

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:	Аксенов А.Е.
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.

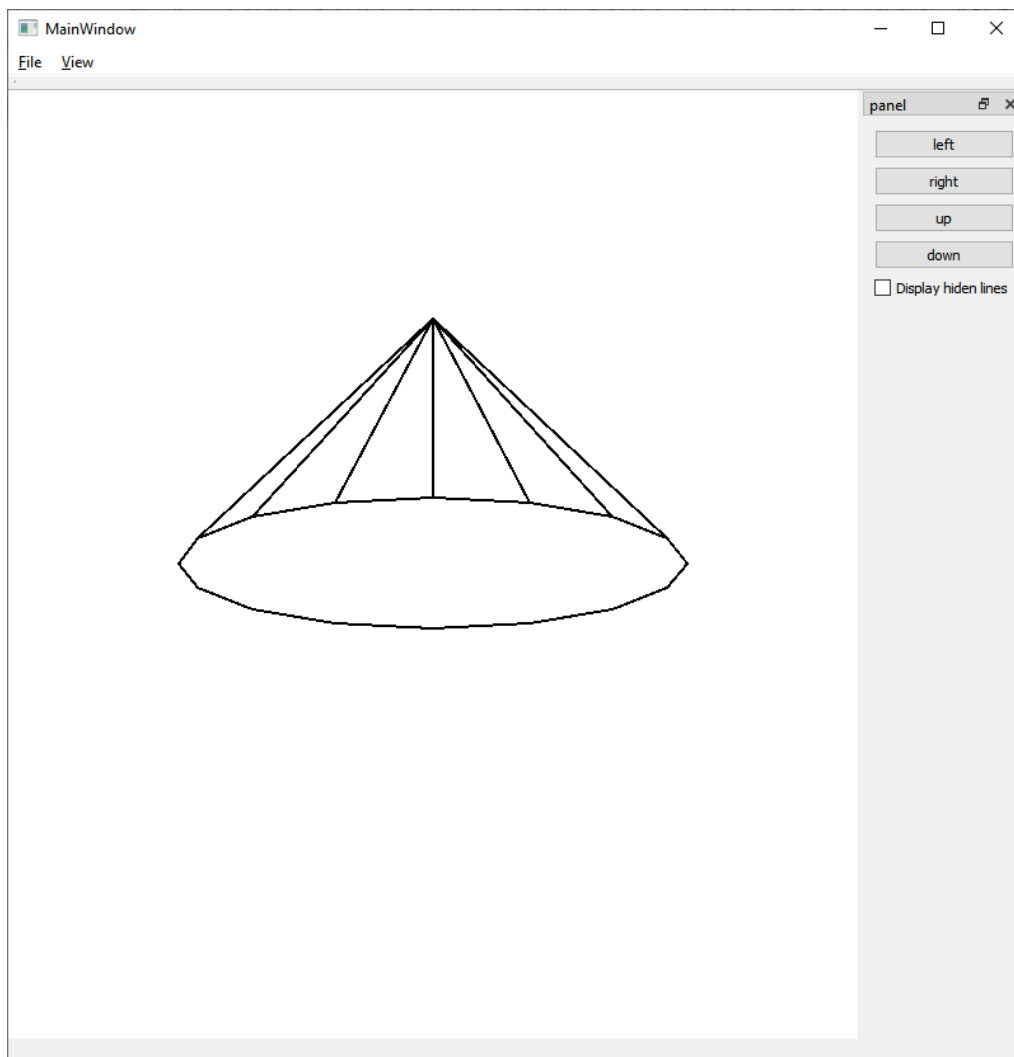
Вариант №16: 16 – гранная прямая правильная пирамида.

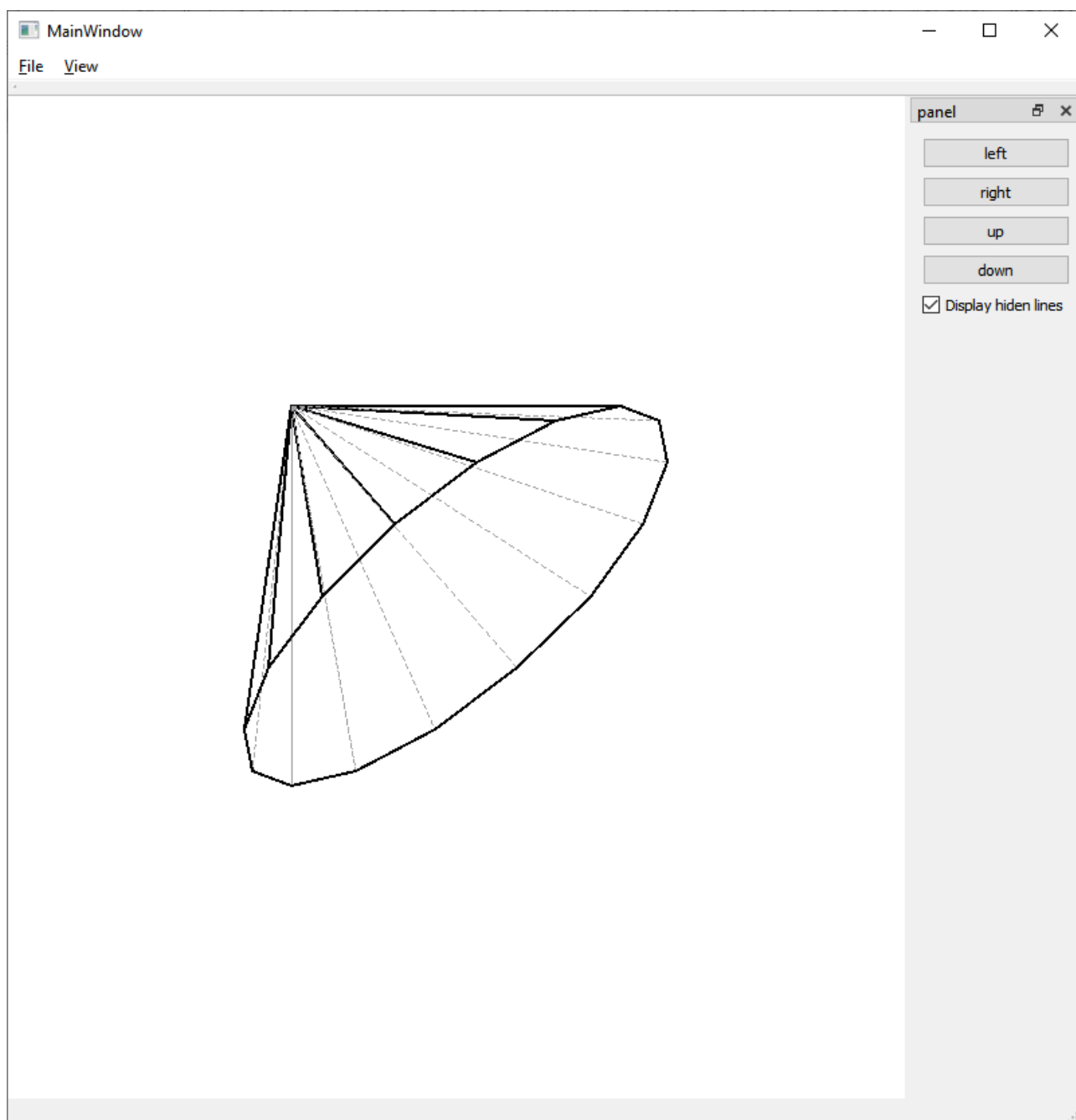
## 2. Решение задачи.

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

Я создал класс Polygon для хранения полигонов, класс Pyramid, представляющий фигуру 16 – гранной пирамиды. Такая фигура состоит из семнадцати 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.





#### 4. Листинг программы.

##### **polygon.h:**

```
#ifndef POLYGON_H
#define POLYGON_H

#include <vector>
#include <QPainter>

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

```

    std::vector<double> get_normal();
    void change_vertices(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_vertices();
    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) : vertices(v) {}

std::vector<double> Polygon::get_normal() {
    std::vector<double> first = {
        vertices[1][0] - vertices[0][0],
        vertices[1][1] - vertices[0][1],
        vertices[1][2] - vertices[0][2]
    };
    std::vector<double> second = {
        vertices[vertices.size() - 1][0] - vertices[0][0],
        vertices[vertices.size() - 1][1] - vertices[0][1],
        vertices[vertices.size() - 1][2] - vertices[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_vertices(const std::vector<std::vector<double>> &v) {
    for (auto &it: vertices) {
        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) {
    vertices.push_back(v);
}

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

void Polygon::clear_vertices() {

```

```

        vertices.clear();
    }

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

```

## pyramid.h:

```

#ifndef PYRAMID_H
#define PYRAMID_H

#include "polygon.h"

class Pyramid
{
private:
    std::vector<Polygon> polygons;
    bool displayHiddenLines;
public:
    Pyramid();
    Pyramid(const std::vector<Polygon> &p);

    void set_displayHiddenLines(bool b);
    bool get_displayHiddenLines();
    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 // PYRAMID_H

```

## pyramid.cpp:

```

#include "pyramid.h"

Pyramid::Pyramid() : displayHiddenLines(false)
{
}

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

void Pyramid::set_displayHiddenLines(bool b) {
    displayHiddenLines = b;
}

bool Pyramid::get_displayHiddenLines() {
    return displayHiddenLines;
}

```

```

}

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

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

void Pyramid::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 (displayHiddenLines) {
                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.