Министерство образования и науки РФ

Федеральное государственное бюджетное образовательное учреждение

высшего профессионального образования

Тульский государственный университет

КАФЕДРА АВТОМАТИКИ И ТЕЛЕМЕХАНИКИ

**АФИННЫЕ ПРЕОБРАЗОВАНИЯ В ПРОСТРАНСТВЕ И ЦЕНТРАЛЬНОЕ ПРОЕЦИРОВАНИЕ**

Лабораторная работа № 3

по курсу «Компьютерная графика»

Вариант № 3

Выполнил: студент группы 220601 \_\_\_\_\_\_\_\_\_ Белым А.А.

(подпись)

Проверил: \_\_\_\_\_\_\_\_\_ Фомичев А.М.

(подпись)

Тула 2012

# Цель работы

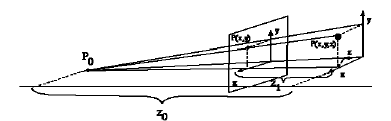
Освоить математические основы аффинных и проективных преобразований в пространстве и уметь их использовать в практике программирования.

# Задание

Разработать программу, обеспечивающую вывод графическогоизображенияобъекта на плоскость до и после заданных преобразований. Построить центральную проекцию объекта на плоскость экрана после преобразования.

# Теоретическая справка

Т.к. экран является плоскостью, то на нем можно изобразить только проекции трехмерных объектов. Наиболее простым и быстродействующим в машинной графике является центральное перспективное преобразование (центральная проекция).



Задача состоит в том, чтобы определить проекцию точки p(х,у,z) в трехмерном пространстве на некоторую плоскость, называемую картинной плоскостью. Такая проекция Р(X,Y) называется образом точки р(х,у,z).

При центральной проекции точка зрения (или центральная проекция) находятся на одной из главных осей трехмерной ортогональной системы координат, например на Z – оси. Следовательно ось Z является оптической осью. Картинная плоскость перпендикулярна оптической оси и параллельна плоскости XY.

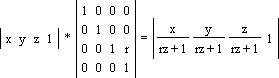
**Матрица общего перспективного преобразования**

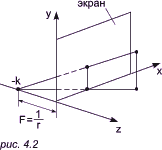


В этой матрице элементы a, d, е отвечают за масштабирование, m, n, L — за смещение, p, q, r — за проецирование, s — за комплексное масштабирование, х — за вращение.

**Одноточечное проецирование на плоскость z = 0**

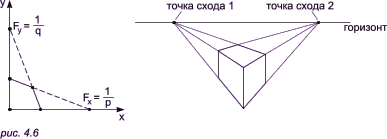
Для того, чтобы точки, лежащие на линии, параллельной оси z, не терялись друг за другом, используется одноточечное проецирование на линию; исчезла z-координата, но, поскольку дальние предметы стали более мелкими, чем такие же близкие, у зрителя появляется ощущение глубины.





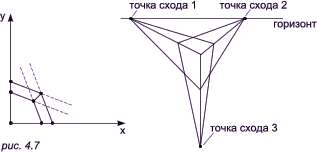
**Двухточечное проецирование**

Если проекция двухточечная (например, по p <> 0 и q <> 0), то имеются две точки схода на соответствующие оси. Обратите внимание: так как по z в данном случае реализуется параллельное проецирование, то удвоения контура куба на экране (x, y) нет. Меняя p и q, мы регулируем точку схода:



**Трехточечное проецирование по p, q, r**

В данном случае p <> 0, q <> 0, r <> 0, и проекция будет иметь следующий вид:



# Текст программы

Ниже представлен текст программы, написанной на языке C++, в среде Qt Creator 2.5.2 + MinGW-GCC 4.6 с использованием библиотеки Qt.

compgraph.h:

#ifndef COMPGRAPHVIEW\_H

#define COMPGRAPHVIEW\_H

#include <QGraphicsView>

//Размерность точки в однородных координатах

const double defN**=-**0.004**;**

const int N**=**4**;**

enum Axes**{**

OX**,**OY**,**OZ

**};**

//Мой собственный вектор(со всеми прилагающимися) для

//точки в однородных координатах

class Vector **{**

double **\***data**;**

public**:**

int n**;**

//Конструктор по декартовым координатам точки

Vector**(**int x**,**int y**);**

Vector**(**int x**,**int y**,**int z**);**

//Конструктор заданной размерности

Vector**(**int n**);**

**~**Vector**();**

//Ниасилил перегрузку typecast(если она есть)

QPoint getPoint**();**

Vector**&** **operator** **=(**const Vector**&** other**);**

double**&** **operator** **[](**int n**)** const**;**

**};**

class Matrix**{**

Vector **\*\***data**;**

public**:**

int n**,**m**;**

//Конструктор заданной размерности

Matrix**(**int n**,**int m**);**

**~**Matrix**();**

Vector**&** **operator** **[](**int m**)** const**;**

Matrix**&** **operator** **=(**const Matrix**&** other**);**

//Перемножение матриц

Matrix **operator** **\*(**const Matrix**&** other**);**

//Умножение на вектор. Учитывается однородный масштаб

//Есть функция простого перемножения матрицы на вектор, ниже

Vector **operator** **\*(**const Vector**&** other**);**

**};**

**typedef** QList**<**Vector**\*>** figure\_t**;**

class CompGraphView **:** public QGraphicsView

**{**

Q\_OBJECT**;**

public**:**

//Матрицы трансформаций - поворота и смещения/масштаба

Matrix **\***ScaleMatrix**,\***RotOXMatrix**,\***RotOYMatrix**,\***RotOZMatrix**,\***MoveMatrix**;**

//Знак Каспера

QList**<**figure\_t**>** figures**;**

explicit CompGraphView**(**QWidget **\***parent **=** 0**);**

private**:**

//Отрисовка

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

**};**

//Перемножение матриц

Matrix multMnM**(**const Matrix **&**a**,** const Matrix **&**b**);**

//Умножение матрицы на вектор

Vector multMnV**(**const Matrix **&**a**,**const Vector **&**v**);**

//Умножение матрицы на вектор с учетом однородного масштаба

Vector multMnVNorm**(**const Matrix **&**a**,**const Vector **&**v**);**

//Получение матрицы вращения

Matrix RotM**(**const double alpha**=**0**,**Axes axis**=**OY**);**

/\*Получение матрицы смещения/масштаба

p,q - координаты смещения

scl\_x,scl\_y - масштаб по осям X и Y

scl\_gen - однородный масштаб\*/

Matrix MovM**(**const double p**=**0**,** const double q**=**0**,** const double r**=**0**,**

const double scl\_x**=**1**,** const double scl\_y**=**1**,** const double scl\_z**=**1**,**

const double scl\_gen**=**1**,**

const double l**=**0**,** const double m**=**0**,** const double n**=**0**);**

#endif // COMPGRAPHVIEW\_H

compgraph.cpp:

#include "compgraphview.h"

#include <cmath>

#include <iostream>

#include <QDebug>

#include <QTimer>

**using** **namespace** std**;**

const int main\_size**=**300**,**arrow\_size**=**130**,**tail\_wdth**=**50**,**wdth**=**85**,**zwdth**=**20**;**

const double deg**=**cos**(**45**\***M\_PI**/**180**);**

double x**=-**1**;**

CompGraphView**::**CompGraphView**(**QWidget **\***parent**)** **:**

QGraphicsView**(**parent**)**

**{**

//Описание точек фигуры

figure\_t figure**;**

figure**<<new** Vector**(**0**,**0**,**zwdth**);**

figure**<<new** Vector**(**0**,**main\_size**,**zwdth**);**

figure**<<new** Vector**(**0**,**main\_size**,-**zwdth**);**

figure**<<new** Vector**(**0**,**0**,-**zwdth**);**

figures**<<**figure**;**

figure**.**clear**();**

figure**<<new** Vector**(**0**,**main\_size**,**zwdth**);**

figure**<<new** Vector**(**main\_size**/**2**-**deg**\***tail\_wdth**,**main\_size**/**2**+**deg**\***tail\_wdth**,**zwdth**);**

figure**<<new** Vector**(**main\_size**/**2**-**deg**\***tail\_wdth**,**main\_size**/**2**+**deg**\***tail\_wdth**,-**zwdth**);**

figure**<<new** Vector**(**0**,**main\_size**,-**zwdth**);**

figures**<<**figure**;**

figure**.**clear**();**

figure**<<new** Vector**(**main\_size**/**2**-**deg**\***tail\_wdth**,**main\_size**/**2**+**deg**\***tail\_wdth**,**zwdth**);**

figure**<<new** Vector**(**1.**/**2**\*(-**arrow\_size **+** 2**\***main\_size **-** 2**\***deg**\***tail\_wdth**),**1.**/**2**\*(-**arrow\_size **+** 2**\***main\_size **+** 2**\***deg**\***tail\_wdth**),**zwdth**);**

figure**<<new** Vector**(**1.**/**2**\*(-**arrow\_size **+** 2**\***main\_size **-** 2**\***deg**\***tail\_wdth**),**1.**/**2**\*(-**arrow\_size **+** 2**\***main\_size **+** 2**\***deg**\***tail\_wdth**),-**zwdth**);**

figure**<<new** Vector**(**main\_size**/**2**-**deg**\***tail\_wdth**,**main\_size**/**2**+**deg**\***tail\_wdth**,-**zwdth**);**

figures**<<**figure**;**

figure**.**clear**();**

figure**<<new** Vector**(**1.**/**2**\*(-**arrow\_size **+** 2**\***main\_size **-** 2**\***deg**\***tail\_wdth**),**1.**/**2**\*(-**arrow\_size **+** 2**\***main\_size **+** 2**\***deg**\***tail\_wdth**),**zwdth**);**

figure**<<new** Vector**(**main\_size**-**arrow\_size**,+**main\_size**,**zwdth**);**

figure**<<new** Vector**(**main\_size**-**arrow\_size**,+**main\_size**,-**zwdth**);**

figure**<<new** Vector**(**1.**/**2**\*(-**arrow\_size **+** 2**\***main\_size **-** 2**\***deg**\***tail\_wdth**),**1.**/**2**\*(-**arrow\_size **+** 2**\***main\_size **+** 2**\***deg**\***tail\_wdth**),-**zwdth**);**

figures**<<**figure**;**

figure**.**clear**();**

figure**<<new** Vector**(**main\_size**-**arrow\_size**,+**main\_size**,**zwdth**);**

figure**<<new** Vector**(**main\_size**,**main\_size**,**zwdth**);**

figure**<<new** Vector**(**main\_size**,**main\_size**,-**zwdth**);**

figure**<<new** Vector**(**main\_size**-**arrow\_size**,+**main\_size**,-**zwdth**);**

figures**<<**figure**;**

figure**.**clear**();**

figure**<<new** Vector**(**main\_size**,**main\_size**,**zwdth**);**

figure**<<new** Vector**(**main\_size**,**main\_size**-**arrow\_size**,**zwdth**);**

figure**<<new** Vector**(**main\_size**,**main\_size**-**arrow\_size**,-**zwdth**);**

figure**<<new** Vector**(**main\_size**,**main\_size**,-**zwdth**);**

figures**<<**figure**;**

figure**.**clear**();**

figure**<<new** Vector**(**main\_size**,**main\_size**-**arrow\_size**,**zwdth**);**

figure**<<new** Vector**(**1.**/**2**\*(-**arrow\_size **+** 2**\***main\_size **+** 2**\***deg**\***tail\_wdth**),**1.**/**2**\*(-**arrow\_size **+** 2**\***main\_size **-** 2**\***deg**\***tail\_wdth**),**zwdth**);**

figure**<<new** Vector**(**1.**/**2**\*(-**arrow\_size **+** 2**\***main\_size **+** 2**\***deg**\***tail\_wdth**),**1.**/**2**\*(-**arrow\_size **+** 2**\***main\_size **-** 2**\***deg**\***tail\_wdth**),-**zwdth**);**

figure**<<new** Vector**(**main\_size**,**main\_size**-**arrow\_size**,-**zwdth**);**

figures**<<**figure**;**

figure**.**clear**();**

figure**<<new** Vector**(**1.**/**2**\*(-**arrow\_size **+** 2**\***main\_size **+** 2**\***deg**\***tail\_wdth**),**1.**/**2**\*(-**arrow\_size **+** 2**\***main\_size **-** 2**\***deg**\***tail\_wdth**),**zwdth**);**

figure**<<new** Vector**(**main\_size**/**2**+**deg**\***tail\_wdth**,**main\_size**/**2**-**deg**\***tail\_wdth**,**zwdth**);**

figure**<<new** Vector**(**main\_size**/**2**+**deg**\***tail\_wdth**,**main\_size**/**2**-**deg**\***tail\_wdth**,-**zwdth**);**

figure**<<new** Vector**(**1.**/**2**\*(-**arrow\_size **+** 2**\***main\_size **+** 2**\***deg**\***tail\_wdth**),**1.**/**2**\*(-**arrow\_size **+** 2**\***main\_size **-** 2**\***deg**\***tail\_wdth**),-**zwdth**);**

figures**<<**figure**;**

figure**.**clear**();**

figure**<<new** Vector**(**main\_size**/**2**+**deg**\***tail\_wdth**,**main\_size**/**2**-**deg**\***tail\_wdth**,**zwdth**);**

figure**<<new** Vector**(**main\_size**,**0**,**zwdth**);**

figure**<<new** Vector**(**main\_size**,**0**,-**zwdth**);**

figure**<<new** Vector**(**main\_size**/**2**+**deg**\***tail\_wdth**,**main\_size**/**2**-**deg**\***tail\_wdth**,-**zwdth**);**

figures**<<**figure**;**

figure**.**clear**();**

figure**<<new** Vector**(**main\_size**,**0**,**zwdth**);**

figure**<<new** Vector**(**main\_size**-**2**\***deg**\***wdth**,**0**,**zwdth**);**

figure**<<new** Vector**(**main\_size**-**2**\***deg**\***wdth**,**0**,-**zwdth**);**

figure**<<new** Vector**(**main\_size**,**0**,-**zwdth**);**

figures**<<**figure**;**

figure**.**clear**();**

figure**<<new** Vector**(**main\_size**-**2**\***deg**\***wdth**,**0**,**zwdth**);**

figure**<<new** Vector**(**wdth**,-**wdth**-**2**\***deg**\***wdth **+** main\_size**,**zwdth**);**

figure**<<new** Vector**(**wdth**,-**wdth**-**2**\***deg**\***wdth **+** main\_size**,-**zwdth**);**

figure**<<new** Vector**(**main\_size**-**2**\***deg**\***wdth**,**0**,-**zwdth**);**

figures**<<**figure**;**

figure**.**clear**();**

figure**<<new** Vector**(**wdth**,-**wdth**-**2**\***deg**\***wdth **+** main\_size**,**zwdth**);**

figure**<<new** Vector**(**wdth**,**0**,**zwdth**);**

figure**<<new** Vector**(**wdth**,**0**,-**zwdth**);**

figure**<<new** Vector**(**wdth**,-**wdth**-**2**\***deg**\***wdth **+** main\_size**,-**zwdth**);**

figures**<<**figure**;**

figure**.**clear**();**

figure**<<new** Vector**(**wdth**,**0**,**zwdth**);**

figure**<<new** Vector**(**0**,**0**,**zwdth**);**

figure**<<new** Vector**(**0**,**0**,-**zwdth**);**

figure**<<new** Vector**(**wdth**,**0**,-**zwdth**);**

figures**<<**figure**;**

figure**.**clear**();**

//Создание пустых матриц трансформаций по умолчанию

RotOXMatrix**=new** Matrix**(**4**,**4**);**

**\***RotOXMatrix**=**RotM**();**

RotOYMatrix**=new** Matrix**(**4**,**4**);**

**\***RotOYMatrix**=**RotM**();**

RotOZMatrix**=new** Matrix**(**4**,**4**);**

**\***RotOZMatrix**=**RotM**();**

ScaleMatrix**=new** Matrix**(**4**,**4**);**

**\***ScaleMatrix**=**MovM**();**

MoveMatrix**=new** Matrix**(**4**,**4**);**

**\***MoveMatrix**=**MovM**(**0**,**0**,**0**,**1**,**1**,**1**,**1**,**0**,**0**,**defN**);**

**}**

void CompGraphView**::**paintEvent**(**QPaintEvent **\***event**){**

//Рисовалки

QPainter painter**(**viewport**());** QPainterPath path**;**

QPen pen**(**QColor**(**"Red"**));**QBrush brush**(**QColor**(**"Red"**));**

//Текущая точка и конечная матрица преобразований

Vector t**(**4**);**Matrix M**(**4**,**4**);**

//Расчет координат центра экрана

double centx**=**viewport**()->**width**()/**2**,**centy**=**viewport**()->**height**()/**2**;**

//Получаем матрицу сложного преобразования

//Перенос центра фигуры в начало координат

M**=**MovM**(-**main\_size**/**2.**,-**main\_size**/**2.**);**

M**=\***ScaleMatrix**\***M**;**

//Поворот фигуры

M**=\***RotOXMatrix**\***M**;**

M**=\***RotOYMatrix**\***M**;**

M**=\***RotOZMatrix**\***M**;**

//Перенос и масштабирование

M**=\***MoveMatrix**\***M**;**

//Перенос результата в центр экрана

M**=**MovM**(**centx**,**centy**)\***M**;**

//Матрица готова!

**for** **(**int j**=**0**;**j**<**figures**.**size**();++**j**){**

figure\_t figure**=**figures**[**j**];**

//Получаем первую точку и перемещаемся в неё

//Со списком указателей получается кривовато

t**=**M**\*(\***figure**[**0**]);**

path**.**moveTo**(**t**.**getPoint**());**

**for** **(**int i**=**1**;**i**<**figure**.**size**();++**i**){**

//Получаем точку и строим до неё линию до предыдущей

t**=**M**\*(\***figure**[**i**]);**

path**.**lineTo**(**t**.**getPoint**());**

**}**

**}**

//Прорисовываем все линии на экране

painter**.**strokePath**(**path**,**pen**);**

**}**

//Перемножение матриц

Matrix multMnM**(**Matrix const **&**a**,**Matrix const **&**b**){**

Matrix res**(**a**.**n**,**b**.**m**);**

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

**for(**int j**=**0**;**j**<**b**.**m**;**j**++){**

res**[**j**][**i**]=**0**;**

**for(**int k**=**0**;**k**<**a**.**m**;**k**++){**

res**[**j**][**i**]+=**a**[**k**][**i**]\***b**[**j**][**k**];**

**}**

**}**

**}**

**return** res**;**

**};**

//Умножение матрицы на вектор

Vector multMnV**(**const Matrix **&**a**,** const Vector **&**v**){**

Vector res**(**v**.**n**);**

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

res**[**i**]=**0**;**

**for(**int j**=**0**;**j**<**a**.**m**;**j**++){**

res**[**i**]+=**a**[**j**][**i**]\***v**[**j**];**

**}**

**}**

**return** res**;**

**}**

//Умножение матрицы на вектор с учетом однородного масштаба

Vector multMnVNorm**(**const Matrix **&**a**,** const Vector **&**v**){**

Vector res**=**multMnV**(**a**,**v**);**

res**[**0**]/=**res**[**3**];**

res**[**1**]/=**res**[**3**];**

res**[**2**]/=**res**[**3**];**

**return** res**;**

**}**

//Получение матрицы вращения

Matrix RotM2D**(**const double alpha**){**

Matrix res**(**3**,**3**);**

res**[**0**][**0**]=**cos**(**alpha**);**

res**[**1**][**0**]=**sin**(**alpha**);**

res**[**2**][**0**]=**0**;**

res**[**0**][**1**]=-**sin**(**alpha**);**

res**[**1**][**1**]=**cos**(**alpha**);**

res**[**2**][**1**]=**0**;**

res**[**0**][**2**]=**0**;**

res**[**1**][**2**]=**0**;**

res**[**2**][**2**]=**1**;**

**return** res**;**

**};**

//Получение матрицы вращения

Matrix RotM**(**const double alpha**,**Axes axis**){**

Matrix res**(**4**,**4**);**

**for(**int i**=**0**;**i**<**4**;**i**++)**

**for(**int j**=**0**;**j**<**4**;**j**++)**

res**[**j**][**i**]=**0**;**

**switch(**axis**){**

**case** OZ**:**

res**[**0**][**0**]=**cos**(**alpha**);**

res**[**1**][**0**]=-**sin**(**alpha**);**

res**[**0**][**1**]=**sin**(**alpha**);**

res**[**1**][**1**]=**cos**(**alpha**);**

res**[**2**][**2**]=**1**;**

**break;**

**case** OY**:**

res**[**0**][**0**]=**cos**(-**alpha**);**

res**[**2**][**0**]=-**sin**(-**alpha**);**

res**[**0**][**2**]=**sin**(-**alpha**);**

res**[**2**][**2**]=**cos**(-**alpha**);**

res**[**1**][**1**]=**1**;**

**break;**

**case** OX**:**

res**[**1**][**1**]=**cos**(**alpha**);**

res**[**2**][**1**]=-**sin**(**alpha**);**

res**[**1**][**2**]=**sin**(**alpha**);**

res**[**2**][**2**]=**cos**(**alpha**);**

res**[**0**][**0**]=**1**;**

**}**

res**[**3**][**3**]=**1**;**

**return** res**;**

**};**

/\*Получение матрицы смещения/масштаба

m,n - координаты смещения

scl\_x,scl\_y - масштаб по осям X и Y

scl\_gen - однородный масштаб\*/

Matrix MovM2D**(**const double p**,** const double q**,**

const double scl\_x**,** const double scl\_y**,**const double scl\_gen**){**

Matrix res**(**3**,**3**);**

res**[**0**][**0**]=**scl\_x**;**

res**[**1**][**1**]=**scl\_y**;**

res**[**2**][**2**]=**scl\_gen**;**

res**[**2**][**0**]=**p**;**

res**[**2**][**1**]=**q**;**

res**[**1**][**0**]=**0**;**

res**[**0**][**1**]=**0**;**

res**[**0**][**2**]=**0**;**

res**[**1**][**2**]=**0**;**

**return** res**;**

**};**

Matrix MovM**(**const double p**,** const double q**,** const double r**,**

const double scl\_x**,** const double scl\_y**,** const double scl\_z**,**

const double scl\_gen**,** const double l**,** const double m**,** const double n**){**

Matrix res**(**4**,**4**);**

**for(**int i**=**0**;**i**<**4**;**i**++)**

**for(**int j**=**0**;**j**<**4**;**j**++)**

res**[**j**][**i**]=**0**;**

res**[**0**][**0**]=**scl\_x**;**

res**[**1**][**1**]=**scl\_y**;**

res**[**2**][**2**]=**scl\_z**;**

res**[**3**][**3**]=**scl\_gen**;**

res**[**3**][**0**]=**p**;**

res**[**3**][**1**]=**q**;**

res**[**3**][**2**]=**r**;**

res**[**0**][**3**]=**l**;**

res**[**1**][**3**]=**m**;**

res**[**2**][**3**]=**n**;**

**return** res**;**

**};**

Vector**::**Vector**(**int x**,** int y**){**

**this->**n**=**3**;**

data**=new** double**[this->**n**];**

**(\*this)[**0**]=**x**;(\*this)[**1**]=**y**;(\*this)[**2**]=**1**;**

**}**

Vector**::**Vector**(**int x**,** int y**,** int z**){**

**this->**n**=**4**;**

data**=new** double**[this->**n**];**

**(\*this)[**0**]=**x**;(\*this)[**1**]=**y**;(\*this)[**2**]=**z**;(\*this)[**3**]=**1**;**

**}**

Vector**::**Vector**(**int n**){**

**this->**n**=**n**;**

data**=new** double**[this->**n**];**

**}**

Vector**::~**Vector**(){**

**delete** data**;**

**}**

double**&** Vector**::operator** **[](**const int n**)** const**{**

**if(**n**>=**0**&&**n**<=this->**n**){**

**return** data**[**n**];**

**}**

**}**

QPoint Vector**::**getPoint**(){**

**return** QPoint**((\*this)[**0**],(\*this)[**1**]);**

**}**

Vector**&** Vector**::operator** **=(**const Vector**&** other**){**

**for(**int i**=**0**;**i**<this->**n**;**i**++){**

**this->**data**[**i**]=**other**.**data**[**i**];**

**}**

**return** **\*this;**

**}**

Matrix**::**Matrix**(**int n**,**int m**){**

**this->**n**=**n**;**

**this->**m**=**m**;**

data **=** **new** Vector**\*[**m**];**

**for** **(**int i**=**0**;**i**<**m**;**i**++){**

data**[**i**]=new** Vector**(**n**);**

**}**

**}**

Matrix**::~**Matrix**(){**

**for** **(**int i**=**0**;**i**<**m**;**i**++){**

**delete** data**[**i**];**

**}**

**delete** data**;**

**}**

Vector**&** Matrix**::operator** **[](**const int m**)** const**{**

**if(**m**>=**0**&&**m**<=this->**m**){**

**return** **\***data**[**m**];**

**}**

**}**

Matrix**&** Matrix**::operator** **=(**const Matrix**&** other**){**

**for(**int i**=**0**;**i**<this->**n**;**i**++){**

**for(**int j**=**0**;**j**<this->**m**;++**j**){**

**\*this->**data**[**j**]=\***other**.**data**[**j**];**

**}**

**}**

**return** **\*this;**

**}**

Matrix Matrix**::operator** **\*(**const Matrix**&** other**){**

**return** multMnM**(\*this,**other**);**

**}**

Vector Matrix**::operator** **\*(**const Vector**&** other**){**

**return** multMnVNorm**(\*this,**other**);**

**}**

# Тестовый пример

На рисунке 1 показан пример работы программы рисования и трансформаций символа с помощью аффинных преобразований.

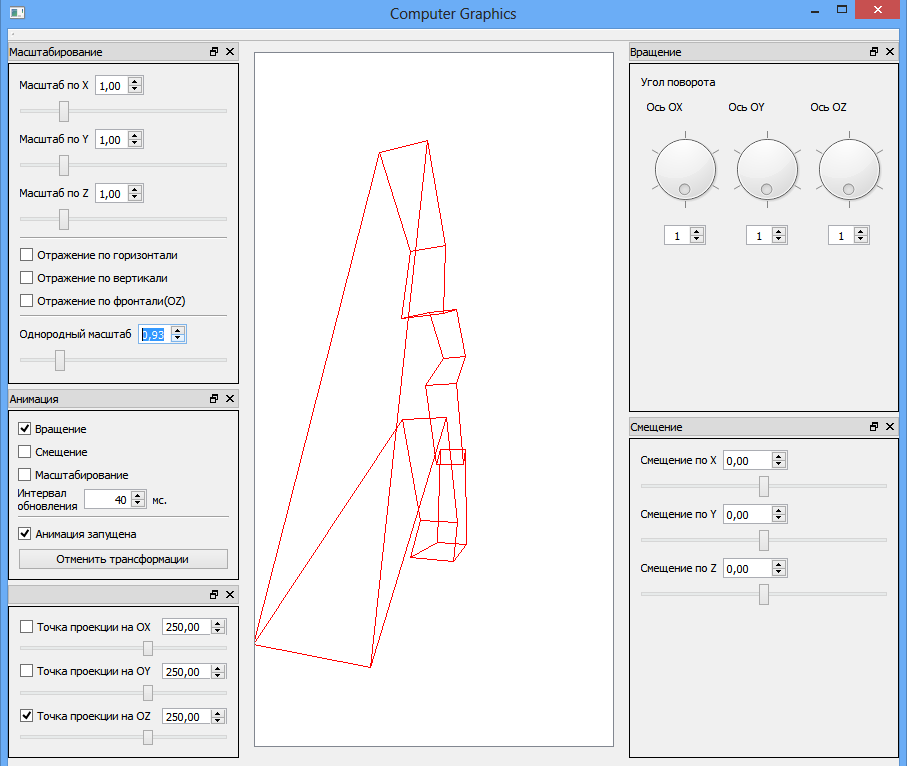


Рисунок 1— Пример работы разработанной программы.

# Вывод

В ходе выполнения данной лабораторной работы я освоил математические и алгоритмические основы аффинных преобразований в трехмерном пространстве и центральное (перспективное) проецирование пространства на плоскость, а также написал программу, позволяющую совершать такие преобразования над изображением символа «Антивируса Касперского».

Перспективное проецирование использует особенности восприятия человеческого глаза, и позволяет воссоздать эффект глубины, что делает таку проекцию востребованной в различных областях: компьютерных играх, анимации и т.д.