Computer Graphics

Prof. Jibum Kim

Department of Computer Science & Engineering

Incheon National University

- **Computing normal vectors**

- **The normal vector at each vertex is simply normal vector to the plane itself containing the surface**
- **How does one determine the normal direction to a plane p?**
- **Two vectors u and v are known to lie on p, then the cross-product n=uxv is normal to p**
- **In this figure, $u=P_1-P_0$ and $v=P_4-P_0$, $n=uxv$**

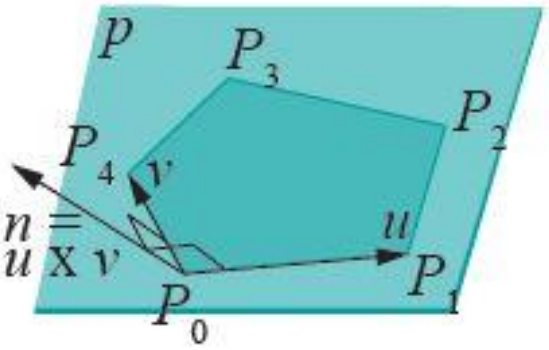
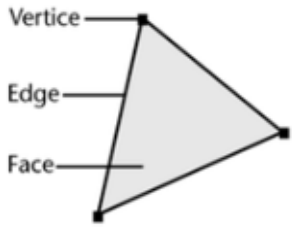


Figure 11.52: Vector n is normal to the plane p.

- **Nomalizing normal vectors**
- **Normalizing a vector means dividing it by its magnitude to obtain a vector with the same direction but of unit length**
- **It is important to specify normalized normal vectors because OpenGL uses the dot product to compute the cosine of the angle between two vectors, which is correct if they are both of unit length**
- **normalize - OpenGL 4 Reference Pages (khronos.org)**
- **glm::normalize**

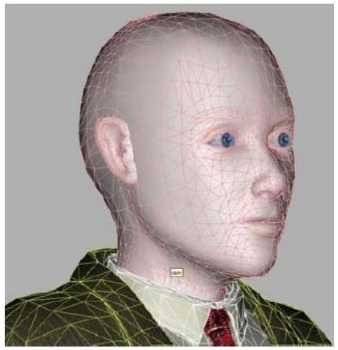
- **Polygonal mesh**

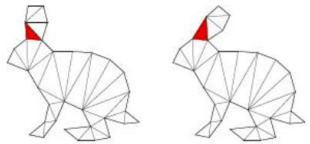
- **Polygon: polygons are straight-sided shapes (3 or more sides), defined by vertices and the straight lines that connect them (edges)**
- **Face: polygon의 내부 영역**
- **Polygon의 기본 요소: vertices, edges, face**
- **Polygon은 가장 단순하면서 더 많이 사용되는 2D 물체임**
- **우리가 사용하는 polygon은 simple and planar polygon임**



- How to compute the normal vector for a polygonal mesh?
- (Polygonal) mesh are simply collection of polygons, or faces that fit together to form the skin of the object
- 현재 컴퓨터 그래픽스에서 solid shape을 표현하는 표준 적인 방법이다
- 물체가 solid하다는 것의 의미는?
- The object is considered to be solid of the polygonal faces fit together to enclose space
-

Mesh examples:

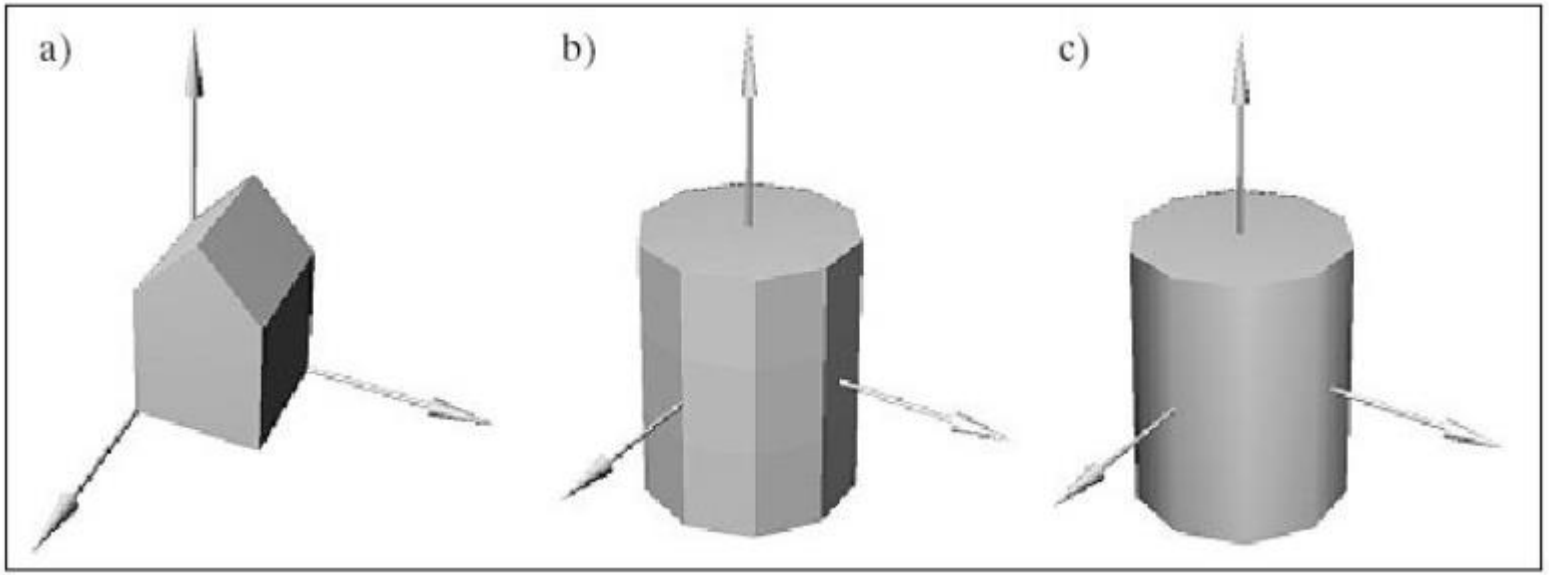




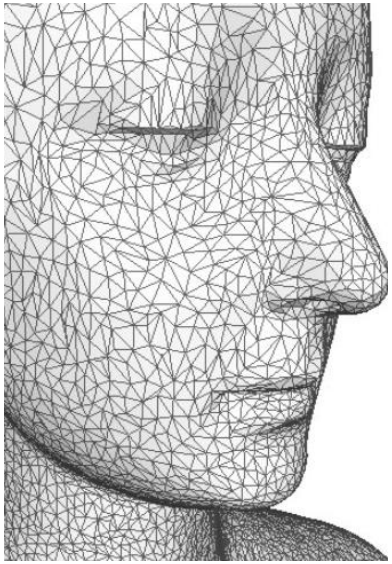
- **A polygonal mesh is given by a list of polygons, along with information about the direction in which each polygon is facing**

- **The important directional information is often simply the outward-pointing normal vector to the plane of the face**

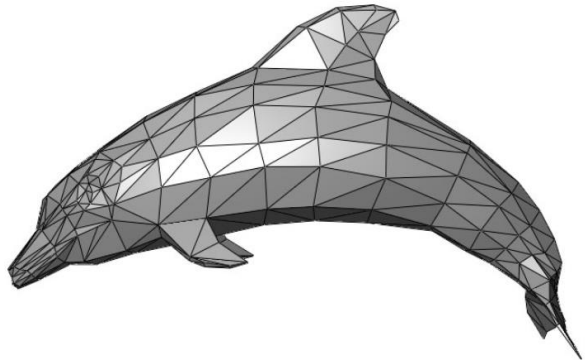
- 일부 object는 polygonal mesh로 완벽히 표현 가능
- 예: flat한 face를 갖는 물체 (a)
- 근사적으로 밖에 표현할 수 없는 objects들도 있음(b, c)



- **컴퓨터 그래픽스에서 사용하는 가장 일반적인 mesh는 triangular mesh로 모든 face가 삼각형들인 mesh를 의미한다**



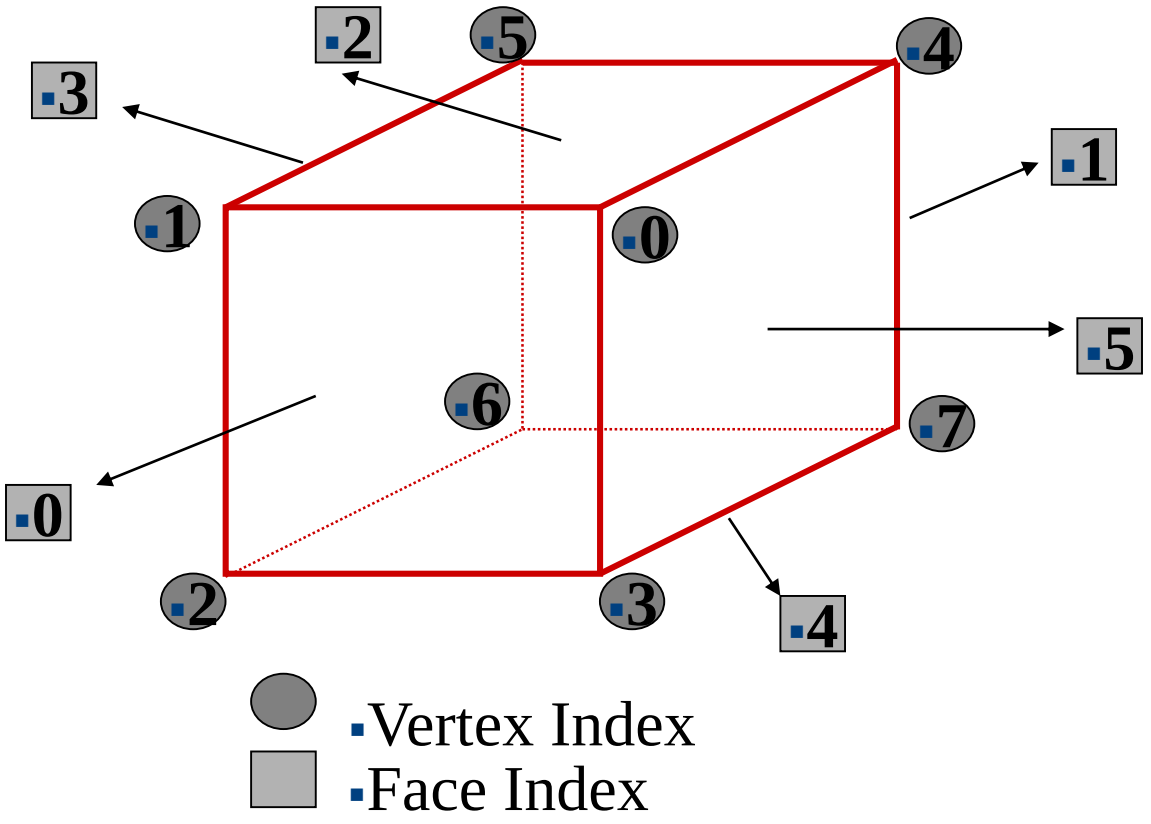
Triangular mesh로 모델링한 사람 얼굴



Triangular mesh로 모델링한 돌고래

- **Mesh example**

- **Cube mesh: it has 8 vertices and 6 polygonal faces**



- Data representation using vertex, face and normal lists:
- Vertex list: 각 vertex의 (x, y, z) coordinates
- Normal list: 각 face의 normal vector 방향 (face 별로 혹은 vertex 별로)
- Polygon list: 각 polygon (face)이 어떤 verte로 이루어져 있는가? Orientation 정함

Vertex List

30	30	30
-30	30	30
-30	-30	30
30	-30	30
30	30	-30
-30	30	-30
-30	-30	-30
30	-30	-30

Normal List

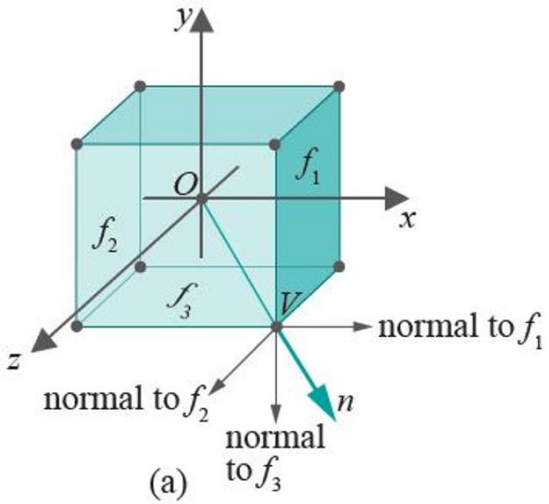
0.0	0.0	1.0
0.0	0.0	-1.0
0.0	1.0	0.0
-1.0	0.0	0.0
0.0	-1.0	0.0
1.0	0.0	0.0

Polygon List

0	1	2	3
4	7	6	5
0	4	5	1
1	5	6	2
2	6	7	3
3	7	4	0

- **Normal vector for polygonal meshes**

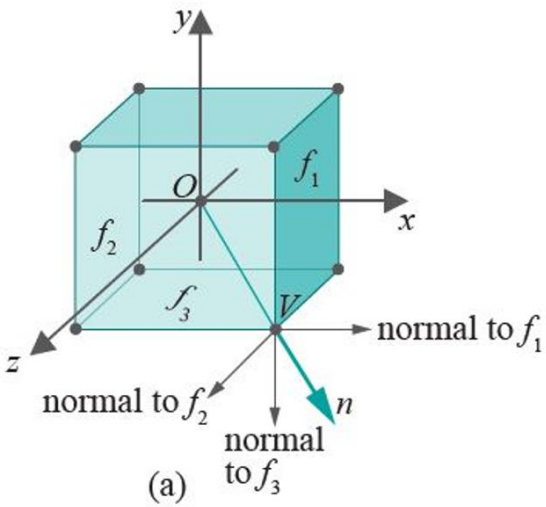
- The one method is to specify the normal at each vertex as an average of incident face normal
- The unit outward normal vectors to f_1, f_2, and f_3, the three faces which meet at V, are [1 0 0], [0 0 1], and [0 -1 0], respectively
- The average is [1/3 -1/3 1/3]
- After normalization, $\left[\frac{1}{\sqrt{3}} -\frac{1}{\sqrt{3}} \frac{1}{\sqrt{3}}\right]$



Chapter11/SphereinBox1 예제

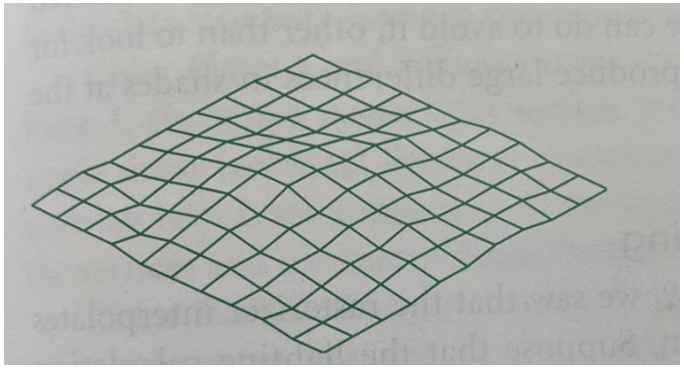
```
#define ONE_BY_ROOT_THREE 0.57735

// Box vertex normal vectors = normalized unit vector pointing from origin to vertex
static float normals[] =
{
    ONE_BY_ROOT_THREE, -ONE_BY_ROOT_THREE, ONE_BY_ROOT_THREE,
    ONE_BY_ROOT_THREE, ONE_BY_ROOT_THREE, ONE_BY_ROOT_THREE,
    ONE_BY_ROOT_THREE, ONE_BY_ROOT_THREE, -ONE_BY_ROOT_THREE,
    ONE_BY_ROOT_THREE, -ONE_BY_ROOT_THREE, -ONE_BY_ROOT_THREE,
    -ONE_BY_ROOT_THREE, -ONE_BY_ROOT_THREE, ONE_BY_ROOT_THREE,
    -ONE_BY_ROOT_THREE, ONE_BY_ROOT_THREE, ONE_BY_ROOT_THREE,
    -ONE_BY_ROOT_THREE, ONE_BY_ROOT_THREE, -ONE_BY_ROOT_THREE,
    -ONE_BY_ROOT_THREE, -ONE_BY_ROOT_THREE, -ONE_BY_ROOT_THREE
};
```



- **Polygonal shading**

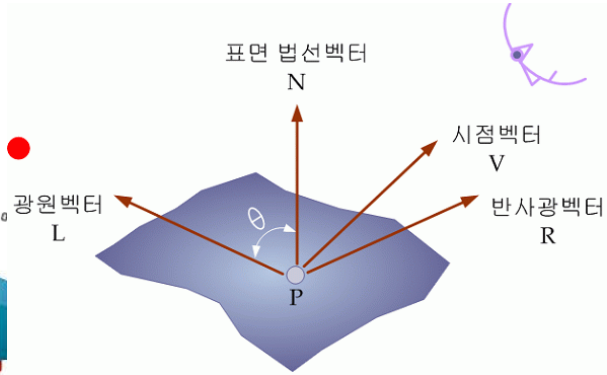
- The lighting models that we have developed can be applied at every point on a surface. However, the amount of computation required can be large
- Most graphics systems exploit the efficiencies possible for rendering flat polygons by decomposing curved surfaces into many small, flat polygons
- Consider a polygonal mesh, where each polygon is flat and thus has a well-defined normal vector. We consider three ways to shae the polygons: flat shading, smoothing (Gouraud shading) and Phong shading

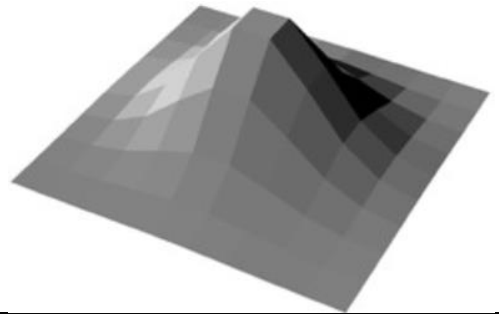


Polygonal mesh

- **Flat shading**

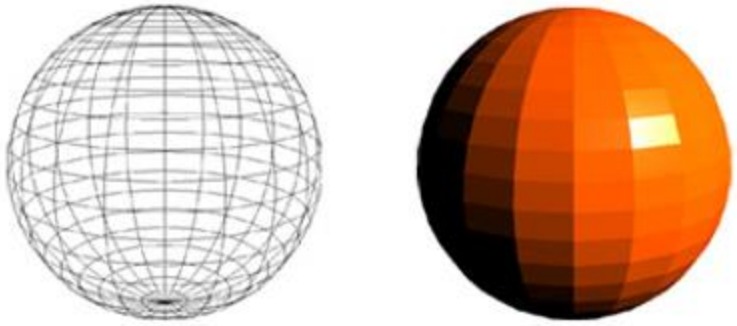
- For a flat polygon, the normal vector N is constant
- If we assume a distant viewer, view vector (V) is constant over the polygon.
- If the light source is distant, L is constant
- If the three vectors constant, the shading calculation need to be carried out only once for each polygon, and each point on the polygon is assigned the same shade.
- This techniques is known as flat shading





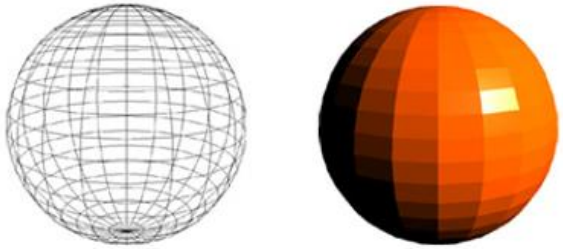
- **Flat shading의 예**
- **1. 주어진 polygon에 대하여 polygon 내부의 중심점 (centroid)를 구한다**
- **(polygon을 구성하는 vertex의 위치를 평균하여 계산)**
- **2. 이 중심점에서 normal vector (법선 벡터), 광원 벡터, 시점 벡터를 계산한다.**
- **3. 앞에서 배운 Phong 조명 모델을 사용하여 이 vertex에서의 reflection을 계산하여 색을 정한다**
- **4. 이 색으로 이 polygon면 내부를 모두 채운다**

- OpenGL에서의 flat shading 사용방법
- glShadeModel(GL_FLAT);



- Flat shading은 Mach Band Effect 라고 불리는 현상이 생긴다
- 밑의 그림을 보면 각각의 사각형 내부는 완전히 동일한 색으로 별도의 경계선은 그리지 않았다. 하지만, 경계선이 있는 것 처럼 보인다

이 그림을 보면,
마치 각 영역의 경계선에 무언가 선이 그려져 있는 것처럼 느껴진다.
이 것이 마하 밴드 효과이다~!



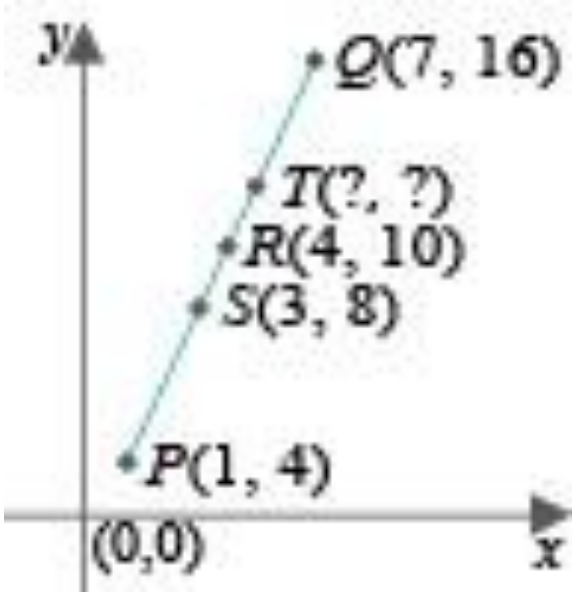
Flat shading의 예

- **https://www.dropbox.com/s/2gnmwnqsagnokg0/flat_shading.txt?dl=0**

- 두번째 Shading 방법은 Gouraud shading 이다
- 이 shading 방법은 양방향 선형보간 (bilinear interpolation)방법을 사용한다
- Bilinear interpolation 방법은 barycentric coordinates을 사용한다. 배우기 전에 먼저 barycentric coordinates에 대해서 배워보자

- **Barycentric coordinates**

- 두 점 P(1, 4)와 Q(7, 16)으로 이루어진 선분에 대해서 생각해 보자
- 이 선분 사이의 임의의 점 x 는 다음과 같은 식으로 표현 가능한가?
- $x(t)=(1-t)P+tQ,\ 0 \le t \le 1$



- 예: t=1이면 어느 점인가? Q
- 예: t=0이면 어느 점인가? P
- 예: t=0.5이면 어느 점인가? R
- 예: t=1/3 이면 어느 점인가? S

- Q) $0 \le t < 0.5$이면 오른쪽 선분에서 어디를 의미?

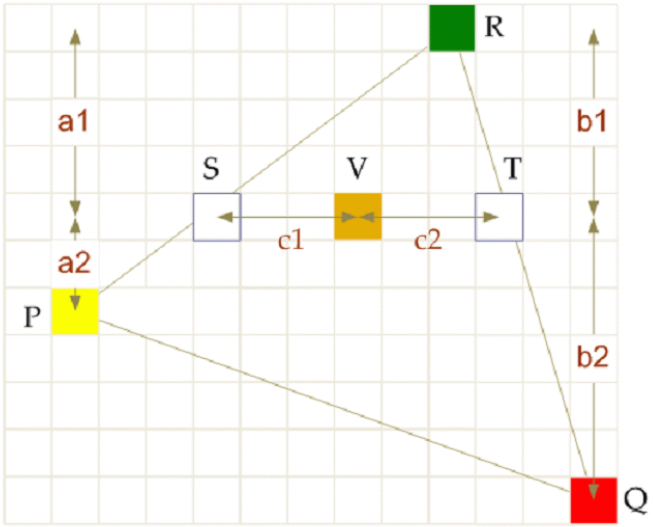
- 두 점 P와 Q로 이루어진 선분이 있을 때 이 두 점 사이에 있는 선분 위의 모든 점은 다음과 같이 표현 가능하다
- x= α *P + β*Q
- (단, $\alpha + \beta = 1$, $0 \leq \alpha, \beta \leq 1$)

- 이때 (α, β)를 무게 중심 좌표, barycentric coordinates라고 한다
- 점 x가 P에 가까워 질 수록 α는 1에 가까워 짐
- 점 x가 Q에 가까워 질 수록 β는 1에 가까워 짐

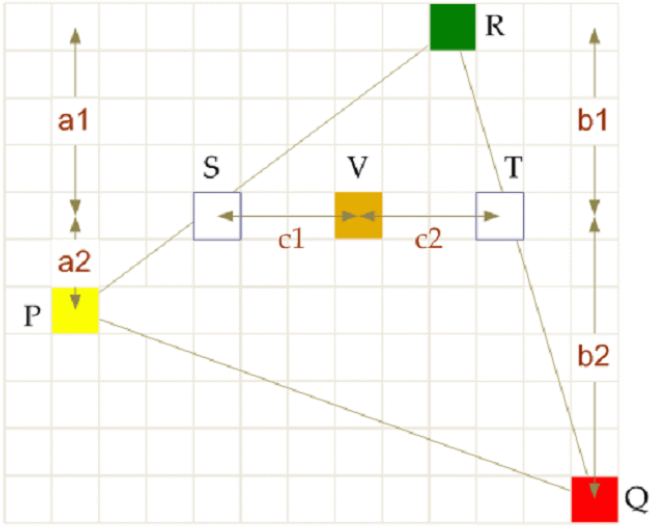
- 선분 PQ 위의 점 V의 무게 중심 좌표
- $V = \alpha P + \beta Q$, 단, $0 \le \alpha, \beta \le 1, \alpha + \beta = 1$
- α 는 P의 영향력, β는 Q의 영향력.
- V가 P에서 멀어질수록 α 는 줄어들어야 함, 이만큼 β는 늘어나야 한다
- 이를 이용, 가중치 비율을 선분의 길이 비율로 표현 가능
- $\alpha : \beta = |VQ| : |VP|$, 즉, $\alpha = \frac{|VQ|}{|VQ|+|VP|} = \frac{|VQ|}{|PQ|}, \beta = \frac{|VP|}{|VQ|+|VP|} = \frac{|VP|}{|PQ|}$

- **Bilinear interpolation (양방향 선형보간)**

- 세 개의 vertex P, Q, R로 이루어지는 polygon에서 P, Q, R의 색이 주어진 경우 polygon 내부의 점 (예: V)에서의 색을 보간하고자 한다
- 이 경우 무게 중심 좌표와 양방향 선형 보간을 이용하여 polygon 내부의 색을 보간할 수 있다



- 양방향 선형 보간
- P, Q, R의 색이 주어짐. 목적: Polygon 내부 V의 색을 보간
- 1. Y 방향 보간
- 1) P와 R을 이용하여 S의 색을 보간
- 2) R과 Q를 이용하여 T의 색을 보간

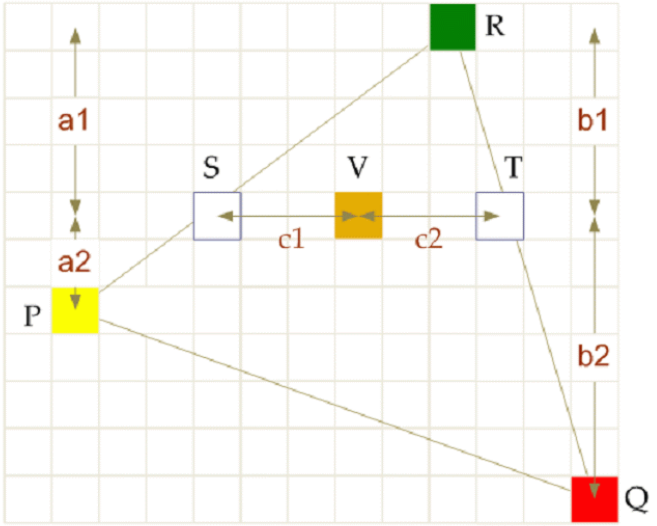


- 2. X 방향 보간
- S와 T를 이용하여 V의 색을 보간

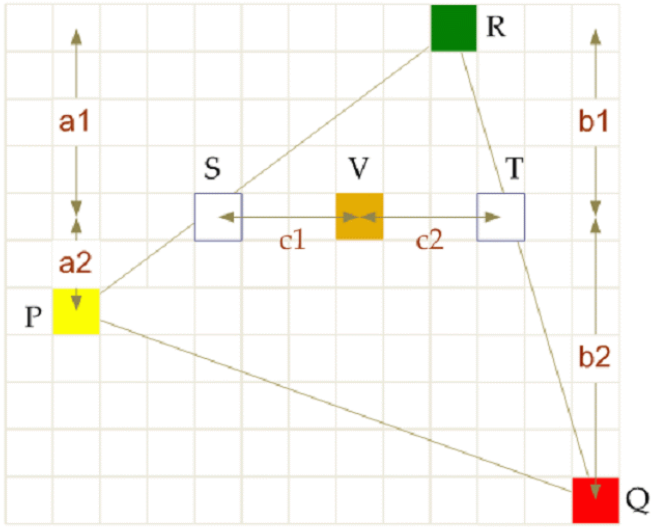
- 1. Y 방향 보간
- 1) P와 R을 이용하여 S의 색을 보간

- 닮은 꼴 삼각형을 이용

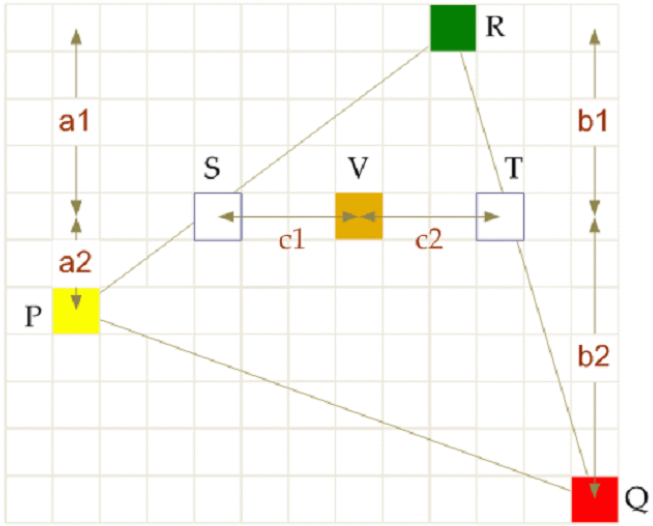
- $S(\text{의색}) = \frac{a1}{a1+a2} P(\text{색}) + \frac{a2}{a1+a2} R(\text{색})$



- 1. Y 방향 보간
- 2) R과 Q를 이용하여 T의 색을 보간
- 닮은 꼴 삼각형을 이용
- $T(\text{의색}) = \frac{b1}{b1+b2} Q(\text{색}) + \frac{b2}{b1+b2} R(\text{색})$



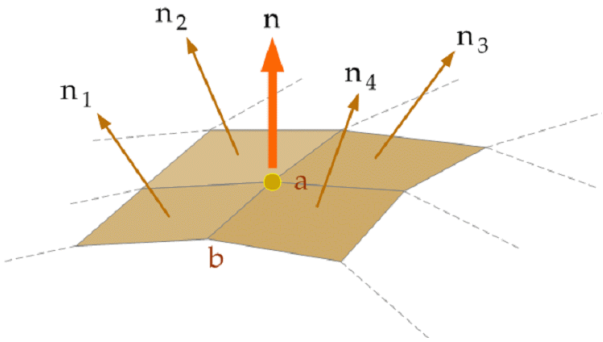
- **2. S와 T의 보간된 색을 이용 V의 색 보간**
- $\boldsymbol{V}(\text{의색}) = \frac{\boldsymbol{c2}}{\boldsymbol{c1}+\boldsymbol{c2}}\boldsymbol{S}(\text{색}) + \frac{\boldsymbol{c1}}{\boldsymbol{c1}+\boldsymbol{c2}}\boldsymbol{T}(\text{색})$



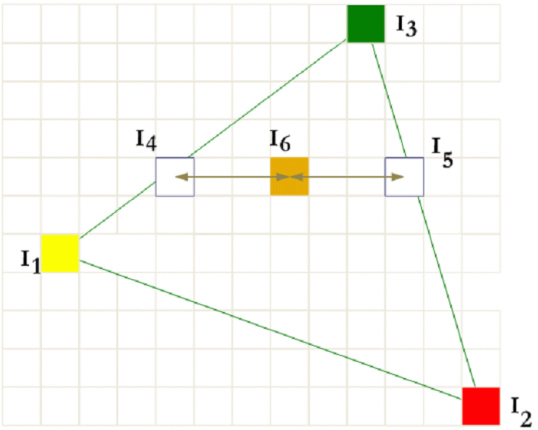
- **Smooth (Gouraud) shading**

- **Because multiple polygons meet at interior vertices of the mesh, each of which has its own normal, the normal at the vertex is discontinuous**
- **Gouraud realized that the normal at the vertex could be defined in such a way as to achieve smoother shading through interpolation**
- **In Gouraud shading, we define the normal at a vertex to be the normalized average of the normal of the polygons that share the vertex. For our example, the vertex normal is given by**

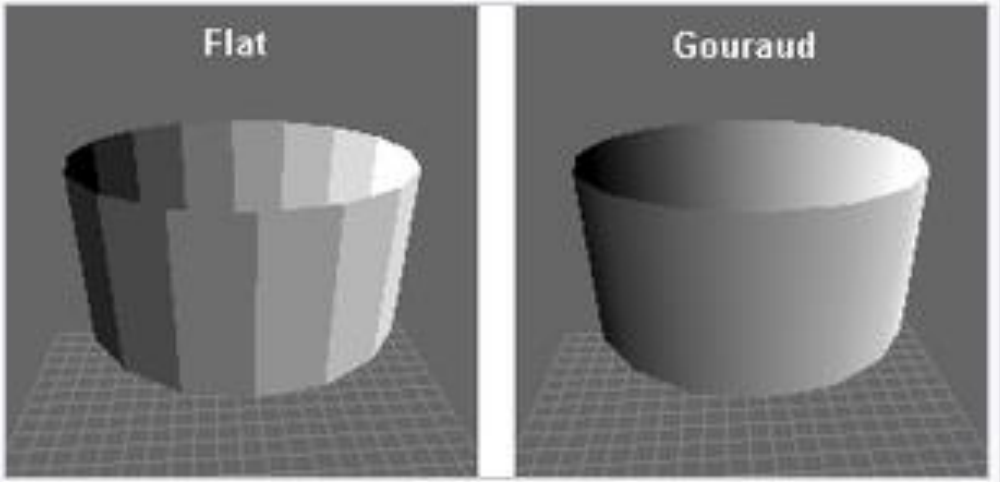
$$n = \frac{n_1 + n_2 + n_3 + n_4}{|n_1 + n_2 + n_3 + n_4|}$$



- 각 vertex의 법선 벡터를 구하고 그 vertex에서의 색이 계산되면 polygon 내부의 색을 양방향 선형 보간으로 계산한다
- 이렇게 양방향 선형 보간을 사용하면 polygon 내부 화소들의 색이 서서히 변하게 된다



- **Gouraud shading에서 다각형 내부의 음영이 보다 부드럽게 이어진다. 하지만, 단점으로는 선형 보간 계산 때문에 flat shading 보다 오랜 시간이 필요하다**



Comparison of flat shading and Gouraud shading.

- OpenGL에서의 shading
- Gouraud shading 사용시
- glShadeModel(GL_SMOOTH);
- Flat shading 사용시
- glShadeModel(GL_flat);

- **glutSolidTeapot (1.0);**

- **https://www.dropbox.com/s/87pkyzn20e0xsnd/shading_1.txt?dl=0**

- **Phong shading (퐁 쉐이딩)**

- **Phong proposed that instead of interpolating vertex intensities, as we do in smooth shading, we interpolate normal across each polygon**
- **We can compute vertex normal by interpolating over the normal of the polygons that share the vertex**
- **Then, we can use interpolate the normal over the polygon**

- Phong shading 예: polygon의 세 vertex에서의 normal vector ($n1$, $n2$, $n3$)가 주어지고 $n6$보간

- 1. $n4$보간: $n4 = \frac{4}{6} * n1 + \frac{2}{6} * n3$

- 2. $n5$보간: $n4 = \frac{4}{10} * n2 + \frac{6}{10} * n3$

- 3. $n6$보간: $n6 = \frac{1}{2} * n4 + \frac{1}{2} * n5$