계층 자료구조의 결합과 3차원 클러스터링을 이용하여 적응적으로 부하 균형된 GPU-클러스터 기반 병렬 볼륨 렌더링

(Adaptive Load Balancing Scheme using a Combination of Hierarchical Data Structures and 3D Clustering for Parallel Volume Rendering on GPU Clusters)

이 원 종 † 박 우 찬 †† 한 탁 돈 †††
(Won-Jong Lee) (Woo-Chan Park) (Tack-Don Han)

요 약 대용량 볼륨 데이타를 가시화하는 효과적인 방법인 후-정열 병렬 렌더링은 부하균형에 의해 성능이 결정된다. 기존의 정적 데이타 분할 방법은 태스크 병렬성만의 관점에서는 자기균형을 쉽게 얻을 수 있었지만, 데이타 내부의 빈 공간을 고려하지 않았기 때문에 데이타 병렬성의 관점에서는 심각한 불균형을 초래할 수 있었다. 본 논문은 태스크 병렬성과 데이타 병렬성이 함께 고려된, 적응적이며 확장적인 부하 균형 기법을 제안한다. 우리는 계층적 자료 구조인 옥트리와 BSP-트리를 효과적으로 결합하여 볼륨 데이타의 실제 영역만을 추출하여 렌더링 노드들로 균등하게 분산시켰으며, 각 렌더링 노드들에서는 3차원 클러스터링 알고리즘을 적용하여 렌더링 순서를 효과적으로 결정하였다. 제안하는 방법은 기존의 정적 데이타 분산 기법에 비해 최대 22배의 병렬성을 높였고 동기화 비용을 낮추어 렌더링 성능을 크게 향상시켰음을 실험을 통해 알 수 있었다.

키워드 : 부하 균형, 옥트리, BSP 트리, 계층적 가시화, 병렬 렌더링, 볼륨 렌더링

Abstract Sort-last parallel rendering using a cluster of GPUs has been widely used as an efficient method for visualizing large-scale volume datasets. The performance of this method is constrained by load balancing when data parallelism is included. In previous works static partitioning could lead to self-balance when only task level parallelism is included. In this paper, we present a load balancing scheme that adapts to the characteristic of volume dataset when data parallelism is also employed. We effectively combine the hierarchical data structures (octree and BSP tree) in order to skip empty regions and distribute workload to corresponding rendering nodes. Moreover, we also exploit a 3D clustering method to determine visibility order and save the AGP bandwidths on each rendering node. Experimental results show that our scheme can achieve significant performance gains compared with traditional static load distribution schemes.

Key words : load balancing, octree, BSP tree, hierarchical visualization, prarallel rendering, volume rendering

1. 서 론

유체역학, 지질학, 생명과학, 의공학 등의 다양한 분야에서는 시뮬레이션과 수치 해석을 통해 3차원 데이타를 생성하고 다루고 있다. 이러한 3차원 볼륨 데이타를 가시화(visualization)하는 효과적인 방법은 직접 볼륨 렌더링(direct volume rendering)이다. 최근에는 보다 사실적으로 데이타를 가시화 하기 위해 높은 해상도 및 화질의 결과를 요구하고 있으며, 이는 원시적인 볼륨 데이타의 크기를 크게 증가하게 하였다. 수십에서 수백 기가바이트를 넘는 이러한 대용량 데이타는 사실상 단일 PC에서는 처리가 불가능하므로, 렌더링 노드가 추가되

† 학생회원 : 연세대학교 컴퓨터과학과
airtight@yonsei.ac.kr
†† 정 회 원 : 세종대학교 인터넷공학과 교수
pwchan@sejong.ac.kr
††† 종신회원 : 연세대학교 컴퓨터과학과 교수
hantack@kurene.yonsei.ac.kr
논문접수 : 2005년 4월 26일
심사완료 : 2005년 10월 19일

는 수만큼 메모리와 계산 성능을 확장할 수 있는 병렬 렌더링 기법을 널리 이용하고 있다. 병렬 렌더링은 객체의 정렬 및 분류 단계를 기준으로, 이미지 공간 기반(전-정렬, sort-first) 또는 객체 공간 기반(후-정렬sort-last)으로 분류할 수 있는데[1], 그 중 후-정렬의 접근방법은 데이타의 분할 및 분산이 용이하기 때문에, 볼륨 가시화의 대표적 알고리즘인 광선 투사법(ray casting) 기반[2-6] 및 3차원 텍스쳐 기반[7-10] 등의 많은 선행 연구에 사용되었다.

후-정렬 기반 병렬 볼륨 렌더링의 성능은 크게 두 가지 요인에 의해 좌우된다[11]. 첫째는 네트워의 주어진 시간 내 처리 작업량(communication network *throughput*)이고 둘째는 부하 균형(*load balancing*)이다. 후-정렬 방법은 최종 영상을 만들기 위해 마지막 단계에서 영상 합성(image compositing)을 수행 하는데 이 단계에서는 병렬 프로세스의 스케줄링을 위한 동기화가 수반되며 렌더링 노드간 과다한 대역폭을 유발시키게 된다. 이 문제의 효과적인 해결을 위한 소프트웨어 기반 합성 방법[12], 하드웨어 기반 합성 방법[9,13,14] 등이 제안되었다. 보다 직접적으로 성능에 영향을 미치는 두 번째 요인은 부하 균형이다. 순차적으로 처리되어야만 하는 부분이 존재한다면 병렬 알고리즘은 그 성능이 제한될 수 밖에 없다. 따라서, 병렬성을 극대화하고 동기화 비용을 최소화할 수 있는 잘 설계된 부하 균형 기법이 필수적으로 요구된다.

최근 들어 GPU(graphic processing unit), 메모리, 네트웍 장비, 저장 장치 등의 기술이 빠르게 발전함으로써 이러한 상용 장비(commodity off-the-shelf, COTS)들을 효과적으로 결합하여 고성능의 컴퓨팅 환경을 구축하는 GPU-클러스터가 다양하게 이용되고 있다. 특히, 3차원 텍스쳐 매핑은 최근의 상용 GPU에서 가속이 완벽하게 지원되고 있기 때문에 GPU-클러스터를 이용한 3차원 텍스쳐 기반의 병렬 볼륨 렌더링이 많은 각광을 받고 있다. 하지만, 이를 이용한 이전의 관련 연구들[7-10]에서는 부하 균형 문제를 구체적으로 고려하지 않았는데 대부분 볼륨 데이타를 동일한 크기의 부볼륨으로 분할하여 정적으로 분산시켰기 때문이다. 이러한 정적 분할 기법(*static partitioning*)은 분산된 부볼륨들이 같은 크기의 3차원 텍스쳐로 GPU에 적재되기 때문에, 태스크 병렬성(task parallelism)의 관점에서는 자기 균형(self-balance)을 쉽게 이룰 수 있다. 그러나, 이 방법을 이용하면 그림 1에서 보듯이 특정 부볼륨들은 다른 부볼륨들보다 상당히 많은 빈 공간을 포함하여 실제 데이타의 분배 비율이 비균등해 질 수 있다. 게다가, 이 문제는 렌더링 시 전달 함수(transfer function)가 수정되어 볼륨 데이타의 가시영역이 변경될 때는 더욱 자주 일어난다. 이는 각 렌더링 노드들에 빈 공간 생략(empty space skipping) 기법들[15,16]을 적용해도 해결되지 않는데, 이는 클러스터 내의 특정 노드에서 렌더링이 가속된다 하더라도 전체적인 부하 균형에는 영향을 주지 못하기 때문이다.

빈 공간 제거(empty space skipping)는 기존의 단일 GPU상의 텍스쳐 기반 볼륨 렌더링에서 사용되었던 옥트리를 활용하면 쉽게 구현 할 수 있지만, 병렬 렌더링의 부하 균형을 위해서는 빈 공간이 아닌 부볼륨들을 같은 수로 묶어 각 렌더링 노드들로 전송해야만 하는데 이는 전체 계층구조를 깨트리게 되어 부볼륨들의 가시순서를 소실시킬 수 있다. 계층을 유지하지 위한 차선책으로 트리 구조 자체를 렌더링 노드의 수만큼 부-트리로 분할하여 각 부-트리와 해당 부볼륨들을 전송하는 병렬 옥트리 등을 생각할 수도 있지만, 이 방법으로는 여전히 부하 불균형을 피할 수가 없다.

우리는 이러한 문제를 서로 다른 두 개의 계층적 자료구조(옥트리 and BSP 트리)의 결합으로 해결하였다.

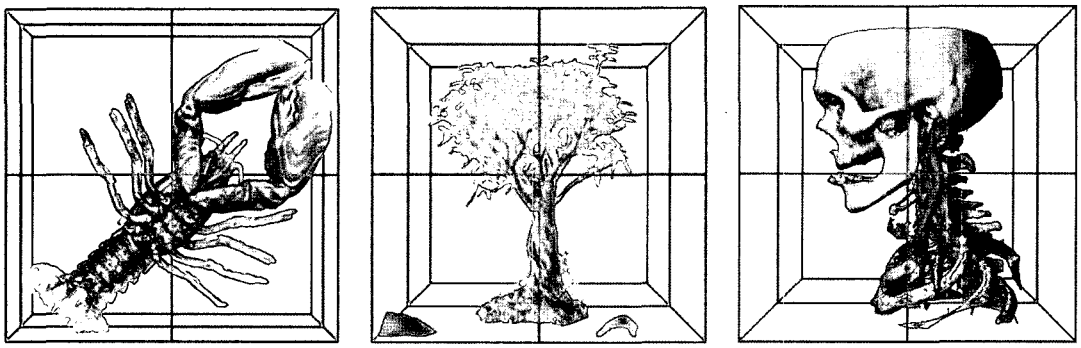


그림 1 **정적 분할 방법의 맹점.** 전통적인 정적 분할방법은 태스크 병렬성의 관점에서는 자기균형을 얻을 수 있지만(같은 크기로의 분할) 실제 볼륨 데이타에는 공백 공간이 상당부분 포함되어 있기 때문에 데이타 병렬성의 관점에서는 심각한 불균형을 초래한다.

옥트리에 의해 공백 공간이 제거된 부볼륨만을 효과적으로 재구성하고, 이들의 공간상의 분포를 기준으로 직교적 BSP-트리(orthogonal BSP-tree)를 통해 이를 균등하게 그룹화하여 해당 렌더링 노드들로 전송함으로써, 실제 데이타에 대해서의 부하 균형을 효과적으로 얻을 수 있었다. 또한, 각 렌더링 노드들에서는 3차원 클러스터링 알고리즘[17]을 적용하였는데 이는 전송된 옥트리 블릭들을 가시 순서에 따라 새롭게 지역 BSP트리로 계층화하여 올바른 렌더링과 합성순서를 유지할 수 있게 한다.

제안하는 기법은 정적 부하균형 기법에 비해 렌더링의 병렬성을 높일 수 있고, 노드간 동기화 시간을 감소시켜 전체 성능을 향상시킬 수 있다. 또한 전달 함수가 변경되었을 때 모든 데이타의 재전송이 아닌 필요한 옥트리 블릭들만을 노드간 교환하여 동적으로 부하 재균형 및 가시 순서 재결정을 수행할 수 있다.

제안하는 기법은 nVidia GeForce 5950 Ultra가 장착된 9-PC의 GPU-cluster(교토(京都) 대학의 VG-cluster 시스템[13])에서 평가되었고, 이를 통해 전체 병렬성 및 렌더링 성능 측면에서 전통적인 정적 분할 방법에 비해 큰 폭의 향상이 있었음을 알 수 있었다.

본 논문의 구성은 다음과 같다. 2장에서는 관련연구에 대해 간단히 살펴보고, 3창에서는 전통적 정적 부하 균형 방법을 기술한다. 4장에서는 제안하는 적응적 부하 균형 방법을 설명하고 5장에서는 실험결과를 기술하고, 마지막으로 6장에서 결론을 맺는다.

2. 관련 연구

부하 균형은 광선 투사 기반의 병렬 렌더링에서 주로 고려된 문제이다. 전통적인 광선 투사 기반의 볼륨 렌더링은 쉽게 병렬화가 가능한데, 이는 주사되는 가상의 가시광선들에 의해 픽셀 값이 결정되기 때문이다. 만일 객체를 담고 있는 데이타베이스에 빠르게 접근할 수 있다면 각 가시선은 다른 프로세서와 일체의 통신 과정이 필요 없이 독립적으로 계산될 수 있다[18]. 따라서, 이 방법은 상대적으로 고비용의 메모리 공유 구조(shared memory architecture)에 적합하다. Karia 등[2]은 객체 공간상에서 볼륨 데이타를 분할했고, 그 중 투명하지 않은 복셀만을 렌더링하는 방법을 제안했다. 이는 노드간의 부하 불균형을 초래할 수 있었는데 산재화된 분해(scattered decomposition)라는 방법을 통해 이를 해결했다. Silva 등[5]은 볼륨 데이타를 그들의 PVR(parallel volume rendering)시스템을 위한 내용 기반 부하 균형(content based load balancing)이란 방법을 사용했다. 이는 볼륨 데이타를 BSP-트리를 이용하여 효과적으로 분할하였지만 데이타의 빈 공간은 고려하지 않았다. Ma 등[3]은 구동 가능한 렌더링 노드들에 데이타들을 적절히 배분하여, 이진 교환 합성(binary swap compositing)이라는 방법을 통해 최종 영상을 생성하는 병렬 알고리즘을 제안했고, 또한 그들은 [4]에서 라운드-로빈 할당(round-robin assignment) 정책을 이용해 근사적인 정적 부하 균형을 수행했다. 최근 들어, Gao 등[19]과 Wang 등[6]은 공간-채움 곡선(space-filling curve) 탐색 기법을 이용해 효과적으로 공간적 지역성(spatial locality)이 유지된 균형적인 데이타 분배를 수행했다.

텍스쳐 매핑 기반의 병렬 볼륨 렌더링의 경우 부하 균형을 고려한 관련 연구는 그리 많지 않은데, 이는 서론에서도 언급했듯이 정적 분할 방법만으로 자기 균형을 얻을 수 있다고 판단되어 왔기 때문이다. Muraki 등[9]은 최소-경계-입방체(best-fitted bounding cube)를 이용해 효과적으로 볼륨 데이타를 분할했지만, 상대적으로 많은 계산이 요구되는 전처리과정을 소모했고 또한 완전하게 빈 공간에 대해 생략을 하지 못했다. 따라서 그들의 분할 방법은 좀처럼 프레임 비율을 증가시키지 못했다.

빈 공간 생략은 전통적인 광선 투사법 기반의 볼륨 렌더링을 위한 가속화 방안으로 주로 사용되어 온 기술이다. 볼륨 데이타에는 최종 영상을 생성하는 데에는 불필요한 여러 가지 특징들이 존재하고, 이의 조기 발견을 통해 메모리 접근 및 계산을 상당량 줄일 수가 있었다. 최근 프로그래밍 가능한(programmable) GPU 기술이 발전함에 따라 3차원 텍스쳐 뿐만 아니라 광선 투사법도 GPU에서 구현 가능하게 되었는데 Kruger 등[15]은 GPU 기반의 광선 투사 알고리즘을 제안하였고 이를 통해 효과적으로 빈 공간 생략을 적용하였다. Li 등[16]은 볼륨을 비균일한 부볼륨들로 분할하여 투명하지 않은 복셀들만 렌더링하는 최적화된 기법을 제안하였다.

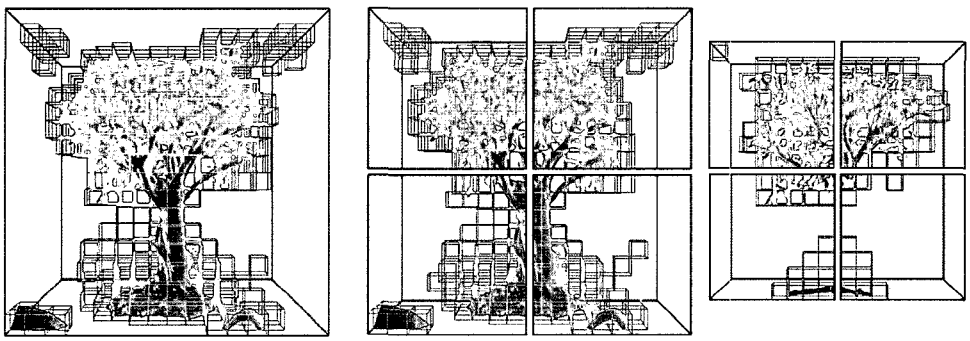
3. 정적 부하 균형 기법

본 절에서는 전통적인 두 가지 부하 균형 기법인 정적 분할과 병렬 옥트리에 대해 기술한다. 정적 분할방법은 후-정렬 접근방법에서 그 분할과 분산방법의 용이함 때문에 다양하게 사용되었다[7-10]. 전처리 단계에서 입력 데이타는 렌더링 노드의 수에 맞추어 동일한 크기의 부볼륨으로 분할된다. 이 때 객체 공간에서 데이타가 분할되기 때문에 렌더링 시 각 노드들에서 생성된 영상은 이미지 공간에서 중첩될 수 있다. 따라서 합성단계에서 이들의 순서를 적절하게 결정 한 후 블렌딩하여 최종 영상을 생성하게 된다. 이 방법은 저비용의 전처리 과정과 자기-균형의 특징을 제공하지만 빈 공간을 고려하지 못하기 때문에 상대적으로 비효율적이다(그림 1).

이 문제를 해결할 수 있는 하나의 대안은 대용량 데이타의 시뮬레이션과 가시화를 위한 몇몇 연구에서 사용된[20-22] 병렬 옥트리를 사용하는 것이다. 병렬 옥트리 알고리즘들에는 약간씩 차이가 있지만, 기본적으로 다음과 같은 순서로 진행된다. 1) 전역 옥트리를 생성한다. 볼륨은 각 축에 대해 공간적 중간값을 기준으로 분할하여 8개의 부볼륨을 재귀적으로 생성한다. 이 과정에서 그림 2(a)와 같이 빈 공간은 효과적으로 검출되고 생략된다. 2) 전역 옥트리가 분해된다. 렌더링 노드의 수에 따라 루트 노드 아래의 8개의 자식노드에 연결된 서브트리들은 그들의 부모-자식 관계를 유지한 채 각각 렌더링 노드들로 전송되어 각각 새로운 지역 옥트리로 정의된다. 3) 마지막으로, 각 렌더링 노드에서 렌더링과 합성이 수행된다. 각 지역 옥트리에 할당된 부볼륨들은 특정한 탐색순서에 의해 렌더링 되는데, 시점에 따른 부볼륨들의 처리 순서는 [23]에서처럼 각 노드들에서 테이블 참조 방법(table-based lookup method)으로 쉽게 구할 수 있다. 병렬 옥트리 방법은 빈 공간을 효과적으로 생략하여 각 노드들의 렌더링을 부분적으로 가속할 수는 있지만, 그림 2(b)와 같이 여전히 정적 분할 방법을 사용하기 때문에 부하 불균형 문제를 근본적으로 해결할 수가 없다.

4. 제안하는 적응적 부하 균형 기법

제안하는 적응적 부하 균형 기법은 크게 네 가지 단계로 구성된다(그림 3). 1)옥트리 생성, 2)BSP 생성,



(a) 전역 옥트리 (b) 분산된 지역 옥트리

그림 2 **병렬 옥트리 기반 정적 분할**. (a) Bonsai 볼륨은 전역 옥트리에 의해 부볼륨들로 분할되고 빈 공간을 제외한 실제 데이타만이 표현된다. (b) 루트 노드 아래 8개의 자식 노드에 해당하는 가지들은 8개의 렌더링 노드들로 분산되어 각각 지역 옥트리로 구성된다.

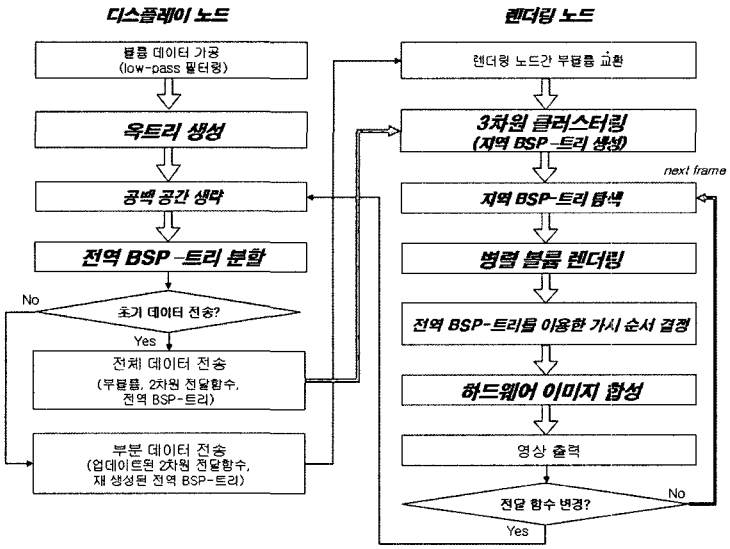


그림 3 제안하는 적응적 부하 균형 기법의 처리 순서도

3)3차원 클러스터링, 4)렌더링 및 합성이 그것이다. 본 장에서는 각 단계별로 자세히 설명한다.

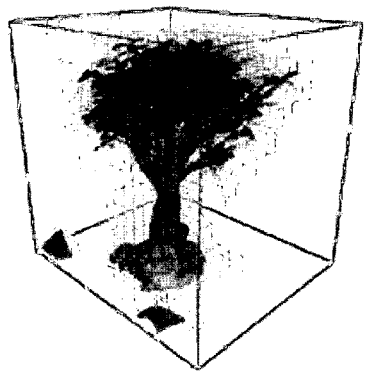
4.1 옥트리 생성

제안하는 기법의 첫 단계는 각 부볼륨의 공간 정보를 담고 있는 옥트리 자료구조를 생성하는 것이다. 각 부볼륨은 전체 옥트리 계층구조의 리프 노드에 할당 되고, 각 노드들은 부볼륨 내부의 복셀 스칼라 값의 최대, 최소값을 저장한다. 이 최대/최소값과 전달 함수에 정의된 투명도 값을 이용하면 투명한 노드들을 완벽하게 검출할 수 있기 때문에 실제 데이타만을 포함하는 노드들만이 전체 옥트리 계층에 효과적으로 삽입된다(그림 4(a)).

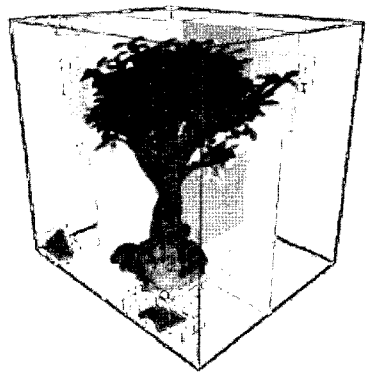
렌더링의 성능에 영향을 주는 또하나의 주요한 요인은 부볼륨의 크기이다. 최대한 많은 빈 공간을 제거하기 위해서는 당연히 볼륨을 구성하는 최소한의 단위인 각 복셀을 부볼륨으로 정의하는 것이다. 하지만, 이는 계산되어야 할 텍스쳐와 다각형 좌표에 대한 정점의 수를 크게 증가시켜 GPU의 버텍스 쉐이더에 오버헤드를 가중시키고 CPU-GPU 간의 AGP 대역폭을 크게 늘리게 된다. 따라서, 빈 공간 생략의 정도와 부볼륨의 크기(즉, 생성되는 부볼륨의 개수)사이의 상충관계는 성능을 결정하는 중요한 요인이 될 수 있다. 우리는 원본 볼륨의 크기에 비례하여 다양한 크기로 실험을 수행한 결과 일반적으로 16×16×16 부볼륨이 가장 좋은 성능을 위한 최적의 크기임을 알 수 있었다. 이에 대한 구체적인 설명은 5장에서 다룬다.

4.2 직교적 BSP-트리 생성

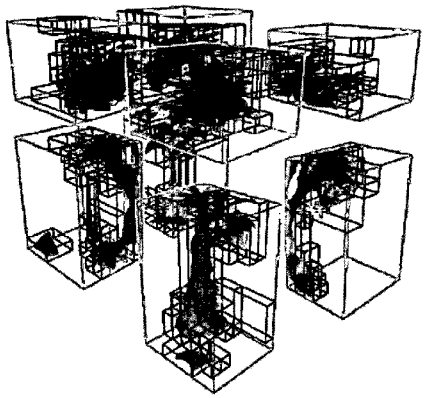
볼륨 데이타를 적절하게 분할하기 위해, 옥트리에 의해 분할된 부볼륨들은 직교적 BSP-트리로 변환된다. 직교적 BSP-트리(또는 K-d 트리)는 모든 분할 단면(splitting plane)의 방향이 항상 좌표축과 같음을 의미



(a) 전역 옥트리 분할



(b) 전역 BSP 트리 분할



(c) 3차원 클러스터링(지역 BSP-트리 분할)

(d) 렌더링 및 합성

그림 4 **제안하는 적응적 부하 균형 기법**. (a) 전역 옥트리에 의해 Bonsai 볼륨은 부볼륨들로 분할되고 불투명한 영역들만 표현된다. (b) 옥트리 계층은 전역 BSP-트리에 의해 균등하게 분해된다. (c) 분해된 부볼륨 그룹들은 8개의 렌더링 노드들로 전송되고 3차원 클러스터링 알고리즘을 이용해 새로운 계층구조(지역 BSP-트리)를 생성한다. (d) 렌더링과 합성이 병렬적으로 수행된다.

한다. 우리는 각 축에 대한 분할 단면의 위치를 구하기 위해 Berger 등[17]이 제안한 *signature list*를 이용하였다. 부볼륨에 대한 위치정보는 아래의 함수로 표현될 수 있다(그림 5).

$$F:[1,...,n_x]\times[1,...,n_y]\times[1,...,n_z]\mapsto\{0,1\} \quad (1)$$

n_x, n_y, n_z는 각 x-, y-, z-축 방향으로의 부볼륨들의 수를 의미한다. 인덱스 x, y, z의 위치에 해당하는 부볼륨이 빈 공간이라면 F(x, y, z) = 0, 그렇지 않다면 F(x, y, z) = 1이 된다. xy, yz, zx평면에 평행인 각 슬랩(slab, 부볼륨들로 이루어진 슬라이스)에 대해서, 빈 공간이 아닌 부볼륨의 수는 signature list, S에 계산되고 기록된다. 예를 들어, yz평면에 평행하고 x축을 가르는 슬랩에 대한 signature list는 아래와 같이 주어진다.

$$S_{yz}(x)=\sum_{y=1}^{n_y}\sum_{z=1}^{n_z}F(x,y,z) \quad (2)$$

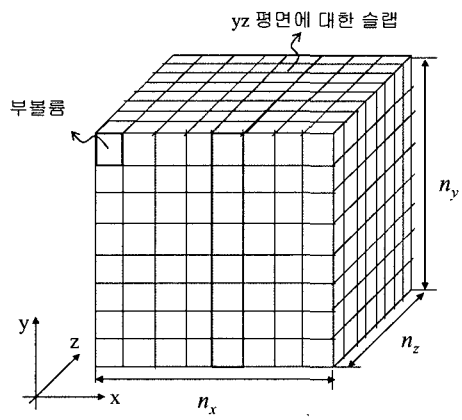
나머지 두 개의 평면(xy, zx)에 대해서도 같은 방법으로 계산된다. 우리는 signature list를 이용하여 현재 축을 기준으로 노드를 양분하는 분할 평면에 대한 위치의 계산은 그림 6의 의사코드에 따라 수행하였다. 일단 분할 평면의 위치가 결정되면, 부볼륨들의 공간은 분할 평면에 의해 전/후 공간으로 나뉘게 되고 이는 두 개의 자식노드에 할당된다. 이 과정은 다음 축에 대해서 분할된 공간의 수가 렌더링 노드의 수와 같을 때까지 재귀적으로 반복된다. 생성된 BSP-트리의 각 리프 노드들에는 자신의 signature list들과 객체 공간상에서의 경계 입방체의 위치정보(최대/최소값)를 저장한다(그림 4(b)). 전역 BSP-트리가 생성되면, 각 리프 노드들의 경계 입방체에 포함된 원본 볼륨의 각 부분 볼륨들, 전역 옥트리, 전역 BSP-트리가 패킹되어 각 렌더링 노드들로 전송된다.

4.3 부볼륨의 3차원 클러스터링

```
1:  분할을 위한 하나의 축을 선택한다.
2:  현재의 축에 대해서 signature lists, S를 계산한다.
3:  두 개의 인덱스 및 합에 대한 변수를 다음과 같이 초기
    화한다.
    lower = 0, upper = 현재 축에 대한 S의 길이
    leftsum = S(lower), rightsum = S(right)
4:  while lower < upper do
5:      if leftsum > rightsum do
6:              lower = lower + 1
7:              leftsum = leftsum + S(lower)
8:      else
9:              upper = upper - 1
10:             rightsum = rightsum + S(right)
11:     end if
12: end while
13: lower 또는 upper가 분할을 위한 인덱스로 선택된다.
```

그림 6 분할평면의 위치계산에 대한 의사코드

렌더링을 위해서는 새로운 계층구조가 필요하다. 이는 부볼륨들이 포함된 공간을 전역 BSP-트리로 분할하면 이전에 기술된 옥트리의 부모-자식관계가 더 이상 유지되지 않기 때문이다. 또한, 영상의 왜곡을 방지하기 위해서는 렌더링시 부볼륨들의 가시 순서를 결정해 주어야 하기 때문이다. 따라서, 우리는 Berger 등[17]에 의해 제안되었고AMR(adaptive mesh refinement) 등의 데이타를 가시화하는데 사용된[24] 3차원 클러스터링 알고리즘을 사용하였다. 그림 7은 클러스터링 알고리즘의 한 예제이며 설명의 편의를 위해 2차원으로 표현하였다. 이는 다음과 같이 처리된다. 1) 그림 7(b)와 같이 최소 경계 상자(best-fitted bounding box)를 구하기 위해 signature list(식 (2))의 0-값을 갖는 외부의 행 또는 열이 제거된다. 2) 모든 0-값을 갖는 내부의 행 또는 열이 주어진 영역을 둘로 나누는 지점으로 선택된다. 3) 더 이상 모든 signature가 0이 아니라면, 모든 signature들에 대해서 다음의 라플라시안 2차 도함수



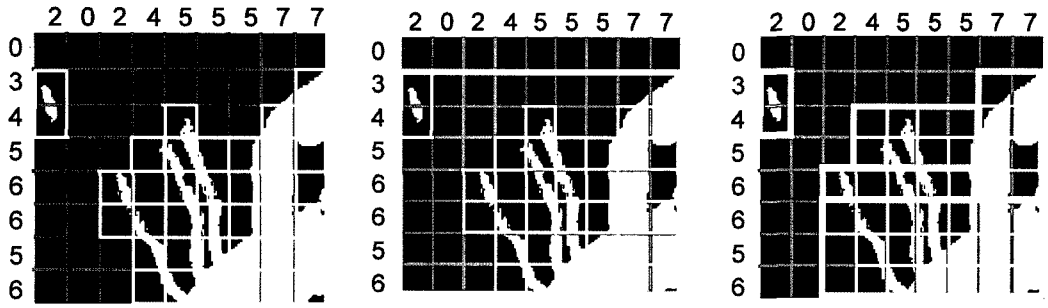
인덱스 (x, y, z)에 해당하는 부볼륨

$$F:[1,...,n_x]\times[1,...,n_y]\times[1,...,n_z]\mapsto\{0,1\}$$

yz평면에 평행한 임의의 한 슬랩에 대한 Signature List

$$S_{yz}(x)=\sum_{y=1}^{n_y}\sum_{z=1}^{n_z}F(x,y,z)$$

그림 5 부볼륨의 위치정보 및 빈공간 여부에 따른 기호화와 signature list



(a) 옥트리 격자와 signature list (b) 외부의 0값이 포함된 행 및 열의 제거 (c) 내부의 0과 최대 굴절 지점은 분할을 위한 인덱스로 선택된다.

그림 7 2차원 클러스터링 알고리즘 예

(*Laplacian* second derivative)가 계산된다(i는 signature list의 인덱스를 의미한다).

$$\Delta_{yz}(i) = S_{yz}(i+1) - 2S_{yz}(i) - S_{yz}(i-1) \quad (3)$$

$\Delta_{zx}(i), \Delta_{xy}(i)$ 에 대해서도 동일

이 들 중 값이 가장 큰 굴절 지점(refraction point)이 분할 인덱스로 선택된다(그림 7(c)). 이 과정은 새롭게 생성된 부영역들에 대해서도 재귀적으로 수행되고, 다음의 탈출 조건을 하나 이상 만족하면 종료된다. 1) 부영역들이 미리 결정된 효율비(efficiency ratio)을 초과했을 때이다. 주어진 영역상에 대한 효율비는 빈 공간이 아닌 부볼륨의 수 당 총 부볼륨 수로 정의 되고, 이 값이 미리 설정한 특정한 값을 넘었을 때이다. 2) 주어진 영역에 포함된 총 부볼륨의 수가 어떤 최소 신장 값(minimal extension value)보다 작을 때이다. 실험치에 따르면[24] 효율비는 0.8~1.0, 최소 신장 값은 15가 가장 좋은 결과를 주었고, 본 논문에서도 같은 값을 설정했다.

결과적으로, 각 렌더링 노드들에 전송된 옥트리 부볼륨들은 임의의 순환 문제(cycle problem)없이 효과적으로 묶이게(클러스터링) 되고 이들은 새롭게 정의된 지역 BSP-트리의 리프 노드들에 할당된다(그림 4(c), 그림 8). 이를 통해 렌더링 시 트리 탐색만으로 가시 순서를 쉽게 구할 수 있으며, 옥트리 부볼륨 다수 개를 묶어 하나의 부볼륨으로 만들기 때문에 렌더링 시 부볼륨과 단면들과의 교점 계산이 줄어들어, 옥트리만을 사용한 방법보다 AGP 대역폭을 감소시키는 효과도 얻을 수 있다[24].

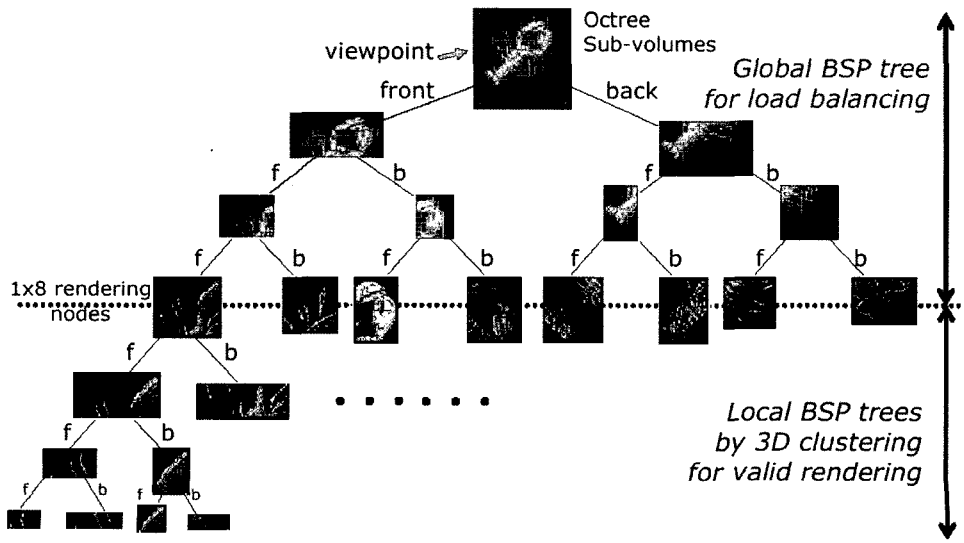


그림 8 **전역 BSP-트리와 지역 BSP-트리**. 임의의 시점에 대해서 렌더링 순서는 각 지역 BSP-트리에 의해 쉽게 결정될 수 있다. 각 렌더링된 영상들은 같은 방법으로 전역 BSP-트리에 의해 정렬되어 합성된다.

4.4 분산된 계층구조의 렌더링 및 합성

전처리 과정이 끝난 후, 클러스터링된 부볼륨들은 3차원 텍스쳐 매핑 방법으로 렌더링된다. 렌더링 노드들에서는 전송받은 부볼륨들을 GPU에 하나의 3차원 텍스쳐로 적재한다. 이 때, 데이타 간 불연속으로 발생하는 영상의 왜곡을 방지하지 위해 경계면에 해당하는 복셀들은 공유되도록 저장된다. 단, 이는 렌더링 노드들의 3차원 텍스쳐간의 접면에서만 중복되고, 3차원 텍스쳐 내부의 부볼륨 간에서는 일체의 중복은 일어나지 않는다. 라이팅 연산시 필요한 변화량 벡터(gradient vector), 변화량 벡터의 크기(gradient magnitude)를 저장하기 위해 또 다른 RGBA 3차원 텍스쳐가 사용된다. 이렇게 두 개의 텍스쳐가 설정되면 각 부볼륨에 대해 동일한 간격의 단면들을 생성하고 각 단면에 텍스쳐가 매핑된 후 각 단면들은 GPU의 프레임 버퍼에서 블렌딩된다. 부볼륨과 단면들의 교차점의 정점 좌표들은 CPU에서 계산된 후 GPU로 전송되는데, 우리는 교차점에 해당하는 텍스쳐 좌표는 CPU에서 계산하여 전송하지 않고 GPU로 전송된 정점좌표를 이용하여 GPU의 버텍스 쉐이더에서 계산함으로써 CPU-GPU 간의 AGP 대역폭을 반으로 감소시켰다. 이는 옥트리 공간상의 정점과 텍스쳐 좌표는 비례적인 특징이 있기 때문이다. 원본 볼륨은 객체 공간상의 (x, y, z)좌표의 범위는 [-1, 1]이며 텍스쳐 공간상의 (s, t, u)좌표의 범위는 [0, 1]이기 때문에 전송된 정점좌표에 대해 x, y, z ,축 방향으로 각각0.5씩 scaling 후 translate을 수행하면 쉽게 계산되고, 이는 버텍스 쉐이더 내에서 한번의 매트릭스 곱셈으로 가능하다.

부볼륨들의 렌더링 순서는 모든 부볼륨들이 BSP-트리에 기술되었기 때문에 쉽게 결정될 수 있다. 지역 BSP-트리의 탐색 및 렌더링은 각 렌더링 노드에서 다음과 같이 진행된다. 분할 평면들에 대한 시점(view point)의 상대적인 전후 관계가 파악되고, 이를 기준으로 두 개의 자식노드(전, 후)에 대해 지역 BSP-트리의 중위 탐색(in-order traversal)이 시작된다. 그리고, 탐색이 리프 노드에 도달하게 되면 해당 부볼륨이 단면(slice-by-slice)의 순서로 렌더링된다. 마찬가지로, 각 렌더링 노드들에서 생성된 부영상(sub-image)들의 가시순서는 전역 BSP-트리를 탐색하며 결정될 수 있고 합성단계 시 이용된다(그림 8). 가시 순서는 평행투형(parallel projection)의 경우 시방향(view direction)이 변경되었을 때, 원근투영(perspective projection)의 경우 시점(view point)가 변경될 때마다 새롭게 계산 되어야 하는데, BSP 트리 기반의 이러한 가시 순서 결정 방법은 임의의 시점에 대해 평행 투형 및 원근 투영의 경우 모두 이용될 수 있다.

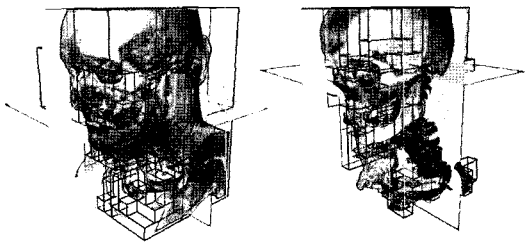


그림 9 전달함수의 변경과 그에 따른 적응적 데이타 재분할

제안하는 옥트리+BSP-트리 기법은 기본적으로 전달함수에 종속되는데, 렌더링 시 전달함수가 변경되었을 때에도 저비용이면서 적응적인 방법으로 부하균형을 유지할 수가 있다. 그림 9는 서로 다른 전달함수에 따른 CT Skull 데이타의 렌더링된 결과를 분할 평면과 클러스터링 박스와 함께 표현한 그림이다. 전달함수가 변경되어 볼륨 데이타의 가시 영역이 변경된다면 분산된 부볼륨들의 부하가 변경된다. 따라서, 이전의 계층 구조들은 재생성 되어야 하고 이에 따라 원본 볼륨의 분할도 다시 이뤄져야 하는데, 우리는 모든 데이타의 완전한 재전송을 수행하지 않고, 필요한 부볼륨만을 렌더링 노드간 교환하는 방법을 사용하여 노드간 통신 비용을 최소화 하였다(그림 3).

5. 실험 결과

제안하는 기법은 C++과 OpenGL로 구현되어 실험에 사용되었다. 렌더링은 NVIDIA의 상위수준 쉐이딩 API인 Cg를 이용하여 프래그먼트당 라이팅(per-fragment lighting) 및 후-음영(post-shading) 연산을 수행하였다. 전달 함수를 위해서 2차원 종속 텍스쳐(dependant texture)를 사용하였고, 또한, 모든 노드간 통신은 병렬 프로그래밍 API인 MPICH/PM을 이용하여 수행하였다.

성능 평가를 위해 교토(京都) 대학의 VG-cluster 시스템[13]이 사용되었다. 이 시스템은 9개의 노드로 구성된 클러스터와 미츠비시 프리시젼(*Mitsubishi Precision, Co. Ltd.*)에서 제작된 이미지 합성 하드웨어[14]로 구성된다. 각 노드들은 2.4GHz의 Intel Pentium 4 프로세서와 1GB의 DDR SDRAM 주메모리, 그리고 256MB의 비디오 메모리를 포함한 NVIDIA GeForce FX 5950 ULTRA GPU 및 렌더링 결과를 이미지 합성 하드웨어로 전송하기 위해 특화된 PCI 인터페이스 카드가 장착되어있다. 클러스터는 노드간 통신을 위해 기가비트 이더넷 네트워크 장비를 사용하고, OS로 Red-Hat Linux 7.3 기반의 SCore 5.6.1을 사용하였다.

우리는 "Lobster"(SUNY Stony Brook 제공), "Leg"

표 1 실험에 사용된 테스트 볼륨 데이타셋

Dataset	Intensity (byte)	Data Resolution	Texture Resolution	Texture Size (Mbytes)
Lobster	1	301×324×56	512×512×64	84
Leg	1	341×341×93	512×512×128	168
VHM	1	430×240×939	512×256×1024	671

(German Federal Institution 제공), and Visible Human Male (VHM, National Library of Medicine 제공) 데이타들을 실험을 위해 사용하였다(표 1). (이들의 렌더링 된 결과는 그림 10에서 볼 수 있다.) 렌더링 노드의 수를 하나부터 8개까지 증가시키며 이에 대한 성능 증가 추이를 테스트하기 위해 단일 GPU에도 적재 가능한 크기의 Lobster 및 Leg데이타를 사용하였다. GPU의 텍스쳐 메모리로 적재하기 위해서는 해상도가 2의 거듭제곱이어야 하는 제한이 있기 때문에, 표 1에서 보듯이 데이타가 패딩(padding)되어 해상도가 확장되었다. 표 1에서 텍스쳐 크기는 GPU의 메모리로 적재되는 볼륨 데이타 텍스쳐 뿐아니라 변화량 벡터 텍스쳐까지 포함된 크기이다.

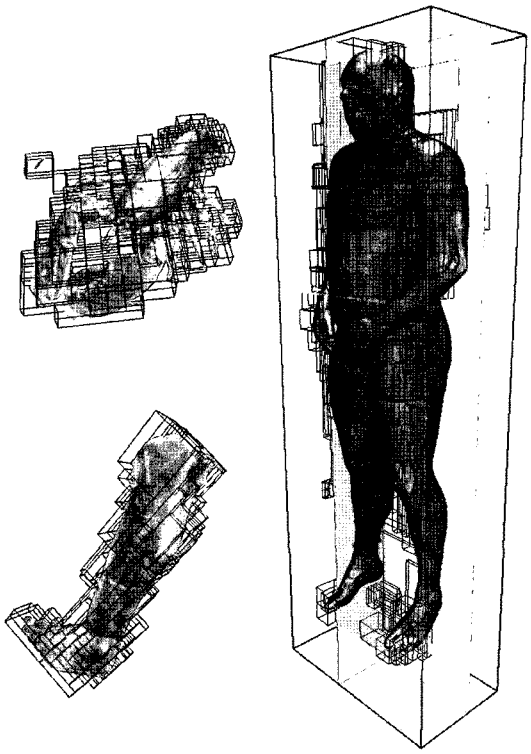


그림 10 제안하는 기법에 의해 렌더링 된 테스트 데이타셋. Lobster(좌측 상단), Leg(좌측 하단), Visible Human Male(우측) Lobster와 Leg는 클러스터링 상자와 함께 표현되어 있고, VHM은 클러스터링 상자, 공간을 균등히 나누는 분할 평면와 함께 표현되어 있다.

Leg 데이타는 흥미로운 특징이 있는데, 볼륨 내부의 실제 '다리'에 해당하는 부분이 한쪽으로 완전히 치우쳐져 있다는 것이다. 따라서, 제안하는 기법의 적응성을 테스트하는데 상당이 유용하였다. 본 연구는 가시화를 위한 볼륨 데이타가 클러스터의 전체 그래픽 메모리에 포함된다는 전제하에 진행되었다.

만일 그렇지 않은 경우는 디스크의 일부를 활용하는 out-of-core 렌더링이나 데이타의 크기를 줄이는 다중-해상도(multi-resolution)[6] 또는 압축 등의 방법을 활용하면 렌더링이 가능할 것으로 판단된다.

제안하는 기법(OCT+BSP)의 성능은 [7-10]에서 사용한 전통적인 방법(STANDARD) 그리고 [20-22]에서 사용한 옥트리만을 사용한 경우(OCT)와 비교 분석되었다.

5.1 전처리 시간

제안하는 OCT+BSP는 부하균형을 이루기 위해 그만큼의 전처리 단계가 요구되기 때문에 이의 수치적인 접근을 시도하였다. 표 2는 STANDARD, OCT, OCT+BSP의 소요되는 전처리 시간의 비교이다. 전처리 시간은 옥트리 생성, 빈 공간 검사, BSP 트리 생성, 네트워크를 통한 데이타 분배, 볼륨 데이타의 변화량 벡터 계산, 3차원 클러스터링으로 분류될 수 있다.

전체적으로 전처리 시간에서 상대적으로 많은 시간이 요구되는 것은 데이타에 대한 모든 접근 및 계산을 필요로 하는 "빈 공간 검사", "변화량 벡터 계산" 및 네트워크를 통해 전송하는 "데이타 분배" 단계이다. OCT와 OCT+BSP는 빈 공간 검사 단계가 요구되므로 그 만큼 STANDARD보다 상대적으로 전처리 시간이 필요했지만, 검사 이후 필요한 데이타만이 전송되므로 STANDARD의 경우보다 "데이타 분배" 단계 시 시간이 덜 소비되었다. OCT+BSP는 OCT에 비해 "BSP트리 생성" 및 "3차원 클러스터링" 단계가 필요한데 이 단계는 상당히 빠르게 처리되므로 충분히 무시될 정도이다.

전체 전처리 시간은 명백히 데이타 크기에 비례하겠지만, 동일 데이타를 기준으로 알고리즘 간의 소요시간 비교 시OCT+BSP는 STANDARD에 비해 1~3초 정도 더, OCT에 비해서는 거의 동일한 전처리 시간이 소비되었음을 알 수 있었다. 이는 제안하는 OCT+BSP의 전처리 비용이 렌더링 결과에 대비하였을 때 그렇게 크지 않음을 알 수 있다.

표 2 전처리 시간 비교

데이타셋	Lobster(301×324×56)			Leg(341×341×93)			VHM(430×240×240)		
알고리즘	STANDARD	OCT	OCT+BSP	STANDARD	OCT	OCT+BSP	STANDARD	OCT	OCT+BSP
옥트리 생성	-	0.114	0.114	-	0.114	0.114	-	0.114	0.114
빈 공간 검사	-	1.074	1.074	-	2.222	2.222	-	5.469	5.469
BSP 트리 생성	-	-	0.002	-	-	0.002	-	-	0.025
데이타 분배	0.576	0.399	0.399	0.509	0.338	0.338	5.018	3.167	3.167
변화량 계산	0.133	0.096	0.096	0.248	0.077	0.077	1.133	0.693	0.693
3D 클러스터링	-	-	0.001	-	-	0.001	-	-	0.007
총 시간	**0.709**	**1.682**	**1.684**	**0.757**	**2.752**	**2.754**	**6.151**	**9.442**	**9.474**

5.2 부하 균형

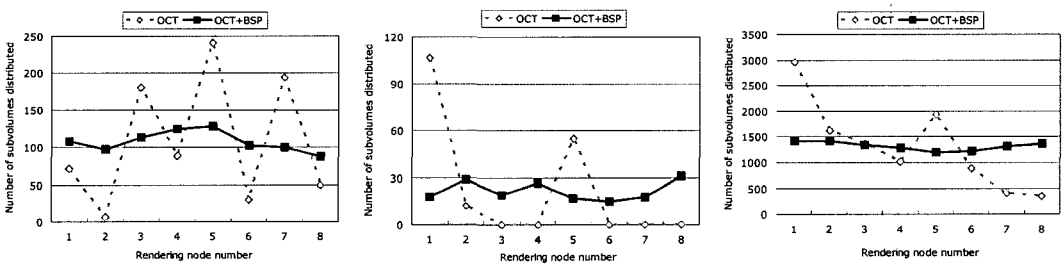
OCT+BSP 기법의 성능은 몇 가지의 기준으로 평가되었다. 그 중 하나는 각 렌더링 노드에 전송된 부볼륨의 개수로, 이는 얼마나 잘 렌더링 노드들에 부볼륨들이 균등하게 분배되었는지를 알 수 있는 지표가 된다. 그림 11은 OCT 및 OCT+BSP를 사용하였을 때 각각 8개의 렌더링 노드로 전송된 부볼륨의 수의 비교 그래프이다. 또한, 더 많은 렌더링 노드들에 대한 실험을 위해서 16개의 렌더링 노드의 경우도 측정을 하였는데(그림 12) 이는 디스플레이 노드에서 전송될 노드들로 부볼륨의 개수를 사전에 계산하면 쉽게 구할 수 있다. 단, STANDARD의 경우는 빈 공간을 생략하는 분할 기반의 방법이 아니기 때문에 본 비교 시 고려되지 않았다.

OCT는 부하 불균형의 경향을 두드러지게 보여주고 있다. 렌더링 노드로 전송된 부볼륨의 개수의 최대값과 최소값의 차는 무려 2,587(VHM, 8-렌더링 노드)이나 되었으며, Leg 데이타셋의 경우 반 이상의 렌더링 노드들은 데이타가 전송되지 않아 유휴상태이었다. 이는 OCT와 같은 정적 분할 방법은 빈 공간 생략을 활용했음에도 효율이 상당히 좋지 않음을 알 수 있다. 반면에 OCT+BSP의 경우 모든 경우에 대해 대략적으로 부하 균형이 유지되고 있음을 알 수 있고, 이는 각 렌더링 노드들에서 GPU에 의한 렌더링이 대략 같은 시간에 종료될 것으로 생각될 수 있다.

보다 수치적인 평가를 위해 우리는 새로운 평가 척도인 부하 균형율(load balance ratio, LBR)을 정의하였다. 이는 얼마나 부하가 잘 분배되었는지를 알려주는 비율로 각 노드에 전송된 부볼륨 수의 최대값과 0이 아닌 최소값을 기준으로 계산된다.

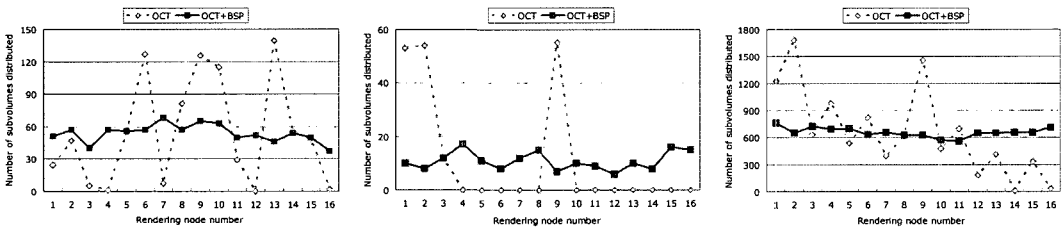
$$LBR = 1 - \frac{N - N_m}{N} \tag{4}$$

N은 전송된 부볼륨 수의 최대값, N_m은 최소값이다.



(a) Lobster (b) Leg (c) VHM

그림 11 8개의 렌더링 노드들로 전송된 부볼륨의 수 비교



(a) Lobster (b) Leg (c) VHM

그림 12 16개의 렌더링 노드들로 전송된 부볼륨의 수 비교

LBR 1은 완전하게 부하 균형이 된 상태를 의미하며, LBR이 0에 가깝다면 그만큼 부하가 불균형이 심한 것을 의미한다. 데이타가 전송되지 않게 된 렌더링 노드가 최소 한 개 이상 존재한다면 LBR은 쉽게 0이 될 수가 있다. 표 3은 렌더링 노드의 수를 변경하며 측정한 LBR의 비교이다. 렌더링 노드의 수가 늘어날 때 VHM에 대한 OCT+BSP 대 OCT의 상대적인 LBR은 함께 증가하였는데, 이는 제안하는 OCT+BSP 알고리즘이 렌더링 노드의 수에 따라 확장적(scalable)임을 알 수 있다.

표 3 LBR의 비교

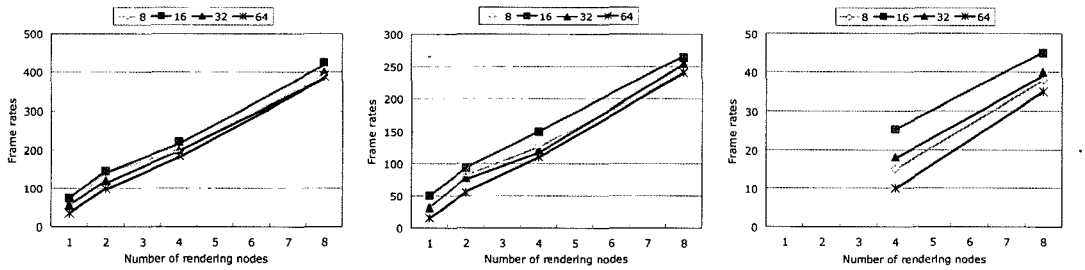
Dataset	Lobster		Leg		VHM	
Nodes	1×8	1×16	1×8	1×16	1×8	1×16
OCT	0.03	0	0	0	0.12	0.04
OCT+BSP	0.68	0.54	0.48	0.35	0.85	0.74
Rates	22.67	-	-	-	7.08	18.5

5.3 렌더링 성능

우리는 제안하는 OCT+BSP 기법이 후-정렬 기반의 병렬 볼륨 렌더링에 얼마나 잘 적용되는지를 보기 위해 실제의 렌더링 성능을 평가하였다. 4.1절에서 언급했듯이 부볼륨의 크기가 작을수록 더 많은 빈 공간을 걸러낼 수 있지만, 그만큼 계산되어야 할 정점 갯수를 증가시켜 성능 감소를 유발할 수 있다. 따라서, 정점증가에 대한 AGP대역폭의 부담을 감당할 수 있으면서 빈 공간을 최대한 걸러낼 수 있는 최적의 부볼륨의 크기를 우선적으로 찾아야 했다. 그림 13은 8^3에서 64^3까지의 부볼륨 크기에 따른 OCT+BSP의 프레임 비율의 비교 그래프이다(합성 단계 제외). 그림에서 보듯이 세 데이타셋 모두 크기 64^3부터 부볼륨의 크기가 작아질수록 볼륨 내 빈 공간을 그 만큼 더 걸러내어 성능이 증가되지만 크기 8^3가 되는 순간 정점수가 급증하여 성능이 오히려 떨어지게 된다. 따라서 우리는 부볼륨의 크기를 16^3로 설정하였다.

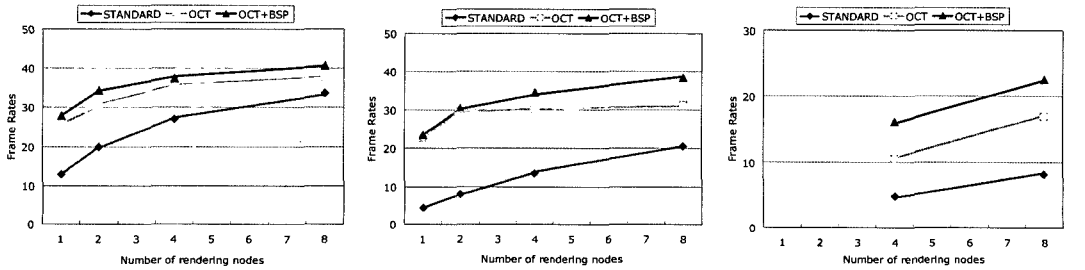
그림 14는 세 테스트 볼륨 데이타에 대해 512×512 해상도에서 렌더링 노드의 수를 변경하며 측정한 프레임 비율의 비교 그래프이다. VHM의 경우는 상대적으로 데이타의 크기가 크기 때문에 GPU 메모리의 한계로 최소한 4개 이상의 렌더링 노드가 필요했다.

대부분의 경우 1에서 8까지의 노드의 수가 늘어남에 따라 프레임 비율이 증가됨을 알 수 있었다. STANDARD는 상대적으로 작은 크기의 데이타(Lobster, Leg)에 대해 거의 선형적으로 프레임 비율이 증가했지만, 빈 공간이 생략되지 못했기 때문에 렌더링 성능은 상대적으로 좋지 않았다. OCT와 OCT+BSP는 실제 데이타만을 렌더링할 수 있었기 때문에 더 나은 성능을 보였고, 특히 OCT+BSP는 균형있는 부하의 배분이 이루어 졌기 때문에 OCT의 성능을 상회하였다. OCT는 초당 16-37 프레임을, OCT+BSP는 초당 22-40프레임



(a) Lobster (b) Leg (c) VHM

그림 13 부볼륨 크기에 따른 OCT+BSP의 프레임 비율 (Hz) 비교 (합성 단계 제외)



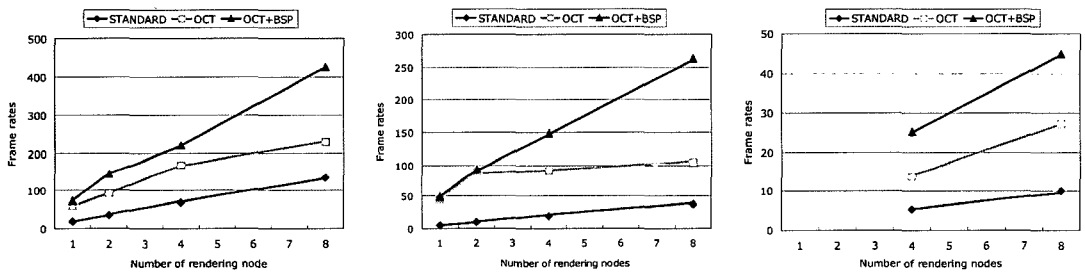
(a) Lobster (b) Leg (c) VHM

그림 14 STANDARD, OCT, OCT+BSP의 프레임 비율 (Hz) 비교

을 렌더링 하였다.

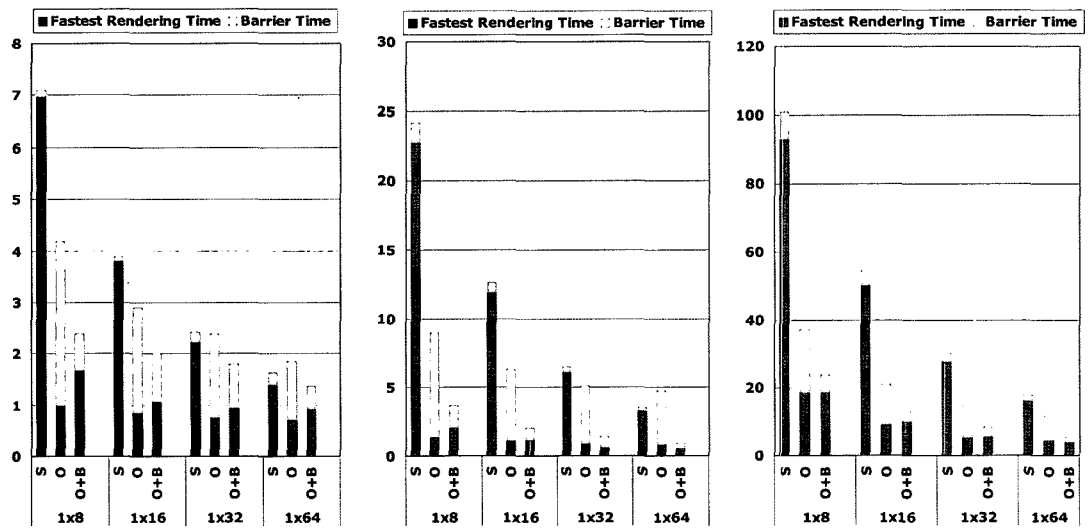
OCT와 OCT+BSP는 STANDARD와 같이 성능이 선형적으로 증가하지 않았는데, 이는 영상 합성 단계에 그 원인이 존재한다. 렌더링된 결과를 하드웨어 합성기에 전송하기 위해서는 GPU의 프레임 버퍼에 저장된 영상을 AGP-버스를 거쳐 주메모리로 다시 읽어 오는 시간 및 이를 다시 PCI 버스를 통과해서 하드웨어 합성기로 전송되는 시간이 요구된다. 이로 인해 렌더링 시간이 합성 시간보다 짧은, 상대적으로 크기가 작은 데이타(Lobster, Leg)의 경우에는 합성 단계가 병목점이 될 수 있다(상대적으로 데이타 크기가 큰, 즉 순수 렌더링 시간이 긴 VHM의 경우는 성능의 증가비율이 상대적으로 선형적임을 주목하라). 따라서, 우리는 영상 합성단계를 제외한 순수 렌더링 단계만의 성능을 측정해 보았는데(그림 15) OCT의 경우 초당 27-229 프레임, OCT+BSP의 경우 초당 46-426 프레임의 성능을 기록하였다. 또한, 그래프에서 보듯이 OCT+BSP의 경우는 렌더링 노드의 수에 따라 성능이 선형적으로 증가된다. 따라서, OCT+BSP의 부하균형 알고리즘 자체는 렌더링 노드의 수에 충분히 확장적임을 알 수 있다. 따라서, 만일 좀 더 지능적인 영상 합성기를 사용하거나, *PCI-Express* 와 같은 좀 더 빠른 버스를 활용한다면 렌더링 시간에 합성 시간이 가려져서(hidden) 보다 확장적인 렌더링이 가능할 것으로 판단된다.

그림 16은 한 프레임을 렌더링 하는데 필요한 렌더링 시간을 클러스터 크기별로 도시화한 비교 그래프이다. 렌더링 시간은 가장 빨리 종료된 렌더링 노드의 수행시간과 노드간의 스케줄링을 위한 소프트웨어 동기화 시간(barrier time, *MPI_barrier*)으로 구분된다. 이는 같은 해상도에서는 항상 동일한 시간을 소모하기 때문에 합성에 대한 시간은 고려되지 않았다. 렌더링 노드의 수 16부터 64까지에 대한 측정은 단일 렌더링 노드에서 각각에 대해 다수번의 순차적 렌더링으로 시뮬레이션하여 값을 구하였다.



(a) Lobster (b) Leg (c) VHM

그림 15 STANDARD, OCT, OCT+BSP의 프레임 비율 (Hz) 비교 (합성 단계 제외)



(a) Lobster (b) Leg (c) VHM

그림 16 한 프레임을 렌더링 시에 필요한 소모시간 비교

STANDARD의 경우는 동기화 시간이 지극히 짧았는데, 이는 단순히 자기-균형의 특징 때문이었고 렌더링은 상대적으로 매우 긴 시간을 소모하였다. OCT는 OCT+BSP에 비해, 가장 먼저 종료된 렌더링 노드의 수행시간이 짧지만 불균형한 부하 분배 때문에 상대적으로 긴 동기화 시간(전체의 57~83%)을 소모하였다. 이에 비해, OCT+BSP는 STANDARD보다 짧은 렌더링 시간과 STANDARD, OCT보다 짧은 동기화 시간(전체의 29~48%)을 기록했고, 이를 통해 전체 렌더링 성능을 높이는데 기여할 수 있었음을 알 수 있었다.

5. 결 론

본 논문에서 우리는 GPU 클러스터를 이용한 후-정렬 기반 볼륨 렌더링을 위한 효과적인 부하 균형 기법을 제안했다. 옥트리와 BSP-트리를 이용하여 빈 공간을 효과적으로 제거하였고 균등하게 데이타를 전송하였다. 3차원 클러스터링 알고리즘은 계층구조를 재생성시켜 렌더링 순서를 오차없이 유지하게 하였다. 빈 공간 생략과 적응적 부하 분배로 우리는 높은 성능 향상을 얻을 수 있었고, 증가된 병렬성과 줄어든 동기화 비용은 전통적인 정적 부하 분배 방법에 비해 더 나은 성능을 얻는데 기여하였다.

현재 우리는 PCIExpress 기반의 GPU(NVIDIA GeForce 6800 GT)와 듀얼 CPU 를 활용하여 더 효과적인 가시화 소프트웨어를 개발하고 있다. 이는 듀얼 프로세서 중 하나에서 이미지 합성 쓰레드를 병렬로 수행하여, 고비용의 하드웨어 이미지 합성기를 대치할 수 있을 것으로 예상된다. 전처리 과정에서 각 시간 단계(time step)별 각 데이타들에 대해 옥트리 및 BSP트리로 적용할 수 있기 때문에, 시간-가변 데이타(time-varying dataset)에 대해서도 확장 가능할 것으로 보인다. 또한, 최근 GPU의 특징을 활용[15]해 조기 광선 종료(early ray termination)와 같은 또 다른 최적화 기법도 통합시킬 예정이다.

참 고 문 헌

[1] S. Molnar, M. Cox, D. Ellsworth, and H. Fuchs., "A sorting classification of parallel rendering," *IEEE Computer Graphics and Applications*, Vol. 14, No.4, pp. 23-32, 1994.

[2] R.J. Karia., "Load balancing of parallel volume rendering with scattered decomposition," In *Proc. Scalable High Performance Computing Conference '94*, pp. 252-258, 1994.

[3] K.L. Ma, J. Painter, C. Hansen, and M. Krogh., "A data distributed parallel rendering algorithm for ray-traced volume rendering," In *Proc. Parallel Rendering Symposium '93*, pp. 117-126, 1993.

[4] K.L. Ma and S. Parker., "Massively parallel software rendering for visualizing large-scale data sets," *IEEE Computer Graphics and Applications*, Vol. 21, No. 4, pp. 72-83, 2001.

[5] C. Silva, A. Kaufman, and C. Pavlakos., "PVR: High-performance volume rendering," In Proc. *IEEE Computational Science and Engineering '96*, pp. 18-28, 1996.

[6] C. Wang, J. Gao, and H.W Shen., "Parallel multi-resolution volume rendering of large data sets with error-guided load balancing," In *Proc. Symposium on Parallel Graphics and Visualization '04*, pp. 18-25, 2004.

[7] J.M. Kniss, P. McCormick, A. McPherson, J. Ahrens, J. Painter, A. Keaheyand, and C. Hansen., "T-Rex: Interactive texture-based volume rendering for large data sets," *IEEE Computer Graphics and Applications*, Vol. 21, No. 4, pp. 52-61, 2001.

[8] M. Magallon, M. Hopf, and T. Ertl., "Parallel volume rendering using PC graphics hardware," In *Proc. Pacific Graphics '01*, pp. 384-389, 2001.

[9] S. Muraki, E.B. Lum, K.L. Ma, M. Ogata, and X. Liu., "A PC cluster system for simultaneous interactive volumetric modeling and visualization," In *Proc. Parallel Visualization and Graphics '03*, pp. 95-102, 2003.

[10] M. Strengert, M. Magallon, D.Weiskopf, S. Guthe, and T. Ertl., "Hierarchical visualization and compression of large volume datasets using GPU clusters," In *Proc. Symposium on Parallel Graphics and Visualization'04*, pp. 1-7, 2004.

[11] D. Bartz, B. Schneider, and C. Silva., "Rendering and visualization in parallel environments," In *Proc. SIGGRAPH 2000 Coursenote*, 2000.

[12] K.L. Ma, J.S. Painter, C. Hansen, and M.F. Krogh., "Parallel volume rendering using bnary-swap image composition," *IEEE Computer Graphics and Applications*, Vol. 14, No. 4, pp.59-68, 1994.

[13] J. Nonaka, N. Kukimoto, N. Sakamoto, H. Hazama, Y. Watashiba, X. Liu, M. Ogata, M. Kanazawa, and K. Koyamada., "Hybrid hardware accelerated image composition for sort-last parallel rendering on graphics clusters with commodity image compositor," In *Proc. Symposium on Volume Visualization and Graphics '04*, pp. 17-24, 2004.

[14] M. Ogata, S. Muraki, X. Liu, and K.L. Ma., "The design and evaluation of a pipelined image compositing device for massively parallel volume rendering," In *Proc. Workshop on Volume Graphics '03*, pp. 61-68, 2003.

[15] J. Kruger and R. Westermann., "Acceleration techniques for GPU based volume rendering," In *Proc. IEEE Visualization '03*, pp. 287-292, 2003.

[16] W. Li, K. Mueller, and A. Kaufman., "Empty space skipping and occlusion clipping for texture-

based volume rendering," In *Proc. IEEE Visualization '03*, pp. 317-324, 2003.
[17] M.J. Berger and I. Rigoutsos., "An algorithm for point clustering and grid generation," *IEEE Transaction on Systems, Man, and Cybernetics*, Vol. 21, No. 5, 1991.
[18] T.W. Crockett., "Parallel rendering," *Encyclopedia of Computer Science and Technology*, Vol. 34, No. 19, pp. 335-371, 1996.
[19] J. Gao, J. Huang, H.W. Shen, and A. Kohl., "Visibility culling using plenoptic opacity functions for large data visualization," In *Proc. IEEE Visualization '03*, pp. 341-348, 2003.
[20] D. Bartz, W. Straßer, R. Grosso, and T. Ertl., "Parallel construction and isosurface extraction of recursive tree tructures," In *Proc. WSCG, '98*, 1998.
[21] J. Bielak E.J. Kim and O. Ghattas., "Large-scale northridge earthquake simulation using octree-based multiresolution mesh method," In *Proc. ASCE 16th Engineering Mechanics Conference '03*, 2003.
[22] L.A. Freitag and R.M. Loy., "Adaptive, multiresolution visualization of large data sets using parallel octrees," In *Proc. ACM/IEEE conference on Supercomputing '99*, 1999.
[23] W.G. Aref and H. Samet., "An algorithm for perspective viewing of objects represented by octrees," *Computer Graphics Forum*, Vol. 14, No. 1, pp. 59-66, 1995.
[24] R. Kahler, M. Simon, and H.C. Hege., "Interactive volume rendering of large sparse data sets using adaptive mesh refinement hierarchies," *IEEE Transaction on Visualization and Computer Graphics*, Vol. 9, No. 3, pp. 341-351, 2003.

이 원 종
1999년 인하대학교 전자계산공학과 졸업(공학사). 2001년 연세대학교 컴퓨터과학과 졸업(공학석사). 2001년~현재 연세대학교 대학원 컴퓨터과학과 박사과정. 관심분야는 가시화 시스템, 볼륨 렌더링, 병렬 렌더링, 영상 기반 렌더링, 3차원 그래픽스 하드웨어, 프로그래머블 쉐이더 설계, 모바일 컴퓨팅

박 우 찬
1993년 연세대학교 이과대학 전산과학과 졸업(학사). 1995년 연세대학교 대학원 전산과학과(이학석사). 2000년 연세대학교 공과대학 컴퓨터과학과(공학박사) 2000년~2003년 연세대학교 연구교수 2003년~현재 세종대학교 인터넷 공학과 조교수. 관심분야는 3차원 렌더링 프로세서, 렌더링 시스템, 고성능 컴퓨터 구조, 레이 트레이싱 아키텍쳐

한 탁 돈
1978년 연세대학교 공과대학 전자공학과 졸업(학사). 1983년 Wayne State University 컴퓨터공학(공학석사). 1987년 University of Massachusetts 컴퓨터공학(공학박사). 1987년~1989년 Cleveland 주립대학 조교수. 1989년~현재 연세대학교 공과대학 컴퓨터과학과 정교수. 관심분야는 3차원 그래픽 가속기, Wearable Computer, HCI, ASIC 설계, 고성능 컴퓨터구조