Министерство образования и науки РФ
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования

Тульский государственный университет

КАФЕДРА АВТОМАТИКИ И ТЕЛЕМЕХАНИКИ

# УДАЛЕНИЕ НЕВИДИМЫХ ЛИНИЙ И ПОВЕРХНОСТЕЙ. АЛГОРИТМ, ИСПОЛЬЗУЮЩИЙ Z-БУФЕР

Лабораторная работа № 4
по курсу «Компьютерная графика»

Вариант № 3

Выполнил: студент группы 220601 _____ Белым А.А.
(подпись)
Проверил: _____ Фомичев А.М.
(подпись)

Тула 2012

## Цель работы

Освоить алгоритмические основы удаления невидимых линий и поверхностей и уметь их использовать в практике программирования.

## Задание

Доработать программу из предыдущей лабораторной работы так, чтобы на экран не отображались невидимые линии каркаса обьекта. По возможности организовать заливку граней обьекта с учетом интенсивности цвета в соответствии с координатой z. При выполнении задания использовать алгоритм z-буфера.

## Теоретическая справка

Задача удаления невидимых линий и поверхностей является одной из наиболее сложных в машинной графике. Алгоритмы удаления невидимых линий и поверхностей служат для определения линий ребер, поверхностей или объемов, которые видимы или невидимы для наблюдателя, находящегося в заданной точке пространства.

Алгоритм, использующий z-буфер – это один из простейших алгоритмов удаления невидимых поверхностей.

Впервые он был предложен Кэтмулом. Работает этот алгоритм в пространстве изображения. Идея z-буфера является простым обобщением идеи о буфере кадра. Буфер кадра используется для запоминания атрибутов (интенсивности) каждого пиксела в пространстве изображения, z-буфер - это отдельный буфер глубины, используемый для запоминания координаты z или глубины каждого видимого пиксела в пространстве изображения. В процессе работы глубина (или значение z) каждого нового пиксела, который нужно занести в буфер кадра, сравнивается с глубиной того пиксела, который уже занесен в z-буфер.

Если это сравнение показывает, что новый пиксел расположен впереди пиксела, находящегося в буфере кадра, то новый пиксел заносится в этот буфер и, кроме того, производится корректировка z-буфера новым значением z. Если же сравнение дает противоположный результат, то никаких действий не

производится. По сути, алгоритм является поиском по x и y наибольшего значения функции z(x, y).

Главное преимущество алгоритма – его простота. Кроме того, этот алгоритм решает задачу об удалении невидимых поверхностей и делает тривиальной визуализацию пересечений сложных поверхностей. Сцены могут быть любой сложности. Поскольку габариты пространства изображения фиксированы, оценка вычислительной трудоемкости алгоритма не более чем линейна. Поскольку элементы сцены или картинки можно заносить в буфер кадра или в z-буфер в произвольном порядке, их не нужно предварительно сортировать по приоритету глубины.

Основной недостаток алгоритма – большой объем требуемой памяти. Если сцена подвергается видовому преобразованию и отсекается до фиксированного диапазона координат z значений, то можно использовать z-буфер с фиксированной точностью. Информацию о глубине нужно обрабатывать с большей точностью, чем координатную информацию на плоскости (x, y).

Уменьшение требуемой памяти достигается разбиением пространства изображения на 4, 16 или больше квадратов или полос. В предельном варианте можно использовать z-буфер размером в одну строку развертки. Для последнего случая имеется интересный алгоритм построчного сканирования.

Поскольку каждый элемент сцены обрабатывается много раз, то сегментирование z-буфера, вообще говоря, приводит к увеличению времени, необходимого для обработки сцены. Однако сортировка на плоскости, позволяющая не обрабатывать все многоугольники в каждом из квадратов или полос, может значительно сократить этот рост.

Другой недостаток алгоритма z-буфера состоит в трудоемкости и высокой стоимости устранения лестничного эффекта, а также реализации эффектов прозрачности и просвечивания. Поскольку алгоритм заносит пикселы в буфер кадра в произвольном порядке, то нелегко получить информацию, необходимую для методов устранения лестничного эффекта, основывающихся на предварительной фильтрации. При реализации эффектов прозрачности и

просвечивания, пикселы могут заноситься в буфер кадра в некорректном порядке, что ведет к локальным ошибкам.

Более формальное описание алгоритма z-буфера таково:

1. Заполнить буфер кадра фоновым значением интенсивности или цвета.
2. Заполнить z -буфер минимальным значением z.
3. Преобразовать каждый многоугольник в растровую форму в произвольном порядке.
4. Для каждого Пиксел(x, y) в многоугольнике вычислить его глубину z(x, y).
5. Сравнить глубину z(x, y) со значением Zбуфер(x, y), хранящимся в z-буфере в этой же позиции.
6. Если z(x, y) > Zбуфер(x, y), то записать атрибут этого многоугольника в буфер кадра и заменить Zбуфер(x, y) на z(x, y).
7. В противном случае никаких действий не производить.

## Текст программы

Ниже представлен текст программы, написанной на языке C++, в среде Qt Creator 2.6 + GCC 4.7 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;
const int MaxColor=250;
enum Axes{
    OX,OY,OZ
};

//Мой собственный вектор(со всеми прилагающимися) для
//точки в однородных координатах
template <typename T> class VectorT {
    T *data;
public:
    int n;
    //Конструктор по декартовым координатам точки
    VectorT(int x,int y);
    VectorT(int x,int y,int z);
    //Конструктор заданной размерности
    VectorT(int n);
    VectorT(const VectorT<T>& other);
    VectorT();
    ~VectorT();
    //Ниасилил перегрузку typecast(если она есть)
```

```
    QPoint getPoint();
    VectorT& operator =(const VectorT& other);
    T& operator [](int n) const;
};

typedef VectorT<double> Vector;

typedef struct{
    int x,y;double z; bool next_y;
} SimplePoint;

template <typename T>
class MatrixT{
    VectorT<T> **data;
public:
    int n,m;
    //Конструктор заданной размерности
    MatrixT(int n,int m);
    ~MatrixT();
    VectorT<T>& operator [](int m) const;
    MatrixT& operator =(const MatrixT& other);
    //Перемножение матриц
    MatrixT operator *(const MatrixT& other);
    //Умножение на вектор. Учитывается однородный масштаб
    //Есть функция простого перемножения матрицы на вектор, ниже
    VectorT<T> operator *(const VectorT<T>& other);
};

typedef MatrixT<double> Matrix;

typedef QList<int> figure_t;
typedef QList<SimplePoint> points;
class CompGraphView : public QGraphicsView
{
    Q_OBJECT;
public:
    //Матрицы трансформаций - поворота и смещения/масштаба
    Matrix *ScaleMatrix,*RotOXMatrix,*RotOYMatrix,*RotOZMatrix,*MoveMatrix;
    //Знак Каспера
    QList<Vector> figure_points;
    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);
```

```
template <typename T>
VectorT<T>::VectorT(int x, int y){
    this->n=3;
    data=new T[this->n];
    (*this)[0]=x;(*this)[1]=y;(*this)[2]=1;
}

template <typename T>
VectorT<T>::VectorT(int x, int y, int z){
    this->n=4;
    data=new T[this->n];
    (*this)[0]=x;(*this)[1]=y;(*this)[2]=z;(*this)[3]=1;
}

template <typename T>
VectorT<T>::VectorT(int n){
    this->n=n;
    data=new T[this->n];
}

template <typename T>
VectorT<T>::VectorT(const VectorT<T>& other){
    this->n=other.n;
    data=new T[this->n];
    for(int i=0;i<this->n;i++){
        this->data[i]=other.data[i];
    }
}

template <typename T>
VectorT<T>::VectorT(){
    this->n=0;
    this->data=NULL;
}

template <typename T>
VectorT<T>::~VectorT(){
    delete data;
}

template <typename T>
T& VectorT<T>::operator [](const int n) const{
    if(n>=0&&n<=this->n){
        return data[n];
    }
}

template <typename T>
QPoint VectorT<T>::getPoint(){
    return QPoint((*this)[0],(*this)[1]);
}

template <typename T>
VectorT<T>& VectorT<T>::operator =(const VectorT<T>& other){
    for(int i=0;i<this->n;i++){
        this->data[i]=other.data[i];
    }
    return *this;
}

template <typename T>
MatrixT<T>::MatrixT(int n,int m){
    this->n=n;
    this->m=m;
    data = new VectorT<T>*[m];
    for (int i=0;i<m;i++){
```

```
        data[i]=new VectorT<T>(n);
    }
}

template <typename T>
MatrixT<T>::~MatrixT(){
    for (int i=0;i<m;i++){
        delete data[i];
    }
    delete data;
}

template <typename T>
VectorT<T>& MatrixT<T>::operator [](const int m) const{
    if(m>=0&&m<=this->m){
        return *data[m];
    }
}

template <typename T>
MatrixT<T>& MatrixT<T>::operator =(const MatrixT<T>& 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;
}

template <typename T>
MatrixT<T> MatrixT<T>::operator *(const MatrixT<T>& other){
    return multMnM(*this,other);
}

template <typename T>
VectorT<T> MatrixT<T>::operator *(const VectorT<T> &other){
    return multMnVNorm(*this,other);
}

#endif // COMPGRAPHVIEW_H
```

compgraph.cpp:

```
#include "compgraphview.h"
#include <cmath>
#include <iostream>
#include <QDebug>
#include <QTimer>
#include <climits>
#include <QtAlgorithms>
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;

QPen zpalette[MaxColor+1];

CompGraphView::CompGraphView(QWidget *parent) :
    QGraphicsView(parent)
{
    //Описание точек фигуры
    figure_points<<Vector(0,0,zwdth); //0
    figure_points<<Vector(0,main_size,zwdth); //1
```

```
    figure_points<<Vector(main_size/2-
deg*tail_wdth,main_size/2+deg*tail_wdth,zwdth); //2
    figure_points<<Vector(1./2*(-arrow_size + 2*main_size -
2*deg*tail_wdth),1./2*(-arrow_size + 2*main_size + 2*deg*tail_wdth),zwdth); //3
    figure_points<<Vector(main_size-arrow_size,+main_size,zwdth); //4
    figure_points<<Vector(main_size,main_size,zwdth); //5
    figure_points<<Vector(main_size,main_size-arrow_size,zwdth); //6
    figure_points<<Vector(1./2*(-arrow_size + 2*main_size +
2*deg*tail_wdth),1./2*(-arrow_size + 2*main_size - 2*deg*tail_wdth),zwdth); //7
    figure_points<<Vector(main_size/2+deg*tail_wdth,main_size/2-
deg*tail_wdth,zwdth); //8
    figure_points<<Vector(main_size,0,zwdth); //9
    figure_points<<Vector(main_size-2*deg*wdth,0,zwdth); //10
    figure_points<<Vector(wdth,-wdth-2*deg*wdth + main_size,zwdth); //11
    figure_points<<Vector(wdth,0,zwdth); //12
    figure_points<<Vector(0,0,-zwdth); //13
    figure_points<<Vector(0,main_size,-zwdth); //14
    figure_points<<Vector(main_size/2-deg*tail_wdth,main_size/2+deg*tail_wdth,-
zwdth); //15
    figure_points<<Vector(1./2*(-arrow_size + 2*main_size -
2*deg*tail_wdth),1./2*(-arrow_size + 2*main_size + 2*deg*tail_wdth),-zwdth); //16
    figure_points<<Vector(main_size-arrow_size,+main_size,-zwdth); //17
    figure_points<<Vector(main_size,main_size,-zwdth); //18
    figure_points<<Vector(main_size,main_size-arrow_size,-zwdth); //19
    figure_points<<Vector(1./2*(-arrow_size + 2*main_size +
2*deg*tail_wdth),1./2*(-arrow_size + 2*main_size - 2*deg*tail_wdth),-zwdth); //20
    figure_points<<Vector(main_size/2+deg*tail_wdth,main_size/2-deg*tail_wdth,-
zwdth); //21
    figure_points<<Vector(main_size,0,-zwdth); //22
    figure_points<<Vector(main_size-2*deg*wdth,0,-zwdth); //23
    figure_points<<Vector(wdth,-wdth-2*deg*wdth + main_size,-zwdth); //24
    figure_points<<Vector(wdth,0,-zwdth); //25

    figure_t figure;

    figure<<0<<1<<2<<3<<4<<5<<6<<7<<
            8<<9<<10<<11<<12;
    figures<<figure;
    figure.clear();

    figure<<13<<14<<15<<16<<17<<18<<19<<
            20<<21<<22<<23<<24<<25;
    figures<<figure;
    figure.clear();


    figure<<0<<1<<14<<13;
    figures<<figure;
    figure.clear();

    figure<<1<<2<<15<<14;
    figures<<figure;
    figure.clear();

    figure<<2<<3<<16<<15;
    figures<<figure;
    figure.clear();

    figure<<3<<4<<17<<16;
    figures<<figure;
    figure.clear();

    figure<<4<<5<<18<<17;
    figures<<figure;
    figure.clear();
```

```
    figure<<5<<6<<19<<18;
    figures<<figure;
    figure.clear();

    figure<<6<<7<<20<<19;
    figures<<figure;
    figure.clear();

    figure<<7<<8<<21<<20;
    figures<<figure;
    figure.clear();

    figure<<8<<9<<22<<21;
    figures<<figure;
    figure.clear();

    figure<<9<<10<<23<<22;
    figures<<figure;
    figure.clear();

    figure<<10<<11<<24<<23;
    figures<<figure;
    figure.clear();

    figure<<11<<12<<25<<24;
    figures<<figure;
    figure.clear();

    figure<<12<<0<<13<<25;
    figures<<figure;
    figure.clear();

    for(int i=0;i<=MaxColor;++i){
        zpalette[i]=QPen(QColor::fromHsl(i,255,127));
    }
    //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 Bresenham(points &figure,const bool for_fill,int xn,int yn,int zn,int xk,int
yk,int zk){
    int  dxn, dyn, dzn, erry,errz, sx, sy,sz,incr1y, incr2y,incr1z,incr2z;
    int *dx,*dy,*dz,*x,*y,*z;
    bool swapx=false,swapz=false;
    /* Вычисление приращений и шагов */
    bool next_y=true;

   dxn= xk-xn;

   dyn=yk-yn;

   dzn=zk-zn;

   if (abs(dyn)<=abs(dxn)) {
       if(abs(dzn)<=abs(dyn)){
           dx=&dxn;  dy= &dyn; dz=&dzn;
           x=&xn; y=&yn; z=&zn;
```

```
    } else {
        if(abs(dzn)<abs(dxn)){
            swapz=true;
            dx=&dxn;  dy= &dzn; dz=&dyn;
            x=&xn; y=&zn; z=&yn;
        } else {
            swapz=true;
            dx=&dzn;  dy= &dxn; dz=&dyn;
            x=&zn; y=&xn; z=&yn;
        }
    }
} else {
    if(abs(dzn)<abs(dxn)){
        swapx=true;
        dx=&dyn;  dy= &dxn; dz=&dzn;
        x=&yn; y=&xn; z=&zn;
    } else {
        if(abs(dzn)<abs(dyn)){
            swapx=true;
            dx=&dyn;  dy= &dzn; dz=&dxn;
            x=&yn; y=&zn; z=&xn;
        } else {
            dx=&dzn;  dy= &dyn; dz=&dxn;
            x=&zn; y=&yn; z=&xn;
        }
    }
}

sx=(*dx>0)?1:(*dx<0)?-1:0;
sy=(*dy>0)?1:(*dy<0)?-1:0;
sz=(*dz>0)?1:(*dz<0)?-1:0;
*dx=abs(*dx); *dy=abs(*dy); *dz=abs(*dz);

incr1y= 2**dy;
incr2y= 2**dx;
erry= (incr1y)-*dx;

incr1z=2**dz;
incr2z=2**dy;
errz=(incr1z)-*dy;

next_y=false;
if(!for_fill||next_y){
 SimplePoint t;
 t.x=xn;t.y=yn,t.z=zn;t.next_y=next_y;
 figure<<t;
}

while (--(*dx) >= 0) {
   next_y=false;
    if (erry >= 0) {
      if(errz>=0){
          *z+=sz;
          errz-=incr2z;
          if(swapz)
              next_y=true;
      }
      if(!swapx&&!swapz)
          next_y=true;
      *y+= sy; //zn+=k*sqrt(sx*sx+sy*sy);} else {next_y=true;yn+= sy;}
      erry-= incr2y;
      errz+= incr1z;
   }
   if(swapx)
       next_y=true;
   *x+= sx;// zn+=k*sqrt(sx*sx+sy*sy);}
```

```
      erry+=  incr1y;
      if(!for_fill||next_y){
       SimplePoint t;
       t.x=xn;t.y=yn,t.z=zn;t.next_y=next_y;
       figure<<t;
      }
   }
}
bool lessByY(const SimplePoint &a,const SimplePoint &b){
    return a.y<b.y;
}

bool lessByX(const SimplePoint &a,const SimplePoint &b){
    return a.x<b.x;
}

void drawFigure(QPainter &painter,MatrixT<int> &zbuffer, points& figure,double
zmin,double zdelta){
    qSort(figure.begin(),figure.end(),lessByY);
    QList<SimplePoint> line;


    int i=0,n=figure.size();

    while(i<n){
        int j=i;
        line.clear();
        int yn=figure.at(i).y;

        while(j<n&&figure.at(j).y==yn){
            if(figure.at(j).next_y)
                line<<figure.at(j);
            j++;
        }

        qSort(line.begin(),line.end(),lessByX);

        if(!(line.size()%2))
          for(int k=0;k<line.size();k+=2){
            int max_x=line.at(k+1).x,min_x=line.at(k).x;
            double max_z=line.at(k+1).z,min_z=line.at(k).z;
            double zk=(max_z-min_z)/(max_x-min_x);
            double zn=min_z;
            for(int xn=min_x+1;xn<=max_x-1;xn++){
                if(xn>=0&&
                   yn>=0&&
                   xn<zbuffer.m&&
                   yn<zbuffer.n){
                    zn+=zk;
                    if(zn>zbuffer[xn][yn]){

                        zbuffer[xn][yn]=(int)zn;
                        int c=(zn-zmin)/zdelta*MaxColor;

                        painter.setPen(zpalette[c]);

                        painter.drawPoint(xn,yn);
                    }
                }
            }
          }
        //qDebug()<<yn;
        i=j;
    }
}
```

```
void drawContour(QPainter &painter,MatrixT<int> &zbuffer, points& figure){
    painter.setPen(QColor("Black"));
    for(int i=0;i<figure.size();++i){
        int xn=figure.at(i).x,yn=figure.at(i).y,zn=figure.at(i).z;
        if(xn>=0&&
           yn>=0&&
           xn<zbuffer.m&&
           yn<zbuffer.n){
               if(zn+1>=zbuffer[xn][yn]){
                zbuffer[xn][yn]=zn+1;

                painter.drawPoint(xn,yn);
            }
        }
    }
}

void CompGraphView::paintEvent(QPaintEvent *event){
    //Рисовалки
    QPainter painter(viewport()); QPainterPath path;
    QPen pen(QColor("Red"));QBrush brush(QColor("Red"));
    int n=viewport()->width(),m=viewport()->height();
    MatrixT<int> zbuffer(m,n);
    points pix;
    for(int i=0;i<n;++i){
        for(int j=0;j<m;++j){
            zbuffer[i][j]=INT_MIN;
        }
    }
    //Текущая точка и конечная матрица преобразований
    Vector t(4),t0(4),t1(4),th(4);Matrix M(4,4);
    bool horiz=false;
    //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!
    QList<Vector> new_points;

    double minz,maxz;
    new_points<<M*figure_points.first();
    minz=new_points.first()[2];maxz=new_points.first()[2];

    for (int i=1;i<figure_points.size();++i){
        new_points<<M*figure_points.at(i);
        if(new_points.at(i)[2]>maxz)
            maxz=new_points.at(i)[2];
        else if(new_points.at(i)[2]<minz)
            minz=new_points.at(i)[2];
    }

    double deltaz=maxz-minz;
    for (int j=0;j<figures.size();++j){
        horiz=false;
```

```
            figure_t figure=figures[j];

            t0=M*figure_points.at(figure[0]);
            t=M*figure_points.at(figure[1]);
            for (int i=2;i<figure.size();++i){
                //Получаем точку и строим до неё линию до предыдущей
                t1=M*figure_points.at(figure[i]);
                Bresenham(pix,false,t0[0],t0[1],t0[2],t[0],t[1],t[2]);
                //painter.drawLine(t0[0],t0[1],t[0],t[1]);
                if(horiz){
                    t0=th;
                    horiz=false;
                }
                if(((int)t1[1]>(int)t[1]&&(int)t0[1]>(int)t[1])||
                   ((int)t1[1]<(int)t[1]&&(int)t0[1]<(int)t[1])){
                    SimplePoint p;
                    p.x=t[0];p.y=t[1];p.z=t[2];p.next_y=true;
                    pix<<p;
                }
                if((int)t[1]==(int)t1[1]){
                    horiz=true;
                    th=t0;
                }
                t0=t;
                t=t1;
                //path.lineTo(t.getPoint());
            }
            t1=M*figure_points.at(figure.first());
            Bresenham(pix,false,t0[0],t0[1],t0[2],t[0],t[1],t[2]);
            if(horiz){
                t0=th;
                horiz=false;
            }
            if(((int)t1[1]>(int)t[1]&&(int)t0[1]>(int)t[1])||
               ((int)t1[1]<(int)t[1]&&(int)t0[1]<(int)t[1])){
                SimplePoint p;
                p.x=t[0];p.y=t[1];p.z=t[2];p.next_y=true;
                pix<<p;
            }
            if((int)t[1]==(int)t1[1]){
                horiz=true;
                th=t0;
            }
            t0=t;
            t=t1;
            t1=M*figure_points.at(figure.at(1));
            Bresenham(pix,false,t0[0],t0[1],t0[2],t[0],t[1],t[2]);
            if(horiz){
                t0=th;
                horiz=false;
            }
            if(((int)t1[1]>(int)t[1]&&(int)t0[1]>(int)t[1])||
               ((int)t1[1]<(int)t[1]&&(int)t0[1]<(int)t[1])){
                SimplePoint p;
                p.x=t[0];p.y=t[1];p.z=t[2];p.next_y=true;
                pix<<p;
            }

            drawFigure(painter,zbuffer,pix,minz,deltaz);
            drawContour(painter,zbuffer,pix);

            pix.clear();

        }

    }
```

```
//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;
};
```

## Тестовый пример

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

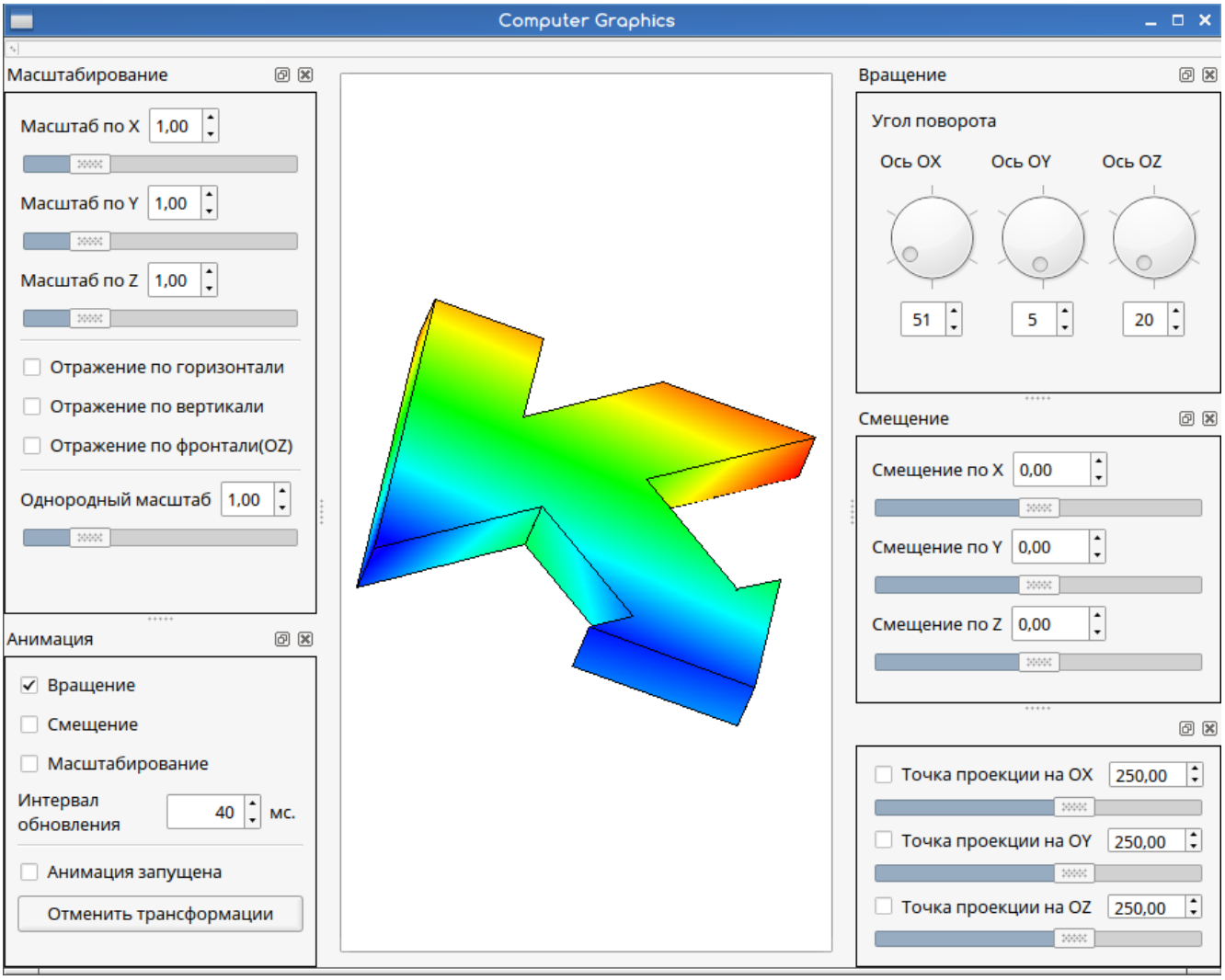


Рисунок 1— Пример работы разработанной программы.

## Вывод

В ходе выполнения данной лабораторной работы я освоил математические и алгоритмические основы удаления невидимых линий и поверхностей, а также написал программу, позволяющую рисовать символ «Антивируса Касперского» с учетом видимости поверхностей.