그래픽스 강의노트 07 - 조명 2 (메시)

강영민

동명대학교

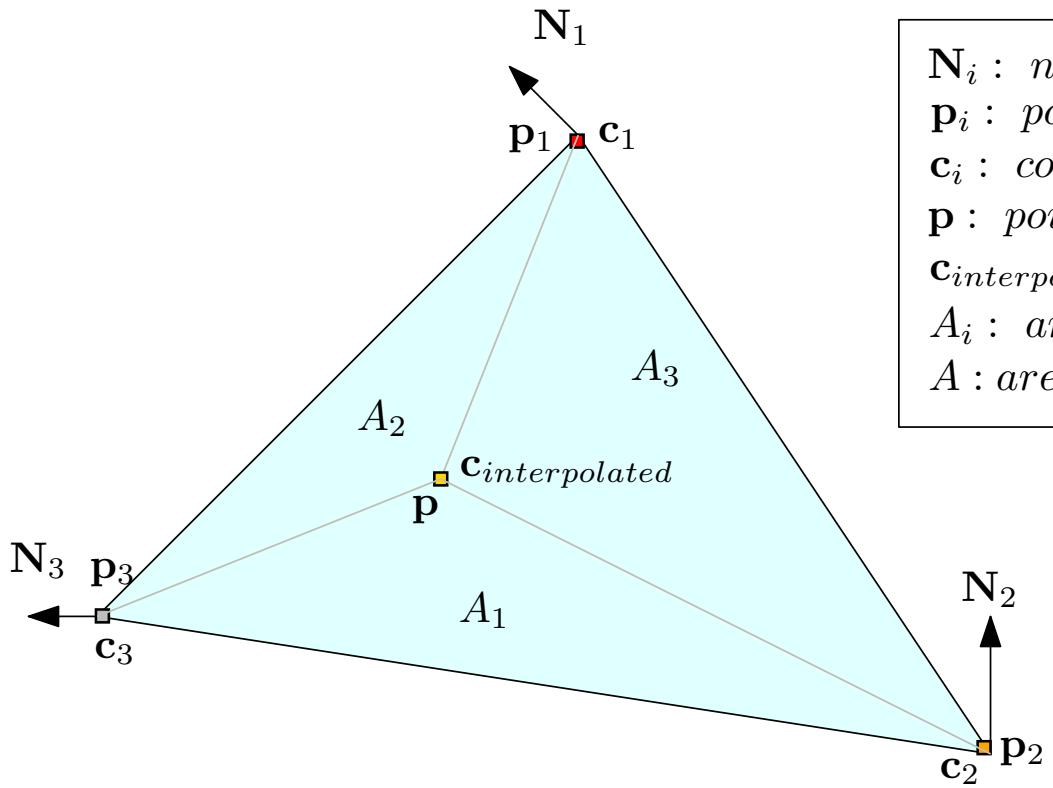
2021년 2학기

법선의 설정과 메시(mesh) 데이터 그리기

- 면의 법선 벡터를 정의하여 면을 그려 보았음
- 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



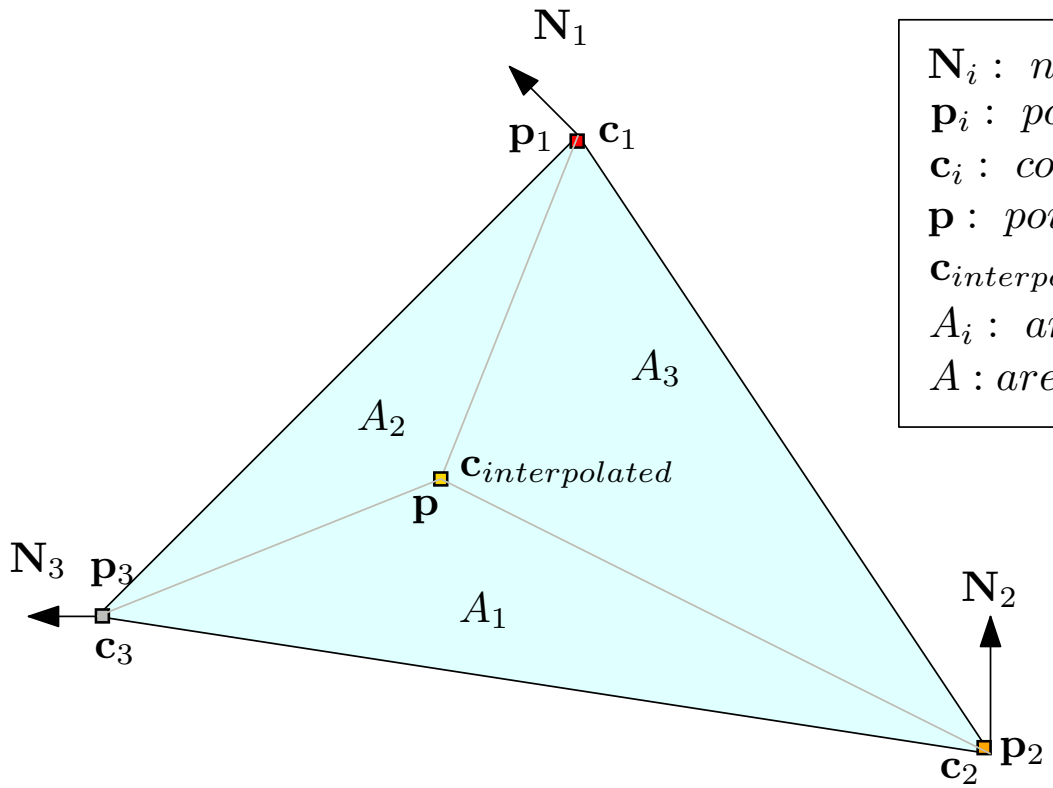
$\mathbf{N}_i$: *normal at point* i
$\mathbf{p}_i$: *point* i
$\mathbf{c}_i$: *color at point* i *(Phong)*
$\mathbf{p}$: *point to be rendered*
$\mathbf{c}_{interpolated}$: *interpolated color*
A_i : *area of* $\triangle(\mathbf{p}, \forall \mathbf{p}_{j \neq i})$
A : *area of* $\triangle(\mathbf{p}_1,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

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



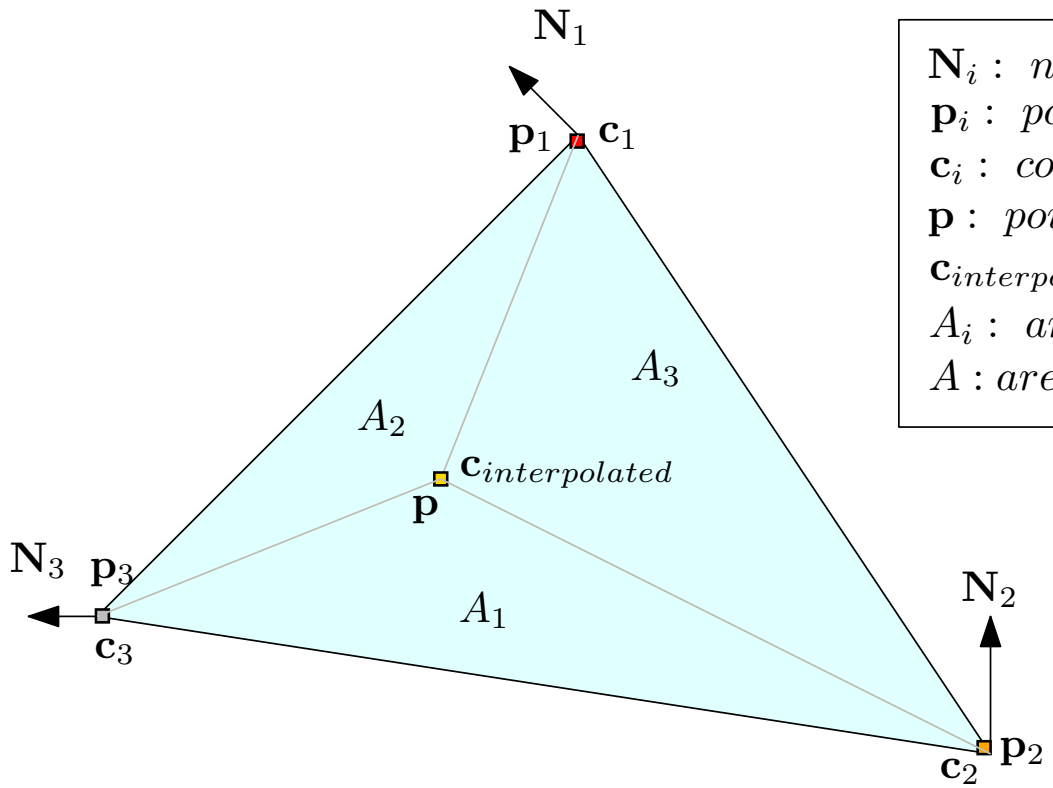
$\mathbf{N}_i$: *normal at point* i
$\mathbf{p}_i$: *point* i
$\mathbf{c}_i$: *color at point* i *(Phong)*
$\mathbf{p}$: *point to be rendered*
$\mathbf{c}_{interpolated}$: *interpolated color*
A_i : *area of* $\triangle(\mathbf{p}, \forall \mathbf{p}_{j \neq i})$
A : *area of* $\triangle(\mathbf{p}_1,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

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

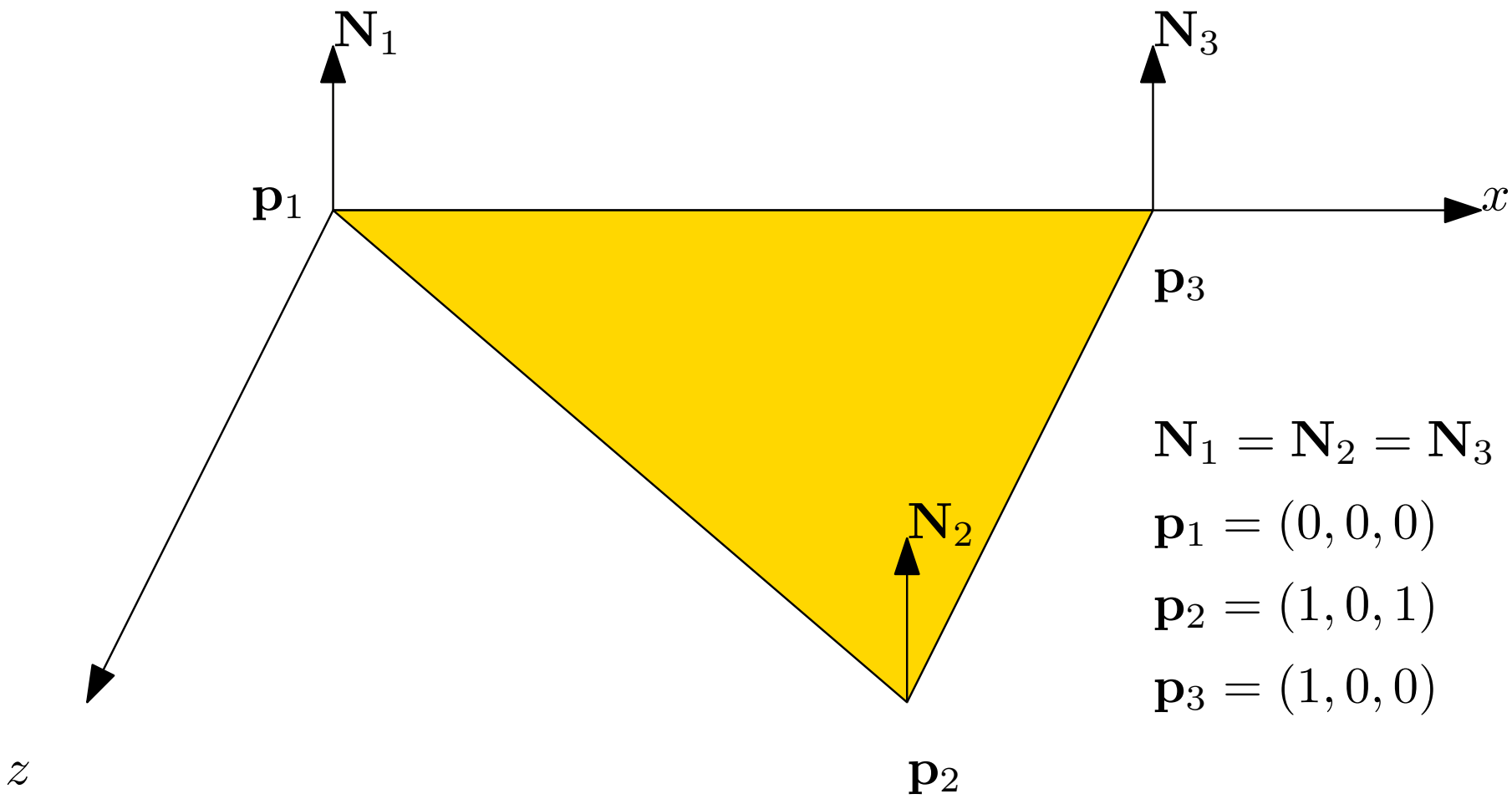


$\mathbf{N}_i$: *normal at point* i
$\mathbf{p}_i$: *point* i
$\mathbf{c}_i$: *color at point* i *(Phong)*
$\mathbf{p}$: *point to be rendered*
$\mathbf{c}_{interpolated}$: *interpolated color*
A_i : *area of* $\triangle(\mathbf{p}, \forall \mathbf{p}_{j \neq i})$
A : *area of* $\triangle(\mathbf{p}_1,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

Gouraud Shading

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

구로(Gouraud) 세이딩 예제



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

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

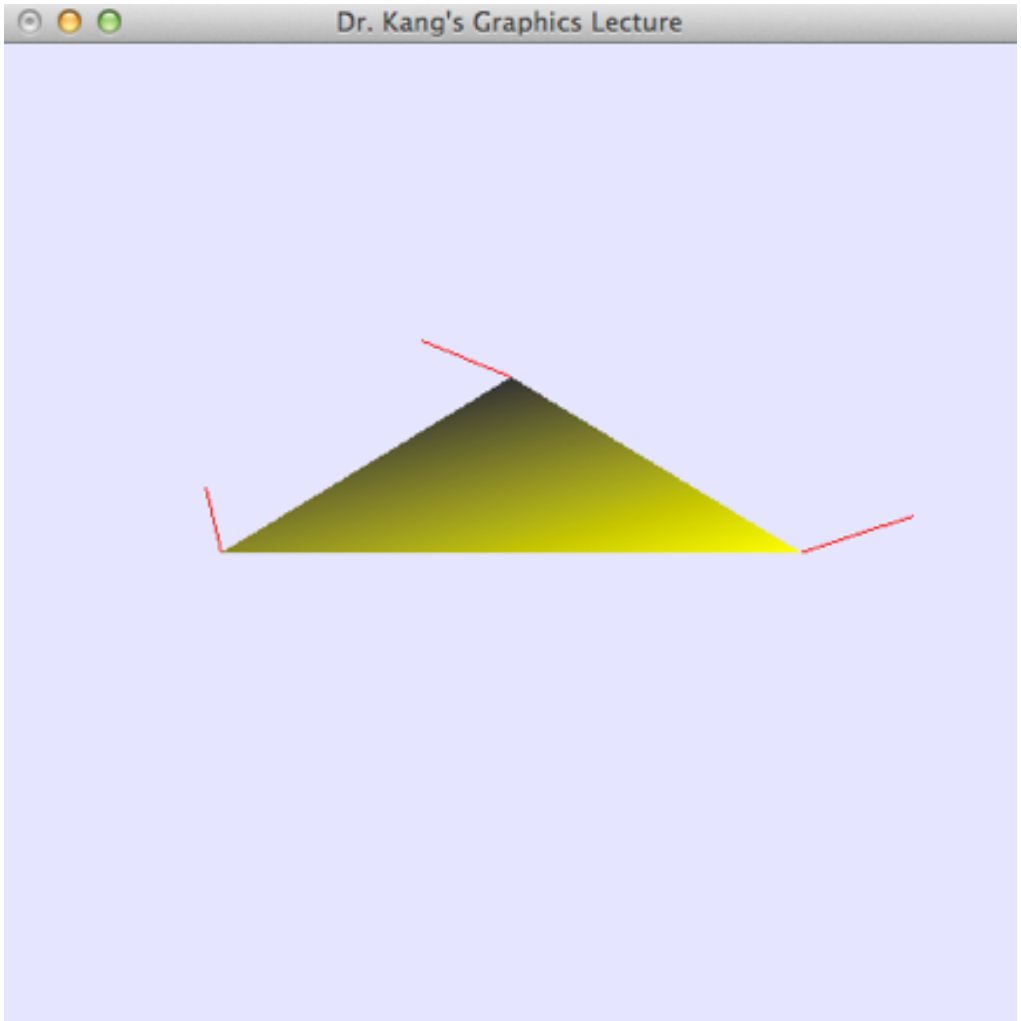
$\mathbf{p}_2 = (1, 0, 1)$

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

구로 세이딩을 위한 코딩

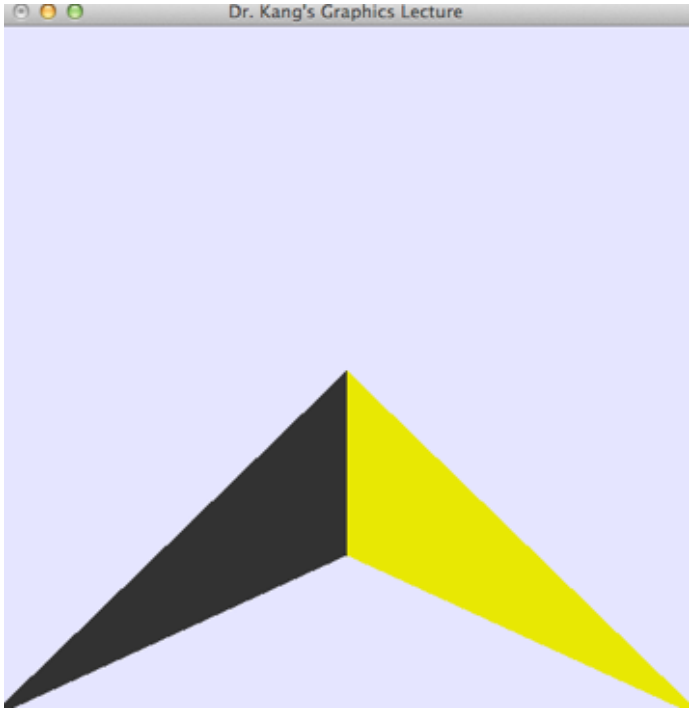
```
sqrt3 = math.sqrt(3)
glBegin(GL_TRIANGLES);
glNormal3f(0,1,0)
glVertex3f(0,0,0)
glNormal3f(2/sqrt3,1/sqrt3,0)
glVertex3f(2,0,0)
glNormal3f(-2/sqrt3,1/sqrt3,0)
glVertex3f(1,0,-1)
glEnd()
```

구로 세이딩 결과



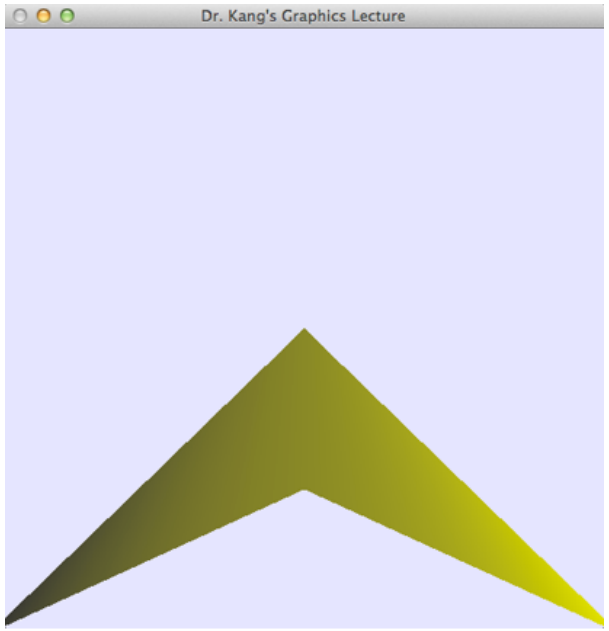
구로 세이딩 - 두 개의 인접한 면 그리기

```
sqrt2 = math.sqrt(2)
glBegin(GL_TRIANGLES)
glNormal3f(-1/sqrt2,1/sqrt2,0)
glVertex3f(0,1,0)
glVertex3f(-1,0,0)
glVertex3f(0,1,1)
glNormal3f(1/sqrt2,1/sqrt2,0)
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/sqrt2,1/sqrt2,0)
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/sqrt2,1/sqrt2,0)
glVertex3f(1,0,0)
glEnd()
```



메시(mesh)

Table 1: 메시 데이터 포맷의 예시

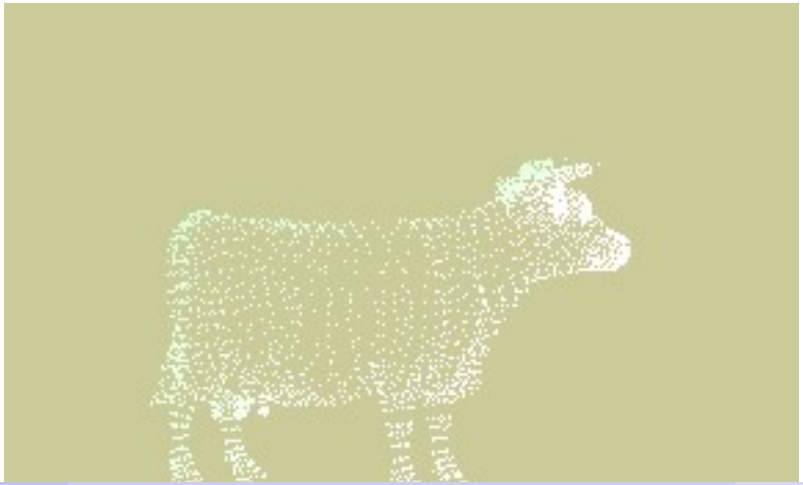
포맷	실제 데이터 예시
numVertices n	4
vertex 1 (x_1, y_1, z_1)	0.0 1.0 0.0
vertex 2 (x_2, y_2, z_2)	-1.0 0.0 0.0
...	0.0 1.0 1.0
vertex n (x_n, y_n, z_n)	1.0 0.0 0.0
numFaces m	2
face 1: no. of indices $(f_1.v_1, f_1.v_2, f_1.v_3)$	3 0 1 2
face 2: no. of indices $(f_2.v_1, f_2.v_2, f_2.v_3)$	3 0 2 3
...	

메시(mesh) 로딩과 그리기

```
class MeshLoader():
    def __init__(self):
        self.nV = 0
        self.nF = 0

    def load(self, filename):
        with open(filename, "rt") as meshdata:
            self.nV = int(next(meshdata))
            print('total number of vertices = ', self.nV)
            self.verts = np.zeros(shape = (self.nV * 3, ), dtype = 'f')

            for i in range(self.nV) :
                start = i*3
                self.verts[start: start+3] = next(meshdata).split()

    def draw(self):
        glBegin(GL_POINTS)
        for i in range(self.nV):
            idx = i*3
            glVertex3fv(self.verts[idx: idx+3])
        glEnd()
```

메시(mesh) 로드와 그리기 호출

```
class MyGLWidget(QOpenGLWidget):

    def __init__(self, parent=None):
        super(MyGLWidget, self).__init__(parent)
        self.mesh = MeshLoader()
        self.mesh.load('./Lighting/mesh.txt')
        self.angle = 0.0
        self.lightx = 0.0

    ...
    def paintGL(self):
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
        ...

        self.mesh.draw()

        # 그려진 프레임버퍼를 화면으로 송출
        glFlush()
```

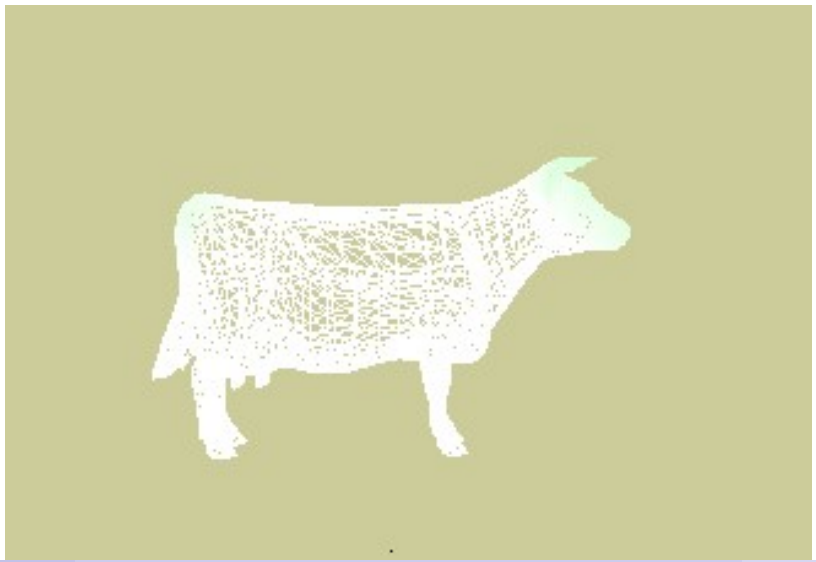


메시(mesh) 그리기 2 - Wireframe

```
class MeshLoader():
    def __init__(self):
        self.nV = 0
        self.nF = 0

    def load(self, filename):
        with open(filename, "rt") as meshdata:
            self.nV = int(next(meshdata))
            print('total number of vertices = ', self.nV)
            self.verts = np.zeros(shape = (self.nV * 3, ), dtype = 'f')

            for i in range(self.nV) :
                start = i*3
                self.verts[start: start+3] = next(meshdata).split()

            self.nF = int(next(meshdata))
            print('total number of faces = ', self.nF)
            self.faces = np.zeros(shape = (self.nF * 3, ), dtype = 'i')

            for i in range(self.nF) :
                start = i*3
                self.faces[start: start+3] = next(meshdata).split()[1:4]
```

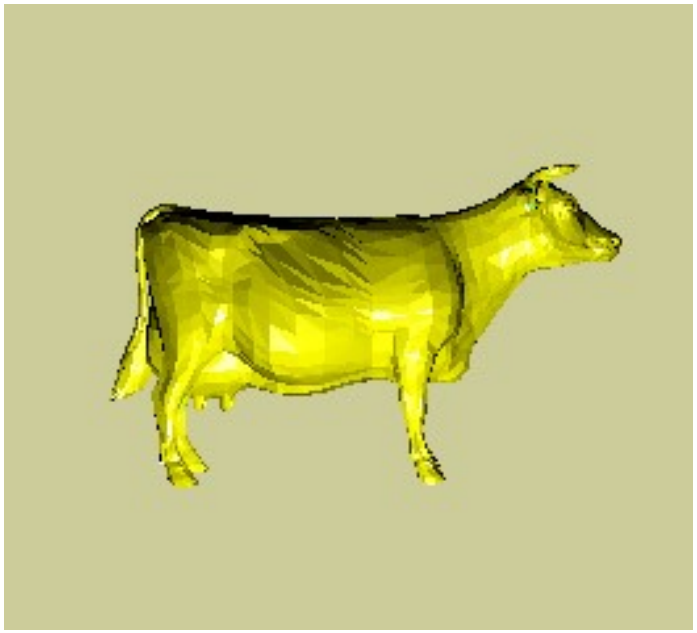
메시(mesh) 그리기 2 - Wireframe

```
    def draw(self):
        glBegin(GL_POINTS)
        for i in range(self.nV):
            idx = i*3
            glVertex3fv(self.verts[idx: idx+3])
        glEnd()


        for i in range(self.nF):
            idx = i*3
            i0, i1, i2 = self.faces[idx: idx+3]
            glBegin(GL_LINE_LOOP)
            glVertex3fv(self.verts[i0*3:i0*3+3])
            glVertex3fv(self.verts[i1*3:i1*3+3])
            glVertex3fv(self.verts[i2*3:i2*3+3])
            glEnd()
```



메시(mesh) 그리기 3 - 면 그리기

```
    def draw(self):

        glBegin(GL_TRIANGLES)
        for i in range(self.nF):
            idx = i*3
            i0, i1, i2 = self.faces[idx: idx+3]
            v0, v1, v2 = self.verts[i0*3:i0*3+3], self.verts[i1*3:i1*3+3],
     self.verts[i2*3:i2*3+3]
            N = computeNormal(v0, v1, v2)
            glNormal3fv(N)
            glVertex3fv(v0)
            glVertex3fv(v1)
            glVertex3fv(v2)
        glEnd()
```



메시(mesh) 그리기 4 - 부드러운 면

```
class MeshLoader():
    def __init__(self):
        self.nV = 0
        self.nF = 0

    def load(self, filename):
        with open(filename, "rt") as meshdata:
            self.nV = int(next(meshdata))
            print('total number of vertices = ', self.nV)

            self.verts = np.zeros(shape = (self.nV * 3, ), dtype = 'f')
            self.norms = np.zeros(shape = (self.nV * 3, ), dtype = 'f')
    ######

            for i in range(self.nV) :
                start = i*3
                self.verts[start: start+3] = next(meshdata).split()

            self.nF = int(next(meshdata))
            print('total number of faces = ', self.nF)
            self.faces = np.zeros(shape = (self.nF * 3, ), dtype = 'i')

            for i in range(self.nF) :
                start = i*3
                self.faces[start: start+3] = next(meshdata).split()[1:4]

            self.setNormals() ########
```

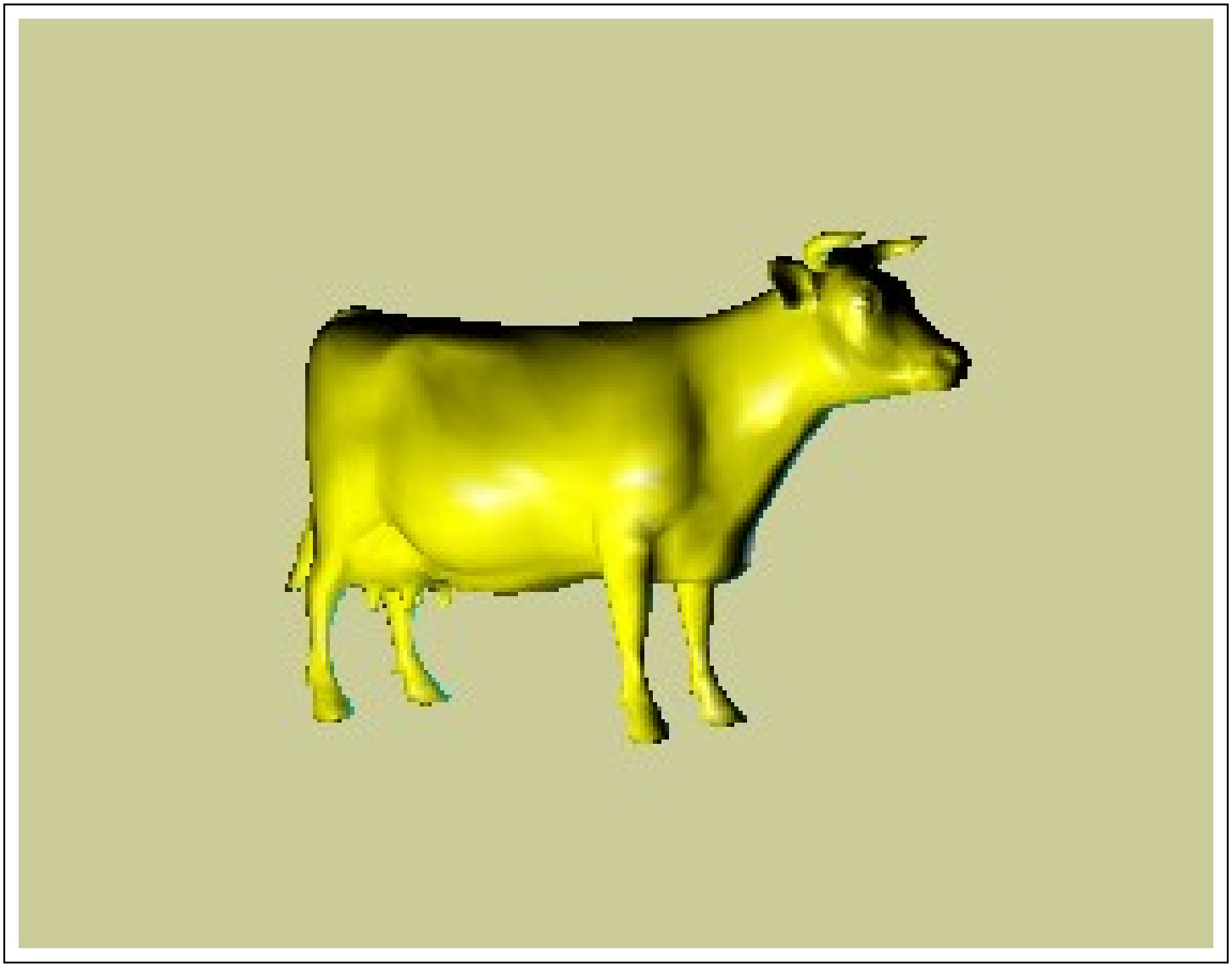
메시(mesh) 그리기 4 - 부드러운 면

```
    def setNormals(self):
        # calculate normals for each face
        for i in range(self.nF):
            i0, i1, i2 = self.faces[i * 3:i * 3 + 3]
            p0 = self.verts[i0 * 3:i0 * 3 + 3]
            p1 = self.verts[i1 * 3:i1 * 3 + 3]
            p2 = self.verts[i2 * 3:i2 * 3 + 3]
            N = computeNormal(p0, p1, p2)
            # normals are accumulated in the vertices in the face
            self.norms[i0 * 3:i0 * 3 + 3] += N
            self.norms[i1 * 3:i1 * 3 + 3] += N
            self.norms[i2 * 3:i2 * 3 + 3] += N
        # normal of each vertex is normalized
        for i in range(self.nV):
            length = np.linalg.norm(self.norms[i * 3: i * 3 + 3])
            self.norms[i*3: i*3+3] /= length
```

메시(mesh) 그리기 4 - 부드러운 면

```
    def draw(self):

        glBegin(GL_TRIANGLES)
        for i in range(self.nF):
            idx = i*3
            i0, i1, i2 = self.faces[idx: idx+3]
            v0, v1, v2 = self.verts[i0*3:i0*3+3], self.verts[i1*3:i1*3+3],
     self.verts[i2*3:i2*3+3]
            n0, n1, n2 = self.norms[i0*3:i0*3+3], self.norms[i1*3:i1*3+3],
     self.norms[i2*3:i2*3+3]
            glNormal3fv(n0)
            glVertex3fv(v0)
            glNormal3fv(n1)
            glVertex3fv(v1)
            glNormal3fv(n2)
            glVertex3fv(v2)
        glEnd()
```

메시(mesh) 면 부드럽게 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) 애니메이션

```
class MyGLWidget(QOpenGLWidget):

    def __init__(self, parent=None):
        super(MyGLWidget, self).__init__(parent)
        self.mesh = MeshLoader()
        self.mesh.load('./Lighting/mesh.txt')
        ...
        self.objRotation = 0

        ########################################
        self._timer = QBasicTimer()                    # 타이머 생성
        self._timer.start(int(1000 / 60), self)        # 초당 60 프레임

    def paintGL(self):
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
        ...

        glPushMatrix()
        glRotatef(self.objRotation, 1, 1, 1)
        self.mesh.draw()
        glPopMatrix()
        ...

    def timerEvent(self, QTimerEvent):
        self.objRotation += 1
        self.update()
```

매우 느린 애니메이션을 보게 될 것임

메시(mesh) 애니메이션 고속화

```
    def draw_fast(self):
        glEnableClientState(GL_VERTEX_ARRAY)
        glEnableClientState(GL_NORMAL_ARRAY)
        glVertexPointer(3, GL_FLOAT, 0, self.verts)
        glNormalPointer(GL_FLOAT, 0, self.norms)
        glDrawElements(GL_TRIANGLES, self.nF * 3, GL_UNSIGNED_INT, self.faces
    )
```

한층 빨라진 애니메이션 관찰 가능 - 메시 데이터 처리의 고속화