

(별지 제12호)

## 지 작 권 양 도 서 (Copyright Transfer Form)

소속 : 인하대학교 정보통신공학과

성명 : 석예찬, 김현진

학번 : 12141719, 12121627

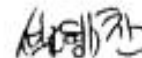
논문제목 : 맵스텝을 통해 계산된 중요도각을 이용한 레이트레이싱 엠비언트 오를루전

본인은 상기 논문을 2018학년도 2학기 정보통신프로젝트 최종 보고서 겸 결과 논문으로 제출하고자 합니다. 본 논문의 내용은 저자가 직접 연구한 결과인 것과 이전에 출판된 적이 없음을 확인합니다. 또한 공저자와 더불어 인하대학교 정보통신공학부에서 발간하는 논문집에 본 논문을 수록하는 것을 허락하며 제반 저작권을 정보통신공학부에 양도합니다.

년 월 일

주저자 :

석예찬



공저자 :

김현진



정보통신공학과장 귀하

(별지 서식 8)

## 맵스맵을 통해 계산된 중요도각을 이용한 레이트 레이싱 앰비언트 오클루전

### Ray Traced Ambient Occlusion using Importance Angle calculated by Depth Map

석예찬, 김현진  
(Yechan Seok and Hyunjin Kim)



**Abstract:** 컴퓨터 그래픽스에서 전역 조명(global illumination) 렌더링을 위해 빛의 차폐 정도를 나타내는 앰비언트 오클루전(ambient occlusion)을 계산한다. 이를 정확하게 계산하기 위해서는 모든 방향으로 재귀적인 레이브레이싱을 수행해야 한다. 최근 이를 실시간으로 돌리기 위해서 픽셀당 1~2개의 레이를 랜덤하게 샘플링한 후에 AI Denoise[1] 시키는 방법이 발표되었다. 하지만 적제 샘플로 denoise된 이미지가 시점이 바뀔 때 갑자기 색이 바뀌는 flickering 현상이 나타난다. 우리는 이를 해결하기 위해 맵스맵에서 차폐될 수 있는 영역만 판별하여 중요 각도 범위를 계산한 뒤에 모든 방향이 아닌 중요각으로 랜덤 샘플링 하는 방법을 제안한다. 맵스맵은 기본적인 렌더링인 레스터라이제이션 이후 얻을 수 있는 정보이고, 중요각 계산은 커널 사이즈를 정해 픽셀마다 다른 주변 픽셀과의 각도를 병렬적으로 계산한다. 이 알고리즘은 중요각도 계산에서 맵스맵이라는 스크린 공간 정보만 이용하기 때문에 더 무질확하다는 단점이 있지만 결과적으로 flickering현상을 개선하였다. 추가되는 연산량은 레이브레이싱과 분리되어있기 때문에 상수 시간이 더해지는 것뿐이므로 더 효율적이다.

**Abstract:** Ambient occlusion, which indicates the degree of occlusion of light, needs to be calculated for rendering global illumination in computer graphics. Accurate calculation of the ambient occlusion requires recursive ray-tracing in all directions, which is time consuming. Recently, it was reported that sampling one or two rays per pixel randomly and conducting AI Denoise[1] could reduce the computation time. This method, however, shows flickering artifacts when the point of view changes, which becomes severe as the number of shooting rays decreases. To solve this problem, we propose a new method. The proposed method determines only the areas that can be occluded from the depth-map, calculates the range of critical angles, and then performs the random sampling only within the critical angles, not in all directions. Depth-map is information that can be obtained after rasterization, which is a basic rendering procedure, and the important angles are calculated in parallel with each pixel by specifying a kernel size. Although this algorithm is more inaccurate because it uses only the screen space information called Depth-map in calculating critical angles, it improves the flickering phenomenon significantly. The additional computations are more efficient because they are separated from Ray Tracing and therefore only add constant time.

**Keywords:** ambient occlusion, ray tracing, monte carlo sampling, global illumination

## 1. 서론

컴퓨터 그래픽스 중 렌더링 과정에서 글로벌 일루미네이션을 표현하기 위해 빛의 차폐 정도를 나타내는 앰비언트 오클루전을 계산한다.

### 1.1 글로벌 일루미네이션(Global illumination)

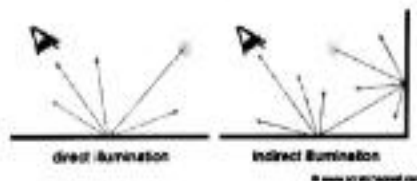
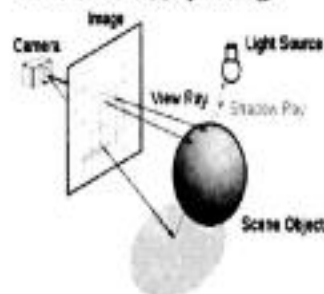


그림 1 글로벌 일루미네이션 (scratchpixel)

글로벌 일루미네이션이란 직접 조명을 제외한 다른 물체에서 반사되는 빛을 말한다. 이를 정확하게 구하기 위해서는 모든 방향에 대해 재귀적으로 추적해야한다.

### 1.2 레이브레이싱(Ray Tracing)



〈그림 2 레이브레이싱 (heretik 2008)〉

레이브레이싱이란 광선이 물체의 표면에 반사되어 카메라로 들어오는 빛을 카메라로부터 시작하여 추적한 뒤 렌더링 하는 방식이다. 글로벌 일루미네이션 계산을 위해 레이브레이싱을 비롯하여 다양한 알고리즘을 사용한다.

# 덱스맵을 통해 계산된 중요도각을 이용한 레이트레이싱 앰비언트 오클루전

## Ray Traced Ambient Occlusion using Importance Angle calculated by Depth Map

석예찬, 김현진

(Yechan Seok and Hyunjin Kim)

**Abstract:** 컴퓨터 그래픽스에서 전역 조명(global illumination) 렌더링을 위해 빛의 차폐 정도를 나타내는 앰비언트 오클루전(ambient occlusion)을 계산한다. 이를 정확하게 계산하기 위해서는 모든 방향으로 재귀적인 레이트레이싱을 수행해야 한다. 최근 이를 실시간으로 돌리기 위해서 픽셀당 1~2개의 레이를 랜덤하게 샘플링한 후에 AI Denoise[1] 시키는 방법이 발표되었다. 하지만 적게 쏠수록 denoise된 이미지가 시점이 바뀔 때 갑자기 색이 바뀌는 flickering 현상이 나타난다. 우리는 이를 해결하기 위해 덱스맵에서 차폐될 수 있는 영역만 판별하여 중요 각도 범위를 계산한 뒤에 모든 방향이 아닌 중요각으로 랜덤 샘플링 하는 방법을 제안한다. 덱스맵은 기본적인 렌더링인 레스터라이제이션 이후 얻을 수 있는 정보이고, 중요각 계산은 커널 사이즈를 정해 픽셀마다 다른 주변 픽셀과의 각도를 병렬적으로 계산한다. 이 알고리즘은 중요도각 계산에서 덱스맵이라는 스크린 공간 정보만 이용하기 때문에 더 부정확하다는 단점이 있지만 결과적으로 flickering현상을 개선하였다. 추가되는 연산량은 레이트레이싱과 분리되어있기 때문에 상수 시간이 더해지는 것뿐이므로 더 효율적이다.

**Abstract:** Ambient occlusion, which indicates the degree of occlusion of light, needs to be calculated for rendering global illumination in computer graphics. Accurate calculation of the ambient occlusion requires recursive ray-racing in all directions, which is time consuming. Recently, it was reported that sampling one or two rays per pixel randomly and conducting AI Denoise[1] could reduce the computation time. This method, however, shows flicking artifacts when the point of view changes, which becomes severe as the number of shooting rays decreases. To solve this problem, we propose a new method. The proposed method determines only the areas that can be occluded from the depth-map, calculates the range of critical angles, and then performs the random sampling only within the critical angles, not in all directions. Depth-map is information that can be obtained after rasterization, which is a basic rendering procedure, and the important angles are calculated in parallel with each pixel by specifying a kernel size. Although this algorithm is more inaccurate because it uses only the screen space information called Depth-map in calculating critical angles, it improves the flickering phenomenon significantly. The additional computations are more efficient because they are separated from Ray Tracing and therefore only add constant time.

**Keywords:** ambient occlusion, ray tracing, monte carlo sampling, global illumination

### 1. 서론

컴퓨터 그래픽스 중 렌더링 과정에서 글로벌 일루미네이션을 표현하기 위해 빛의 차폐 정도를 나타내는 앰비언트 오클루전을 계산한다.

#### 1.1 글로벌 일루미네이션(Global illumination)

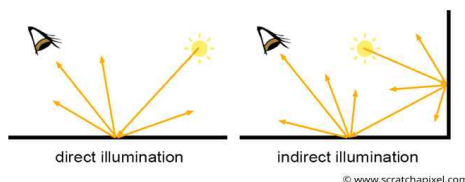
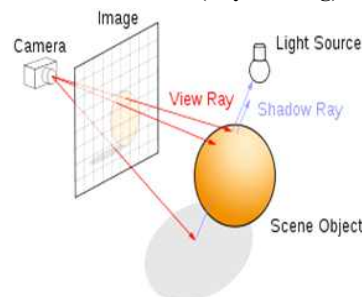


그림 3 글로벌 일루미네이션 (scratchpixel)

글로벌 일루미네이션이란 직접 조명을 제외한 다른 물체에서 반사되는 빛을 말한다. 이를 정확하게 구하기 위해서는 모든 방향에 대해 재귀적으로 추적해야한다.

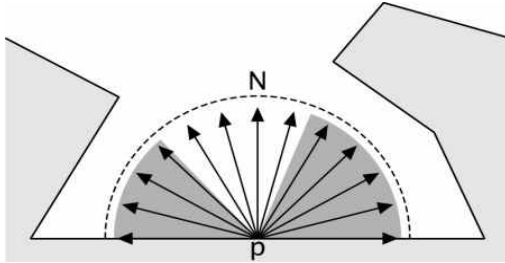
#### 1.2 레이트레이싱(Ray Tracing)



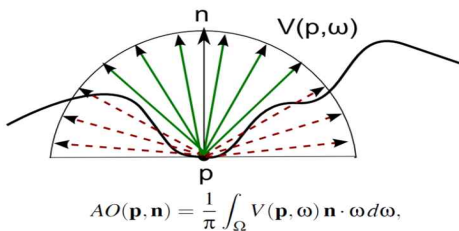
<그림 1 레이 트레이싱 (henrik 2008)>

레이 트레이싱이란 광선이 물체의 표면에 반사되어 카메라로 들어오는 빛을 카메라로부터 시작하여 추적한 뒤 렌더링 하는 방식이다. 글로벌 일루미네이션 계산을 위해 레이트레이싱을 비롯하여 다양한 알고리즘을 사용한다.

### 1.3 앰비언트 오클루전(Ambient occlusion)



<그림 3 앰비언트 오클루전 (Nbertoa 2017)>



<그림 4 앰비언트 오클루전 (sanchez 2011)>

앰비언트 오클루전은 미세표면의 반구를 샘플링 하여 표면에 들어오는 빛이 얼마나 주변 환경에 의해 차폐되어 있는지를 판단하여 글로벌 일루미네이션을 근사한다.

### 1.4 Cosine Weighted Sampling

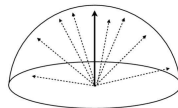
#### Example—Cosine-Weighted Sampling

Cosine-weighted hemisphere sampling in irradiance estimate:

$$f(\omega) = L_i(\omega) \cos \theta \quad p(\omega) = \frac{\cos \theta}{\pi}$$

$$\int_{\Omega} f(\omega) d\omega \approx \frac{1}{N} \sum_i^N \frac{f(\omega_i)}{p(\omega_i)} = \frac{1}{N} \sum_i^N \frac{L_i(\omega_i) \cos \theta_i}{\cos \theta_i / \pi} = \frac{\pi}{N} \sum_i^N L_i(\omega_i)$$

Idea: bias samples toward directions where  $\cos \theta$  is large (if  $L$  is constant, then these are the directions that contribute most)



CMU 15-462/662

<그림 5 Cosine Weighted Sampling (Keenan 2018)>

미세표면의 빛을 계산을 위해 레이트레이싱을 하는데, 각 방향에 대한 빛의 세기는 램버트 코사인 법칙에 의해 노말벡터와 레이의 방향벡터의 코사인 값에 비례한다. 몬테카를로 방법으로 샘플링할 때 랜덤한 방향으로 샘플링한 뒤 코사인값을 적용하는 것 대신에, 코사인 가중 샘플링을 통하여 램버트 코사인 법칙에 따라 계산할 수 있다.

### 1.5 Our Approach

앰비언트 오클루전을 표현하기 위해서 빠르지만 부정확한 방법인 SSAO, HBAO, HBAO+ 같은 스크린공간 연산만을 이용한 방법과 Voxel Cone Tracing을 기반한 VXAO, 느리지만 정확한 계산방법인 레이트레이싱 방법이 있다.

레이트레이싱은 최근 NVIDIA 발표에서 RTX의 레이트레이싱 가속 코어와 AI Denoiser를 이용하여 꽤 빠른 속도로 꽤 좋은 품질의 리얼타임 렌더링을 보여주었다. 그러나 이 역시 리얼타임(for 60 frame)으로 렌더링 하기 위해서는 샘플링 수를 1~2개만 쓸 수밖에 없고 특히 차폐값을 구하기 위해서는 모든 방향으로 몬테카를로 시뮬레이션을 해야한다. 여기서 문제는 모든 방향으로의 적은 샘플링 수로는 부정확한 결과를 초래하기 때문에 denoise 후에 카메라 시점에 따라 한 부분영역의 값이 한꺼번에 갑자기 변화하는 flickering 현상을 피하기 어렵다는 것이다.

우리의 알고리즘은 샘플링수는 그대로지만 차폐될 수 있는 방향인 중요한 각도로만 트레이싱을 하여 노이즈를 개선하여 flickering 현상을 줄이는 방법을 제안한다. 기본적인 렌더링시 주어지는 텍스맵에서 샘플링할 길이인 radius를 정하고 중요도각 샘플 개수와 스텝 사이즈를 정한 뒤 여러 방향( $\phi$ )에 대한 차폐될 수 있는 각도범위( $\theta$ )를 계산한다.

위에서 구한 차폐될 수 있는 영역중 한곳으로 랜덤하게 레이트레이싱을 한 후에 NVIDIA AI Denoiser를 이용하여 Denoise한다. 실험 결과를 통해 flickering 현상의 제거를 보이고 레이트레이싱과 분리된 중요도각 계산비용은 크지않음을 보인다.

## II. 관련연구

실시간으로 앰비언트 오클루전을 구현한 방법들은 여러 가지가 있다.

### 2.1 SSAO[2]



<그림 8 Screen Space Ambient Occlusion>

Screen space ambient occlusion은 기본적인 알고리즘으로 샘플링 할 여러 레이드의 방향을 정해두고 샘플 레이드의 스크린 공간에서 차폐 정도를 검사하여 앰비언트 오클루전값을 결정한다. 스크린 공간 정보만을 이용하는 알고리즘은 보이는 물체 뒤와 스크린 밖은 고려하지 않아 오류가 생긴다. 더 나아가 Polygon Normal과 결합하여 polygon에 대하여 반구로 샘플링 하는 방법을 주로 쓴다.

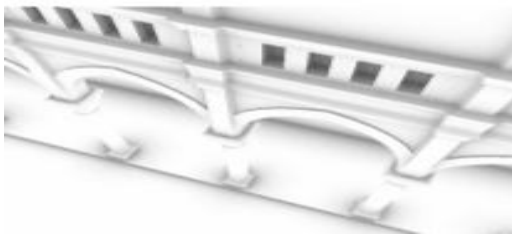
## 2.2 HBAO[3]



<그림 9 Horizon-based Ambient Occlusion>

HBAO는 SSAO 기반의 알고리즘으로 방향( $\phi$ )을 나누고 step을 밟아 나가며 그 방향에 대한 차폐각도( $\theta$ )를 계산해 나가며 각각의 방향에 대해 앰비언트 오클루전값을 계산한다. 이후 blur처리를 하거나, distance를 고려하여 감쇄효과를 적용시켜 구현하는 방식이다. 하지만 이 역시 스크린 공간 정보만 이용하기 때문에 SSAO에서의 오류가 개선되지 않는다.

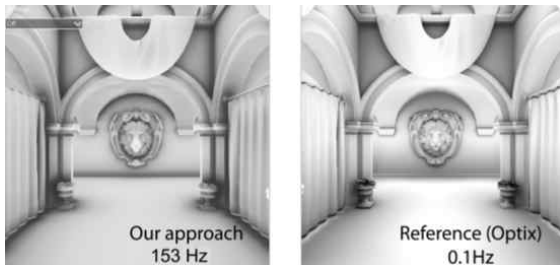
## 2.3 Multi Layered SSAO[4]



<그림 10 Multi-layer dual-resolution SSAO with 3 depth-peeled layer>

Multi Layered SSAO는 다중 depth map으로 계산한 스크린 스페이스 앰비언트 오클루전 방법이다. 특히 게임에서 캐릭터 주위가 어둡게 번지는 것(halo effect)을 방지하기 위해 SSAO를 따로 계산하여 depth오류를 조금이나마 개선하기도 하였다. 여기까지가 스크린 공간을 이용한 알고리즘이다.

## 2.4 VXA0[5]



<그림 11 VXA0 and Ground Truth(Optix)>

VXA0는 VXGI를 이용한 알고리즘이다. VXGI는 polygon데이터로 되어있는 모든 Object들을 geometry shader를 이용하여 복셀화시킨다. 복셀 영역에 포함되는 데이터를 통합하고 이는 LOD Octree로 구성된다. 렌더링 패스에서는 ray를 캐

스팅하며 step에 따라 LOD를 맞게 설정하며 값을 계산해 나간다. 복셀이 있는지 없는지로 차폐는 물론이고, Direct Light 값을 통합하여 reflection, refraction 역시 구할 수 있다. 이는 스크린 공간 알고리즘이 아닌 ray tracing과 가까운 geometric한 알고리즘이기 때문에 SSAO 기법보다 앰비언트 오클루전을 더 잘 표현할 수 있다. 복셀화된 geometry를 통해 레이 캐스팅하기 때문에 검사 속도는 레이트레이싱에 비해 매우 빠르다. 하지만 복셀화할 때의 오류와 속도적 한계 때문에 high resolution을 표현하기는 힘들다는 단점이 있다.

2.5 RayTraced Ambient occlusion[6]은 가장 정확한 방법이지만 Ray를 적어도 256개 이상은 쏘아야 깨끗한 이미지가 나온다. 하지만 레이트레이싱은 하나의 Ray마다 공간에 뿌려져있는 모든 폴리곤과의 충돌여부를 알아야하기 때문에 가속구조인 BVH를 쓰더라도 연산량이 매우 많아 실시간으로 사용하기 부적합하다.

## 2.6 NVIDIA's Approach(2018 GDC.)

최근 NVIDIA의 Raytraced Ambient occlusion은 Ray를 2개만 쏘고 AI Denoiser와 결합하여 Real-Time으로 꽤 괜찮은 품질의 렌더링 이미지를 보여주었다. RTX의 레이트레이싱 전용 core인 RT core와 Deep learning을 위한 tensor core로 하드웨어 가속화 시켰고 AI Denoiser로 노이즈를 개선하였다.

## 2.7 Hybrid Ambient occlusion[7]

GTC에서는 하나의 Ray는 스크린 공간으로 충돌여부를 확인하고, 다른하나의 Ray는 레이트레이싱하여 충돌여부를 결정하여 스크린 공간상의 오류는 있으나 2배에 가까운 속도향상을 보여준 렌더링 방법을 제안하였다.

# III. 알고리즘

## 3.1. 중요도각 구하기

깊이맵과 노말맵은 기존의 래스터라이제이션과 이프라인에서 얻을수있다.

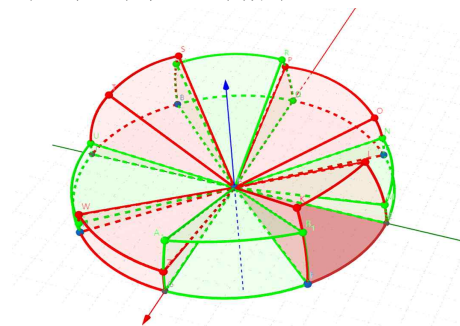


그림 7 중요도각 구하기



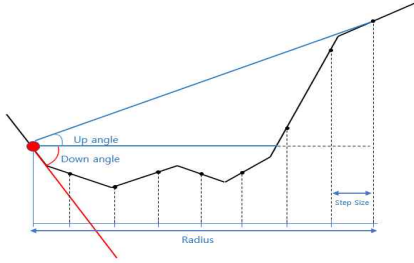


그림 8 중요도각 계산에서 샘플링 방법

### 3.3 중요도각 선택 방법

기존의 코사인가중샘플링 방법을 따르기 위해서는 여러 구간( $\phi_i \sim \phi_{i+1}$ )중에 하나를 선택한 뒤에 중요각( $\theta$ )으로 샘플링 하는데, 구간을 선택할 때 역시 랜덤으로 선택되는 것이 아닌 cosine weight로 선택되어야 한다.

먼저 구간마다 cosine weight를 구한다. 구하는 방법은 그림 7에서 구의 겹넓이 영역들이 원으로 투영된 되었을 때 넓이를 구하면 된다. 이렇게 구한 cosine weight들의 합(1)을 구한 뒤에 random 0~1값을 뽑아 곱한 뒤에 다시 차례대로 빼주면서 0이하가 되는 구간을 선택한다.

## Algorithm Pipeline

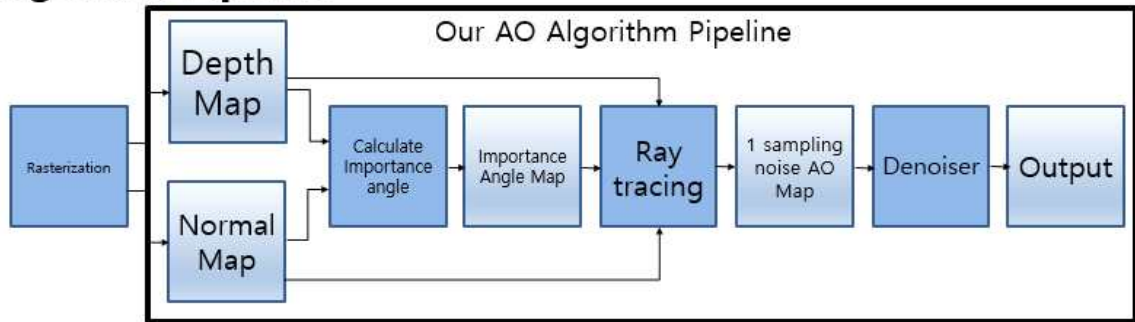


그림 6 본 연구 제안 기법 파이프라인

원을 샘플 갯수로 자르고 각각의 영역( $\phi$ )마다 중요도각( $\theta$ )을 구한다. 이 중요도각은 스크린 공간에서 보고있는 물체들 뒤에 알려지지 않은 영역들을 의미한다. 즉 차폐될 가능성과 그렇지 않을 가능성이 동시에 존재하는 영역이다.

중요도각을 구하기 위한 파라미터는 원의 샘플링 구간을 나누는 샘플 갯수와 각 구간으로 몇 번의 샘플링을 할 지를 결정하는 스텝 횟수와 스텝의 길이를 지정하는 radius가 있다.

계산과정은 샘플 갯수로 나누어진 2차원 방향을 생성한 뒤에 뷰 노말을 구한 뒤 뷰 노말에 대한 반구로 계산한다. 뷰 노말로 회전시킨 방향에 대해 스텝을 밟아가며 텍스처를 검사한다. 텍스처값과 현재 스텝 픽셀(x, y)좌표를 이용하여 뷰 스페이스의 좌표를 구한 뒤에 검사 픽셀의 뷰 스페이스에서의 좌표와 스텝 당 좌표 값을 비교해가며 xy평면과 이루는 각 중 윗부분의 각( $\theta$ )을 구한다. 뷰 노말 때문에 틀어진 각도(xy평면 아랫부분의 각도)를 함께 더해준다. 스텝 사이즈를 밟아가며 가장 큰 각도를 찾는다.

### 3.2 중요도각 버퍼에쓰기

텍스처(버퍼)를 여러 개 쓰기가 부담된다면 output에 중요도각을 쓸 때 RGBA32bit의 경우를 생각해 보면 4개의 값밖에 쓰지 못하는데 그 이상으로 샘플링 하기 위해서는 bit들을 조개어 저장하는 방법이 있다. 일단은 우리의 구현을 쉽게 하기 위해 곱, 나머지 연산을 이용하여 자리 수에 저장하는 방식을 이용했다. 16개의 구간으로 나눌 때 한 개당 8bit를 쓰게 되는데 이는 256가지 수를 표현할 수 있고 각도로 치면 거의 1도 단위로 표현된다. 중요도각의 소수점단위 차이로 인한 오류는 크지 않기 때문에 괜찮은 방법이라 생각한다.

```

[calculate Importance's cosine weighted area]
for i = 1 to sampleCount
    Total Importance cosweighted area += Importance cosw area (i)
    
```

```

[cosine weighted select part]
random area index = Total Importance cosweighted area * random
for i = 1 to sampleCount
    random area index -= Importance cosw area (i)
    if (random area idx < 0) return r0, r1, phi
    
```

샘플 갯수가 그렇게 크지 않기 때문에 알고리즘 자체는 효율적이게 보이지만 추가 저장 공간이 필요없고, 연산비용이 크지 않다.

각 루프에서 90도가 넘어갈 경우 구간이 두개로 나누는 것을 조심해야 한다. 이를 고려하여 r0, r1,  $\phi$ 를 계산해야 한다.

### 3.4 레이트레이싱

우리는 이제 어떤 구간으로 샘플링 하는지와 ( $\phi \sim \phi + (2\pi / \text{sampleCount})$ )  $\theta$  구간을 결정하는 r0, r1를 알고 있다. 이 구간에 대하여 cosine Weighted Hemisphere 샘플링을 한다. 여기서 r0, r1,  $\phi$ 으로 코사인 가중 샘플링된 방향은 뷰 공간 노말의 방향이기 때문에 우리는 이를 월드 공간으로 변환하여야 한다. view행렬의 역행렬을 곱해주어 트레이싱한다.

트레이싱 결과에 최종적으로 Importance에 대한 가중치를 곱해준다.

$$\text{Ambient Occlusion} = \text{IsRayHit} * \left( \frac{\text{Total Importance Surface Area}}{\text{Total Surface Area}} \right)$$

-Importance Sample Count( $\theta$ ) :  $12 \{0,1,2..12\} * \pi * 1/12$   
 -Radius (kernel Size RxR) : 32  
 -Step Size : 6  
 -Ray Tracing Sample count : 1  
 -Ray Tracing Sample Ray Max : 1

기본 알고리즘 : 레이트레이싱 -> Denoiser -> Output  
 본문 알고리즘 : 중요도각계산 -> 중요도각으로 레이트레이싱 -> Denoiser -> Output

#### IV. 결과



그림 9 본 연구 제안 기법

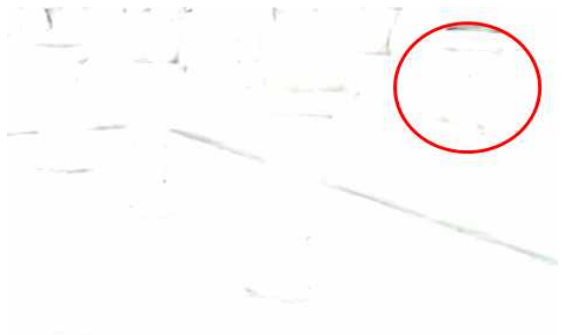


그림 10 기본적인 1sample per pixel + denoiser 알고리즘



그림 11 Ground Truth (RayMax=1)

본 연구는 Intel Core i7 - 7700hq와 GTX1060으로 실험하였다.

이미지 해상도 1280x720

Parameter setting

	Calculate Importance Angle part (per pixel)	Ray tracing part (per ray)	최종 시간 (Denoiser 전)	최종 시간 (+Denoiser)
본문 알고리즘	3.1ms	8ms	11.1ms	134.1
기존 알고리즘	x	6ms	6ms	129ms
차이 (본문 - 기존)	3.1ms	2ms	5.1ms	5.1ms

표4.1 렌더링 소요시간

결과에서 보듯이 기존의 알고리즘은 창문에 드리우는 그림자가 샘플링 수가 적어 어느시점에서는 하얗게 보인다. 하지만 우리의 알고리즘은 Depth정보를 이용하여 뚫려있는 영역은 배제하고 부딪힐수 있는 영역으로만 쏘았기 때문에 샘플링이 잘되어 그림처럼 Flickering 현상을 개선하였다.

#### V. 결론 및 향후 연구

본 연구에서 제안한 기법은 스크린 공간에서의 연산중에 보고있는 물체 뒤를 고려하지 않는 단점을 레이트레이싱으로 보완한 알고리즘이라 할 수 있다. 하지만 중요도각을 Screen Space에서 구하고 있기 때문에 스크린공간의 밖에 있는 물체를 고려하지 않는다. 앞서 보인 실험결과 역시 RayMax를 1 정도로 매우 작게하였고, 중요도각 샘플링시 Radius역시 작게 하였다. 그 이유는 Ray가 커질수록 Screen Space밖의 물체의 영향이 커지고 이로인해 서로간의 오류가 발생하기 때문이다.

현재 Distance에대한 감쇄효과를 고려되지 않았다. 감쇄효과를 추가하면 더욱 그럴듯한 렌더링 결과를 가져다 줄것이라 생각된다. 하지만 RayTracing에서 any-hit연산보다 closed-hit 연산이 더 비싸기 때문에 거리를 알기위해서는 조금 실행속도가 느려질 것이다.

생각보다 Importance영역을 계산하는 과정이 꽤나 소요시간이 크고, selection하는 과정도 무시할 수 없는 속도 차이를 가져다주었다. 본 연구를 생각하게된 계기는 래스터라이제이션 렌더링시에 연

을수있는 정보인 카메라로 찍은 Depth맵에서 카메라쪽으로 오는 각도는 항상 어느정도 차폐되지 않았다는 것을 의미한다고 생각해서 나온 알고리즘이다. 이를 조금 빠르게 계산하고 blur처리를 하거나 샘플링 방식을 개선하면 좋은 결과가 있을 것이다.

## 참고문헌

- [1] Chaitanya, C. R. A., Kaplanyan, A. S., Schied, C., Salvi, M., Lefohn, A., Nowrouzezahrai, D., & Aila, T. (2017). Interactive reconstruction of Monte Carlo image sequences using a recurrent denoising autoencoder. ACM Transactions on Graphics, 36(4), 1–12 doi:10.1145/3072959.3073601
  - [2] Bavoil, Louis, and Miguel Sainz. "Screen space ambient occlusion." NVIDIA developer information: <http://developers.nvidia.com> 6 (2008).
  - [3] Bavoil, L., Sainz, M., & Dimitrov, R. (2008). Image-space horizon-based ambient occlusion. ACM SIGGRAPH 2008 Talks on - SIGGRAPH '08.
  - [4] Bavoil, L., & Sainz, M. (2009). Multi-layer dual-resolution screen-space ambient occlusion. SIGGRAPH 2009: Talks on - SIGGRAPH '09.
  - [5] Crassin, C., Neyret, F., Sainz, M., Green, S., & Eisemann, E. (2011). Interactive Indirect Illumination Using Voxel Cone Tracing. Computer Graphics Forum,
  - [6] Graham Wihlidal, Colin Barre-Brisebois. SEED - Electronic Arts (2018). Modern Graphics Abstractions & Real-Time Ray Tracing. Syssgraph 2018(Siggraph helsinki).
  - [7] Reinbothe, Christoph K., Tamy Boubekeur, and Marc Alexa. "Hybrid Ambient Occlusion." Eurographics (Areas Papers). 2009.
- [그림1] "global-illumination-path-tracing", 스크래치픽셀, 2018년 12월 3일 접속, <http://www.scratchapixel.com/lessons/3d-basic-rendering/global-illumination-path-tracing>
- [그림2] "Ray-tracing", 위키피디아, 2018년 11월 15일 수정됨, 2018년 11월 18일 접속, [https://en.wikipedia.org/wiki/Ray\\_tracing\\_\(graphics\)](https://en.wikipedia.org/wiki/Ray_tracing_(graphics))
- [그림3] "Ambient Occlusion", nbertoa, 2017년 02월 04일 수정, 2018년 11월 18일 접속, <https://nbert>

[oa.wordpress.com/2017/02/04/directx-12-ambient-occlusion-the-second-approach/](http://oa.wordpress.com/2017/02/04/directx-12-ambient-occlusion-the-second-approach/)

[그림4] "Ambient Occlusion", Santi Sanchez, 2011년 7월 16일 수정, 2018년 11월 18일 접속, <https://santisanchez28.wordpress.com/2011/07/16/ambient-occlusion-2/>

[그림5] "Monte Carlo Cosine Weighted Sampling", Carnegie Mellon University, 2018년 11월 20일 접속, <http://15462.courses.cs.cmu.edu/spring2018/lecture/monteCarlo> slide-031

[그림6] "Screen Space Ambient Occlusion", Eletroni cmeteor, 2013년 12월 20일 수정됨, 2018년 11월 18일 접속, <https://www.derschmale.com/2013/12/20/an-alternative-implementation-for-hbao-2/>

[그림7] "Horizon-Based Ambient Occlusion", Eletroni cmeteor, 2013년 12월 20일 수정됨, 2018년 11월 18일 접속, <https://www.derschmale.com/2013/12/20/an-alternative-implementation-for-hbao-2/>

[그림8] Bavoil, Louis, and Miguel Sainz. "Multi-layer dual-resolution screen-space ambient occlusion." SIGGRAPH 2009: Talks. ACM, 2009.

[그림9] Crassin, Cyril, et al. "Interactive indirect illumination using voxel cone tracing." Computer Graphics Forum. Vol. 30. No. 7. Oxford, UK: Blackwell Publishing Ltd, 2011.



김 현 진

2012년 ~ 현재 인하대학교 정보통신공학과 학사 과정 재학 중.

2019년 2월 졸업 예정.

관심분야 Real-time Rendering, Computer Graphics



석 예 찬

2014년 ~ 현재 인하대학교 정보통신공학과 학사 과정 재학 중.

2019년 2월 졸업 예정.

관심분야 Real-time Rendering, Computer Graphics