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
_____ Шафикова М.А.
«____» _____ 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 многогранника);
- сохранение результата в файл;
- выход из программы.

2 ОПИСАНИЕ КЛАССА POLYHEDRON

Класс фигуры (многогранника)

```
#ifndef TASK_H
#define TASK_H

#include <iostream>
#include <vector>
#include <graphics.h>
#include <math.h>
#include <fstream>
#include "control.h"

using namespace std;

class Polyhedron{
private:
   int I[3] = {1, 0, 0};
   int J[3] = {0, 1, 0};
   int K[3] = {0, 0, 1};
   int p[20];
   int surface_num, vertex_num;
   int d[3] = {0, 0, 0};
   vector<vector<double>> vertexs;
   vector<vector<int>> surfaces;
public:
   Polyhedron();
   void rotate(int);
   void move(int, int);
   void draw(int);
};

#endif
```

Файл main.cpp

```
#include "task.h"
#include "control.h"

int main(){
   initwindow(WIDTH, HEIGHT);
   Polyhedron figure = Polyhedron();
   set_bg("bg.jpg");
   create_control(X_ROTATE, 0, 0);
   create_control(Y_ROTATE, 0, 75);
   create_control(Z_ROTATE, 0, 150);
   create_control(MOVE_UP, 0, 225);
   create_control(MOVE_RIGHT, 0, 300);
   create_control(MOVE_LEFT, 0, 375);
   create_control(MOVE_DOWN, 0, 450);
   create_control(EXIT, 0, 645);
   create_control(SAVE, 0, 730);
   figure.draw(0);
   while(1){
      while(mousebuttons() != 1);
      switch(select_control()){
         case X_ROTATE:
            figure.rotate(0);
            set_bg("bg.jpg");
            figure.draw(0);
            break;
         case Y_ROTATE:
            figure.rotate(1);
            set_bg("bg.jpg");
            figure.draw(0);
            break;
         case Z_ROTATE:
            figure.rotate(2);
            set_bg("bg.jpg");
            figure.draw(0);
            break;
         case MOVE_UP:
            figure.move(1, 1);
            set_bg("bg.jpg");
            figure.draw(0);
            break;
         case MOVE_RIGHT:
            figure.move(0, 1);
            set_bg("bg.jpg");
            figure.draw(0);
            break;
         case MOVE_LEFT:
            figure.move(0, -1);
            set_bg("bg.jpg");
```

```
            figure.draw(0);
            break;
         case MOVE_DOWN:
            figure.move(1, -1);
            set_bg("bg.jpg");
            figure.draw(0);
            break;
         case SAVE:
            save();
            break;
         case EXIT:
            return 0;
      }
   }
}
```

Файл task.cpp

```
#include "task.h"

Polyhedron::Polyhedron(){
   ifstream f("info_for_figure.txt");
   f>>vertex_num>>surface_num;
   vertexs.resize(vertex_num, vector<double> (3));
   for(int i=0; i<vertex_num; i++){
      for(int j=0; j<3; j++){
         f>>vertexs[i][j];
      }
   }
   surfaces.resize(surface_num, vector<int> (vertex_num));
   for(int i=0; i<surface_num; i++){
      for(int j=0; j<4; j++){
         f>>surfaces[i][j];
      }
   }
   f.close();
}

void Polyhedron::rotate(int type){
   for(int i=0; i<vertex_num; i++){
      vertexs[i][0] -= d[0];
      vertexs[i][1] -= d[1];
      vertexs[i][2] -= d[2];
   }
   switch(type){
      case 0:
         for(int i=0; i<vertex_num; i++){
            vertexs[i][0] = vertexs[i][0]*cos(acos(-1)/24)+vertexs[i]
            vertexs[i][1] = vertexs[i][1]*cos(acos(-1)/24)-vertexs[i]
         }
         break;
      case 1:
         for(int i=0; i<vertex_num; i++){
            vertexs[i][0] = vertexs[i][0]*cos(acos(-1)/24)-vertexs[i]
            vertexs[i][2] = vertexs[i][0]*sin(acos(-1)/24)+vertexs[i]
         }
         break;
      case 2:
         for(int i=0; i<vertex_num; i++){
            vertexs[i][1] = vertexs[i][1]*cos(acos(-1)/24)+vertexs[i]
            vertexs[i][2] = vertexs[i][2]*cos(acos(-1)/24)-vertexs[i]
         }
         break;
   }
   for(int i=0; i<vertex_num; i++){
      vertexs[i][0] += d[0];
      vertexs[i][1] += d[1];
```

```
      vertexs[i][2] += d[2];
   }
}

void Polyhedron::move(int type, int direction){
   d[0] += direction*I[type];
   d[1] += direction*J[type];
   d[2] += direction*K[type];
   for(int i=0; i<vertex_num; i++){
      vertexs[i][0] += direction*I[type];
      vertexs[i][1] += direction*J[type];
      vertexs[i][2] += direction*K[type];
   }
}

void Polyhedron::draw(int type){
   int k;
   setcolor(BLACK);
   setfillstyle(SOLID_FILL, WHITE);
   switch(type){
      case 0:
         for(int i=0; i<surface_num; i++){
            k = 0;
            for(int j=0; j<4; j++){
               p[2*k] = WIDTH/4 + vertexs[surfaces[i][j]][0];
               p[2*k+1] = HEIGHT/2 - vertexs[surfaces[i][j]][1];
               k++;
            }
            drawpoly(k, p);
         }
         break;
      case 1:
         for(int i=0; i<surface_num; i++){
            k = 0;
            for(int j=0; j<4; j++){
               p[2*k] = WIDTH/2 + vertexs[surfaces[i][j]][0];
               p[2*k+1] = 5*HEIGHT/8 - vertexs[surfaces[i][j]][2];
               k++;
            }
            drawpoly(k, p);
         }
         break;
      case 2:
         for(int i=0; i<surface_num; i++){
            k = 0;
            for(int j=0; j<4; j++){
               p[2*k] = 3*WIDTH/4 + vertexs[surfaces[i][j]][1];
               p[2*k+1] = 5*HEIGHT/8 - vertexs[surfaces[i][j]][2];
               k++;
            }
            drawpoly(k, p);
         }
```

```
            break;
    }
}
```

Файл task.h

```
#ifndef TASK_H
#define TASK_H

#include <iostream>
#include <vector>
#include <graphics.h>
#include <math.h>
#include <fstream>
#include "control.h"

using namespace std;

class Polyhedron{
private:
    int I[3] = {1, 0, 0};
    int J[3] = {0, 1, 0};
    int K[3] = {0, 0, 1};
    int p[20];
    int surface_num, vertex_num;
    int d[3] = {0, 0, 0};
    vector<vector<double>> vertexs;
    vector<vector<int>> surfaces;
public:
    Polyhedron();
    void rotate(int);
    void move(int, int);
    void draw(int);
};

#endif
```

Файл control.h

```
#ifndef CONTROL_H
#define CONTROL_H

#define WIDTH 900
#define HEIGHT 800

enum control_values { NONE = -1, SAVE,
                      X_ROTATE, Y_ROTATE, Z_ROTATE,
                      MOVE_UP, MOVE_RIGHT, MOVE_LEFT, MOVE_DOWN, EXIT

struct Control
```

```
{
   int left;
   int top;
   int right;
   int bottom;
};

void create_control(int, int, int);
int select_control();
void set_bg(const char*);
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 + 74;
   controls[i].bottom = top  + 74;
}

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 set_bg(const char* name){
   IMAGE *image;
   image = loadBMP(name);
```

```
    putimage(0, 0, image, COPY_PUT);
    freeimage(image);
}

void save(){
    IMAGE *output;
    output = createimage(WIDTH+1, HEIGHT+1);
    getimage(0, 0, WIDTH, HEIGHT, output);
    saveBMP("output.jpg", output);
    freeimage(output);
}
```

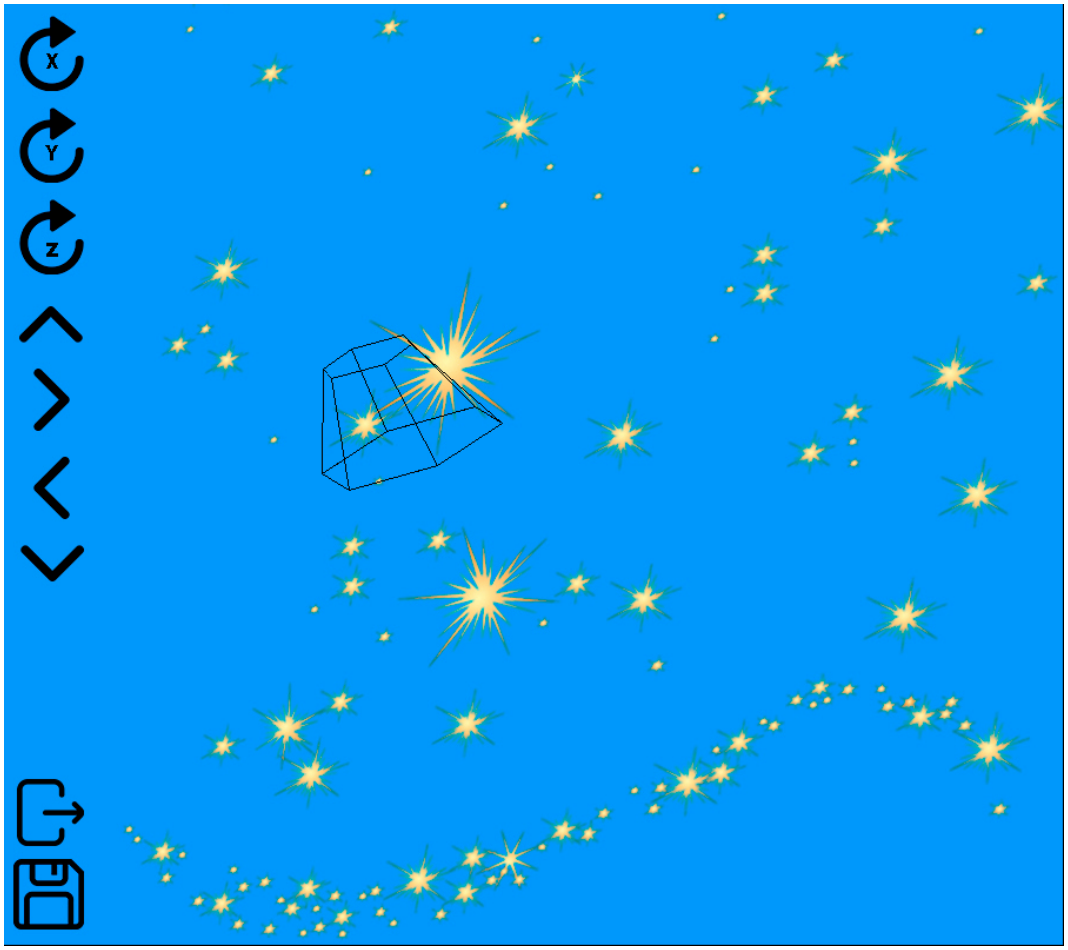

Рисунок 4.1 – Результат выполнения программы