(기말고사 대체 프로젝트)

컴퓨터 그래픽스

- SOR Modeling data를 이용한 가상 공간 렌더링 -

중앙대학교 예술공학대학 컴퓨터예술학부

20194004

양소영

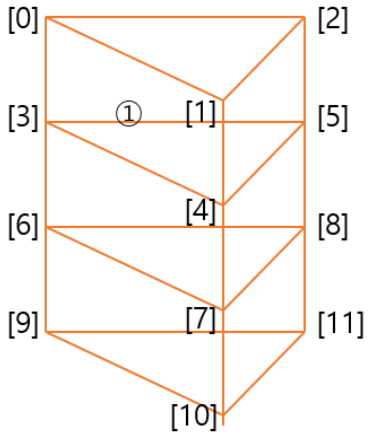
1. 문제 내용 및 해결 방안

본격적으로 스크립트를 짜기에 앞서, 다음 순으로 단계를 나누어 차근차근 구현해보기로 하였다.

중간고사 코드 수정> SOR 모델링 데이터 저장> 바닥 추가> BackFace Culling 기능 추가> Smooth shading 구현> 투영방식 변환> 카메라 회전 및 시점 변환> 기타 기능 구현

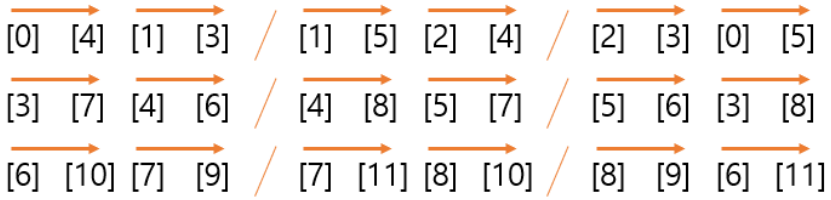
(1) 중간고사 코드 수정

기존 코드에서는 삼각형의 wireframe 을 그리기 위해 cross 라는 함수를 정의하여 모델 예시 [그림 1]의 각 면에 사선을 그려주었다.



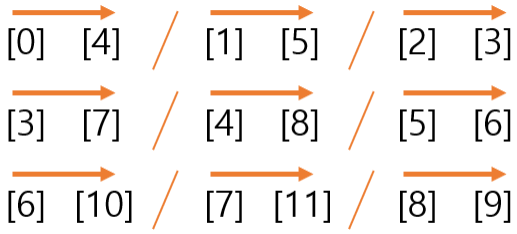
[그림 1]

cross 함수는 ①과 같은 면 마다 'X'자로 선을 그려주었는데, [그림 2]에서 보이는 것처럼 line 을 각각 그려주었다.



[그림 2]

하지만 SOR 모델링 데이터를 저장하기 위해서는 한 면 당 사선이 한 개만 필요했으며, [그림 3]과 같이 좌하향 사선을 지워야 했다. 따라서 cross 함수를 수정하는 과정을 거쳤다.



[그림 3]

```
void cross(int c) {
    glColor3f(1.0, 1.0, 0.0);
    glBegin(GL_LINE_STRIP);  // wireframe 삼각형 그리기
    for (int i = 0; i < (sweepResolutionMod - 1); i++) {
        glVertex3f(arRotPoints[c * sweepResolutionMod + i].x, arRotPoints[c * sweepResolutionMod + i].y, arRotPoints[c * sweepResolutionMod + i].z);
        glVertex3f(arRotPoints[c * sweepResolutionMod + (sweepResolutionMod + 1) + i].x, arRotPoints[c * sweepResolutionMod + (sweepResolutionMod + 1) + i].y,
            arRotPoints[c * sweepResolutionMod + (sweepResolutionMod + 1) + i].z);
    }
    glVertex3f(arRotPoints[c * sweepResolutionMod + (sweepResolutionMod - 1)].x, arRotPoints[c * sweepResolutionMod + (sweepResolutionMod - 1)].y,
        arRotPoints[c * sweepResolutionMod + (sweepResolutionMod - 1)].z);
    glVertex3f(arRotPoints[c * sweepResolutionMod + sweepResolutionMod].x, arRotPoints[c * sweepResolutionMod + sweepResolutionMod].y,
        arRotPoints[c * sweepResolutionMod + sweepResolutionMod].z);
    glVertex3f(arRotPoints[c * sweepResolutionMod].x, arRotPoints[c * sweepResolutionMod].y, arRotPoints[c * sweepResolutionMod].z);

    glEnd();
    glFlush();
}
```

(2) SOR 모델링 데이터 저장

모델링 데이터를 저장하기 위해서는 저장하는 함수가 따로 필요하였다.

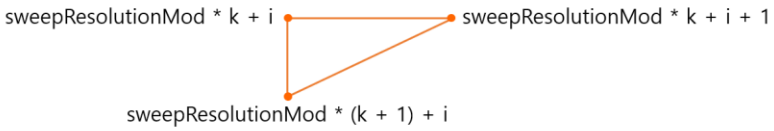
```
void SaveModel() {

    FILE* fout;

    fout = fopen("c:\\data\\myModel.dat", "w");

    fprintf(fout, "VERTEX = %d\n", pnum);
    for (int i = 0; i < pnum; i++) {
        fprintf(fout, "%.1f %.1f %.1f\n", mpoint[i].x, mpoint[i].y, mpoint[i].z);
    }

    fprintf(fout, "FACE = %d\n", fnum);
    for (int i = 0; i < fnum; i++) {
        fprintf(fout, "%d %d %d\n", mface[i].ip[0], mface[i].ip[1], mface[i].ip[2]);
    }
    fclose(fout);
}
```

교안을 참고하여 SaveModel 이라는 함수를 정의해주었고, 여기서 mpoint, mface 라는 배열을 만들어 해당하는 값을 각각 삽입할 필요가 있었다.

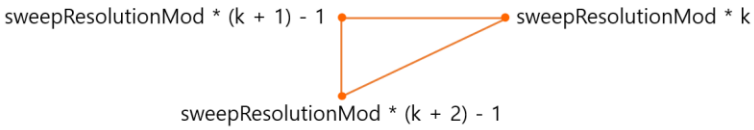
```
    pnum = arRotPoints.size();  // point 개수 지정
    fnum = sweepResolutionMod * (arInputPoints.size() - 1) * 2;  // face 개수 지정
    mpoint = new Point[pnum];
    mface = new Face[fnum];

    for (int i = 0; i < pnum; i++) {
        mpoint[i].x = arRotPoints[i].x * 0.7;
        mpoint[i].y = arRotPoints[i].y * 0.7;
        mpoint[i].z = arRotPoints[i].z * 0.7;
    }

    for (int k = 0; k < arInputPoints.size() - 1; k++) {
        for (int i = 0; i < sweepResolutionMod - 1; i++) {  // up-triangle
            mface[cnum].ip[1] = sweepResolutionMod * k + i + 1;
            mface[cnum].ip[2] = sweepResolutionMod * k + i;
            mface[cnum].ip[0] = sweepResolutionMod * (k + 1) + i;
            cnum += 1;
        }
        mface[cnum].ip[1] = sweepResolutionMod * k;
        mface[cnum].ip[2] = sweepResolutionMod * (k + 1) - 1;
        mface[cnum].ip[0] = sweepResolutionMod * (k + 2) - 1;
        cnum += 1;

        for (int i = 0; i < sweepResolutionMod - 1; i++) {  // down-triangle
            mface[cnum].ip[0] = sweepResolutionMod * k + i + 1;
            mface[cnum].ip[1] = sweepResolutionMod * (k + 1) + i;
            mface[cnum].ip[2] = sweepResolutionMod * (k + 1) + i + 1;
            cnum += 1;
        }
        mface[cnum].ip[0] = sweepResolutionMod * (k);
        mface[cnum].ip[1] = sweepResolutionMod * (k + 2) - 1;
        mface[cnum].ip[2] = sweepResolutionMod * (k + 1);
        cnum += 1;
    }
    SaveModel();
}
```

pnum 은 point 배열의 개수로 지정해야 하기 때문에 입력 받은 모든 점의 개수로 정의하였고, fnum 은 face 배열의 개수로 지정해야 하기 때문에 모델을 이루고 있는 사각형 면의 개수에 2 배를 하여 정의해주었다. 이렇게 각 배열을 선언해준 후, mpoint 배열에는 입력 받은 모든 점 값을 담고 있는 arRotPoints 의 배열을 넣어주었으며 mface 배열에는 삼각형 면을 이루고 있는 mpoint 의 순서를 3 개씩 담아야 했는데 이 알고리즘을 생각하는 과정이 꽤 까다로웠다. 나의 경우는 각 층을 이루고 있는 삼각형 모양의 면을 [그림 4], [그림 5]와 같이 위 삼각형과 아래 삼각형으로 나누어 생각해보았다.

k층에 생기는 삼각형

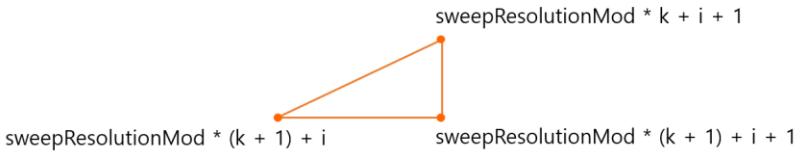


k층에 생기는 마지막 삼각형

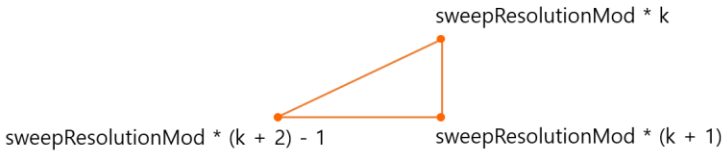


[그림 4] 위 삼각형

k층에 생기는 삼각형



k층에 생기는 마지막 삼각형



[그림 5] 아래 삼각형

각 층마다 삼각형을 이루는 각 point 가 반복되지만, 층의 마지막 삼각형은 처음 삼각형과 이어져야 하기 때문에 반복문에 포함시키지 않고 따로 입력해주어야 한다. 이렇게 코드를 구현하고 프로그램을 실행시키면 점을 찍는 화면이 나타나고, 각도를 입력하여 회전시키면 바로 c:₩₩data 폴더에 myModel.dat 파일이 저장된다.

(3) 바닥 추가

바닥 추가를 구현하는 것은 8 번째 교안 2 번 예시를 참고하여 코드를 작성하였다. 가로, 세로는 (400, 400) 길이가 되도록 만들어주었고 DrawWire 과 DrawShade 함수에 각각 넣어주었다.

```
glColor3f(0.7, 0.7, 0.7);
glPushMatrix();
glTranslatef(0, -100, 0);
glBegin(GL_QUADS);
glVertex3f(200, 0, 200);
glVertex3f(200, 0, -200);
glVertex3f(-200, 0, -200);
glVertex3f(-200, 0, 200);
glEnd();
glPopMatrix();
glutSwapBuffers();
```

(4) BackFace Culling 기능 추가

처음에 모델링 데이터를 불러오면 wireframe 모드로 보여지는데, 이 때의 모델링은 뒷면까지 보이는 상태이다. 여기서 glEnable 함수를 사용하면 제거할 부분의 면을 선택할 수 있다. glEnable(GL_CULL_FACE) 함수를 불러 면 제거를 가능하도록 변경해주고, glCullFace(GL_BACK)을 넣어주면 모델의 뒷면을 제거할 수 있다. 키보드 'e'를 눌렀을 때 BackFace Culling 기능을 On 시킬 수 있다. 만약 이 기능을 다시 Off 시켜주고 싶다면 키보드 'd'를 누르면 된다. 구글링을 하다가 우연히 glDisable 이라는 함수를 발견하게 되었는데, 뒷면을 제거한 상태에서 GL_CULL_FACE 기능을 가능하지 못하게 하면 제거되었던 뒷면이 다시 보이게 된다.

```
case'e':  // hide back
    glEnable(GL_CULL_FACE);
    glCullFace(GL_BACK);
    glutPostRedisplay();
    break;
case'd':  // show back
    glDisable(GL_CULL_FACE);
    glutPostRedisplay();
    break;
```

(5) Smooth shading 구현

조명을 비출 때 빛이 모델링의 각 면에서 반사되어 부드럽게 보이려면 한 vertex 를 기준으로 그 점에 인접해 있는 모든 면의 법선 벡터를 더해서 평균을 구한 뒤, 조명이 해당 평균 벡터로부터 빛을 반사해야 한다. 처음에는 Gouraud shading 을 구현하기 위해 다음과 같이 코드를 작성하여 평균 벡터를 모두 구하려는 시도를 하였다.

```
Point newnormal(Point a) {  // smooth shading
    Point n;

    n.x = 0;
    n.y = 0;
    n.z = 0;

    for (int i = 0; i < pnum; i++) {
        int n_count = 0;
        for (int j = 0; j < fnum; j++) {
            if (i == mface[j].ip[0] || i == mface[j].ip[1] || i == mface[j].ip[2]) {
                a.x += a.x;
                a.y += a.y;
                a.z += a.z;
                n_count++;
            }
        }
        n.x = a.x / n_count;
        n.y = a.y / n_count;
        n.z = a.z / n_count;
    }
    return n;
}
```

하지만 작동이 잘 되지 않아 굉장히 많은 시간을 지체하는 시행착오를 겪었으며, 구글링을 하다가 glNormal3f라는 함수가 vertex에 대한 법선 벡터를 정의하는 함수라는 것을 알게 되어 이 함수의 정확한 쓰임새를 알게 되었다.

```
glEnable(GL_NORMALIZE);
for (i = 0; i < fnum; i++) {
    Point norm = mpoint[mface[i].ip[0]];
    glBegin(GL_TRIANGLES);

    glNormal3f(norm.x, norm.y, norm.z);  // 면을 이루는 첫번째 vertex
    glVertex3f(mpoint[mface[i].ip[0]].x, mpoint[mface[i].ip[0]].y, mpoint[mface[i].ip[0]].z);

    norm = mpoint[mface[i].ip[1]];
    glNormal3f(norm.x, norm.y, norm.z);  // 면을 이루는 두번째 vertex
    glVertex3f(mpoint[mface[i].ip[1]].x, mpoint[mface[i].ip[1]].y, mpoint[mface[i].ip[1]].z);

    norm = mpoint[mface[i].ip[2]];
    glNormal3f(norm.x, norm.y, norm.z);  // 면을 이루는 세번째 vertex
    glVertex3f(mpoint[mface[i].ip[2]].x, mpoint[mface[i].ip[2]].y, mpoint[mface[i].ip[2]].z);
    glEnd();
}
glPopMatrix();
glEndList();
```

먼저 glEnable(GL_NORMALIZE)을 정의하여 단위 법선 벡터를 계산할 수 있도록 해주고, for 반복문 바로 다음에 norm 을 한 번만 설정해준 전과는 달리, 면을 이루는 vertex 전마다 각각 norm 을 적용시켜 주었더니 각 vertex 마다 법선 벡터를 지정해주어 빛이 들어오는 방향과 각을 다르게 계산하여 최종적으로 부드러운 느낌의 표면을 구현할 수 있게 되었다.

(6) 투영방식 변환

원근 투영과 직교 투영을 구현해주기 위해 perspmode, orthomode 함수를 정의해주어 투영 방식을 따로 지정해주었다.

```
void perspmode() {  // perspective mode
    glEnable(GL_DEPTH_TEST);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(40.0, 1.0, 1.0, 2000.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    gluLookAt(600.0, 600.0, 600.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.);
}

void orthomode() {  // ortho mode
    glEnable(GL_DEPTH_TEST);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(-100.0, 100.0, -100.0, 100.0, -800.0, 800.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    gluLookAt(600.0, 600.0, 600.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.);
}
```

처음에 모델을 불러올 때는 원근 투영이 되어 있는 상태이다. 이 때 키보드 'f'버튼을 누르면 orthomode 함수를 불러와 모델이 직교 투영되어 있는 것을 볼 수 있고, 키보드 'r'버튼을 누르면 perspmode 함수를 불러와 모델이 다시 원근 투영되어 있는 것을 볼 수 있다.

```
case'r':  // perspective mode
    display();
    perspmode();
    glutPostRedisplay();
    break;
case'f':  // ortho mode
    display();
    orthomode();
    glutPostRedisplay();
    break;
```

(7) 카메라 회전 및 시점 변환

모델을 여러 방향에서 보기 위해 display 함수 안에 있는 gluLookAt 함수의 카메라 위치를 바꾸면 카메라를 회전시킬 수 있었다. 먼저 xmove, ymove, zmove 이라는 전역변수를 선언하고 다음과 같이 키보드를 누를 때마다 원하는 값만큼 증가 혹은 감소시켜 카메라를 이동하도록 하였다. 키보드 버튼을 이용하여 공간을 이동하는 네비게이션 모드를 구현할 수 있었다.

```
case 'b':  // xmoving
    xmove += 10;
    display();
    glutPostRedisplay();
    break;
case 'n':  // ymoving
    ymove += 10;
    display();
    glutPostRedisplay();
    break;
case 'm':  // zmoving
    zmove += 10;
    display();
    glutPostRedisplay();
    break;
case 'h':  // xmoving
    xmove -= 10;
    display();
    glutPostRedisplay();
    break;
case 'j':  // ymoving
    ymove -= 10;
    display();
    glutPostRedisplay();
    break;
case 'k':  // zmoving
    zmove -= 10;
    display();
    glutPostRedisplay();
    break;
```

또한 물체를 바라보는 시점을 x 축, y 축, z 축마다 변경 가능하도록 카메라 위치를 바꿔주었다.

```
case'q':  // x viewport
    xmove = 600;
    ymove = 0;
    zmove = 0;
    down = 0;
    display();
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glutPostRedisplay();
    break;
case'a':  // y viewports
    xmove = 0;
    ymove = 600;
    zmove = 0;
    down = -1;
    display();
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glutPostRedisplay();
    break;
case'z':  // z viewport
    xmove = 0;
    ymove = 0;
    zmove = 600;
    down = 0;
    display();
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glutPostRedisplay();
    break;
```

(8) 기타 기능 구현

모델의 심미적 부분을 향상시키기 위해 색상 변경을 해주고자 하였다. 먼저 colorR, colorG, colorB 라는 전역변수를 선언하고 초기 모델은 검정색으로 지정한 후, 각 키보드 버튼을 누를 때마다 빨간색, 초록색, 파란색으로 변경되도록 해주었다.

```
case'1':  // red color
    colorR = 0.8; colorG = 0.4; colorB = 0.4;
    display();
    glutPostRedisplay();
    break;
case'2':  // green color
    colorR = 0.4; colorG = 0.8; colorB = 0.4;
    display();
    glutPostRedisplay();
    break;
case'3':  // blue color
    colorR = 0.4; colorG = 0.4; colorB = 0.8;
    display();
    glutPostRedisplay();
    break;
```

2. 소스코드

다음은 SOR Modeling data 저장을 구현한 총 소스코드이다. 추후 과제 제출을 위해 상대경로로 저장하도록 스크립트를 수정하였다.

```
#define _USE_MATH_DEFINES
#define _CRT_SECURE_NO_WARNINGS

#include <GL/glut.h>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <vector>
#include <iostream>
using namespace std;

#define WIRE 0
#define SHADE 1

typedef struct {
    float x;
    float y;
    float z;
} Point;

typedef struct {
    unsigned int ip[3];
} Face;

GLsizei winWidth = 1000, winHeight = 700;
GLboolean revolveFcn = false;  // 점 회전
GLboolean clean = false;  // 화면 초기화
GLboolean draw = true;  // 점 찍기
GLboolean Wireframe = false;  // wireframe 모드
GLfloat angle = 0;

int moving;
int mousebegin;
int light_moving;
float scalefactor = 1.0;
int scaling = 0;
int status = 0;

float fRotAngle;  // 입력받은 각도
float radian;
int sweepResolutionMod = (360 / fRotAngle);  // 점이 회전하는 횟수


class xPoint3D {
public:
    float x, y, z, w;
    xPoint3D() { x = y = z = 0; w = 1; };
};

xPoint3D pt;
vector<xPoint3D> arInputPoints;  // 초기에 찍은 점 받기
vector<xPoint3D> arRotPoints;  // 회전한 점 받기

int pnum;
int fnum;
Point* mpoint = NULL;
Face* mface = NULL;
int cnum = 0;  // count number

void line(int l) {
    glColor3f(1.0, 1.0, 0.0);
    glBegin(GL_LINE_STRIP);  // wireframe 행 그리기
    for (int i = 0; i < sweepResolutionMod; i++) {
        glVertex3f(arRotPoints[l + i].x, arRotPoints[l + i].y, arRotPoints[l + i].z);
    }
    glEnd();
    glFlush();
}

void cross(int c) {
    glColor3f(1.0, 1.0, 0.0);
    glBegin(GL_LINE_STRIP);  // wireframe 삼각형 그리기
    for (int i = 0; i < (sweepResolutionMod - 1); i++) {
        glVertex3f(arRotPoints[c * sweepResolutionMod + i].x, arRotPoints[c * sweepResolutionMod + i].y, arRotPoints[c * sweepResolutionMod + i].z);
```

```
            glVertex3f(arRotPoints[c * sweepResolutionMod + (sweepResolutionMod + 1) + i].x, arRotPoints[c * sweepResolutionMod + (sweepResolutionMod + 1) + i].y,
                arRotPoints[c * sweepResolutionMod + (sweepResolutionMod + 1) + i].z);
        }
        glVertex3f(arRotPoints[c * sweepResolutionMod + (sweepResolutionMod - 1)].x, arRotPoints[c * sweepResolutionMod + (sweepResolutionMod - 1)].y,
            arRotPoints[c * sweepResolutionMod + (sweepResolutionMod - 1)].z);
        glVertex3f(arRotPoints[c * sweepResolutionMod + sweepResolutionMod].x, arRotPoints[c * sweepResolutionMod + sweepResolutionMod].y,
            arRotPoints[c * sweepResolutionMod + sweepResolutionMod].z);
        glVertex3f(arRotPoints[c * sweepResolutionMod].x, arRotPoints[c * sweepResolutionMod].y, arRotPoints[c * sweepResolutionMod].z);

        glEnd();
        glFlush();
    }

void SaveModel() {

    FILE* fout;

    fout = fopen("c:\\data\\myModel.dat", "w");

    fprintf(fout, "VERTEX = %d\n", pnum);
    for (int i = 0; i < pnum; i++) {
        fprintf(fout, "%.1f %.1f %.1f\n", mpoint[i].x, mpoint[i].y, mpoint[i].z);
    }

    fprintf(fout, "FACE = %d\n", fnum);
    for (int i = 0; i < fnum; i++) {
        fprintf(fout, "%d %d %d\n", mface[i].ip[0], mface[i].ip[1], mface[i].ip[2]);
    }
    fclose(fout);
}

void MyDisplay() {
    glViewport(0, 0, 1000, 700);
    glColor3f(1.0, 0.0, 0.0);

    glColor3f(0.5, 0.5, 0.5);  // 축 그리기
    glBegin(GL_LINES);
    glVertex3f(500.0, 0.0, 0.0);  // x축
    glVertex3f(-500.0, 0.0, 0.0);
    glVertex3f(0.0, -500.0, 0.0);  // y축
    glVertex3f(0.0, 500.0, 0.0);
    glVertex3f(0.0, 0.0, -500.0);  // z축
    glVertex3f(0.0, 0.0, 500.0);
    glEnd();

    if (revolveFcn) {  // 점 회전

        float fNewAngle = 0;
        radian = fNewAngle * (M_PI / 180.0);

        for (int i = 0; i < arInputPoints.size(); i++) {
            for (int k = 0; k < sweepResolutionMod; k++) {
                xPoint3D newpt;
                newpt.x = arInputPoints[i].x * cos(radian) + arInputPoints[i].z * sin(radian);
                newpt.y = arInputPoints[i].y;
                newpt.z = -arInputPoints[i].x * sin(radian) + arInputPoints[i].z * cos(radian);

                arRotPoints.push_back(newpt);

                cout << "(" << newpt.x << ", " << newpt.y << ", " << newpt.z << ")\n";

                glColor3f(1.0, 1.0, 0.0);
                glPointSize(5.0);
                glBegin(GL_POINTS);
                glVertex3f(newpt.x, newpt.y, newpt.z);
                glEnd();
                glFlush();

                fNewAngle += fRotAngle;  // 매 반복마다 새로운 각도 업데이트
                radian = fNewAngle * (M_PI / 180.);  // 매 반복마다 라디안 업데이트
            }
        }
```

```
147
148             pnum = arRotPoints.size();  // point 개수 지정
149             fnum = sweepResolutionMod * (arInputPoints.size() - 1) * 2;  // face 개수 지정
150             mpoint = new Point[pnum];
151             mface = new Face[fnum];
152
153             for (int i = 0; i < pnum; i++) {
154                 mpoint[i].x = arRotPoints[i].x * 0.7;
155                 mpoint[i].y = arRotPoints[i].y * 0.7;
156                 mpoint[i].z = arRotPoints[i].z * 0.7;
157             }
158
159             for (int k = 0; k < arInputPoints.size() - 1; k++) {
160                 for (int i = 0; i < sweepResolutionMod - 1; i++) {  // up-triangle
161                     mface[cnum].ip[1] = sweepResolutionMod * k + i + 1;
162                     mface[cnum].ip[2] = sweepResolutionMod * k + i;
163                     mface[cnum].ip[0] = sweepResolutionMod * (k + 1) + i;
164                     cnum += 1;
165                 }
166                 mface[cnum].ip[1] = sweepResolutionMod * k;
167                 mface[cnum].ip[2] = sweepResolutionMod * (k + 1) - 1;
168                 mface[cnum].ip[0] = sweepResolutionMod * (k + 2) - 1;
169                 cnum += 1;
170
171                 for (int i = 0; i < sweepResolutionMod - 1; i++) {  // down-triangle
172                     mface[cnum].ip[0] = sweepResolutionMod * k + i + 1;
173                     mface[cnum].ip[1] = sweepResolutionMod * (k + 1) + i;
174                     mface[cnum].ip[2] = sweepResolutionMod * (k + 1) + i + 1;
175                     cnum += 1;
176                 }
177                 mface[cnum].ip[0] = sweepResolutionMod * (k);
178                 mface[cnum].ip[1] = sweepResolutionMod * (k + 2) - 1;
179                 mface[cnum].ip[2] = sweepResolutionMod * (k + 1);
180                 cnum += 1;
181             }
182             SaveModel();
183         }
184
185         if (clean) {  // 화면 초기화
186             arInputPoints.clear();
187             arRotPoints.clear();
188             glClearColor(0.0, 0.0, 0.0, 1.0);
189             glClear(GL_COLOR_BUFFER_BIT);
190             glColor3f(0.5, 0.5, 0.5);  // 축 그리기
191             glBegin(GL_LINES);
192             glVertex3f(500.0, 0.0, 0.0);  // x축
193             glVertex3f(-500.0, 0.0, 0.0);
194             glVertex3f(0.0, -500.0, 0.0);  // y축
195             glVertex3f(0.0, 500.0, 0.0);
196             glVertex3f(0.0, 0.0, -500.0);  // z축
197             glVertex3f(0.0, 0.0, 500.0);
198             glEnd();
199             glFlush();
200         }
201
202         if (Wireframe) {  // wireframe 모드
203             for (int i = 0; i < arInputPoints.size(); i++) {  // arInputPoints 갯수만큼 행 그리기
204                 line(i * sweepResolutionMod);
205             }
206
207             for (int i = 0; i < (arInputPoints.size() - 1); i++) {  // arInputPoints 갯수 - 1만큼 행 그리기
208                 cross(i);
209             }
210         }
211     }
212
213
214     void Drawing(GLint button, GLint action, GLint xMouse, GLint yMouse) {
215
216         if (button == GLUT_LEFT_BUTTON && action == GLUT_DOWN && draw == true) {  // 마우스 위치 입력 및 점 찍기
217             pt.x = xMouse - winWidth / 2;
218             pt.y = -yMouse + winHeight / 2;
219             pt.z = 0.0;
```

```
            arInputPoints.push_back(pt);
            cout << "(" << pt.x << ", " << pt.y << ", " << pt.z << ")\n";

            glColor3f(1.0, 1.0, 0.0);
            glPointSize(5.0);
            glBegin(GL_POINTS);
            glVertex3f(pt.x, pt.y, pt.z);
            glEnd();
            glFlush();
        }
        glutPostRedisplay();
        glutSwapBuffers();
    }


void MyMainMenu(int entryID) {
    if (entryID == 1) {  // 모든 점 지우기
        revolveFcn = false;
        clean = true;
        Wireframe = false;
        MyDisplay();
        clean = false;
        draw = true;
    }

    else if (entryID == 2) {  // wireframe 모드
        Wireframe = true;
        draw = false;
        revolveFcn = false;
    }
    glutPostRedisplay();
}


void MySubMenu(int entryID) {
    if (entryID == 1) {  // 30도 회전
        cout << "(" << pt.x << ", " << pt.y << ", 0.0) by 30 radians: \n ";
        fRotAngle = 30;
        radian = fRotAngle * (M_PI / 180.);
        sweepResolutionMod = 360 / fRotAngle;
    }

    else if (entryID == 2) {  // 60도 회전
        cout << "(" << pt.x << ", " << pt.y << ", 0.0) by 60 radians: \n ";
        fRotAngle = 60;
        radian = fRotAngle * (M_PI / 180.);
        sweepResolutionMod = 360 / fRotAngle;
    }

    else if (entryID == 3) {  // 90도 회전
        cout << "(" << pt.x << ", " << pt.y << ", 0.0) by 90 radians: \n ";
        fRotAngle = 90;
        radian = fRotAngle * (M_PI / 180.);
        sweepResolutionMod = 360 / fRotAngle;
    }

    else if (entryID == 4) {  // 120도 회전
        cout << "(" << pt.x << ", " << pt.y << ", 0.0) by 120 radians: \n ";
        fRotAngle = 120;
        radian = fRotAngle * (M_PI / 180.);
        sweepResolutionMod = 360 / fRotAngle;
    }

    else if (entryID == 5) {  // 180도 회전
        cout << "(" << pt.x << ", " << pt.y << ", 0.0) by 180 radians: \n ";
        fRotAngle = 180;
        radian = fRotAngle * (M_PI / 180.);
        sweepResolutionMod = 360 / fRotAngle;
    }
    draw = false;
    revolveFcn = true;
    glutPostRedisplay();

}
int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_RGB);
    glutInitWindowSize(winWidth, winHeight);
    glutInitWindowPosition(0, 0);
    glutCreateWindow("YANGSOYEONG");

    glClearColor(0.0, 0.0, 0.0, 1.0);
    glClear(GL_COLOR_BUFFER_BIT);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(-500.0, 500.0, -350.0, 350.0, -500.0, 500.0);

    GLint MySubMenuID = glutCreateMenu(MySubMenu);
    glutAddMenuEntry("30", 1);
    glutAddMenuEntry("60", 2);
    glutAddMenuEntry("90", 3);
    glutAddMenuEntry("120", 4);
    glutAddMenuEntry("180", 5);

    GLint MyMainMenuID = glutCreateMenu(MyMainMenu);
    glutAddSubMenu("Revolve angle", MySubMenuID);
    glutAddMenuEntry("Remove All", 1);
    glutAddMenuEntry("Wireframe Mode", 2);
    glutAttachMenu(GLUT_RIGHT_BUTTON);
    glutDisplayFunc(MyDisplay);
    glutMouseFunc(Drawing);
    glutMainLoop();
    return 0;
}
```

다음은 가상 공간 렌더링을 구현한 총 소스코드이다.

```
#define _CRT_SECURE_NO_WARNINGS

#include<GL/glut.h>
#include<stdio.h>
#include<math.h>
#include<iostream>

using namespace std;

#define WIRE 0
#define SHADE 1

typedef struct {
    float x;
    float y;
    float z;
} Point;

typedef struct {
    unsigned int ip[3];
} Face;

int pnum;
int fnum;
Point* mpoint = NULL;
Face* mface = NULL;
Point norm;

GLfloat angle = 0;

int moving;
int mousebegin;
int light_moving;
float scalefactor = 1.0;
int scaling = 0;
int status = 0;

float xmove = 500.0;
float ymove = 500.0;
float zmove = 500.0;
float down = 0.0;

float colorR = 0.3;
float colorG = 0.3;
float colorB = 0.3;

string fname = "c:\\data\\myModel.dat";

void InitLight() {
    GLfloat mat_diffuse[] = { 0.5, 0.4, 0.3, 1.0 };
    GLfloat mat_specular[] = { 1.0, 1.0, 1.0, 1.0 };
    GLfloat mat_ambient[] = { 0.5, 0.4, 0.3, 1.0 };
    GLfloat mat_shininess[] = { 50.0 };
    GLfloat light_specular[] = { 1.0, 1.0, 1.0, 1.0 };
    GLfloat light_diffuse[] = { 0.8, 0.8, 0.8, 1.0 };
    GLfloat light_ambient[] = { 0.3, 0.3, 0.3, 1.0 };
    GLfloat light_position[] = { 200, 200, -200.0, 0.0 };

    glShadeModel(GL_SMOOTH);
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    glLightfv(GL_LIGHT0, GL_POSITION, light_position);
    glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse);
    glLightfv(GL_LIGHT0, GL_SPECULAR, light_specular);
    glLightfv(GL_LIGHT0, GL_AMBIENT, light_ambient);
    glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
    glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
    glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient);
    glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);
}

Point cnormal(Point a, Point b, Point c) {  // flat shading
    Point p, q, r;
    double val;
```

```
    p.x = a.x - b.x; p.y = a.y - b.y; p.z = a.z - b.z;
    q.x = c.x - b.x; q.y = c.y - b.y; q.z = c.z - b.z;

    r.x = p.y * q.z - p.z * q.y;
    r.y = p.z * q.x - p.x * q.z;
    r.z = p.x * q.y - p.y * q.x;

    val = sqrt(r.x * r.x + r.y * r.y + r.z * r.z);
    r.x = r.x / val; r.y = r.y / val; r.z = r.z / val;
    return r;
}

void ReadModel() {
    FILE* f1;
    char s[81];
    int i;

    if (mpoint != NULL)
        delete mpoint;
    if (mface != NULL)
        delete mface;

    if ((f1 = fopen(fname.c_str(), "rt")) == NULL) { printf("No file\n"); exit(0); }
    fscanf(f1, "%s", s);
    printf("%s", s);
    fscanf(f1, "%s", s);
    printf("%s", s);
    fscanf(f1, "%d", &pnum);
    printf("%d\n", pnum);
    mpoint = new Point[pnum];

    for (i = 0; i < pnum; i++) {
        fscanf(f1, "%f", &mpoint[i].x);
        fscanf(f1, "%f", &mpoint[i].y);
        fscanf(f1, "%f", &mpoint[i].z);
        printf("%f %f %f\n", mpoint[i].x, mpoint[i].y, mpoint[i].z);
    }

    fscanf(f1, "%s", s);
    printf("%s", s);
    fscanf(f1, "%s", s);
    printf("%s", s);
    fscanf(f1, "%d", &fnum);
    printf("%d\n", fnum);

    mface = new Face[fnum];
    for (i = 0; i < fnum; i++) {
        fscanf(f1, "%d", &mface[i].ip[0]);
        fscanf(f1, "%d", &mface[i].ip[1]);
        fscanf(f1, "%d", &mface[i].ip[2]);
        printf("%d %d %d\n", mface[i].ip[0], mface[i].ip[1], mface[i].ip[2]);
    }
    fclose(f1);
}

void axis() {
    glLineWidth(2);
    glBegin(GL_LINES);
    glColor3f(1.0, 0.0, 0.0);  // draw x-axis
    glVertex3f(0.0, -100.0, 0.0);
    glVertex3f(160.0, -100.0, 0.0);
    glColor3f(0.0, 1.0, 0.0);  // draw y-axis
    glVertex3f(0.0, -100.0, 0.0);
    glVertex3f(0.0, 60.0, 0.0);
    glColor3f(0.0, 0.0, 1.0);  // draw z-axis
    glVertex3f(0.0, -100.0, 0.0);
    glVertex3f(0.0, -100.0, 160.0);
    glEnd();
    glLineWidth(1.0);
}

void DrawWire(void) {
    glClearColor(1.0f, 1.0f, 1.0f, 1.0f);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

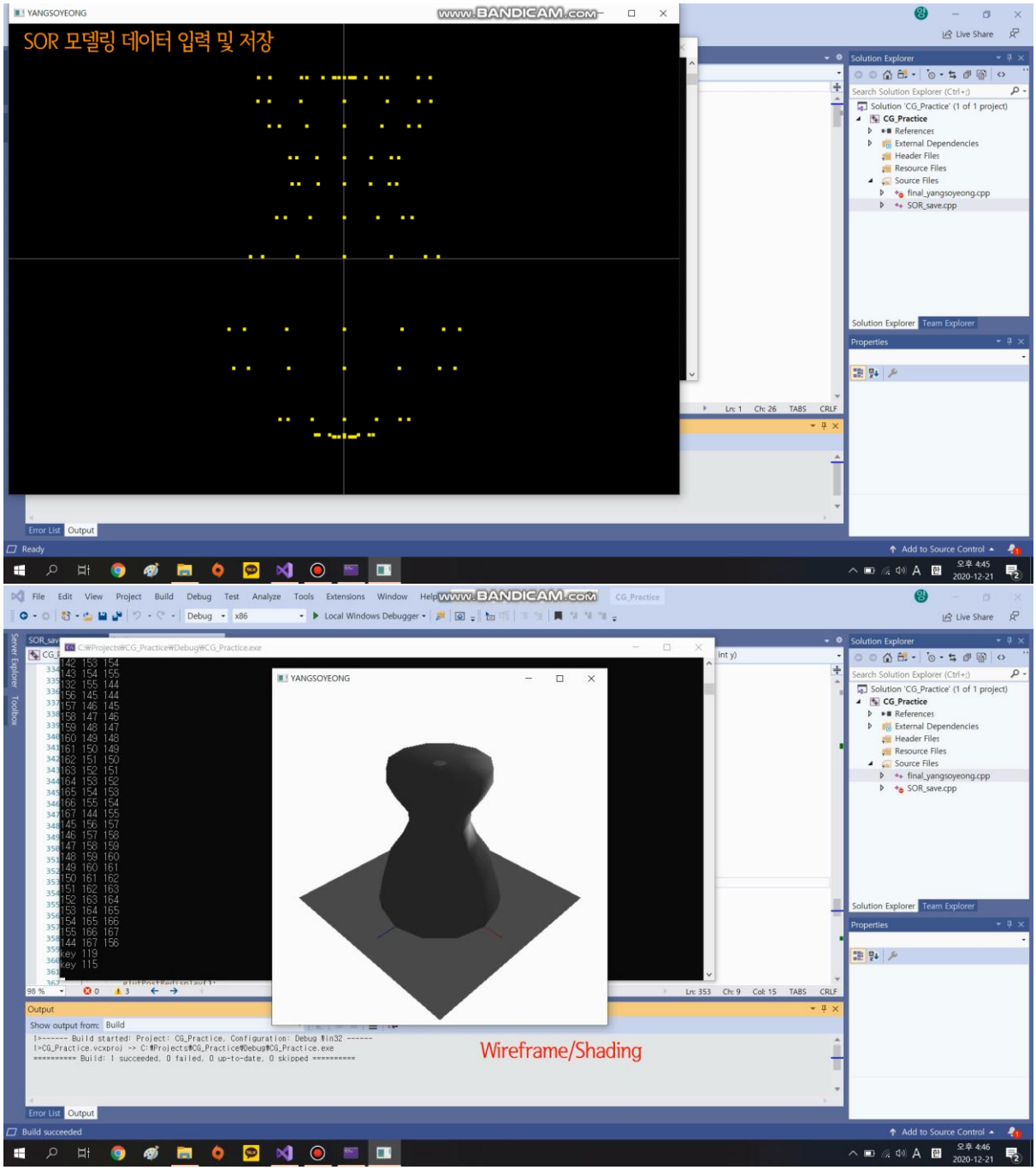
```
    glEnable(GL_DEPTH_TEST);
    axis();
    glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
    glCallList(1);

    glColor3f(0.7, 0.7, 0.7);
    glPushMatrix();
    glTranslatef(0, -100, 0);
    glBegin(GL_QUADS);
    glVertex3f(200, 0, 200);
    glVertex3f(200, 0, -200);
    glVertex3f(-200, 0, -200);
    glVertex3f(-200, 0, 200);
    glEnd();
    glPopMatrix();
    glutSwapBuffers();
}

void DrawShade(void) {
    glClearColor(1.0f, 1.0f, 1.0f, 1.0f);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glEnable(GL_DEPTH_TEST);
    axis();
    glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
    glCallList(1);

    glColor3f(0.6, 0.6, 0.6);  // draw plane
    glPushMatrix();
    glTranslatef(0, -100, 0);
    glBegin(GL_QUADS);
    glVertex3f(200, 0, 200);
    glVertex3f(200, 0, -200);
    glVertex3f(-200, 0, -200);
    glVertex3f(-200, 0, 200);
    glEnd();
    glPopMatrix();
    glutSwapBuffers();
}

void MakeGL_Model(void) {
    int i;
    glShadeModel(GL_SMOOTH);

    if (glIsList(1)) glDeleteLists(1, 1);
    glNewList(1, GL_COMPILE);
    glPushMatrix();

    glRotatef(angle, 0.0, 1.0, 0.0);
    glScalef(scalefactor, scalefactor, scalefactor);
    glColor3f(1, 0, 0);

    glColor3f(colorR, colorG, colorB);

    glEnable(GL_NORMALIZE);
    for (i = 0; i < fnum; i++) {
        Point norm = mpoint[mface[i].ip[0]];
        glBegin(GL_TRIANGLES);

        glNormal3f(norm.x, norm.y, norm.z);  // 면을 이루는 첫번째 vertex
        glVertex3f(mpoint[mface[i].ip[0]].x, mpoint[mface[i].ip[0]].y, mpoint[mface[i].ip[0]].z);

        norm = mpoint[mface[i].ip[1]];
        glNormal3f(norm.x, norm.y, norm.z);  // 면을 이루는 두번째 vertex
        glVertex3f(mpoint[mface[i].ip[1]].x, mpoint[mface[i].ip[1]].y, mpoint[mface[i].ip[1]].z);

        norm = mpoint[mface[i].ip[2]];
        glNormal3f(norm.x, norm.y, norm.z);  // 면을 이루는 세번째 vertex
        glVertex3f(mpoint[mface[i].ip[2]].x, mpoint[mface[i].ip[2]].y, mpoint[mface[i].ip[2]].z);
        glEnd();
    }
    glPopMatrix();
    glEndList();
}
```

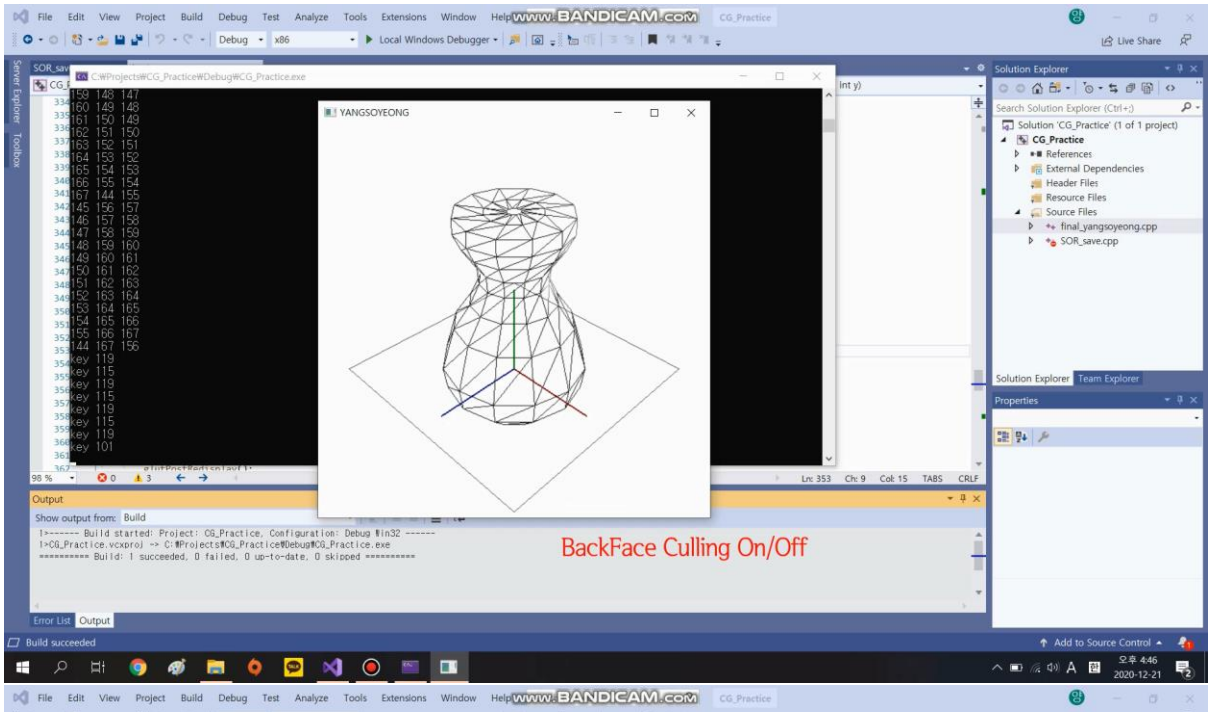
```
void perspmode() {  // perspective mode
    glEnable(GL_DEPTH_TEST);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(40.0, 1.0, 1.0, 2000.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    gluLookAt(600.0, 600.0, 600.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.);
}

void orthomode() {  // ortho mode
    glEnable(GL_DEPTH_TEST);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(-100.0, 100.0, -100.0, 100.0, -800.0, 800.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    gluLookAt(600.0, 600.0, 600.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.);
}

void GLSetupRC(void) {
    glEnable(GL_DEPTH_TEST);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(40.0, 1.0, 1.0, 2000.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glutPostRedisplay();
}

void display(void) {
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    gluLookAt(xmove, ymove, zmove, 0.0, 0.0, 0.0, 0.0, 1.0, down);
    MakeGL_Model();
    if (status == WIRE)
        DrawWire();
    else
        DrawShade();
}

void keyboard(unsigned char key, int x, int y) {
    printf("key %d\n", key);
    switch (key) {
    case 'w':  // wire mode
        status = WIRE;
        glutPostRedisplay();
        break;
    case 's':  // shade mode
        status = SHADE;
        glutPostRedisplay();
        break;

    case 'e':  // hide back
        glEnable(GL_CULL_FACE);
        glCullFace(GL_BACK);
        glutPostRedisplay();
        break;
    case 'd':  // show back
        glDisable(GL_CULL_FACE);
        glutPostRedisplay();
        break;

    case 'r':  // perspective mode
        display();
        perspmode();
        glutPostRedisplay();
        break;
    case 'f':  // ortho mode
        display();
        orthomode();
        glutPostRedisplay();
        break;

```

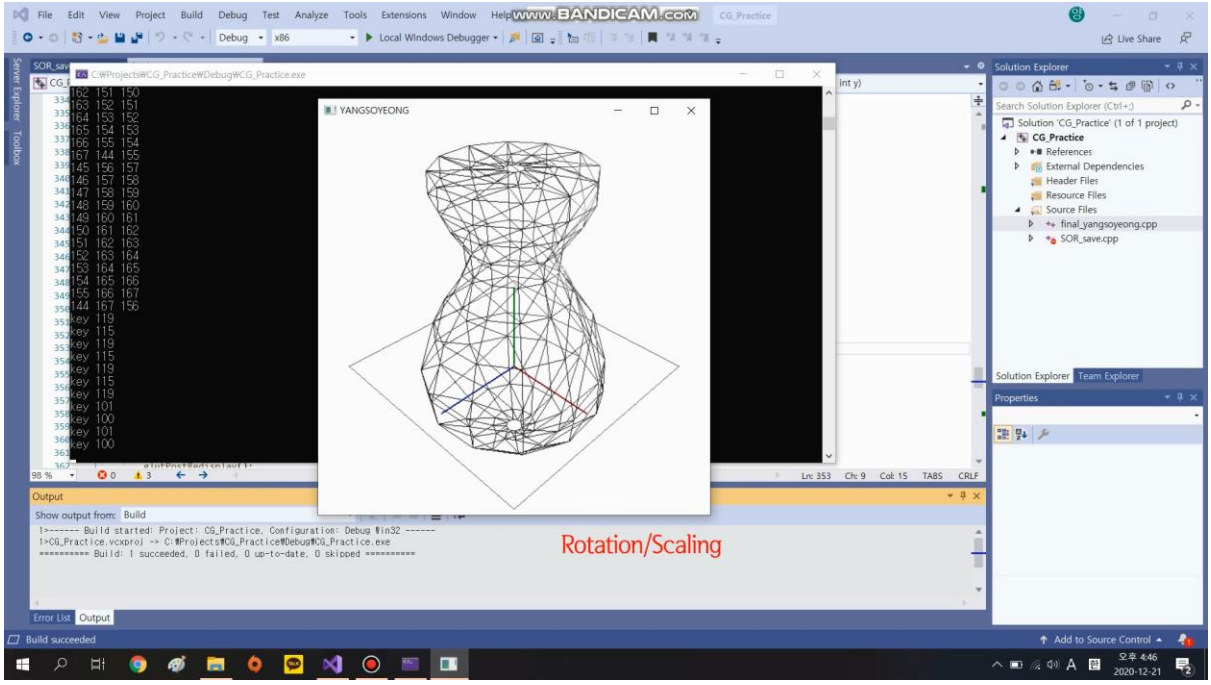
```
    case'q':  // x viewport
        xmove = 600;
        ymove = 0;
        zmove = 0;
        down = 0;
        display();
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
        glutPostRedisplay();
        break;
    case'a':  // y viewports
        xmove = 0;
        ymove = 600;
        zmove = 0;
        down = -1;
        display();
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
        glutPostRedisplay();
        break;
    case'z':  // z viewport
        xmove = 0;
        ymove = 0;
        zmove = 600;
        down = 0;
        display();
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
        glutPostRedisplay();
        break;

    case'1':  // red color
        colorR = 0.8; colorG = 0.4; colorB = 0.4;
        display();
        glutPostRedisplay();
        break;
    case'2':  // green color
        colorR = 0.4; colorG = 0.8; colorB = 0.4;
        display();
        glutPostRedisplay();
        break;
    case'3':  // blue color
        colorR = 0.4; colorG = 0.4; colorB = 0.8;
        display();
        glutPostRedisplay();
        break;

    case 'b':  // xmoving
        xmove += 10;
        display();
        glutPostRedisplay();
        break;
    case 'n':  // ymoving
        ymove += 10;
        display();
        glutPostRedisplay();
        break;
    case 'm':  // zmoving
        zmove += 10;
        display();
        glutPostRedisplay();
        break;
    case 'h':  // xmoving
        xmove -= 10;
        display();
        glutPostRedisplay();
        break;
    case 'j':  // ymoving
        ymove -= 10;
        display();
        glutPostRedisplay();
        break;
    case 'k':  // zmoving
        zmove -= 10;
```

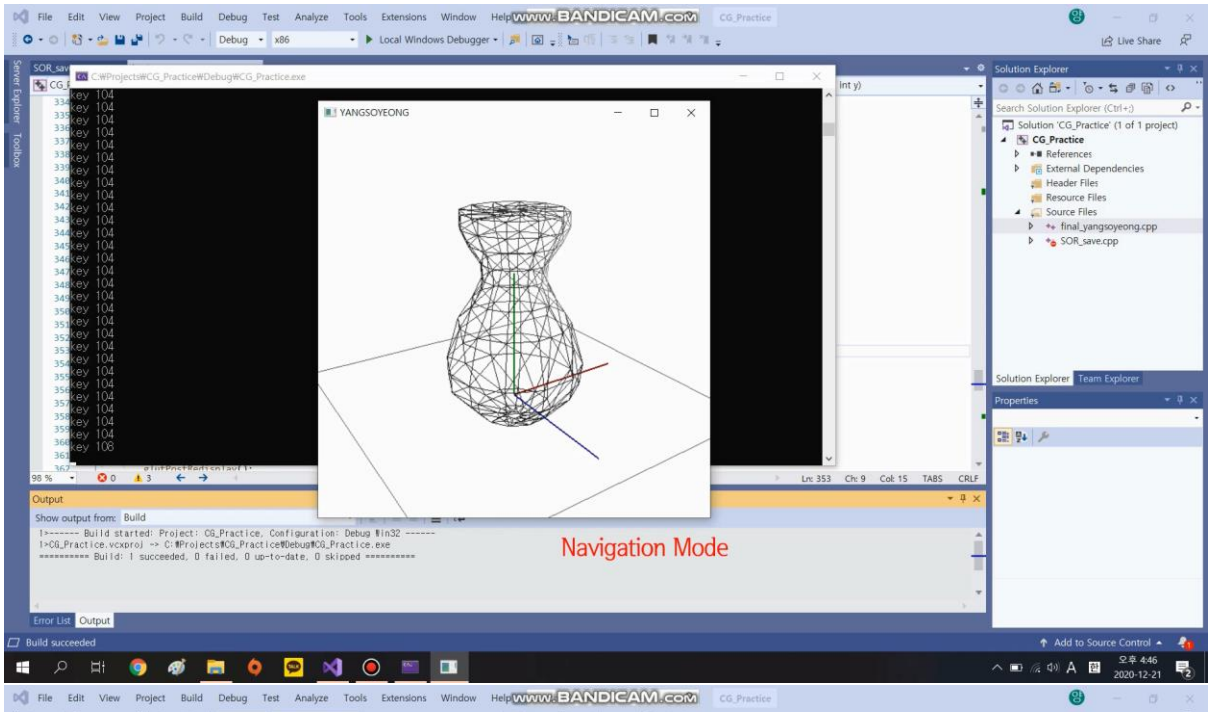
```
            display();
            glutPostRedisplay();
            break;
        }
    }

void mouse(int button, int state, int x, int y) {
    if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN) {
        moving = 1;
        mousebegin = x;
    }
    if (button == GLUT_LEFT_BUTTON && state == GLUT_UP) {
        moving = 0;
    }
    if (button == GLUT_RIGHT_BUTTON && state == GLUT_DOWN) {
        scaling = 1;
        mousebegin = x;
    }
    if (button == GLUT_RIGHT_BUTTON && state == GLUT_UP) {
        scaling = 0;
    }
}

void motion(int x, int y) {
    if (scaling) {
        scalefactor = scalefactor * (1.0 + (mousebegin - x) * 0.0001);
        glutPostRedisplay();
    }
    if (moving) {
        angle = angle + (x - mousebegin);
        mousebegin = x;
        glutPostRedisplay();
    }
}

int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitWindowSize(500, 500); glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glutInitWindowPosition(100, 100);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
    glutCreateWindow("YANGSOYEONG");
    glutDisplayFunc(display);
    glutKeyboardFunc(keyboard);
    glutMouseFunc(mouse);
    glutMotionFunc(motion);
    InitLight();
    ReadModel();
    GLSetupRC();
    glColorMaterial(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE);
    glEnable(GL_COLOR_MATERIAL);
    glutMainLoop();
    return 0;
}
```

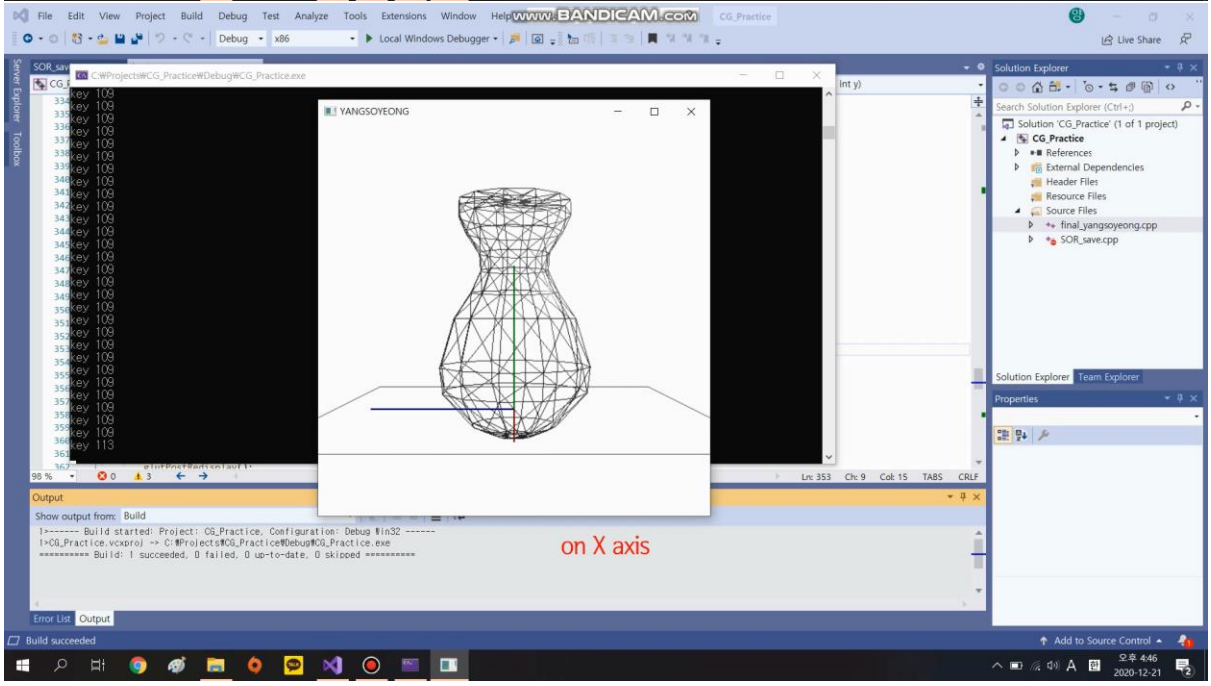
3. 실행 결과

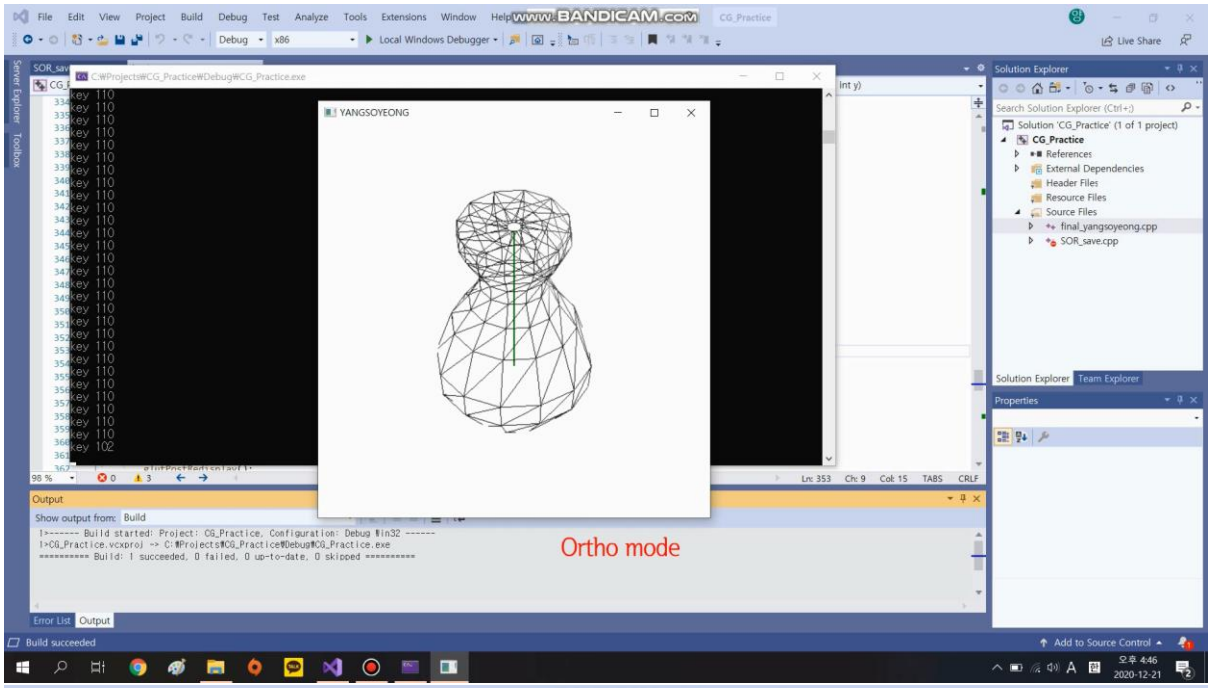

YANGSOYEONG
SOR 모델링 데이터 입력 및 저장
YANGSOYEONG
Wireframe/Shading

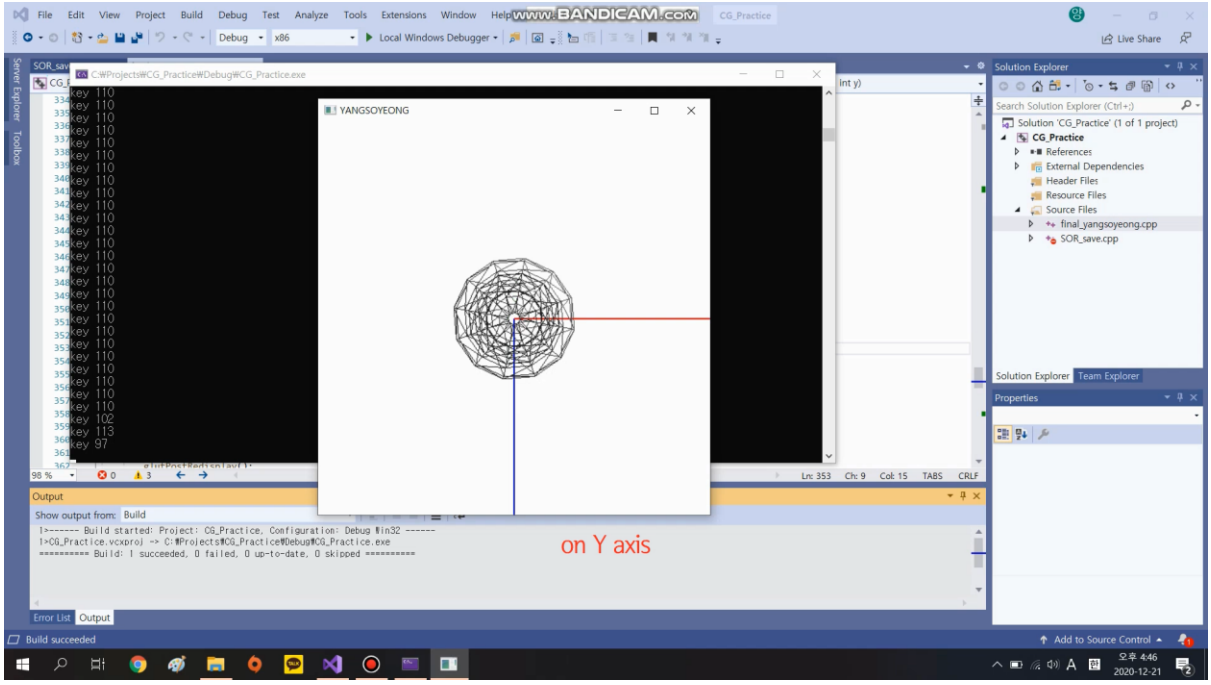

YANGSOYEONG
BackFace Culling On/Off

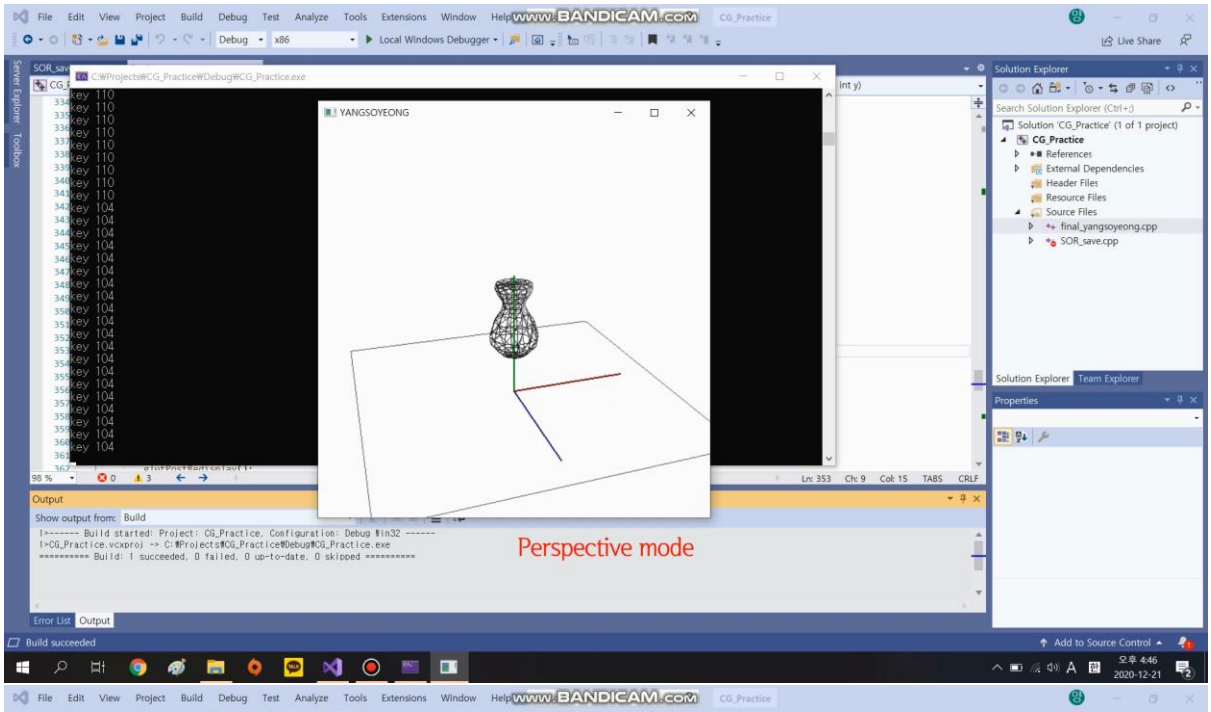

YANGSOYEONG
Rotation/Scaling

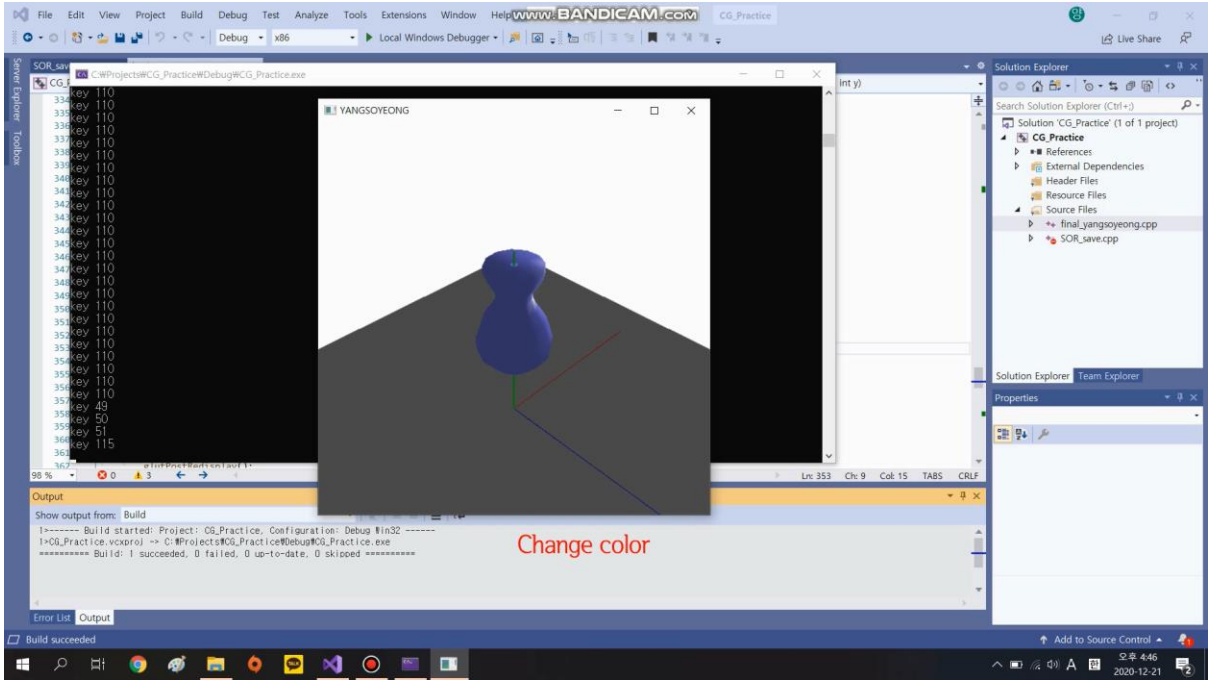

YANGSOYEONG
Navigation Mode


YANGSOYEONG
on X axis


YANGSOYEONG
Ortho mode


YANGSOYEONG
on Y axis


YANGSOYEONG
Perspective mode


YANGSOYEONG
Change color