ГУАП

КАФЕДРА № 14

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

доцент, канд. техн. наук		А.В. Шахомиров
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ

3D-ФИГУРЫ, УДАЛЕНИЕ НЕВИДИМЫХ ПОВЕРХНОСТЕЙ И ЗАКРАСКА

по курсу: КОМПЬЮТЕРНАЯ ГРАФИКА

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. № 1041		М.А. Смоляков
	подпись, дата	инициалы, фамилия

Санкт-Петербург 2022

1. Цель работы

Изучение методов работы с библиотеками для графики, рисования и изменения проекций трёхмерных фигур и определение видимых поверхностей.

2. Постановка задачи

С помощью библиотеки для графики нарисовать трёхгранную пирамиду, подписать её точки, а также реализовать методы работы с ней:

- перемещение;
- вращение;
- масштабирование;
- заливка видимых поверхностей.

3. Формализация задачи

При запуске программы создаётся окно размерами 1400 на 700 функцией initwindow. Для создания и работы с точками используется класс Point, хранящий координаты x, y и z точки её имя и метод для отображения её имени. Линии между точками отрисовываются при помощи функции line_DDA алгоритмом рисования DDA-линии. Класс Surface хранит имя поверхности (грани призмы), состояние её видимости и цвет, которым она должна быть закрашена. В классе Prism содержатся данные для призмы: восемь её точек, их имена, цвет линий. Конструктор класса задаёт начальные координаты точек, их имена и отрисовывает треугольник на экране при помощи метода класса drawPrism. Для задания цветов используются следующие обозначения: TEXTCOL – цвет текста, MAINCOL – цвет границ треугольника и WHITE – белый цвет. Перемещение осуществляется тремя методами: moveX, moveY и moveZ, сдвигающим фигуру вертикально, горизонтально, на нас и от нас соответственно. Методы rotateZ, rotateY и rotateX поворачивают фигуру по или против часовой стрелки вокруг осей z, y и x соответственно. Метод scale масштабирует фигуру. Метод colouring обеспечивает закраску видимых граней фигуры. Вызывая методы seenL для проверки на видимость пересекающихся линий (определяется методом cross) и seenS для определения видимости поверхностей, если никакие линии не пересекаются. Метод fill заливает каждую поверхность, у которой поднят флаг видимости. По завершении каждой из функций экран отчищается и выводится новая отредактированная призма. Управление производится при помощи следующих клавиш (без учёта раскладки и регистра):

W – перемещение вверх;
A – перемещение влево;
S – перемещение вниз;
D – перемещение вправо;
Z – перемещение вперёд;
X – перемещение назад;
Q, E – повороты против и по часовой стрелки вокруг оси z;
R, T – поворот против и по часовой стрелки вокруг оси y;
F, G – поворот против и по часовой стрелки вокруг оси x;
- – уменьшение в масштабе;
+ – увеличение в масштабе.

Нажатие любой другой клавиши приводит к завершению программы.

По завершении программы функция closegraph освобождает всю память, выделенную под графическую систему, затем восстанавливает экран в режим, который был до вызова initwindow.

Использованное ПО:

Microsoft Visual Studio Enterprise 2019.

Версия компилятора: 16.11.8

Библиотека graphics.h (https://github.com/ahuynh359/Graphics).

4. Тестовый пример

Точка A (50; 170; 0).
Точка B (170; 170; 0).
Точка C (170; 170; -36).
Точка D (50; 170; -36).
Точка E (62; 50; -12).
Точка F (158; 50; -12).
Точка G (158; 50; -24).
Точка H (62; 50; -24).

5. Листинг программы

```
#include <iostream>
#include <math.h>
#include "graphics.h"
#pragma comment(lib,"graphics.lib")

#define P 3.14
#define TEXTCOL 15
#define MAINCOL 10 // светло-зелёный
#define WHITE 15
#define BLACK 0

#define GREEN 2
#define CYAN 11
#define BLUE 9
#define RED 4
#define YELLOW 14
#define MAGENTA 13

using namespace std;


// класс для точек
class Point {
public:
   float x;
   float y;
   float z;
   char* name;
   void namePoint(char* name) {
      int tmpX = x - 0.5 * z;
      int tmpY = y + 0.5 * z;
      outtextxy(tmpX, tmpY, name);
   }
};

// класс поверхностей
class Surface {
public:
   char* name;
   bool isVisible;
```

```
    COLORREF colour;

    Surface(char* nam, COLORREF col) {
        name = nam;
        isVisible = true;
        colour = col;
    }
};

// класс фигуры
class Prism {
public:
    Point A, B, C, D, E, F, G, H;;
    char name_A[2] = "A";
    char name_B[2] = "B";
    char name_C[2] = "C";
    char name_D[2] = "D";
    char name_E[2] = "E";
    char name_F[2] = "F";
    char name_G[2] = "G";
    char name_H[2] = "H";
    int col = MAINCOL;

    char name_ADCB[5] = "ADCB";
    char name_ADHE[5] = "ADHE";
    char name_EHGF[5] = "EHGF";
    char name_FGCB[5] = "FGCB";
    char name_ABFE[5] = "ABFE";
    char name_DCGH[5] = "DCGH";

    Surface ADCB = Surface(name_ADCB, RED);
    Surface ADHE = Surface(name_ADHE, GREEN);
    Surface EHGF = Surface(name_EHGF, CYAN);
    Surface FGCB = Surface(name_FGCB, YELLOW);
    Surface ABFE = Surface(name_ABFE, BLUE);
    Surface DCGH = Surface(name_DCGH, MAGENTA);


    // конструктор
    Prism() {
        A.x = 50; A.y = 170; A.z = 0;
        A.name = name_A;
        B.x = 170; B.y = 170; B.z = 0;
        B.name = name_B;
        C.x = 170; C.y = 170; C.z = -36;
        C.name = name_C;
        D.x = 50; D.y = 170; D.z = -36;
        D.name = name_D;

        E.x = 62; E.y = 50; E.z = -12;
        E.name = name_E;
        F.x = 158; F.y = 50; F.z = -12;
```

```
    F.name = name_F;
    G.x = 158; G.y = 50; G.z = -24;
    G.name = name_G;
    H.x = 62; H.y = 50; H.z = -24;
    H.name = name_H;
    drawPrism();
}

// отрисовка
void drawPrism() {
    // оси
    setcolor(WHITE);
    line(700, 350, 730, 350);
    line(700, 350, 700, 320);
    line(700, 350, 685, 365);

    char w[20] = "W - Вверх";
    char a[20] = "A - Влево";
    char s[20] = "S - Вниз";
    char d[20] = "D - Вправо";
    char z[20] = "Z - Вперёд";
    char x[20] = "X - Назад";
    char qrf[40] = "Q, R, F - Повороты против стрелки";
    char etg[30] = "E, T, G - Повороты по стрелке";
    char plus[20] = "+ - Увеличить";
    char minus[20] = "- - Уменьшить";
    char other[30] = "Любая другая - Выход";

    // вывод инструкций
    outtextxy(1140, 20, w);
    outtextxy(1140, 40, a);
    outtextxy(1140, 60, s);
    outtextxy(1140, 80, d);
    outtextxy(1140, 100, z);
    outtextxy(1140, 120, x);
    outtextxy(1140, 140, qrf);
    outtextxy(1140, 160, etg);
    outtextxy(1140, 180, plus);
    outtextxy(1140, 200, minus);
    outtextxy(1140, 220, other);


    // Вывод имён точек
    setcolor(TEXTCOL);
    A.namePoint(A.name);
    B.namePoint(B.name);
    C.namePoint(C.name);
    D.namePoint(D.name);
    E.namePoint(E.name);
    F.namePoint(F.name);
    G.namePoint(G.name);
    H.namePoint(H.name);
```

```
    setcolor(MAINCOL);
    // Учёт координаты z при отрисовке в двумерном пространстве при задании x и y

    // Нижнее основание
    line(A.x - 0.5 * A.z, A.y + 0.5 * A.z, B.x - 0.5 * B.z, B.y + 0.5 * B.z); // линия 1
    line(B.x - 0.5 * B.z, B.y + 0.5 * B.z, C.x - 0.5 * C.z, C.y + 0.5 * C.z); // линия 2
    line(C.x - 0.5 * C.z, C.y + 0.5 * C.z, D.x - 0.5 * D.z, D.y + 0.5 * D.z); // линия 3
    line(D.x - 0.5 * D.z, D.y + 0.5 * D.z, A.x - 0.5 * A.z, A.y + 0.5 * A.z); // линия 4

    // Верхнее основание
    line(E.x - 0.5 * E.z, E.y + 0.5 * E.z, F.x - 0.5 * F.z, F.y + 0.5 * F.z); // линия 5
    line(F.x - 0.5 * F.z, F.y + 0.5 * F.z, G.x - 0.5 * G.z, G.y + 0.5 * G.z); // линия 6
    line(G.x - 0.5 * G.z, G.y + 0.5 * G.z, H.x - 0.5 * H.z, H.y + 0.5 * H.z); // линия 7
    line(H.x - 0.5 * H.z, H.y + 0.5 * H.z, E.x - 0.5 * E.z, E.y + 0.5 * E.z); // линия 8

    // Боковые грани
    line(A.x - 0.5 * A.z, A.y + 0.5 * A.z, E.x - 0.5 * E.z, E.y + 0.5 * E.z); // линия 9
    line(B.x - 0.5 * B.z, B.y + 0.5 * B.z, F.x - 0.5 * F.z, F.y + 0.5 * F.z); // линия 10
    line(C.x - 0.5 * C.z, C.y + 0.5 * C.z, G.x - 0.5 * G.z, G.y + 0.5 * G.z); // линия 11
    line(D.x - 0.5 * D.z, D.y + 0.5 * D.z, H.x - 0.5 * H.z, H.y + 0.5 * H.z); // линия 12

    // Закраска граней фигуры
    colouring();

}

// пермещение
void moveX(float amt) {
    A.x += amt;
    B.x += amt;
    C.x += amt;
    D.x += amt;
    E.x += amt;
    F.x += amt;
    G.x += amt;
    H.x += amt;
}
void moveY(float amt) {
    A.y += amt;
    B.y += amt;
    C.y += amt;
    D.y += amt;
    E.y += amt;
    F.y += amt;
    G.y += amt;
    H.y += amt;
}
void moveZ(float amt) {
    A.z += amt;
    B.z += amt;
```

```
    C.z += amt;
    D.z += amt;
    E.z += amt;
    F.z += amt;
    G.z += amt;
    H.z += amt;
}

// поворот одной точки вокруг z
Point rotDotZ(int u, float ang, Point Cen, Point L) {
    L.x = L.x - Cen.x; // расстояние от a до центра по x
    L.y = L.y - Cen.y; // по y

    float tmpX = L.x * cos(ang) + L.y * sin(ang);
    float tmpY = -L.x * sin(ang) + L.y * cos(ang);
    L.x = tmpX + Cen.x;
    L.y = tmpY + Cen.y;

    return L;
}
// поворот фигуры вокруг z
void rotateZ(int u) { // u = -1 по часовой, u = 1 против
    float ang = u * 0.05; // угол поворота

    Point Cen; // точка центра
    Cen.x = (A.x + B.x + C.x + D.x + E.x + F.x + G.x + H.x) / 8;
    Cen.y = (A.y + B.y + C.y + D.y + E.y + F.y + G.y + H.y) / 8;
    Cen.z = (A.z + B.z + C.z + D.z + E.z + F.z + G.z + H.z) / 8;

    A = rotDotZ(u, ang, Cen, A);
    B = rotDotZ(u, ang, Cen, B);
    C = rotDotZ(u, ang, Cen, C);
    D = rotDotZ(u, ang, Cen, D);

    E = rotDotZ(u, ang, Cen, E);
    F = rotDotZ(u, ang, Cen, F);
    G = rotDotZ(u, ang, Cen, G);
    H = rotDotZ(u, ang, Cen, H);
}

// поворот одной точки вокруг y
Point rotDotY(int u, float ang, Point Cen, Point L) {
    L.x = L.x - Cen.x; // расстояние от a до центра по y
    L.z = L.z - Cen.z; // по z

    float tmpX = L.x * cos(ang) + L.z * sin(ang);
    float tmpZ = -L.x * sin(ang) + L.z * cos(ang);
    L.x = tmpX + Cen.x;
    L.z = tmpZ + Cen.z;

    return L;
}
```

```
// поворот фигуры вокруг y
void rotateY(int u) { // u = -1 по часовой, u = 1 против
   float ang = u * 0.05; // угол поворота

   Point Cen; // точка центра
   Cen.x = (A.x + B.x + C.x + D.x + E.x + F.x + G.x + H.x) / 8;
   Cen.y = (A.y + B.y + C.y + D.y + E.y + F.y + G.y + H.y) / 8;
   Cen.z = (A.z + B.z + C.z + D.z + E.z + F.z + G.z + H.z) / 8;

   A = rotDotY(u, ang, Cen, A);
   B = rotDotY(u, ang, Cen, B);
   C = rotDotY(u, ang, Cen, C);
   D = rotDotY(u, ang, Cen, D);

   E = rotDotY(u, ang, Cen, E);
   F = rotDotY(u, ang, Cen, F);
   G = rotDotY(u, ang, Cen, G);
   H = rotDotY(u, ang, Cen, H);
}

// поворот одной точки вокруг x
Point rotDotX(int u, float ang, Point Cen, Point L) {
   L.y = L.y - Cen.y; // расстояние от a до центра по y
   L.z = L.z - Cen.z; // по z

   float tmpY = L.y * cos(ang) + L.z * sin(ang);
   float tmpZ = -L.y * sin(ang) + L.z * cos(ang);
   L.y = tmpY + Cen.y;
   L.z = tmpZ + Cen.z;

   return L;
}
// поворот фигуры вокруг x
void rotateX(int u) { // u = -1 по часовой, u = 1 против
   float ang = u * 0.05; // угол поворота

   Point Cen; // точка центра
   Cen.x = (A.x + B.x + C.x + D.x + E.x + F.x + G.x + H.x) / 8;
   Cen.y = (A.y + B.y + C.y + D.y + E.y + F.y + G.y + H.y) / 8;
   Cen.z = (A.z + B.z + C.z + D.z + E.z + F.z + G.z + H.z) / 8;

   A = rotDotX(u, ang, Cen, A);
   B = rotDotX(u, ang, Cen, B);
   C = rotDotX(u, ang, Cen, C);
   D = rotDotX(u, ang, Cen, D);

   E = rotDotX(u, ang, Cen, E);
   F = rotDotX(u, ang, Cen, F);
   G = rotDotX(u, ang, Cen, G);
   H = rotDotX(u, ang, Cen, H);
}
```

```
// масштабирование одной точки
Point dotScale(float e, Point Cen, Point L) {
   // L.x
   float xe = (Cen.x + L.x) / 2;
   float lx = Cen.x - L.x;
   lx = lx * e;
   L.x = xe - lx / 2;
   // L.y
   float ye = (Cen.y + L.y) / 2;
   float ly = Cen.y - L.y;
   ly = ly * e;
   L.y = ye - ly / 2;
   // L.z
   float ze = (Cen.z + L.z) / 2;
   float lz = Cen.z - L.z;
   lz = lz * e;
   L.z = ze - lz / 2;

   return L;
}
// масштабирование всей фигуры
void scale(float e) {
   Point Cen; // точка центра
   Cen.x = (A.x + B.x + C.x + D.x + E.x + F.x + G.x + H.x) / 8;
   Cen.y = (A.y + B.y + C.y + D.y + E.y + F.y + G.y + H.y) / 8;
   Cen.z = (A.z + B.z + C.z + D.z + E.z + F.z + G.z + H.z) / 8;

   if (((abs(A.x - Cen.x) >= 1 && abs(A.y - Cen.y) >= 1 && abs(A.z - Cen.z) >= 1) &&
      (abs(B.x - Cen.x) >= 1 && abs(B.y - Cen.y) >= 1 && abs(B.z - Cen.z) >= 1) &&
      (abs(C.x - Cen.x) >= 1 && abs(C.y - Cen.y) >= 1 && abs(C.z - Cen.z) >= 1) &&
      (abs(D.x - Cen.x) >= 1 && abs(D.y - Cen.y) >= 1 && abs(D.z - Cen.z) >= 1) &&
      (abs(E.x - Cen.x) >= 1 && abs(E.y - Cen.y) >= 1 && abs(E.z - Cen.z) >= 1) &&
      (abs(F.x - Cen.x) >= 1 && abs(F.y - Cen.y) >= 1 && abs(F.z - Cen.z) >= 1) &&
      (abs(G.x - Cen.x) >= 1 && abs(G.y - Cen.y) >= 1 && abs(G.z - Cen.z) >= 1) &&
      (abs(H.x - Cen.x) >= 1 && abs(H.y - Cen.y) >= 1 && abs(H.z - Cen.z) >= 1)
      ) || e > 1) { // предотвращение сжатия в точку

      A = dotScale(e, Cen, A);
      B = dotScale(e, Cen, B);
      C = dotScale(e, Cen, C);
      D = dotScale(e, Cen, D);

      E = dotScale(e, Cen, E);
      F = dotScale(e, Cen, F);
      G = dotScale(e, Cen, G);
      H = dotScale(e, Cen, H);
   }

}


Point dot;  // точка пересечения
```

```
// проверка на пересечение линий
bool cross(Point a1, Point a2, Point a3, Point a4) {
    Point p1 = a1, p2 = a2, p3 = a3, p4 = a4;

    // учёт координаты z при отрисовке в двумерном пространстве
    // точка пересечения смотрится не прямо вдоль оси z, а под углом 45, как видит пользователь
    p1.x -= 0.5 * p1.z;     p1.y += 0.5 * p1.z;
    p2.x -= 0.5 * p2.z;     p2.y += 0.5 * p2.z;
    p3.x -= 0.5 * p3.z;     p3.y += 0.5 * p3.z;
    p4.x -= 0.5 * p4.z;     p4.y += 0.5 * p4.z;

    // расстановка точек так, чтобы начальная точка находилась левее конечной относительно оси x
    if (p2.x < p1.x) {
        Point tmp = p1;
        p1 = p2;
        p2 = tmp;
    }
    if (p4.x < p3.x) {
        Point tmp = p3;
        p3 = p4;
        p4 = tmp;
    }

    // если конец первого отрезка находится левее начала второго, то отрезки точно не пересекаются
    if (p2.x < p3.x) { return false; }

    // если оба отрезка вертикальные
    if ((p1.x - p2.x == 0) && (p3.x - p4.x == 0)) {
        // если они лежат на одном X
        if (p1.x == p3.x) {
            // проверка пересекаются ли они, т.е. есть ли у них общий Y
            // берётся отрицание от случая, когда они НЕ пересекаются
            if (!((max(p1.y, p2.y) < min(p3.y, p4.y)) ||
                (min(p1.y, p2.y) > max(p3.y, p4.y)))) {
                dot.x = p1.x;
                dot.y = (p1.y + p2.y) / 2;
                return true;
            }
        }
        return false;
    }

    // если первый отрезок вертикальный
    if (p1.x - p2.x == 0) {
        // Xa, Ya - точки пересечения двух прямых
        double Xa = p1.x;
        double A2 = (p3.y - p4.y) / (p3.x - p4.x); // A — тангенс угла между прямой и осью x
        double b2 = p3.y - A2 * p3.x; // b — смешение относительно оси
        double Ya = A2 * Xa + b2;
```

```
        // проверка, что точка принадлежит отрезкам
        if (p3.x <= Xa && p4.x >= Xa && min(p1.y, p2.y) <= Ya && max(p1.y, p2.y) >= Ya) {
            dot.x = Xa;
            dot.y = Ya;
            return true;
        }

        return false;
    }

    // если второй отрезок вертикальный
    if (p3.x - p4.x == 0) {
        // Xa, Ya - точки пересечения двух прямых
        double Xa = p3.x;
        double A1 = (p1.y - p2.y) / (p1.x - p2.x);
        double b1 = p1.y - A1 * p1.x;
        double Ya = A1 * Xa + b1;

        if (p1.x <= Xa && p2.x >= Xa && min(p3.y, p4.y) <= Ya && max(p3.y, p4.y) >= Ya) {
            dot.x = Xa;
            dot.y = Ya;
            return true;
        }

        return false;
    }

    // оба отрезка невертикальные
    double A1 = (p1.y - p2.y) / (p1.x - p2.x);
    double A2 = (p3.y - p4.y) / (p3.x - p4.x);
    double b1 = p1.y - A1 * p1.x;
    double b2 = p3.y - A2 * p3.x;

    if (A1 == A2) { return false; } // отрезки параллельны

    // Xa - абсцисса точки пересечения двух прямых
    double Xa = (b2 - b1) / (A1 - A2);
    double Ya = A1 * Xa + b1; // Ya - ордината

    // проверка, что точка персечения находится в границах отрезка
    if ((Xa < max(p1.x, p3.x)) || (Xa > min(p2.x, p4.x))) {
        return false; // точка Xa находится вне пересечения проекций отрезков на ось X
    }
    else {
        dot.x = Xa;
        dot.y = Ya;
        return true;
    }

}
```

```
// видимость пересекающихся линий. возвращает 1, если есть пересечение
int seenL(Point One, Point Two, Point Three, Point Four) {

    if (cross(One, Two, Three, Four)) {

        cout << "Lines " << One.name << Two.name << " and " << Three.name << Four.name << " cross at " << dot.x << ";" << dot.y << ".\n";

        // сравнение координаты z для точек с координатами точки пересечения на каждой из линий
        int x1 = One.x - 0.5 * One.z, x2 = Two.x - 0.5 * Two.z; // учёт координаты z при отрисовке в двумерном пространстве
        int z1 = One.z, z2 = Two.z;
        int x = dot.x;

        if ((x2 - x1) != 0) {

            int zOT = (((x - x1) * (z2 - z1)) / (x2 - x1)) + z1;
            x1 = Three.x - 0.5 * Three.z, x2 = Four.x - 0.5 * Four.z; // учёт координаты z при отрисовке в двумерном пространстве
            z1 = Three.z, z2 = Four.z;

            if ((x2 - x1) != 0) {
                int zTF = (((x - x1) * (z2 - z1)) / (x2 - x1)) + z1;

                if (zOT == zTF)
                    cout << "\n\n\tsame point\n\n";
                // если первая линия ближе к наблюдателю, чем вторая
                else if (zOT > zTF) {
                    cout << "line " << Three.name << Four.name << " is not seen.\n";
                    // если плоскость содержит обе точки невидимой линии
                    if (strstr(ADCB.name, Three.name) && strstr(ADCB.name, Four.name))
                        ADCB.isVisible = false; // то и она сама не видна
                    if (strstr(ADHE.name, Three.name) && strstr(ADHE.name, Four.name))
                        ADHE.isVisible = false;
                    if (strstr(EHGF.name, Three.name) && strstr(EHGF.name, Four.name))
                        EHGF.isVisible = false;
                    if (strstr(FGCB.name, Three.name) && strstr(FGCB.name, Four.name))
                        FGCB.isVisible = false;
                    if (strstr(ABFE.name, Three.name) && strstr(ABFE.name, Four.name))
                        ABFE.isVisible = false;
                    if (strstr(DCGH.name, Three.name) && strstr(DCGH.name, Four.name))
                        DCGH.isVisible = false;
                }
                // если вторая линия ближе к наблюдателю, чем первая
                else if (zOT < zTF) {
                    cout << "line " << One.name << Two.name << " is not seen.\n";
                    if (strstr(ADCB.name, One.name) && strstr(ADCB.name, Two.name))
                        ADCB.isVisible = false;
                    if (strstr(ADHE.name, One.name) && strstr(ADHE.name, Two.name))
                        ADHE.isVisible = false;
                    if (strstr(EHGF.name, One.name) && strstr(EHGF.name, Two.name))
```

```
                EHGF.isVisible = false;
            if (strstr(FGCB.name, One.name) && strstr(FGCB.name, Two.name))
                FGCB.isVisible = false;
            if (strstr(ABFE.name, One.name) && strstr(ABFE.name, Two.name))
                ABFE.isVisible = false;
            if (strstr(DCGH.name, One.name) && strstr(DCGH.name, Two.name))
                DCGH.isVisible = false;
          }

        }
      }


      return 1;
    }
    else return 0; // если линии не пересекаются

  }

  // видимость поверхностей, если линии не пересекаются
  int seenS(Point a1, Point a2, Point a3, Point a4, Point b1, Point b2, Point b3, Point b4) {

    int surf1 = (a1.z + a2.z + a3.z + a4.z) / 4; // первая плоскость
    int surf2 = (b1.z + b2.z + b3.z + b4.z) / 4; // вторая плоскость

    if (surf1 > surf2) { // если первая плоскость ближе к нам, чем вторая, то она видна
      cout << "\n\n\tSurface " << a1.name << a2.name << a3.name << a4.name << " is VISIBLE.\n\n";

      // если плоскость не содержит ни одной видимой точки
      if (strstr(ADCB.name, a1.name) == NULL && strstr(ADCB.name, a2.name) == NULL && strstr(ADCB.name, a3.name) == NULL && strstr(ADCB.name, a4.name) == NULL)
        ADCB.isVisible = false; // то и она сама не видна
      if (strstr(ADHE.name, a1.name) == NULL && strstr(ADHE.name, a2.name) == NULL && strstr(ADHE.name, a3.name) == NULL && strstr(ADHE.name, a4.name) == NULL)
        ADHE.isVisible = false;
      if (strstr(EHGF.name, a1.name) == NULL && strstr(EHGF.name, a2.name) == NULL && strstr(EHGF.name, a3.name) == NULL && strstr(EHGF.name, a4.name) == NULL)
        EHGF.isVisible = false;
      if (strstr(FGCB.name, a1.name) == NULL && strstr(FGCB.name, a2.name) == NULL && strstr(FGCB.name, a3.name) == NULL && strstr(FGCB.name, a4.name) == NULL)
        FGCB.isVisible = false;
      if (strstr(ABFE.name, a1.name) == NULL && strstr(ABFE.name, a2.name) == NULL && strstr(ABFE.name, a3.name) == NULL && strstr(ABFE.name, a4.name) == NULL)
        ABFE.isVisible = false;
      if (strstr(DCGH.name, a1.name) == NULL && strstr(DCGH.name, a2.name) == NULL && strstr(DCGH.name, a3.name) == NULL && strstr(DCGH.name, a4.name) == NULL)
        DCGH.isVisible = false;

      return 1;
    }
    else {
```

```
        cout << "\n\n\tSurface " << b1.name << b2.name << b3.name << b4.name << " is VISIBLE.\n\n";

        if (strstr(ADCB.name, b1.name) == NULL && strstr(ADCB.name, b2.name) == NULL && strstr(ADCB.name, b3.name) == NULL && strstr(ADCB.name, b4.name) == NULL)
            ADCB.isVisible = false;
        if (strstr(ADHE.name, b1.name) == NULL && strstr(ADHE.name, b2.name) == NULL && strstr(ADHE.name, b3.name) == NULL && strstr(ADHE.name, b4.name) == NULL)
            ADHE.isVisible = false;
        if (strstr(EHGF.name, b1.name) == NULL && strstr(EHGF.name, b2.name) == NULL && strstr(EHGF.name, b3.name) == NULL && strstr(EHGF.name, b4.name) == NULL)
            EHGF.isVisible = false;
        if (strstr(FGCB.name, b1.name) == NULL && strstr(FGCB.name, b2.name) == NULL && strstr(FGCB.name, b3.name) == NULL && strstr(FGCB.name, b4.name) == NULL)
            FGCB.isVisible = false;
        if (strstr(ABFE.name, b1.name) == NULL && strstr(ABFE.name, b2.name) == NULL && strstr(ABFE.name, b3.name) == NULL && strstr(ABFE.name, b4.name) == NULL)
            ABFE.isVisible = false;
        if (strstr(DCGH.name, b1.name) == NULL && strstr(DCGH.name, b2.name) == NULL && strstr(DCGH.name, b3.name) == NULL && strstr(DCGH.name, b4.name) == NULL)
            DCGH.isVisible = false;

        return 1;
    }

    return 0;
}

// заливка одной поверхности
void fill(Point p1, Point p2, Point p3, COLORREF col) {
    // учёт координаты z при отрисовке в двумерном пространстве
    // точка пересечения смотрится не прямо вдоль оси z, а под углом 45, как видит пользователь
    p1.x -= 0.5 * p1.z, p1.y += 0.5 * p1.z;
    p2.x -= 0.5 * p2.z, p2.y += 0.5 * p2.z;
    p3.x -= 0.5 * p3.z, p3.y += 0.5 * p3.z;

    double x1 = p1.x, y1 = p1.y;
    double x2 = p2.x, y2 = p2.y;
    double x3 = p3.x, y3 = p3.y;
    setcolor(col);

    // нахождение наивысшей, средней и низшей точек
    if (y2 < y1) {
        swap(y1, y2);
        swap(x1, x2);
    }
    if (y3 < y1) {
        swap(y1, y3);
        swap(x1, x3);
    }
    if (y2 > y3) {
```

```
        swap(y2, y3);
        swap(x2, x3);
    }

    float y_const[4]; // x0, y0, x1, y1

    // y1 - наивысшая точка, y2 - средняя точка, y3 - низшая точка
    for (int y = y1; y <= y2; y++) {
        y_const[1] = y_const[3] = y; // y0 y1
        y_const[0] = x1 + (x2 - x1) * ((y - y1) / (y2 - y1)); // x0
        y_const[2] = x1 + (x3 - x1) * ((y - y1) / (y3 - y1)); // x1
        line(y_const[0], y_const[1], y_const[2], y_const[3]);
    }
    for (int y = y2; y <= y3; y++) {
        y_const[1] = y_const[3] = y;
        y_const[0] = x2 + (x3 - x2) * ((y - y2) / (y3 - y2));
        y_const[2] = x1 + (x3 - x1) * ((y - y1) / (y3 - y1));
        line(y_const[0], y_const[1], y_const[2], y_const[3]);
    }
}

// закраска всех видимых поверхностей
void colouring() {

    // изначально все плоскости видны
    ADCB.isVisible = true;
    ADHE.isVisible = true;
    EHGF.isVisible = true;
    FGCB.isVisible = true;
    ABFE.isVisible = true;
    DCGH.isVisible = true;


    // для AB
    // несмежные рёбра и верх
    int ab = seenL(A, B, D, H) + seenL(A, B, C, G) +
        seenL(A, B, E, H) + seenL(A, B, H, G) + seenL(A, B, G, F) + seenL(A, B, F, E);
    // для BC
    int bc = seenL(C, B, A, E) + seenL(C, B, D, H) +
        seenL(C, B, E, H) + seenL(C, B, H, G) + seenL(C, B, G, F) + seenL(C, B, F, E);
    // для CD
    int cd = seenL(C, D, A, E) + seenL(C, D, B, F) +
        seenL(C, D, E, H) + seenL(C, D, H, G) + seenL(C, D, G, F) + seenL(C, D, F, E);
    // для DA
    int da = seenL(D, A, C, G) + seenL(D, A, B, F) +
        seenL(D, A, E, H) + seenL(D, A, H, G) + seenL(D, A, G, F) + seenL(D, A, F, E);

    // только несмежные рёбра, т.к. с низом уже сравнивалось
    int ef = seenL(E, F, C, G) + seenL(E, F, D, H); // EF
    int fg = seenL(F, G, A, E) + seenL(F, G, D, H); // FG
    int gh = seenL(G, H, A, E) + seenL(G, H, B, F); // GH
    int he = seenL(H, E, C, G) + seenL(H, E, B, F); // HE
```

```
        // если никакие линии не пересекаются
        if ((ab + bc + cd + da + ef + fg + gh + he) <= 0) {
            if(!seenS(A, B, C, D, E, F, G, H))
                if(!seenS(B, C, G, F, A, D, H, E))
                    seenS(A, B, F, E, D, C, G, H);
        }

        // заливка граней в соответствии с их видимостью
        if (ADHE.isVisible) {
            cout << "\n\tADHE is visible\n";
            fill(A, D, H, ADHE.colour);
            fill(A, E, H, ADHE.colour);
        }
        if (EHGF.isVisible) {
            cout << "\n\tEHGF is visible\n";
            fill(E, H, G, EHGF.colour);
            fill(E, F, G, EHGF.colour);
        }
        if (FGCB.isVisible) {
            cout << "\n\tFGCB is visible\n";
            fill(F, G, C, FGCB.colour);
            fill(F, B, C, FGCB.colour);
        }
        if (ABFE.isVisible) {
            cout << "\n\tABFE is visible\n";
            fill(A, B, F, ABFE.colour);
            fill(A, E, F, ABFE.colour);
        }
        if (DCGH.isVisible) {
            cout << "\n\tDCGH is visible\n";
            fill(D, C, G, DCGH.colour);
            fill(D, H, G, DCGH.colour);
        }
        if (ADCB.isVisible) {
            cout << "\n\tADCB is visible\n";
            fill(A, D, C, ADCB.colour);
            fill(A, B, C, ADCB.colour);
        }

    }

};



int main() {
    initwindow(1400, 700); // создаём консольное окно 1400 на 700
    Prism Pri; // создание фигуры

    // управление
    int i = 1; // условие выхода
```

```
while (i) {
    switch (getch()) {
    case 'w':
    case 'W':
    case 'ц':
    case 'Ц':
        cout << 'w' << endl;
        Pri.moveY(-10); // вверх
        break;
    case 'a':
    case 'A':
    case 'ф':
    case 'Ф':
        cout << 'a' << endl;
        Pri.moveX(-10); // влево
        break;
    case 's':
    case 'S':
    case 'ы':
    case 'Ы':
        cout << 's' << endl;
        Pri.moveY(10); // вниз
        break;
    case 'd':
    case 'D':
    case 'в':
    case 'В':
        cout << 'd' << endl;
        Pri.moveX(10); // вправо
        break;
    case 'x':
    case 'X':
    case 'ч':
    case 'Ч':
        cout << 'x' << endl;
        Pri.moveZ(-10); // назад
        Pri.scale(0.9);
        break;
    case 'z':
    case 'Z':
    case 'я':
    case 'Я':
        cout << 'z' << endl;
        Pri.moveZ(10); // вперёд
        Pri.scale(1.1);
        break;

        // вокруг z
    case 'q':
    case 'Q':
    case 'й':
    case 'Й':
```

```
    cout << 'q' << endl;
    Pri.rotateZ(1); // против часовой
    break;
case 'e':
case 'E':
case 'у':
case 'У':
    cout << 'e' << endl;
    Pri.rotateZ(-1); // по часовой
    break;
    // вокруг y
case 'r':
case 'R':
case 'к':
case 'К':
    cout << 'r' << endl;
    Pri.rotateY(1); // против часовой
    break;
case 't':
case 'T':
case 'е':
case 'Е':
    cout << 't' << endl;
    Pri.rotateY(-1); // по часовой
    break;
    // вокруг x
case 'f':
case 'F':
case 'а':
case 'А':
    cout << 'f' << endl;
    Pri.rotateX(1); // против часовой
    break;
case 'g':
case 'G':
case 'п':
case 'П':
    cout << 'g' << endl;
    Pri.rotateX(-1); // по часовой
    break;

case '=':
case '+':
    cout << '+' << endl;
    Pri.scale(1.5); // увеличение
    break;
case '-':
case '_':
    cout << '-' << endl;
    Pri.scale(0.5); // уменьшение
    break;
default:
```

```
            cout << "default -> exit" << endl;
            i = 0;
            break;
        }
        cleardevice(); // отичстка экрана
        Pri.drawPrism(); // перерисовка фигуры
    }

    getch(); // чтение одного символа с клавиатуры
    closegraph(); // освобождает всю память, выделенную под графическую систему, затем восстанавливает экран в режим, который был до вызова initwindow

    return 0;
}
```

6. Результаты работы программы

Результаты работы программы представлены на рисунках 1-11.

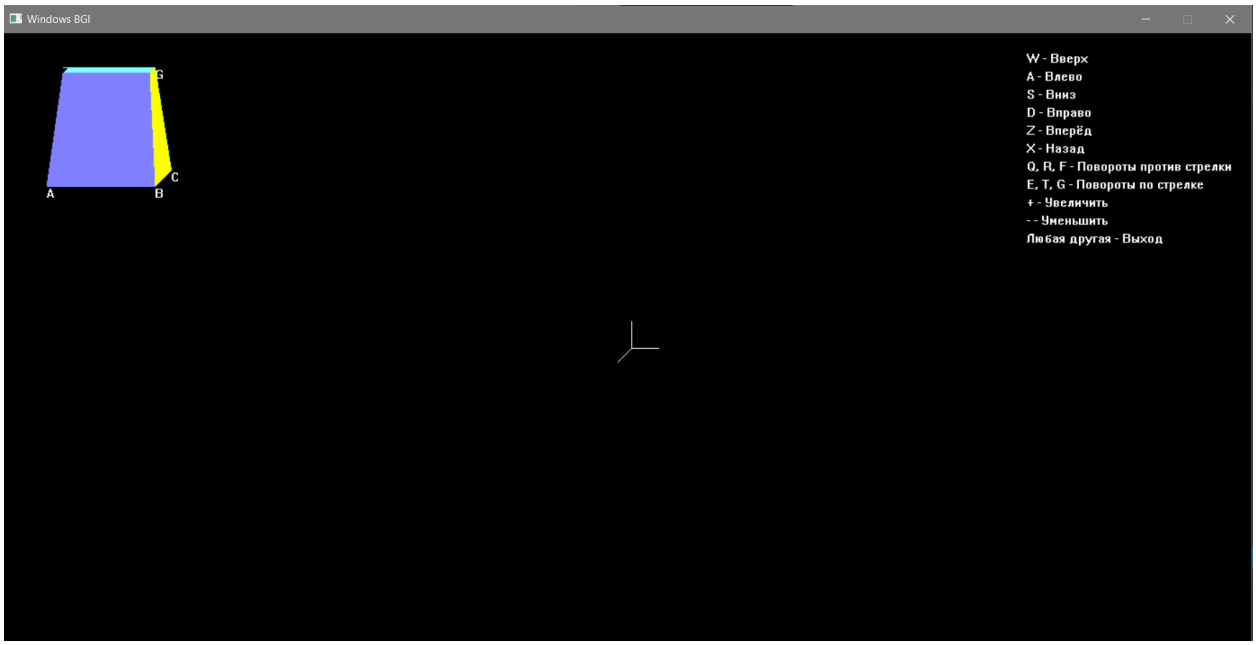

Рисунок 1 – Отрисовка фигуры

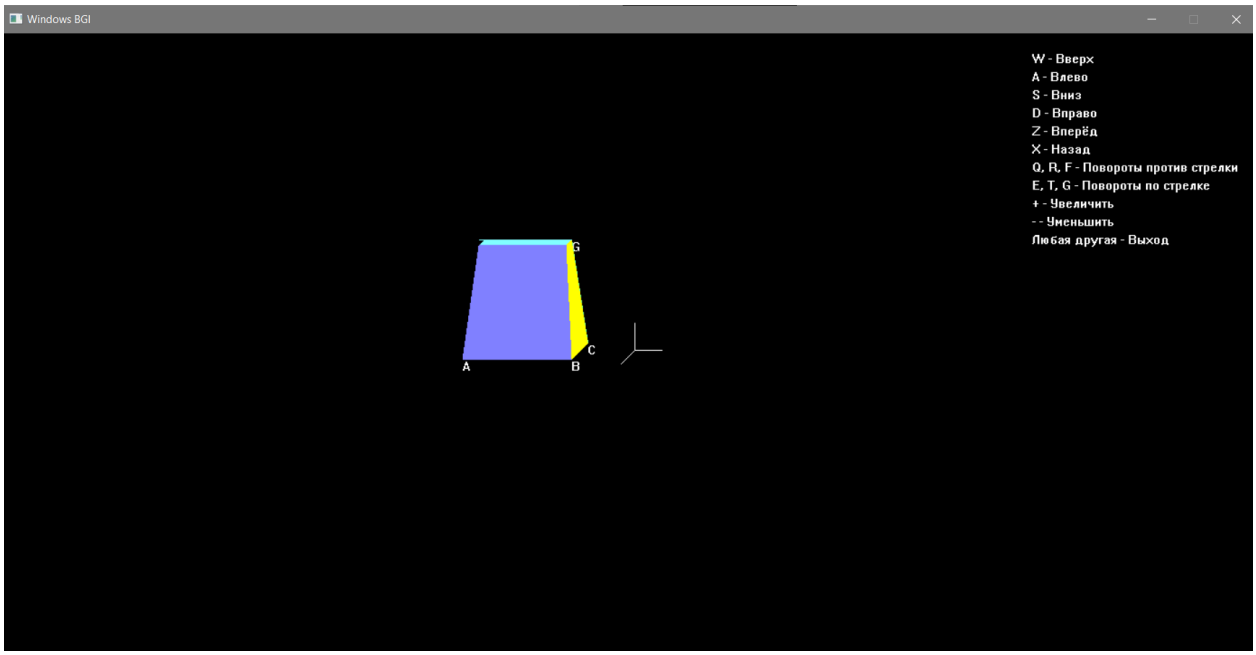

Рисунок 2 – Перемещение фигуры вдоль осей x и y

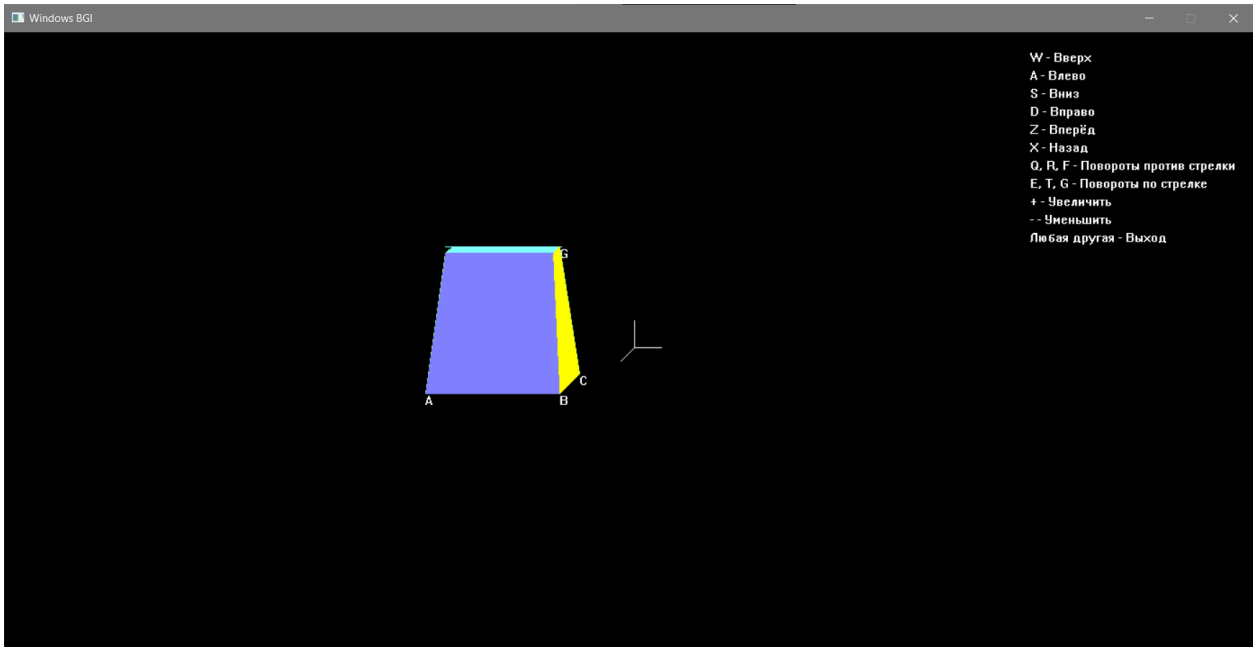

Рисунок 3 – Перемещение фигуры вдоль оси z

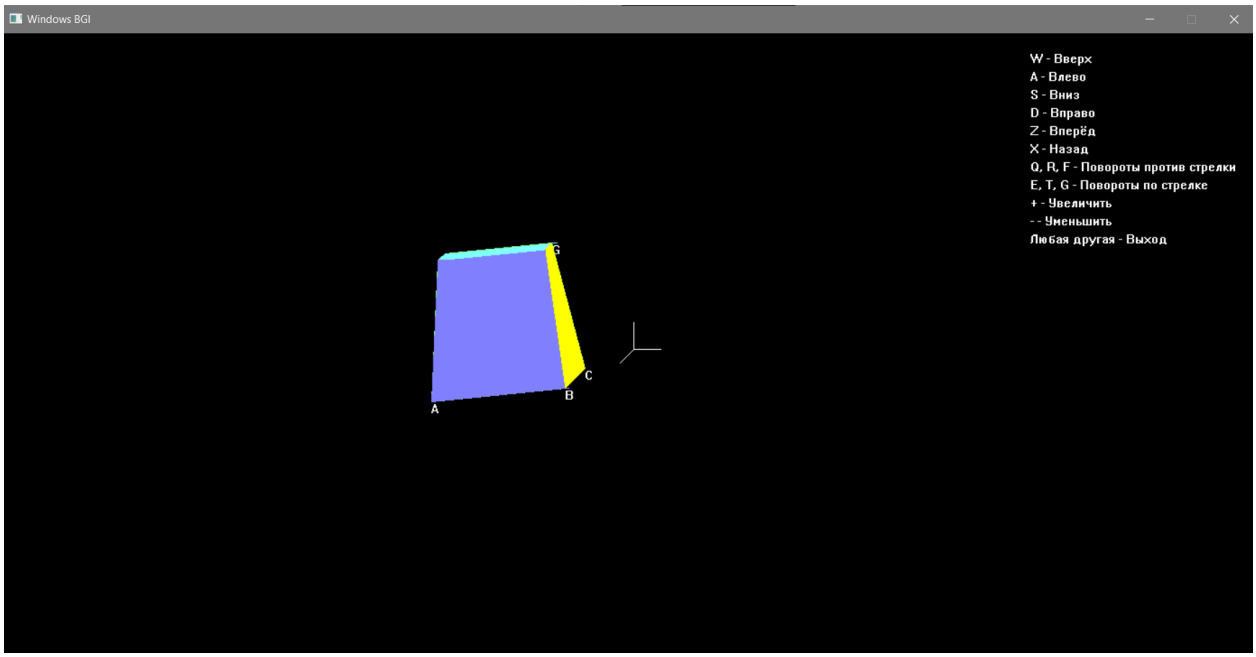

Рисунок 4 – Поворот фигуры против часовой стрелки (ось z)

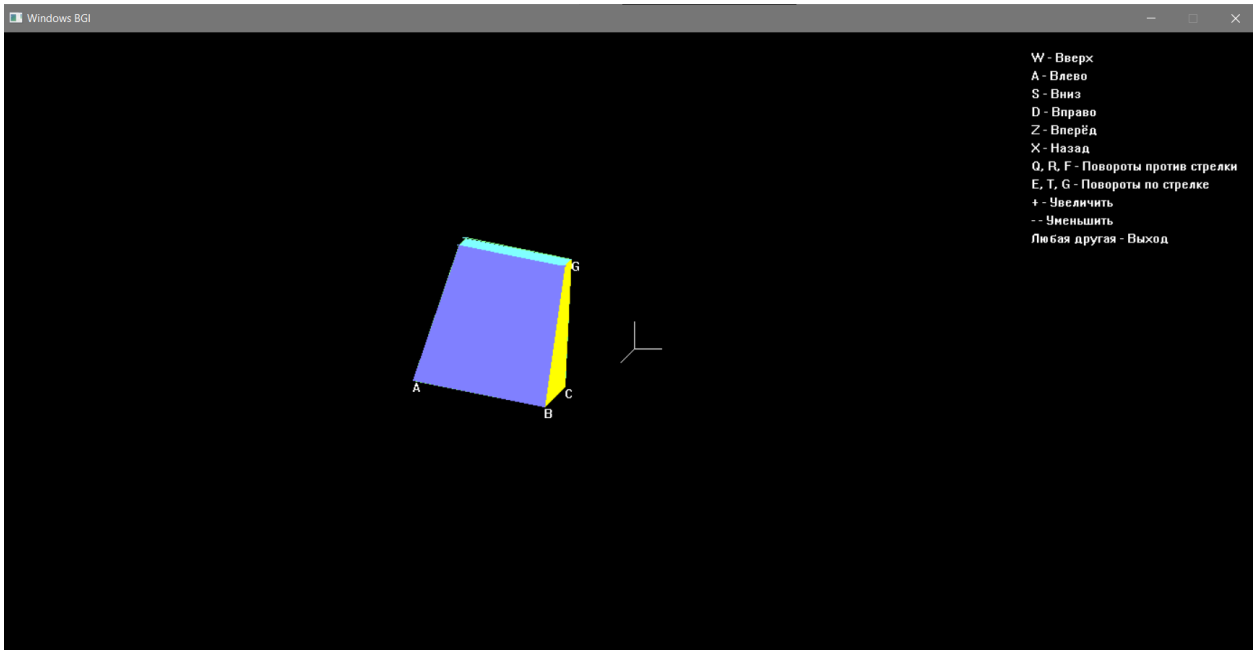

Рисунок 5 – Поворот фигуры по часовой стрелке (ось z)

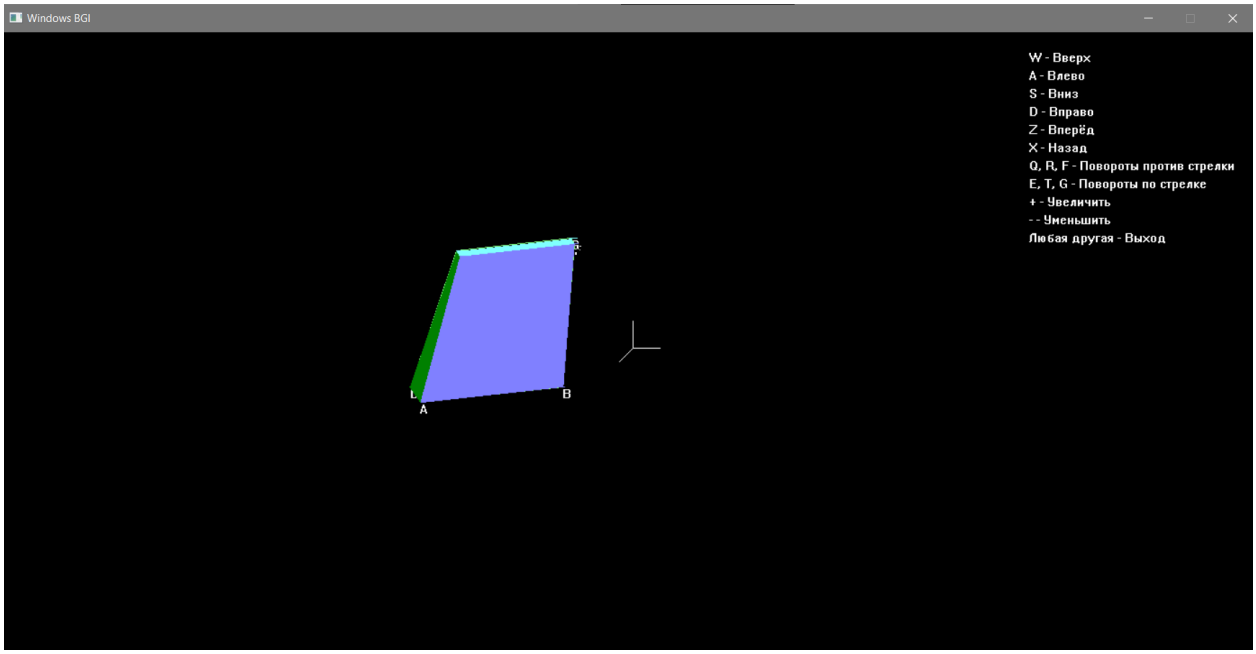

Рисунок 6 – Поворот фигуры против часовой стрелки (ось y)

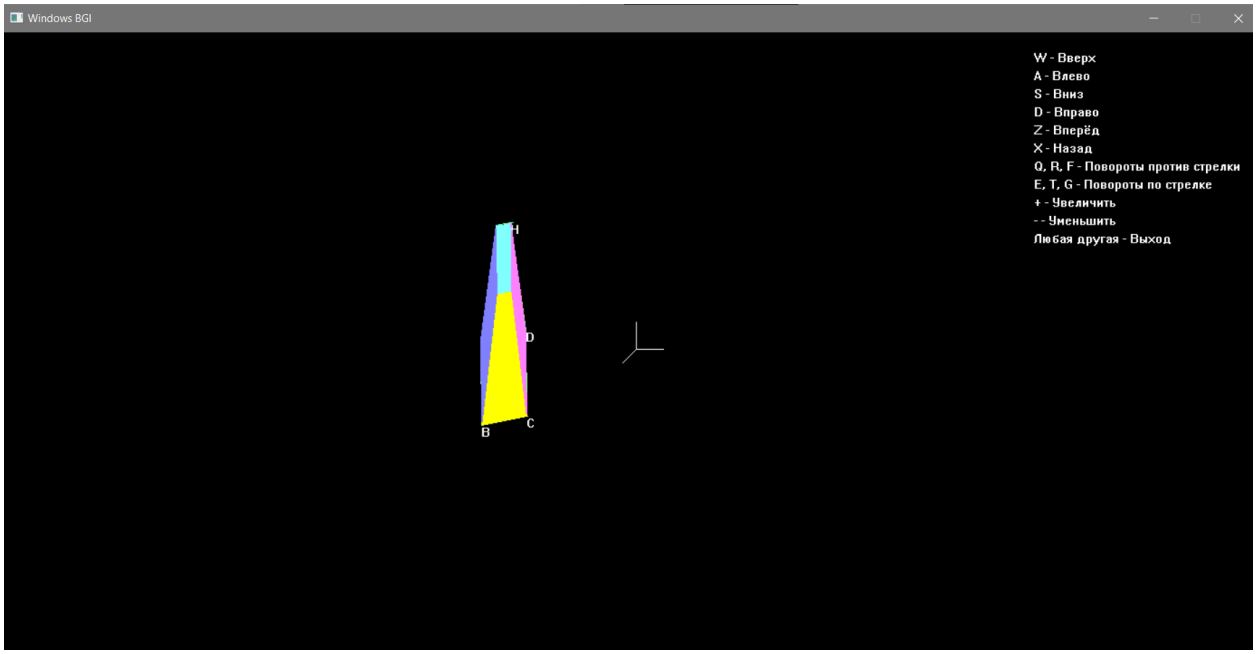

Рисунок 7 – Поворот фигуры по часовой стрелке (ось y)

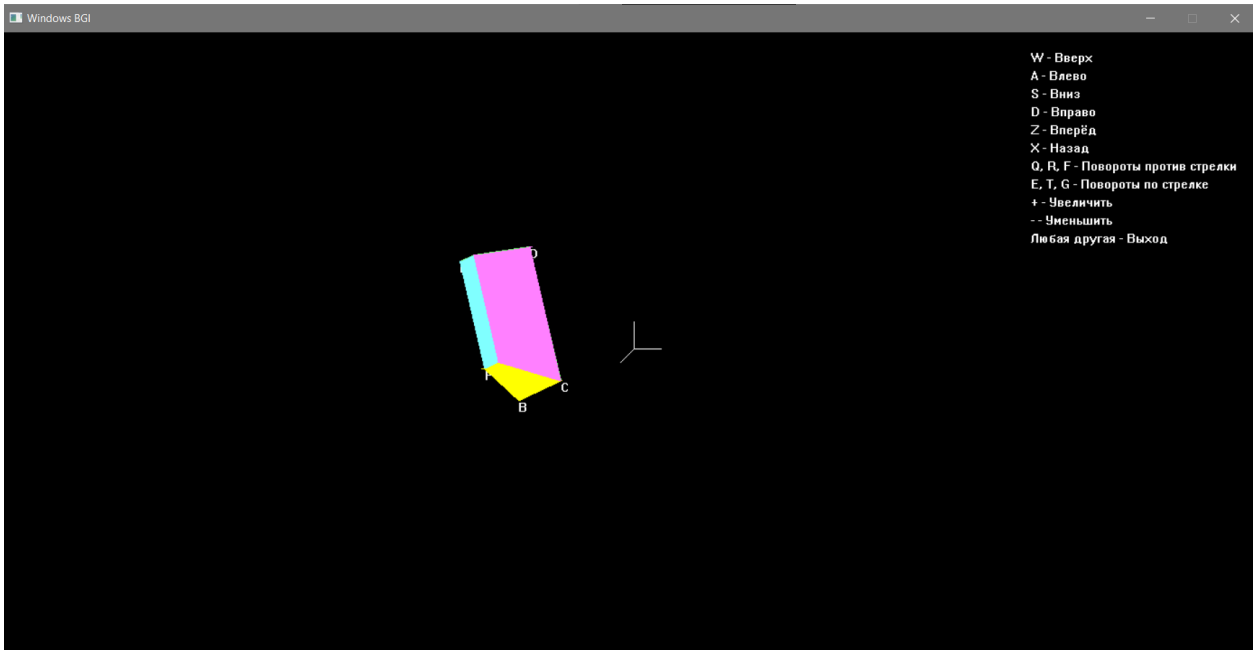

Рисунок 8 – Поворот фигуры против часовой стрелки (ось x)

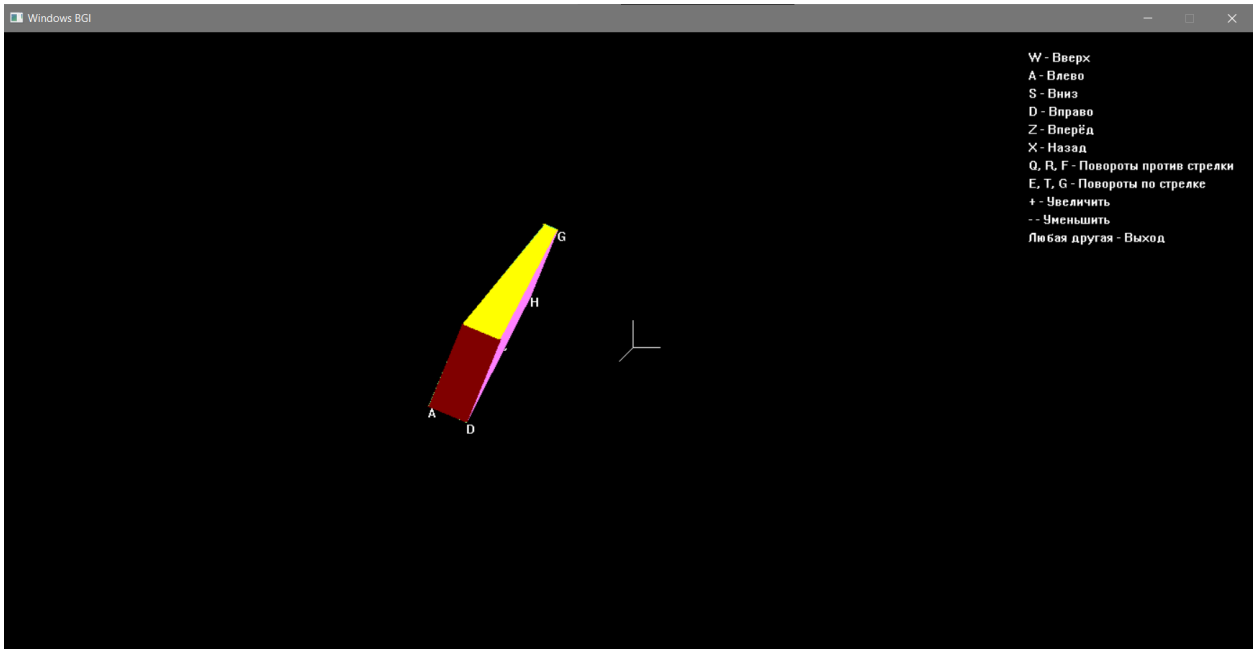

Рисунок 9 – Поворот фигуры по часовой стрелке (ось x)

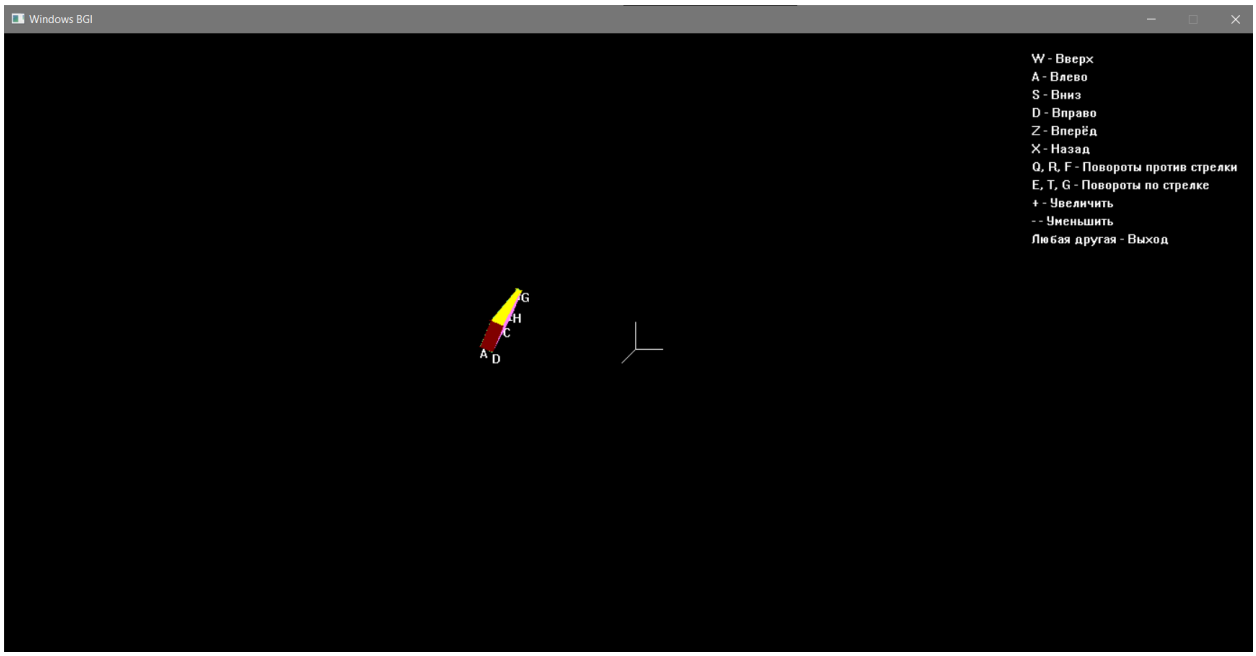

Рисунок 10 – Уменьшение фигуры

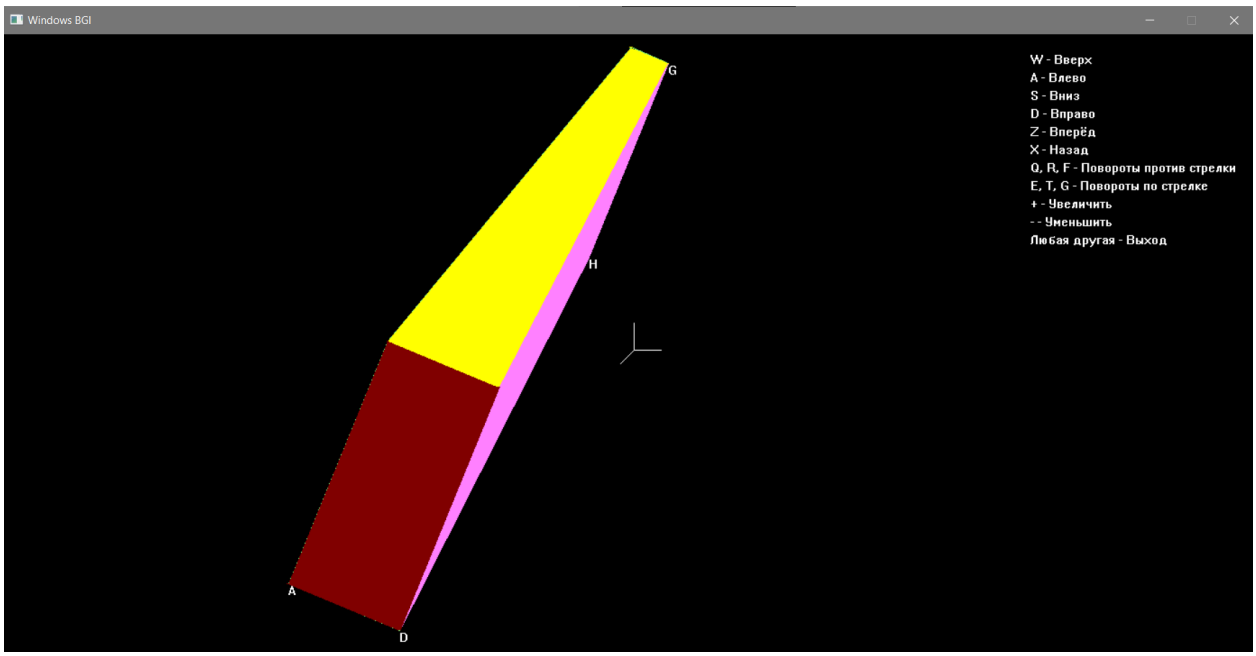

Рисунок 11 – Увеличение фигуры

7. Выводы

В ходе выполнения лабораторной работы были получены навыки работы с библиотекой для графики graphics.h, рисования, закрашивания и изменения проекции трёхмерной фигуры. Результат программы корректен.