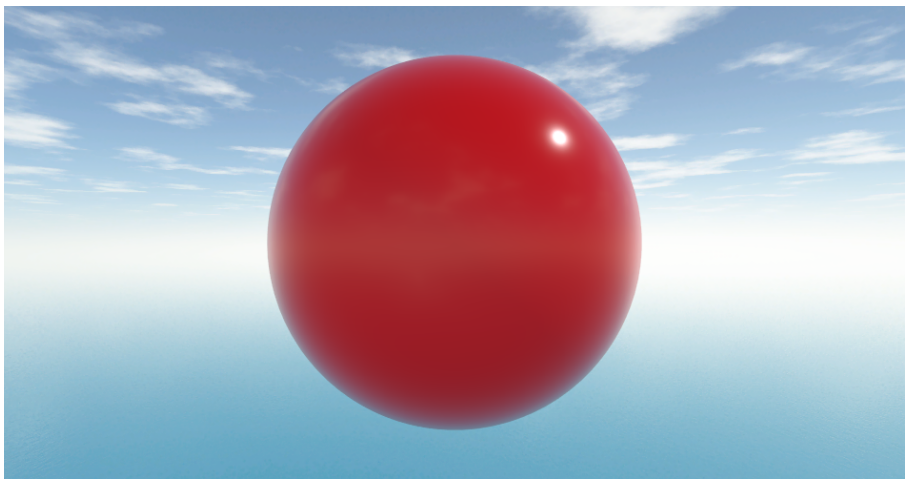
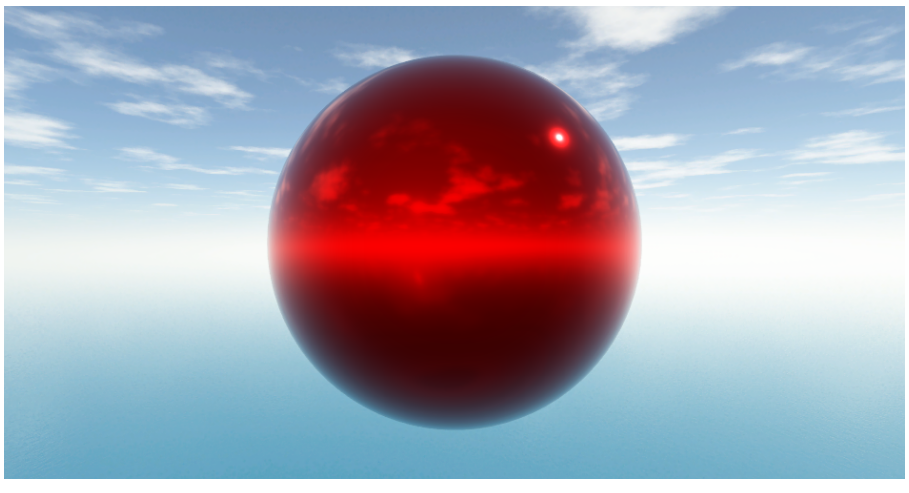
Specular IBL





목차

개요

Spherical Harmonics 글에서 Diffuse IBL을 주제로 Irradiance Map에 대해 알아보았던 것에 이어서 이번에는 Specular IBL에 대해 알아보도록 하겠습니다.

Specular 반사는 Diffuse 반사와 다르게 조명과 카메라의 방향에 따라 음영의 변화가 급격하게 이뤄질 수 있으므로 Diffuse IBL과는 다른 방식으로 접근하게 됩니다.

여기서는 이미지를 통하여 Specular 반사를 계산하기 위한 기반 지식을 알아보고 Direct3D 11/12로 구현한 결과물을 통해 표면의 재질에 따른 비주얼의 차이를 살펴보도록 하겠습니다.

Specular BRDF

Spherical Harmonics 글에서 살펴보았듯이 조명 계산은 다음과 같은 렌더링 방정식을 계산하는 과정입니다.

$$L_\mathrm{r}(\mathbf{x}, \omega_\mathrm{o}) = L_\mathrm{e}(\mathbf{x}, \omega_\mathrm{o}) + \int_\Omega f_\mathrm{r}(\mathbf{x}, \omega_\mathrm{i}, \omega_\mathrm{o}) L_\mathrm{i}(\mathbf{x}, \omega_\mathrm{i}) (\omega_\mathrm{i} \cdot \mathbf{n}) \, \mathrm{d}\, \omega_\mathrm{i}$$

각 항의 자세한 의미는 Spherical Harmonics 글에서 설명하였으므로 해당 글을 참고하시기를 바랍니다.

Diffuse 반사에서는 Lambert BRDF를 사용했지만, Specular 반사를 위해서는 다른 BRDF 함수를 사용할 필요가 있습니다. 지금부터 Specular 반사를 위해 널리 사용되는 Cook-Torrance BRDF를 살펴보겠습니다. Cook-Torrance BRDF는 다음과 같습니다.

$$f_{cook-torrance} = \frac{DFG}{4(w_o \cdot \mathbf{n})(w_i \cdot \mathbf{n})}$$

n : 표면의 법선 벡터

w_o, w_i : 빛이 나가는 방향, 빛이 들어오는 방향

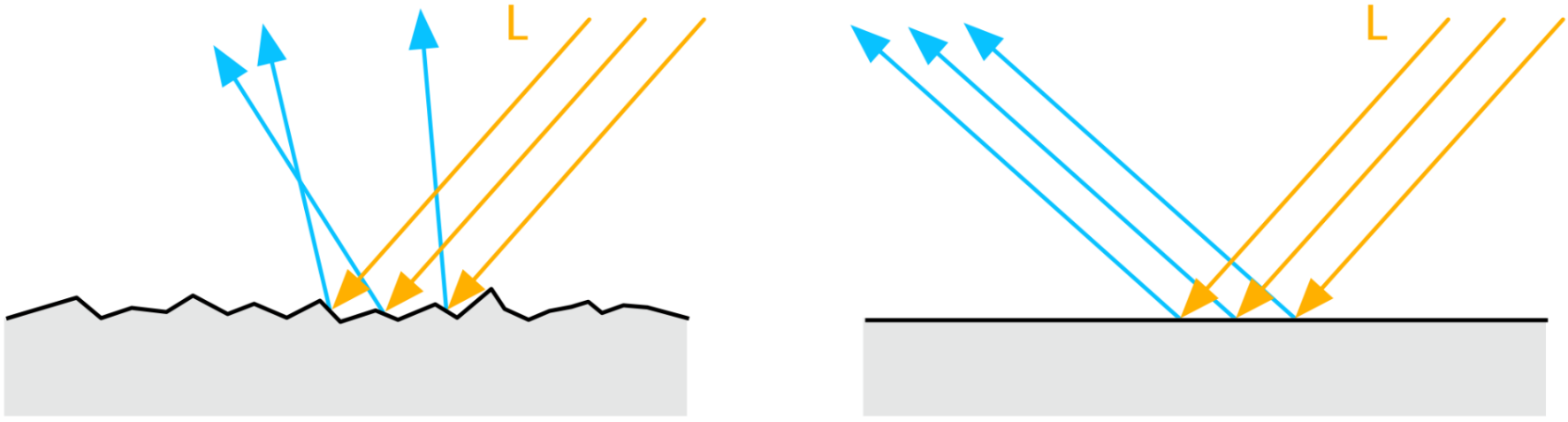
이를 조명 벡터(l)와 카메라 벡터(v)로 치환하면 다음과 같이 표현할 수도 있습니다.

$$f_{cook-torrance} = \frac{DFG}{4(\mathbf{n} \cdot l)(\mathbf{n} \cdot v)}$$

D, F, G는 Cook-Torrance BRDF의 근간을 이루는 함수로 다양한 함수를 선택하여 사용할 수 있으나 여기서는 UE4에서 사용된 함수를 기반으로 하여 각 항을 살펴보도록 하겠습니다.

Specular D

D는 미세면 분포 항입니다. Cook-Torrance BRDF는 표면을 여러 미세한 굴곡을 지닌 표면으로 모델링하는데 이를 미세면(Microfacet) 이라고 합니다.



출처 : https://google.github.io/filament/Filament.html

Roughness는 미세면이 얼마나 거친지를 나타내는 값으로 0이면 매끈하고 1이면 거친 표면을 의미합니다. 미세면 분포 함수는 Roughness에 따라 halfway 벡터에 정확하게 정렬된 미세면의 상대 표면적을 통계적으로 근사합니다. halfway 벡터에 정확하게 정렬된 미세면이 많을수록 해당 표면에서는 정확한 거울반사가 많이 일어나게 됩니다.

함수로는 GGX/Trowbridge-Reitz를 사용합니다.

$$D(h) = \frac{\alpha^2}{\pi((n \cdot h)^2(\alpha^2 - 1) + 1)^2} = \frac{\alpha^2}{\pi((\alpha^2 - 1)cos^2\theta + 1)^2}$$

α : Roughness^2

h : halfway 벡터

코드로는 다음과 같습니다.

```
float DistributionGGX( float ndoth, float roughness )
{
    float a = roughness * roughness;
    float a2 = a * a;
    float ndoth2 = ndoth * ndoth;

    float num = a2;
    float denom = ( ndoth2 * ( a2 - 1.f ) + 1.f );
    denom = PI * denom * denom;

    return num / denom;
}
```

Specular F

F는 프레넬 항입니다. 이 항은 프레넬 효과를 위한 항으로 빛이 물체와 수직에 가까울수록 투과하는 비율이 높아지고 수평과 가까울수록 반사되는 비율이 높아지는 현상입니다.

함수로는 다음과 같은 Schlick's approximation을 사용합니다.

$$F = F_0 + (F_{90} - F_0)(1 - cos(\theta))^5 = F_0 + (1 - F_0)(1 - cos(\theta))^5$$

F_{90} : 입사각이 90도일 때 반사율로 유전체와 도체의 경우 1이므로 대부분 1로 사용합니다.

F_0 : 표면에 수직으로 입사한 빛에 대한 반사율로 다음과 같이 계산합니다.

$$F_0 = \left(\frac{n_1 - n_2}{n_1 + n2} \right)^2$$

n_1, n_2 : 매질의 굴절률 (IOR : indices of refraction)

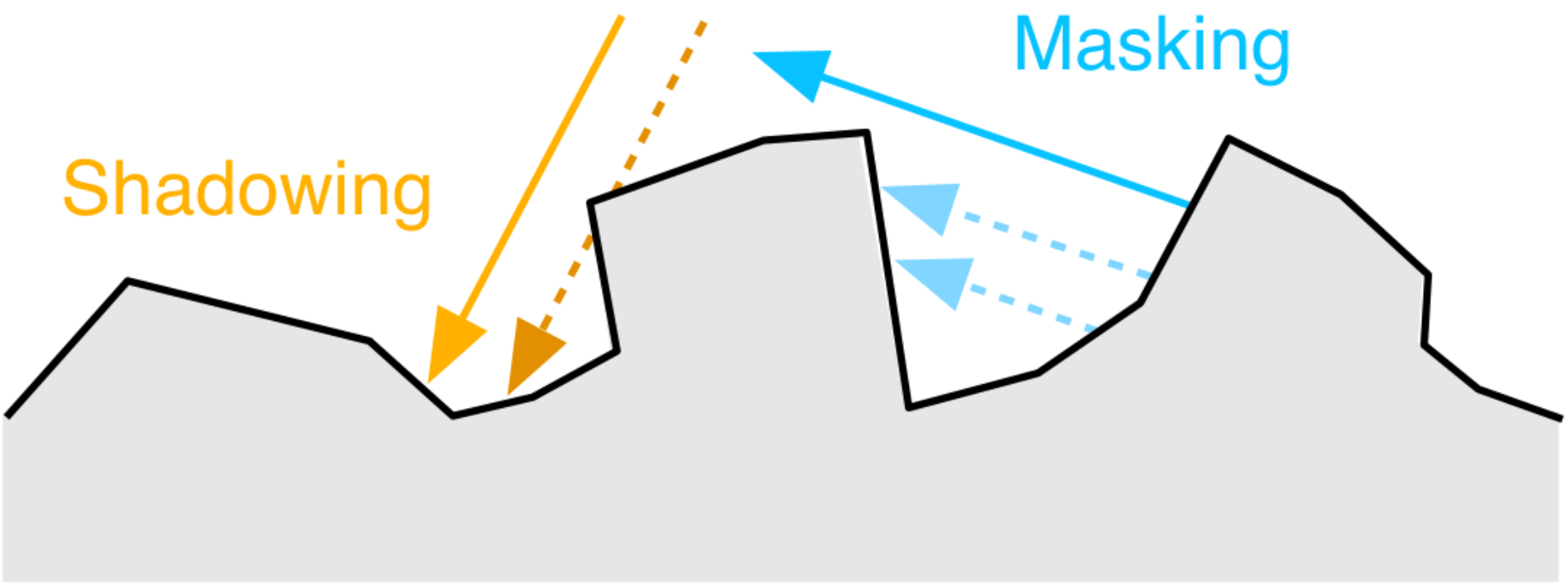
다만 실제 구현에서는 매질의 굴절률을 통해서 F_0을 실제로 계산하지 않고 스페큘러 반사 색상을 F_0으로 사용합니다.

코드로는 다음과 같습니다.

```
float3 FresnelSchlick( float cosTheta, float3 f0 )
{
    return f0 + ( 1.f - f0 ) * pow( saturate( 1.f - cosTheta ), 5.f );
}
```

Specular G

G는 기하 차폐 항입니다. 미세면의 분포에 따라 입사광이 표면에 도달하지 못하거나(Shadowing) 반사광이 카메라에 도달하지 못하는(Masking) 상대적인 표면 면적을 통계적으로 근사합니다.



출처 : https://google.github.io/filament/Filament.html

함수로는 Schlick-GGX로 알려진 GGX와 Schlick-Beckmann approximation을 사용합니다.

$$G(l, v, h) = G_1(l)G_1(v)$$

G_1은 다음과 같습니다.

$$G_1(v) = \frac{n \cdot v}{(n \cdot v)(1 - k) + k}$$

k는 다음과 같습니다.

$$k = \frac{Roughness^2}{2}$$

코드로는 다음과 같습니다.

```
float GeometrySchlickGGXForIBL( float ndotv, float roughness )
{
    float a = roughness;
    float k = ( a * a ) / 2.f;

    float num = ndotv;
    float denom = ndotv * ( 1.f - k ) + k;

    return num / denom;
}

float GemoetrySmithForIBL( float ndotl, float ndotv, float roughness )
{
    float ggx1 = GeometrySchlickGGXForIBL( ndotl, roughness );
    float ggx2 = GeometrySchlickGGXForIBL( ndotv, roughness );

    return ggx1 * ggx2;
}
```

다시 렌더링 방정식으로 돌아와 식을 정리하면 다음 적분을 계산하는 것으로 조명의 Specular 반사를 계산할 수 있습니다.

$$L_{\mathrm{r}}(\mathbf{x}, \omega_{\mathrm{o}}) = \int_{\Omega} \frac{DFG}{4(\mathbf{n} \cdot l)(\mathbf{n} \cdot v)} L_{\mathrm{i}}(\mathbf{x}, \omega_{\mathrm{i}})(\omega_{\mathrm{i}} \cdot \mathbf{n}) \, \mathrm{d}\, \omega_{\mathrm{i}}$$

Monte Carlo Integration

적분을 컴퓨터를 통해 계산하기 위해 Diffuse 반사에서는 리만합을 이용하였습니다. 하지만 Specular 반사에서는 리만합이 아닌 몬테카를로 방법을 사용하여 적분을 계산하게 됩니다. 몬테카를로 방법은 무작위로 추출된 난수를 이용하여 함수의 값을 근사하는 알고리즘으로 계산하려는 값이 복잡한 경우에 사용됩니다.

가령 한 나라의 평균 키를 구하고자 할 때 모든 인구의 키를 조사하는 것은 시간이 매우 오래 걸리는 일입니다. 이렇게 모집단이 방대한 경우에 일부 사람을 무작위로 추출하여 평균을 낸다면 모든 인구의 키를 조사하는 것보다 빠르게 평균 키를 근사할 수 있을 것입니다.

빛이 들어오는 방향과 표면의 거칠기에 독립적이었던 Lambert BRDF와 달리 Cook-Torrance BRDF는 리만합으로 해결하기에는 계산하려는 값이 복잡하므로 몬테카를로 방법을 사용하는 것이 적절합니다. 그럼 몬테카를로 방법을 통해서 어떻게 적분을 계산할 수 있을까요? 우선은 기댓값(Expected value)에 대해 알 필요가 있습니다.

기댓값은 각 사건이 벌어졌을 때의 이득과 그 사건이 벌어질 확률을 곱한 것을 합한 값을 의미합니다. 연속 확률 변수일 경우 다음과 같이 나타내고

$$E[x] = \int_{-\infty}^{\infty} xp(x) \, \mathrm{d}\, x$$

이산 확률 변수일 경우에는 다음과 같습니다.

$$E(x) = \sum_{i} p(x_i) x_i$$

여기서 함수 $p(x)$는 확률 밀도 함수(Probability Density Function, PDF)로 x사건이 일어날 확률을 반환하는 함수입니다.

주사위를 예로 들면 주사위 확률 밀도 함수 $p(x)$는 무조건 $\frac{1}{6}$을 반환하기 때문에 (=주사위의 한 눈이 나올 확률은 $\frac{1}{6}$) 주사위의 기댓값은 다음과 같습니다.

$$E(x) = \sum_{i} p(x_i) x_i = 1 * \frac{1}{6} + 2 * \frac{1}{6} + 3 * \frac{1}{6} + 4 * \frac{1}{6} + 5 * \frac{1}{6} + 6 * \frac{1}{6} = 3.5$$

이제 적분을 계산하기 위해 적분식을 바꿔보겠습니다. 다음과 같은 적분식이 있다고 하면

$$\int_{-\infty}^{\infty} f(x) \, \mathrm{d}\, x$$

어떤 PDF함수를 이용하여 다음과 같이 쓸 수 있습니다.

$$\int_{-\infty}^{\infty} \frac{f(x)}{p(x)} p(x) \, \mathrm{d}\, x$$

앞에서 소개한 기댓값과 비교해 보면 다음과 같이 $\frac{f(x)}{p(x)}$의 기댓값이 $f(x)$에 대한 적분이라는 것을 알 수 있습니다.

$$\int_{-\infty}^{\infty} f(x) \, \mathrm{d}\, x = \int_{-\infty}^{\infty} \frac{f(x)}{p(x)} p(x) \, \mathrm{d}\, x = E\left[\frac{f(x)}{p(x)}\right]$$

이를 N개의 표본을 뽑는 경우로 바꿔 표현하면

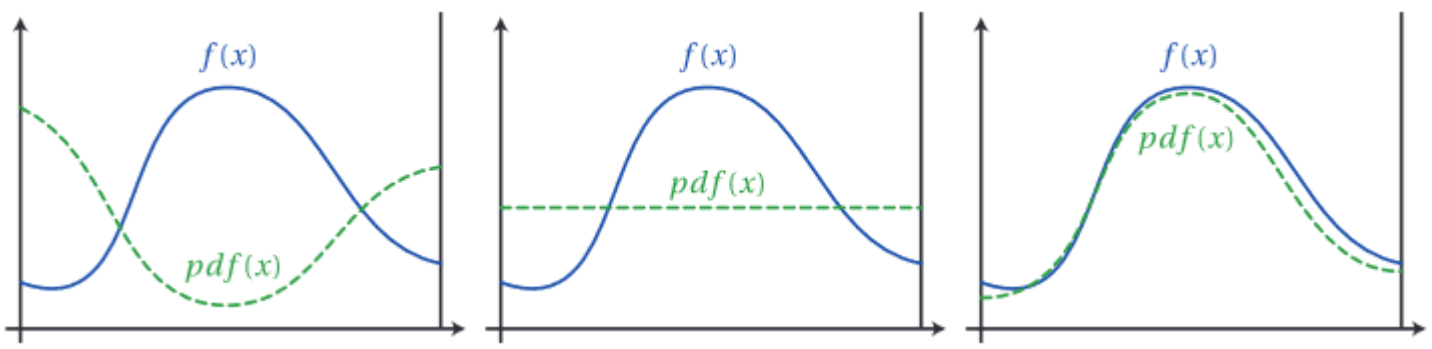
$$E\left[\frac{f(x)}{p(x)}\right] = \frac{1}{N} \sum_{i=1}^{N} \frac{f(x)}{p(x)}$$

가 됩니다.

Importance Sampling

무작위로 추출하는 표본의 수 N이 커질수록 몬테카를로 적분은 큰 수의 법칙에 따라 실제 $f(x)$의 적분값에 가까워집니다. 다만 컴퓨팅 능력에 한계가 있기 때문에 N을 무작정 크게 잡을 수가 없어 최적화를 위한 기법이 적용되는데 Importance Sampling이 그중 하나입니다.

Importance Sampling은 표본을 추출할 때 구하고자 하는 $f(x)$의 분포와 유사한 PDF 함수를 선택하여 표본을 추출하는 방식입니다. 다음 그림을 살펴보겠습니다.



출처 : https://cs.dartmouth.edu/~wjarosz/publications/dissertation/appendixA.pdf

좌측 그림은 $f(x)$와 완전히 다른 분포의 PDF 함수를 선택한 경우입니다. 양 끝의 표본이 추출될 확률이 높은데 해당 표본에서의 $f(x)$함수의 값은 매우 낮기 때문에 실제 적분값이 가까워지기에는 많은 표본 수가 필요합니다. 중간 그림은 균등한 확률로 표본을 추출합니다. $f(x)$의 분포와 잘 맞지는 않지만, 좌측에 비해서는 좀 더 빠르게 적분값에 수렴할 것입니다. 우측 그림은 $f(x)$와 거의 유사한 분포를 가져 적은 수의 표본으로도 적분을 근사할 수 있습니다.

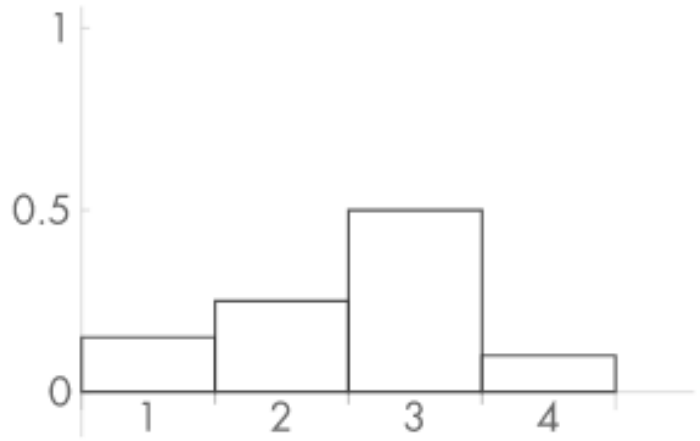
즉 임의의 PDF의 분포를 따르는 표본을 뽑을 수만 있으면 효율적으로 컴퓨터를 통한 적분이 가능합니다.

PDF, CDF, InverseCDF

InverseCDF는 하나 이상의 균등한 분포를 가진 임의의 변수를 사용하여 원하는 분포의 표본을 뽑을 방법을 제공합니다. InverseCDF는 CDF의 역함수로 InverseCDF를 알아보기 위해서는 우선 누적 분포 함수(Cumulative Distribution Function, CDF)가 무엇인지 알아야 합니다.

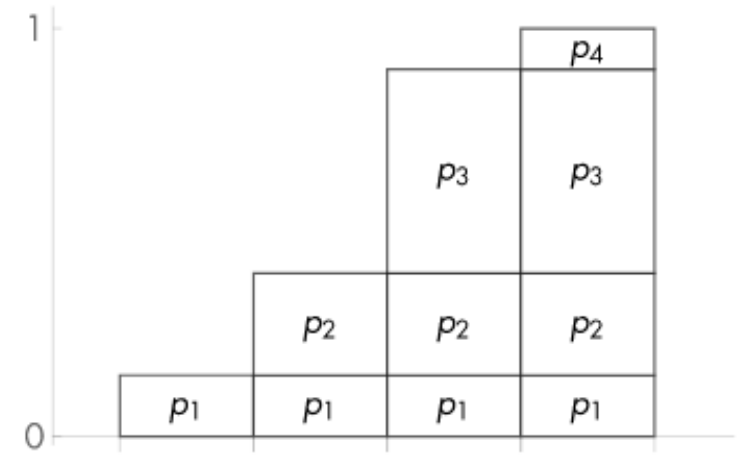
CDF는 확률론에서 주어진 확률 변수가 특정 값보다 작거나 같은 확률을 나타내는 함수로 PDF를 적분하면 CDF, CDF를 미분하면 PDF가 됩니다.

이해를 돕기 위해서 그래프로 살펴보면 어떤 PDF가 다음과 같이 구성되어 있다고 할 때



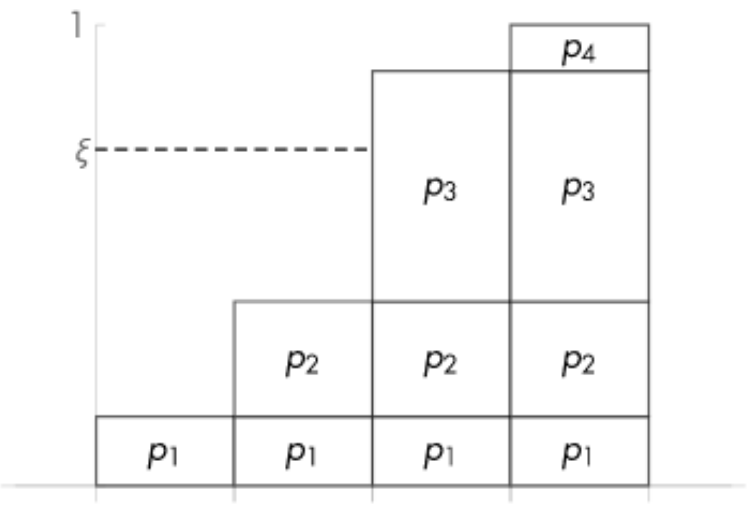
출처 : https://pbr-book.org/3ed-2018/Monte_Carlo_Integration/Sampling_Random_Variables

CDF는 다음과 같습니다.



출처 : https://pbr-book.org/3ed-2018/Monte_Carlo_Integration/Sampling_Random_Variables

그리고 이 CDF의 역을 취하면 원하는 분포로 표본을 뽑을 수 있습니다.



출처 : https://pbr-book.org/3ed-2018/Monte_Carlo_Integration/Sampling_Random_Variables

Full approximation

이제 몬테카를로 적분을 통해 렌더링 방정식을 계산해 보겠습니다. Specular 반사에 대한 몬테카를로 적분식은 다음과 같습니다.

$$L_{\mathrm{r}}(\mathbf{x}, \omega_{\mathrm{o}}) = \int_{\Omega} \frac{DFG}{4(n \cdot l)(n \cdot v)} L_{\mathrm{i}}(\mathbf{x}, \omega_{\mathrm{i}})(\omega_{\mathrm{i}} \cdot \mathbf{n}) \, \mathrm{d}\, \omega_{\mathrm{i}} = \frac{1}{N} \sum_{i=1}^{N} \frac{DFG}{4(n \cdot l)(n \cdot v)p(x)} L_{\mathrm{i}}(\mathbf{x}, l)(\mathbf{n} \cdot l)$$

이제 PDF와 해당 PDF에서 유도된 InverseCDF를 알아내면 모든 준비는 끝납니다.

Cook-Torrance BRDF의 각 항중 가장 지배적인 항은 D항입니다. 따라서 D항의 분포를 따르는 PDF를 사용할 것입니다. PDF를 유도하기 전에 우선 D항이 따라야 하는 한 가지 법칙을 살펴보겠습니다.

$$\int_{\Omega} D(h) cos\theta_h \, \mathrm{d}\, \omega_h = 1$$

즉 D항에 cos 가중치를 적용하여 반구에 대해 적분한 결과가 1이라는 것입니다. 이를 PDF를 전 구간에 대해 적분한 결과가 1이라는 점($\int_{-\infty}^{\infty} p(x)\,\mathrm{d}\,x = 1$)에 적용하면 입체각을 통해 나타낸 PDF가 다음과 같다는 것을 알 수 있습니다.

$$D(h) = \frac{\alpha^2}{\pi((\alpha^2-1)cos^2\theta+1)^2}$$

$$p_h(\omega) = D(h)cos\theta = \frac{\alpha^2 cos\theta}{\pi((\alpha^2-1)cos^2\theta+1)^2}$$

추가로 입체각을 직접 사용하지 않고 구형 좌표계를 사용하도록 변경하면 다음과 같습니다. ($dw = sin\theta\,\mathrm{d}\,\theta\,\mathrm{d}\,\phi$)

$$p_h(\theta,\phi) = \frac{\alpha^2 cos\theta sin\theta}{\pi((\alpha^2-1)cos^2\theta+1)^2}$$

이 방정식에는 ϕ가 포함되어 있기 않기 때문에 다음과 같이 θ에 대한 식으로 변경할 수 있습니다.

$$p_h(\theta) = \int_0^{2\pi} p_h(\theta,\phi)\,\mathrm{d}\,\phi = \left[\frac{\alpha^2 cos\theta sin\theta}{\pi((\alpha^2-1)cos^2\theta+1)^2}\phi\right]_0^{2\pi} = \frac{2\alpha^2 cos\theta sin\theta}{((\alpha^2-1)cos^2\theta+1)^2}$$

이것으로 PDF를 구했습니다. 이제 PDF를 적분하면 CDF라는 점을 이용해서 다음과 같은 순서로 CDF를 구할 수 있습니다.

$$P_h(\theta) = \int_0^{\theta} \frac{2\alpha^2 cos(t)sin(t)}{((\alpha^2-1)cos^2t+1)^2}\,\mathrm{d}\,t$$

$u = cos^2t$로 치환 적분

$\mathrm{d}\,u = -2cos(t)sin(t)\,\mathrm{d}\,t$

$\mathrm{d}\,t = \frac{\mathrm{d}\,u}{-2cos(t)sin(t)}$

$$P_h(\theta) = \int_0^{\theta} \frac{-\alpha^2}{((\alpha^2-1)u+1)^2}\,\mathrm{d}\,u$$

$k = \frac{1}{(\alpha^2-1)u+1}$로 치환 적분

$\mathrm{d}\,k = -\frac{\alpha^2-1}{((\alpha^2-1)u+1)^2}\,\mathrm{d}\,u$

$\mathrm{d}\,u = -\frac{((\alpha^2-1)u+1)^2}{\alpha^2-1}\,\mathrm{d}\,k$

$$P_h(\theta) = \int_0^{\theta} \frac{\alpha^2}{\alpha^2-1}\,\mathrm{d}\,k$$

$$P_h(\theta) = \frac{\alpha^2}{\alpha^2-1}\int_0^{\theta} \mathrm{d}\,k$$

$$P_h(\theta) = \frac{\alpha^2}{\alpha^2-1}\,[k]_0^{\theta} = \frac{\alpha^2}{\alpha^2-1}\left[\frac{1}{(\alpha^2-1)u+1}\right]_0^{\theta} = \frac{\alpha^2}{\alpha^2-1}\left[\frac{1}{(\alpha^2-1)cos^2t+1}\right]_0^{\theta}$$

$$P_h(\theta) = \frac{\alpha^2}{\alpha^2-1}\left(\frac{1}{(\alpha^2-1)cos^2\theta+1} - \frac{1}{\alpha^2}\right)$$

따라서 최종적으로 CDF는 다음과 같습니다.

$$P_h(\theta) = \left(\frac{\alpha^2}{(\alpha^2-1)^2cos^2\theta+\alpha^2-1} - \frac{1}{\alpha^2-1}\right)$$

이제 InverseCDF를 구해보겠습니다.

$$\epsilon = \left(\frac{\alpha^2}{(\alpha^2-1)^2cos^2\theta+\alpha^2-1} - \frac{1}{\alpha^2-1}\right)$$

$u = \alpha^2 - 1$로 치환

$$\epsilon = \left(\frac{\alpha^2}{u^2 cos^2\theta + u} - \frac{1}{u} \right)$$

$$\epsilon = \left(\frac{\alpha^2}{u^2 cos^2\theta + u} - \frac{(cos^2\theta u + 1)}{u(cos^2\theta u + 1)} \right)$$

$$\epsilon = \left(\frac{\alpha^2 - (cos^2\theta u + 1)}{u^2 cos^2\theta + u} \right)$$

$$\epsilon(u^2 cos^2\theta + u) = \alpha^2 - (cos^2\theta u + 1)$$

$\alpha^2 = u + 1$이므로

$$\epsilon(u^2 cos^2\theta + u) = u - cos^2\theta u$$

$$\epsilon(u cos^2\theta + 1) = 1 - cos^2\theta$$

$$u cos^2\theta \epsilon + \epsilon = 1 - cos^2\theta$$

$$cos^2\theta(u\epsilon + 1) = 1 - \epsilon$$

$$cos^2\theta = \frac{1 - \epsilon}{u\epsilon + 1}$$

$$cos\theta = \sqrt{\frac{1 - \epsilon}{u\epsilon + 1}}$$

따라서 최종적으로 InverseCDF는 다음과 같습니다.

$$\theta = arccos\sqrt{\frac{1 - \epsilon}{u\epsilon + 1}} = arccos\sqrt{\frac{1 - \epsilon}{(\alpha^2 - 1)\epsilon + 1}}$$

InverseCDF를 통해 샘플링을 위한 halfway 벡터를 반환하는 셰이더 코드를 살펴보겠습니다.

```
float3 ImportanceSampleGGX( float2 xi, float3 normal, float roughness )
{
    float a = roughness * roughness;

    float phi = 2.f * PI * xi.x;
    float cosTheta = sqrt( ( 1.f - xi.y ) / ( 1.f + ( a * a - 1.f ) * xi.y ) ); // InverseCDF
    float sinTheta = sqrt( 1.f - cosTheta * cosTheta );

    float3 h = { cos( phi ) * sinTheta, sin( phi ) * sinTheta, cosTheta };

    float3 up = ( abs( normal.y ) < 0.999f ) ? float3( 0.f, 1.f, 0.f ) : float3( 0.f, 0.f, 1.
f );
  float3 right = normalize( cross( up, normal ) );
  up = normalize( cross( normal, right ) );

  float3x3 toWorld = float3x3( right, up, normal );

    return normalize( mul( h, toWorld ) );
}
```

매개 변수 xi는 Hammersley를 통해 생성된 저불일치 점으로 다음과 같은 코드를 통해 생성합니다.

```
// VanDerCorput을 다차원으로 확장하면 Halton 수열
float VanDerCorput( uint bits )
{
    bits = ( bits << 16 ) | ( bits >> 16 );
```

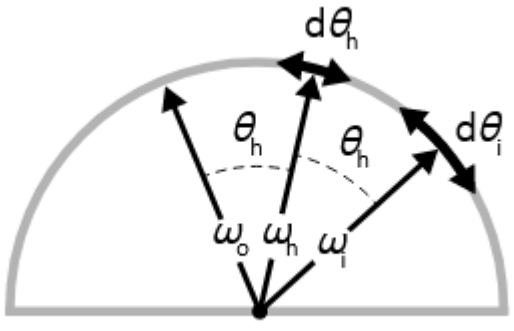
```
  bits = ( ( bits & 0x55555555 ) << 1 ) | ( ( bits & 0xAAAAAAAA ) >> 1 );
  bits = ( ( bits & 0x33333333 ) << 2 ) | ( ( bits & 0xCCCCCCCC ) >> 2 );
  bits = ( ( bits & 0x0F0F0F0F ) << 4 ) | ( ( bits & 0xF0F0F0F0 ) >> 4 );
  bits = ( ( bits & 0x00FF00FF ) << 8 ) | ( ( bits & 0xFF00FF00 ) >> 8 );
    return float( bits ) * 2.3283064365386963e-10; // 0x100000000
}

float2 Hammersley( uint i, uint n )
{
    return float2( float( i ) / float( n ), VanDerCorput( i ) );
}
```

이제 PDF도 구했고 샘플링을 위한 벡터도 구했습니다. 렌더링 방정식의 적분을 풀기 위한 모든 준비가 끝났을까요? 안타깝게도 한가지 빠진 부분이 있습니다. 그 전에 먼저 최종 코드를 살펴보도록 하겠습니다.

```
float3 SpecularIBL( float3 SpecularColor , float Roughness , float3 N, float3 V )
{
    float3 SpecularLighting = 0;
    const uint NumSamples = 1024;
    for( uint i = 0; i < NumSamples; i++ )
    {
        float2 Xi = Hammersley( i, NumSamples );
        float3 H = ImportanceSampleGGX( Xi, Roughness , N );
        float3 L = 2 * dot( V, H ) * H - V;
        float NoV = saturate( dot( N, V ) );
        float NoL = saturate( dot( N, L ) );
        float NoH = saturate( dot( N, H ) );
        float VoH = saturate( dot( V, H ) );
        if( NoL > 0 )
        {
            float3 SampleColor = EnvMap.SampleLevel( EnvMapSampler , L, 0 ).rgb;
            float G = G_Smith( Roughness , NoV, NoL );
            float Fc = pow( 1 - VoH, 5 );
            float3 F = (1 - Fc) * SpecularColor + Fc;
            // Incident light = SampleColor * NoL
            // Microfacet specular = D*G*F / (4*NoL*NoV)
            // pdf = D * NoH / (4 * VoH)
            SpecularLighting += SampleColor * F * G * VoH / (NoH * NoV);
        }
    }
    return SpecularLighting / NumSamples;
}
```

우리가 알고 있는 PDF는 $D(h)cos\theta$ 인데 4 * VoH는 어디서 나온 것일까요? 이는 우리가 구한 PDF가 halfway 벡터에 대한 PDF이기 때문입니다. 반면 렌더링 방정식은 halfway 벡터가 아닌 빛이 들어온 방향 벡터에 대한 식이기 때문에 입사 방향(w_i)에 대한 확률 밀도가 필요합니다.

간단한 기하학적 구조를 통해서 우리는 halfway 벡터의 분포와 입사 벡터의 분포 간의 관계를 알 수 있습니다.



출처 : https://pbr-book.org/3ed-2018/Light_Transport_I_Surface_Reflection/Sampling_Reflection_Functions

위와 같이 w_o를 기준으로 하면 입사 벡터는 halfway 벡터의 2배입니다. 따라서 $\theta_i = 2\theta_h$가 되고 등방성 반사이기 때문에 $\phi_i = \phi_h$가 됩니다. 이때 입사 벡터에서 halfway 벡터로의 변환은 1대1 대응 관계를 맺는 함수이므로 이 두 벡터의 PDF는 다음식을 만족합니다.

$$p_h(w_h)\frac{\mathrm{d}\,w_h}{\mathrm{d}\,w_i} = p_i(w_i)$$

> 위와 같은 식이 성립하는 이유는 여기를 참고 바랍니다.

여기서 $\frac{\mathrm{d}\,w_h}{\mathrm{d}\,w_o}$을 구해보면

$$\frac{\mathrm{d}\,w_h}{\mathrm{d}\,w_o} = \frac{sin\theta_h\,\mathrm{d}\,\theta_h\,\mathrm{d}\,\phi_h}{sin\theta_i\,\mathrm{d}\,\theta_i\,\mathrm{d}\,\phi_i} = \frac{sin\theta_h\,\mathrm{d}\,\theta_h\,\mathrm{d}\,\phi_h}{sin2\theta_h\,\mathrm{d}\,2\theta_h\,\mathrm{d}\,\phi_h}$$

$$\frac{\mathrm{d}\,w_h}{\mathrm{d}\,w_o} = \frac{sin\theta_h}{4cos\theta_h sin\theta_h} = \frac{1}{4cos\theta_h} = \frac{1}{4(w_i \cdot w_h)} = \frac{1}{4(w_o \cdot w_h)}$$

가 됩니다. 그러므로 최종적으로 몬테카를로 적분식을 정리하면 다음과 같습니다.

$$\frac{1}{N}\sum_{i=1}^{N}\frac{FG(v\cdot h)}{(\mathbf{n}\cdot v)(\mathbf{n}\cdot h)}L_{\mathrm{i}}(\mathbf{x},l)$$

Split sum approximation

이렇게 컴퓨터로 계산할 수 있는 식을 얻어냈지만, Diffuse IBL와 마찬가지로 해당 식을 실시간에 계산하는 것은 성능에 치명적입니다. 따라서 Irradiance Map과 같은 전처리 과정이 필요합니다.

문제는 BRDF가 상수항($\frac{\sigma}{\pi}$)이였기 때문에 입사광의 방향에만 의존하던 Irradiance Map과 달리 Specular IBL은 반사광의 방향에도 의존적이라는 점입니다. 입사광과 반사광의 조합을 모두 고려하여 미리 적분을 계산하는 것은 여전히 까다롭습니다.

그래서 Epic Games에서는 다음과 같이 적분을 2부분으로 나누어 이를 미리 계산하고 실시간에 합쳐서 적분식을 근사하는 것으로 이를 해결하고자 하였습니다. 이것이 Split sum approximation입니다.

$$L_{\mathrm{r}}(\mathbf{x},\omega_{\mathrm{o}}) \approx \int_{\Omega} L_{\mathrm{i}}(\mathbf{x},\omega_{\mathrm{i}})\,\mathrm{d}\,\omega_{\mathrm{i}} * \int_{\Omega} f_{\mathrm{r}}(\mathbf{x},\omega_{\mathrm{i}},\omega_{\mathrm{o}})(\omega_{\mathrm{i}}\cdot\mathbf{n})\,\mathrm{d}\,\omega_{\mathrm{i}}$$

첫 번째 부분은 Irradiance Map과 유사한 전처리 과정을 통해 구할 수 있는데 이번에는 거칠기를 고려해야 합니다. 거칠기가 높아질수록 샘플링 벡터가 넓게 흩어지게 되고 환경 맵 또한 흐려집니다. 따라서 낮은 거칠기에서의 이미지보다 낮은 해상도로도 충분히 저장할 수 있을 정도가 되기 때문에 이를 밉맵으로 저장할 수 있습니다. 이러한 전처리 과정이 끝난 텍스쳐를 Prefiltered Environment Map이라 부릅니다.

전처리를 수행하는 컴퓨트 셰이더 코드를 살펴보겠습니다.

```
#include "Common/PBR.fxh"

cbuffer PrefilterSpecularParameter : register( b0 )
{
    float Roughness;
};

TextureCube EnvMap;
SamplerState EnvMapSampler;
RWTexture2DArray<float4> Prefiltered;

[numthreads(8, 8, 1)]
void main( uint3 DTid : SV_DispatchThreadId )
{
    uint3 dims = (uint3)0;
    Prefiltered.GetDimensions( dims.x, dims.y, dims.z );

    if ( all( DTid < dims ) )
    {
        float2 uv = float2( DTid.xy + 0.5f ) / dims.xy;
```

```
        uv = ( uv - 0.5f ) * float2( 2.f, -2.f );

        // https://learn.microsoft.com/en-us/windows/win32/direct3d9/cubic-environment-mappin
g
        float3 normal = 0;
        if ( DTid.z == 0 ) // +x
        {
            normal = float3( 1.f, uv.y, -uv.x );
        }
        else if ( DTid.z == 1 ) // -x
        {
            normal = float3( -1.f, uv.y, uv.x );
        }
        else if ( DTid.z == 2 ) // +y
        {
            normal = float3( uv.x, 1.f, -uv.y );
        }
        else if ( DTid.z == 3 ) // -y
        {
            normal = float3( uv.x, -1.f, uv.y );
        }
        else if ( DTid.z == 4 ) // +z
        {
            normal = float3( uv.x, uv.y, 1.f );
        }
        else // -z
        {
            normal = float3( -uv.x, uv.y, -1.f );
        }

        normal = normalize( normal );
        float3 prefilteredSpecular = PrefilterSpecular( normal, Roughness, EnvMap, EnvMapSamp
ler );
        Prefiltered[DTid] = float4( prefilteredSpecular, 0.f );
    }
}
```

실제 작업은 PrefilterSpecular 함수에서 수행됩니다. 앞에서 이야기했던 것처럼 거칠기를 고려한 샘플링 방향 벡터를 얻어서 해당 방향의 텍스쳐를 누적하여 적분을 계산합니다. 코드에서 주목할 만한 부분은 가중치로 $cos\theta$를 사용한다는 점이며 Epic Games에 따르면 이를 통해서 더 나은 결과를 얻을 수 있었다고 합니다.

```
float3 PrefilterSpecular( float3 normal, float roughness, TextureCube envMap, SamplerState en
vMapSampler )
{
    float3 viewDirection = normal;

    const uint SampleCount = 1024;
    float totalWeight = 0.f;
    float3 prefilteredColor = (float3)0.f;

    for ( uint i = 0; i < SampleCount; ++i )
    {
        float2 xi = Hammersley( i, SampleCount );
        float3 halfWay = ImportanceSampleGGX( xi, normal, roughness );
        float3 toLight = normalize( 2.f * dot( viewDirection, halfWay ) * halfWay - viewDirec
tion );

        float ndotl = saturate( dot( normal, toLight ) );
```

```
        if (ndotl > 0.f)
        {
            prefilteredColor += envMap.SampleLevel( envMapSampler, toLight, 0 ).rgb * ndotl;
            totalWeight += ndotl;
        }
    }

    return prefilteredColor / totalWeight;
}
```

이어서 두 번째 부분을 살펴보겠습니다.

$$\int_\Omega f_\mathrm{r}(\mathbf{x}, \omega_\mathrm{i}, \omega_\mathrm{o})(\omega_\mathrm{i} \cdot \mathbf{n}) \, \mathrm{d}\, \omega_\mathrm{i}$$

두 번째 부분을 미리 계산하기 위해 식을 변경하여 F_0을 적분 밖으로 빼낼 필요가 있습니다. 앞에서 살펴본 Specular BRDF를 이루는 각 항을 주의 깊게 살펴보면 F_0는 재질의 영향을 받고 있기 때문에 이를 포함하여 계산하면 특정 재질에 종속되어 재사용할 수 없게 됩니다.

이제 F_0을 빼내는 과정을 차례차례 살펴보겠습니다.

$$\int_\Omega \frac{f_\mathrm{r}(\mathbf{x}, \omega_\mathrm{i}, \omega_\mathrm{o})}{F(\omega_o \cdot h)} F(\omega_o \cdot h)(\omega_\mathrm{i} \cdot \mathbf{n}) \, \mathrm{d}\, \omega_\mathrm{i}$$

$$\int_\Omega \frac{f_\mathrm{r}(\mathbf{x}, \omega_\mathrm{i}, \omega_\mathrm{o})}{F(\omega_o \cdot h)} (F_0 + (1 - F_0)(1 - \omega_o \cdot h)^5)(\omega_\mathrm{i} \cdot \mathbf{n}) \, \mathrm{d}\, \omega_\mathrm{i}$$

그리고 식을 풀기 쉽게하기 위해 $(1 - \omega_o \cdot h)^5$를 α로 치환하겠습니다.

$$\int_\Omega \frac{f_\mathrm{r}(\mathbf{x}, \omega_\mathrm{i}, \omega_\mathrm{o})}{F(\omega_o \cdot h)} (F_0 + (1 - F_0)\alpha)(\omega_\mathrm{i} \cdot \mathbf{n}) \, \mathrm{d}\, \omega_\mathrm{i}$$

$$\int_\Omega \frac{f_\mathrm{r}(\mathbf{x}, \omega_\mathrm{i}, \omega_\mathrm{o})}{F(\omega_o \cdot h)} (F_0 + \alpha - F_0\alpha)(\omega_\mathrm{i} \cdot \mathbf{n}) \, \mathrm{d}\, \omega_\mathrm{i}$$

$$\int_\Omega \frac{f_\mathrm{r}(\mathbf{x}, \omega_\mathrm{i}, \omega_\mathrm{o})}{F(\omega_o \cdot h)} (F_0(1 - \alpha) + \alpha)(\omega_\mathrm{i} \cdot \mathbf{n}) \, \mathrm{d}\, \omega_\mathrm{i}$$

$$\int_\Omega \frac{f_\mathrm{r}(\mathbf{x}, \omega_\mathrm{i}, \omega_\mathrm{o})}{F(\omega_o \cdot h)} F_0(1 - \alpha)(\omega_\mathrm{i} \cdot \mathbf{n}) \, \mathrm{d}\, \omega_\mathrm{i} + \int_\Omega \frac{f_\mathrm{r}(\mathbf{x}, \omega_\mathrm{i}, \omega_\mathrm{o})}{F(\omega_o \cdot h)} (\alpha)(\omega_\mathrm{i} \cdot \mathbf{n}) \, \mathrm{d}\, \omega_\mathrm{i}$$

$$F_0 \int_\Omega \frac{f_\mathrm{r}(\mathbf{x}, \omega_\mathrm{i}, \omega_\mathrm{o})}{F(\omega_o \cdot h)} (1 - (1 - \omega_o \cdot h)^5)(\omega_\mathrm{i} \cdot \mathbf{n}) \, \mathrm{d}\, \omega_\mathrm{i} + \int_\Omega \frac{f_\mathrm{r}(\mathbf{x}, \omega_\mathrm{i}, \omega_\mathrm{o})}{F(\omega_o \cdot h)} ((1 - \omega_o \cdot h)^5)(\omega_\mathrm{i} \cdot \mathbf{n}) \, \mathrm{d}\, \omega_\mathrm{i}$$

가 되어 F_0을 적분 밖으로 빼낼 수 있고 이는 실시간 렌더링 과정에서 적용하게 됩니다. 또한 $f_\mathrm{r}(\mathbf{x}, \omega_\mathrm{i}, \omega_\mathrm{o})$에는 이미 $F(\omega_o \cdot h)$항이 존재하기 때문에 상쇄되어 $f_\mathrm{r}(\mathbf{x}, \omega_\mathrm{i}, \omega_\mathrm{o})$에서 $F(\omega_o \cdot h)$항이 제거됩니다. 따라서 몬테카를로 적분을 적용한 최종 계산식은 다음과 같습니다.

$$\frac{1}{N} \sum_{i=1}^{N} \frac{G(v \cdot h)}{(\mathbf{n} \cdot v)(\mathbf{n} \cdot h)} (1 - (1 - \omega_o \cdot h)^5) + \frac{1}{N} \sum_{i=1} \frac{G(v \cdot h)}{(\mathbf{n} \cdot v)(\mathbf{n} \cdot h)} (1 - \omega_o \cdot h)^5$$

위 식의 앞부분 뒷부분을 계산하여 512x512 RG16텍스쳐의 x채널, y채널에 각각 기록합니다.

```
agl::RefHandle<agl::Texture> CreateBRDFLookUpTexture()
{
    agl::TextureTrait trait = {
        .m_width = 512,
        .m_height = 512,
        .m_depth = 1,
        .m_sampleCount = 1,
        .m_sampleQuality = 0,
        .m_mipLevels = 1,
        .m_format = agl::ResourceFormat::R16G16_FLOAT,
```

```
        .m_access = agl::ResourceAccessFlag::Default,
        .m_bindType = agl::ResourceBindType::ShaderResource | agl::ResourceBindType::RandomAc
cess,
        .m_miscFlag = agl::ResourceMisc::None
    };

    agl::RefHandle<agl::Texture> brdfLUT = agl::Texture::Create( trait, "BrdfLookUpTexture",
nullptr );
    EnqueueRenderTask( [brdfLUT]()
        {
            brdfLUT->Init();

            PrecomputedBrdfCS precomputedBrdfCS;
            agl::RefHandle<agl::ComputePipelineState> pso = PrepareComputePipelineState( prec
omputedBrdfCS );

            auto commandList = GetCommandList();
            commandList.BindPipelineState( pso );

            agl::ShaderBindings shaderBindings = CreateShaderBindings( precomputedBrdfCS );
            BindResource( shaderBindings, precomputedBrdfCS.Precomputed(), brdfLUT );

            commandList.BindShaderResources( shaderBindings );

            commandList.Dispatch( 512 / 8, 512 / 8 );

            commandList.AddTransition( Transition( *brdfLUT.Get(), agl::ResourceState::PixelS
haderResource ) );

            commandList.Commit();
            GetInterface<agl::IAgl>()->WaitGPU();
        } );

    return brdfLUT;
}
```

텍스쳐의 u좌표가 $(\mathbf{n} \cdot v)$, v좌표가 거칠기입니다.

```
#include "Common/PBR.fxh"

RWTexture2D<float2> Precomputed;

[numthreads(8, 8, 1)]
void main( uint3 DTid : SV_DispatchThreadId )
{
    uint2 dims = (uint2)0;
    Precomputed.GetDimensions( dims.x, dims.y );

    if ( all( DTid < dims ) )
    {
        float2 uv = ( (float2)DTid.xy + 0.5f ) / dims;
        Precomputed[DTid.xy] = IntegrateBRDF( uv.x, uv.y );
    }
}
```

IntegrateBRDF함수는 다음과 같습니다.

```
float2 IntegrateBRDF( float ndotv, float roughness )
{
    float3 viewDirection = { sqrt( 1.f - ndotv * ndotv ), 0.f, ndotv };

    float a = 0.f;
    float b = 0.f;

    float3 normal = { 0.f, 0.f, 1.f };

    const uint SampleCount = 1024;
    for ( uint i = 0; i < SampleCount; ++i )
    {
        float2 xi = Hammersley( i, SampleCount );
        float3 halfWay = ImportanceSampleGGX( xi, normal, roughness );
        float3 toLight = normalize( 2.f * dot( viewDirection, halfWay ) * halfWay - viewDirec
tion );

        float ndotl = saturate( toLight.z );
        float ndoth = saturate( halfWay.z );
        float vdoth = saturate( dot( viewDirection, halfWay ) );

        if ( ndotl > 0.f )
        {
            float g = GemoetrySmithForIBL( ndotl, ndotv, roughness );
            float gVis = ( g * vdoth ) / ( ndoth * ndotv );
            float fc = pow( 1.f - vdoth, 5.f );

            a += ( 1.f - fc ) * gVis;
            b += fc * gVis;
        }
    }

    a /= (float)SampleCount;
    b /= (float)SampleCount;

    return float2( a, b );
}
```

최종 렌더링 시에는 사전 계산한 결과를 다음과 같이 합치게 됩니다. 적분 밖으로 빼낸 F_0(코드에서 F)가 적용되는 것을 볼 수 있습니다.

```
float3 r = reflect( -viewDirection, normal );
float3 prefilteredColor = PrefilterMap.SampleLevel( LinearSampler, r, surface.roughness * ( R
eflectionMipLevels - 1 ) ).rgb  ;
float2 brdf = BrdfLUT.Sample( LinearSampler, float2( ndotv, surface.roughness ) ).rg;

cColor.m_specular.rgb += prefilteredColor * ( F * brdf.x + brdf.y );
```

마치며

준비한 내용은 여기까지 입니다. 세부 코드는 아래의 Changelist들을 참고 부탁드립니다. 감사합니다.

https://github.com/xtozero/SSR/commit/e56e46b9f8a97bc505e467302d51f4c4d16b318c

https://github.com/xtozero/SSR/commit/0040afadbb70804672fb8128b8fd066119e91282

Reference

- https://blog.selfshadow.com/publications/s2013-shading-course/karis/s2013_pbs_epic_notes_v2.pdf
- https://learnopengl.com/PBR/IBL/Specular-IBL
- https://scahp.tistory.com/96
- https://pbr-book.org/3ed-2018/Light_Transport_I_Surface_Reflection/Sampling_Reflection_Functions