Journal of Korea Game Society 2013 Jun; 13(3): 85-94
http://dx.doi.org/10.7583/JKGS.2013.13.3.85

컴퓨터 그래픽스

실시간 광선추적기를 위한 바운딩 박스 기반의 그림자 검사 컬링 기법

김상덕*, 김진우*, 박우찬**, 한탁돈*
연세대학교 컴퓨터과학과*, 세종대학교 컴퓨터공학과**
{sdk, jwkim}@msl.yonsei.ac.kr, pwchan@sejong.ac.kr, hantack@msl.yonsei.ac.kr

Bounding Box based Shadow Ray Culling Method for Real-Time Ray Tracer

Sangduk Kim*, Jin-Woo Kim*, Woo-Chan Park**, Tack-Don Han*
Dept. of Computer Science, Yonsei University*
Dept. of Computer Engineering, Sejong University**

요 약

본 논문에서는 광선 추적기법에서 그림자 검사의 연산량을 줄이는 기법을 제안한다. 그림자 검사는 부드러운 그림자와 같이 실감 있는 영상을 생성하는 경우 광선 추적기법 렌더링 과정 중 매우 큰 비중을 차지한다. 제안하는 방식은 전처리 단계에서 kd-tree를 생성하며 계산한 그림자 정보를 기반으로 렌더링 단계에서 그림자 검사의 컬링 여부를 판별하는 방식으로 기존 렌더링 과정에 작은 변경으로 적용가능하다. 기존의 방식들과 유사하게 광원과 기하학적 구조가 변하지 않는 정적 장면에 적합하다. 사이클 정확도를 갖는 시뮬레이터를 이용하여 제안하는 방법의 유효성을 검증하고 성능을 측정하였으며 실험 결과 제안하는 그림자 컬링 기법으로 최대 17%까지 줄일 수 있다.

ABSTRACT

In this paper, we propose a scheme to reduce the number of shadow tests conducted during rendering of ray tracing. The shadow test is a very important process in ray tracing to generate photo-realistic images. In the rendering phase, the ray tracer determines whether to cull the shadow test based on information calculated from a shadow test conducted on the kd-tree in the preprocessing phase. In conventional rendering process, the proposed method can be used with little modification. The proposed method is suitable for a static scene, in which the geometry and light source does not change in the same manner as it does in the conventional method. The validity of the proposed scheme is verified and its performance is evaluated during cycle-accurate simulation. Through experiment results, we found that we could reduce up to 17% of the shadow test.

Keywords : 3D graphics(3차원 그래픽스), ray tracing(광선 추적), shadow test(그림자 검사)

Received: Mar. 24, 2013 Accepted: Jun. 07, 2013
Corresponding Author: Tack-Don Han(Yonsei University)
E-mail: hantack@msl.yonsei.ac.kr

ISSN: 1598-4540 / eISSN: 2287-8211

1. 서 론

현재 대다수의 3차원 그래픽 렌더링 프로세서는 지역 조명(local illumination)을 바탕으로 하는 래스터라이제이션(rasterization) 알고리즘을 사용한다. 사실적 영상을 생성하기 위해서는 전역 조명(global illumination) 문제의 해결이 필요하지만 래스터라이제이션 기반 알고리즘은 근본적인 해결이 불가능하기 때문에 모사기법(trick)을 사용한다. 이 과정에서 콘텐츠 제작의 생산 효율이 떨어지고 여러 제약이 발생하게 된다[1]. 반면 광선 추적 기법은 물체간의 상호 영향과 광원 및 시점에서 출발한 빛의 이동 경로와 속성을 정밀하게 모델링하기 때문에 반사(reflection), 굴절(refraction), 확산(scattering, diffuse reflection) 등과 같은 사실적 그래픽 효과를 재현하는 것이 가능하다. 따라서 광선추적기법을 지원하는 GPU가 시장을 주도할 것으로 예상하고 있다.

광선추적기법의 대표적인 문제점은 대규모의 연산량이 필요한 것이다. 이러한 연산량 문제를 해결하기 위한 전용하드웨어 연구[2]가 진행되어 왔으며 최근에는 GPU를 기반으로 한 가속 기법 연구[3,4]도 진행되고 있다. 하지만 실시간 처리에는 성능이 여전히 부족하다.

광선추적은 빛의 이동 경로를 시점(view point)에서부터 물체를 거쳐 광원(light source)까지 추적하여 이미지를 생성하는 알고리즘이다. 광선이 이동하는 경로 중, 3차원 공간상에 존재하는 물체들과 교차상태를 조사하여 각 픽셀(pixel)의 색상을 결정한다. 이를 위하여 먼저 각 픽셀에 대하여 시점을 원점으로 하는 1차 광선(primary ray)을 생성하고 이 광선과 교차하는 물체를 찾는다. 이 물체가 반사도 또는 굴절도가 있으면 물체의 교차점(hit point)에서 2차 광선(secondary ray)을 생성하고 이 과정을 재귀적(recursive)으로 처리한다.

이러한 1차 광선과 2차 광선은 물체와의 교차점(hit point)에서 각 광원 방향으로 그림자 광선(shadow ray)을 생성한다. 이 그림자 광선이 불투명한 물체와 만나는지 여부를 판별하는 것이 그림자 검사이다. 그림자 광선은 광원과 비례하여 그 수가 증가하며, 사실적 영상을 생성하기 위해서 엄청난 양의 그림자 검사가 필요하다.

본 연구에서는 전처리 과정에서 kd-tree의 축에 정렬된 바운딩 박스(axis-aligned bounding box)에 대하여 그림자 검사를 미리 수행하여 이 결과를 kd-tree에 저장함으로써 그림자 검사의 연산량을 줄이는 방식을 제안한다. 이는 전처리 단계에서 각 광원에 대하여 해당 리프 노드(leaf node)가 그림자 검사를 수행해야 하는지의 여부를 판별하여 이를 해당 kd-tree의 리프 노드에 포함된 LVB(light visibility bit)에 저장한다. 렌더링(rendering) 단계에서는 1차 또는 2차 광선이 교차된 kd-tree의 리프 노드에 저장된 LBV의 값에 따라 그림자 검사의 컬링(culling) 여부를 판별한다.

제안하는 방식의 성능은 싸이클 정확도를 갖는 시뮬레이터를 이용하여 예측하였으며 실험 결과 제안하는 그림자 컬링 기법으로 그림자 광선의 연산 시간을 최대 17%까지 줄일 수 있다.

2. 관련 연구

광선추적기법에서 그림자 처리 연산을 감소하기 위한 연구들은 크게 두 가지로 분류할 수 있다. 첫 번째는 전처리를 기반으로 연산을 감소시키는 방법으로 정적인 장면의 렌더링에 적용이 가능한 연구이고 두 번째는 그림자 처리 결과를 캐싱(caching)하는 방법으로 동적인 장면의 렌더링에 적용이 가능한 연구이다.

2.1 정적인 장면에서 그림자 검사 컬링

정적인 장면에서의 그림자 검사 컬링(shadow test culling) 방법은 전처리 단계에서 그림자에 대한 정보를 계산하여 이를 저장하고 렌더링 단계에서 이를 이용하기 때문에 동적인 장면이면서 조명도 고정된 경우에만 유효하다. 이에 대하여 다음의

2가지 대표적인 연구가 진행되었다.

첫째, 라이트 버퍼(light buffer)[5]는 광선 추적 기법 상의 그림자 처리를 위한 초기 연구로 그리드(grid) 형태로 라이트 버퍼를 구성하고 모든 방향에 있는 물체들을 거리 기준으로 정렬하여 방향 셀(directional cell)에 저장하는 방식이다. 생성된 라이트 버퍼를 이용하면 셀 안에 저장된 프리미티브에 대해서만 교차 검사를 수행하며 거리기준으로 정렬되어있기 때문에 그림자 검사는 빠르게 결정할 수 있다. 그러나 이 알고리즘의 효율은 셀 크기에 반비례하기 때문에 작은 셀을 가지는 그리드 구조 생성을 위해 큰 메모리 용량과 방향셀을 생성하기 위한 스캔 컨버전(scan conversion) 및 정렬의 많은 연산을 필요로 하는 단점이 있다.

둘째, 복셀 차폐 검사(voxel occlusion test)[6]는 라이트 버퍼와 달리 전처리 결과를 광원이 아닌 물체의 가속 구조에 저장하는 방식으로 차폐물체(Occluder)의 그림자 암영부(shadow umbra)를 미리 계산하여 full, null, complicated occlusion 세 가지 모드로 복셀(voxel)을 분류한다.

full 또는 null로 구분된 복셀은 그림자 검사를 필요로 하지 않으며 complicated occlusion으로 분류된 복셀에 대해서만 수행되므로 그림자 검사는 빠르게 결정할 수 있다. 그러나 방식은 가속 구조를 성능이 떨어지는 정규 그리드(uniform grid)로만 한정시켜 성능이 높은 계층적 트리 기반의 가속 구조인 kd-tree나 BVH(bounding volume hierarchy)에는 적용이 불가능한 단점이 있다.

2.2 동적인 장면에서 그림자 검사 컬링

동적인 장면에서의 그림자 검사 컬링(shadow test culling) 방법은 그림자 처리 결과의 일관성(coherence)을 이용한다. 물체 간의 일관성은 어떤 물체가 어떠한 광선에 대해 다른 물체에 의해 가려져 있는 것으로 판명이 되었다면 인접한 광선에서도 계속 가려질 가능성이 높다는 것을 의미한다. 이에 대하여 다음의 2가지 대표적인 연구가 진행되었다.

첫째, 그림자 캐싱(shadow caching)은 라이트 버퍼와 같은 논문[5]에서 제시된 알고리즘이다. 그림자 캐싱은 그림자를 발생시킨 차폐물체(occluder)를 캐시에 저장한다. 그리고 다음 번 그림자 광선 추적시, 일반적인 교차 검사 전에 캐시에 저장된 차폐물체(occluder)를 검사한다. 만약 광선이 이 물체와 교차한다면 그림자 검사는 종료되고 그림자가 생기는 것으로 결정된다. 이 알고리즘은 간단하고 그림자 검사에도 효과적이다. 하지만 일관적인 그림자 광선과 큰 프리미티브로 구성된 경우에만 적합하며, 가려지지 않는 그림자 광선에는 적용할 수 없는 문제점이 있다[7].

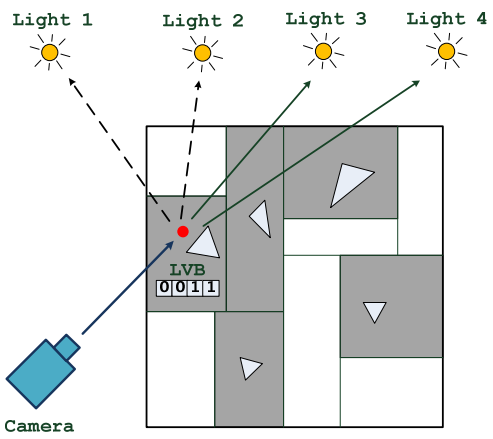
둘째, 차폐/그림자 모드 기반 컬링 기법[8]은 동적인 장면의 렌더링에서 프레임 간 일관성을 이용하는 방식으로 렌더링 과정에서 교차된 모든 프리미티브의 차폐/그림자 정보를 갱신하고 이를 그림자 검사 컬링에 이용한다. 차폐/그림자 정보는 전체, 부분, 초기화 모드로 구분되며 이는 투영된 프리미티브의 크기와 렌더링된 픽셀수의 관계를 이용하여 설정된다. 이 알고리즘은 프레임 내 일관성만을 이용하는 그림자 캐싱의 문제점인 비일관적인 그림자 광선과 작은 프리미티브로 구성된 장면에 대해 적용할 수 있다. 하지만 모든 프리미티브에 대한 정보 갱신은 일반적으로 읽기 연산만을 수행하는 렌더링 과정에서 비효율적이며 특히 실시간 처리를 위한 전용 하드웨어에는 적용이 어려운 문제점이 있다.

3. 제안하는 그림자 컬링 방법

제안하는 방식은 전처리 과정에서 축에 정렬된(axis-aligned) kd-tree의 리프 노드에 대하여 그림자 광선의 발생 여부인 LVB를 결정하여 이를 리프 노드에 저장하는 과정과, 이 결과를 렌더링 과정에서 사용하여 그림자 검사의 필요 여부를 결정하는 과정으로 나누어진다. 이는 2.1절의 라이트 버퍼(light buffer)[5]와 복셀 차폐 검사(voxel

occlusion test)[6]와 유사하게 장면의 기하학적 구조와 광원의 위치가 변하지 않는 경우 유효하며, 시점이 변화에는 독립적이다. 만약 동적인 장면인 경우 kd-tree를 재구성해야 하므로 그에 대한 LVB를 다시 계산해야 한다.

[Fig. 1]은 제안하는 방식을 2차원으로 단순화하여 나타낸 예제이다. 먼저 사각형들은 kd-tree를 구성하는 바운딩 박스(bounding box)이며, 회색 사각형들만이 물체를 포함하고 있다. 바운딩 박스 B_A와 광원 1과 2의 사이에는 물체를 포함하는 바운딩 박스가 존재하지 않으므로, B_A에 포함되어 있는 물체는 광원 1과 2로 생성되는 그림자가 존재할 수 없으므로 해당 광원에 대한 LVB는 0으로 결정된다. 반면 B_A와 광원 3과 4사이에는 물체를 포함하는 바운딩 박스가 위치하므로 B_A에 포함되어 있는 물체들은 그림자가 생성될 가능성이 있으므로 렌더링 시에 그림자 검사를 수행해야 한다. 그러므로 해당 광원에 대한 LVB는 1로 결정된다.

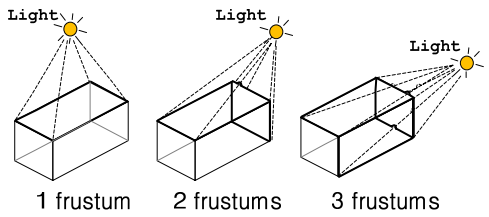


[Fig. 1] An example of a LVB calculation in 2D space

LVB 계산 단계는 임의의 kd-tree 리프 노드의 바운딩 박스와 광원 사이에 형성되는 절두체(frustum) 내에 다른 리프 노드의 바운딩 박스가 존재하는지 여부를 계산하는 과정이고, 이는 크게 3단계로 이루어진다. 첫 번째는 광원과 kd-tree 리프 노드의 바운딩 박스의 위치 관계를 이용하여 절두체(frustum)의 형태를 결정하는 단계이다. 두 번째는 절두체(frustum)를 구성하는 모든 평면의 방정식을 구하는 단계이다. 세 번째는 앞서 계산된 광원과 리프 노드의 바운딩 박스로 이루어지는 절두체(frustum)와 다른 모든 리프 노드의 바운딩 박스의 포함 관계를 조사하여 최종적으로 LVB를 결정하는 단계이다. 각 단계에 대한 설명은 다음에 이어진다.

3.1 절두체의 형태 결정 단계

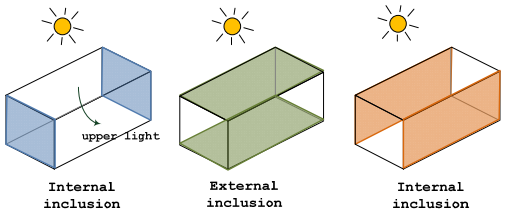
본 과정은 물체를 포함하고 있는 kd-tree 리프 노드의 바운딩 박스와 광원의 위치정보를 이용하여 이들로 이루어진 절두체(frustum)를 생성하는 과정이다. 각각의 위치 관계에 따라 [Fig. 2]의 3가지 경우의 절두체(frustum)가 생성된다.



[Fig. 2] Three cases of frustum

[Fig. 2]의 3가지 절두체(frustum) 형태는 광원과 리프 노드의 바운딩 박스에서 생성되는 3개의 슬랩(slab)간의 위치 관계에 의해 결정되며 [Fig. 3]는 1개의 절두체(frustum)로 결정되는 예시를 보여준다.

절두체(frustum) 형태 결정을 위해 광원에 대해 총 3번의 슬랩 포함 검사(slab inclusion test)가 요구된다. [Fig. 3]의 예제에서 첫 번째와 세 번째의 검사에서 광원은 슬랩(slab)의 내부에 있다고 결정되고, 두 번째 검사에서는 광원이 슬랩(slab)의 외부에 있다고 결정된다. 따라서 광원은 바운딩 박스의 바로 위에 존재하며 1개의 절두체(frustum)가 생성되는 것을 알 수 있다.



[Fig. 3] Slab inclusion test for 1 frustum case.

[Fig. 3]의 예제와 달리 첫 번째와 두 번째 슬랩(slab)에 광원이 외부에 있고, 세 번째 슬랩(slab)에 광원이 내부에 있다고 결정되는 경우 2개의 절두체(frustum)로 생성되는 것으로 결정된다. [Table 1]은 슬랩 포함 검사(Slab inclusion test)결과로 절두체(frustum)의 형태를 구분한 표이다. 광원을 포함한 슬랩(light included slab)의 개수가 3개인 경우 광원이 바운딩 박스 내부에 포함된 내부 광원(internal light)을 의미한다.

[Table 1] Determining type of frustum by Slab inclusion test

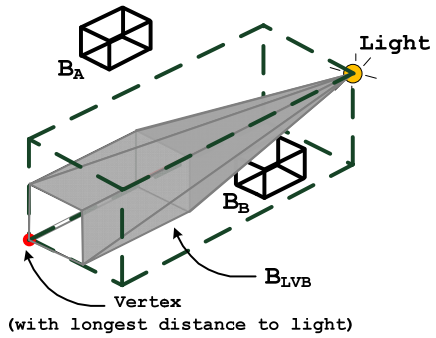
Number of light included slabs	Number of frustum
3	0 (Internal Light)
2	1
1	2
0	3

3.2 평면 방정식 계산 단계

본 과정은 절두체(frustum)를 구성하고 있는 모든 평면에 대한 평면의 방정식을 생성한다. 평면의 방정식은 총 4개의 계수(coefficient)가 필요하며, [Fig. 2]의 1개의 절두체(frustum)의 경우 4개의 평면이 존재하므로, 총 16개의 계수(coefficient)를 계산이 필요하다.

3.3 절두체 포함 검사를 통한 LVB 결정 단계

본 과정은 물체를 포함하고 있는 각각의 리프 노드의 바운딩 박스와 각각의 광원들로 생성되는 절두체(frustum)들 각각에 대해 다른 모든 리프 노드들의 바운딩 박스와 포함 관계가 있는지 검사를 수행한다. 이 과정은 모든 리프 노드를 모든 리프 노드에 대해 검사를 수행해야 하므로 $O(n^2)$의 복잡도를 갖는다. 본 연구에서는 연산을 줄이기 위해 [Fig. 4]와 같이 검사를 수행할 필요가 없는 리프 노드의 바운딩 박스를 선별하는 과정이 선행된다.



[Fig. 4] Finding bounding box candidate for inclusion test

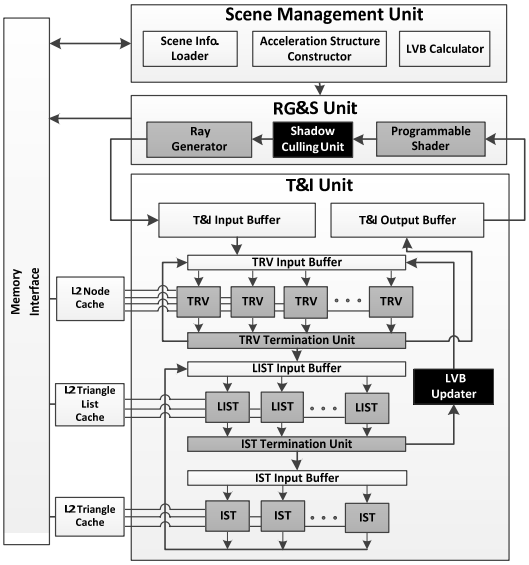
이 과정은 LVB를 결정하기 위한 리프 노드의 바운딩 박스 B_{LVB}와 광원을 포함하는 가장 적은 점선의 바운딩 박스를 생성하는데, 이때에는 바운딩 박스 B_{LVB}의 정점(vertex) 중 광원과 가장 먼 거리를 가지는 정점(vertex)을 이용한다. 점선의 바운딩 박스 생성 후, 절두체 포함 검사(frustum inclusion test)의 목표가 되는 다른 모든 리프 노드의 바운딩 박스들이 점선의 바운딩 박스에 포함되거나 걸쳐지는지 여부를 3차원 축을 기준으로 한 SAT(Separating Axis Theorem)[9]를 이용하여 조사한다. 이 검사 결과 포함되거나 걸치는 리프 노드의 바운딩 박스는 절두체 포함 검사(frustum inclusion test)를 수행하며 그렇지 않은 바운딩 박스는 절두체(frustum)의 외부에 있는 것이므로 절두체 포함 검사(frustum inclusion test)를 수행하지 않는다.

최종적으로 절두체 포함 검사(frustum inclusion test) 단계로 전달된 리프 노드의 바운딩 박스가 절두체(frustum)에 포함되어 있다고 결정되면 바

운딩 박스 B_{LVB}는 해당 바운딩 박스에 의해 광원이 가려질 확률이 있으므로 LVB를 1로 세팅한다.

3.4 LVB기법의 하드웨어 적용

제안하는 그림자 컬링 기법을 적용은 현재 처리 중인 광선의 LVB값을 유지하고 최종적으로 쉐이딩 단계에서 LVB값을 기반으로 그림자 광선을 생성할지 여부를 결정하게 된다. 이 때 광선의 LVB는 교차된 물체들 중 시점과 가장 가까운 물체를 포함하고 있는 리프노드의 LVB값으로 물체-광선 간 교차검사 이후 교차결과가 갱신될 때 함께 갱신이 되어야 한다.



[Fig. 5] The architecture of ray tracing hardware with proposed scheme

제안하는 방식을 적용하기 위해 LVB 값을 저장하는 연산과 LVB 값을 비트단위로 비교하는 연산이 추가되어야 하며 이는 전체 하드웨어 구조에 큰 영향을 주지 않으며 하드웨어 비용도 적다.

[Fig. 5]는 LVB 기반 그림자 컬링 기법을 렌더링 하드웨어에 적용한 것으로 LVB 갱신 유닛(LVB Updater 부분)과 LVB 기반 그림자 컬링 유닛(Shadow Culling Unit)이 적용되어있다.

4. 실험 및 결과

4.1 실험 환경

제안하는 방식의 성능 예측을 위해 사이클 정확도를 갖는 시뮬레이터에서 실험하였다. 실험에 사용한 사이클 정확도를 갖는 시뮬레이터는 광선누적 버퍼(ray accumulation buffer)가 적용된 T&I 엔진의 시뮬레이터[2]를 기반으로 구현하였다.

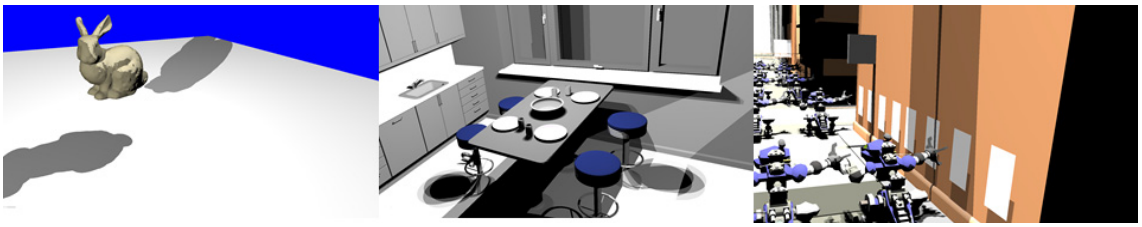
정확한 메모리 연산 시간의 시뮬레이션을 위해 GPGPU-Sim[10]의 GDDR3 시뮬레이션 코드를 이용하여 동작클락을 1GHz로 설정하였다.

각각의 유닛의 파이프라인의 수행 시간은 TRV를 8사이클, LIST를 2싸이클, IST를 36사이클로 설정하였다. 각각의 유닛들은 1단계(L1) 및 2단계(L2) 캐시를 포함하고 있으며 [Table 2]는 각 유닛의 세부 설정내용이다.

[Table 2] Cache configuration of each unit

Unit	Level	Size/Type
TRV 24 units	L1 Cache	8KB 2-way
	L2 Cache	128KB 2-way
LIST 12 units	L1 Cache	4KB 2-way
	L2 Cache	32KB 2-way
IST 12 units	L1 Cache	16KB 4-way
	L2 Cache	128KB 2-way

실험의 벤치마크로는 [Fig. 6]의 Stanford bunny, BART Kitchen/Robot[11]의 장면을 이용하였다. [Table 3]은 실험에 사용된 장면들의 세부정보이다.



[Fig. 6] Three test scenes: Stanford Bunny, Kitchen and Robot

[Table 3] The information of test scenes

Scene	Triangle	light	kd-tree	
			total	leaf
Bunny	10125	2	7234	3617
Kitchen	102142	5	61320	30660
Robot	71581	6	69018	34509

4.2 실험 결과

[Table 4]는 제안하는 방식을 적용하여 LVB를 계산하였을 때 컬링되는 리프노드의 비율이며 [Table 5]는 렌더링 과정에서 컬링되는 그림자 광선의 비율을 정리한 표이다.

[Table 4] The experimental results of leaf-node culling using proposed scheme.

Scene	Light	Culled leaf-node	Total leaf-node	Culling rate
Bunny	1	0	3617	0.0%
	2	43		1.2%
Kitchen	1	2441	30660	8.0%
	2	3498		11.4%
	3	1105		3.6%
	4	1082		3.5%
	5	31		0.1%
Robot	1	20173	34509	58.5%
	2	20176		58.5%
	3	20412		59.1%
	4	20421		59.2%
	5	1745		5.1%
	6	27202		78.8%

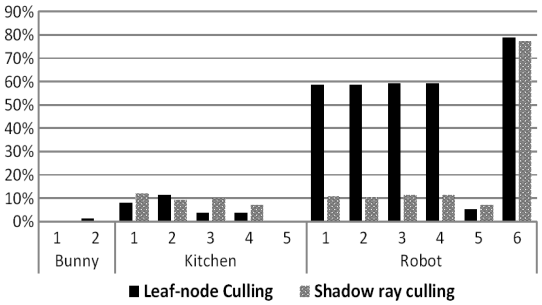
Stanford bunny의 경우 컬링이 거의 되지 않았지만 BART robot의 경우 조명에 따라 최대 78%까지 컬링이 되었다. BART Kitchen이 BART Robot 장면 보다 낮은 컬링 비율은 보이는 것은 영상을 구성하고 있는 물체의 특징이 서로 상이하기 때문이다. BART Kitchen의 경우 영상에서 차지하는 물체의 크기가 크고, BART Robot의 경우는 그 반대이다. 물체의 크기가 작은 경우 물체를 포함하는 kd-tree의 리프노드의 바운딩 박스의 크기가 상대적으로 작게 구성이 되기 때문에 BART Robot의 더 높은 컬링 효과를 보인다.

[Table 5] The experimental results of shadow ray culling using proposed scheme.

Scene	Light	Culled leaf-node	Total leaf-node	Culling rate
Bunny	1	0	1645550	0.00%
	2	160		0.01%
Kitchen	1	249864	2073599	12.05%
	2	193107		9.31%
	3	207110		9.99%
	4	147122		7.10%
	5	880		0.04%
Robot	1	220764	2073599	10.65%
	2	217152		10.47%
	3	236312		11.39%
	4	236285		11.40%
	5	148670		7.17%
	6	1604685		77.39%

[Fig. 7]는 컬링된 리프노드와 광선을 정리한 그래프이다. BART Kitchen은 큰 차이가 없지만 BART Robot 장면은 시점의 특성상 공간의 일부만 탐색되었기 때문에 리프 노드와 광선 컬링의 결과에 큰 차이를 보인다.

[Table 6]은 제안하는 방식을 적용한 사이클 정확도를 갖는 시뮬레이터를 이용하여 각각의 벤치마크를 실험한 결과로 1차 광선과 그림자 광선을 모두 포함한 '전체' 결과와 그림자 광선만을 포함한 '그림자' 결과로 구분하였다.



[Fig. 7] Culling rate(%) for test scened accoring to number of lights

[Table 6]의 결과에서 가장 큰 렌더링 시간 감소를 보인 것은 그림자 광선이 가장 많이 컬링이 된 BART Robot 장면으로 최대 17%의 그림자 처리 시간을 감소시켰다.

[Table 6] Simulation results.

Scene	Ray	Proposed scheme		Reduction (%)
		On	Off	
Bunny	all	13.75ms	13.76ms	0.02
	shadow	7.75ms	7.96ms	0.03
Kitchen	all	83.55ms	87.17ms	4.16
	shadow	57.65ms	61.27ms	5.91
Robot	all	102.95ms	118.30ms	12.97
	shadow	70.75ms	86.10ms	17.83

5. 결론 및 향후 계획

렌더링 과정에서 추가 연산이 필요한 대부분의 이전 연구와는 다르게 본 논문에서 제안하는 방법은 전처리 과정에서 대부분의 컬링이 결정되어 전체 렌더링 시간에 미치는 영향이 거의 없다. 또한 렌더링 과정에서 참조되어야 하는 LVB값은 각 리프 노드 당 조명의 개수만큼 한 비트의 저장공간이 필요하며 이러한 LVB값은 광선 추적에서 일반적으로 사용하는 가속구조체에 저장하는 방식이므로 기존의 렌더링 방법의 변경 없이 적용이 가능한 장점이 있다.

향후 연구에서 LVB값을 계산하는 전처리과정의 연산을 공간의 재귀적 탐색으로 최적화하는 것에 주안점을 두고 있다. 이러한 최적화를 통해 정적장면으로 제한된 그림자 컬링 기법을 동적 장면에도 적용이 기능하도록 할 것이다.

REFERENCES

[1] J. Hurley, "Ray tracing goes mainstream," Intel Technology Journal, Vol. 9, No 2, pp 98-107, 2005.

[2] Jae-Ho Nah, Jeong-Soo Park, Chanmin Park, Jin-Woo Kim, Yun-Hye Jung, Woo-Chan Park and Tack-Don Han, "T&I engine: traversal and intersection engine for hardware accelerated ray tracing," ACM Transactions on Graphics (TOG), Vol.30, No.6, 2011.

[3] Timo Aila and Samuli Laine, "Understanding the efficiency of ray traversal on gpus," In Proc. of High Performance Graphics, 2009.

[4] Sung-Min Bae and Hyun-Ki Hong, "Implementation of Real-time Interactive Ray Tracing on GPU," Journal of Korea Game Society, Vol. 7, No. 3, pp59-66, 2007.

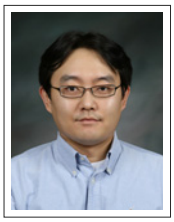
[5] E. Haines and D. Greenberg, "The Light Buffer: A Shadow-Testing Accelerator," IEEE CGGA, Vol. 6, No. 9, pp6-16, 1986.

[6] A. Woo and J. Amanatides, "Voxel Occlusion Testing: A Shadow Determination Accelerator for Ray Tracing," Graphics Interface 90, pp 213-220, 1990.

[7] Ingo Wald, "Realtime Ray Tracing and Interactive Global Illumination," Ph.d thesis, Sarrland University, 2004.

[8] Jae-Ho Nah, Woo-Chan Park and Tack-Don Han, "A Shadow Culling Algorithm for Interactive Ray Tracing," Korea Game

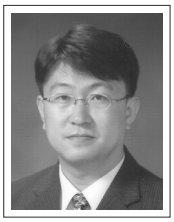
Society, Vol. 9, No. 6, pp179-189, 2009.
[9] S. Gottschalk, “Separating axis theorem,” Technical Report TR96-024, Department of Computer Science, UNC Chapel Hill, 1996.
[10] A. Bakhoda, G. Yuan,W. Fung, H.Wong, and T. Aamodt, “Analyzing CUDA workloads using a detailed GPU simulator,” in Proc. ISPASS, pp163-174, 2009.
[11] J. Lext, U. Assarsson and T. Moller, “A Benchmark for Animated Ray Tracing,” IEEE Computer Graphics and Applications, Vol. 21, No. 2, pp22-31, Mar. 2001.



김 상 덕(Sangduk Kim)

2001년 연세대학교 수학과 학사
2003년 연세대학교 컴퓨터과학과 석사
2003년-현재 연세대학교 컴퓨터과학과 박사과정

관심분야 : 컴퓨터 그래픽스, 광선 추적 알고리즘



박 우 찬(Woo-Chan Park)

1993년 연세대학교 전산과학과 학사
1995년 연세대학교 컴퓨터과학과 석사
2000년 연세대학교 컴퓨터과학과 박사
2001-2003년 연세대학교 연구교수
2003-2008년 세종대학교 컴퓨터공학과 조교수
2008-현재 세종대학교 컴퓨터공학과 부교수

관심분야 : GPU 구조, 실시간 Ray Tracing 가속기, 컴퓨터 구조, 실시간 렌더링, ASIC 설계

김 진 우(Kim, Jinwoo)

2006년 상명대학교 소프트웨어전공 학사
2007-현재 연세대학교 컴퓨터과학과 석박통합과정

관심분야 : 컴퓨터 그래픽스 SW/HW, GPGPU, 렌더링 알고리즘

한 탁 돈(Han, Tack-Don)

1978년 연세대학교 전자공학과 학사
1983년 Wayne State Univ. 석사
1987년 Univ. of Massachusetts at Amherst 컴퓨터공학과 박사
1989-현재 연세대학교 컴퓨터과학과 교수

관심분야 : 고성능 컴퓨터 구조, 미디어 시스템 구조, HCI(Human Computer Interface), 유비쿼터스 컴퓨팅