

Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное
учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Институт естественных и точных наук
Кафедра прикладной математики и программирования

ОТЧЕТ
о выполнении лабораторной работы № 8 по дисциплине
«Математические основы компьютерной графики»

Автор работы,
студент группы ЕТ-212
_____ Никитин М.С.
«____» _____ 2022 г.

Руководитель работы,
старший преподаватель
_____ Шелудько А.С.
«____» _____ 2022 г.

Челябинск 2022

1 ЗАДАНИЕ

Написать программу для выполнения аффинных преобразований многогранника. Использовать векторную полигональную модель, составленную в лабораторной работе 7. Предварительно определить структуру данных (класс) и разработать соответствующие подпрограммы (методы). Число вершин, число граней, координаты вершин и списки вершин, образующих грани, считать из файла. Интерфейс программы должен содержать следующие элементы управления:

- перемещение многогранника;
- поворот многогранника (относительно центра фигуры);
- сохранение результата в файл;
- выход из программы.

2 МАТЕМАТИЧЕСКАЯ МОДЕЛЬ

Описание класса:

```
struct Point
{
    double x = 0;
    double y = 0;
};

class Figura
{
    int count_vertices;
    Point *plg_points;
    Point plg_center;
    Point center;
    double plg_angle = 0;
    double plg_factor = 1;

public:
    Figura();
    ~Figura() = default;
    void calc_center();
    void move(double x, double y);
    void rotate(double angle);
    void scale(double factor);
    void draw();
};


```

Описание методов класса:

```
const double PI = acos(0.0)*2.0;
// конструктор
Figura::Figura()
{
    ifstream file;
    file.open("info_for_figure.txt");
    file >> count_vertices;

    plg_points = new Point[count_vertices];

    for (int i=0; i<count_vertices; i++)
        file >> plg_points[i].x >> plg_points[i].y;
    calc_center();
    draw();
}
// Рассчет центра фигуры
void Figura::calc_center(){
    for(int i=0; i<count_vertices; i++){
        center.x += plg_points[i].x;
        center.y += plg_points[i].y;
    }
    center.x /= count_vertices;
```

```

        center.y /= count_vertices;
    }
    // Изменение угра фигуры
    void Figura::rotate(double angle)
    {
        plg_angle += angle*(PI/180.0);
        draw();
    }
    // Смещение фигуры
    void Figura::move(double x, double y)
    {
        plg_center.x += x;
        plg_center.y += y;
        draw();
    }
    // Изменение масштаба
    void Figura::scale(double factor)
    {
        plg_factor *= factor;
        draw();
    }
    // Прорисовка
    void Figura::draw()
    {
        Point points[count_vertices];
        int point4draw[2*count_vertices];
        putimage(0, 0, loadBMP("BrutalMagica.bmp"), COPY_PUT);
        for (int i = 0; i < count_vertices; i++)
        {
            points[i] = plg_points[i];

            // Перемещение
            points[i].x += plg_center.x;
            points[i].y += plg_center.y;
            calc_center();

            // Поворот
            points[i].x =
                center.x+(-center.x+plg_points[i].x)*cos(plg_angle)+(
                center.y-plg_points[i].y)*sin(plg_angle);
            points[i].y =
                center.y+(-center.x+plg_points[i].x)*sin(plg_angle)+(
                -center.y+plg_points[i].y)*cos(plg_angle);

            // Масштабирование
            points[i].x *= plg_factor;
            points[i].y *= plg_factor;

            // Преобразование из декартовых в экранные
            points[i].x = points[i].x + WIDTH/2.0;
            points[i].y = HIEGHT/2.0 - points[i].y;
        }
    }

```

```
// Подготовка массива для построения многоугольника
for (int i = 0; i < 2*count_vertices; i+=2){
    printf("(%.2lf ; %.2lf) - ", points[i/2].x, points[i/2].y);
    point4drow[i] = points[i/2].x;
    point4drow[i+1] = points[i/2].y;
}
cout << endl;

setfillstyle(SOLID_FILL, BLACK);
fillpoly(count_vertices, point4drow);

setcolor(YELLOW);
setfillstyle(SOLID_FILL, YELLOW);

}
```

3 ТЕКСТ ПРОГРАММЫ

Файл main.cpp

```
#include "graphics.h"
#include "control.h"
#include "figura.hpp"

int main()
{
    initwindow(WIDTH, HIEGHT, "BrutalMagica");
    putimage(0, 0, loadBMP("BrutalMagica.bmp"), COPY_PUT);
    create_control_S(CCWROT, FRAMESIDE, FRAMELOWER);
    create_control_S(CWROT, INDENT+3*FRAMESIDE,
                     FRAMELOWER);
    create_control(MOVE, SIZEBUTTONB - 2*INDENT,
                  FRAMELOWER);
    create_control(SCALE, 2*SIZEBUTTONB,
                  FRAMELOWER);
    create_control_S(SAVE, WIDTH - SIZEBUTTONB+ FRAMESIDE,
                     FRAMELOWER);
    create_control_S(EXIT, WIDTH-3*FRAMESIDE+INDENT,
                     FRAMELOWER);
    int type = TNONE;

    Figura figura;

    while (true)
    {
        while (mousebuttons() != 1) {}
        switch (select_control()) {
        case NONE:
            break;
        case CCWROT:
            figura.rotate(-1);
            type = TNONE;
            break;
        case CWROT:
            figura.rotate(1);
            type = TNONE;
            break;
        case SCALE:
            type = TSCALE;
            break;
        case MOVE:
            type = TMOVE;
            break;
        case SAVE:
            save();
            type = TNONE;
            break;
        }
    }
}
```

```

        case EXIT:
            return 0;
    }
    switch (type) {
        case TMOVE:
            switch (getch(kbhit())) {
                case TNONE:
                    break;
                case KEY_UP:
                    figura.move(0, 1);
                    break;
                case KEY_DOWN:
                    figura.move(0, -1);
                    break;
                case KEY_LEFT:
                    figura.move(-1, 0);
                    break;
                case KEY_RIGHT:
                    figura.move(1, 0);
                    break;
            }
            break;
        case TSCALE:
            switch (getch(kbhit())) {
                case KEY_LEFT:
                    figura.scale(1.025);
                    break;
                case KEY_RIGHT:
                    figura.scale(0.975);
                    break;
            }
        case TNONE:
            break;
    }
}
}

```

Файл figura.hpp

```
#ifndef Figura_H
#define Figura_H

struct Point
{
    double x = 0;
    double y = 0;
};

class Figura
{
    int count_vertices;
    Point *plg_points;
    Point plg_center;
    Point center;
    double plg_angle = 0;
    double plg_factor = 1;

public:
    Figura();
    ~Figura() = default;
    void calc_center();
    void move(double x, double y);
    void rotate(double angle);
    void scale(double factor);
    void draw();
};

#endif
```

Файл figura.cpp

```
#include "graphics.h"
#include "figura.hpp"
#include "control.h"
#include <cmath>
#include <control.h>
#include <fstream>

#include <iostream>

const double PI = acos(0.0)*2.0;
using namespace std;
Figura::Figura()
{
    ifstream file;
    file.open("info_for_figure.txt");
    file >> count_vertices;
```

```

plg_points = new Point[count_vertices];

for (int i=0; i<count_vertices; i++) {
    file >> plg_points[i].x >> plg_points[i].y;

    printf("(%.2lf ; %.2lf) - ", plg_points[i].x, plg_points[i].y);
}
calc_center();
draw();
}

void Figura::calc_center(){
    for(int i=0; i<count_vertices; i++){
        center.x += plg_points[i].x;
        center.y += plg_points[i].y;
    }
    center.x /= count_vertices;
    center.y /= count_vertices;
}

void Figura::rotate(double angle)
{
    plg_angle += angle*(PI/180.0);
    draw();
}

void Figura::move(double x, double y)
{
    plg_center.x += x;
    plg_center.y += y;
    draw();
}

void Figura::scale(double factor)
{
    plg_factor *= factor;
    draw();
}

void Figura::draw()
{
    Point points[count_vertices];
    int point4draw[2*count_vertices];
    putimage(0, 0, loadBMP("BrutalMagica.bmp"), COPY_PUT);
    for (int i = 0; i < count_vertices; i++)
    {
        points[i] = plg_points[i];

        //      points[i].x += plg_center.x;
        points[i].y += plg_center.y;
        calc_center();
    }
}

```

```

//      //~ points[i].x = cos(plg_angle)*plg_points[i].x - sin(plg_angle)*plg_points[i].y;
//~ points[i].y = sin(plg_angle)*plg_points[i].x + cos(plg_angle)*plg_points[i].y;
points[i].x = center.x+(-center.x+plg_points[i].x)*cos(plg_angle);
points[i].y = center.y+(-center.x+plg_points[i].x)*sin(plg_angle);

//~ points[i].x *= plg_factor;
//~ points[i].y *= plg_factor;

//      points[i].x = points[i].x + WIDTH/2.0;
//~ points[i].y = HIEGHT/2.0 - points[i].y;
}

//      for (int i = 0; i < 2*count_vertices; i+=2){
printf("(%.2lf ; %.2lf) - ", points[i/2].x, points[i/2].y);
point4drow[i] = points[i/2].x;
point4drow[i+1] = points[i/2].y;
}
cout << endl;

setfillstyle(SOLID_FILL, BLACK);
fillpoly(count_vertices, point4drow);

setcolor(YELLOW);
setfillstyle(SOLID_FILL, YELLOW);

}

```

Файл control.h

```

#ifndef CONTROL_H
#define CONTROL_H

#define WIDTH 1000
#define HIEGHT 648
#define SIZEBUTTONB 240
#define SIZEBUTTONS 70
#define INDENT 10

#define FRAMESIDE 40
#define FRAMELOWER 540


enum control_values { NONE = -1, EXIT, SAVE,
                     CCWROT, CWROT, MOVE, SCALE, N_CONTROLS};
enum type_function {TNONE, TMOVE, TSCALE};

struct Control
{

```

```

    int left;
    int top;
    int right;
    int bottom;
};

void create_control(int, int, int);
void create_control_S(int, int, int);
int select_control();
void save();

#endif

```

Файл control.cpp

```

#include "graphics.h"
#include "control.h"

Control controls[N_CONTROLS];

void create_control(int i, int left, int top)
{
    controls[i].left = left;
    controls[i].top = top;
    controls[i].right = left + SIZEBUTTONB;
    controls[i].bottom = top + SIZEBUTTONS;
}

void create_control_S(int i, int left, int top)
{
    controls[i].left = left;
    controls[i].top = top;
    controls[i].right = left + SIZEBUTTONS;
    controls[i].bottom = top + SIZEBUTTONS;
}
int select_control()
{
    int x, y;

    x = mousex();
    y = mousey();

    for (int i = 0; i < N_CONTROLS; i++)
    {
        if (x > controls[i].left && x < controls[i].right &&
            y > controls[i].top && y < controls[i].bottom)
        {
            return i;
        }
    }
}

```

```
    return NONE;
}

void save()
{
    int width, height;
    IMAGE *output;

    width = getmaxx() + 1;
    height = getmaxy() + 1;
    output = createimage(width, height);

    getimage(0, 0, width - 1, height - 1, output);
    saveBMP("output.bmp", output);
    freeimage(output);
}
```

4 РЕЗУЛЬТАТ РАБОТЫ

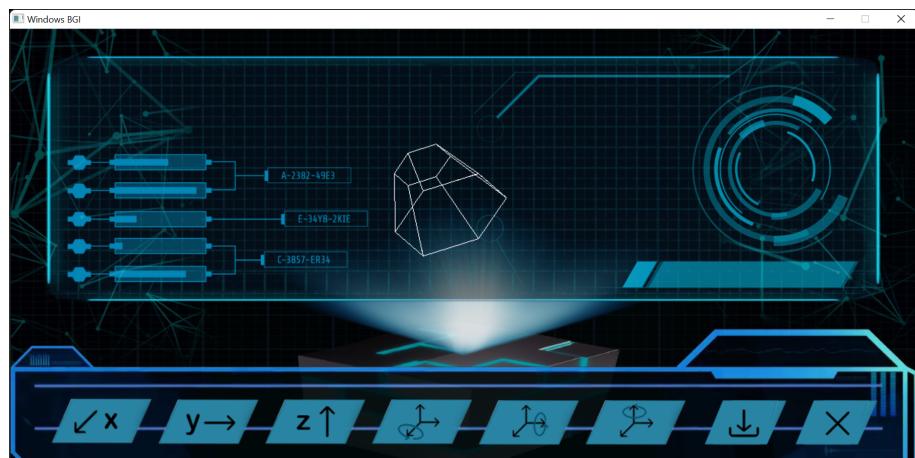


Рисунок 4.1 – Начальное положение

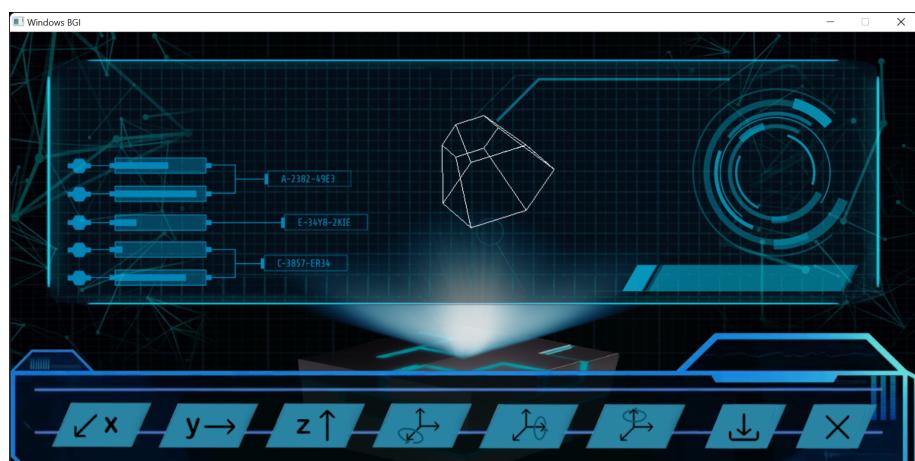


Рисунок 4.2 – Перемещение

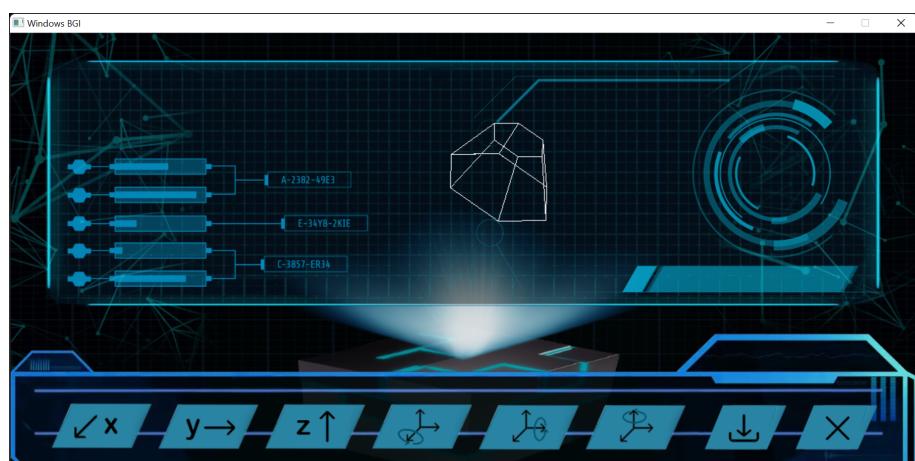


Рисунок 4.3 – Поворот вокруг оси Ох

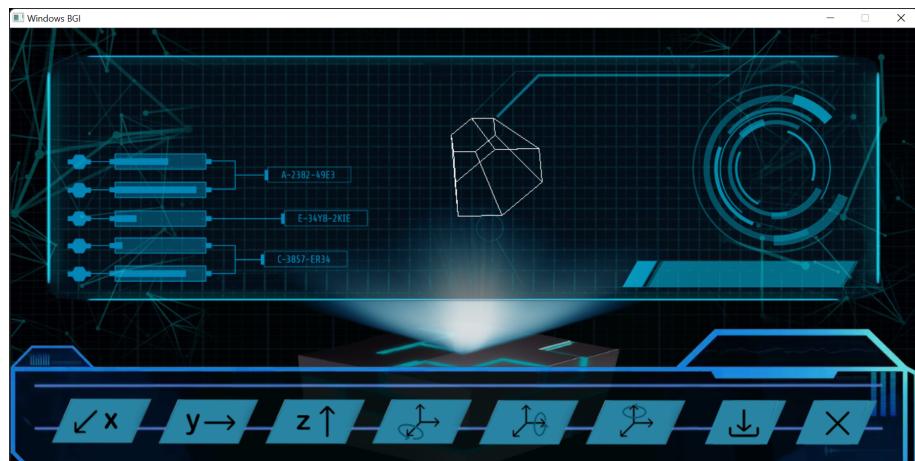


Рисунок 4.4 – Поворот вокруг оси Оу

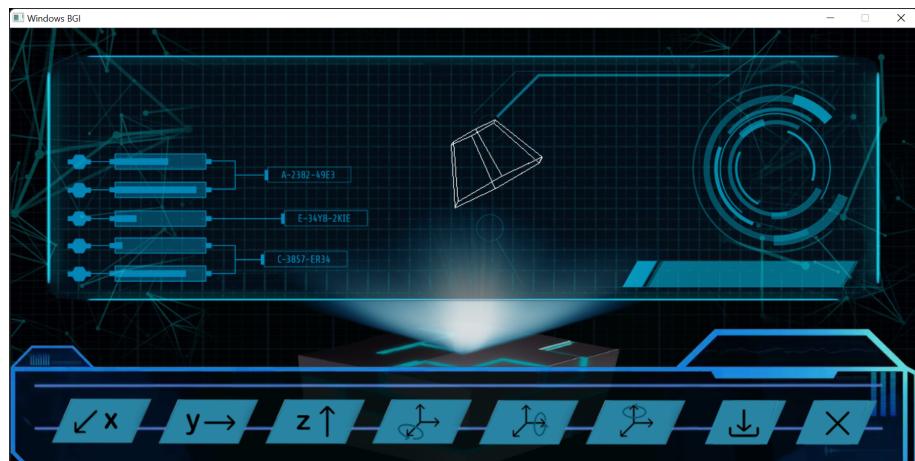


Рисунок 4.5 – Поворот вокруг оси Оз