Московский Авиационный Институт
(Национальный исследовательский Университет)

Факультет: «Информационные технологии и прикладная математика»
Кафедра: 806 «Вычислительная математика и программирование»

# Лабораторная работа №2
# по курсу «Компьютерная графика»

| | |
|---|---|
| Студент: | Марков А.Н. |
| Группа: | М80-308Б-18 |
| Преподаватель: | Филиппов Г.С. |
| Оценка: | |
| Дата: | |

Москва
2020

**1. Постановка задачи.**

Разработать формат представления многогранника и процедуру его каркасной отрисовки в ортографической и изометрической проекциях. Обеспечить удаление невидимых линий и возможность пространственных поворотов и масштабирования многогранника. Обеспечить автоматическое центрирование и изменение размеров изображения при изменении размеров окна.

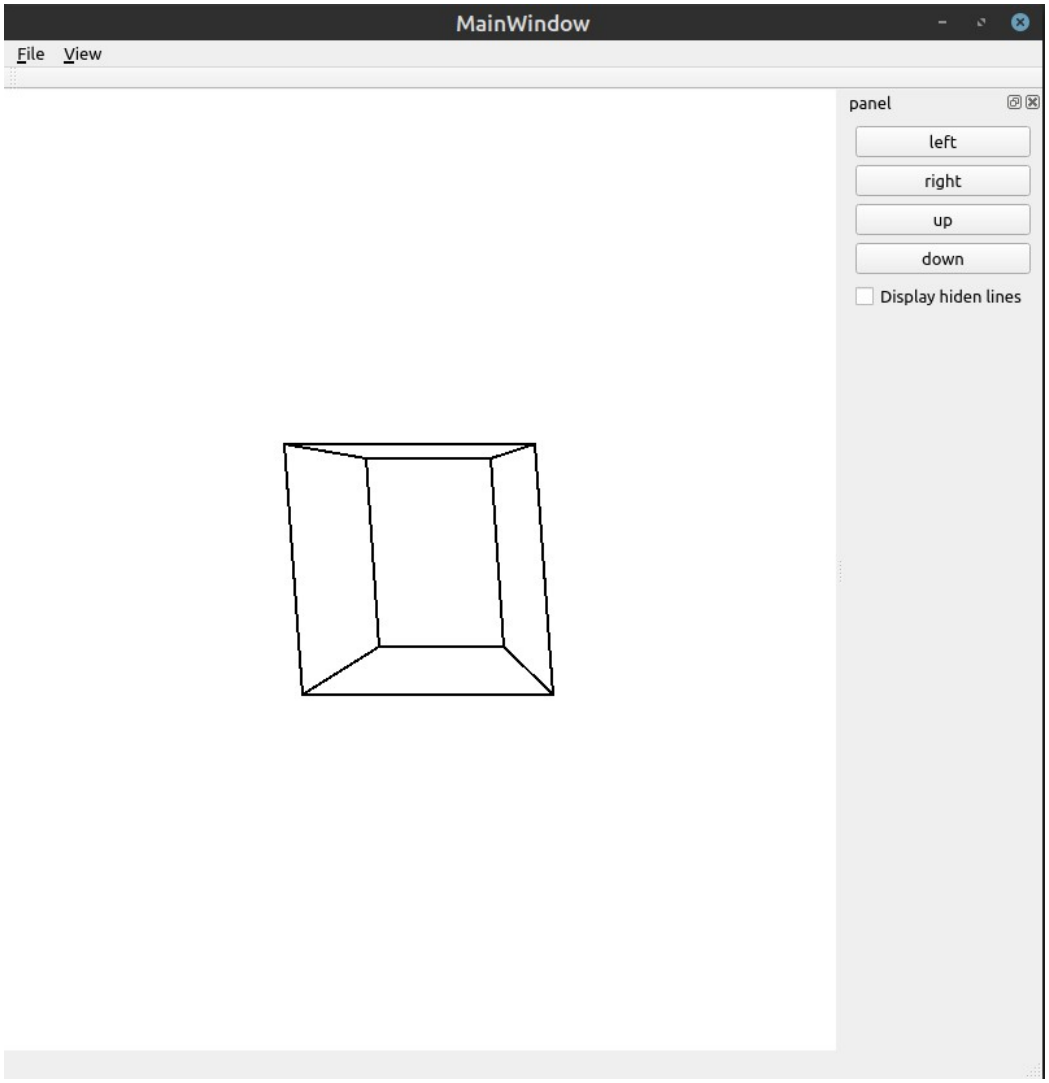
Вариант №5: Обелиск (усеченный клин).

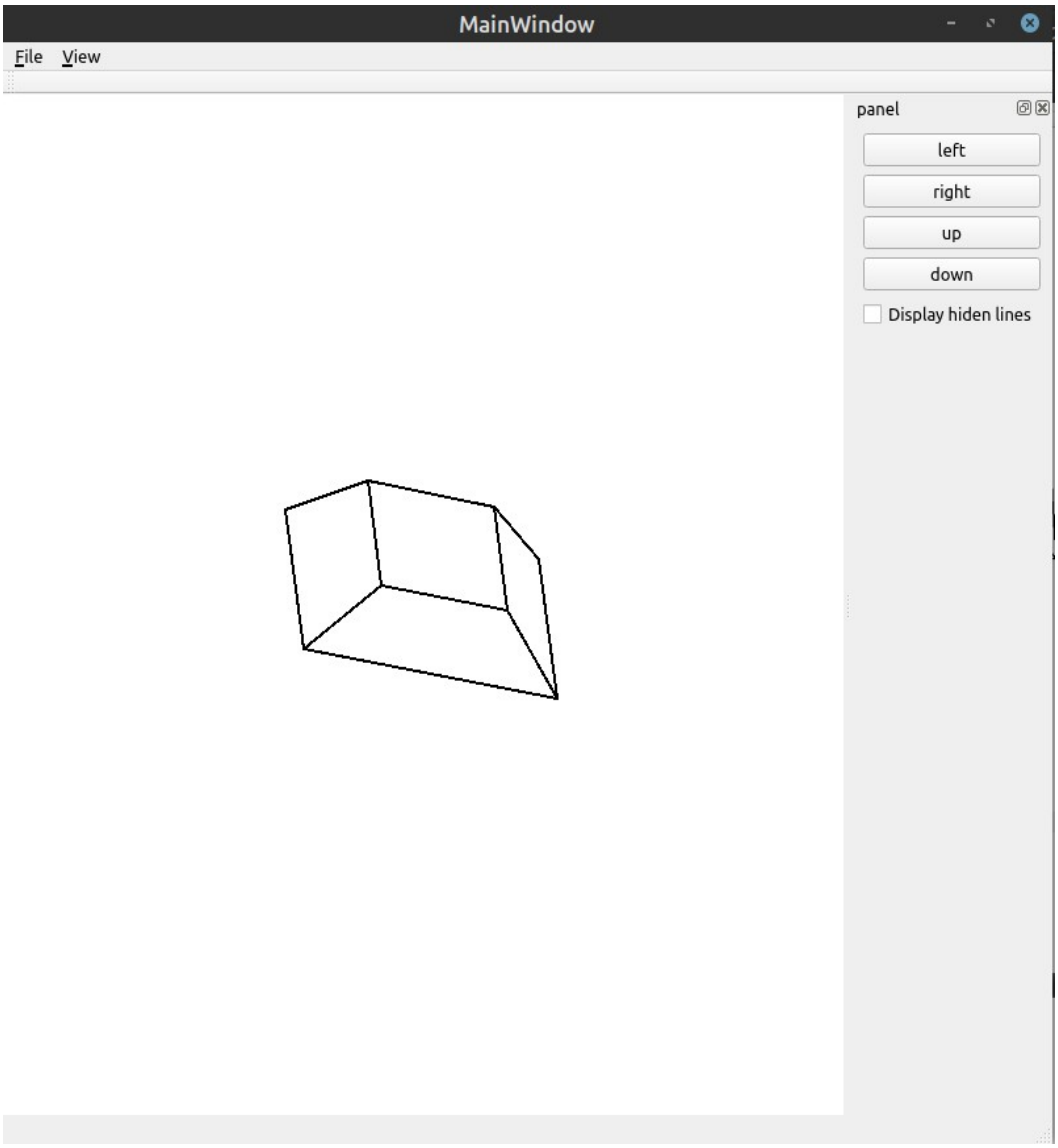
**2. Решение задачи.**

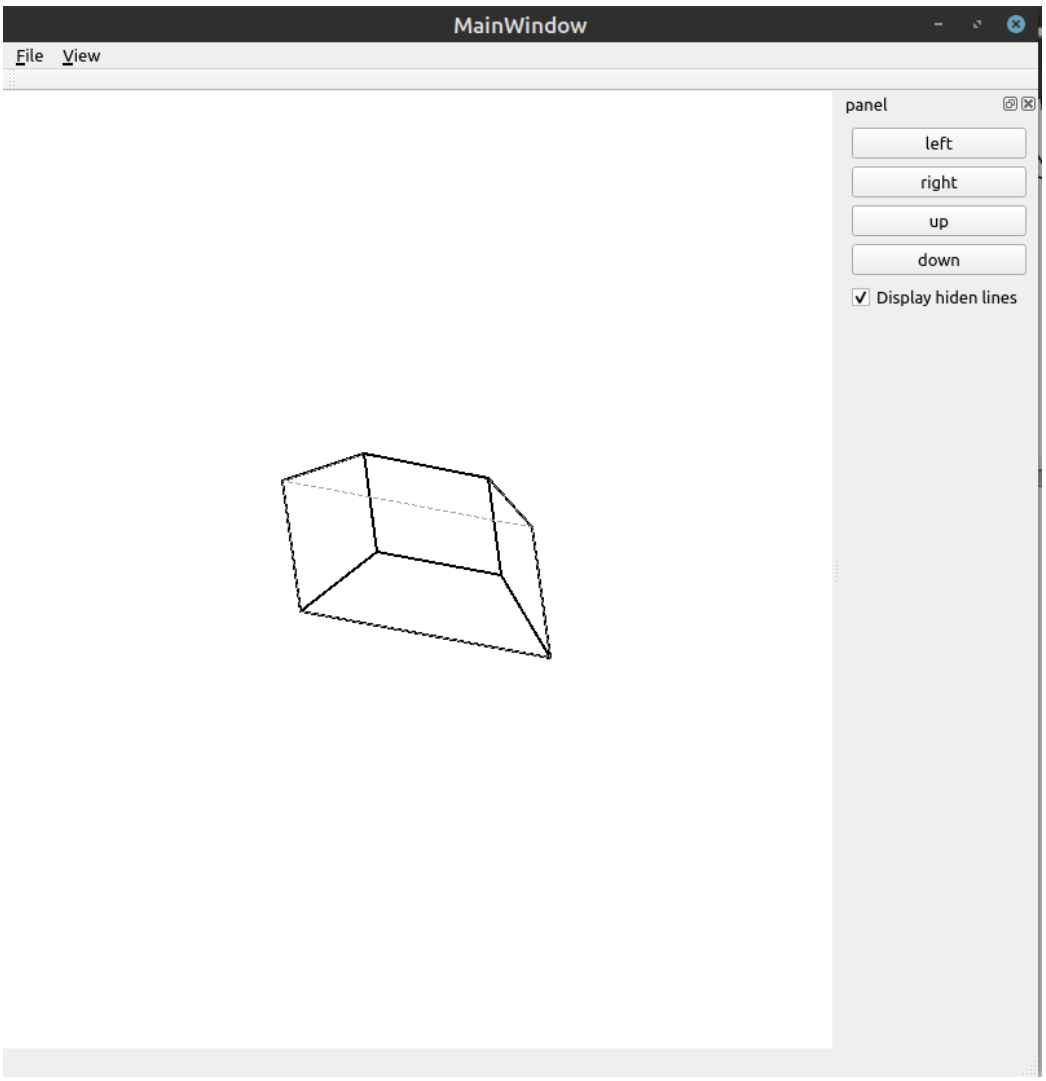
Для решения задачи я решил использовать C++ и фреймворк Qt, в котором использовал библиотеку QPainter.

Я создал класс Polygon для хранения полигонов, класс Obelisk, представляющий фигуру обелиск. Такая фигура состоит из шести полигонов. Все преобразования для фигуры выполняются для каждой полигона, и в каждом полигоне преобразования выполняются для каждой точки. Так выполняются пространственные повороты фигуры и масштабирование фигуры.

**3. Демонстрация работы программы.**




MainWindow
File
View
panel
left
right
up
down
Display hiden lines




**4. Листинг программы.**

**polygon.h:**

```
#ifndef POLYGON_H
#define POLYGON_H

#include <vector>
#include <QPainter>

class Polygon
{
    std::vector<std::vector<double>> verticies;
public:
    Polygon();
    Polygon(const std::vector<std::vector<double>> &v);

    std::vector<double> get_normal();
    void change_verticies(const std::vector<std::vector<double>> &v);
    void add_vertex(const std::vector<double> &v);
```

```
    void add_vertex(double x, double y, double z, double d);
    void clear_verticies();
    void draw(QPainter *ptr, int center_x, int center_y);
};

#endif // POLYGON_H
```

**polygon.cpp:**

```
#include "polygon.h"

#include <QTextStream>

Polygon::Polygon()
{

}

Polygon::Polygon(const std::vector<std::vector<double>> &v) : verticies(v) {}

std::vector<double> Polygon::get_normal() {
    std::vector<double> first = {
        verticies[1][0] - verticies[0][0],
        verticies[1][1] - verticies[0][1],
        verticies[1][2] - verticies[0][2]
    };
    std::vector<double> second = {
        verticies[verticies.size() - 1][0] - verticies[0][0],
        verticies[verticies.size() - 1][1] - verticies[0][1],
        verticies[verticies.size() - 1][2] - verticies[0][2]
    };
    std::vector<double> normal = {
        first[1] * second[2] - second[1] * first[2],
        second[0] * first[2] - first[0] * second[2],
        first[0] * second[1] - second[0] * first[1]
    };

    return normal;
}

void Polygon::change_verticies(const std::vector<std::vector<double>> &v) {
    for (auto &it: verticies) {
        std::vector<double> res(4);
        for (size_t i = 0; i < 4; i++) {
            for (size_t j = 0; j < 4; j++) {
                res[i] += v[i][j] * it[j];
```

```
            }
        }
        it = res;
    }
}

void Polygon::add_vertex(const std::vector<double> &v) {
    verticies.push_back(v);
}

void Polygon::add_vertex(double x, double y, double z, double d) {
    verticies.push_back(std::vector<double>{x, y, z, d});
}

void Polygon::clear_verticies() {
    verticies.clear();
}

void Polygon::draw(QPainter *ptr, int center_x, int center_y) {
    for (size_t i = 0; i < verticies.size() - 1; i++) {
        ptr->drawLine(static_cast<int>(verticies[i][0] + center_x),
                      static_cast<int>(verticies[i][1] + center_y),
                      static_cast<int>(verticies[i + 1][0] + center_x),
                      static_cast<int>(verticies[i + 1][1] + center_y));
    }
    ptr->drawLine(static_cast<int>(verticies[0][0] + center_x),
                  static_cast<int>(verticies[0][1] + center_y),
                  static_cast<int>(verticies[verticies.size() - 1][0] + center_x),
                  static_cast<int>(verticies[verticies.size() - 1][1] + center_y));
}
```

**obelisk.h:**

```
#ifndef OBELISK_H

#define OBELISK_H

#include "polygon.h"

class Obelisk
{
private:
    std::vector<Polygon> polygons;
    bool displayHidenLines;
public:
    Obelisk();
```

```
    Obelisk(const std::vector<Polygon> &p);

    void set_displayHidenLines(bool b);
    bool get_displayHidenLines();
    void change_all_polygons(const std::vector<std::vector<double>> &v);
    void add_polygon(const Polygon &p);
    void draw(QPainter *ptr, int center_x, int center_y);
};

#endif // OBELISK_H
```

**obelisk.cpp:**

```
#include "obelisk.h"

Obelisk::Obelisk() : displayHidenLines(false)
{

}

Obelisk::Obelisk(const std::vector<Polygon> &p) : Obelisk() {
    polygons = p;
}

void Obelisk::set_displayHidenLines(bool b) {
    displayHidenLines = b;
}

bool Obelisk::get_displayHidenLines() {
    return displayHidenLines;
}

void Obelisk::change_all_polygons(const std::vector<std::vector<double>> &v) {
    for (auto &it: polygons) {
        it.change_verticies(v);
    }
}

void Obelisk::add_polygon(const Polygon &p) {
    polygons.push_back(p);
}

void Obelisk::draw(QPainter *ptr, int center_x, int center_y) {
    for (auto p : polygons) {
        auto p_normal = p.get_normal();
        if (p_normal[2] > 0) {
```

```
            p.draw(ptr, center_x, center_y);
        } else {
            if (displayHidenLines) {
                QPen new_pen(Qt::gray, 1, Qt::DashLine);
                QPen old_pen = ptr->pen();
                ptr->setPen(new_pen);
                p.draw(ptr, center_x, center_y);
                ptr->setPen(old_pen);
            }
        }
    }
}
```

**5. Вывод.**

В ходе выполнения данной лабораторной работы я научился строить трехмерные модели, освежил свои знания по линейной алгебре.