Journal of Korea Game Society 2014 Dec; 14(6): 29-38
http://dx.doi.org/10.7583/JKGS.2014.14.6.29

게임 프로그래밍

실내 3D 게임 장면의 잠재적 가시 집합을 위한 효과적인 하드웨어 압축 구조

김영식
한국산업기술대학교 게임공학과
kys@kpu.ac.kr

An Effective Structure of Hardware Compression for Potentially Visible Set of Indoor 3D Game Scenes

Youngsik Kim
Dept. of Game & Multimedia Engineering, Korea Polytechnic University

요 약

대규모 실내 3D 게임 장면에서 차폐 컬링 정보를 미리 계산하는 잠재적 가시 집합 (potentially visible set: PVS)은 데이터를 처리하고 저장해야하는 양이 상당히 크지만 많은 부분이 0으로 표현된다. 본 논문에서는 모바일 환경의 3D 게임 장면 트리 구성 중에 PVS 데이터를 ZRLE (zero run length encoding) 방식으로 압축하는 효과적인 하드웨어 압축 구조를 설계한다. 3D 게임 시뮬레이션을 통하여 제안하는 구조의 PVS 데이터 압축 비율, PVS 컬링과 절두체 컬링에 따른 렌더링 속도 (frame per second: FPS)를 분석하였다.

ABSTRACT

In the large scale indoor 3D game scenes, the data amount of potentially visible set (PVS) which pre-computes the information of occlusion culling can be huge. However, the large part of them can be represented as zero. In this paper, the effective hardware structure is designed, which compresses PVS data as the way of zero run length encoding (ZRLE) during building the scene trees of 3D games in mobile environments. The compression ratio of the proposed structure and the rendering speed (frame per second: FPS) according to both PVS culling and frustum culling are analyzed under 3D game simulations.

Keywords : 3D Game Scene (3D 게임 장면), Potentially Visible Set (잠재적 가시 집합), Occlusion Culling (차폐 컬링), Hardware Compression (하드웨어 압축)

Received: Sept. 30, 2014 Accepted: Oct. 28, 2014
Corresponding Author: Youngsik Kim(Korea Polytechnic Univ.)
E-mail: kys@kpu.ac.kr

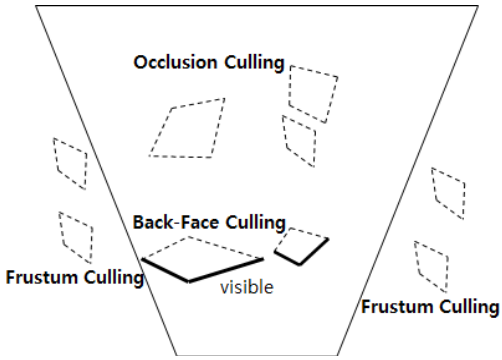
ISSN: 1598-4540 / eISSN: 2287-8211

1. 서 론

대규모 실내 3D 게임 장면은 수백만 개의 폴리곤과 많은 방(room)으로 이루어져 있지만 한 장면에서 보이는 부분은 매우 제한적이다[1]. 따라서 이러한 대규모 실내 3D 게임 장면에서 보이지 않는 오브젝트들을 포함한 모든 컨텐츠들을 렌더링하는 것은 매우 비효율적이며 실시간 렌더링을 위해서 적합하지 않다. 이를 해결하기 위해서 렌더링 프리미티브 개수를 줄이는 방법으로 [Fig. 1]에서와 같이 절두체 컬링(frustum culling), 뒷면 컬링(back-face culling), 차폐 컬링(occlusion culling) 등의 방법이 제시되었다[2].

절두체 컬링은 시야 절두체(view frustum)의 바깥에 있는 장면의 프리미티브들을 결정해서 선별하는 방법으로 이러한 프리미티브들은 당연히 보이지 않으므로 선별되어야 한다[3]. 3D 게임에서 충돌 처리 및 카메라 워크에 대한 절두체 컬링을 효율적으로 하기 위해서 공간 분할 트리(Spatial Partitioning Tree)를 구성 한다[4,5,6]. BSP(binary space partitioning)-트리는 분할 평면(split plane)이 축에 정렬되지 않지만 나머지 Quad-트리, Octree, KD-트리 등은 분할 평면이 축에 정렬(axis aligned)되어서 트리 빌드가 비교적 빠르다. back-to-front painter's 알고리즘을 적용하여 알파 블렌딩 렌더링을 해야 할 때 BSP-트리를 사용할 수 있다[6].

뒷면 컬링은 시야에서 보이지 않는 폴리곤의 면을 결정하는 방법이다[7]. 뒷면 컬링은 계층적으로 수행하면 모든 폴리곤에 대해서 개별적으로 검사할 필요 없다[8]. 절두체 컬링과 뒷면 컬링만 을 수행하면 여전히 상당한 양의 프리미티브들을 렌더링해야 한다. 이를 해결하기 위해서 장면 내에서 차폐되는 프리미티브들을 선별하여 컬링하는 차폐 컬링 연구가 많이 수행되었다[1,2, 9,10,11,12,13,14,15,16, 17,18].



[Fig. 1] Three types of visibility culling techniques: (i) frustum culling, (ii) back-face culling and (iii) occlusion culling[2].

차폐 컬링 방법은 현재 시야로부터 보일 가능성이 있는 잠재적 가시 집합(potentially visible set: PVS)을 구성해야 한다. PVS 데이터는 단일 시야점(point)을 기준으로 구성할 수 도 있고, 장면 내에서 다중 점을 갖는 방(room, cell)을 기준으로 구성할 수도 있다. 장면을 확대하여도 렌더링 속도를 떨어뜨리지 않도록 장면을 타일로 분할하여 차폐 컬링 연구가 수행되었다[16,17].

본 논문에서는 메모리 크기에 제약을 가지는 모바일 실내 3D 게임 콘텐츠의 BSP 공간 분할 트리에서 개별 리프(leaf) 노드 방을 기반으로 모든 다른 방들의 PVS 데이터를 계산하고 저장한다고 가정한다.

본 논문에서는 모바일 환경의 대규모 실내 3D 게임 장면에서 차폐 컬링 정보를 미리 계산해야 하는 PVS 데이터가 0을 많이 포함한다는 점을 착안하여 ZRLE(zero run length encoding) 방식으로 압축하는 효과적인 하드웨어 압축 구조를 설계한다. 3D 게임 시뮬레이션을 통하여 제안하는 구조의 PVS 데이터 압축 비율, PVS 컬링과 절두체 컬링에 따른 렌더링 속도(frame per second: FPS)를 분석하였다.

2. PVS 저장 공간 관련연구

3D 게임 공간 분할 트리를 구성하면서 PVS 데이터를 생성할 때 발생하는 저장 공간에 관한 연구가 많이 수행되었다[9,11,12,13,14,15].

대규모 실내 3D 장면에 대한 PVS 데이터를 미리 계산하는 것은 저장 공간 문제를 야기한다. 예를 들어서 10,000개의 방을 가지는 장면이고 각 방의 PVS 데이터가 10KB 라면 대략 100MB 저장 공간이 필요하다. 이러한 데이터는 실시간 3D 게임 렌더링을 위해서 미리 계산되어야 하는데 뷰 셀(view cell)을 형성하는 방의 개수는 매우 커지는 추세이므로 PVS 데이터 저장 공간 문제는 매우 중요하다. 논문 [9]에서는 오브젝트들과 방들을 클러스터링하면서 PVS 데이터를 압축하는 방법을 제시하였다. 논문 [11]에서는 대단위 3D 장면의 PVS 데이터를 미리 계산하기 위해 GPU (graphics processing unit) 기반의 병렬 가속 방법을 제시하였다. 논문 [12]에서는 PVS 데이터를 계층적으로 인코딩하는 방법을 제시하였다. 논문 [13,14]에서는 서버와 클라이언트 사이의 PVS 데이터 네트워크 전송 문제를 연구하였으며, 논문 [15]에서는 PVS 데이터의 중간 형태를 렌더링 도중에 온-더-플라이(on-the-fly)로 생성하고 저장하는 방법을 제시하였다. 그러나 PVS 데이터 저장 공간 문제를 다룬 이전 연구들은 PVS 데이터를 압축하는 전용(dedicated) 하드웨어 구조를 연구하지 않았다.

본 논문에서는 하드웨어 비용이 많이 들지 않고 PVS 데이터에서 0이 연속되는 경우가 많은 것을 고려한 효과적인 PVS 데이터 압축 하드웨어 구조를 제안한다.

3. PVS 데이터 압축 하드웨어 구조

이번 절에서는 실내 3D 게임 장면 트리에서 PVS 데이터의 단계 별 구성 방법과 ZRLE 방식으로 PVS 데이터를 압축하는 하드웨어 구조를 제안하고 설명한다. PVS 데이터를 계산하는 프로세스는 [18]의 3D 게임 엔진 알고리즘을 기반으로 하였다.

3.1 PVS 계산 프로세스

PVS 계산은 3D 장면을 컴파일하는 데 많은 시간이 소요되므로 보통 미리 계산해서 공간 분할 BSP 트리와 함께 파일로 저장한다. 그런 후 실제 3D 게임에서 BSP 트리와 PVS 데이터를 로딩해서 사용한다. BSP 트리를 위한 PVS 데이터 계산 프로세스는 크게 (1) 포탈 생성(portal generation)과 (2) PVS 계산(PVS Calculation)으로 나눌 수 있다.

[Fig. 2]에서는 포탈 생성에 대한 의사 알고리즘을 기술한다. 포탈은 방에서 다른 방으로 연결될 수 있는 연결 통로(doorway)이다. 이를 생성하기 위해서 BSP 트리의 각 노드들을 검사하여 자식 노드가 둘이면 해당 노드를 포함한 평면을 포탈 리스트에 포함시킨다. 그런 후 해당 평면을 BSP 트리 노드들로 클리핑해서 최종적으로 남는 부분을 포탈로 생성한다.

[Fig. 3]에서는 PVS 계산 5단계의 의사 알고리즘을 기술한다. 1단계는 양방향(two-way) 포탈들을 두 개의 단방향(one-way) 포탈로 재구성한다. 2단계는 모든 단방향 포탈 마다 뒤에 있는 포탈들은 포함하지 않고, 서로 같은 방향을 바라보는 앞쪽 포탈들의 리스트를 구성한다. 3단계는 각 포탈마다 홍수 채움(flood fill) 방식으로 가능한 가시적 리프 노드(방) 리스트를 계산한다. 4단계는 각 포탈마다 재귀적인 비음영 구역 클리핑(anti-penumbra clipping)을 수행하여 실제적인 리프 노드 가시성 리스트를 계산한다. 5단계는 각각의 리프 노드 (방)에 존재하는 포탈들의 리프 노드 가시성 데이터를 합산하여 PVS 데이터를 계산한다. 각각의 리프노드들을 기준으로 모든 리프 노드들의 가시성을 0 혹은 1로 표시한 데이터의 집합이 PVS 이다.

Pseudo Algorithm Portal Generation

```
foreach (NodeA in BSP Tree of Scene)
 If ( NodeA has two children )
    Generate initial portal on NodeA plane;
    Clip portal to BSP Tree;
    Add surviving fragments to a master list;
 endif
end foreach
```

[Fig. 2] A pseudo algorithm of portal generation[18]

Pseudo Algorithm PVS Calculation

STEP1 One-way portal
Create one way portals from two-way portals;

STEP2 Portal to portal visibility
Calculate temporary portal/portal visibility array for each portal;

STEP3 Leaf visibility
Calculate possible leaf visibility for each portal using flood fill;

STEP4 Anti-penumbra clipping
Calculate actual leaf visibility for each portal performing anti-penumbra clipping;

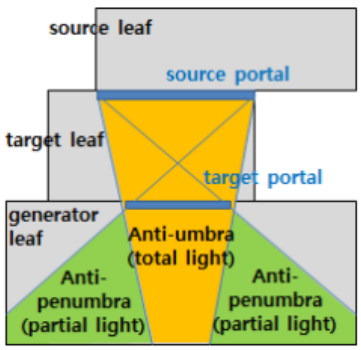
STEP5 Compiling leaf visibility
Calculate PVS of each leaf by accumulating leaf visibility of each portal;

[Fig. 3] A pseudo algorithm of PVS calculation[18]

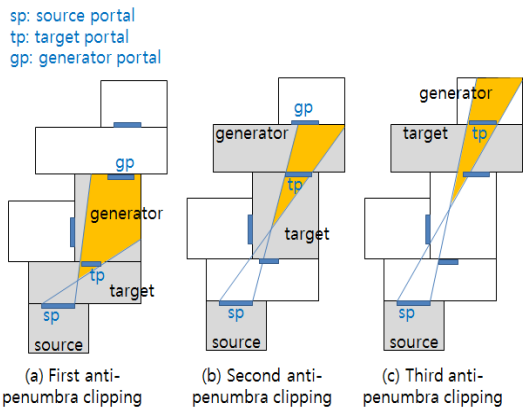
[Fig. 4]는 전영 구역(anti-umbra, total light)과 비음영 구역(anti-penumbra, partial light)를 설명한다. 3개의 방은 소스 리프(source leaf), 목표 리프(target leaf), 생성자 리프(generator leaf)로 나뉘고 각각의 방은 소스 포탈(source portal)과 목표 포탈(target portal)로 연결된다. 소스 리프에서 소스 포탈과 목표 포탈을 통해서 바라볼 때 생성되는 가시 영역은 전영 구역과 비음영 구역을 모두 포함한다.

[Fig. 5]에서는 [Fig. 3] 의사 알고리즘의 4단계를 재귀적인 비음영 구역 클리핑(anti-penumbra clipping)을 설명한다. 절두체 컬링과 유사하게 비음영 구역을 이루는 평면들을 구하고 그 평면들로 이루어진 볼륨(비음영 구역)의 외부에 있는 프리미티브들을 클리핑한다. [Fig. 5]에서 노란색으로 표시된 부분이 비음영 구역이다.

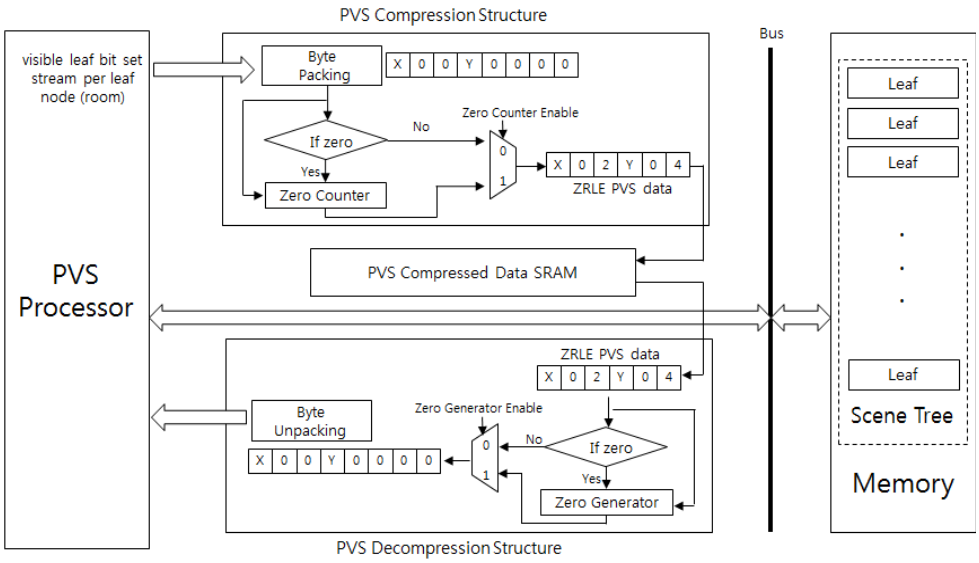
[Fig. 5] (a)의 첫 번째 클리핑을 수행한 후 (b)에서는 생성자 포탈은 목표 포탈로 바뀌어 두 번째 크리핑을 수행하고 이를 (c)에서 다시 생성자 포탈이 목표 포탈로 바뀌어 세 번째 클리핑을 수행한다. 이러한 수행을 비음영 구역 안에 생성자 포탈이 더 이상 존재하지 않을 때까지 재귀적으로 반복한다.



[Fig. 4] Anti-umbra and anti-penumbra



[Fig. 5] Recursive anti-penumbra clipping



[Fig. 6] PVS Compression Hardware Structure

3.2 ZRLE PVS 데이터 압축 구조

이번 절에서는 [Fig. 3]의 5단계 리프 노드마다 포함된 포탈들의 PVS 원시 데이터를 합산하고 저장하는 데이터를 효과적으로 압축하는 ZRLE 기반 하드웨어 구조를 설명한다.

[Fig. 6]의 압축 하드웨어 구성은 3.1절에서 설명한 PVS 프로세스를 수행하는 PVS 프로세서, BSP 트리 데이터가 저장되는 외부 메모리, PVS 압축기(compressor), PVS 복원기(decompressor), 그리고 압축 PVS 데이터를 저장하는 내부 SRAM으로 구성되어 있다. PVS 프로세서는 범용 CPU로 대치할 수 있다. PVS 압축기의 입력은 각 리프 노드마다 존재하는 포탈들의 원시 PVS 비트 데이터 스트림이다. 이를 바이트 단위로 묶고 바이트별로 0이 아니면 PVS 바이트 데이터로 기록하고 0이면 개수를 세어서 0과 함께 개수(zero run length)를 기록한다. 각 포탈들이 갖고 있는 PVS 원시 데이터들은 다른 리프 노드들의 가시성에 따라서 0 또는 1의 값을 갖는다. 많은 리프 노드들이 현재 시야에서 보이지 않으므로 각 포탈 별 원시 데이터는 많은 부분이 0이다.

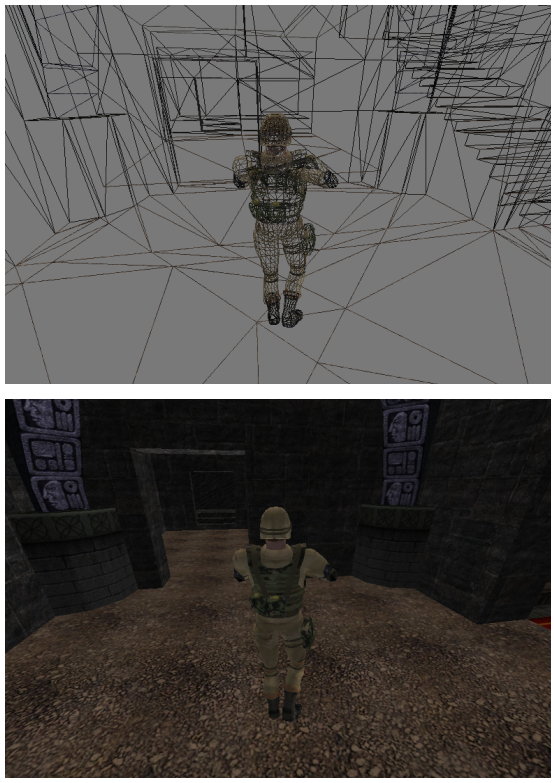
4장에서 실험한 결과 각 리프 노드마다 압축 비율(압축된 PVS 데이터 크기 / 원본 PVS 데이터 크기 x 100)이 100보다 큰 경우가 없기 때문에 PVS 압축기에서 압축한 데이터가 원본 데이터보다 큰 경우를 고려하고 처리하는 별도의 하드웨어는 필요하지 않다. PVS 압축기는 미리 정적 3D 콘텐츠의 공간 분할 트리를 구성하면서 PVS를 구성할 때 사용될 수 있다. 또한 동적 3D 콘텐츠의 PVS 데이터 구성에서는 PVS 압축기 하드웨어가 간단하고 압축 비율(4장 참조)이 좋기 때문에 온-더-플라이(on-the-fly)로 실시간 렌더링과 함께 사용될 수 있다.

PVS 복원기는 3D 게임 실행 도중에 압축된 PVS 데이터를 복원하면서 실시간 렌더링에 사용될 수 있다. PVS 복원기는 PVS 압축데이터가 저장된 내부 SRAM에서 바이트 단위의 압축 데이터를 읽어서 0이 아니면 그대로 기록하고 0이면 그다음 개수 데이터만큼 0을 생성한 후 연속으로 기록한다. 그런 후 비트단위 스트림으로 변환하여 PVS 프로세서에 전달한다.

이와 같이 제안하는 하드웨어 ZRLE PVS 데이터 압축 구조는 소프트웨어로 처리하였을 때 보다 외부 메모리 접근 없이 내부 SRAM 을 사용하여 처리할 수 있고, 간단한 하드웨어와 높은 압축 비율에 의해서 동적인 3D 콘텐츠 실시간 렌더링과 함께 사용될 수 있다.

4. 실험 및 평가

본 논문에서는 Game Institute 3D 게임[18]에서 정적 콘텐츠의 충돌 체크 및 절두체 컬링을 위해서 사용하는 BSP-트리 공간분할 트리와 방 기반의 PVS 데이터를 실험에 사용하였다. Game Institute 3D 게임 렌더링에서 절두체 컬링, PVS 컬링, BSP 트리 빌드, PVS 압축 데이터 생성 등의 실험 데이터를 수집하도록 프로그램을 수정하였다. 실험은 MacBookPro MC371KH/A 모델 (Intel(R) Core(TM) i5 CPU M520 @ 2.40GHz 2.40GHz 8GB RAM)에서 윈도우 모드로 측정하였다.

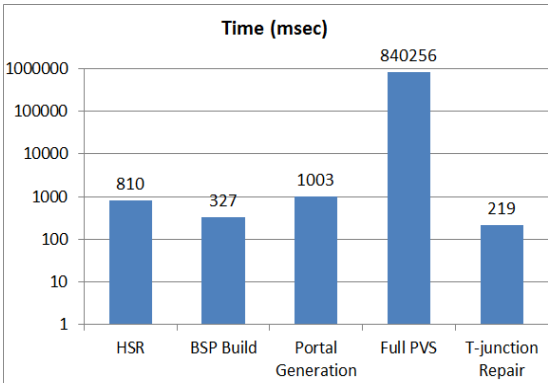


[Fig. 7] Game Institute 3D Game [18]

[Fig. 7]은 Game Institute 3D 게임의 첫 장면이다. Game Institute 3D 게임의 공간 분할 트리는 BSP-트리를 사용한다. BSP-트리를 구성하면 투명 건물을 표현하기 위해서 알파블렌딩을 위한 painter's 알고리즘을 적용하기 쉽고 PVS 데이터 생성과 연동할 수 있다. 즉 카메라 시점에서 back-to-front 정렬을 시키면서 렌더링하기 쉽다. Game Institute 3D게임에서 BSP-트리와 PVS 데이터를 구성한 정적 건물 콘텐츠의 크기는 plane 개수 = 393, tree node 개수 = 1423, leaf 개수 = 562, portal 개수 = 1568, vertex 개수 = 11287, polygon 개수 = 3909 이다. 비교적 크지 않은 콘텐츠이지만 PVS 데이터를 생성하는 시간은 매우 많이 소요되기 때문에 PVS가 성능에 미치는 영향은 크다.

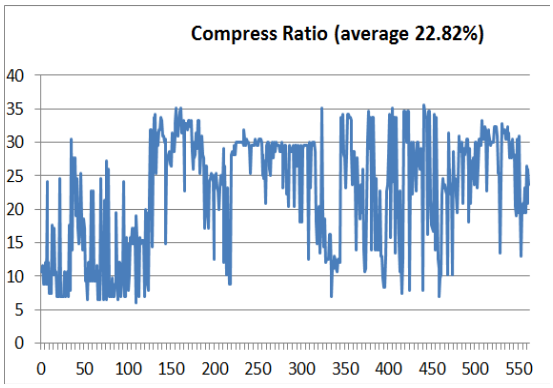
[Fig. 8]은 Game Institute 3D 게임에서 뒷면 제거 (HSR: hidden surface removal), BSP-트리 구성(BSP Build), 포탈 생성 (Portal Generation), PVS 데이터 계산(Full PVS), T-junction 제거 프로세스들의 시간(T-junction Repair: msec)을 측정한 것이다. T-junction 은 공간 분할 트리 구성 시에 분할 평면에 의해 쪼개지는 폴리곤에 의해 나타나는 시각적 불일치 현상이다. T-junction을 제거하는 방법은 T-junction 찾아내서 T-junction 위치에 정점을 추가하여 폴리곤을 쪼갠다 (tessellation)[5]. [Fig. 8]에서 알 수 있듯이 PVS 데이터 생성과 관련된 포탈 생성 프로세스 시간과, PVS 데이터 계산 시간이 다른 프로세스에 비하여 크다. 특히 PVS 데이터 계산 시간은 다른 프로세스들에 비하여 1000배 가까이 많이 소요됨을 알 수 있다. 본 논문에서 사용한 Game Institute 3D

컨텐츠는 포탈의 개수가 1568개로 비교적 크지 않는 데도 많은 PVS 데이터 계산 시간 (840sec)이 소요된다. 이는 [Fig. 3] 알고리즘에서 4단계 재귀적인 비음영 구역 클리핑 프로세스 부분이 절대적으로 많은 시간을 소요하기 때문이다. 모든 리프와 포탈에 대해서 비음영 구역을 찾아내고 클리핑을 재귀적으로 수행하므로 시간이 많이 소요된다.



[Fig. 8] Processing time for each step (msec).

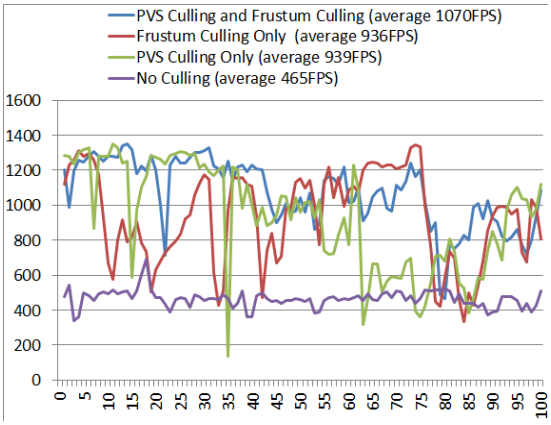
[Fig. 9]는 3.2절에 설명한 ZRLE PVS 데이터 압축 구조를 적용했을 때 각 리프 노드 562개에 대한 압축 비율(압축된 PVS 데이터 크기 / 원본 PVS 데이터 크기 x 100)을 측정한 것이다. 최고 6.02%, 최저 35.65%, 평균 22.82%로 비교적 압축 비율이 높다. 본 논문에서 실험에 사용한 컨텐츠보다 훨씬 많은 프리미티브, 방, 포털로 이루어진 대규모 실내 3D 장면의 경우에는 PVS 값이 0으로 표현되는 보이지 않는 방의 개수가 더 커져서 PVS 데이터 압축 비율이 높아질 것으로 예상된다. 따라서 ZRLE 압축 하드웨어 구조가 더욱 유용하게 사용될 수 있다. 리프 노드 방의 개수가 5000개 이하이고 각 방의 원시 PVS 데이터가 5KB인 컨텐츠의 경우에는 약 5MB 내부 SRAM을 사용하여 압축한 PVS 데이터를 저장하고 실시간 렌더링에 사용할 수 있을 것으로 예측된다.



[Fig. 9] Compress ratio according to leaf nodes [1:562].

[Fig. 10]은 Game Institute 3D 게임에서 절두체 컬링과 PVS 컬링 적용 유무에 따라 게임을 수행하면서 일정한 간격으로 동일하게 연속적으로 100번 렌더링 속도(FPS)를 측정하였다. 4가지 경우로 구분하여 (1) PVS 컬링과 절두체 컬링을 모두 적용한 경우 평균 1070FPS, (2) 절두체 컬링만 적용한 경우 평균 936FPS, (3) PVS 컬링만 적용한 경우 평균 939FPS, (4) 컬링을 적용하지 않은 경우 평균 465 FPS 로 나타났다. [Fig. 10]에서 보듯이 컬링을 하나도 적용하지 않은 경우는 모두 적용한 경우에 비하여 평균 렌더링 성능 저하가 56.5%에 달할 정도로 성능 저하가 심각함을 알 수 있다. 절두체 컬링과 PVS 컬링을 각각 하나만 적용한 경우는 비슷한 렌더링 성능을 보여주었으며 두 가지 컬링을 모두 적용한 경우에 비하여 각각 12.5%와 12.2% 성능 저하를 보여 주었다. 컬링을 하나라도 적용한 경우는 하나도 적용하지 않은 경우에 비하여 렌더링 속도의 변동(fluctuation)이 컸다. 그 이유는 3D 게임 플레이어의 움직임에 따라서 실내 3D 장면 프리미티브들에 대한 컬링의 효과가 달라지기 때문이다. 또한 같이 절두체 컬링이나 PVS 컬링 하나만 수행한 경우는 성능 저하가 컬링을 하나도 적용하지 않은 경우에 비하여 비교적 크지 않았다. 그 이유는 컬링을 하나만 적용해도 보이지 않는 프리미티브들을 많이 선별하고 PVS 컬링과 절두체 컬링이 선별하는 프리미티브

가 상당부분 겹치기 때문이다. 그러나 만약 컨텐츠가 커진다면 절두체 컬링과 PVS 컬링이 선별하는 프리미티브들의 겹치는 부분이 감소하여 각각의 컬링이 렌더링에 미치는 비중이 더 커질 것이다.



[Fig. 10] Rendering speed (FPS) for four cases in time slots [1:100].

5. 결 론

본 논문에서는 모바일 환경의 대규모 실내 3D 게임 장면에서 차폐 컬링 정보를 미리 계산해야 하는 PVS 데이터가 0을 많이 포함한다는 점을 착안하여 ZRLE 방식으로 압축하는 효과적인 하드웨어 압축 구조를 설계하였다. 3D 게임 시뮬레이션을 통하여 제안하는 구조의 PVS 데이터 압축 비율을 실험한 결과 22.82%로 우수하였다. 또한 PVS 컬링과 절두체 컬링이 렌더링 성능에 미치는 영향이 커서 컬링을 하나도 적용하지 않은 경우의 렌더링 속도는 모두 적용한 경우에 비하여 성능 저하가 56.5%로 심각하였다.

REFERENCES

[1] Hey, Heinrich, and Werner Purgathofer. "Occlusion culling methods." Proceedings of EUROGRAPHICS. 2001.

[2] D. Cohen-Or, Y.L. Chrysanthou, C.T. Silva, and F. Durand, "A survey of visibility for walkthrough applications," IEEE Transactions on Visualization and Computer Graphics, Vol. 9, No. 3, pp.412-431, 2003.

[3] Ulf Assarsson and Tomas Moller, "Optimized View Frustum Culling Algorithms for Bounding Boxes," Journal of Graphics Tools vol. 5 no. 1 pp. 9-22, 2000.

[4] Sangduk Kim, Jin-Woo Kim, Woo-Chan Park, Tack-Don Han, "Bounding Box based Shadow Ray Culling Method for Real-Time Ray Tracer," Journal of Korea Game Society, Vol. 13, No. 3, pp.85~94, 2013.

[5] Christer Ericson. Real-Time Collision Detection. The Morgan Kaufman series in interactive 3D technology. Morgan Kaufmann, 2005.

[6] Youngsik Kim, "Tree Build Heuristics for Spatial Partitioning Trees of 3D Games," Journal of Korea Game Society, Vol. 13, No. 4, pp.25~34, 2013.

[7] Tomas Moller and Eric Haines, Real-Time Rendering, pp.192-200, 1999

[8] Subodh Kumar, Dinesh Manocha, William Garrett and Ming Lin, "Hierarchical Back-Face Computation," Proceedings of Eurographics Workshop on Rendering, pp.235-244, 1996.

[9] Michiel van de Panne and James Stewart, "Efficient compression techniques for precomputed visibility," In Proceedings of Eurographics Workshop on Rendering '99, pp.305-316, 1999.

[10] D. Luebke and C. Georges "Portals and mirrors: Simple, fast evaluation of potentially visible sets," In Proceedings of the symposium on Interactive 3D graphics. ACM, pp.105-ff, 1995

[11] B. LIU, Z. LIU and Y. LAN, "GPU accelerated preprocessing for potential-visible set." In Computer-Aided Industrial Design & Conceptual Design, 2009 CAID & CD 2009, IEEE 10th International Conference on. IEEE, pp.957-962, 2009.

[12] Craig Gotsman, Oded Sudarsky, and Jeffry Fayman, "Optimized occlusion culling,"

Computer & Graphics, 23(5):645 - 654, 1999.
[13] Daniel Cohen-Or, Gadi Fibich, Dan Halperin, and Eyal Zadicario, "Conservative visibility and strong occlusion for viewspace partitioning of densely occluded scenes," Computer Graphics Forum, 17(3):243 - 254, 1998.
[14] Daniel Cohen-Or and Eyal Zadicario, "Visibility streaming for network-based walkthroughs," Graphics Interface '98, pp.1-7, June 1998.
[15] Vladlen Koltun, Daniel Cohen-Or, and Yiorgos Chrysanthou, "Hardware-accelerated from-region visibility using a dual ray space," Rendering Techniques 2001: 12th Eurographics Workshop on Rendering, June 2001.
[16] D. Gomez, P. Poulin, and M. Paulin, "Occlusion Tiling," in Graphics Interface 2011, May 2011, pp. 71 - 77.
[17] D. Gomez, et al. "Time and Space Coherent Occlusion Culling for Tileable Extended 3D Worlds." Computer-Aided Design and Computer Graphics (CAD/Graphics), 2013 International Conference on. IEEE, 2013.
[18] Game Institute, Graphics Programming with Direct X: Module II (Book 2) chapter 17, E-INSTITUTE PUBLISHING INC, www.gameinstitute.com, 2004.

김 영 식 (Youngsik Kim)

1993년 연세대학교 컴퓨터과학과 학사
1995년 연세대학교 컴퓨터과학과 석사
1999년 연세대학교 컴퓨터과학과 박사
1999년-2005년 삼성전자 System LSI 책임연구원
2013년 University of Pittsburgh 방문교수
2005년-현재 한국산업기술대학교 부교수

관심분야 : 게임기구조, 컴퓨터구조, 3차원 그래픽가속기, 임베디드 시스템 등