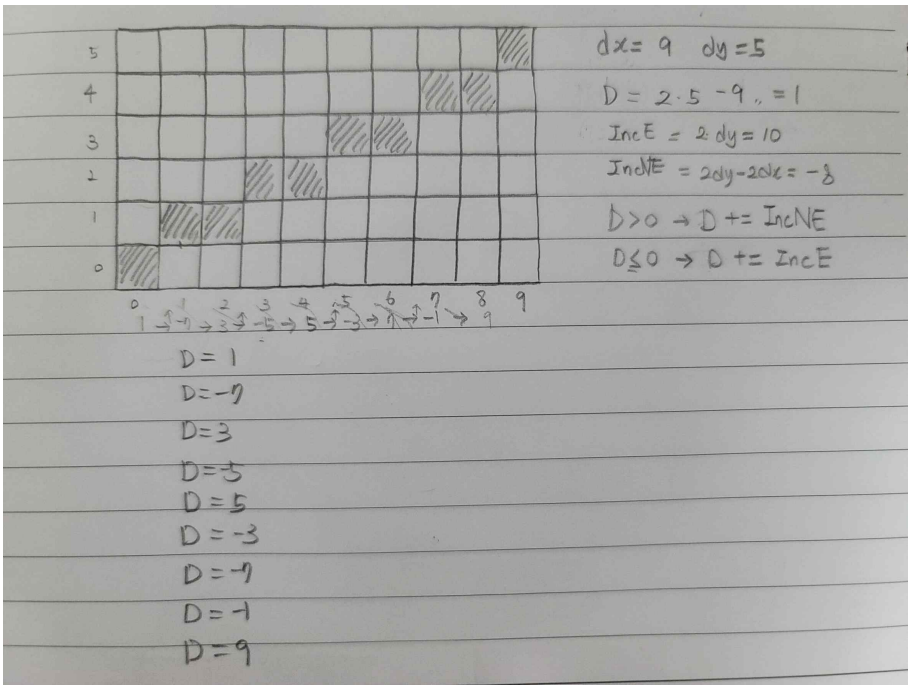
컴퓨터 그래픽스 HW3

201501489 최영진

1. 1)

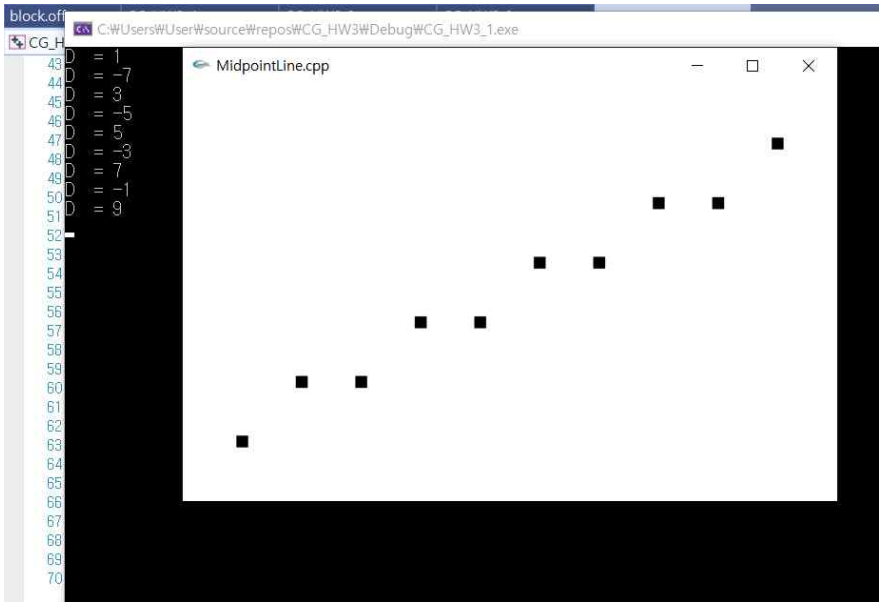
① 실행 결과



② 설명 및 토의

D>0일 때는 동북쪽으로, D≤0일 때는 동쪽으로 이동하며 D값을 변형해서 적용했다. 반복 행위이기에 D값을 먼저 구한 다음에 pixel을 선택한다는 것만 헷갈리지 않도록 유의하면 된다.

2) ① 실행 결과



② 설명 및 토의

1)과 동일한 결과의 D값이 나왔다. (-1,-1)부터 (10,6)까지의 가시공간에서 vertex의 크기를 10×10pixel로 출력했다. (glPointSize는 glBegin과 glEnd사이에 호출될 수 있는 명령어가 아님 주의)

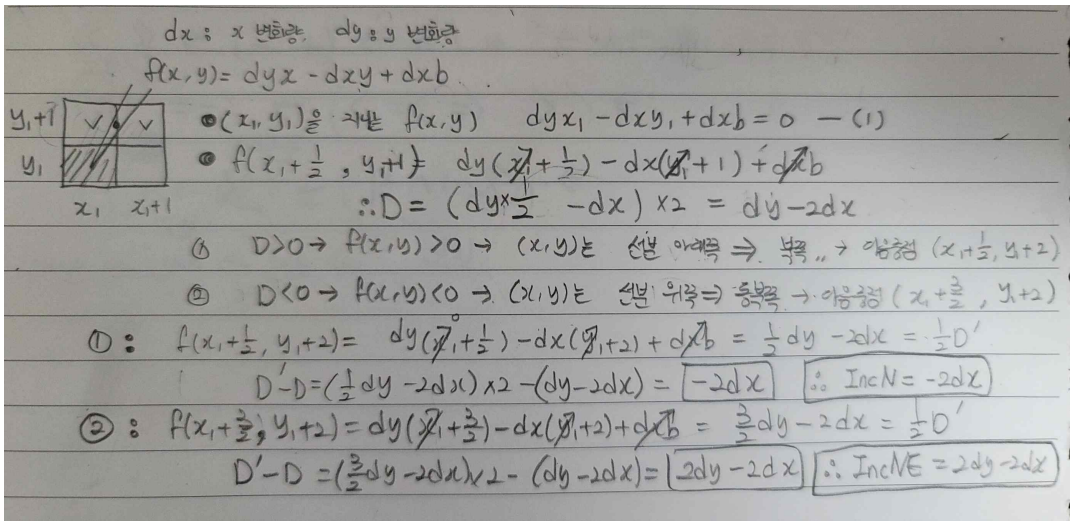
3) ① 실행 결과

② 설명 및 토의

D값이 0보다 큰지, 작거나 같은지 판별하는 것은 모두 중점이 선분 아래에 있는지, 선분 위에 있는지 구별하는 것과 동일하다. 즉, D값의 판별이 이루어지는 횟수가 중점이 이용된 횟수이자, 이용한 중점의 수이다.

2. 1)

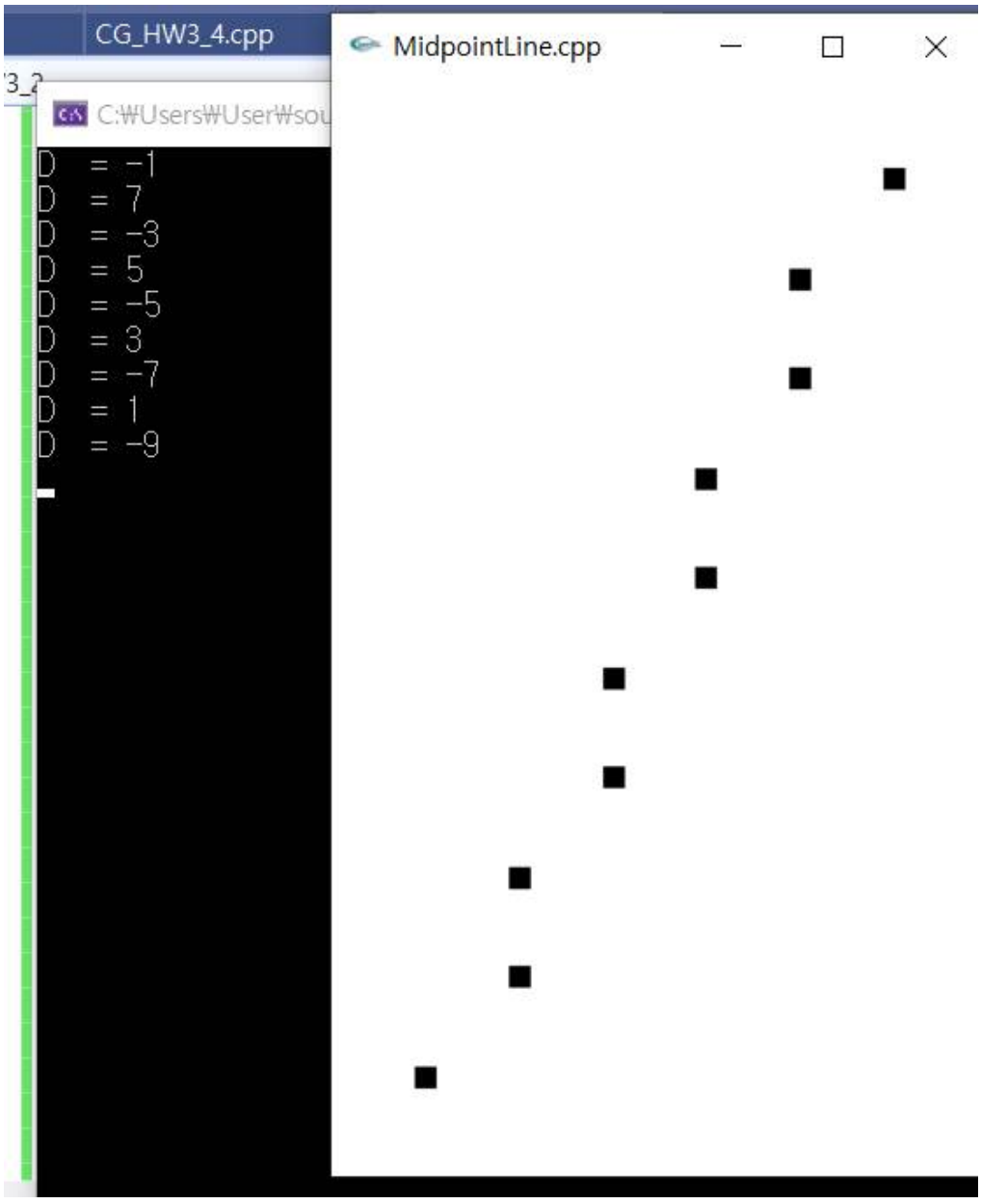
① 실행 결과



② 설명 및 토의

기울기가 1보다 클 때, 후보 픽셀의 위치는 현재 위치의 북쪽과 동북쪽에 있다. 따라서 (x_1, y_1)을 기준으로 다음 픽셀을 결정지을 중점은 $(x_1+1/2, y_1+1)$ 이다. $f(x_1, y_1)=0$이므로, $f(x_1+1/2, y_1+1)$ = dy*1/2 – dx가 되고 결정변수 D는 dy-2dx가 된다. D값이 0보다 크면, 중점이 선분보다 아래쪽에 위치한다는 뜻이므로 북쪽 픽셀을 선택하게 되고 다음 중점은 $(x_1+1/2, y_1+2)$이 된다. D값이 0보다 작으면 선분이 중점의 아래 쪽에 존재한다는 의미이므로 동북쪽 픽셀을 선택, 다음 중점은 $(x_1+3/2, y_1+2)$이 된다. D값의 변화량을 알기 위해 각각의 경우에 대해 기존 D값과의 차를 구해주었다.

2) ① 실행 결과



② 설명 및 토의

D=0인 경우를 포함했을 때는 북쪽으로 움직이도록 설정했다. (-1,-1)부터 (6,10)까지의 가시공간에서 vertex의 크기를 10×10pixel로 출력했다.

3.

1) ① 실행 결과

```
#include <iostream>
#include <fstream>
#include <string>
#include <GL/glut.h>

using namespace std;

class MyVertex
{
public:
	float x;
	float y;
	float z;
	MyVertex(float x=0.0f , float y=0.0f , float z=0.0f) :x(x), y(y), z(z) {};
	~MyVertex() {};
};


class Myface
{
public:
	int n;   int v1; int v2; int v3;
	Myface(int n=3, int v1=0, int v2=0, int v3=0) : n(n), v1(v1), v2(v2), v3(v3) {};
	~Myface() {};
};


int main()
{
	int v_num;
	int f_num;
	int edge;
	int sn;
	int su;
	string sf;
	string sx;
```

```
string sy;
string sz;


ifstream fin("block.off");
if (!fin) {
	cout << "파일을 읽어오지 못했습니다." <<endl;
}
else {
	string line;
	//off
	getline(fin, line);
	// vertex수
	fin >> line;
	v_num = (int)stoi(line);
	MyVertex* vertex = new MyVertex[v_num];
	// face수
	fin >> line;
	f_num = (int)stoi(line);
	Myface* face = new Myface[f_num];
	// edge
	fin >> line;
	edge = (int)stoi(line);
	fin.ignore();

	//vertext배열 저장
	for (int i = 0; i < v_num; i++) {
		getline(fin, line);
		sn = line.find(" ");
		sx = line.substr(0, sn);
		su = line.find(" ", sn + 1);
		sy = line.substr(sn+1, su-sn-1);
		sz = line.substr(su+1);

		MyVertex vi((float)stof(sx), (float)stof(sy), (float)stof(sz));
		vertex[i] = vi;
	}


	for (int j = 0; j < f_num; j++) {
		getline(fin, line);
		sf = line.substr(0, 1);
```

```
            sn = line.find(" ",2);
            sx = line.substr(2, sn-2);
            su = line.find(" ", sn + 1);
            sy = line.substr(sn + 1, su - sn - 1);
            sz = line.substr(su + 1);
            Myface fi((int)stoi(sf), (int)stoi(sx), (int)stoi(sy), (int)stoi(sz));
            face[j] = fi;
        }
    }
}
```

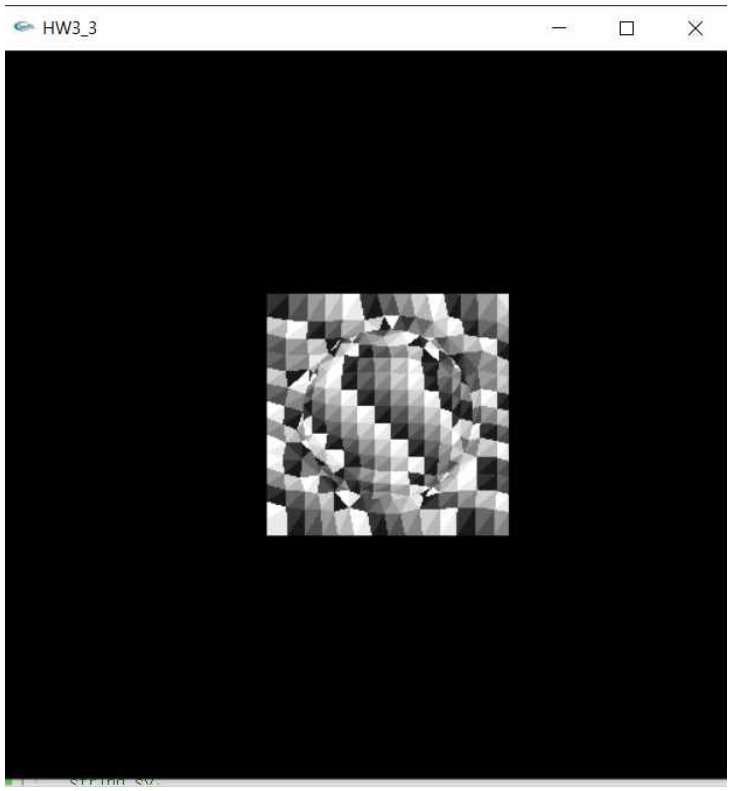
② 설명 및 토의

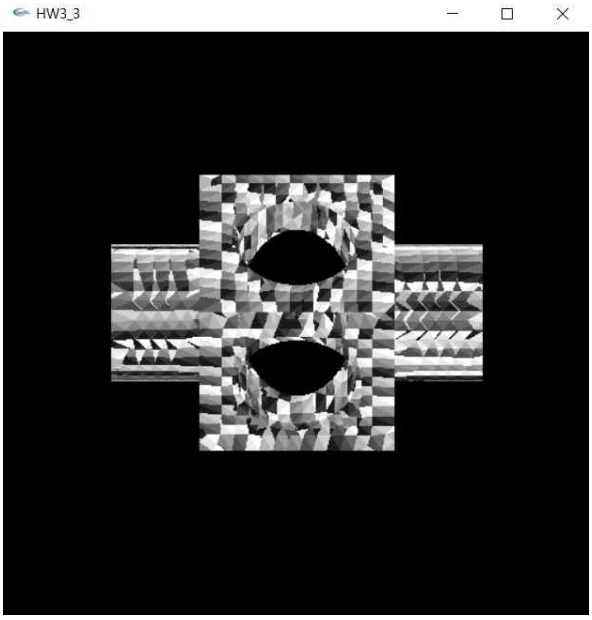
MyVertex 클래스와 Myface 클래스를 만들어 각각 저장할 정보들을 포함하고 있는 vertex배열과 face배열을 만들었다. for문을 돌며 파일에서 한 줄씩 읽고 " "를 기준으로 정보를 나누어 배열의 원소에 집어넣었다.

토의하고 싶은 사항은 파일에서 실수를 float으로 읽어들이며 정밀도가 떨어지게 되는 점이다.

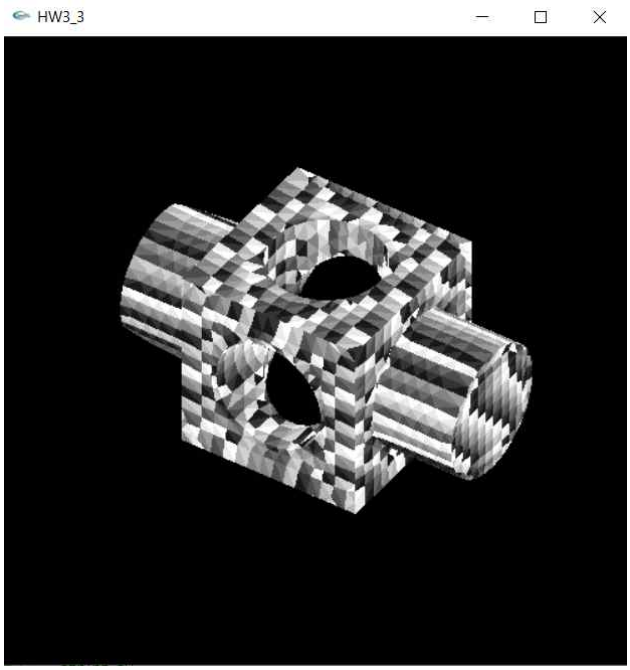
2)

① 실행 결과



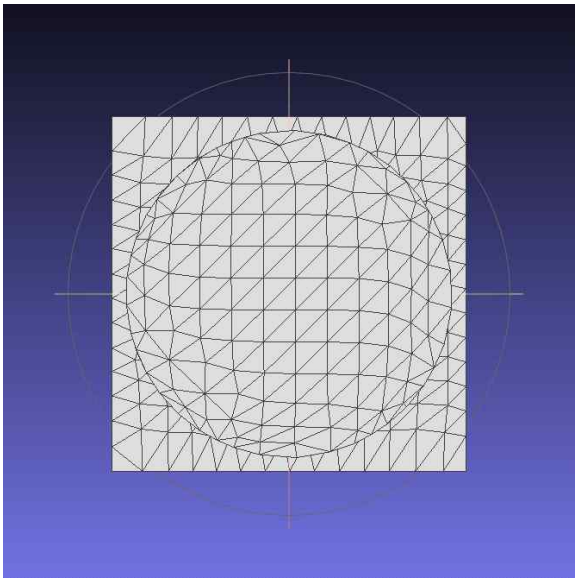


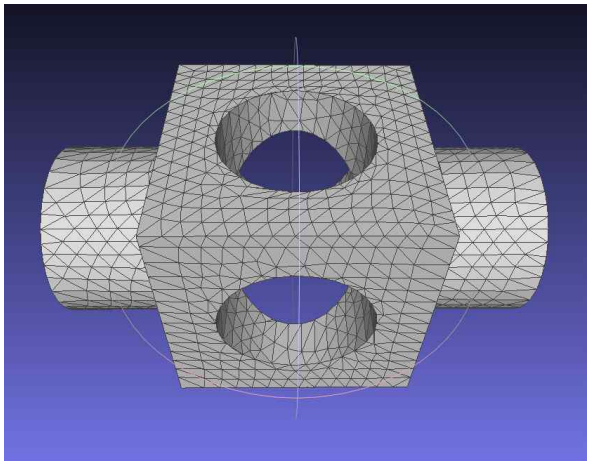
↑ (-15.0, 15.0, 0.0)에서 본 모습

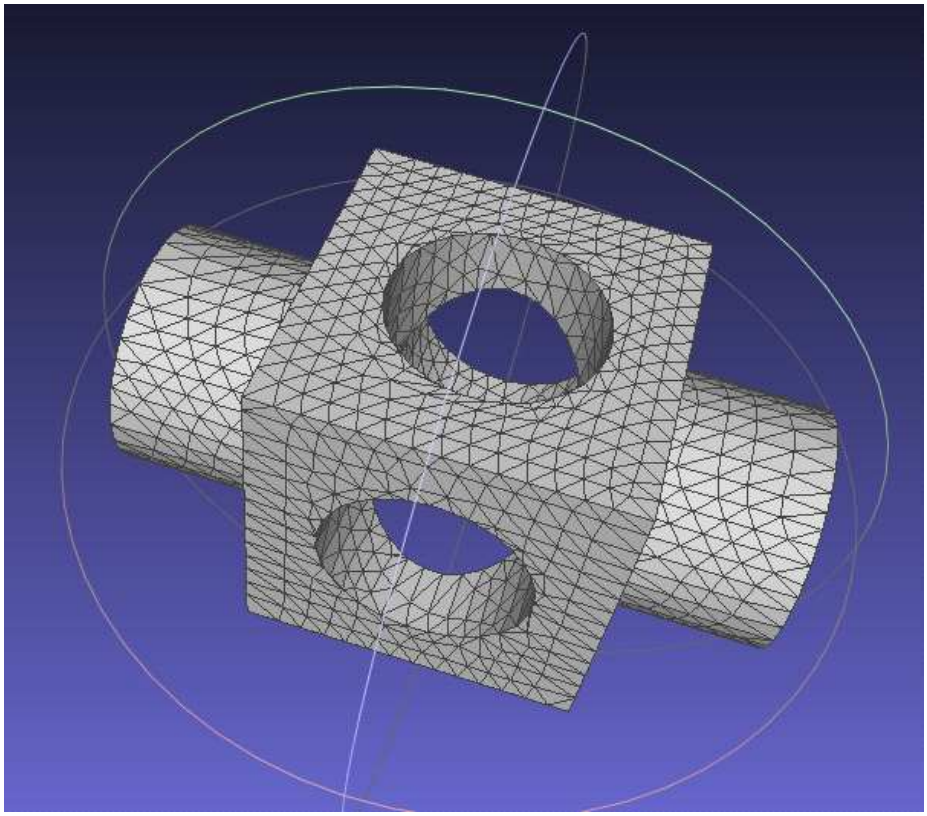


↑ (-15.0, 15.0, 10.0)에서 본 모습

② 설명 및 토의

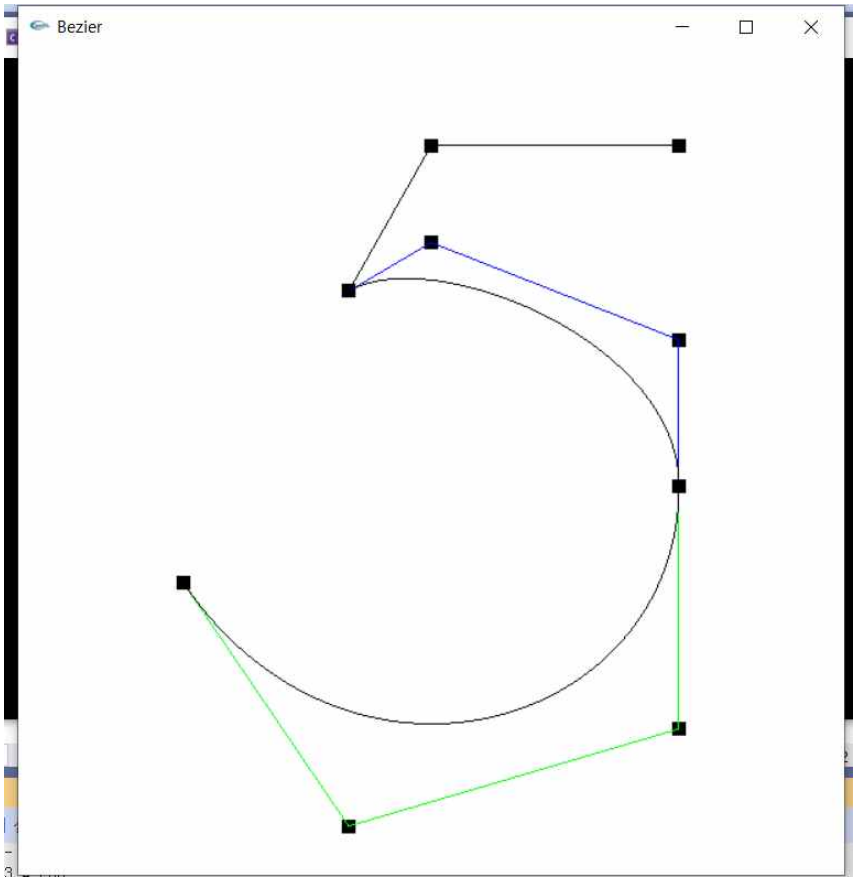






gluLookAt 함수로 이리저리 돌려가며 확인한 결과, meshlab으로 시각화한 것과 비슷하다. 다만 조명 처리를 하지 않고 각 폴리곤을 구별하기 위해 임의로 지정한 색으로만 그렸더니 원근감이 떨어져서 물체의 모양을 바로 인식하기는 어렵다. 가시 부피는 glOrtho(-30.0, 30.0, -30.0, 30.0, -30.0, 30.0)로 설정했다.

4. ① 실행 결과



② 설명 및 토의

가시공간은 gluOrtho2D(-2, 8, -6, 11)로 설정했으며 controlpoint는 4개씩, 총 7개(하나는 공통이라서 8개가 아님.)를 만들었다. (controlpoint들을 포함하여 모든 vertex를 ctrlpoints배열에 넣어 처리)

첫 번째 Bezier curve의 control point들은 (0.0, 0.0, 0.0), (2.0, -5.0, 0.0), (6.0, -3.0, 0.0), (6.0, 2.0, 0.0)으로 설정했고, 두 번째 Bezier curve의 control point들은 (6.0, 5.0, 0.0), (3.0, 7.0, 0.0), (2.0, 6.0, 0.0)이다.

u값을 잘게 쪼갠 후 0부터 1까지 Bezier curve 식($u(x) = (1-u)^{3*}P0(x) + (1-u)^2u*P1(x) + (1-u)u^2*P2(x) + u^3*P3(x)$, $u(y) = (1-u)^{3*}P0(y) + (1-u)^2u*P1(y) + (1-u)u^2*P2(y) + u^3*P3(y)$)에 맞추어 vertex를 생성, GL_LINE_STRIP을 이용하여 마지막 두 점 (3.0, 9.0, 0.0), (6.0, 9.0, 0.0)까지 그렸다.