Seoul National University
College of Engineering
Department of Naval Architecture and Ocean Engineering
1, Gwanak-ro, Gwanak-gu, Seoul 151-744, Korea

Spring 2023

컴퓨터 이용 선박 설계

(Computer-Aided Ship Design)

PA #5

Instructor name	김 태 완
Student name	장 민 준
Department	조선해양공학과
Student ID	2019-10738
Submission date	23/06/21
Grade	

목차

1. Coons-patch

Coons-patch의 메인 아이디어는 Boundary curve 사이를 보간하여 부드러운 표면을 생성하는 것이다.

Rectangular Coons patch를 위해서는 네 개의 경계 곡선이 필요하다, 하단(v=0)과 상단(v=1)에 두 개의 "수평" 곡선, 그리고 좌측(u=0)과 우측(u=1)에 두 개의 "수직" 곡선. 그리고 표면은 이 네 개의 곡선의 가중 평균, 그리고 모서리가 올바르게 매치되도록 하는 수정 항을 더한 것으로 정의된다. 이를 계산하기 위한 식은 LN_8 18~19p에 나와 있다.

Triangle Coons patch는 비슷하지만, 네 개의 곡선 대신 세 개의 곡선을 사용한다. 표면은 여전히 곡선의 가중 평균으로 정의되지만, 계산 방식은 각 삼각형에 대해서 barycentric coordinate를 따라 구한 후, 다시 재귀적으로 반복하여 최종 삼각형의 점을 구해서 찾을 수 있다.

이러한 과정을 위해서는 먼저 해당 Bezier Surface를 구성하는 Control Point를 찾아야 한다. 이를 찾는 방법을 이어서 말하도록 하겠다.

2. Point normal

Point normal이란 Surface 위 Vertex에서의 법선 벡터를 나타낸다. 이를 근사하기 위해서 강의에서 사용하는 방식은, Vertex에 인접하는 Edge 바탕으로 Cross product를 수행하고, 이를 평균 내어 점의 Point normal로 결정한다. 이렇게 찾은 Point normal은 각 boundary curve를 근사하는 데 사용한다.

이 보고서에서는 Face 객체를 통해 각 Vertex에 연결된 Edge마다 Cross product를 수행하고, 그 값을 Vertex class의 Member variable로 저장한 후, 모든 Face 객체에 대해 반복하고 나서 Vertex마다 Point normal을 정할 것이다.

Point normal을 구하는 방법은 다음과 같다.

먼저, Vertex class에 다음과 같은 member variable을 추가한다

```
    vector<Vector3d> setting_point_normal;
    Vector3d point_normal = Vector3d(0.0, 0.0, 0.0);

    void calculate_ponit_normal(){
        for (Vector3d normal : this->setting_point_normal){
            this->point_normal + normal;
        }
        this->point_normal = this->point_normal/this->setting_point_normal.size();
    }
```

Face 객체를 탐색하며 Vertex에 Setting_Point_Normal을 할당할 것이며, 이후에 Calculate_ponit_normal을 통해 point_nomal을 세팅할 것이다. 이를 사용한 Face의 메서드는 다음과 같다.

```
void allocate_normal_to_vertex(){
    if(this->vertices.size() == 3){ // 삼각형의 경우
        Vector3d counterclock_0_point = Vector3d(vertices[1]->x - vertices[0]->x, vertices[1]->y -
vertices[0]->y, vertices[1]->z - vertices[0]->z);
        Vector3d clock_0_point = Vector3d(vertices[2]->x - vertices[0]->x, vertices[2]->y - vertices[0]->y,
vertices[2]->z - vertices[0]->z);
        this->vertices[0]->setting_point_normal.push_back(counterclock_0_point.cross(clock_0_point));

        Vector3d counterclock_1_point = Vector3d(vertices[2]->x - vertices[1]->x, vertices[2]->y -
vertices[1]->y, vertices[2]->z - vertices[1]->z);
        Vector3d clock_1_point = Vector3d(vertices[0]->x - vertices[1]->x, vertices[0]->y - vertices[1]->y,
vertices[0]->z - vertices[1]->z);
        this->vertices[1]->setting_point_normal.push_back(counterclock_1_point.cross(clock_1_point));

        Vector3d counterclock_2_point = Vector3d(vertices[0]->x - vertices[2]->x, vertices[0]->y -
vertices[2]->y, vertices[0]->z - vertices[2]->z);
        Vector3d clock_2_point = Vector3d(vertices[1]->x - vertices[2]->x, vertices[1]->y - vertices[2]->y,
vertices[1]->z - vertices[2]->z);
        this->vertices[2]->setting_point_normal.push_back(counterclock_2_point.cross(clock_2_point));
    }
    else{ // 사각형 혹은 그 이상의 경우
        for(int i = 0; i < this->vertices.size(); i++){
            if(i == 0){ // 0번 점 혹은 종점
                Vector3d counterclock_point = Vector3d(vertices[1]->x - vertices[0]->x, vertices[1]->y -
vertices[0]->y, vertices[1]->z - vertices[0]->z);
                Vector3d clock_point = Vector3d(vertices[this->vertices.size()-1]->x - vertices[0]->x,
vertices[this->vertices.size()-1]->y - vertices[0]->y, vertices[this->vertices.size()-1]->z - vertices[0]-
>z);
                this->vertices[i]->setting_point_normal.push_back(counterclock_point.cross(clock_point));
            }
            else if (i == this->vertices.size()-1){
                Vector3d counterclock_point = Vector3d(vertices[0]->x - vertices[this->vertices.size()-1]-
>x, vertices[0]->y - vertices[this->vertices.size()-1]->y, vertices[0]->z - vertices[this->vertices.size()-
1]->z);
                Vector3d clock_point = Vector3d(vertices[this->vertices.size()-2]->x - vertices[this-
>vertices.size()-1]->x, vertices[this->vertices.size()-2]->y - vertices[this->vertices.size()-1]->y,
vertices[this->vertices.size()-2]->z - vertices[this->vertices.size()-1]->z);
                this->vertices[i]->setting_point_normal.push_back(counterclock_point.cross(clock_point));
            }
            else{
                Vector3d counterclock_point = Vector3d(vertices[i+1]->x - vertices[i]->x, vertices[i+1]->y -
vertices[i]->y, vertices[i+1]->z - vertices[i]->z);
                Vector3d clock_point = Vector3d(vertices[i-1]->x - vertices[i]->x, vertices[i-1]->y -
vertices[i]->y, vertices[i-1]->z - vertices[i]->z);
                this->vertices[i]->setting_point_normal.push_back(counterclock_point.cross(clock_point));
            }
        }
    }
}
```

이를 각 Face 객체에 호출하면 된다.

3. Boundary Curve, Control Point of Surface

각 Face에 대해 탐색해 모든 Vertex의 Point normal을 할당했다면, 이를 통해 Face의 Control Point를 구할 수 있다.

```
Vector3d calculate_near_controlpoint(shared_ptr<Vertex> near, shared_ptr<Vertex> far){
    Vector3d projected_point = 1.0/3.0 * (vertex_to_vec3(far) - vertex_to_vec3(near));
    Vector3d point_normal_nomalized = near->point_normal.normalized(); // point normal의
단위벡터화
    Vector3d reproject_projected_point = projected_point -
point_normal_nomalized.dot(projected_point) * point_normal_nomalized; // point normal을
법선으로 갖는 평면에 정사영 때린 벡터
    double multiply = projected_point.norm()/reproject_projected_point.norm(); // 배율
구하고
    Vector3d controlpoint = vertex_to_vec3(near) + multiply * multiply *
reproject_projected_point; // 시컨트 두번 때림
    cout << "x : " << controlpoint.x() << ", y : " << controlpoint.y() << ", z = " <<
controlpoint.z() << endl;
    return controlpoint;
}
```

Utils 내부에 구성한 메서드이며, 이를 통해 boundary curve의 vertex 2개를 지정해, near에 가까운 ControlPoint를 구한다. 모든 Face 내부의 Vertex에 순서에 맞게 호출하여, Controlpoint를 반환할 수 있다.

이제 Triangle과 Rectangle 두 가지의 경우에 나눠서 새로 메서드를 구성하였다.

```
void triangle_coons_patch(shared_ptr<Face>& ptr_face){
    for(shared_ptr<Vertex> inner_vertex : ptr_face->vertices){ // vertices의 normal
vector 계산
        inner_vertex->calculate_ponit_normal();
    }
    vector<Vector3d> temp;
    temp.push_back(Vector3d(ptr_face->vertices[0]->x, ptr_face->vertices[0]->y,
ptr_face->vertices[0]->z)); // 첫 번째 control point는 face의 첫 vertex
    temp.push_back(calculate_near_controlpoint(ptr_face->vertices[0], ptr_face-
>vertices[1]));
    temp.push_back(calculate_near_controlpoint(ptr_face->vertices[1], ptr_face-
>vertices[0]));
    temp.push_back(Vector3d(ptr_face->vertices[1]->x, ptr_face->vertices[1]->y,
ptr_face->vertices[1]->z)); // 4번째 control point는 face의 두 번째 vertex
    temp.push_back(calculate_near_controlpoint(ptr_face->vertices[1], ptr_face-
>vertices[2]));
    temp.push_back(calculate_near_controlpoint(ptr_face->vertices[2], ptr_face-
>vertices[1]));
    temp.push_back(Vector3d(ptr_face->vertices[2]->x, ptr_face->vertices[2]->y,
ptr_face->vertices[2]->z)); // 7번째 control point는 face의 세 번째 vertex
    temp.push_back(calculate_near_controlpoint(ptr_face->vertices[2], ptr_face-
>vertices[0]));
    temp.push_back(calculate_near_controlpoint(ptr_face->vertices[0], ptr_face-
>vertices[2]));
    // 외곽의 control point 계산 완료

    Vector3d middle_controlpoint = (1.0/2)*((1.0/2) *
(temp[1]+temp[2]+temp[4]+temp[5]+temp[7]+temp[8]) - (1.0/3) *
(temp[0]+temp[3]+temp[6]));
    // 내부 control point 계산 완료

    ptr_face->controlpoints.push_back(temp[0]);
    ptr_face->controlpoints.push_back(temp[8]);
    ptr_face->controlpoints.push_back(temp[1]);
```

```
    ptr_face->controlpoints.push_back(temp[7]);
    ptr_face->controlpoints.push_back(middle_controlpoint);
    ptr_face->controlpoints.push_back(temp[2]);
    ptr_face->controlpoints.push_back(temp[6]);
    ptr_face->controlpoints.push_back(temp[5]);
    ptr_face->controlpoints.push_back(temp[4]);
    ptr_face->controlpoints.push_back(temp[3]);

//                0
//            1       2
//        3       4       5
//    6       7       8       9

}
```

마지막의 주석의 순서와 같이 Face member variable인 controlpoint에 할당한다. Middle_controlpoint 계산 과정은 PPT의 수식을 따랐다.

같은 방식으로 Rectangle 또한 구현하였다.

```
     void rectangular_coons_patch(shared_ptr<Face>& ptr_face){
    for(shared_ptr<Vertex> inner_vertex : ptr_face->vertices){ // vertices의 normal
vector 계산
        inner_vertex->calculate_ponit_normal();
    }
    vector<Vector3d> temp;
    temp.push_back(Vector3d(ptr_face->vertices[0]->x, ptr_face->vertices[0]->y,
ptr_face->vertices[0]->z)); // 첫 번째 control point는 face의 첫 vertex
    temp.push_back(calculate_near_controlpoint(ptr_face->vertices[0], ptr_face-
>vertices[1]));
    temp.push_back(calculate_near_controlpoint(ptr_face->vertices[1], ptr_face-
>vertices[0]));
    temp.push_back(Vector3d(ptr_face->vertices[1]->x, ptr_face->vertices[1]->y,
ptr_face->vertices[1]->z)); // 4번째 control point는 face의 두 번째 vertex
    temp.push_back(calculate_near_controlpoint(ptr_face->vertices[1], ptr_face-
>vertices[2]));
    temp.push_back(calculate_near_controlpoint(ptr_face->vertices[2], ptr_face-
>vertices[1]));
    temp.push_back(Vector3d(ptr_face->vertices[2]->x, ptr_face->vertices[2]->y,
ptr_face->vertices[2]->z)); // 7번째 control point는 face의 세 번째 vertex
    temp.push_back(calculate_near_controlpoint(ptr_face->vertices[2], ptr_face-
>vertices[3]));
    temp.push_back(calculate_near_controlpoint(ptr_face->vertices[3], ptr_face-
>vertices[2]));
    temp.push_back(Vector3d(ptr_face->vertices[3]->x, ptr_face->vertices[3]->y,
ptr_face->vertices[3]->z)); // 10번째 control point는 face의 네 번째 vertex
    temp.push_back(calculate_near_controlpoint(ptr_face->vertices[3], ptr_face-
>vertices[0]));
    temp.push_back(calculate_near_controlpoint(ptr_face->vertices[0], ptr_face-
>vertices[3]));
    // 외곽의 control point 계산 완료

    Vector3d middle_controlpoint11 = (1.0/3)*(2*temp[1]+temp[8]+2*temp[11]+temp[4])-
(1.0/9)*(4*temp[0]+2*temp[3]+2*temp[9]+temp[6]);
    Vector3d middle_controlpoint12 = (1.0/3)*(2*temp[2]+temp[7]+2*temp[4]+temp[11])-
(1.0/9)*(4*temp[3]+2*temp[6]+2*temp[0]+temp[9]);
    Vector3d middle_controlpoint22 = (1.0/3)*(2*temp[7]+temp[2]+2*temp[5]+temp[10])-
(1.0/9)*(4*temp[6]+2*temp[9]+2*temp[3]+temp[0]);
    Vector3d middle_controlpoint21 = (1.0/3)*(2*temp[8]+temp[1]+2*temp[10]+temp[5])-
(1.0/9)*(4*temp[9]+2*temp[0]+2*temp[6]+temp[3]);
    // 내부 control point 계산 완료

    ptr_face->controlpoints.push_back(temp[0]);
    ptr_face->controlpoints.push_back(temp[11]);
```

```
    ptr_face->controlpoints.push_back(temp[10]);
    ptr_face->controlpoints.push_back(temp[9]);
    ptr_face->controlpoints.push_back(temp[1]);
    ptr_face->controlpoints.push_back(middle_controlpoint11);
    ptr_face->controlpoints.push_back(middle_controlpoint12);
    ptr_face->controlpoints.push_back(temp[8]);
    ptr_face->controlpoints.push_back(temp[2]);
    ptr_face->controlpoints.push_back(middle_controlpoint21);
    ptr_face->controlpoints.push_back(middle_controlpoint22);
    ptr_face->controlpoints.push_back(temp[7]);
    ptr_face->controlpoints.push_back(temp[3]);
    ptr_face->controlpoints.push_back(temp[4]);
    ptr_face->controlpoints.push_back(temp[5]);
    ptr_face->controlpoints.push_back(temp[6]);

//    30-31-32-33       12-13-14-15
//    20-21-22-23       8-9-10-11
//    10-11-12-13       4-5-6-7
//    00-01-02-03       0-1-2-3
//    이 순서로 입력해줌

}
```

마지막 주석의 순서를 따른다, 이는 이후 적용할 De Casteljau Algorithm을 수월히 적용하고자 주석의 배치를 따랐다.

4. Find POS and visualize Mesh

저번 과제와는 달리, 이번에는 Face 객체를 통해 POS를 생성하고, 이를 통해 바로 Mesh를 추가하고자 한다.

먼저 De Casteljau Algorithm을 살펴보자.

```
Vector3d deCasteljau_triangle(shared_ptr<Face> face, double u, double v){
    // 첫번째 과정
    Vector3d first = deCasteljau_for_triangle(face->controlpoints[0], face-
>controlpoints[1], face->controlpoints[2], u, v);
    Vector3d second = deCasteljau_for_triangle(face->controlpoints[1], face-
>controlpoints[3], face->controlpoints[4], u, v);
    Vector3d third = deCasteljau_for_triangle(face->controlpoints[2], face-
>controlpoints[4], face->controlpoints[5], u, v);
    Vector3d fourth = deCasteljau_for_triangle(face->controlpoints[3], face-
>controlpoints[6], face->controlpoints[7], u, v);
    Vector3d fifth = deCasteljau_for_triangle(face->controlpoints[4], face-
>controlpoints[7], face->controlpoints[8], u, v);
    Vector3d sixth = deCasteljau_for_triangle(face->controlpoints[5], face-
>controlpoints[8], face->controlpoints[9], u, v);
    // 두번째 과정
    Vector3d second_first = deCasteljau_for_triangle(first, second, third, u, v);
    Vector3d second_second = deCasteljau_for_triangle(second, fourth, fifth, u, v);
    Vector3d second_third = deCasteljau_for_triangle(third, fifth, sixth, u, v);
    // 마지막
    return deCasteljau_for_triangle(second_first, second_second, second_third, u, v);
}

Vector3d deCasteljau_for_triangle(Vector3d cp0, Vector3d cp1, Vector3d cp2, double u,
double v){
    return (1.0-u-v) * cp0 + u * cp1 + v * cp2;
```

```
}

Vector3d deCasteljau_rectangle(shared_ptr<Face> face, double u, double v){
    Vector3d first = deCasteljau(face->controlpoints[0], face->controlpoints[1], face->controlpoints[2], face->controlpoints[3], u);
    Vector3d second = deCasteljau(face->controlpoints[4], face->controlpoints[5], face->controlpoints[6], face->controlpoints[7], u);
    Vector3d third = deCasteljau(face->controlpoints[8], face->controlpoints[9], face->controlpoints[10], face->controlpoints[11], u);
    Vector3d fourth = deCasteljau(face->controlpoints[12], face->controlpoints[13], face->controlpoints[14], face->controlpoints[15], u);

    return deCasteljau(first, second, third, fourth, v);
}

Vector3d deCasteljau(Vector3d cp0, Vector3d cp1, Vector3d cp2, Vector3d cp3, double t){
    Vector3d a = cp0 + t * (cp1 - cp0);
    Vector3d b = cp1 + t * (cp2 - cp1);
    Vector3d c = cp2 + t * (cp3 - cp2);

    Vector3d d = a + t * (b - a);
    Vector3d e = b + t * (c - b);

    Vector3d pointOnCurve = d + t * (e - d);

    return pointOnCurve;
}
```

De Casteljau Algorithm은 Triangle과 Rectangle의 경우가 다른데, Rectangle의 경우는 1.Coons-patch에 설명했던 대로 기존의 De Casteljau를 따르고, Triangle은 Barycentric Coordinate를 따라 찾을 수 있다.

이를 통해 Face의 POS를 구할 수 있는데, 면을 1/n으로 나눌 수 있고, 기존 face가 약 1200개였기 때문에 대부분이 Rectangle인 Face이기 때문에 약 $(n+1)^2$ 배로 늘어난다. 실제로 n을 10으로 잡았을 경우, RAM이 오버되어 약 15만개 이상의 mesh일때부터 극도로 느려지는 현상을 겪었다. 따라서 5분할 기준으로 진행했다.

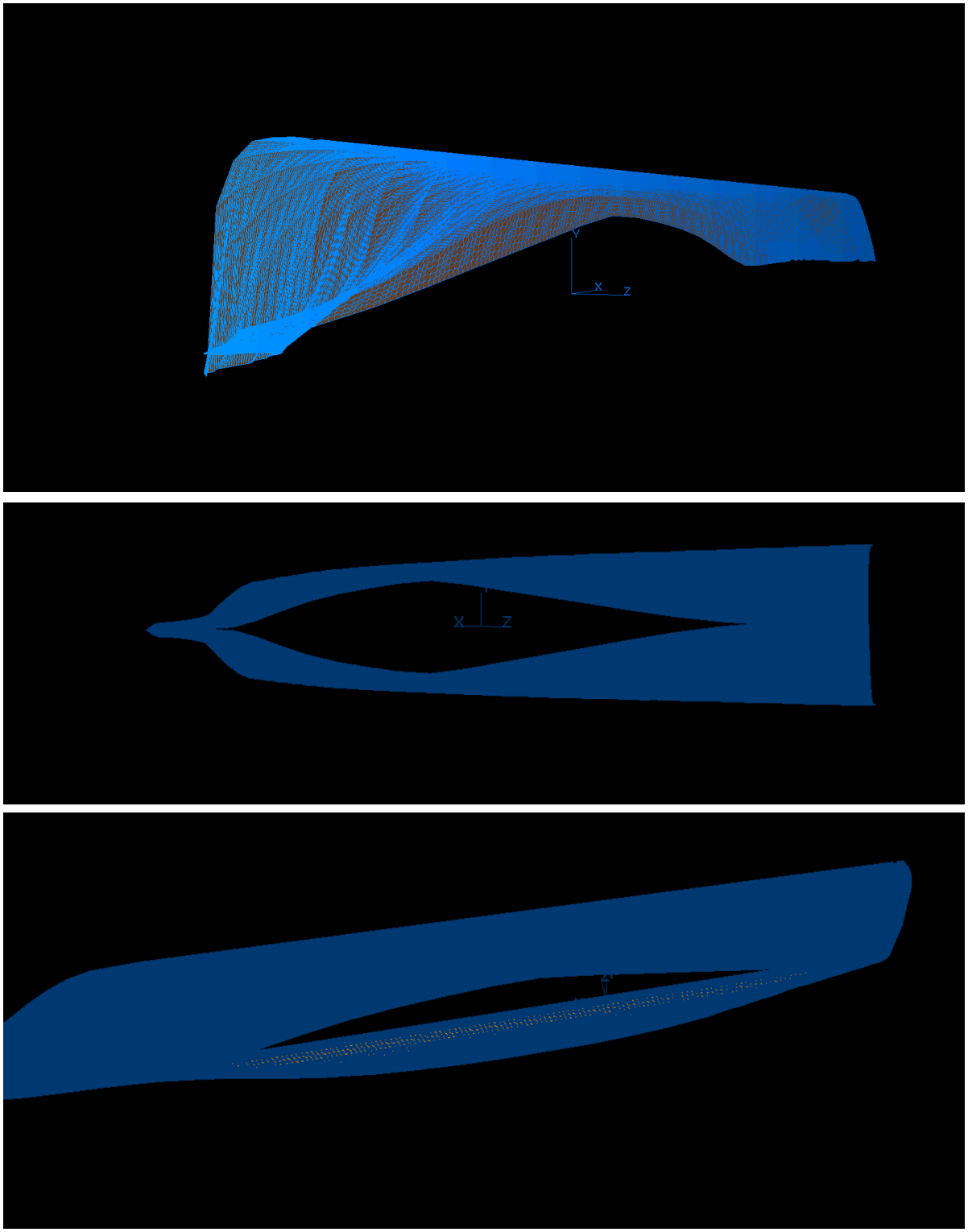
```
void build_triangle_mesh(shared_ptr<Face> ptr_face){
    int step_size = 5;
    double step = 1.0/step_size;
    vector<vector<Vector3d>> triangle_points;
    for(int i = 0; i <= step_size; i++){
        vector<Vector3d> temp;
        for(int j = 0; j <= i; j++){
            // 이 행에서 u, v의 합은 (1/step_size) * i, triangle points에 삼각형 모양으로 데이터 입력
//                          00
//                     10       11
//                20       21       22
//           30       31       32       33
            temp.push_back(deCasteljau_triangle(ptr_face, step*(i-j), step*j));
            cout << i << ", " << j << " : " ;
            cout << deCasteljau_triangle(ptr_face, step*(i-j), step*j).x() << ", ";
            cout << deCasteljau_triangle(ptr_face, step*(i-j), step*j).y() << ", ";
            cout << deCasteljau_triangle(ptr_face, step*(i-j), step*j).z() << ", " << endl;
        }
        triangle_points.push_back(temp);
    }
    //  n층의 피라미드 형식으로 pos 찾기 완료
```

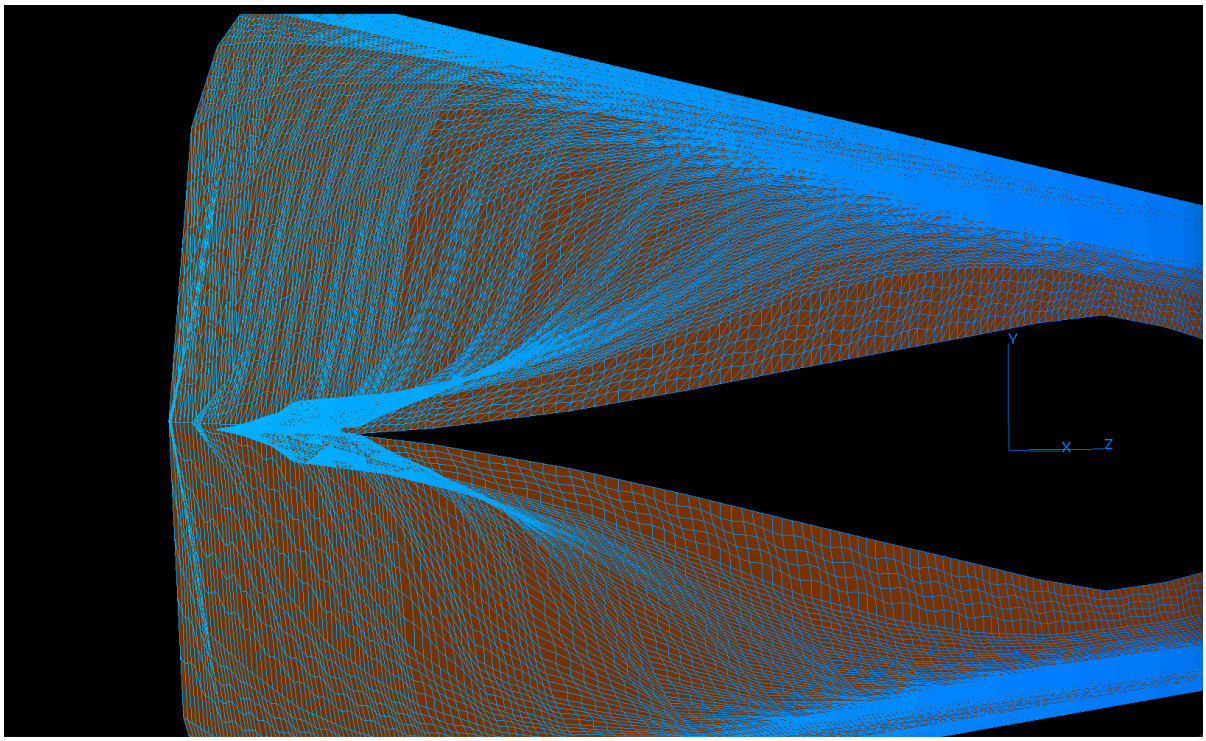
```
    for(int i = 0; i < step_size; i++){
        for(int j = 0; j < triangle_points[i].size(); j++){
            vector<Vector3d> temp;
            temp.push_back(triangle_points[i][j]);
            temp.push_back(triangle_points[i+1][j+1]);
            temp.push_back(triangle_points[i+1][j]);
            mesh.push_back(temp);
            // 정방향 삼각형 mesh 입력, 시계방향
        }
    }
    for(int i = 1; i < step_size; i++){
        for(int j = 0; j < triangle_points[i].size()-1; j++){
            vector<Vector3d> temp;
            temp.push_back(triangle_points[i][j]);
            temp.push_back(triangle_points[i][j+1]);
            temp.push_back(triangle_points[i+1][j+1]);
            mesh.push_back(temp);
            // 역삼각형 mesh 입력, 시계방향
        }
    }
}

void build_rectangle_mesh(shared_ptr<Face> ptr_face){
    int step_size = 5;
    double step = 1.0/step_size;
    vector<vector<Vector3d>> rectangle_points;
    for(int i = 0; i <= step_size; i++){
        vector<Vector3d> temp;
        for(int j = 0; j <= step_size; j++){

            // 바둑판 모양
//            00 01 02 03
//            10 11 12 13
//            20 21 22 23
//            30 31 32 33
            temp.push_back(deCasteljau_rectangle(ptr_face, i*step, j*step));
        }
        rectangle_points.push_back(temp);
    }

    for(int i = 0; i < step_size; i++){
        for(int j = 0; j < step_size; j++){
            mesh.push_back({rectangle_points[i][j], rectangle_points[i][j+1],
rectangle_points[i+1][j+1]});
            mesh.push_back({rectangle_points[i+1][j+1], rectangle_points[i+1][j],
rectangle_points[i][j]});
            // 바둑판의 좌상귀 잡듯이 점 선택해서 시계방향
        }
    }
}

void build_coons_mesh(vector<shared_ptr<Face>> ptr_faces){
    for(shared_ptr<Face> ptr_face : ptr_faces){
        if(ptr_face->vertices.size() == 3){ //삼각형
            build_triangle_mesh(ptr_face);
        }
        else if(ptr_face->vertices.size() == 4){ // 사각형
            build_rectangle_mesh(ptr_face);
        }
    }
}
```

내부 각주 형태로 데이터를 저장하였고, 여기서 중요한 점은 Triangle의 경우는 원활한 계산을 위해 0번째 벡터에 1개, 1번째 벡터에 2개 식으로 데이터를 피라미드식으로 저장하였다. 이는 아래 단계에서 반복문을 사용해 mesh를 추가할 때, 정방향 삼각형과 역삼각형을 입력하기 위해서이다.

이후 메인 함수에서 build_coons_mesh를 실행시키면 다음과 같은 결과를 얻는다.

5. Visualization





6. Reference

1. 쉽게 배우는 자료구조 with JAVA(문병로, 2022)

2. Computer Aid Ship Design Lecture Note(김태완, 2023)