그래픽스 강의노트 06 - 조명 2 (메시)

강영민

동명대학교

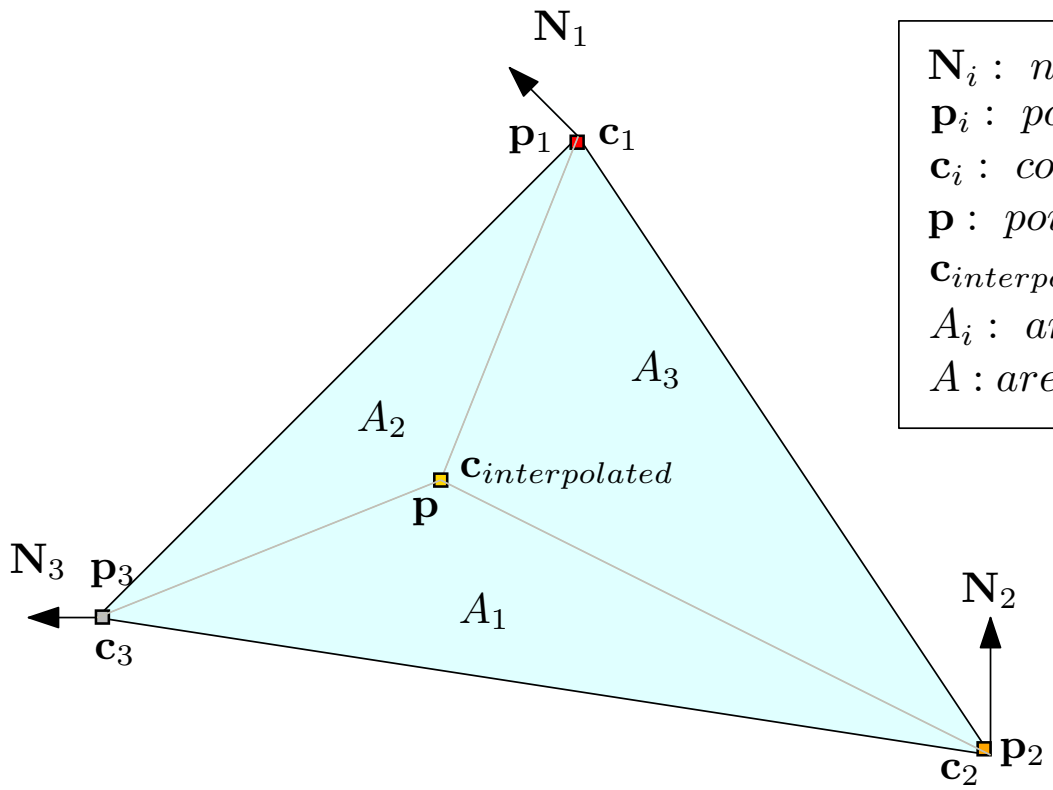
2015년 2학기

법선의 설정과 메시(mesh) 데이터 그리기

- 앞서 그려본 주전자는 그려지는 면의 법선 벡터가 내장되어 있는 glutSolidTeapot 함수를 호출
- 임의의 면을 그릴 때는 이러한 미리 정의된 법선 벡터가 존재하지 않음
- 퐁 쉐이딩의 계산이 요구되는 법선 벡터를 오픈지엘에 넘겨주어야 함

구로(Gouraud) 세이딩

- 각 정점에 법선 벡터 정의
- 법선 벡터와 조명의 관계를 이용하여 정점별 퐁 쉐이딩
- 정점의 색을 이용하여 내부의 픽셀은 선형보간(linear interpolation)을 통해 얻음



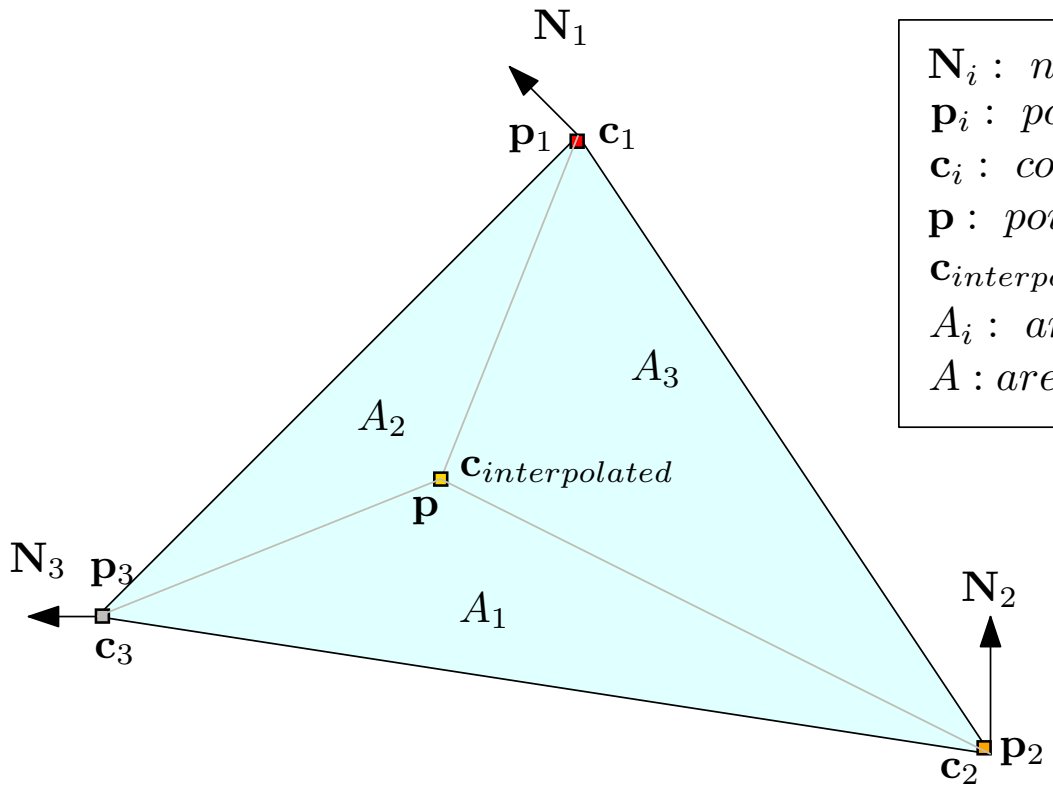
$\mathbf{N}_i$: *normal at point* i
$\mathbf{p}_i$: *point* i
$\mathbf{c}_i$: *color at point* i *(Phong)*
$\mathbf{p}$: *point to be rendered*
$\mathbf{c}_{interpolated}$: *interpolated color*
A_i : *area of* $\triangle(\mathbf{p}, \forall \mathbf{p}_{j \neq i})$
A : *area of* $\triangle(\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3)$

Gouraud Shading

$$\mathbf{c}_{interpolated} = \frac{\sum_{i=1}^{3} A_i \mathbf{c}_i}{A}$$

구로(Gouraud) 세이딩

- 각 정점에 법선 벡터 정의
- 법선 벡터와 조명의 관계를 이용하여 정점별 퐁 쉐이딩
- 정점의 색을 이용하여 내부의 픽셀은 선형보간(linear interpolation)을 통해 얻음



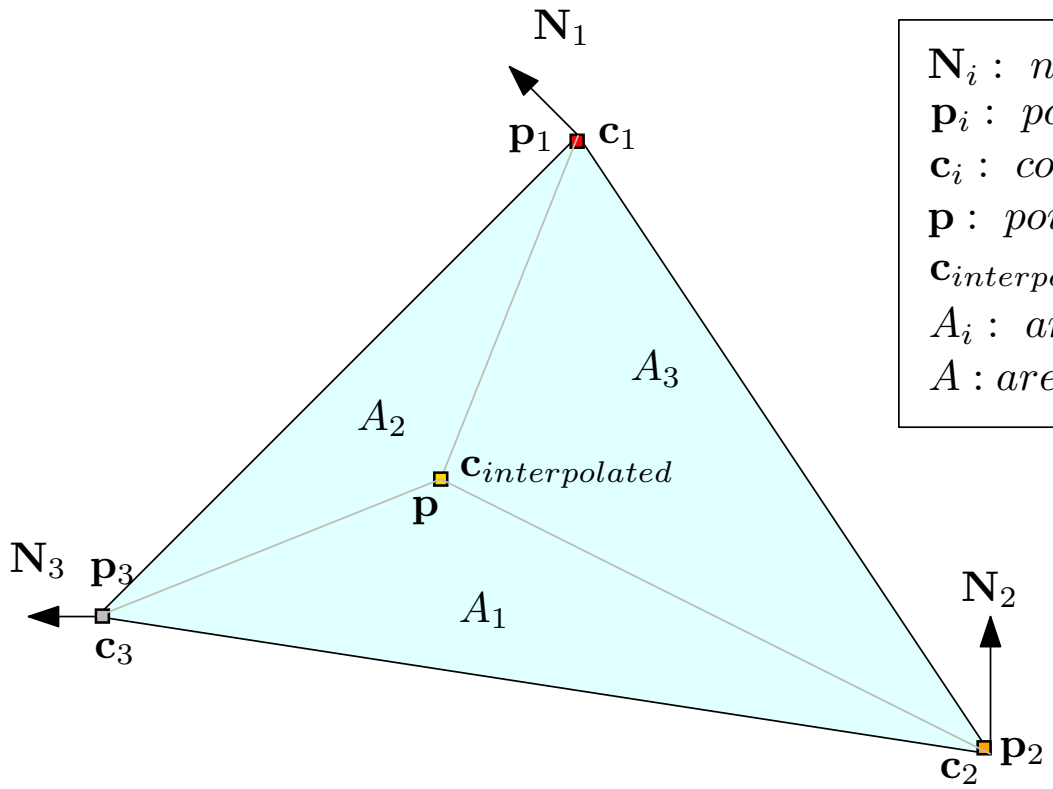
$\mathbf{N}_i$: *normal at point* i
$\mathbf{p}_i$: *point* i
$\mathbf{c}_i$: *color at point* i *(Phong)*
$\mathbf{p}$: *point to be rendered*
$\mathbf{c}_{interpolated}$: *interpolated color*
A_i : *area of* $\triangle(\mathbf{p}, \forall \mathbf{p}_{j \neq i})$
A : *area of* $\triangle(\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3)$

Gouraud Shading

$$\mathbf{c}_{interpolated} = \frac{\sum_{i=1}^{3} A_i \mathbf{c}_i}{A}$$

구로(Gouraud) 세이딩

- 각 정점에 법선 벡터 정의
- 법선 벡터와 조명의 관계를 이용하여 정점별 퐁 쉐이딩
- 정점의 색을 이용하여 내부의 픽셀은 선형보간(linear interpolation)을 통해 얻음

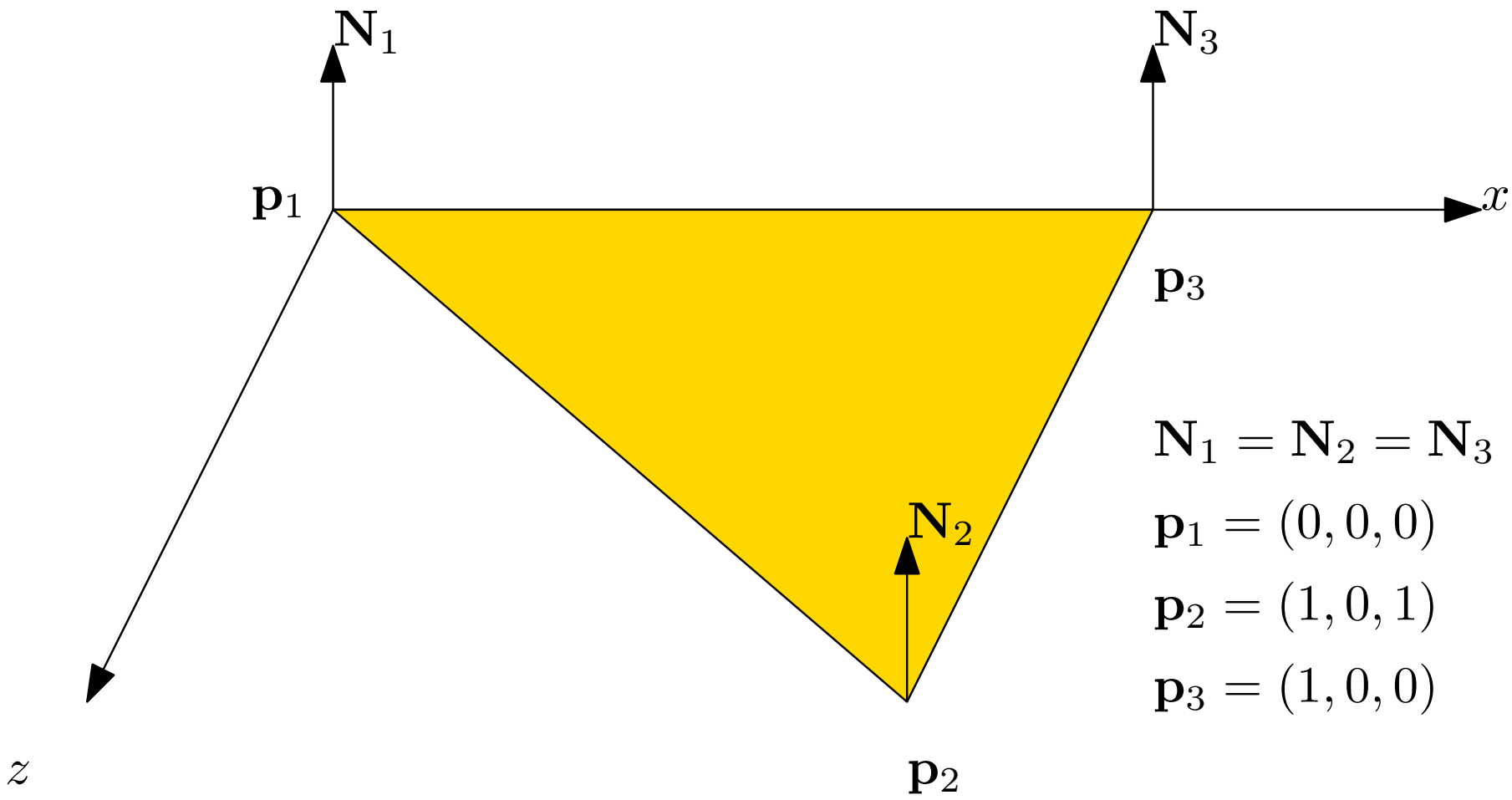


$\mathbf{N}_i$: *normal at point* i
$\mathbf{p}_i$: *point* i
$\mathbf{c}_i$: *color at point* i *(Phong)*
$\mathbf{p}$: *point to be rendered*
$\mathbf{c}_{interpolated}$: *interpolated color*
A_i : *area of* $\triangle(\mathbf{p}, \forall \mathbf{p}_{j \neq i})$
A : *area of* $\triangle(\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3)$

Gouraud Shading

$$\mathbf{c}_{interpolated} = \frac{\sum_{i=1}^{3} A_i \mathbf{c}_i}{A}$$

구로(Gouraud) 세이딩 예제



$\mathbf{N}_1 = \mathbf{N}_2 = \mathbf{N}_3 = (0,$

$\mathbf{p}_1 = (0, 0, 0)$

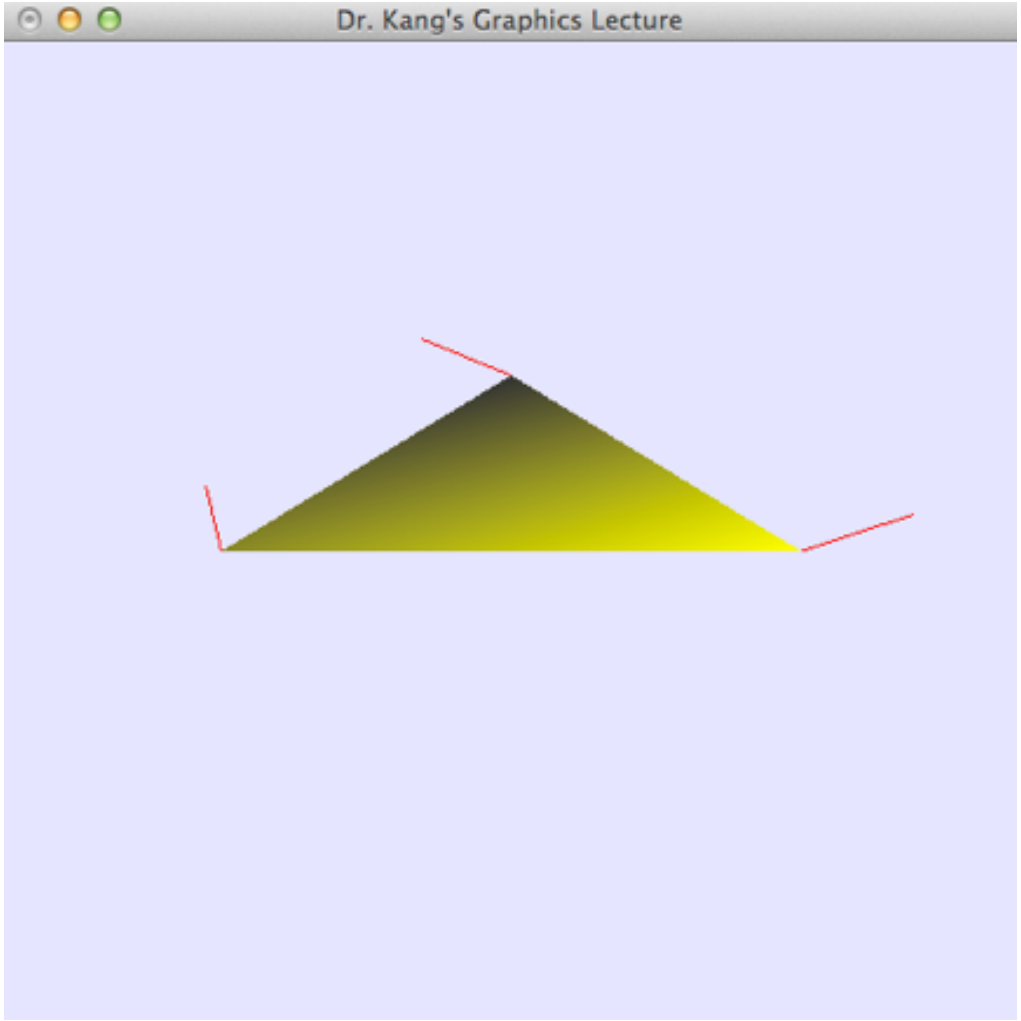
$\mathbf{p}_2 = (1, 0, 1)$

$\mathbf{p}_3 = (1, 0, 0)$

구로 세이딩을 위한 코딩

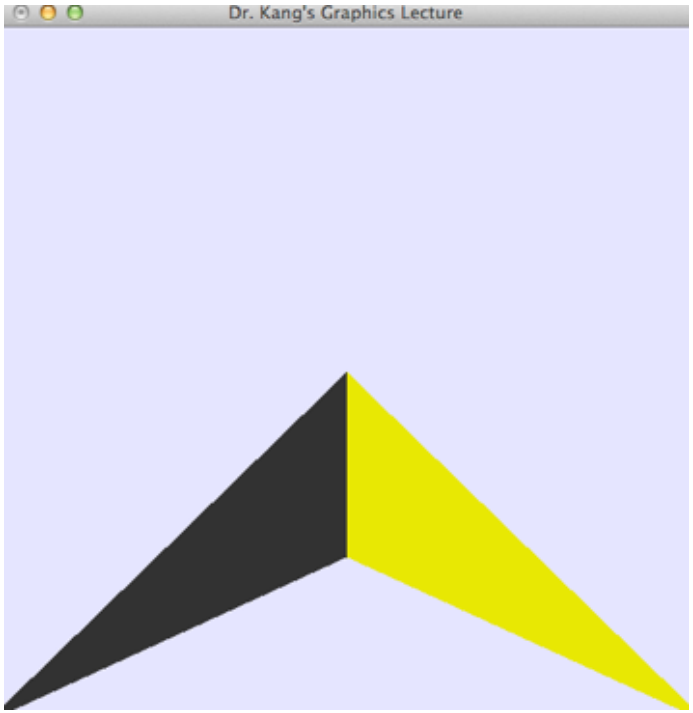
```
\begin{lstlisting}
  glBegin(GL_TRIANGLES);
  glNormal3f(0,1,0);
  glVertex3f(0,0,0);
  glNormal3f(2/sqrt(3),1/sqrt(3),0);
  glVertex3f(2,0,0);
  glNormal3f(-2/sqrt(3),1/sqrt(3),0);
  glVertex3f(1,0,-1);
  glEnd();
```

구로 세이딩 결과



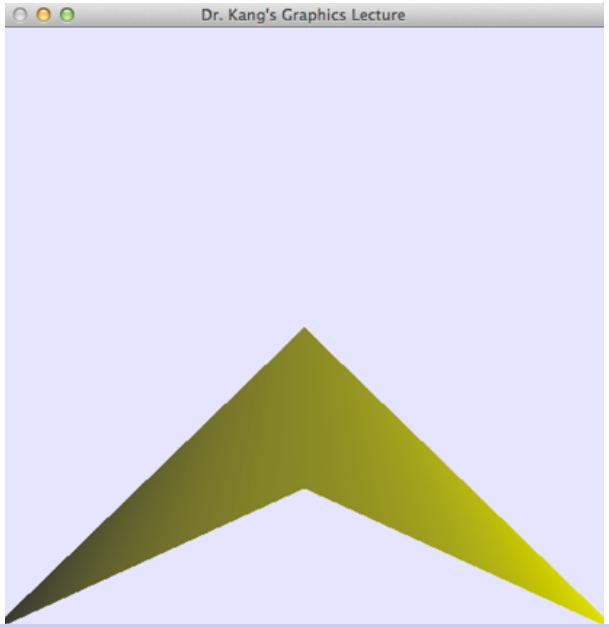
구로 세이딩 - 두 개의 인접한 면 그리기

```
glBegin(GL_TRIANGLES);
glNormal3f(-1/sqrt(2),1/sqrt(2),0);
glVertex3f(0,1,0);
glVertex3f(-1,0,0);
glVertex3f(0,1,1);
glNormal3f(1/sqrt(2),1/sqrt(2),0);
glVertex3f(0,1,0);
glVertex3f(0,1,1);
glVertex3f(1,0,0);
glEnd();
```



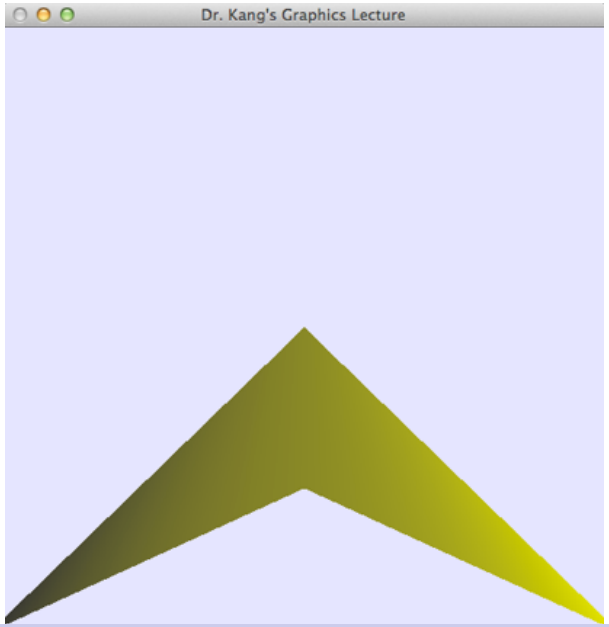
구로 세이딩 - 법선 벡터 공유하기

```
glBegin(GL_TRIANGLES);
glNormal3f(0,1,0);
glVertex3f(0,1,0);
glNormal3f(-1/sqrt(2),1/sqrt(2),0);
glVertex3f(-1,0,0);
glNormal3f(0,1,0);
glVertex3f(0,1,1);
glNormal3f(0,1,0);
glVertex3f(0,1,0);
glNormal3f(0,1,0);
glVertex3f(0,1,1);
glNormal3f(1/sqrt(2),1/sqrt(2),0);
glVertex3f(1,0,0);
glEnd();
```



구로 세이딩 - 법선 벡터 공유하기

```
glBegin(GL_TRIANGLES);
glNormal3f(0,1,0);
glVertex3f(0,1,0);
glNormal3f(-1/sqrt(2),1/sqrt(2),0);
glVertex3f(-1,0,0);
glNormal3f(0,1,0);
glVertex3f(0,1,1);
glNormal3f(0,1,0);
glVertex3f(0,1,0);
glNormal3f(0,1,0);
glVertex3f(0,1,1);
glNormal3f(1/sqrt(2),1/sqrt(2),0);
glVertex3f(1,0,0);
glEnd();
```



메시(mesh)

Table 1 : 메시 데이터 포맷의 예시

포맷	실제 데이터 예시
numVertices n	4
vertex 1 (x_1, y_1, z_1)	0.0 1.0 0.0
vertex 2 (x_2, y_2, z_2)	-1.0 0.0 0.0
...	0.0 1.0 1.0
vertex n (x_n, y_n, z_n)	1.0 0.0 0.0
numFaces m	2
face 1 $(f_1.v_1, f_1.v_2, f_1.v_3)$	0 1 2
face 2 $(f_2.v_1, f_2.v_2, f_2.v_3)$	0 2 3
...	

메시(mesh) 로딩

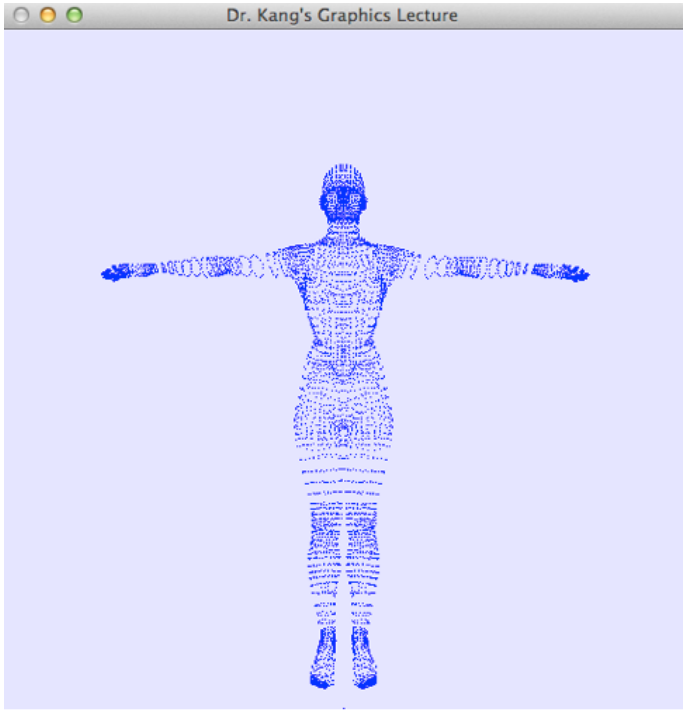
```
#ifndef _mesh_sms_hh_
#define _mesh_sms_hh_
class cvertex {
public:
    float x;
    float y;
    float z;
}; // 하나의 점을 구성하는 좌표값 3 개
class cface {
public:
    int v0; int v1; int v2;
}; //하나의 삼각형 면을 구성하는 세 개 정점의 인덱스들

class CMesh {
    int nV;   // 정점의 개수
    int nF;   // 메시 구성 삼각형 면의 수
    cvertex *v; // 정점 데이터 배열
    cface   *f; // 면 데이터 배열

public:
    float minx, miny, minz;
    float maxx, maxy, maxz; // 메시를 둘러싸는 AABB 경계상자

public:
    CMesh();   // constructor
    ~CMesh(); // destructor
    // 메시를 읽고 그리는 메소드들
    void loadMesh(char *meshFileName);
    void drawMesh(void);
};
#endif
```

메시(mesh) 로딩

```
#include "Mesh.h"
[[필요한 헤더 파일들 포함]]
#define BIGNUMBER 10000000000000000000000000.0
CMesh::CMesh() : nV(0), nF(0), v(NULL), f(NULL), minx(BIGNUMBER), miny(
    BIGNUMBER), minz(BIGNUMBER), maxx(-BIGNUMBER), maxy(-BIGNUMBER), maxz(-
    BIGNUMBER) { }

CMesh::~CMesh() {
    if(v) delete[] v;
    if(f) delete[] f;
}
void CMesh::loadMesh(char *meshFileName) {
    FILE *fptr = fopen(meshFileName, "r");
    if(!meshFileName || !fptr) { printf("file open error\n"); exit(0);
     }
    fscanf(fptr, "%d", &nV);      // 정점의 개수 읽기
    v = new cvertex[nV];
    for (int i=0; i<nV; i++) {
        // nV개의 정점 정보를 읽음
        fscanf(fptr, "%f", &v[i].x);
        fscanf(fptr, "%f", &v[i].y);
        fscanf(fptr, "%f", &v[i].z);
    }
    fscanf(fptr, "%d", &nF);      // 면의 개수 읽기
    f = new cface[nF];
    for (int i=0; i<nF; i++) { // nF개의 면 정보를 읽음
        fscanf(fptr, "%d", &f[i].v0);
        fscanf(fptr, "%d", &f[i].v1);
        fscanf(fptr, "%d", &f[i].v2);
    }
}
```

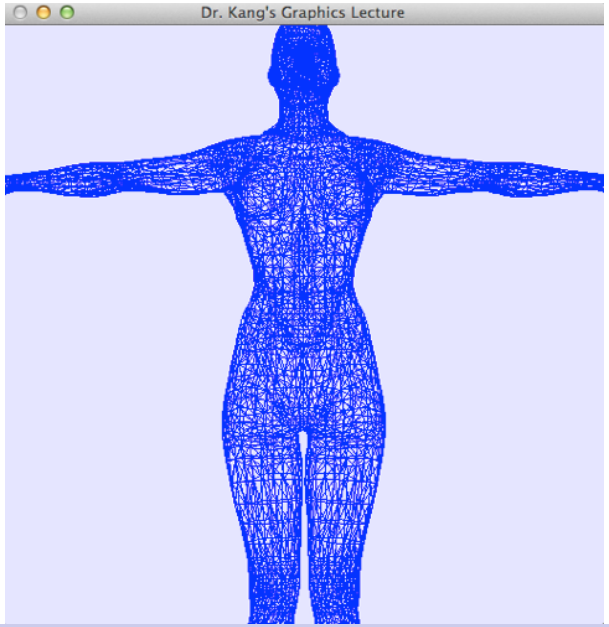
메시(mesh) 그리기 1

```
void CMesh::drawMesh(void) {
  if(!v || !f) return;
  glBegin(GL_POINTS);
  for(int i=0;i<nV;i++) {
    glVertex3f( v[i].x, v[i].y, v[i].z);
  }
  glEnd();
}
```



메시(mesh) 그리기 2

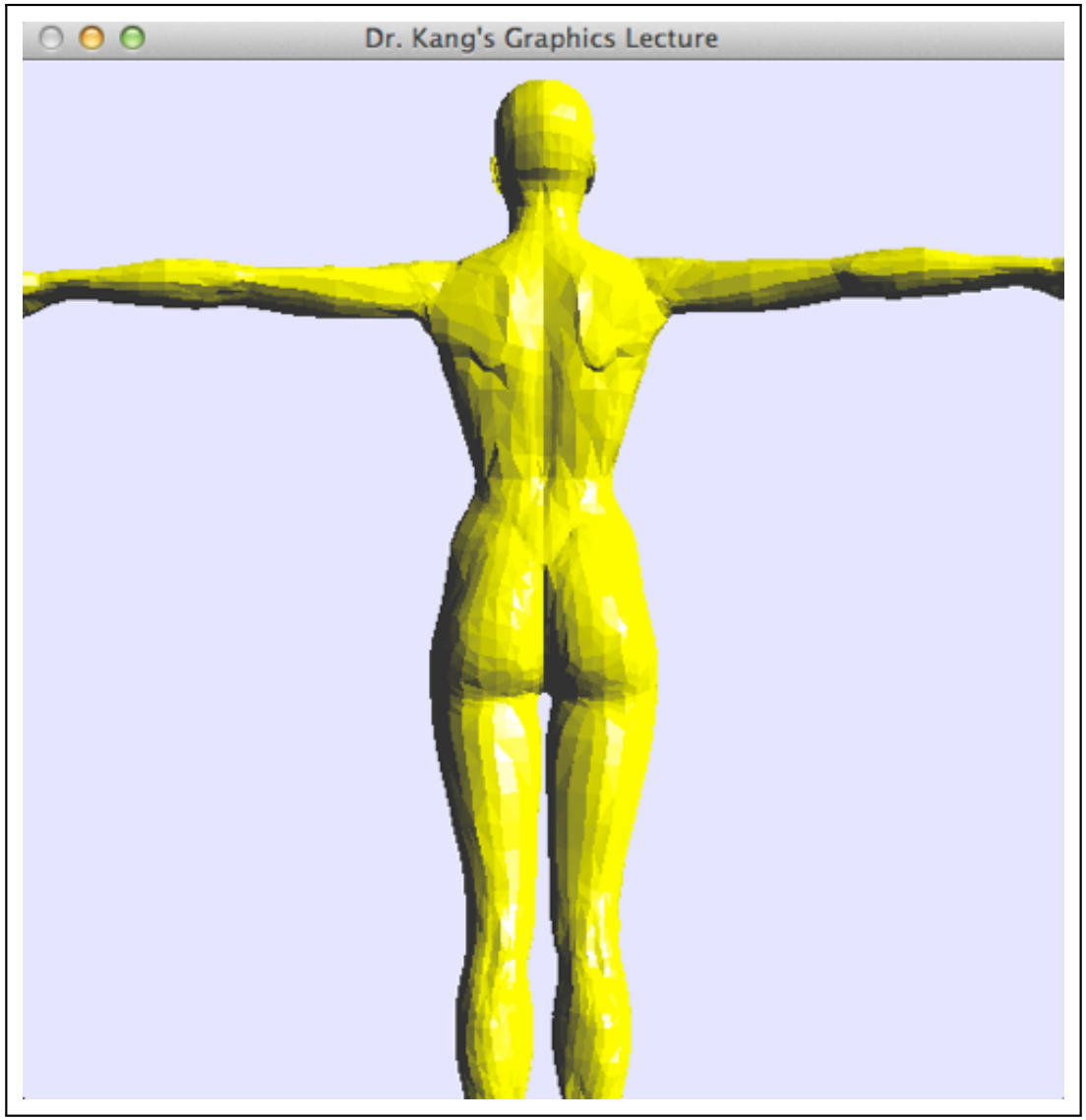
```
void CMesh::drawMesh(void) {
    for (int i=0; i<nF; i++) {
        // i-번째 면을 그리는 작업
        int a, b, c; // 삼각형을 구성하는 세 정점의 인덱스
        a = f[i].v0; // i-번째 면의 0번 정점
        b = f[i].v1; // i-번째 면의 1번 정점
        c = f[i].v2; // i-번째 면의 2번 정점
        glBegin(GL_LINE_LOOP);
        glVertex3f(v[a].x, v[a].y, v[a].z); // a 정점의 좌표
        glVertex3f(v[b].x, v[b].y, v[b].z);// b정점의 좌표
        glVertex3f(v[c].x, v[c].y, v[c].z); // c 정점의 좌표
        glEnd();
    }
}
```



메시(mesh) 그리기 3 - 면

```
void CMesh::drawMesh(void) {
    if(!v || !f) return;
    glBegin (GL_TRIANGLES) ;

    for(int i=0;i<nF;i++) {
        // 법선 벡터를 계산하기 위한 정보
        cvertex p0, p1, p2;   // 세 개의 정점 p0, p1, p2
        cvertex v0, v1; cvertex n; //면 위의 두 벡터 v0, v1 와 법선벡터 n
        p0 = v[f[i].v0];
        p1 = v[f[i].v1];
        p2 = v[f[i].v2];

        // v0 = p1 − p0
        v0.x = p1.x-p0.x; v0.y = p1.y-p0.y; v0.z = p1.z-p0.z;
        // v1 = p2 − p0
        v1.x = p2.x-p0.x; v1.y = p2.y-p0.y; v1.z = p2.z-p0.z;

        // n = (v1×v0)/|v1×v0|
        n.x = v0.y*v1.z-v0.z*v1.y;
        n.y = v0.z*v1.x-v0.x*v1.z;
        n.z = v0.x*v1.y-v0.y*v1.x;
        float len = sqrt(n.x*n.x+n.y*n.y+n.z*n.z);
        n.x /= len; n.y /= len; n.z /= len;

        glNormal3f(n.x, n.y, n.z);
        glVertex3f( p0.x, p0.y, p0.z);
        glVertex3f( p1.x, p1.y, p1.z);
        glVertex3f( p2.x, p2.y, p2.z);
    }
    glEnd() ;
}
```

메시(mesh) 면 그리기 결과



메시(mesh) 그리기 - 법선 벡터의 저장

- 매번 면을 그릴 때마다 법선벡터를 계산하는 것은 비효율적
- 한 번 법선 벡터를 계산한 뒤, 이 결과를 각 정점별로 저장

```
class CMesh {
    int nV;   // number of vertices
    int nF;   // number of faces
    cvertex *v; // vertex array
    cface   *f; // face array
    cvertex *n; // 법선 벡터의 배열
...
public:
...
    void computeNormals(void); // 법선 벡터를 계산하여 n에 채움
};

#endif
```

메시(mesh) 그리기 - 법선 벡터의 계산

- 입력된 데이터를 이용하여 법선 벡터를 계산하는 computeNormals 메소드를 호출
- computeNormals 메소드는 앞에서 법선 벡터를 계산했던 방식과 동일한 방법으로 각각의 면에 대해 법선 N을 계산
- 이 법선 벡터는 바로 사용되지 않음
- 어떤 면을 구성하는 정점이 $\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2$ 라면 각각의 법선 벡터를 $\mathbf{n}_0, \mathbf{n}_1, \mathbf{n}_2$.
- 이 면에서 얻어진 법선 벡터는 $\mathbf{n}_0, \mathbf{n}_1, \mathbf{n}_2$ 에 누적

$$\mathbf{n}_0 = \mathbf{n}_0 + \mathbf{N}$$
$$\mathbf{n}_1 = \mathbf{n}_1 + \mathbf{N}$$
$$\mathbf{n}_2 = \mathbf{n}_2 + \mathbf{N}$$

모든 $\mathbf{n}_i$ 에 대해 정규화를 수행하면 각 정정별 법선을 얻을 수 있다.

메시(mesh) 그리기 - 법선 벡터의 저장

```
void CMesh::computeNormals(void) { // private method
    for(int i=0;i<nV;i++) {
        n[i].x = n[i].y = n[i].z = 0.0;
    }
    for(int i=0; i<nF; i++) {
        // 각각의 면에 대해서 외적을 이용한 법선 계산을 수행한다
        cvertex p0, p1, p2;
        cvertex v0, v1; cvertex N;
        int vert0, vert1, vert2;
        vert0 = f[i].v0; vert1 = f[i].v1; vert2 = f[i].v2;
        p0 = v[vert0];
        p1 = v[vert1];
        p2 = v[vert2];
        v0.x = p1.x-p0.x; v0.y = p1.y-p0.y; v0.z = p1.z-p0.z;
        v1.x = p2.x-p0.x; v1.y = p2.y-p0.y; v1.z = p2.z-p0.z;
        N.x = v0.y*v1.z-v0.z*v1.y;
        N.y = v0.z*v1.x-v0.x*v1.z;
        N.z = v0.x*v1.y-v0.y*v1.x;
        // 이렇게 얻어진 법선 벡터는 이 면을 구성하고 있는 정점 3 개의 법선 데이터에 누적된다
        n[vert0].x += N.x; n[vert0].y += N.y; n[vert0].z += N.z;
        n[vert1].x += N.x; n[vert1].y += N.y; n[vert1].z += N.z;
        n[vert2].x += N.x; n[vert2].y += N.y; n[vert2].z += N.z;
    }
    for(int i=0;i<nV;i++) {
        // 모든 정점에 대해 누적된 법선 벡터를 정규화한다
        float len = sqrt(n[i].x*n[i].x+n[i].y*n[i].y+n[i].z*n[i].z);
        n[i].x /= len; n[i].y /= len; n[i].z /= len;
    }
}
#endif
```

메시(mesh) 그리기 - 렌더링 결과