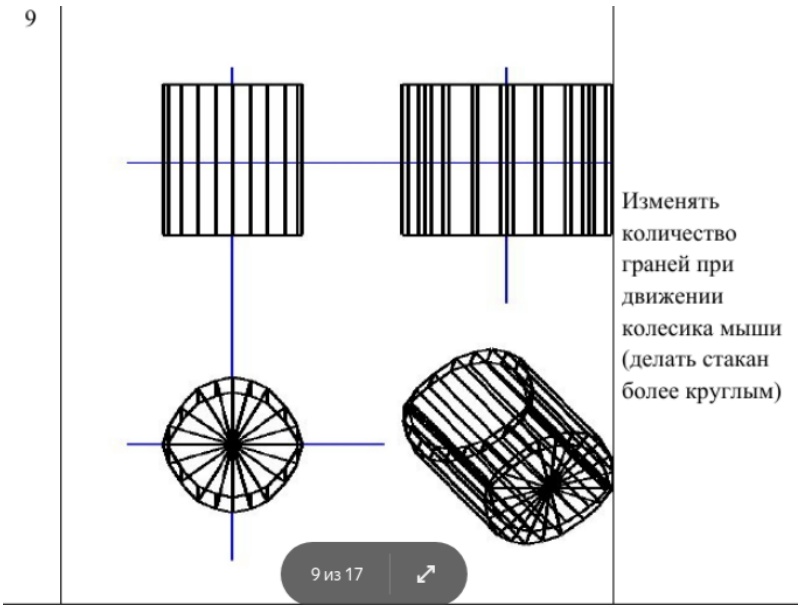

Рис. 1: Задание к работе

Ход работы

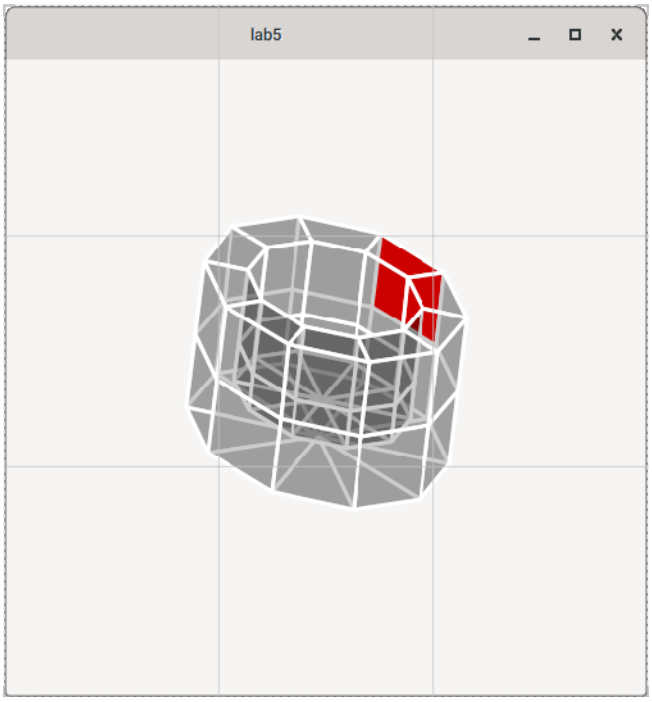

Рис. 2: Пример работы программы

Приложение

Содержимое файла Model3D.cpp

```
#include "Model3D.hpp"

QList<QPolygon> Model3D::listPoly(){
    QList<QPolygon> result;
    for(int i=0,size=_list.size();i<size;i++){
        result.append(_list[i].qPolygon());
    };
    return result;
};
void Model3D::multWithAfin(QMatrix4x4 matr){
    QList<Polygon3D> tempList;
    for(int i = 0,size = renderList.size();i<size;i++){
        tempList.append(renderList[i]*matr);
    };
    renderList = tempList;
};

void Model3D::rotate(float angle,QVector3D vector){
    QMatrix4x4 r;
    r.setToIdentity();
    r.rotate(angle,vector);
    QList<Polygon3D> tempList;
    for(int i = 0,size = renderList.size();i<size;i++){
        tempList.append(renderList[i]*r);
    };
    renderList = tempList;
};
void Model3D::translate(QVector3D vector){
    QMatrix4x4 t;
    t.setToIdentity();
    t.translate(vector);
    QList<Polygon3D> tempList;
    for(int i = 0,size = renderList.size();i<size;i++){
        tempList.append(renderList[i]*t);
    };
    renderList = tempList;
};
void Model3D::scale(QVector3D vector){
    QMatrix4x4 s;
    s.setToIdentity();
    s.scale(vector);
    QList<Polygon3D> tempList;
    for(int i = 0,size = renderList.size();i<size;i++){
        tempList.append(renderList[i]*s);
    };
    renderList = tempList;
};

Model3D::Model3D(){

};

void Model3D::resetAfin(){
    renderList.clear();
    for(int i = 0,size = _list.size();i<size;i++){
        Polygon3D tempPoly = _list[i];
```

```
        tempPoly.setHighlight(_list[i].highlight());
        renderList.append(tempPoly);
    };
};
```

Содержимое файла main.cpp

```
#include <QtWidgets>
#include "Draw.hpp"
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    DrawArea* l = new DrawArea;

    QWidget *settings = new QWidget;
    QVBoxLayout *layout = new QVBoxLayout;
    QLabel *sName =new QLabel("Количество секторов:");
    QLabel *bSelect =new QLabel("Тип проектирования:");
    QSpinBox *s = new QSpinBox;
    QComboBox *b = new QComboBox;

    b->addItem(QString("Центральное проектирование"));
    b->addItem(QString("Косоугольное кабинетное проектирование"));
    b->addItem(QString("Косоугольное свободное проектирование"));
    b->addItem(QString("Параллельное проектирование"));
    s->setRange(3,4000);

    QObject::connect(s,SIGNAL(valueChanged(int)),l,SLOT(reGenModel(int)));
    QObject::connect(b,SIGNAL(currentIndexChanged(int)),l,SLOT(selectProjection(int)));

    layout->addWidget(sName);
    layout->addWidget(s);
    layout->addWidget(bSelect);
    layout->addWidget(b);
    settings->setLayout(layout);
    settings->show();
    l->show();
    return a.exec();
}
```

Содержимое файла Model3D.hpp

```
#include "Polygon3D.hpp"
#include <QtWidgets>
#pragma once



class Model3D{
    private:
        QList<Polygon3D> _list;
        QList<Polygon3D> renderList;

    public:
        Model3D();
        void operator() (QPainter *painter){
            for(int i = 0,size = renderList.size();i<size;i++){
                (renderList[i])(painter);
            };
        }
```

```
void operator << (Polygon3D vector){
    _list.append(vector);
}

Model3D& operator= (Model3D right){
    if(this == &right){
        return *this;
    }
    this->_list = right._list;
    this->renderList=right.renderList;
    return *this;
}
Model3D operator * (QMatrix4x4 matr){
    QList<Polygon3D> tempList;
    for(int i = 0,size = renderList.size();i<size;i++){
        tempList.append(renderList[i]*matr);
    };
    renderList = tempList;
    return *this;
}
void multWithAfin(QMatrix4x4 matr);
friend QDebug operator<<(QDebug stream, Model3D model){
    stream<<"Текущие полигоны для рендера\n";
    for(int i = 0,size = model.renderList.size();i<size;i++){
        stream<<model.renderList[i];
    };
    return stream;
}
QList<QPolygon> listPoly();
void rotate(float angle,QVector3D vector);
void translate(QVector3D vector);
void scale(QVector3D vector);
void resetAfin();
void sortOriginModel(){
    qStableSort(_list.begin(),_list.end());
};
void sortRenderModel(){
    qSort(renderList.begin(),renderList.end());
};

void selectAndColorizePolygon(QPoint point){
    bool flag = true;
    auto i = renderList.rbegin();
    for (;(i!=renderList.rend())&&flag;i++){
        if((*i).containPoint(point)){
            auto j=_list.rbegin();
            for(;j!=_list.rend();j++){
                if((*j).num()==(*i).num()){
                    qDebug() <<"render list: "<< (*i);
                    qDebug() <<"_list: "<< (*j);
                    (*i).setHighlight(!(*i).highlight());
                    (*j).setHighlight(!(*j).highlight());
                    qDebug() <<"render list: "<< (*i);
                    qDebug() <<"_list: "<< (*j);
                }
            }
            flag = false;
        }
    }
```

```
        };
            /*
                Поиск полигона которому пренадлежит точка в списке полигонов для рендера.
                Поиск полигона в списке исходных с номером совпадающим с полигоном для
↪   рендера.
                Смена цвета.

            */

};
```

Содержимое файла Draw.cpp

```
#include "Draw.hpp"
#include "Model3D.hpp"
#include "Polygon3D.hpp"
#include <float.h>

int mouseWheelCount = 200;
float Ccentral = 200+mouseWheelCount;
float Cparallel = FLT_MAX;
float KCab = qCos(M_PI_4)/2;
float KFree = qCos(M_PI_4);

QColor polyColor= QColor(0,0,0,125);
QColor clickedPolyColor= QColor(255,0,0);

//матрицы проектирования
QMatrix4x4 centralMatr = {1,0,0,0,0,1,0,0,0,0,0,0,0,0,-1/Ccentral,1};
QMatrix4x4 axMatrCab = {1,0,KCab,0,0,1,KCab,0,0,0,0,0,0,0,0,1};
QMatrix4x4 axMatrFree = {1,0,KFree,0,0,1,KFree,0,0,0,0,0,0,0,0,1};
QMatrix4x4 parallelMatr = {1,0,0,0,0,1,0,0,0,0,0,0,0,0,-1/Cparallel,1};
QMatrix4x4 ortoMatr = {1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1};

Model3D genTaskModel(int n){
    Model3D resultModel;
    qreal step = 360.0/n;
    int i=0;
    QVector4D c1(0,0,0,1);
    QVector4D c2(0,0,1,1);
    QVector4D prevB1,b1(0,3,0,1);
    QVector4D prevB2,b2(0,2,1,1);
    QVector4D prevK1,k1(0,3,2,1);
    QVector4D prevK2,k2(0,2,2,1);
    QVector4D prevT1,t1(0,3,4,1);
    QVector4D prevT2,t2(0,2,4,1);

    int count=0;
    QMatrix4x4 rotateM;
    Polygon3D tempPoly = Polygon3D(count);
    rotateM.setToIdentity();
    rotateM.rotate(step,QVector3D(0,0,1));
    do{

        prevB1 = b1;
        prevB2 = b2;
        prevT1 = t1;
        prevT2 = t2;
        prevK1 = k1;
        prevK2 = k2;
```

```
b1 = rotateM*b1;
b2 = rotateM*b2;
t1 = rotateM*t1;
t2 = rotateM*t2;
k1 = rotateM*k1;
k2 = rotateM*k2;

//Верхняя кромка
tempPoly = Polygon3D(count);
count++;
tempPoly.setNum(count);
tempPoly<<prevT1;
tempPoly<<prevT2;
tempPoly<<t2;
tempPoly<<t1;
resultModel<<tempPoly;

//Внешняя грань
tempPoly = Polygon3D(count);
count++;
tempPoly.setNum(count);
tempPoly<<prevK1;
tempPoly<<prevB1;
tempPoly<<b1;
tempPoly<<k1;
resultModel<<tempPoly;

tempPoly = Polygon3D(count);
count++;
tempPoly.setNum(count);
tempPoly<<t1;
tempPoly<<k1;
tempPoly<<prevK1;
tempPoly<<prevT1;
resultModel<<tempPoly;

//Внутренняя грань
tempPoly = Polygon3D(count);
count++;
tempPoly.setNum(count);
tempPoly<<k2;
tempPoly<<prevK2;
tempPoly<<prevB2;
tempPoly<<b2;
resultModel<<tempPoly;

tempPoly = Polygon3D(count);
count++;
tempPoly.setNum(count);
tempPoly<<prevK2;
tempPoly<<prevT2;
tempPoly<<t2;
tempPoly<<k2;
resultModel<<tempPoly;

tempPoly = Polygon3D(count);
count++;
tempPoly.setNum(count);
tempPoly<<prevB2;
```

```
        tempPoly<<b2;
        tempPoly<<c2;
        resultModel<<tempPoly;

        tempPoly = Polygon3D(count);
        count++;
        tempPoly.setNum(count);
        tempPoly<<prevB1;
        tempPoly<<b1;
        tempPoly<<c1;
        resultModel<<tempPoly;

        i++;
    }while(i<n);
    //resultModel<<outerBottom;
    //resultModel<<innerBottom;
    return resultModel;
};


DrawArea::DrawArea(){
    mouseRotateMatr.setToIdentity();
    projectionMatr = axMatrCab;
    QPoint lastPos= QPoint(0,0);
    resize(500,500);
    model=genTaskModel(10);
};

void DrawArea::reGenModel(int n){
    model=genTaskModel(n);
    this->update();
}

void DrawArea::selectProjection(int index){
    switch(index){
        case 0:
            projectionMatr = centralMatr;
        break;
        case 1:
            projectionMatr = axMatrCab;
        break;
        case 2:
            projectionMatr = axMatrFree;
        break;
        case 3:
            projectionMatr = parallelMatr;
        break;
    };
    this->update();
};

void DrawArea::paintEvent(QPaintEvent* event){
    //проверка полей
    if (this->width() < 40 || this->height() < 40) return;
    QPainter *painter = new QPainter(this);
    painter->setRenderHint(QPainter::HighQualityAntialiasing);
    Ccentral = 200+mouseWheelCount;
    centralMatr = {1,0,0,0,0,1,0,0,0,0,0,0,0,0,-1/Ccentral,1};
    //вычисление коэффицента маштабирования
    qreal scale_mult;
```

```
    if(this->height()>this->width()){
        scale_mult = this->width()/15;
    }
    else{
        scale_mult = this->height()/15;
    };
    QPoint p1 = QPoint(this->width()/2,2*this->height()/3);

    //создания пера для границ
    QPen borderPen;
    borderPen.setWidth(3);
    borderPen.setColor(Qt::white);
    painter->setPen(borderPen);

    //проектирование модели
    model.resetAfin();
    model.rotate(90,QVector3D(1,0,0));
    model.translate(QVector3D(0,2,1));
    model.multWithAfin(mouseRotateMatr.transposed());
    model.translate(QVector3D(0,-2,-1));
    model.scale(QVector3D(scale_mult,scale_mult,scale_mult));
    model.translate(QVector3D(p1.x(),p1.y(),1));
    model.sortRenderModel();
    model.multWithAfin(projectionMatr);
    if(needColorize){
    model.selectAndColorizePolygon(colorizePoint);
    needColorize = false;
    };
    model(painter);

    painter->end();
    event->accept();
};

void DrawArea::wheelEvent(QWheelEvent *event)
{
    QPoint numPixels = event->pixelDelta();
    mouseWheelCount += numPixels.x();
    this->update();
    event->accept();
}

void DrawArea::mousePressEvent(QMouseEvent *event){
    lastPos = event->pos();
};

void DrawArea::mouseDoubleClickEvent(QMouseEvent *m_event){
    needColorize = true;
    colorizePoint = m_event->pos();
    qDebug() <<m_event->pos();
    this->update();
    m_event->accept();
};


void DrawArea::mouseMoveEvent(QMouseEvent *event){
    double k=10;
    QPoint dp =  event->pos() - lastPos;
    mouseRotateMatr.rotate(-dp.x()/k,QVector3D(0,1,0));
    mouseRotateMatr.rotate(dp.y()/k,QVector3D(1,0,0));
```

```
    lastPos = event->pos();
    this->update();
    event->accept();
}
```

Содержимое файла Draw.hpp

```
#include <QtWidgets>
#include "Model3D.hpp"
Model3D genTaskModel(int n);

class DrawArea:public QWidget{
    Q_OBJECT
    private:
        QMatrix4x4 mouseRotateMatr;
        Model3D model;
        QMatrix4x4 projectionMatr;
        QPoint lastPos;
        bool needColorize = false;
        QPoint colorizePoint;
    public:
        DrawArea();
        void paintEvent(QPaintEvent *event);
        void wheelEvent(QWheelEvent *event);
        void mouseMoveEvent(QMouseEvent *m_event);
        void mouseDoubleClickEvent(QMouseEvent *m_event);
        void mousePressEvent(QMouseEvent *event);
    public slots:
        void reGenModel(int n);
        void selectProjection(int index);
};
```

Содержимое файла Polygon3D.cpp

```
#include "Polygon3D.hpp"

Polygon3D::Polygon3D(int num){
    number = num;
};


QPolygon Polygon3D::qPolygon(){
    QPolygon result;
    for(int i=0;i<_list.size();i++){
        result<<vectToPoint(_list[i]);
    };
    return result;
};

QPoint vectToPoint(QVector4D vect){
    QPoint result;
    result.setX(vect.x()/vect.w());
    result.setY(vect.y()/vect.w());
    return result;
};
```

Содержимое файла Polygon3D.hpp

```
#pragma once
#include <QtWidgets>
```

```
class Polygon3D{
    private:
        QList<QVector4D> _list;
        int number;
        bool highlight_flag = false;
    public:
        QList<QVector4D> vectorList(){
            return _list;
        };
        void setHighlight(bool flag){
            this->highlight_flag = flag;
        };
        bool highlight(){
            return highlight_flag;
        };
        QList<QVector4D> setVectorList(QList<QVector4D> points){
            this->_list=points;
        };
        int num(){
            return number;
        };
        void setNum(int num){
            number = num;
        }
        Polygon3D(int num);
        void operator() (QPainter *painter){
            painter->save();
            if(highlight_flag){
                painter->setBrush(QColor(255,0,0));
            }else{
                painter->setBrush(QColor(0,0,0,50));
            };
            painter->drawPolygon(qPolygon());
            painter->restore();
        }
        void operator << (QVector4D vect){
            _list.append(vect);
        }
        Polygon3D& operator= (Polygon3D right){
            if(this == &right){
                return *this;
            }
            this->_list = right._list;
            this->number = right.number;
            this->highlight_flag = right.highlight_flag;
            return *this;
        }
        friend Polygon3D operator *(Polygon3D poly,QMatrix4x4 matr){
            Polygon3D result(poly.number);
            result.setHighlight(poly.highlight());
            for(int i=0,size=poly._list.size();i<size;i++){
                result._list.append(matr*poly._list[i]);
            }
            return result;
        }
        QPolygon qPolygon();
        friend QDebug operator<<(QDebug stream,const Polygon3D poly){
            stream<<poly.number<<" "<<poly.highlight_flag;
            stream<<"\n";
```

```
            return stream;
        }
        double baricenter(){
            double result = 0.0;
            double size = _list.size();
            for(int i = 0;i<size;i++){
                result += _list[i].z();
            }
            return result;
        }
        friend bool operator< (Polygon3D l,Polygon3D r){return
        ↪  l.baricenter()<r.baricenter();};
        bool containPoint(QPoint point){return this->qPolygon().containsPoint(point,
        ↪  Qt::OddEvenFill);};


};

QPoint vectToPoint(QVector4D vect);
```