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 |  |  |  | Ю.В. Ахромова |
|  |  |  | 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.

1. **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.

1. **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 пирамиды), состояние её видимости и цвет, которым она должна быть закрашена. В классе Piramid содержатся данные для пирамиды: четыре её точки, их имена, цвет линий. Конструктор класса задаёт начальные координаты точек, их имена и отрисовывает треугольник на экране при помощи метода класса drawPiramid. 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 треугольник 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 пирамида. 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).

1. **Тестовый пример**

Точка A (10; 100; 0).

Точка B (80; 100; 0).

Точка C (40; 100; 40).

Точка D (50; 50; 20).

1. **Листинг программы**

#include <iostream>

#include <math.h>

#include "graphics.h"

#pragma comment(lib,"graphics.lib")

#define P 3.14

#define TEXTCOL 13 // маджента

#define MAINCOL 13

#define WHITE 15

#define BLACK 0

#define GREEN 2

#define CYAN 11

#define RED 4

#define YELLOW 14

using namespace std;

// отрисовка линии попиксельно

void line\_DDA(float x1, float y1, float z1, float x2, float y2, float z2, COLORREF cColor) {

// Учёт координаты z при отрисовке в двумерном пространстве

x1 -= 0.5 \* z1;

y1 += 0.5 \* z1;

x2 -= 0.5 \* z2;

y2 += 0.5 \* z2;

// Целочисленные значения координат начала и конца отрезка, округлённые до ближайшего целого

int iX1 = roundf(x1);

int iY1 = roundf(y1);

int iX2 = roundf(x2);

int iY2 = roundf(y2);

// Длина и высота линии

int deltaX = abs(iX1 - iX2);

int deltaY = abs(iY1 - iY2);

// Считаем минимальное количество итераций, необходимое для отрисовки отрезка

// Выбирая максимум из длины и высоты линии, обеспечиваем связность линии

int length = max(deltaX, deltaY);

// особый случай, на экране закрашивается ровно один пиксель

if (length == 0) {

putpixel(iX1, iY1, cColor);

return;

}

// Вычисляем приращения на каждом шаге по осям абсцисс и ординат

double dX = (x2 - x1) / length;

double dY = (y2 - y1) / length;

// Начальные значения

double x = x1;

double y = y1;

// Основной цикл

length++;

while (length--) {

x += dX;

y += dY;

putpixel(roundf(x), roundf(y), cColor);

}

}

// 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 Piramid {

public:

Point A, B, C, D;

char name\_A[2] = "A";

char name\_B[2] = "B";

char name\_C[2] = "C";

char name\_D[2] = "D";

int col = MAINCOL;

char name\_ABC[4] = "ABC";

char name\_ADC[4] = "ADC";

char name\_ABD[4] = "ABD";

char name\_BCD[4] = "BCD";

Surface ABC = Surface(name\_ABC, RED);

Surface ADC = Surface(name\_ADC, GREEN);

Surface ABD = Surface(name\_ABD, CYAN);

Surface BCD = Surface(name\_BCD, YELLOW);

// конструктор

Piramid() {

A.x = 10; A.y = 100; A.z = 0;

A.name = name\_A;

B.x = 80; B.y = 100; B.z = 0;

B.name = name\_B;

C.x = 40; C.y = 100; C.z = 40;

C.name = name\_C;

D.x = 50; D.y = 50; D.z = 20;

D.name = name\_D;

drawPiramid();

}

// отрисовка

void drawPiramid() {

// оси

setcolor(WHITE);

line(700, 350, 730, 350);

line(700, 350, 700, 320);

line(700, 350, 685, 365);

char w[20] = "W - Up";

char a[20] = "A - Left";

char s[20] = "S - Down";

char d[20] = "D - Right";

char z[20] = "Z - Forward";

char x[20] = "X - Backward";

char qrf[30] = "Q, R, F - Counter clockwise";

char etg[20] = "E, T, G - Clockwise";

char plus[20] = "+ - Scale up";

char minus[20] = "- - Scale down";

char other[20] = "Any other - Exit";

// 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);

// нижнее основание

line\_DDA(A.x, A.y, A.z, B.x, B.y, B.z, col); // линия 1

line\_DDA(B.x, B.y, B.z, C.x, C.y, C.z, col); // линия 2

line\_DDA(C.x, C.y, C.z, A.x, A.y, A.z, col); // линия 3

// боковые грани

line\_DDA(D.x, D.y, D.z, A.x, A.y, A.z, col); // линия 4

line\_DDA(D.x, D.y, D.z, B.x, B.y, B.z, col); // линия 5

line\_DDA(D.x, D.y, D.z, C.x, C.y, C.z, col); // линия 6

// закраска граней фигуры

colouring();

}

// пермещение

void moveX(float amt) {

A.x += amt;

B.x += amt;

C.x += amt;

D.x += amt;

}

void moveY(float amt) {

A.y += amt;

B.y += amt;

C.y += amt;

D.y += amt;

}

void moveZ(float amt) {

A.z += amt;

B.z += amt;

C.z += amt;

D.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 а до центра по х

L.y = L.y - Cen.y; // по у

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) / 4;

Cen.y = (A.y + B.y + C.y + D.y) / 4;

Cen.z = (A.z + B.z + C.z + D.z) / 4;

A = rotDotZ(u, ang, Cen, A);

B = rotDotZ(u, ang, Cen, B);

C = rotDotZ(u, ang, Cen, C);

D = rotDotZ(u, ang, Cen, D);

}

// поворот одной точки вокруг y

Point rotDotY(int u, float ang, Point Cen, Point L) {

L.x = L.x - Cen.x; // расстояние от а 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) / 4;

Cen.y = (A.y + B.y + C.y + D.y) / 4;

Cen.z = (A.z + B.z + C.z + D.z) / 4;

A = rotDotY(u, ang, Cen, A);

B = rotDotY(u, ang, Cen, B);

C = rotDotY(u, ang, Cen, C);

D = rotDotY(u, ang, Cen, D);

}

// поворот одной точки вокруг x

Point rotDotX(int u, float ang, Point Cen, Point L) {

L.y = L.y - Cen.y; // расстояние от а 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) / 4;

Cen.y = (A.y + B.y + C.y + D.y) / 4;

Cen.z = (A.z + B.z + C.z + D.z) / 4;

A = rotDotX(u, ang, Cen, A);

B = rotDotX(u, ang, Cen, B);

C = rotDotX(u, ang, Cen, C);

D = rotDotX(u, ang, Cen, D);

}

// 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) / 4;

Cen.y = (A.y + B.y + C.y + D.y) / 4;

Cen.z = (A.z + B.z + C.z + D.z) / 4;

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)

) || 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);

}

}

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

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(ABC.name, Three.name) && strstr(ABC.name, Four.name))

ABC.isVisible = false; // то и она сама не видна

else ABC.isVisible = true; // иначе видна

if (strstr(ADC.name, Three.name) && strstr(ADC.name, Four.name))

ADC.isVisible = false;

else ADC.isVisible = true;

if (strstr(ABD.name, Three.name) && strstr(ABD.name, Four.name))

ABD.isVisible = false;

else ABD.isVisible = true;

if (strstr(BCD.name, Three.name) && strstr(BCD.name, Four.name))

BCD.isVisible = false;

else BCD.isVisible = true;

}

// если вторая линия ближе к наблюдателю, чем первая

else if(zOT < zTF){

cout << "line " << One.name << Two.name << " is not seen.\n";

if (strstr(ABC.name, One.name) && strstr(ABC.name, Two.name))

ABC.isVisible = false;

else ABC.isVisible = true;

if (strstr(ADC.name, One.name) && strstr(ADC.name, Two.name))

ADC.isVisible = false;

else ADC.isVisible = true;

if (strstr(ABD.name, One.name) && strstr(ABD.name, Two.name))

ABD.isVisible = false;

else ABD.isVisible = true;

if (strstr(BCD.name, One.name) && strstr(BCD.name, Two.name))

BCD.isVisible = false;

else BCD.isVisible = true;

}

}

}

return 1;

}

else return 0; // если линии не пересекаются

}

// видимость поверхностей, если линии не пересекаются

void seenS(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;

// нет пересечениий -- одна из точек в центре на xoy

// какая к центру ближе, у той и смотрим z

Point centerOfAll; // точка центра фигуры

centerOfAll.x = (p1.x + p2.x + p3.x + p4.x) / 4;

centerOfAll.y = (p1.y + p2.y + p3.y + p4.y) / 4;

int p1Prox, p2Prox, p3Prox, p4Prox; // расстояние от каждой вершины до центра фигуры на плоскости xoy

p1Prox = abs(p1.x - centerOfAll.x) + abs(p1.y - centerOfAll.y);

p2Prox = abs(p2.x - centerOfAll.x) + abs(p2.y - centerOfAll.y);

p3Prox = abs(p3.x - centerOfAll.x) + abs(p3.y - centerOfAll.y);

p4Prox = abs(p4.x - centerOfAll.x) + abs(p4.y - centerOfAll.y);

int res = min(min(p1Prox, p2Prox), min(p3Prox, p4Prox)); // расстояние от центра фигуры до ближайшей точки

Point closestPt; // точка, ближайшая к центру фигуры

float surfCen; // координата z центра фигуры

if (res == p1Prox) { // если p1Prox ближайшее расстояниа

closestPt = p1; // то точка p1 - ближайшая точка

surfCen = (p2.z + p3.z + p4.z) / 3; // находитсякоордината z центра плоскости, не содержащей эту точку

}

if (res == p2Prox) {

closestPt = p2;

surfCen = (p1.z + p3.z + p4.z) / 3;

}

if (res == p3Prox) {

closestPt = p3;

surfCen = (p1.z + p2.z + p4.z) / 3;

}

if (res == p4Prox) {

closestPt = p4;

surfCen = (p1.z + p2.z + p3.z) / 3;

}

// если ближайшая к центру точка надодится ближе к зрителю, чем центр

if (closestPt.z >= surfCen) {

cout << "\n\n\t THE Point " << closestPt.name << " is VISIBLE.\n\n";

// по умолчанию все поверхности видны

ABC.isVisible = true;

ADC.isVisible = true;

ABD.isVisible = true;

BCD.isVisible = true;

// если поверхность не содержит видимой точки

if (!strstr(ABC.name, closestPt.name))

ABC.isVisible = false; // то эту поверхность не видно

else ABC.isVisible = true; // иначе видно

if (!strstr(ADC.name, closestPt.name))

ADC.isVisible = false;

else ADC.isVisible = true;

if (!strstr(ABD.name, closestPt.name))

ABD.isVisible = false;

else ABD.isVisible = true;

if (!strstr(BCD.name, closestPt.name))

BCD.isVisible = false;

else BCD.isVisible = true;

}

else { // ближайшая к центру точка надожится дальше от зрителя, чем центр

cout << "\n\n\t THE Point " << closestPt.name << " is NOT VISIBLE AT ALL.\n\n";

// по умолчанию ни одна поверхность не видна

ABC.isVisible = false;

ADC.isVisible = false;

ABD.isVisible = false;

BCD.isVisible = false;

// если плоскость НЕ содержит НЕВИДИМУЮ точку

if (!strstr(ABC.name, closestPt.name))

ABC.isVisible = true; // то она видна

if (!strstr(ADC.name, closestPt.name))

ADC.isVisible = true;

if (!strstr(ABD.name, closestPt.name))

ABD.isVisible = true;

if (!strstr(BCD.name, closestPt.name))

BCD.isVisible = true;

}

}

// 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() {

int abcd = seenL(A, B, C, D);

int acbd = seenL(A, C, B, D);

int adbc = seenL(A, D, B, C);

// если никакие линии не пересекаются

if (abcd != 1 && acbd != 1 && adbc != 1) {

// определение видимости плоскости относительно не принадлежащей ей точки

seenS(A, B, C, D);

}

if (ABC.isVisible) {

cout << "\n\tABC is visible\n";

fill(A, B, C, ABC.colour);

}

if (ADC.isVisible) {

cout << "\n\tADC is visible\n";

fill(A, D, C, ADC.colour);

}

if (ABD.isVisible) {

cout << "\n\tABD is visible\n";

fill(A, B, D, ABD.colour);

}

if (BCD.isVisible) {

cout << "\n\tBCD is visible\n";

fill(B, C, D, BCD.colour);

}

}

};

int main() {

initwindow(1400, 700); // создаём консольное окно 1400 на 700

Piramid Tri; // создание фигуры

// управление

int i = 1; // условие выхода

while (i) {

switch (getch()) {

case 'w':

case 'W':

case 'ц':

case 'Ц':

cout << 'w' << endl;

Tri.moveY(-10); // вверх

break;

case 'a':

case 'A':

case 'ф':

case 'Ф':

cout << 'a' << endl;

Tri.moveX(-10); // влево

break;

case 's':

case 'S':

case 'ы':

case 'Ы':

cout << 's' << endl;

Tri.moveY(10); // вниз

break;

case 'd':

case 'D':

case 'в':

case 'В':

cout << 'd' << endl;

Tri.moveX(10); // вправо

break;

case 'x':

case 'X':

case 'ч':

case 'Ч':

cout << 'x' << endl;

Tri.moveZ(-10); // назад

Tri.scale(0.9);

break;

case 'z':

case 'Z':

case 'я':

case 'Я':

cout << 'z' << endl;

Tri.moveZ(10); // вперёд

Tri.scale(1.1);

break;

// вокруг z

case 'q':

case 'Q':

case 'й':

case 'Й':

cout << 'q' << endl;

Tri.rotateZ(1); // против часовой

break;

case 'e':

case 'E':

case 'у':

case 'У':

cout << 'e' << endl;

Tri.rotateZ(-1); // по часовой

break;

// вокруг y

case 'r':

case 'R':

case 'к':

case 'К':

cout << 'r' << endl;

Tri.rotateY(1); // против часовой

break;

case 't':

case 'T':

case 'е':

case 'Е':

cout << 't' << endl;

Tri.rotateY(-1); // по часовой

break;

// вокруг x

case 'f':

case 'F':

case 'а':

case 'А':

cout << 'f' << endl;

Tri.rotateX(1); // против часовой

break;

case 'g':

case 'G':

case 'п':

case 'П':

cout << 'g' << endl;

Tri.rotateX(-1); // по часовой

break;

case '=':

case '+':

cout << '+' << endl;

Tri.scale(1.5); // увеличение

break;

case '-':

case '\_':

cout << '-' << endl;

Tri.scale(0.5); // уменьшение

break;

default:

cout << "default -> exit" << endl;

i = 0;

break;

}

cleardevice(); // отичстка экрана

Tri.drawPiramid(); // перерисовка фигуры

}

getch(); // чтение одного символа с клавиатуры

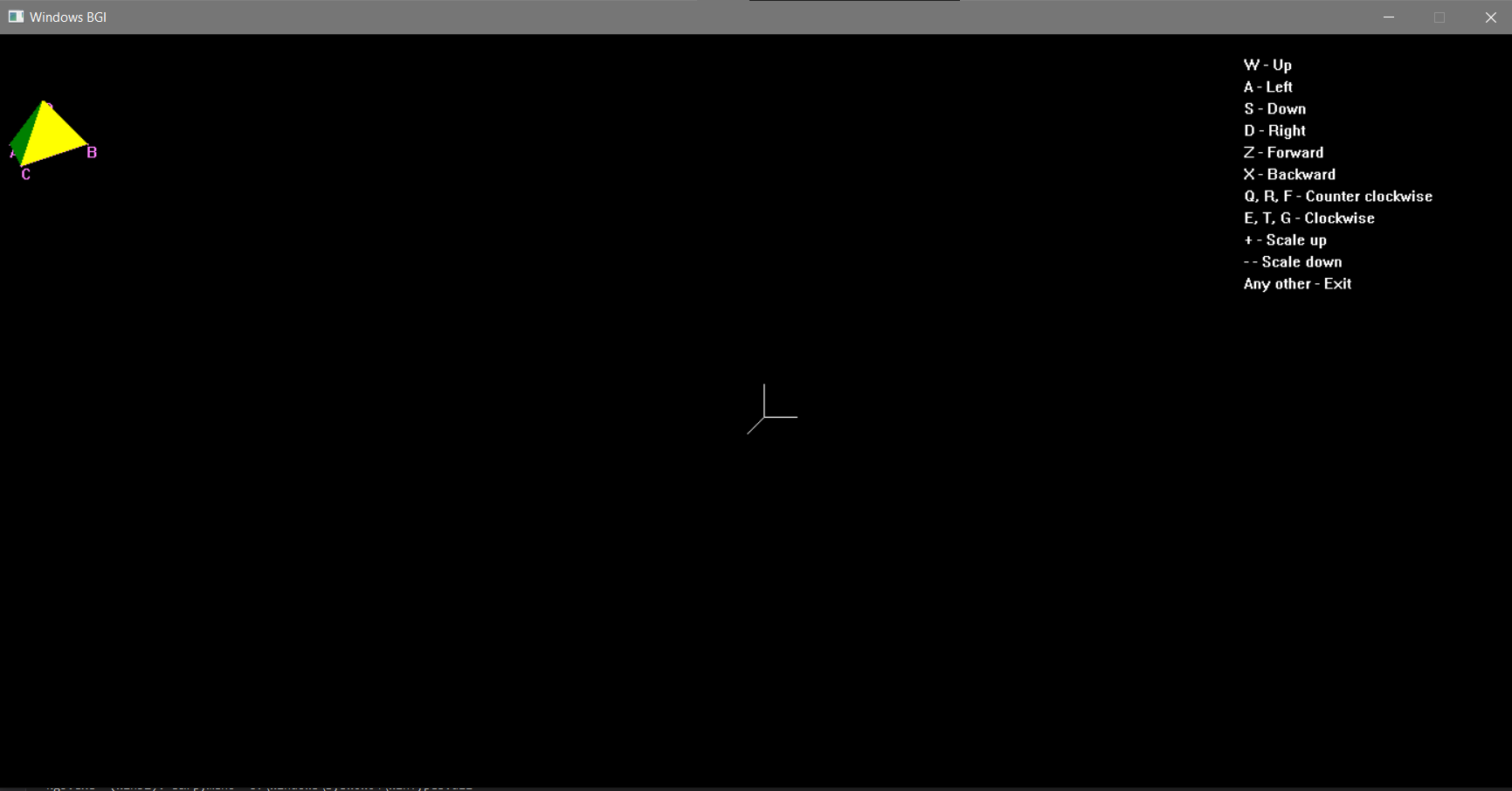
closegraph(); // освобождает всю память, выделенную под графическую систему, затем восстанавливает экран в режим, который был до вызова initwindow

return 0;

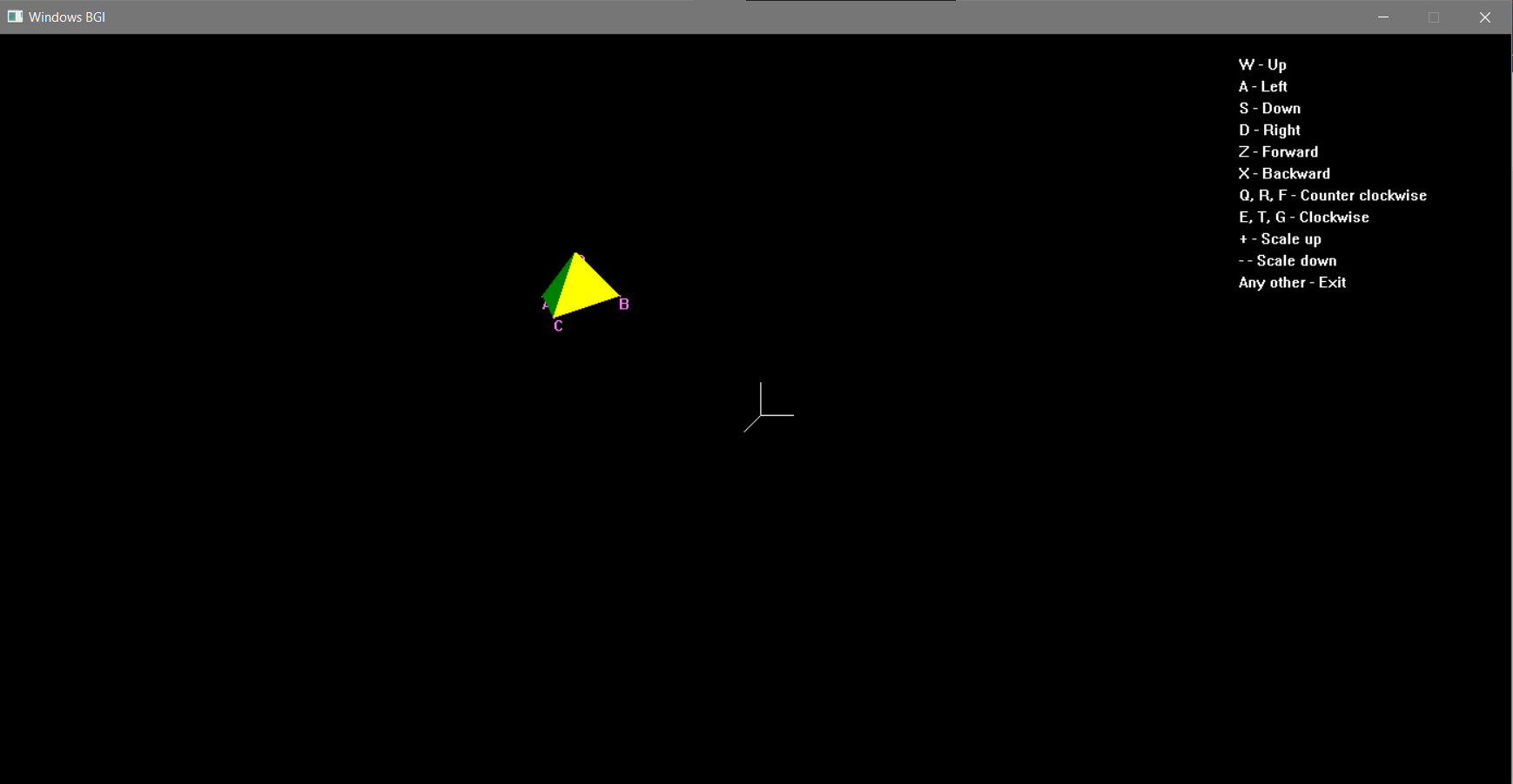
}

1. **Результаты работы программы**

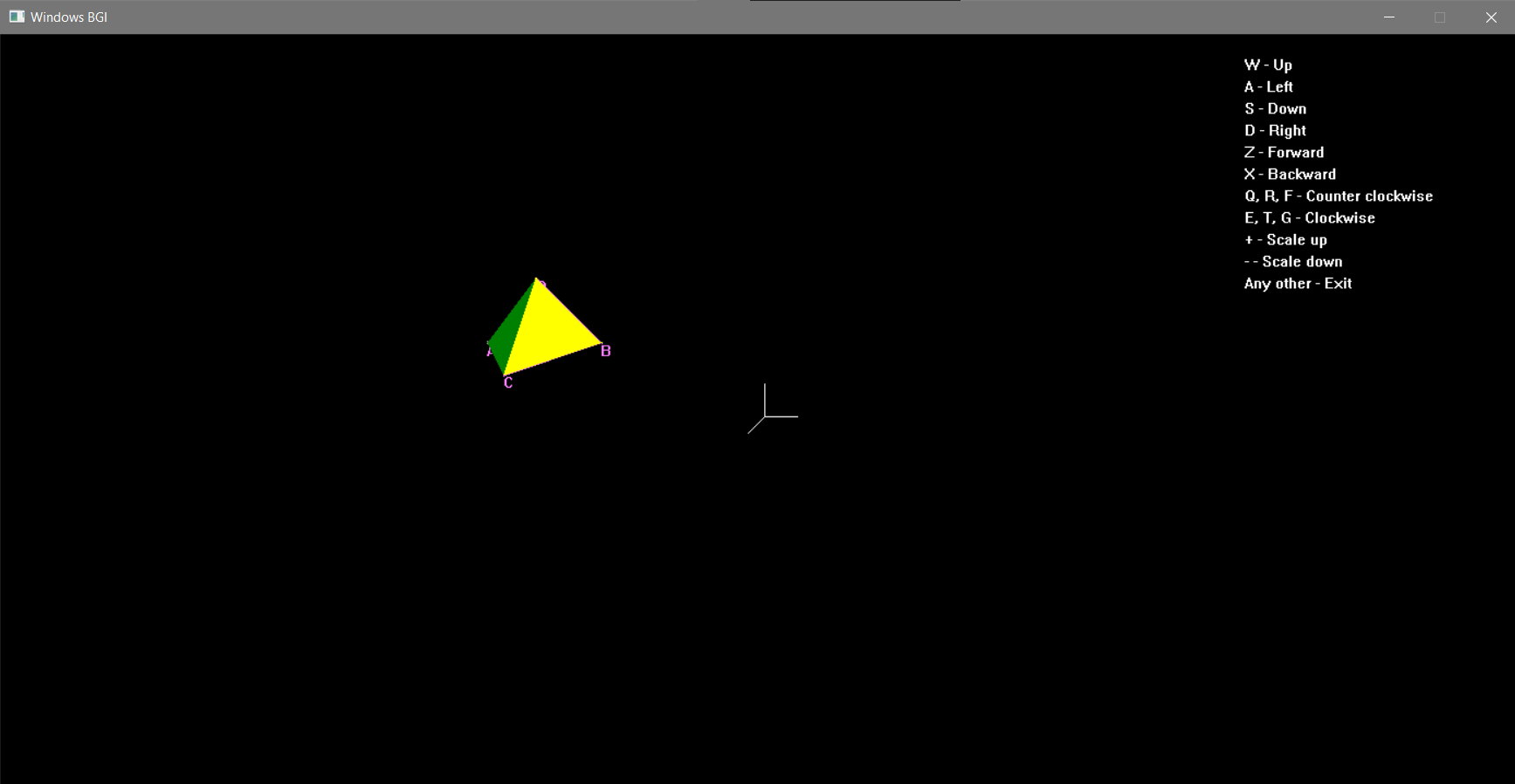
Результаты работы программы представлены на рисунках 1-11.



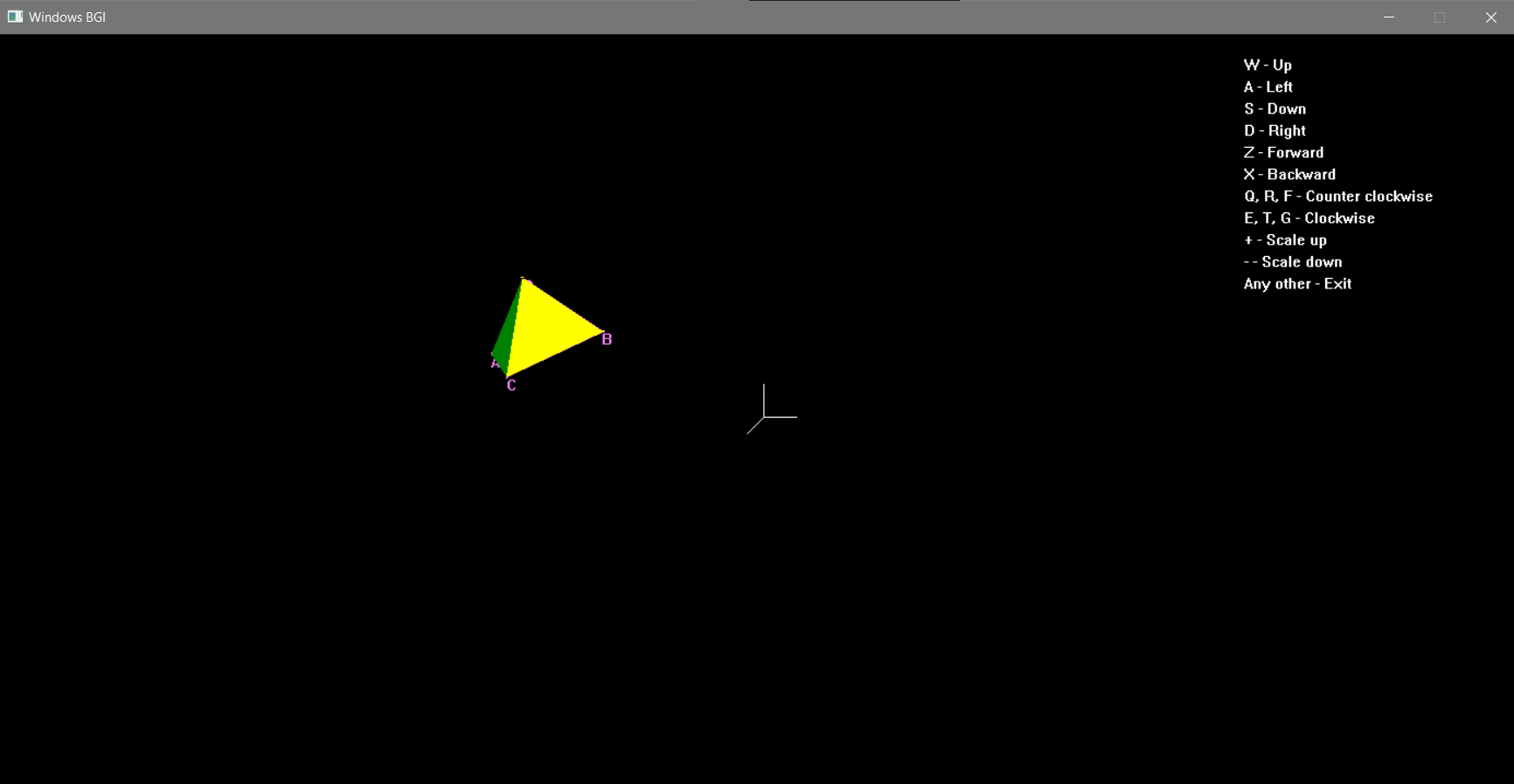
*Рисунок 1 – Отрисовка фигуры*



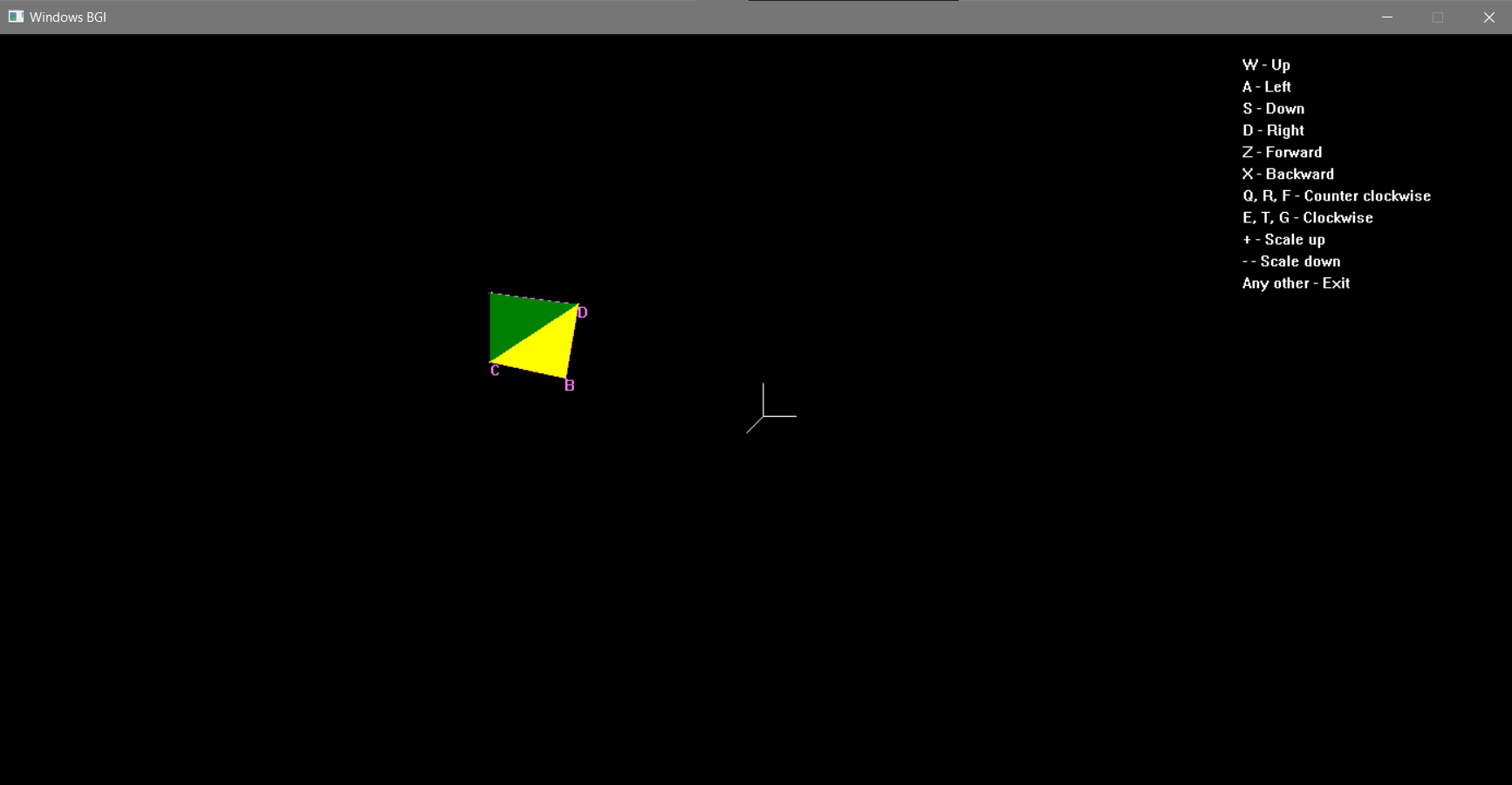
*Рисунок 2 – Перемещение фигуры вдоль осей x и y*



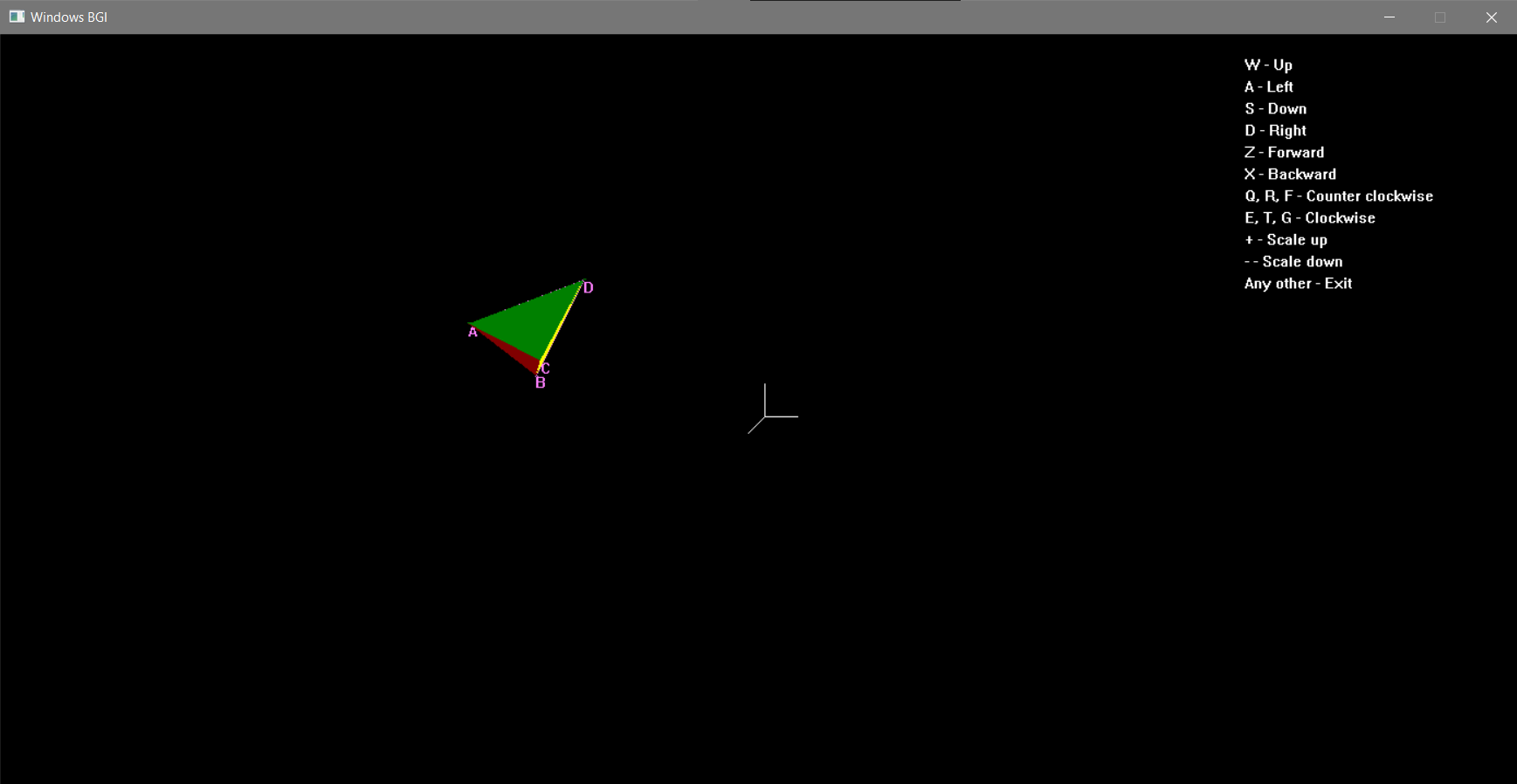
*Рисунок 3 – Перемещение фигуры вдоль оси z*

**

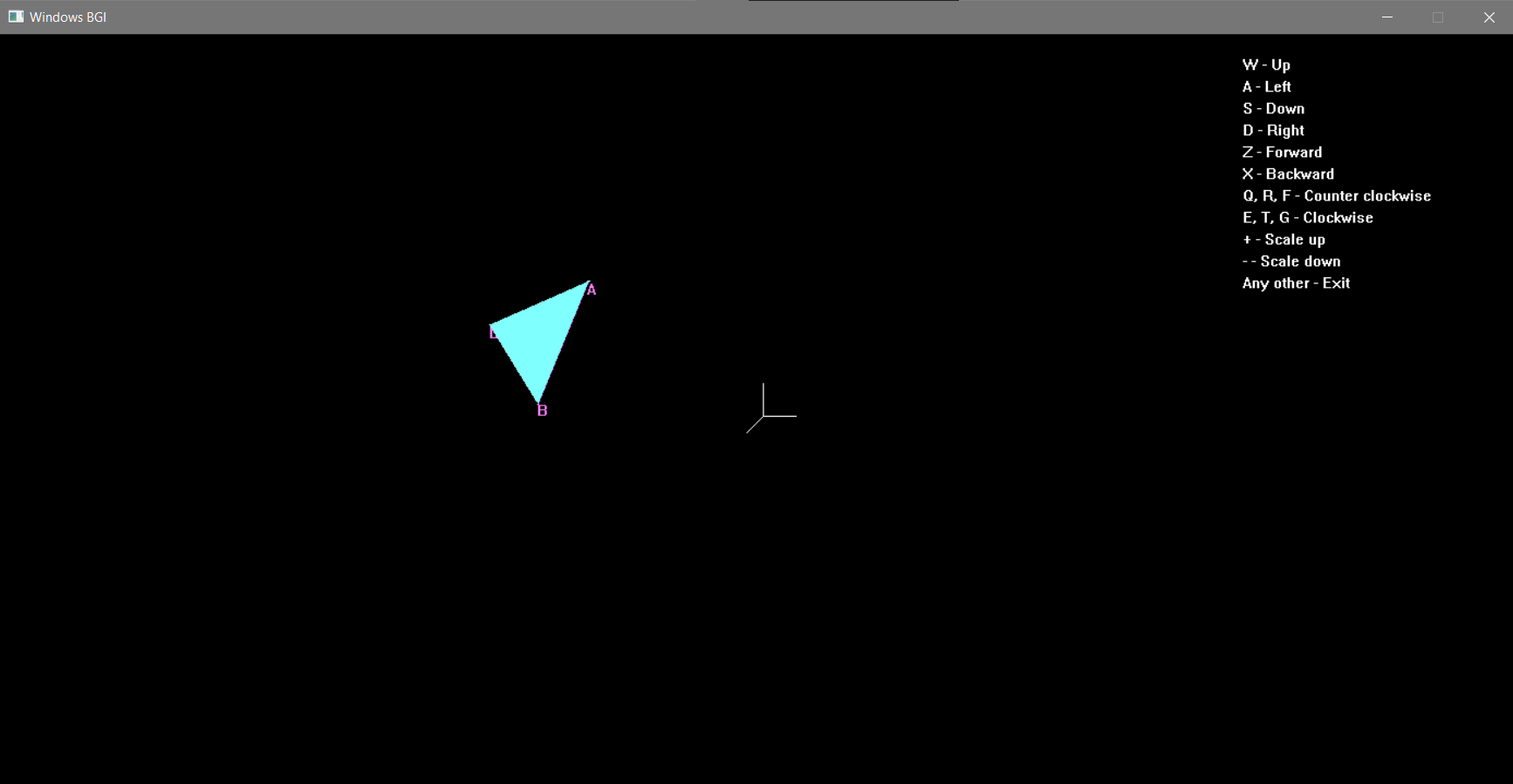
*Рисунок 4 – Поворот фигуры против часовой стрелки (ось z)*



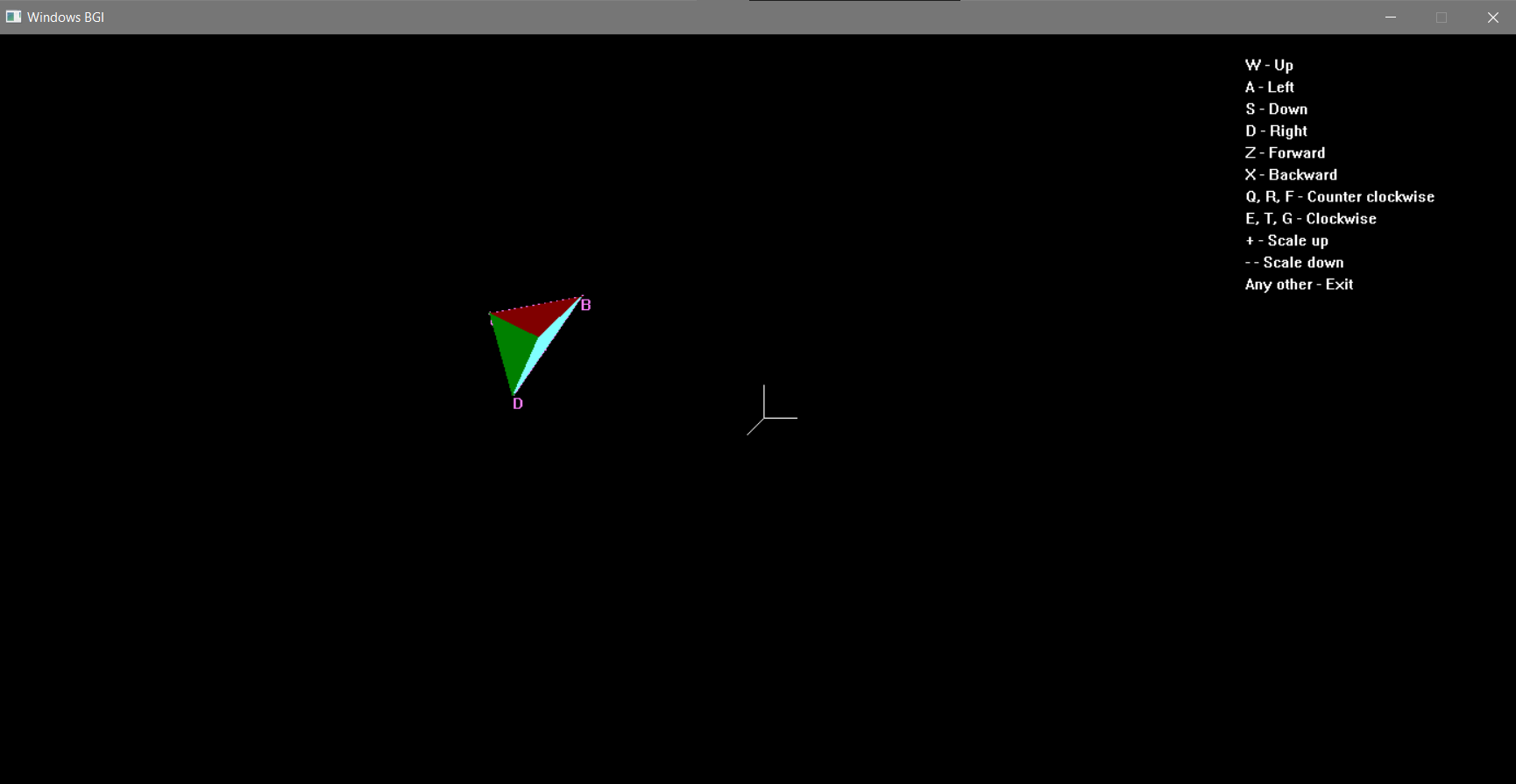
*Рисунок 5 – Поворот фигуры по часовой стрелке (ось z)*

**

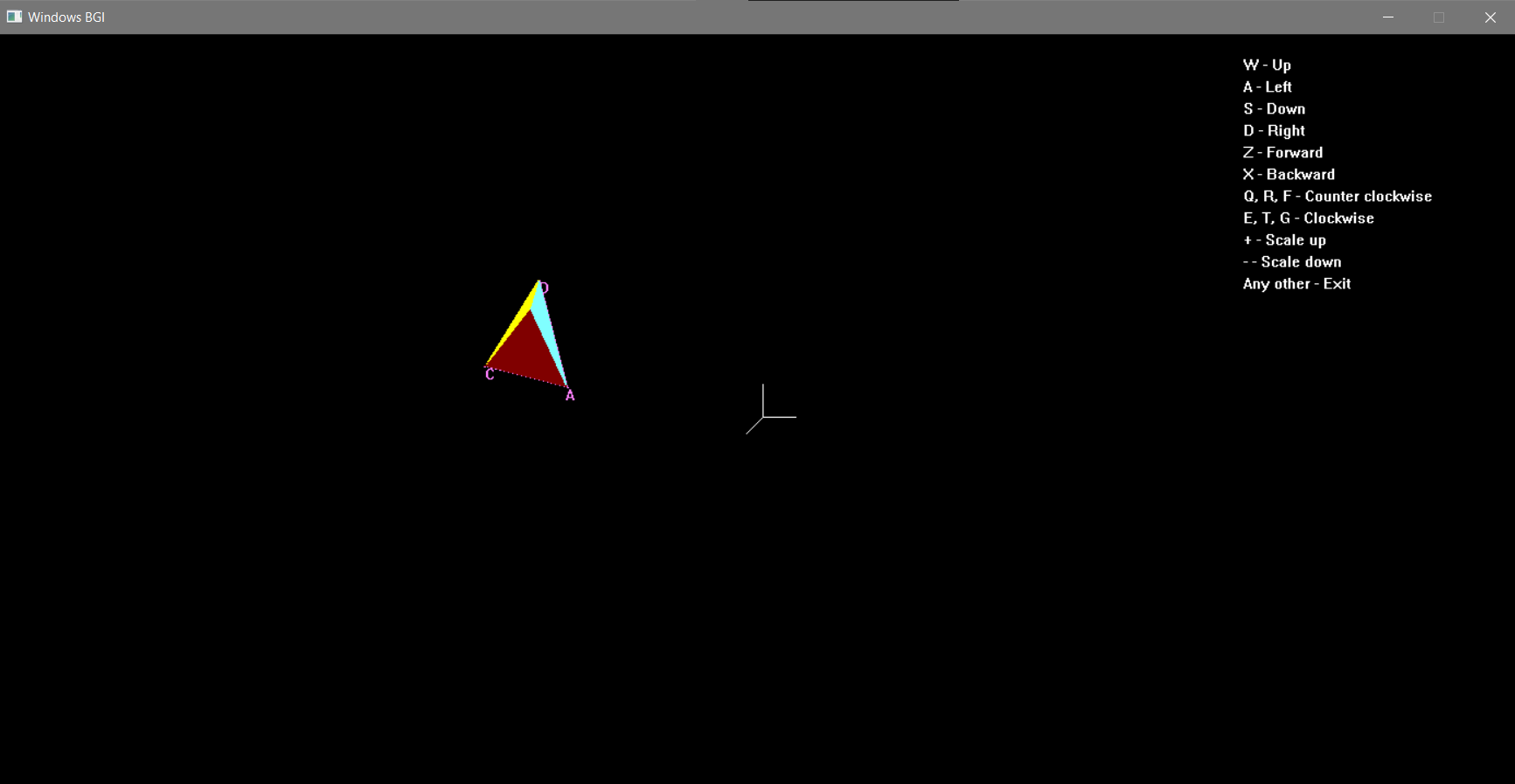
*Рисунок 6 – Поворот фигуры против часовой стрелки (ось y)*



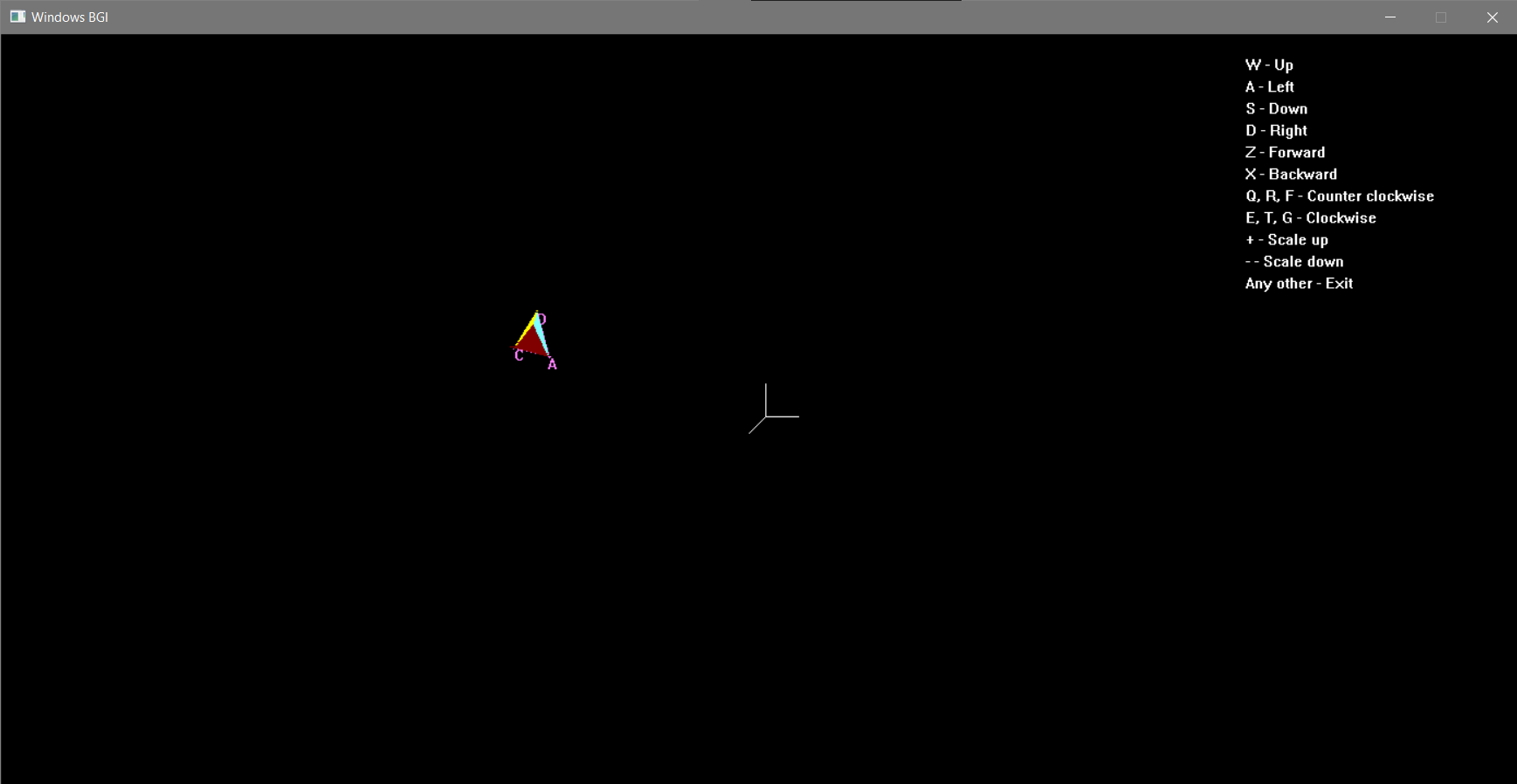
*Рисунок 7 – Поворот фигуры по часовой стрелке (ось y)*

**

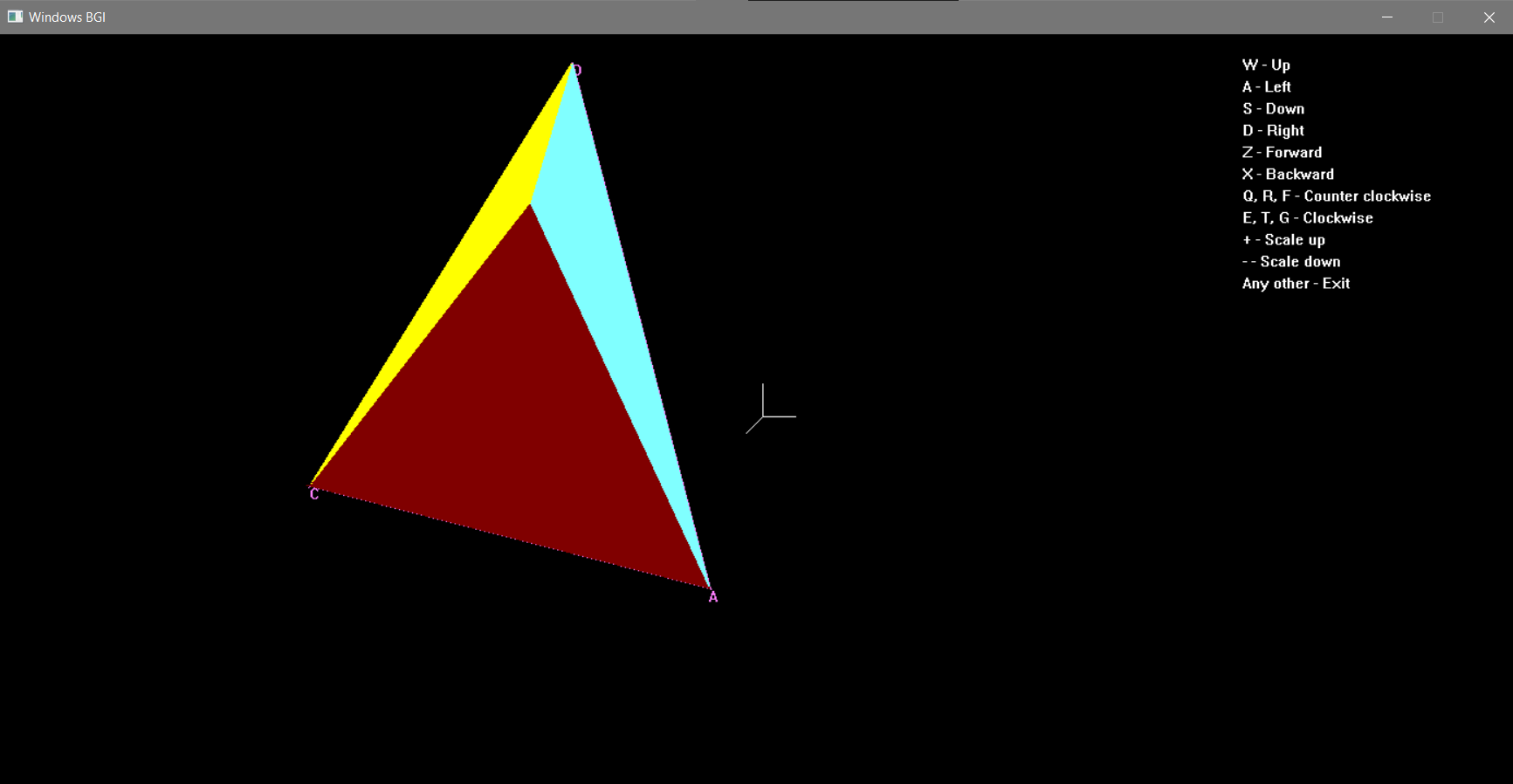
*Рисунок 8 – Поворот фигуры против часовой стрелки (ось x)*



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



*Рисунок 10 – Уменьшение фигуры*



*Рисунок 11 – Увеличение фигуры*

1. **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.