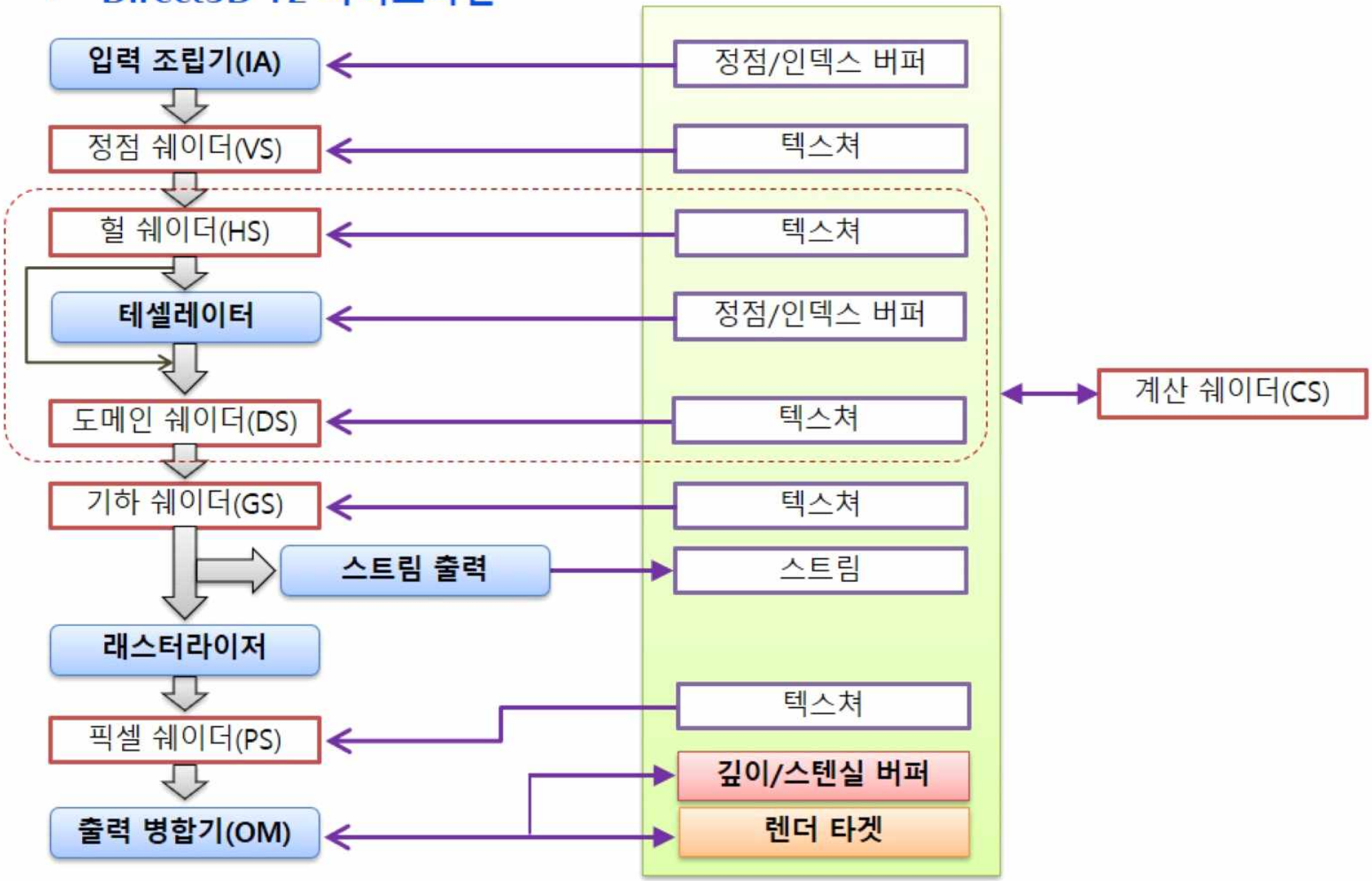
Game Programming with DirectX

Direct3D Graphics Pipeline
(Shader Samples)

Shading

Direct3D 파이프라인

- Direct3D 12 파이프라인



조명(Lighting)

- **정점의 조명 계산식(Light Equation)**

$$I = A + D + S + E$$

A (주변 광원 색상: Ambient Color)

$A = M_a * \{G_a + \sum L_a(i)\}$

M_a : 재질의 주변 광원 반사 색상

G_a : 전역 주변 광원의 색상

L_a : 정점에 영향을 주는 광원의 주변 광원 색상

D (확산 광원 색상: Diffuse Light)

$D = M_d * \sum\{L_d(i) * (n \bullet L_{dir}(i))\}$

M_d : 재질의 확산 반사 색상

L_{dir} : 입사광 방향 벡터(정점에서 광원으로)

L_d : 정점에 영향을 주는 광원의 확산 광원 색상

n : 정점의 법선 벡터

S (스펙큘러 색상: Specular Light)

$S = M_s * \sum\{L_s(i) * (n \bullet H(i))^P\}$

M_s : 재질의 스펙큘러 반사 색상

L_s : 정점에 영향을 주는 스펙큘러 광원

n : 정점의 법선 벡터

P : 재질의 스펙큘러 파워

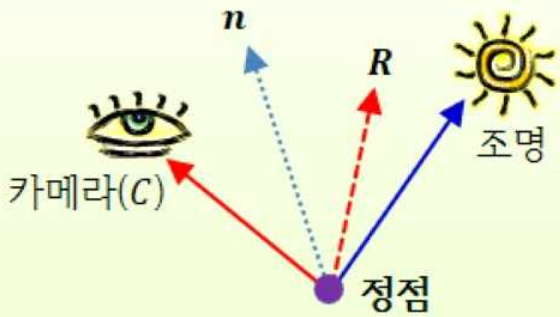
H : 중간 벡터(Halfway Vector)

$H = Normalize(Normalize(Camera - Vertex) + L_{dir})$

$H = Normalize((0,0,1) + L_{dir})$

E (Emissive Light)

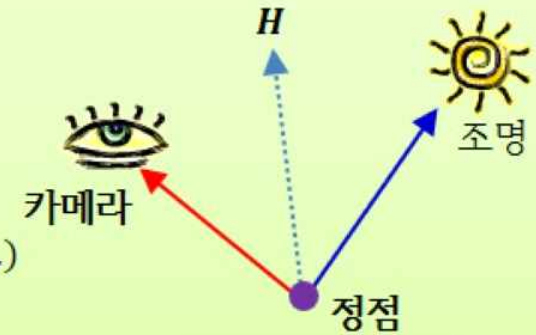
정점 자체에서 발산되는 색상



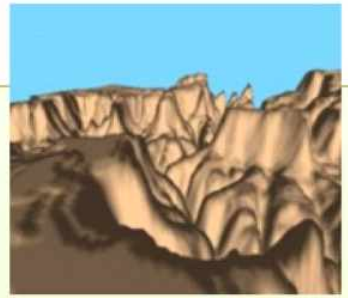
$$R = -C + 2(C \cdot n)n$$

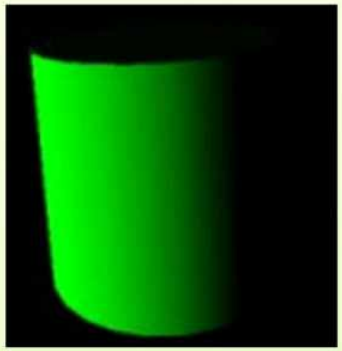
$$S = M_s * \sum\{L_s(i) * (L_{dir}(i) \bullet R)^P\}$$

Phong



Blinn-Phong





조명(Lighting)

- 점 조명(Point Lights)

```
struct LIGHT
{
    LIGHTTYPE m_nType;
    XMFLOAT4 m_xmf4Diffuse;
    XMFLOAT4 m_xmf4Specular;
    XMFLOAT4 m_xmf4Ambient;
    XMFLOAT3 m_xmf3Position;
    XMFLOAT3 m_xmf3Direction;
    float m_fRange;
    float m_fFalloff;
    float m_fAttenuation0;
    float m_fAttenuation1;
    float m_fAttenuation2;
    float m_fTheta;
    float m_fPhi;
};
```

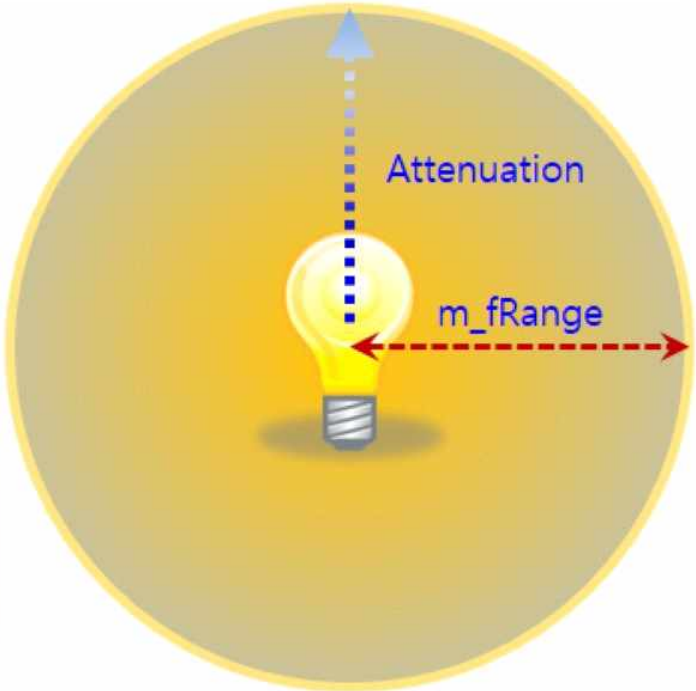
```
typedef enum _LIGHTTYPE
{
    LIGHT_POINT,
    LIGHT_SPOT,
    LIGHT_DIRECTIONAL
} LIGHTTYPE;
```

점 광원은 모든 방향으로 빛을 발산한다.

광원의 위치 벡터

$$I = A + D + S + E$$
$$A = M_a * \{G_a + \sum L_a(i) * L_{af}(i)\}$$
$$D = M_d * \sum[L_d(i) * \{\boldsymbol{n} \bullet \boldsymbol{L}_{dir}(i) * L_{af}(i)\}]$$
$$S = M_s * \sum[L_s(i) * (\boldsymbol{n} \bullet \boldsymbol{H}(i))^P * L_{af}(i)]$$

정점과 광원까지의 거리가 m_fRange 보다 크면 조명의 영향을 받지 않는다.



감쇠(Attenuation)
점 광원은 거리에 비례하여 빛의 양이 점차적으로 줄어든다.

D = 정점과 점 광원의 거리

$$L_{af}(i) = \frac{1}{Attenuation0 + (Attenuation1 * D) + (Attenuation2 * D^2)}$$

분모 $= (Attenuation0, Attenuation1, Attenuation2) \cdot (1, D, D^2)$

조명(Lighting)

- 스팟 조명(Spot Lights)

```
struct LIGHT
{
    LIGHTTYPE m_nType;
    XMFLOAT4 m_xmf4Diffuse;
    XMFLOAT4 m_xmf4Specular;
    XMFLOAT4 m_xmf4Ambient;
    XMFLOAT3 m_xmf3Position;
    XMFLOAT3 m_xmf3Direction;
    float m_fRange;
    float m_fFalloff;
    float m_fAttenuation0;
    float m_fAttenuation1;
    float m_fAttenuation2;
    float m_fTheta;
    float m_fPhi;
};
```

LIGHT_SPOT

스팟 광원은 한 방향으로 빛을 발산

광원의 위치 벡터

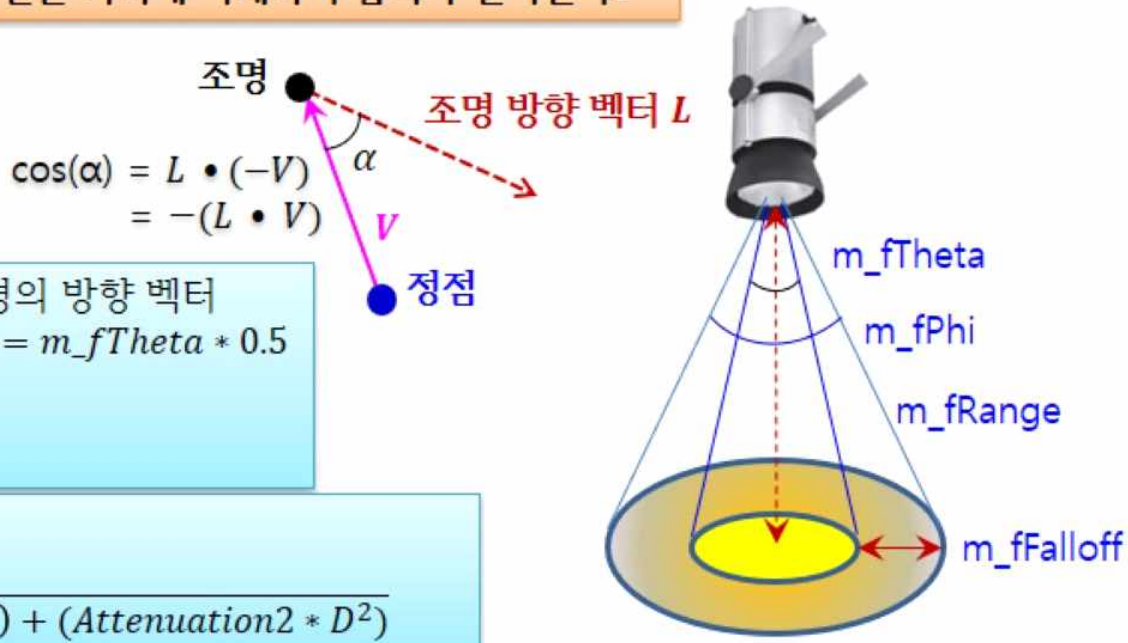
광원의 방향 벡터

$$I = A + D + S + E$$
$$A = M_a * \{G_a + \sum L_{a(i)} * L_{af}(i) * L_{sf}(i)\}$$
$$D = M_d * \sum[L_d(i) * \{\boldsymbol{n} \bullet \boldsymbol{L}_{dir}(i) * L_{af}(i) * L_{sf}(i)\}]$$
$$S = M_s * \sum[L_s(i) * (\boldsymbol{n} \bullet \boldsymbol{H}(i))^P * L_{af}(i) * L_{sf}(i)]$$

조명은 m_fRange 거리까지 영향을 준다.

안쪽 원의 끝에서 바깥 원 방향으로 감쇠(Falloff)가 일어난다.

스팟 광원은 거리에 비례하여 감쇠가 일어난다.



$$\cos(\alpha) = L \bullet (-V) = -(L \bullet V)$$

V = 정점에서 광원까지의 벡터, L = 스팟 조명의 방향 벡터
α = V와 L 사이의 각도, $\varphi = m\_fPhi * 0.5$, $\theta = m\_fTheta * 0.5$

$$L_{sf}(i) = \left\{\frac{cos(\alpha) - cos(\varphi)}{cos(\theta) - cos(\varphi)}\right\}^{m\_fFallOff}$$

D = 정점과 광원 사이의 거리

$$L_{af}(i) = \frac{1}{Attenuation0 + (Attenuation1 * D) + (Attenuation2 * D^2)}$$

조명(Lighting)

- 방향성 조명(Directional Lights)

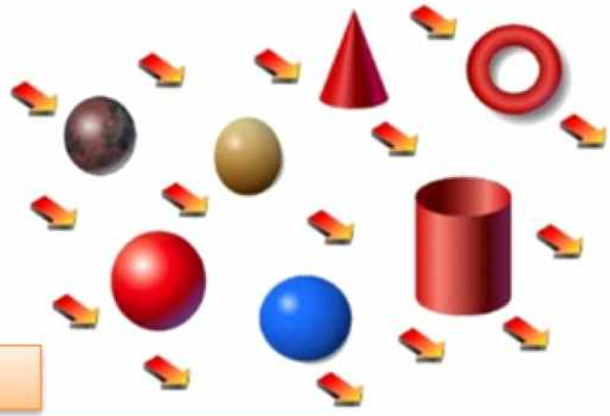
```
struct LIGHT
{
    LIGHTTYPE m_nType;
    XMFLOAT4 m_xmf4Diffuse;
    XMFLOAT4 m_xmf4Specular;
    XMFLOAT4 m_xmf4Ambient;
    XMFLOAT3 m_xmf3Position;
    XMFLOAT3 m_xmf3Direction;
    float m_fRange;
    float m_fFalloff;
    float m_fAttenuation0;
    float m_fAttenuation1;
    float m_fAttenuation2;
    float m_fTheta;
    float m_fPhi;
};
```

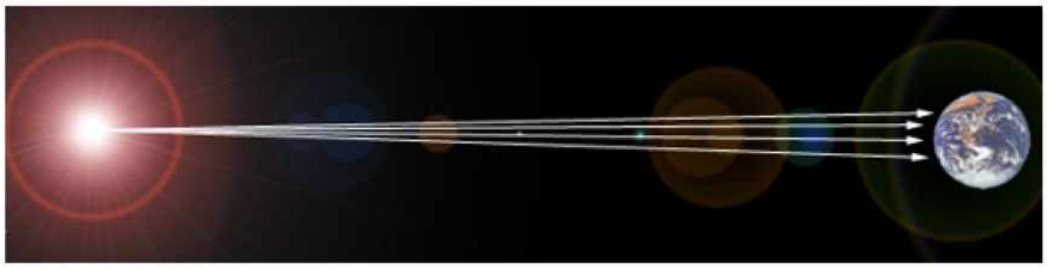
LIGHT_DIRECTIONAL

광원의 방향 벡터

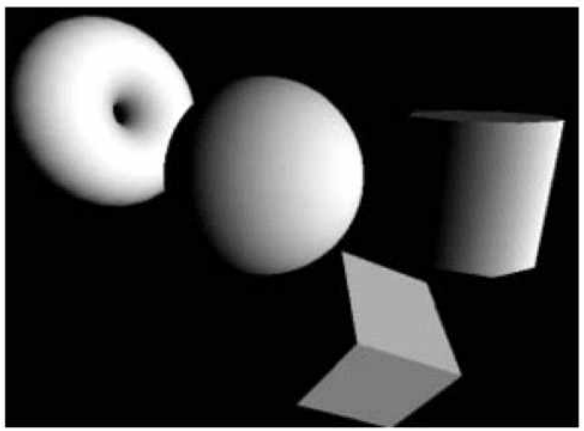
한 방향으로 평행한 빛을 발산

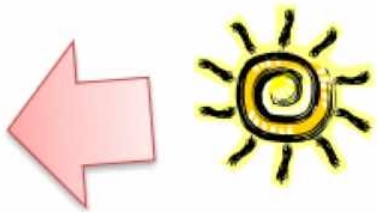
방향성 조명은 거리에 따라 감쇠가 일어나지 않음





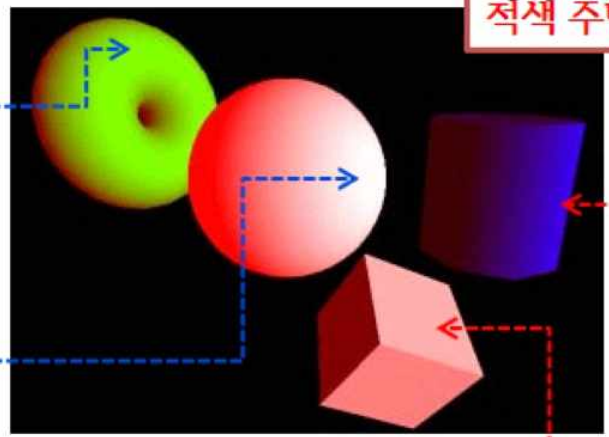
$$I = A + D + S + E$$
$$A = M_a * \{G_a + \sum L_a(i)\}$$
$$D = M_d * \sum\{L_d(i) * (\boldsymbol{n} \bullet \boldsymbol{L}_{dir}(\boldsymbol{i}))\}$$
$$S = M_s * \sum\{L_s(i) * (\boldsymbol{n} \bullet \boldsymbol{H}(\boldsymbol{i}))^P\}$$





조명(Lighting)

- **재질(Materials)**
  - 재질은 정점이 조명(빛)에 어떻게 반응(흡수와 반사)하는 가를 결정



4개의 물체는 모두 다른 재질로 구성되어 있음(왜냐하면 색상이 다르기 때문)

```
struct MATERIAL
{
    XMFLOAT4 m_xmf4Diffuse;
    XMFLOAT4 m_xmf4Ambient;
    XMFLOAT4 m_xmf4Specular;
    XMFLOAT4 m_xmf4Emissive;
    float m_fSpecularPower;
};
```

```
struct LIGHT
{
    XMFLOAT4 m_xmf4Diffuse;
    XMFLOAT4 m_xmf4Specular;
    XMFLOAT4 m_xmf4Ambient;
};
```

$$I = A + D + S + E$$
$$A = \boldsymbol{M}_a * \{G_a + \sum L_a(i)\}$$
$$D = \boldsymbol{M}_d * \sum\{L_d(i) * (\boldsymbol{n} \bullet \boldsymbol{L}_{dir}(i))\}$$
$$S = \boldsymbol{M}_s * \sum\{L_s(i) * (\boldsymbol{n} \bullet \boldsymbol{H}(i))^P\}$$

조명(Lighting)

- **조명 계산의 효율성**
  - 월드 좌표계
    조명 위치, 방향, 카메라 위치: 월드 좌표계
    정점(위치, 법선 벡터): 모델 좌표계 → 월드 좌표계
  - 모델 좌표계
    정점(위치, 법선 벡터): 모델 좌표계
    조명 위치, 방향, 카메라 위치: 월드 좌표계 → 모델 좌표계
  - 조명의 색상과 재질의 색상의 곱을 미리 계산

$$I = A + D + S + E$$
$$D = M_d * \{L_d * (\boldsymbol{n} \bullet \boldsymbol{L}_{dir})\} = (M_d * L_d) * (\boldsymbol{n} \bullet \boldsymbol{L}_{dir}) = Kd * (\boldsymbol{n} \bullet \boldsymbol{L}_{dir})$$
$$A = (M_a * Ga) + (M_a * L_a) = K_a$$
$$S = (M_s * L_s) * (\boldsymbol{n} \bullet \boldsymbol{H})^P = Ks * (\boldsymbol{n} \bullet \boldsymbol{H})^P$$

```
float4 vToLight;
float4 cLight; //(Kd, Ka)
color = cLight * max(cLight.a, dot(n, vToLight));
```

  - 조명 맵(Light Map)
    정적인 조명과 객체들의 조명 효과는 미리 계산할 수 있음
    미리 계산된 조명의 효과를 텍스쳐로 저장하여 쉐이더에서 샘플링하여 조명 처리
    조명 맵에서 샘플링하기 위한 텍스쳐 좌표가 추가적으로 필요함
  - 지연 조명 계산(Deferred Lighting)
    화면에 나타나지 않는 픽셀에 대한 조명 계산은 할 필요가 없음
    화면에 최종적으로 그려지는 픽셀에 대하여 조명 계산을 하려면?

조명(Lighting)

- 조명 계산의 효율성

접선 좌표계

```
struct VS_INPUT {
    float3 position : POSITION;
    float3 normal : NORMAL;
    float3 tangent  : TANGENT;
    float2 uv : TEXCOORD;
};
```

```
VS_OUTPUT VS(VS_INPUT input)
{
    VS_OUTPUT output = (VS_OUTPUT)0;
    float4 positionW = mul(float4(input.position, 1.0f), gmtxWorld);
    output.position = mul(mul(positionW, gmtxView), gmtxProjection);
    output.uv = input.uv;

    float3x3 mtxTangentToWorld;
    mtxTangentToWorld[0] = mul(input.tangent, gmtxWorld);
    mtxTangentToWorld[1] = mul(cross(input.tangent, input.normal), gmtxWorld));
    mtxTangentToWorld[2] = mul(input.normal, gmtxWorld);

    output.toLight = normalize(mul(mtxTangentToWorld, gvLightPosition – positionW.xyz)); //v * W^-1
    output.toCamera = normalize(mul(mtxTangentToWorld, gvCameraPosition – positionW.xyz));

    return(output);
};
```

픽셀 쉐이더에서 각 픽셀 마다 계산 또는 래스터라이저에서 보간

```
struct VS_OUTPUT {
    float4 position : SV_POSITION;
    float2 uv : TEXCOORD0;
    float3 toLight : TEXCOORD1;
    float3 toCamera : TEXCOORD2;
};
```

```
float4 PS(VS_OUTPUT input) : SV_Target
{
    float3 normal = gtxtNormal.Sample(gssNormal, input.uv).rgb;
    float3 normal = 2.0f * normal - 1.0f; //[0, 1] → [-1, 1]

    float4 cIllumination = Lighting(input.toLight, input.toCamera, normal);
    return(cIllumination);
};
```

```
float4 cIllumination = Lighting(input.positionW, input.normalW);
```

조명(Lighting)

- 조명 계산의 효율성

모델 좌표계

```
VS_OUTPUT VS(VS_INPUT input)
{
   VS_OUTPUT output = (VS_OUTPUT)0;

   float4 positionW = mul(float4(input.position, 1.0f), gmtxWorld);
   output.position = mul(mul(positionW, gmtxView), gmtxProjection);
   output.normal = input.normal;

   return(output);
};
```

```
struct VS_INPUT {
   float3 position : POSITION;
   float2 uv : TEXCOORD0;
   float3 normal : NORMAL;
};
```

```
struct VS_OUTPUT {
   float4 position : SV_POSITION;
   float2 uv : TEXCOORD0;
   float3 normal : NORMAL;
};
```

```
float4 PS(VS_OUTPUT input) : SV_Target
{
   float4 cIllumination = Lighting(gvToLight, gvToCamera, input.normal);

   return(cIllumination);
};
```

```
cbuffer cbLight : register(b8)
{
   float4 gvToLight;
   float4 gvToCamera;
};
```

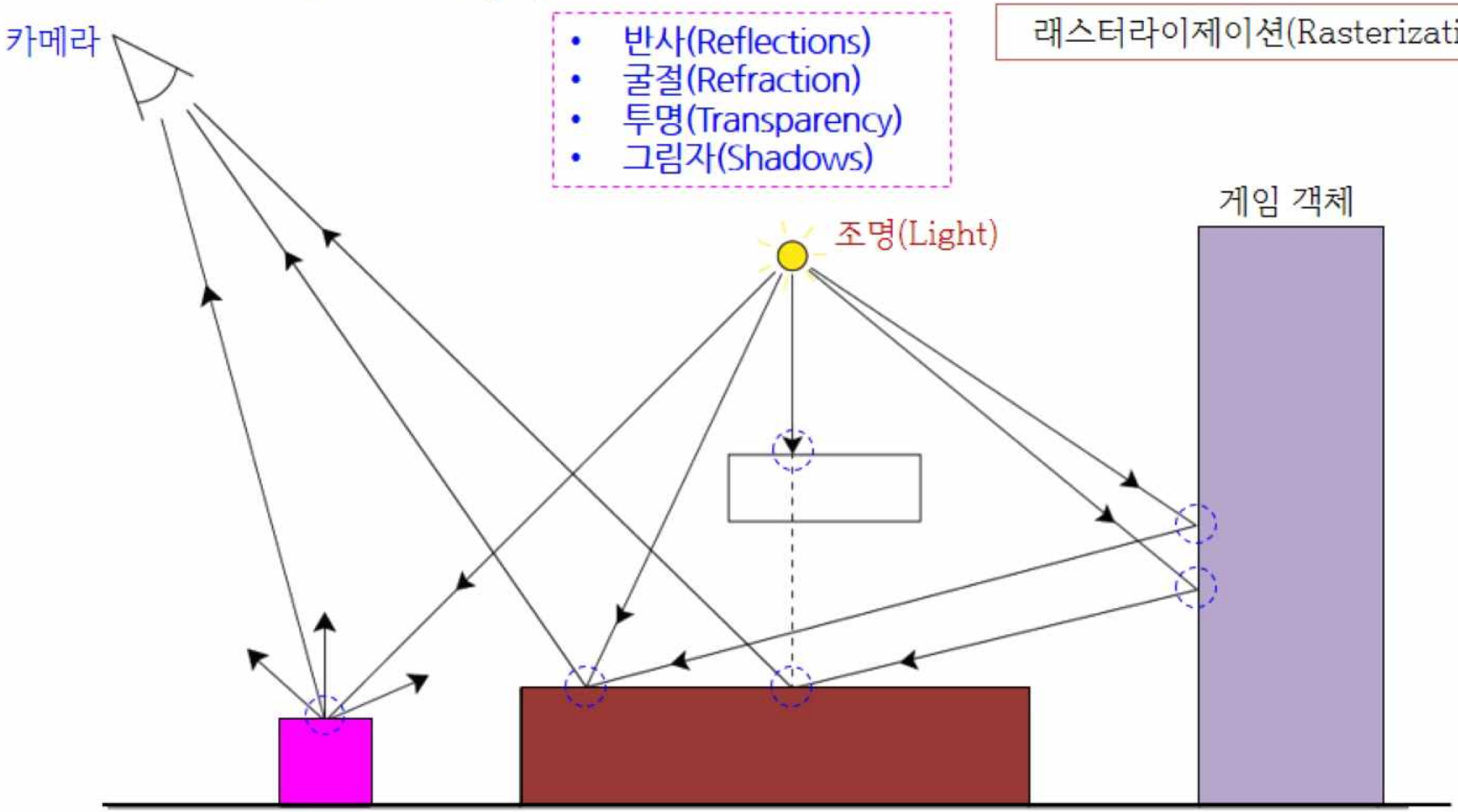
```
XMMATRIX xmmtxInverseWorld = XMMatrixInverse(NULL, XMLoadFloat4x4(&pObject->m_mtxWorld));
for (int i = 0; i < MAX_LIGHTS; i++)
{
   XMFLOAT4 xmf4Direction = m_pLights->m_pLights[i].m_xmf3Direction;
   XMVECTOR xmvToLight = XMVector3TransformNormal(xmf4Direction, xmmtxInverseWorld);
   XMStoreFloat4(&m_pd3dcbLight->m_pLights[i].m_xmf3Direction, XMVector3Normalize(xmvToLight));
   ...
}
```

조명(Lighting)

- **조명(Lights)**
  - 조명 계산, 조명 효과 계산(Lighting)
  - 조명(Emitted Light, Illumination)
    직접 조명(Direct Light)
    간접 조명(Indirect Light)

광선 추적(Ray Tracing)

래스터라이제이션(Rasterization)



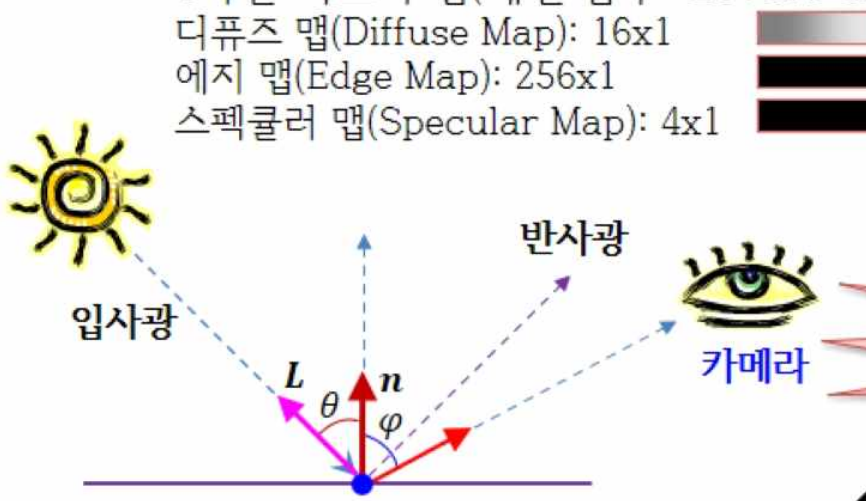
쉐이딩(Shading)

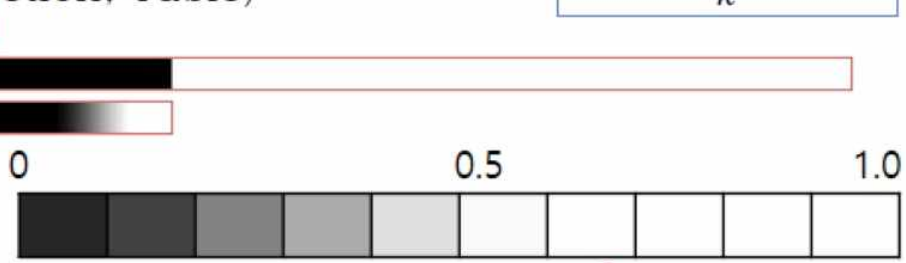
- **카툰 쉐이딩(Cartoon Shading)**
  - 셀 쉐이딩(Cel Shading)
    비사실적 조명(Non-photorealistic Lighting) 효과
    손으로 그린 애니메이션과 유사한 느낌

쉐이딩(Shading)

- **카툰 쉐이딩(Cartoon Shading)**
  - 셀 쉐이딩(Cel Shading)
    객체의 윤곽선(Outline)과 이산적인(Discrete, Segmented) 쉐이딩
  - 1차원 텍스쳐 맵(레벨 함수: Level Function, Table)
    디퓨즈 맵(Diffuse Map): 16x1
    에지 맵(Edge Map): 256x1
    스펙큘러 맵(Specular Map): 4x1

$$\frac{floor\big((\boldsymbol{n}\cdot\boldsymbol{L})*k\big)}{k}$$

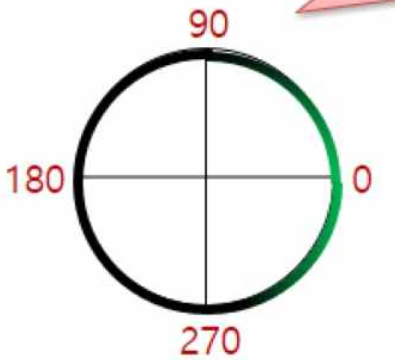




$cos(\theta)$ 값을 맵의 인덱스(u)로 사용

디퓨즈: $(90 \geq \theta \geq 0) \Rightarrow (0 \leq cos(\theta) \leq 1)$
경계선: $(90 \geq \varphi \geq 0) \Rightarrow (0 \leq cos(\varphi) \leq 1)$

$cos(\theta) < 0$: 디퓨즈 색상은 검정색(0)



쉐이딩(Shading)

- 카툰 쉐이딩(Cartoon Shading)

```
float4 ToonShading(float3 positionW, float3 normalW, float3 vLightDirection, float fSpecularPower)
{
    float3 vCameraPosition = gvCameraPosition.xyz;
    float3 vToCamera = normalize(vCameraPosition - positionW);
    float fEdge = max(dot(vToCamera, normalW), 0.0f);
    fEdge = gtxtEdgeRamp.Sample(gsToonSamplerState, fEdge).x;
    float fDiffuse = max(dot(-vLightDirection, normalW), 0.0f);
    fDiffuse = gtxtDiffuseRamp.Sample(gsToonSamplerState, fDiffuse).x;
    float3 vReflect = reflect(-vLightDirection, normalW);
    float fSpecular = pow(max(dot(vReflect, vToCamera), 0.0f), fSpecularPower);
    fSpecular = gtxtSpecularRamp.Sample(gsToonSamplerState, fSpecular).x;
    float4 cColor = fEdge * (gcMaterialDiffuse * fDiffuse + gcMaterialSpecular * fSpecular);
    return(cColor);
}
```

```
SamplerState gsToonSamplerState : register(s1);
```

디퓨즈: $(0 \le cos(\theta) \le 1)$
경계선: $(0 \le cos(\varphi) \le 1)$

```
Texture1D gtxtDiffuseRamp : register(t1);
Texture1D gtxtSpecularRamp : register(t2);
Texture1D gtxtEdgeRamp : register(t3);
```

```
D3D12_STATIC_SAMPLER_DESC d3dSamplerDesc;
::ZeroMemory(&d3dSamplerDesc, sizeof(D3D12_STATIC_SAMPLER_DESC));
d3dSamplerDesc.Filter = D3D12_FILTER_MIN_MAG_MIP_POINT;
d3dSamplerDesc.AddressU = d3dSamplerDesc.AddressV = D3D12_TEXTURE_ADDRESS_WRAP;
d3dSamplerDesc.AddressW = D3D12_TEXTURE_ADDRESS_WRAP;
d3dSamplerDesc.ComparisonFunc = D3D12_COMPARISON_NEVER;
D3D12_ROOT_SIGNATURE_DESC d3dRootSignatureDesc;
...
d3dRootSignatureDesc.pStaticSamplers = &d3dSamplerDesc;
D3D12SerializeRootSignature(&d3dRootSignatureDesc, ...);
pd3dDevice->CreateRootSignature(0, ...);
```

쉐이딩(Shading)

- **카툰 쉐이딩(Cartoon Shading)**
  - 객체의 윤곽선(외곽선: Outline)
    ① 카메라 시선 벡터와 법선 벡터의 내적: $(90 \geq \varphi \geq 0)$ ⇨ $(0 \leq cos(\varphi) \leq 1)$
    메쉬를 구성하는 정점의 수가 적으면 윤곽선을 찾기 어려움
    ② 이미지 처리(Image Processing) 기법
    컨벌루션(Convolution): 이미지 필터링(Filtering)

-1	0	1
-2	0	2
-1	0	1

k_x

1	2	1
0	0	0
-1	-2	-1

k_y

소벨(Sobel) 에지 필터

$$p_{ij} = \sqrt{{e_x}^2 + {e_y}^2}$$

-1	-1	-1
-1	8	-1
-1	-1	-1

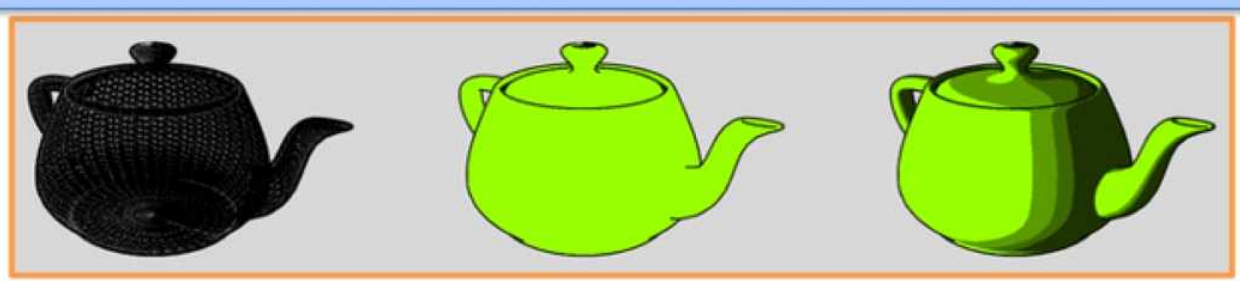
라플라시안(Laplacian) 에지 필터

③ 다른 방법은?
이미지 필터링 방법은 너무 민감하고 잡음의 영향을 받음
렌더링 과정에서 사용할 수 있는 정보(색상, 법선 벡터, …)

쉐이딩(Shading)

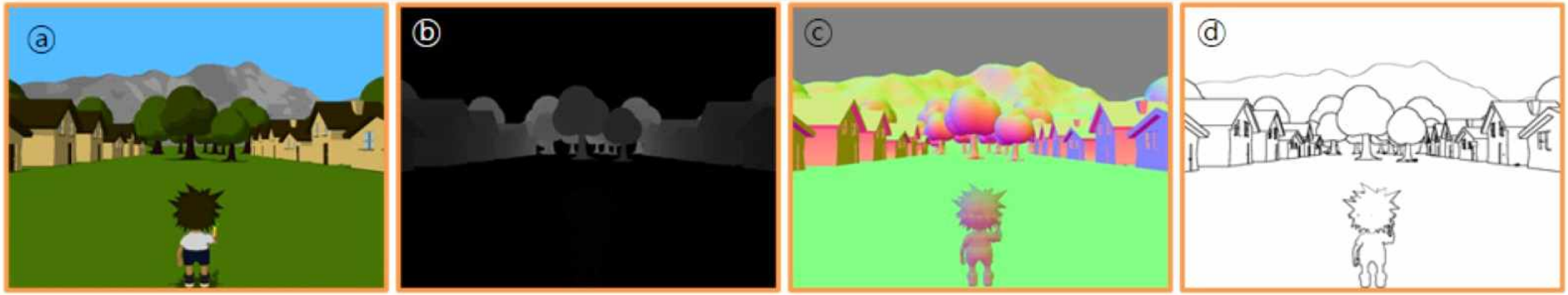
- **카툰 쉐이딩(Cartoon Shading)**
  - 객체의 윤곽선(Outline)

① 객체(메쉬)의 은면(Back Face)을 검정색으로 렌더링한다. 위치를 조금씩 이동하면서 와이어프레임으로 여러 번 그리거나 정점을 법선 벡터 방향으로 이동시켜 면을 검정색으로 그려도 된다.
② 객체를 정상적으로 렌더링한다. 은면은 전면보다 뒤에 있으므로 객체의 경계선(에지)를 제외한 영역에 전면이 그려진다. 전면을 그릴 때 레벨 함수를 사용하여 이산적인 쉐이딩을 할 수 있다.



  - 씬 전체에 대한 윤곽선

① 화면 전체를 이산적인 쉐이딩을 사용하여 텍스쳐(ⓐ)에 렌더링한다.
색상(Color), 깊이값(Depth), 법선 벡터(Normal Vector)를 각각 다른 텍스쳐에 렌더링한다.
② 깊이값/법선 벡터 텍스쳐에서 객체의 윤곽선(ⓓ)을 이미지 처리 기법을 사용하여 생성한다(예: 소벨 매스크).
③ 에지 텍스쳐(ⓓ)와 색상 텍스쳐(ⓐ)를 곱한다.



쉐이딩(Shading)

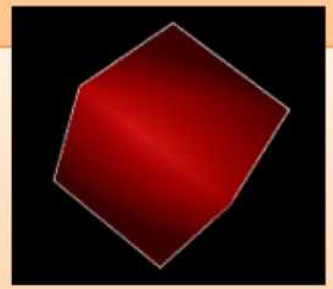
- **카툰 쉐이딩(Cartoon Shading)**
  - 씬 전체에 대한 윤곽선 렌더링

```
static float gfLaplacians[9] = { -1.0f, -1.0f, -1.0f, -1.0f, 8.0f, -1.0f, -1.0f, -1.0f, -1.0f };
static int2 gnOffsets[9] = { {-1,-1}, {0,-1}, {1,-1}, {-1,0}, {0,0}, {1,0}, {-1,1}, {0,1}, {1,1} };

float4 PSTextureToScreen(float4 position : SV_POSITION) : SV_Target {
   float fEdgeness = 0.0f;
   float3 cEdgeness = float3(0.0f, 0.0f, 0.0f);
   int2 pos = int2(position.xy);
   if ((pos.x > 0) || (pos.y > 0) || (pos.x < gtxtNormal.Length.x-1) || (pos.y < gtxtNormal.Length.y-1)) {
      for (int i = 0; i < 9; i++) cEdgeness += gfLaplacians[i] * gtxtNormal[pos.xy+gnOffsets[i]].xyz * 2.0f - 1.0f;
      fEdgeness = cEdgeness.r * 0.3f + cEdgeness.g * 0.59f + cEdgeness.b * 0.11f;
      cEdgeness = float3(fEdgeness, fEdgeness, fEdgeness);
   }
   float3 cColor = gtxtScene[int2(input.position.xy)].rgb;
   cColor = (fEdgeness < 0.5f) ? cColor : ((fEdgeness < 0.9f) ? (cColor + cEdgeness) : cEdgeness);
   return(float4(cColor, 1.0f));
}
```

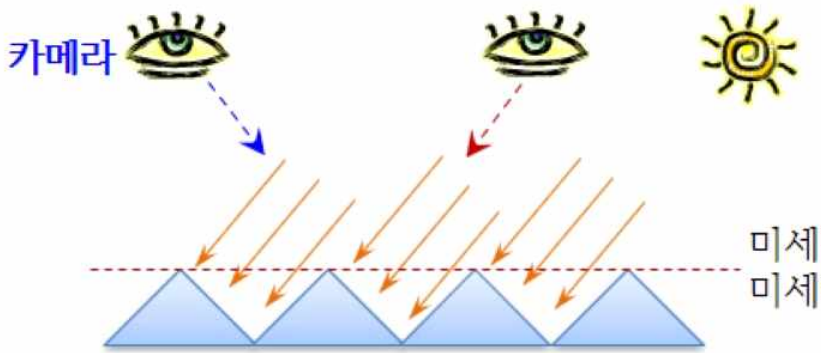
```
Texture2D<float4> gtxtScene : register(t0);
Texture2D<float4> gtxtNormal : register(t1);
```

```
float4 VSTextureToScreen(uint nVertexID : SV_VertexID) : SV_POSITION {
   if (nVertexID == 0) return(float4(-1.0f, +1.0f, 0.0f, 1.0f));
   if (nVertexID == 1) return(float4(+1.0f, +1.0f, 0.0f, 1.0f));
   if (nVertexID == 2) return(float4(+1.0f, -1.0f, 0.0f, 1.0f));
   if (nVertexID == 3) return(float4(-1.0f, +1.0f, 0.0f, 1.0f));
   if (nVertexID == 4) return(float4(+1.0f, -1.0f, 0.0f, 1.0f));
   if (nVertexID == 5) return(float4(-1.0f, -1.0f, 0.0f, 1.0f));
   return(float4(0.0f, 0.0f, 0.0f, 0.0f));
}
```

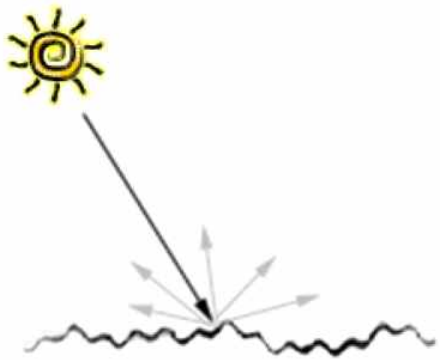


쉐이딩(Shading)

- **반사 모델(Reflectance Model)**
  - Cook-Torrance 반사 모델
    스펙큘러 하이라이트가 희미해지는 것은 **미세면(Micro Facet)**이 여러 방향으로 향하기 때문
    표면의 기하학적 표현: 표면이 작은 V-형 홈(Groove)들로 구성(Torrance-Sparrow)
    계산의 복잡도 때문에 미세면을 직접 사용하지 않고 **확률 분포 함수로 미세면을 근사**하여 사용
  - 미세면 분포 함수(Microfacet Distribution Function)
  - 기하학적 감쇠 요소(Geometrical Attenuation Factor)



미세면의 기울기가 같다고 가정
미세면의 높이가 같다고 가정



시선 벡터가 조명의 방향 벡터와 일치할 때 미세면에서 스펙큘러

램버트 모델(디퓨즈 표면, 균일 반사)

$$I = c * \frac{D * G * F}{\boldsymbol{n} \cdot \boldsymbol{E}} \qquad I = c * \frac{F}{\pi} \frac{D * G}{(\boldsymbol{n} \cdot \boldsymbol{E})(\boldsymbol{n} \cdot \boldsymbol{L})}$$

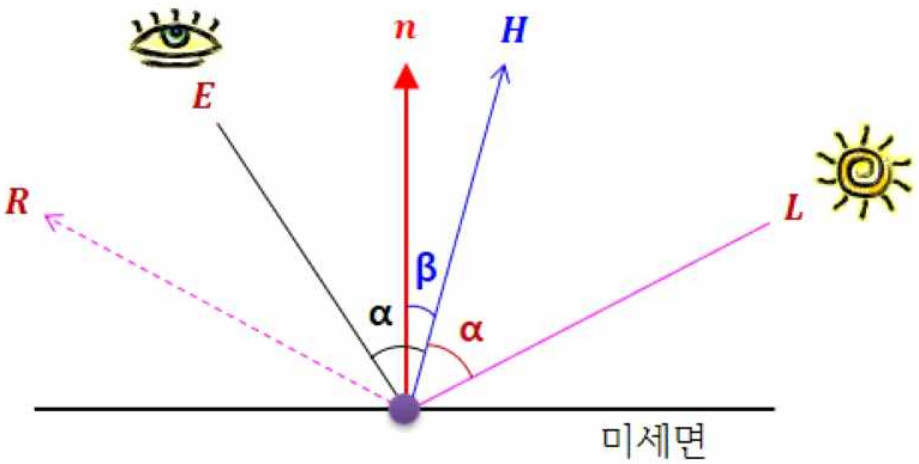
c: 재질의 색상
D: 미세면 분포함수
G: 기하 감쇠 함수(Geometrical Attenuation Factor)
F: 프레넬(Fresnel) 반사 계수
$\boldsymbol{n}$: 법선 벡터
$\boldsymbol{E}$: 시선 벡터(카메라 방향 벡터) $\boldsymbol{L}$: 조명 방향 벡터

쉐이딩(Shading)

- **반사 모델(Reflectance Model)**
  - Cook-Torrance 반사 모델

① **미세면 분포 함수(Microfacet Distribution Function)**

미세면들의 기울기가 표면의 법선 벡터와 어떻게 어긋나는 가의 분포를 나타내는 함수
거울은 미세면들의 법선 벡터가 대부분 표면의 법선 벡터와 일치한다고 가정
미세면들의 법선 벡터가 표면의 법선 벡터와 다르면 표면에서 빛이 산란되어 희미하게 됨



$D = ae^{-(b\beta)^2}$: Gaussian 분포 함수

$D = \left[\frac{1}{(\cos\beta)^2(c^2-1)+1}\right]^2$: 회전 타원체 분포 함수

$D = (\cos\beta)^n$: Blinn 분포 함수

$\boldsymbol{H} = \boldsymbol{L} + \boldsymbol{E}$

$\cos(\alpha) = \boldsymbol{H}\cdot\boldsymbol{E} = \boldsymbol{L}\cdot\boldsymbol{H}$

$\cos(\beta) = \boldsymbol{n}\cdot\boldsymbol{H}$

$(tan\ \beta)^2 = \frac{(\sin\beta)^2}{(\cos\beta)^2} = \frac{1-(\cos\beta)^2}{(\cos\beta)^2} = \frac{1-(\boldsymbol{n}\cdot\boldsymbol{H})^2}{(\boldsymbol{n}\cdot\boldsymbol{H})^2}$

Beckmann 분포 함수

$$D = \frac{1}{4m^2(cos\beta)^4} e^{-(\frac{\tan\beta}{m})^2} = \frac{1}{4m^2(cos\beta)^4} e^{-\frac{1}{m^2}(\frac{1-(\cos\beta)^2}{(\cos\beta)^2})} = \frac{1}{4m^2(\boldsymbol{n}\cdot\boldsymbol{H})^4} e^{-\frac{1}{m^2}(\frac{1-(\boldsymbol{n}\cdot\boldsymbol{H})^2}{(\boldsymbol{n}\cdot\boldsymbol{H})^2})}$$

m: 미세면들의 기울기의 RMS(Root Mean Square), 표면의 거친 정도를 결정
0에 가까우면 반사광이 좁게 반사(표면이 부드러움, 즉 표면의 홈들의 깊이가 크지 않음)
큰 값은 표면의 홈들이 깊고 빛이 넓게 반사됨을 의미

쉐이딩(Shading)

- **반사 모델(Reflectance Model)**
  - Cook-Torrance 반사 모델

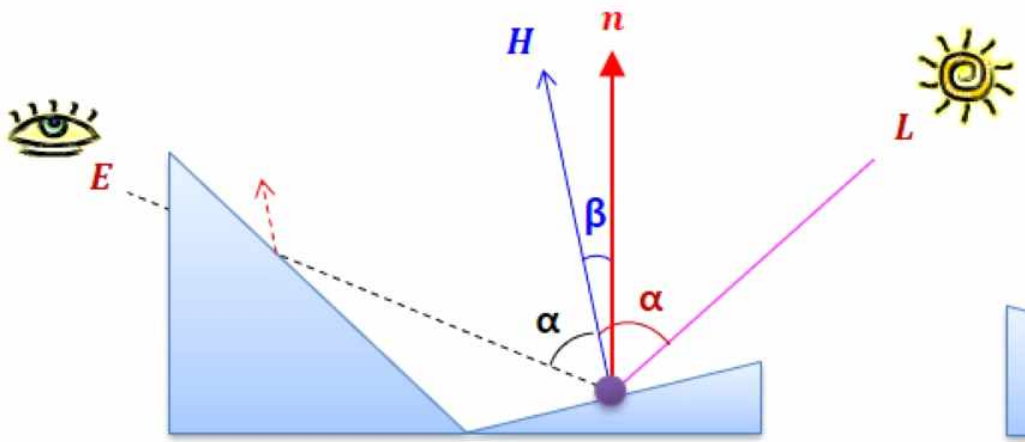
② **기하학적 감쇠 요소(Geometrical Attenuation Factor)**

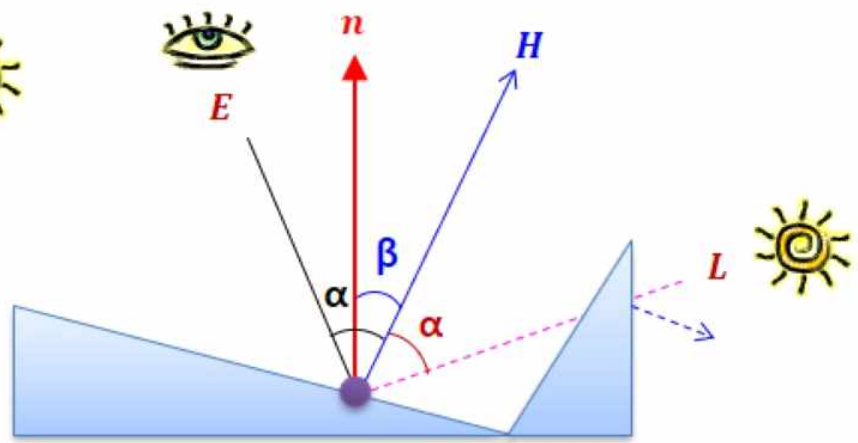
미세면 분포함수는 표면의 거친 정도를 근사적으로 표현하지만 다음의 경우를 설명하지 못함

- 반사광 차폐(Masking): G_1
  하나의 미세면에서 반사된 빛이 다른 미세면 때문에 카메라에 도달하지 않는 경우
  미세면에서 반사된 빛이 다른 미세면 때문에 차폐되는(가려지는) 경우
  시선 벡터(E)와 법선 벡터(n) 사이의 각도가 90°에 가까우면 차폐가 발생 가능
- 입사광 차폐(Shadowing): G_2
  빛이 하나의 미세면에 도달하기 전에 인접한 다른 미세면에서 먼저 반사되는 경우
  입사광이 하나의 미세면 때문에 차폐되는(가려지는) 경우
  조명 벡터(L)와 법선 벡터(n) 사이의 각도가 90°에 가까우면 차폐가 발생 가능

$$G_1 = \frac{2 * (\boldsymbol{n} \cdot \boldsymbol{H}) * (\boldsymbol{n} \cdot \boldsymbol{E})}{(\boldsymbol{E} \cdot \boldsymbol{H})}$$

$$G_2 = \frac{2 * (\boldsymbol{n} \cdot \boldsymbol{H}) * (\boldsymbol{n} \cdot \boldsymbol{L})}{(\boldsymbol{E} \cdot \boldsymbol{H})}$$

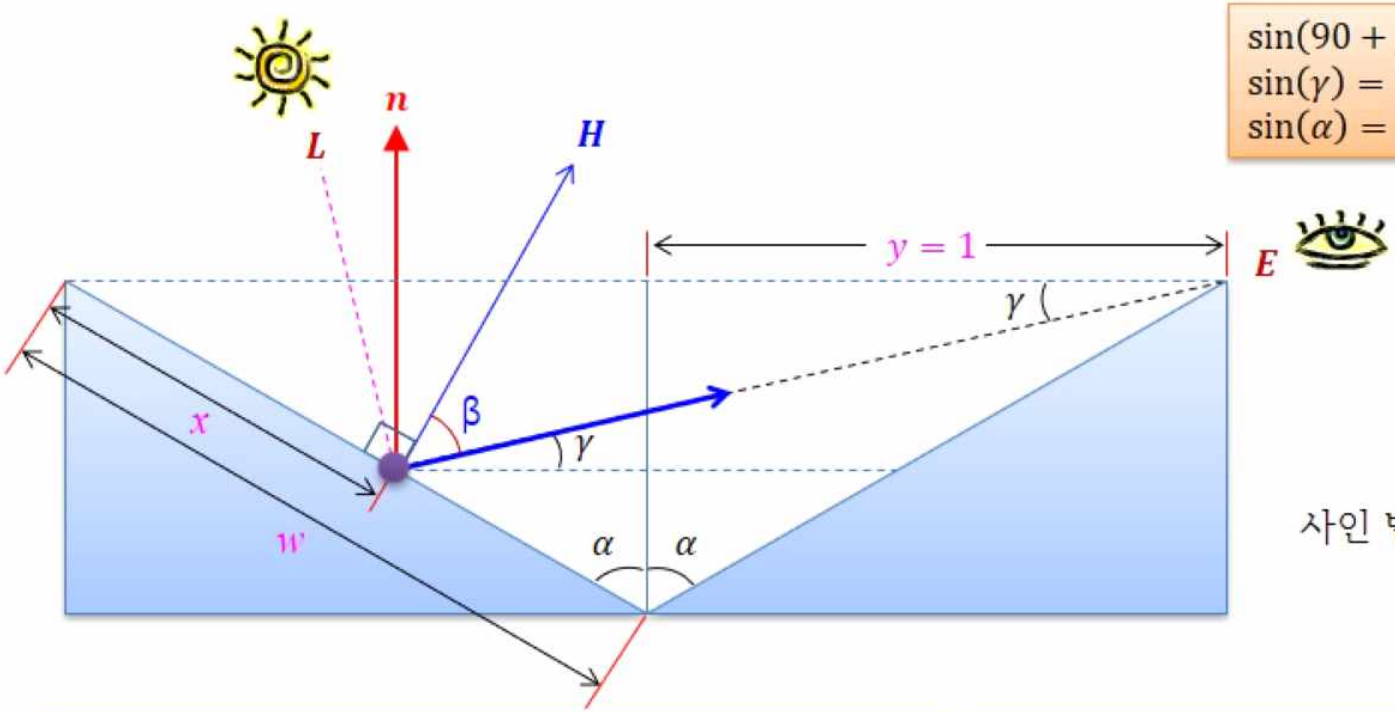




쉐이딩(Shading)

- **반사 모델(Reflectance Model)**
  - Cook-Torrance 반사 모델
  - ② 기하학적 감쇠 요소(Geometrical Attenuation Factor)
    G_1과 G_2를 계산하기 위하여 미세면의 홈의 길이가 같다고 가정(Blinn)
    차폐가 되지 않는 경우 G_1과 G_2의 값은 1.0

$$\sin(90+\beta)=\cos(\beta)=\boldsymbol{E}\cdot\boldsymbol{H}=\boldsymbol{L}\cdot\boldsymbol{H}$$
$$\sin(\gamma)=\cos(90-\gamma)=\boldsymbol{n}\cdot\boldsymbol{E}$$
$$\sin(\alpha)=\cos(90-\alpha)=\boldsymbol{n}\cdot\boldsymbol{H}$$



$$\sin(\alpha)=\frac{y}{w}=\frac{1}{w}$$
$$w=\frac{1}{\sin(\alpha)}$$

사인 법칙 $\dfrac{x}{\sin(\gamma)}=\dfrac{2y}{\sin(90+\beta)}$

$$x=\frac{2*\sin(\gamma)}{\sin(90+\beta)}$$

$$G_1=\frac{x}{w}=\frac{2*\sin(\gamma)*\sin(\alpha)}{\sin(90+\beta)}=\frac{2*(\boldsymbol{n}\cdot\boldsymbol{E})*(\boldsymbol{n}\cdot\boldsymbol{H})}{(\boldsymbol{E}\cdot\boldsymbol{H})}$$

$$G_2=\frac{x}{w}=\frac{2*\sin(\gamma)*\sin(\alpha)}{\sin(90+\beta)}=\frac{2*(\boldsymbol{n}\cdot\boldsymbol{L})*(\boldsymbol{n}\cdot\boldsymbol{H})}{(\boldsymbol{L}\cdot\boldsymbol{H})}$$

G_1에서 E를 L로 대체

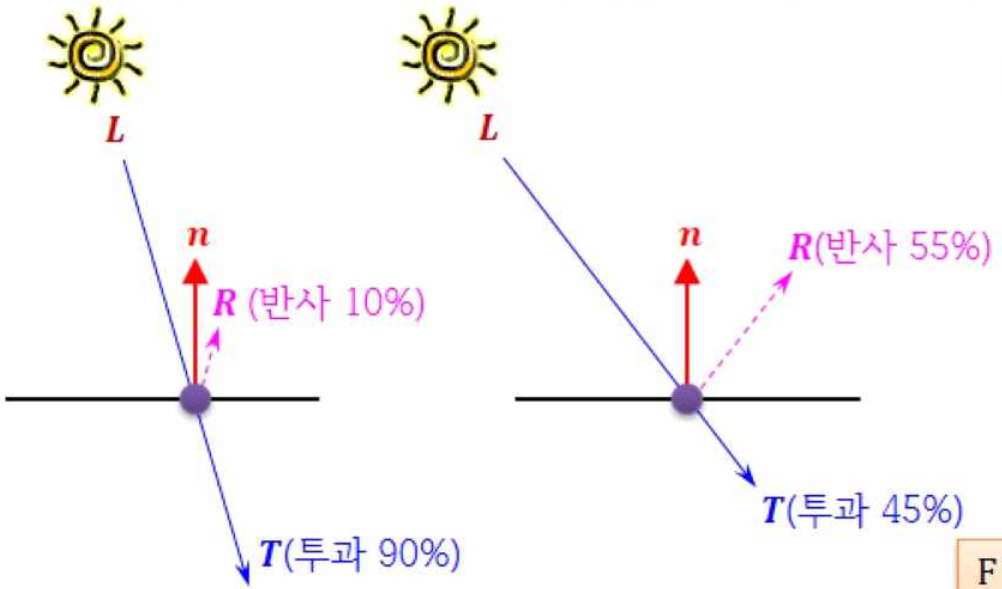
$$\boldsymbol{G}=\min(G_1,G_2,1)=\min\left(\frac{2*(\boldsymbol{n}\cdot\boldsymbol{E})*(\boldsymbol{n}\cdot\boldsymbol{H})}{(\boldsymbol{E}\cdot\boldsymbol{H})},\frac{2*(\boldsymbol{n}\cdot\boldsymbol{L})*(\boldsymbol{n}\cdot\boldsymbol{H})}{(\boldsymbol{E}\cdot\boldsymbol{H})},1\right)$$

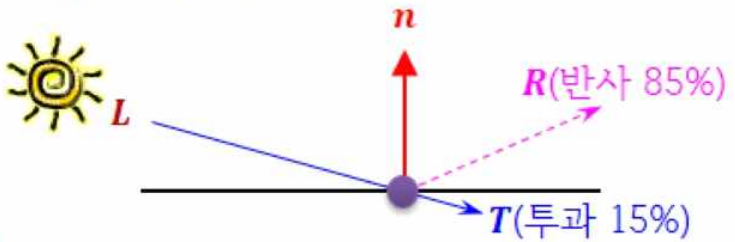
쉐이딩(Shading)

- **반사 모델(Reflectance Model)**
  - Cook-Torrance 반사 모델

③ **프레넬 반사(Fresnel Reflection)**

수영장의 물을 위에서 보면 수영장 바닥이 보임(표면에서 반사되는 빛을 보지 못한다는 의미)
수영장의 물을 옆에서 보면 수영장 바닥을 볼 수 없음(표면에서 반사되는 빛을 많이 보게 됨)
모든 물체의 표면에서 빛의 일부분은 투과하고 나머지 빛은 반사됨
반사되는 빛의 양은 표면 재질의 성질(굴절률)과 입사각에 따라 달라짐
프레넬 반사는 부분 투명한 재질(플라스틱, 유리, 물, 피부, 페인트 등)에서 중요함





$$F(\theta_i) = \frac{1}{2}\left[\frac{g-c}{g+c}\right]^2\left\{1+\left[\frac{c(g+c)-1}{c(g-c)+1}\right]^2\right\}$$

$g^2 = \lambda^2 + c^2 - 1$ (λ: 굴절률)

$c = \cos(\theta_i) + (L \cdot H) \approx L \cdot H$

프레넬 반사 요소(Fresnel Reflection Factor)
Schlick 근사: 실외환경에서 비교적 정확한 근사

$F = f_\lambda + (1 - f_\lambda)(1 - \cos(\theta_i))^5$

$F = f_\lambda + (1 - f_\lambda)(1 - \boldsymbol{n} \cdot \boldsymbol{E})^5$

f_λ: 프레넬 반사 계수

4, 2를 사용

④ **단위 면적 당 미세면의 농도**

단위 면적당 미세면의 개수는 옆에서 볼 때가 앞에서 볼 때보다 많음
가장자리에서 빛의 반사가 더 많이 일어나게 함

$$\frac{1}{\boldsymbol{n} \cdot \boldsymbol{E}}$$

쉐이딩(Shading)

- **반사 모델(Reflectance Model)**
  - Cook-Torrance 반사 모델

스펙큘러 반사 요소: $R_s = \frac{F}{\pi} * \frac{D*G}{(n \cdot E)(n \cdot L)}$ $\quad R_s = \frac{D*G*F}{n \cdot E}$

fm=0.35f; fFRI=0.01f;

```
float SpecularFactor(float3 vPosition, float3 vNormal, float3 vCamera, float3 vLight, float fm, float fFRI) {
    float3 vToLight = vLight - vPosition;
    float3 vToCamera = vCamera - vPosition;
    float3 N = normalize(vNormal);
    float3 L = normalize(vToLight);
    float3 E = normalize(vToCamera);
    float3 H = normalize(vToLight + vToCamera);
    float NH = saturate(dot(N, H));
    float EH = saturate(dot(E, H));
    float NE = saturate(dot(N, E));
    float NL = saturate(dot(L, N));
    float NH2 = NH * NH;
    float m2 = fm * fm;
    float D = (1 / 4 * m2 * NH2 * NH2) * (exp(-((1 - NH2) / (m2 * NH2))));
    float G = min(1.0f, min((2 * NH * NL) / EH, (2 * NH * NE) / EH));
    float F = fFRI  +  (1 - fFRI) * pow((1 - NE), 5.0f);
    float fSF = (F * D * G) / (3.1415926535 * NL * NE);
    return(fSF);
}
```

$$D = \frac{1}{4m^2(\boldsymbol{n} \cdot \boldsymbol{H})^4} e^{-\frac{1}{m^2}\left\{\frac{1-(n \cdot H)^2}{(n \cdot H)^2}\right\}}$$

$$G_1 = \frac{2 * (\boldsymbol{n} \cdot \boldsymbol{H}) * (\boldsymbol{n} \cdot \boldsymbol{E})}{(\boldsymbol{E} \cdot \boldsymbol{H})}$$

$$G_2 = \frac{2 * (\boldsymbol{n} \cdot \boldsymbol{H}) * (\boldsymbol{n} \cdot \boldsymbol{L})}{(\boldsymbol{E} \cdot \boldsymbol{H})}$$

$$\mathrm{F} = f_\lambda + (1 - f_\lambda)(1 - \boldsymbol{n} \cdot \boldsymbol{E})^5$$

$$\mathrm{F}(\theta_i) = \frac{1}{2}\left[\frac{g-c}{g+c}\right]^2 \left\{1 + \left[\frac{c(g+c)-1}{c(g-c)+1}\right]^2\right\}$$

$g^2 = \lambda^2 + c^2 - 1$ (λ: 굴절률)

$c = \cos(\theta_i) + (L \cdot H) \approx L \cdot H$

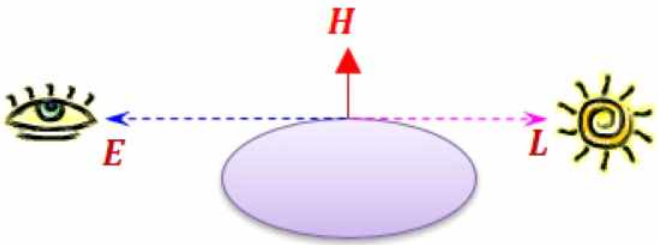
```
float fL = 20.0f;
float g = sqrt(fL * fL + LH * LH – 1);
float gpc = g + LH, gnc = g – LH;
float cgpc = LH * gpc – 1, cgnc = LH * gnc + 1;
float F = 0.5f * (gnc * gnc) / (gpc * gpc) * (1 + (cgpc * cgpc) / (cgnc * cgnc));
```

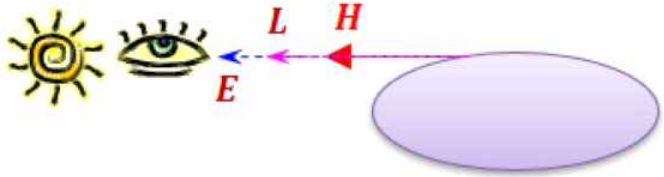
쉐이딩(Shading)

- **반사 모델(Reflectance Model)**
  - Cook-Torrance 반사 모델

⑤ **색상 쉬프트(Color Shift)**

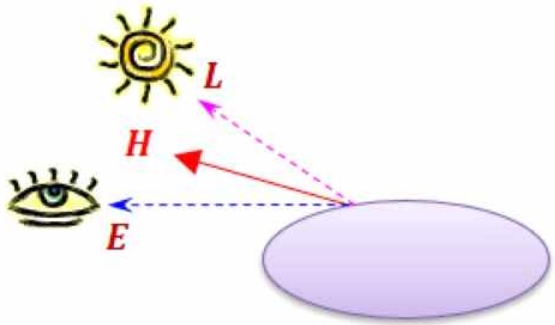
빛의 입사 방향이 수평에 가까우면 반사광의 색이 입사광의 색이 됨
입사 방향과 반사 방향이 거의 같으면 반사광에는 재질(물체)의 색이 첨가됨





$c = L \cdot H = 0 \rightarrow \mathrm{F}(\theta_i) = \mathrm{F}(90) = 1$
빛은 굴절률을 따르지 않고 전반사
입사광의 색이 그대로 반사

$c = L \cdot H = 1 \rightarrow \mathrm{F}(\theta_i) = \mathrm{F}(0) = \left[\frac{\lambda-1}{\lambda+1}\right]^2$
빛의 반사는 재질의 굴절률을 따름
재질의 색이 나타남



$$\mathrm{F}(\theta_i) = \frac{1}{2}\left[\frac{g-c}{g+c}\right]^2\left\{1+\left[\frac{c(g+c)-1}{c(g-c)+1}\right]^2\right\}$$

$g^2 = \lambda^2 + c^2 - 1$ (λ: 굴절률)

$c = \cos(\theta_i) + (\boldsymbol{L} \cdot \boldsymbol{H}) \approx \boldsymbol{L} \cdot \boldsymbol{H}$

스펙큘러 반사 색상: $R_s * Color$

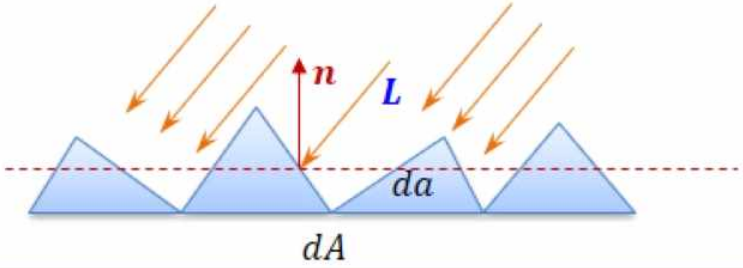
$$Color = (1-t) * Color_0 + t * Color_1 = Color_0 + (Color_1 - Color_0) * t$$

$$t = \frac{\max(0, F - F(0))}{F(90) - F(0)} = \frac{\max(0, F - F(0))}{1 - F(0)}$$

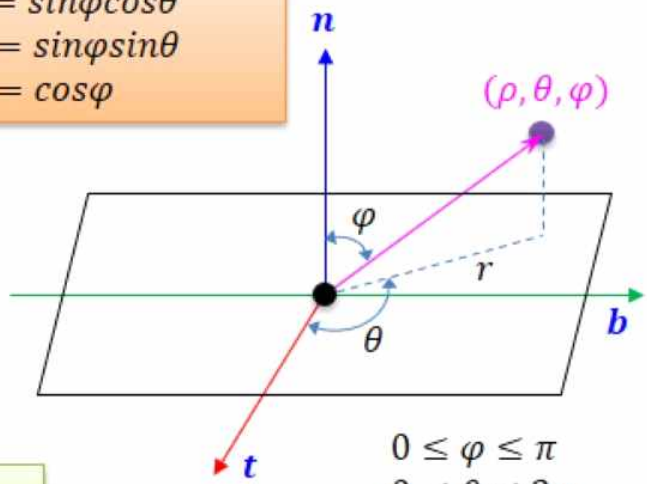
F(90)일 때의 색상: $Color_0$
F(0)일 때의 색상: $Color_1$ = 조명 색상

쉐이딩(Shading)

- **반사 모델(Reflectance Model)**
  - **Oren-Nayar 반사 모델**
    구면 좌표계(Spherical Coordinates)를 사용
    미세면(Microfacet) 모델
    미세면의 기울기가 다를 수 있음
    표면의 거친 정도를 기울기 면적의 분포로 표현
    정규 분포(Gaussian) 함수: 평균(0), 표준 편차(σ)



$(1, \theta, \varphi) \rightarrow (t, b, n)$
$t = sin\varphi cos\theta$
$b = sin\varphi sin\theta$
$n = cos\varphi$



$0 \le \varphi \le \pi$
$0 \le \theta \le 2\pi$
$\rho = 1$

$D = ae^{\frac{\theta^2}{\sigma^2}}$: Gaussian 분포 함수
a: 정규화 상수, θ: 표면 패치 법선 벡터에 대한 미세면 법선 벡터의 각도

표면의 밝기는 모든 미세면들의 밝기의 합으로 표현

$I = P * E_0 * cos\theta_L * (A + B * max(0, cos(\theta_V - \theta_L)) * sin\alpha * tan\beta)$
$\frac{P}{\pi}$: 표면 반사 성질(표면 디퓨즈 색상, 텍스쳐 색상)
E_0: 조명 색상
$cos\theta_L$: $\boldsymbol{n} \cdot \boldsymbol{L}$
$A = 1 - \frac{0.5 * \sigma^2}{\sigma^2} + 0.33 \quad B = 0.45 * \frac{\sigma^2}{\sigma^2} + 0.09$
$\alpha = min(\varphi_V, \varphi_L) \quad \beta = max(\varphi_V, \varphi_L)$
σ: 표면의 거친 정도(미세면 기울기 각도의 표준 편차)

φ_V: 정점 법선 벡터와 카메라 방향 벡터 사이의 각
φ_L: 정점 법선 벡터와 조명 방향 벡터 사이의 각
$(\theta_V - \theta_L)$: 카메라 방향 벡터와 조명 방향 벡터 사이의 회전 각

벡터(Vectors)

- **극 좌표와 구면 좌표(Polar and Spherical Coordinates)**
  - 극 좌표(2차원)

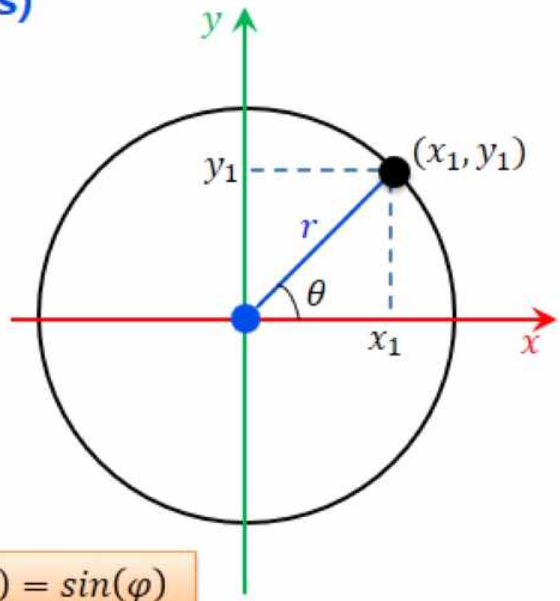
$(r,\theta) \rightarrow (x_1,y_1)$
$x_1 = r * cos\theta$
$y_1 = r * sin\theta$

$(x_1,y_1) \rightarrow (r,\theta)$
$r = \sqrt{{x_1}^2 + {y_1}^2}$
$\theta = \cos^{-1}(\frac{x_1}{r})$
$\frac{y_1}{x_1} = tan\theta$
$\theta = tan^{-1}(\frac{y_1}{x_1})$

$acos(\cdot) \rightarrow [0,\pi]$

$x_1 = 0$

$arctan2(y,x) \rightarrow [-\pi,\pi]$



  - 구면 좌표(3차원)

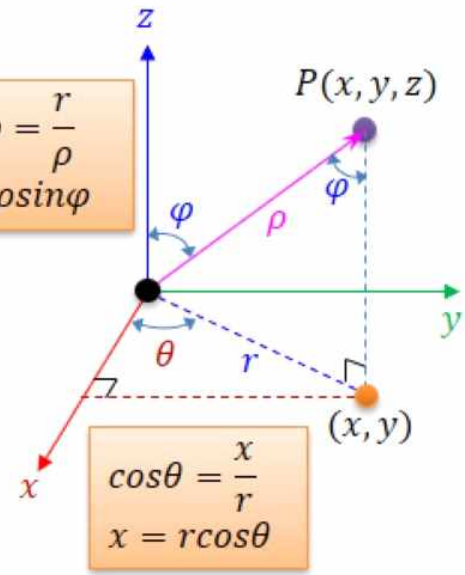
$cos(90-\varphi) = sin(\varphi)$

$(\rho,\theta,\varphi) \rightarrow (x,y,z)$
$x = \rho sin\varphi cos\theta$
$y = \rho sin\varphi sin\theta$
$z = \rho cos\varphi$

$0 \le \varphi \le \pi$
$0 \le \theta \le 2\pi$

$(x,y,z) \rightarrow (\rho,\theta,\varphi)$
$\rho = \sqrt{x^2+y^2+z^2}$
$\frac{r}{z} = \frac{\rho sin\varphi}{\rho cos\varphi} = tan\varphi \rightarrow \varphi = arctan2(r,z)$
$\frac{y}{x} = \frac{sin\theta}{cos\theta} = tan\theta \rightarrow \theta = arctan2(y,x)$

$sin\varphi = \frac{r}{\rho}$
$r = \rho sin\varphi$



$r = \sqrt{x^2+y^2}$
$= \sqrt{(\rho sin\varphi cos\theta)^2 + (\rho sin\varphi sin\theta)^2}$
$= \sqrt{(\rho sin\varphi)^2\{cos^2\theta + sin^2\theta\}} = \rho sin\varphi$

$cos\theta = \frac{x}{r}$
$x = r cos\theta$

쉐이딩(Shading)

- 반사 모델(Reflectance Model)
  - Oren-Nayar 반사 모델

```
VS_OUTPUT VS(VS_INPUT input)
{
   VS_OUTPUT output = (VS_OUTPUT)0;
   float4 positionW = mul(float4(input.position, 1.0f), gmtxWorld);
   output.position = mul(mul(positionW, gmtxView), gmtxProjection);
   output.normal = mul(input.normal, gmtxWorld);
   output.toCamera = normalize(gvCameraPosition - positionW.xyz);
   output.toLight = normalize(gvLightPosition - positionW.xyz);

   return(output);
}
```

```
struct VS_INPUT
{
   float4 position : POSITION;
   float4 normal : NORMAL;
};
```

```
struct VS_OUTPUT
{
   float4 position : SV_POSITION;
   float4 color : COLOR;
   float3 normal : TEXCOORD0;
   float3 toCamera : TEXCOORD1;
   float3 toLight : TEXCOORD2;
};
```

```
float4 PS(VS_OUTPUT input) : SV_Target
{
   float3 N = normalize(input.normal);
   float3 L = normalize(input.toLight);
   float3 V = normalize(input.toCamera);
   float NL = dot(L, N);
   float VN = dot(V, N);

   float fSinTan = gtxtSinTan.Sample(gssSinTan, float2(saturate(NL), saturate(VN))).x;
   float3 vLProjected = normalize(L - NL * N);
   float3 vVProjected = normalize(V - VN * N);
   float color = saturate(dot(vLProjected, vVProjected));
   return(color * saturate(NL * (A + B * C * fSinTan)));
}
```

```
Texture2D gtxtSinTan : register(t2);
SamplerState gssSinTan : register(s2);
```