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 №4
дисциплина «Компьютераная графика »
по теме «Аффинные преобразования в пространстве»

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

Белгород 2019

Лабораторная работа №4

«Аффинные преобразования в пространстве»

Цель работы:получение навыков использования аффинных преобразований в пространстве и создание графического приложения с использованием GDI в среде Qt Creator для визуализации простейших трёхмерных объектов.

Вариант 9

Требования к программе.

1. Окно поделить на 4 части одинаковые части:
2. На верхней левой части должна отображаться фронтальная проекция (вид спереди);
3. Правая верхняя часть – профильная проекция (вид сбоку);
4. Левая нижняя часть должна отображать вид сверху (горизонтальную проекцию);
5. На правой нижней части должна отображаться проекция, вид которой выбирает пользователь:

   - центральная, косоугольная
   - кабинетная,
   - косоугольная
   - свободная,
   - параллельная,
   - ортографическая.

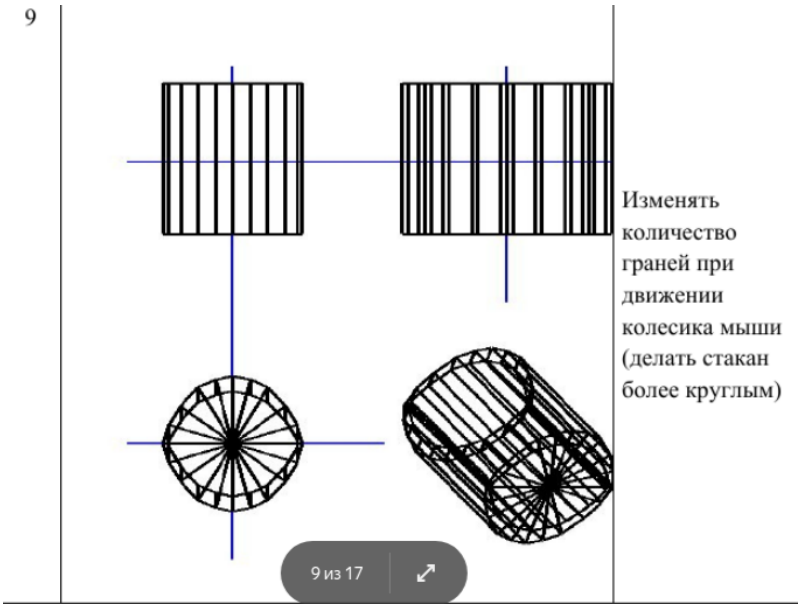

Рис. 1: Задание к работе

Ход работы

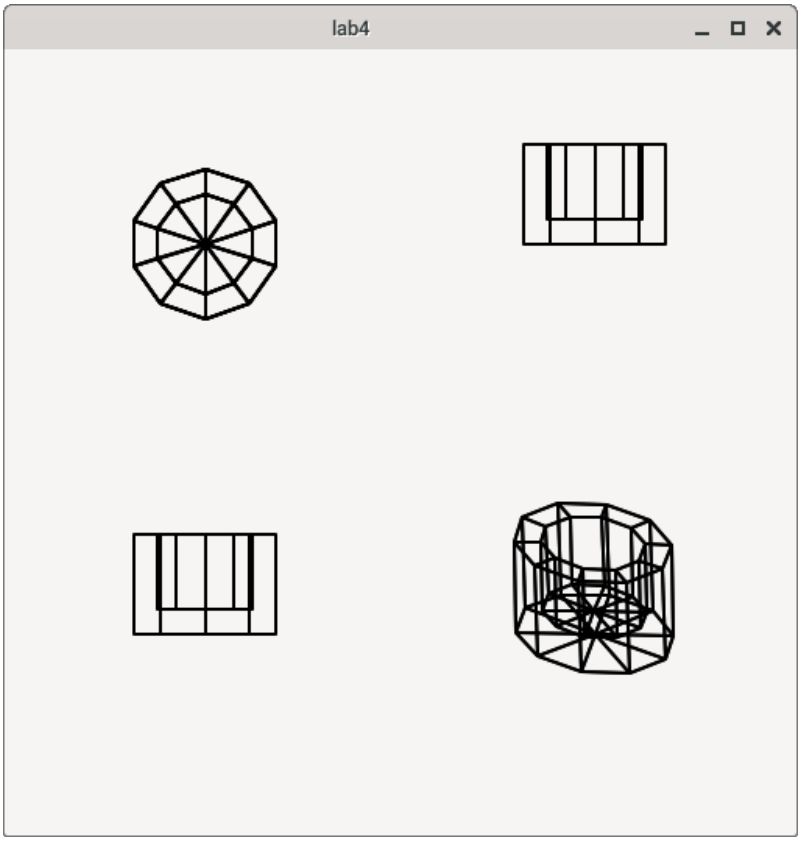

Рис. 2: Пример работы программы

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){
    afinMatr = afinMatr*matr;
};

void Model3D::rotate(float angle,QVector3D vector){
    QMatrix4x4 r;
    r.setToIdentity();
    r.rotate(angle,vector);
    afinMatr = afinMatr*r;
};
void Model3D::translate(QVector3D vector){
    QMatrix4x4 t;
    t.setToIdentity();
    t.translate(vector);
    afinMatr = afinMatr*t;
};
void Model3D::scale(QVector3D vector){
    QMatrix4x4 s;
    s.setToIdentity();
    s.scale(vector);
    afinMatr = afinMatr*s;
};

Model3D::Model3D(){
    afinMatr.setToIdentity();
};

void Model3D::resetAfin(){
    afinMatr.setToIdentity();
};
```

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;
        QMatrix4x4 afinMatr;
    public:
        Model3D();
        void operator() (QPainter *painter){
            for(int i = 0,size = _list.size();i<size;i++){
                (_list[i]*afinMatr)(painter);
            }
        }

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

        Model3D& operator= (Model3D right){
            if(this == &right){
                return *this;
            }
            this->_list = right._list;
            this->afinMatr=right.afinMatr;
            return *this;
        }
        Model3D operator * (QMatrix4x4 matr){
            this->afinMatr*matr;
            return *this;
        }
        void multWithAfin(QMatrix4x4 matr);
        friend QDebug operator<<(QDebug stream,const Model3D model){
            stream <<"Матрица афинных приобразований.\n"
                   << model.afinMatr
                   <<"\nТочки\n";
            for(int i = 0,size = model._list.size();i<size;i++){
                stream<<(model._list[i]*model.afinMatr);
```

```
            }
            return stream;
        }
        QList<QPolygon> listPoly();
        void rotate(float angle,QVector3D vector);
        void translate(QVector3D vector);
        void scale(QVector3D vector);
        void resetAfin();
};
```

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);

//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 prevT1,t1(0,3,4,1);
    QVector4D prevT2,t2(0,2,4,1);
    //Polygon3D outerBottom= Polygon3D(QColor(0,0,0,0));
    //Polygon3D innerBottom= Polygon3D(QColor(0,0,0,0));
    QMatrix4x4 rotateM;
    Polygon3D tempPoly = Polygon3D(QColor(0,0,0,0));
    rotateM.setToIdentity();
    rotateM.rotate(step,QVector3D(0,0,1));
    do{

        prevB1 = b1;
        prevB2 = b2;
        prevT1 = t1;
        prevT2 = t2;

        b1 = rotateM*b1;
        b2 = rotateM*b2;
        t1 = rotateM*t1;
        t2 = rotateM*t2;
        //Верхняя кромка
        tempPoly = Polygon3D(QColor(0,0,0,0));
        tempPoly<<prevT1;
```

```
        tempPoly<<prevT2;
        tempPoly<<t2;
        tempPoly<<t1;
        resultModel<<tempPoly;

        //Внешнее дно
        tempPoly = Polygon3D(QColor(0,0,0,0));
        tempPoly<<c1;
        tempPoly<<b1;
        tempPoly<<prevB1;
        resultModel<<tempPoly;
        //Дно без разделения на сектора
        //outerBottom<<b1;

        //Внутреннее дно
        tempPoly = Polygon3D(QColor(0,0,0,0));
        tempPoly<<c2;
        tempPoly<<b2;
        tempPoly<<prevB2;
        resultModel<<tempPoly;
        //Дно без разделения на сектора
        //innerBottom<<b2;

        //Внешняя грань
        tempPoly = Polygon3D(QColor(0,0,0,0));
        tempPoly<<prevT1;
        tempPoly<<prevB1;
        tempPoly<<b1;
        tempPoly<<t1;
        resultModel<<tempPoly;

        //Внутренняя грань
        tempPoly = Polygon3D(QColor(0,0,0,0));
        tempPoly<<prevT2;
        tempPoly<<prevB2;
        tempPoly<<b2;
        tempPoly<<t2;
        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()/30;
    }
    else{
        scale_mult = this->height()/30;
    };
    QPoint p1 = QPoint(this->width()/4,this->height()/4);
    QPoint p2 = QPoint(3*this->width()/4,this->height()/4);
    QPoint p3 = QPoint(this->width()/4,3*this->height()/4);
    QPoint p4 = QPoint(3*this->width()/4,3*this->height()/4);
    //создания пера для границ
    QPen borderPen;
    borderPen.setWidth(2);
    borderPen.setColor(Qt::black);
    painter->setPen(borderPen);
    //фронтальная
    model.resetAfin();
    model.multWithAfin(ortoMatr);
    model.translate(QVector3D(p1.x(),p1.y(),1));
    model.scale(QVector3D(scale_mult,scale_mult,scale_mult));
    model(painter);
    //верх
    model.resetAfin();
    model.multWithAfin(ortoMatr);
    model.translate(QVector3D(p2.x(),p2.y(),1));
    model.rotate(90,QVector3D(1,0,0));
    model.scale(QVector3D(scale_mult,scale_mult,scale_mult));
    model(painter);
    //бок
    model.resetAfin();
    model.multWithAfin(ortoMatr);
    model.translate(QVector3D(p3.x(),p3.y(),1));
    model.rotate(90,QVector3D(1,0,0));
```

```
    model.scale(QVector3D(scale_mult,scale_mult,scale_mult));
    model(painter);
    //проектирование модели
    model.resetAfin();
    model.multWithAfin(projectionMatr);
    model.translate(QVector3D(p4.x(),p4.y(),1));
    model.scale(QVector3D(scale_mult,scale_mult,scale_mult));
    model.multWithAfin(mouseRotateMatr.transposed());
    model.rotate(90,QVector3D(1,0,0));
    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::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;
    public:
        DrawArea();
        void paintEvent(QPaintEvent *event);
        void wheelEvent(QWheelEvent *event);
        void mouseMoveEvent(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(QColor color){
    _color = color;
};

QColor Polygon3D::color(){
    return _color;
};

void Polygon3D::setColor(QColor color){
    _color=color;
};

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

```
#include <QtWidgets>
#pragma once

class Polygon3D{
    private:
        QList<QVector4D> _list;
        QColor _color;
    public:
        QColor color();
        void setColor(QColor color);
        Polygon3D(QColor color);
        void operator() (QPainter *painter){
            painter->save();
            painter->setBrush(color());
            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->_color = right._color;
            return *this;
```

```
        }
        friend Polygon3D operator *(Polygon3D poly,QMatrix4x4 matr){
            Polygon3D result(poly.color());
            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){
            for(int i=0,size=poly._list.size();i<size;i++){
                stream<<"("<<poly._list[i].x()<<","<<poly._list[i].y()<<")\n";
            }
            stream<<"\n";
            return stream;
        }
};

QPoint vectToPoint(QVector4D vect);
```