Comp380

Student Name & ID: Cho In-Young (조인영, ciy405x@kaist.ac.kr) 20150720

Programming Assignment #5

Due May 17th (Wed) 11:59 PM

<wavefront_obj.cc>

```
void wavefront_obj_t::draw() {

    int ptr[1] = { -1 }; //edit: numRasterized 를 저장. -1이면 F4 상태가 아닌 것.
    for ( std::size_t f = 0; f < faces.size(); f++ ) {
        face_t &face = faces[f];
        glBegin( gl_primitive_mode );
        for ( std::size_t v = 0; v < face.count; v++ ) {
            int vi = face.idx_begin + v;
            int i;
            if ( is_flat ) {
                if ( v == 0 ) {
                    glNormal3dv( glm::value_ptr( face.normal ) );
                }
            } else if ( ( i = normal_indices[vi] ) >= 0 ) {
                glNormal3dv( glm::value_ptr( normals[i] ) );
            }
            if ( ( i = texcoord_indices[vi] ) >= 0 ) {
                glTexCoord2dv( glm::value_ptr( texcoords[i] ) );
            }
            if ( ( i = vertex_indices[vi] ) >= 0 ) {
                glVertex3dv( glm::value_ptr( vertices[i] ) );
            }
        }

        glEnd(ptr);
    }

        //edit
        if (ptr[0] != -1) {
                printf("numRasterized = %d₩n", ptr[0]);
        }
        else { // 기본으로 0을 출력.
                printf("numRasterized = %d₩n", 0);
        }

        ptr[0] = -1;
        //end
}
```

<GLRenderer.h>

```
void End(int * ptr); //edit wavefront_obj.cc 에서 glEnd이 ptr을 argument로 받기위해. processPolygon으로 ptr을 주기 위해.
//-------------------------------------------------------------------------
void  processPolygon( vector<GLVertex> &poly, int * nrPtr ); //edit 내부에서 rasterized된 삼각형의 개수를 센다.
```

<GLRenderer.cc>

```
void GLRenderer::End(int * ptr) {
//-------------------------------------------------------------------------
processPolygon( poly, ptr);//edit
//-------------------------------------------------------------------------
void GLRenderer::processPolygon( vector<GLVertex> &verts, int * nrPtr) {

// rasterization
nrPtr[0] = 0;

for( i = 0; i < ( int )triVerts.size(); i += 3 ) {
        if( !RasterizeTriangle( &triVerts[i] ) ) {
            passVerticesToGL( triVerts, coords, true );
            return;
        }
                //edit
                if (!glIsEnabled(GL_CULL_FACE)) {
                        //edit : 그냥 넘어간다.
                } else {
                        double x0, y0, x1, y1, x2, y2;

                        x0 = triVerts[i].position[0];
                        x1 = triVerts[i+1].position[0];
                        x2 = triVerts[i+2].position[0];

                        y0 = triVerts[i].position[1];
                        y1 = triVerts[i + 1].position[1];
                        y2 = triVerts[i + 2].position[1];

                        if ( (x0*y1 - y0*x1) + (x1*y2 - y1*x2) + (x2*y0 - y2*x0) < 0) { // edit: whether back face or front face
                                continue; //edit: 즉, 백페이스인 경우는 더하지 않는다.
                        }
                }

                if (nrPtr[0] == -1) {
                        nrPtr[0] = 1;
                }
                else {
                        nrPtr[0] ++;
                }
                //end
    }
```

<MyGL.cc>

```
double interpolation(glm::vec4 v1, glm::vec4 v2, double u, double v, double w, double d) { // interpolation 계수(비율)을 계산한다.
	double b = u * v2[0] + v * v2[1] + w * v2[2] + d * v2[3]; // 평면 ux + vy + wz + d 에 대한 상대 거리. v2[3] != 1 이면 clipping space. 이때는 원점을 지나는 hyperplane이라 생각 가능.
	double a = u * v1[0] + v * v1[1] + w * v1[2] + d * v1[3];
	return -a / (b - a);
}
//--------------------------------------------------------------------------------
void ClipLine( const vector<GLVertex> &vertsIn, vector<GLVertex> &vertsOut, double u, double v, double w, double d ) {
	// suppose: a line segment vector can be represented by : v1 -> v2
	int size = vertsIn.size();

	if (size == 1) {
		vertsOut = vertsIn;
		return;
	}

	for (int i = 0; i < size; i++) { // Sutherland-Hodgman Algorithm. There are four cases between one line segment and one clipping plane (or line).
		glm::vec4 v1 = vertsIn[i].position;
		glm::vec4 v2 = vertsIn[(i + 1) % size].position; // % size: for cyclical accessing, and avoiding [index out of bounds] exception
		bool v1In, v2In;
		v1In = (u*v1[0] + v*v1[1] + w*v1[2] + d*v1[3] >= 0) ? true : false; // inside(or on) or outside
		v2In = (u*v2[0] + v*v2[1] + w*v2[2] + d*v2[3] >= 0) ? true : false; // inside(or on) or outside

		if (!v1In && v2In) { //case 1: v1: out, v2: in => output: v1', v2		where v1' is interpolated vertex from v1 and v2
			double k = interpolation(v1, v2, u, v, w, d); // compute interpolation coefficient k
			GLVertex newVert;
			for (int j = 0; j < 4; j++)	newVert.position[j] = v1[j] + k * (v2[j] - v1[j]); // linear interpolating for position
			for (int j = 0; j < 4; j++)	newVert.color[j] = vertsIn[i].color[j] + k * (vertsIn[(i + 1) % size].color[j] - vertsIn[i].color[j]); // linear interpolating for color
			for (int j = 0; j < 3; j++)	newVert.normal[j] = vertsIn[i].normal[j] + k * (vertsIn[(i + 1) % size].normal[j] - vertsIn[i].normal[j]); // linear interpolating for normal vector
			for (int j = 0; j < 2; j++)	newVert.texCoord[j] = vertsIn[i].texCoord[j] + k * (vertsIn[(i + 1) % size].texCoord[j] - vertsIn[i].texCoord[j]); // linear interpolating for texture coordinates
			vertsOut.push_back(newVert);
			vertsOut.push_back(vertsIn[(i + 1) % size]);
		}
		else if (v1In && v2In) { //case 2: v1: in, v2: in -> output: v2
			//if (i == 0) vertsOut.push_back(vertsIn[i]); // if v1 is the first vertex of the polygon, push v1
			vertsOut.push_back(vertsIn[(i + 1) % size]);
		}
		else if (v1In && !v2In) { //case 3: v1: in, v2: out -> output: v1'
			double k = interpolation(v1, v2, u, v, w, d); // compute interpolation coefficient k
			GLVertex newVert;
			for (int j = 0; j < 4; j++)	newVert.position[j] = v1[j] + k * (v2[j] - v1[j]);
			for (int j = 0; j < 4; j++)	newVert.color[j] = vertsIn[i].color[j] + k * (vertsIn[(i + 1) % size].color[j] - vertsIn[i].color[j]);
			for (int j = 0; j < 3; j++)	newVert.normal[j] = vertsIn[i].normal[j] + k * (vertsIn[(i + 1) % size].normal[j] - vertsIn[i].normal[j]);
			for (int j = 0; j < 2; j++)	newVert.texCoord[j] = vertsIn[i].texCoord[j] + k *
```

```
(vertsIn[(i + 1) % size].texCoord[j] - vertsIn[i].texCoord[j]);
                        //if (i == 0) vertsOut.push_back(vertsIn[i]); // if v1 is the first vertex of the
polygon, push v1
                        vertsOut.push_back(newVert);
                } //case 4: v1: out, v2: out -> output: none
        }
}

//------------------------------------------------------------------------------
bool MyGL::ClipPolygon( const vector<GLVertex> &vertsIn, vector<GLVertex> &vertsOut ) {
    if( !_doClipping )
        return false;

        vector<GLVertex> cv1, cv2, cv3; //temporary vector

        ClipLine(vertsIn, cv1, -1, 0, 0, 1); // 여기서 자른 것을
        ClipLine(cv1, cv2, 1, 0, 0, 1); // 여기서 또 자르고, 다시 그걸
        ClipLine(cv2, cv3, 0, -1, 0, 1); // 여기서 또 자르고,...
        ClipLine(cv3, vertsOut, 0, 1, 0, 1);

        //vertsOut = vertsIn;

    return true;
}

//------------------------------------------------------------------------------
bool MyGL::TriangulatePolygon( const vector<GLVertex> &polygonVerts, vector<GLVertex> &triangleVerts ) {
    if( !_doTriangulate )
        return false;

    if( polygonVerts.size() >= 3 ) {
                int size = polygonVerts.size();
                for (int i = 1; i <= size - 2; i++) { // 그냥 첫 버텍스를 공통 꼭짓점이라 생각하고 차례대로
triangleVerts에 집어넣는다.
                        triangleVerts.push_back(polygonVerts[0]);
                        triangleVerts.push_back(polygonVerts[i]);
                        triangleVerts.push_back(polygonVerts[i + 1]);
                }
        return true;

    } else {
        return false;
    }
}
//------------------------------------------------------------------------------
bool insideOut(double x, double y, double x0, double y0, double x1, double y1) { // 직선의 내부인지 외부인지
판별한다. 더 정확히 말하면 칠해도 되는지 안되는지.
        double A = y0 - y1;
        double B = x1 - x0;
        double C = x0 * y1 - x1 * y0;

        bool t = (A != 0) ? (A > 0) : (B > 0); // tie-breaker

        return (A*x + B*y + C > 0) || (A*x + B*y + C == 0 && t);
}
//------------------------------------------------------------------------------
bool insideTriangle(double x, double y, glm::vec4 v0, glm::vec4 v1, glm::vec4 v2) { // 삼각형의 내부인지
외부인지 판별.
        return (insideOut(x, y, v0[0], v0[1], v1[0], v1[1]) &&
```

```
                insideOut(x, y, v1[0], v1[1], v2[0], v2[1]) &&
                insideOut(x, y, v2[0], v2[1], v0[0], v0[1]));
}

//-------------------------------------------------------------------------
bool MyGL::RasterizeTriangle( GLVertex verts[3] ) {
    if( !_doRasterize )
        return false;

        glm::vec4 v0 = verts[0].position;
        glm::vec4 v1 = verts[1].position;
        glm::vec4 v2 = verts[2].position;

        double x0 = v0[0], x1 = v1[0], x2 = v2[0],
               y0 = v0[1], y1 = v1[1], y2 = v2[1],
               z0 = v0[2], z1 = v1[2], z2 = v2[2];

        // for Z-buffering, evaluate the coefficients of the linear interpolation w.r.t z (a.k.a plane
equation)
        double det = (x1 - x0) * (y2 - y0) - (x2 - x0) * (y1 - y0);
        double A = (z1 - z0) * (y2 - y0) - (z2 - z0) * (y1 - y0); A /= det;
        double B = - (z1 - z0) * (x2 - x0) + (z2 - z0) * (x1 - x0); B /= det;
        double z; // z-value

        double x_max = __max(__max(x0, x1), x2); // variables for computing total bounding box vertex.
이것은 삼각형 v0v1v2와 접하는 직사각형이다.
        double y_max = __max(__max(y0, y1), y2);
        double x_min = __min(__min(x0, x1), x2);
        double y_min = __min(__min(y0, y1), y2);

        // Solving for Linear Interpolation Equations
        // ..............  by the Formula on p.22, Lecture08.pdf  (slightly modified)
        // [Ar Br Cr 0]          [r0 r1 r2 0]     [(e0)^t 0]
        // [Ag Bg Cg 0]          [g0 g1 g2 0]     [(e1)^t 0]
        // [Ab Bb Cb 0]  =       [b0 b1 b2 0]  *  [(e2)^t 0]  /  (div)
        // [Aa Ba Ca 0]          [a0 a1 a2 0]     [0 0 0  0]
        // mulOut        =       in1           *  in2  라고 쓰자.

        glm_vec4 initial = { 0,0,0,0 };

        glm_vec4 color0 = { verts[0].color[0], verts[0].color[1], verts[0].color[2], verts[0].color[3] };
        glm_vec4 color1 = { verts[1].color[0], verts[1].color[1], verts[1].color[2], verts[1].color[3] };
        glm_vec4 color2 = { verts[2].color[0], verts[2].color[1], verts[2].color[2], verts[2].color[3] };

        glm_vec4 inT[4] = { color0,
                                          color1,
                                          color2,
                                          initial };
        glm_vec4 in1[4];

        glm_mat4_transpose(inT, in1);

        double div = (x1*y2 - x2*y1) - (x0*y2 - x2*y0) + (x0*y1 - x1*y0);

        glm_vec4 in2[4] = { { y1 - y2, x2 - x1, x1*y2 - x2*y1, 0 },
                                          { y2 - y0, x0 - x2, x2*y0 - x0*y2, 0 },
                                          { y0 - y1, x1 - x0, x0*y1 - x1*y0, 0 },
                                          initial,
        };

        glm_vec4 mulOut[4];
```

```
        glm_mat4_mul(in2, in1, mulOut);
        glm_vec4 rgba; // glm_vec4 -> glm::vec4로의 변환을 위해 임시저장용.
        glm::vec4 color;

        for (int x = x_min; x <= x_max; x++) { // total bounding box를 설정.
                for (int y = y_min; y <= y_max; y++) {
                        if (insideTriangle(x, y, v0, v1, v2) || insideTriangle(x, y, v0, v2, v1)) { //
삼각형 버텍스들의 orientation에 독립적으로 내.외부 판별.
                                glm_vec4 vecMulIn2 = { x, y, 1, 0 };
                                rgba = glm_vec4_mul_mat4(vecMulIn2, mulOut);
                                color = { rgba.m128_f32[0]/div, rgba.m128_f32[1]/div,
                                        rgba.m128_f32[2]/div, rgba.m128_f32[3]/div };

                                // interpolating for Z-buffering
                                z = z0 + A * (x - x0) + B * (y - y0);

                                //z_buffering
                                if (depthTestEnabled){
                                        if (frameBuffer.GetDepth(x, y) > z) {
                                                frameBuffer.SetDepth(x, y, z);
                                                frameBuffer.SetPixel(x, y, color);
                                        }
                                }
                                else if (!depthTestEnabled) {
                                        frameBuffer.SetPixel(x, y, color);
                                }

                                /*
                                main.cc의
                                if( optBackFaceCulling ) {
                                glFrontFace( GL_CCW );
                                glEnable( GL_CULL_FACE );
                                glCullFace( GL_BACK );
                                }
                                부분 덕분에 따로 여기서 백페이스 컬링을 구현하지 않아도 충분.
                                */
                        }
                }
        }
    return true;
}
```