3차원 텍스쳐 기반 볼륨 가시화를 위한 GPU 대역폭 효과적인 렌더링 기법

이원종° 한탁돈
연세대학교 컴퓨터과학과 미디어시스템연구실
airtight°@yonsei.ac.kr
hantack@kurene.yonsei.ac.kr

KISS Korea Computer Congress 2005

Bandwidth-Effective Rendering Scheme for 3D Texture-based Volume Visualization on GPU

Won-Jong Lee° Tack-Don Han
Media System Laboratory, Department of Computer Science, Yonsei University

요 약

본 논문은 3차원 텍스쳐 기반의 볼륨 가시화를 위한 GPU 대역폭에 효과적인 렌더링 기법을 제안한다. 전처리 과정에서 옥트리를 이용하여 원본 볼륨 데이터를 계층적으로 균일한 크기로 분할하여 실제 영역만을 효과적으로 검출하게 되고, 렌더링 시에는 가시순서에 따라 옥트리를 탐색하며 리프 노드의 각 부볼륨을 텍스쳐 매핑 유닛에서 처리하고 블렌딩 유닛에서 이를 합성한다. 작은 크기(16^3 또는 32^3)의 부볼륨 처리는 텍스쳐와 픽셀 캐시의 이용율을 높이고 공백 공간 생략을 가용하게 하여 GPU의 메모리 대역폭을 크게 줄여 렌더링을 가속할 수 있다. 제안하는 기법의 캐시 효율, 메모리 트래픽, 렌더링 시간 등 다양한 실험결과와 성능분석이 제공된다. 실험결과는 제안하는 기법이 전통적인 렌더링 방법에 비해 평균 11배의 대역폭 감소와 3배 빠른 렌더링을 가능하게 하여 GPU를 이용한 볼륨 렌더링에 효과적인 방법임을 보여주었다.

1. 서 론

직접 볼륨 렌더링(direct volume rendering)은 의학, 과학, 공학 등의 다양한 분야에서 시뮬레이션과 수치 해석으로 생성한 3차원 데이터를 가시화하는 강력한 수단으로, 공간상의 각 점(voxel)에 대해 빛의 흡수, 방사에 대한 밀도 값을 할당하고, 가시선를 따라서 시점까지 도달하는 각 점의 빛을 계산하는 방법이다. 특히 최근 GPU(graphic processing unit)는 3차원 텍스쳐 매핑에 대한 하드웨어 가속을 완벽하게 지원하고 있기 때문에 이를 이용한 직접 볼륨 렌더링 기법들이 활발하게 연구되어 왔다[1, 2].

3차원 텍스쳐 기반 볼륨 렌더링은 저비용으로 좋은 품질의 렌더링 결과를 얻는 장점이 있지만, 상대적으로 다음의 단점이 존재한다. 1) GPU의 메모리와 텍스쳐 매핑 유닛간 많은 대역폭을 발생시킨다. 3차원 텍스쳐 매핑의 삼선형 보간(tri-linear interpolation)은 메모리 접근의 지역성(locality)이 높지 않아 2차원 텍스쳐를 위한 텍스쳐 캐시로는 만족할만한 성능을 기대하기 힘들다. 2) 프레임버퍼와 블렌딩 유닛간 많은 대역폭을 발생시킨다. 일반적으로 단면을 구성하는 다각형은 전 화면을 걸치게 되므로 픽셀 캐쉬 효율이 떨어지게 된다. 3) 3차원 텍스쳐 매핑은 단면 순서적 연산이기 때문에 공백 공간 생략(empty space skipping) 방법을 적용하기 어렵다. 4) GPU 파이프라인에 부하 불균형(load imbalance)이 발생한다. 텍스쳐 기반 볼륨 렌더링은 적은 수의 하지만 크기가 큰 프리미티브(primitives)를 사용하기 때문에 레스터라이저는 다각형당 매우 많은 픽셀을 생성하게 된다. 따라서 GPU의 버텍스 쉐이더에 비해 픽셀 쉐이더에 많은 부하가 집중된다.

본 논문은 계층적 자료구조인 옥트리를 이용하여 GPU의 부하 균형을 유지하면서 GPU 캐시 효율을 높여 메모리 대역폭 문제를 완화하는 효과적인 렌더링 기법을 제안한다. 전처리 과정에서, 옥트리을 이용하여 균일한 크기로 부볼륨들로 분할하고, 공백 공간이 생략된 가시적(visible) 부볼륨만을 포함하는 옥트리 계층을 생성한다. 렌더링 단계에서는 생성된 옥트리를 탐색하며 리프 노드들에 할당된 부볼륨들을 가시 순서대로 처리하고, 각각의 결과 영상들을 블렌딩 유닛에서 합성하여 최종 프레임 영상을 생성한다. 본 논문의 주요 공헌(contribution)은 다음과 같다.

1) GPU 메모리 접근의 시, 공간적 지역성를 높이고 GPU 캐시의 효율을 증가시켜 내부 대역폭을 감소시키는 분할 기법 소개
2) 불필요한 메모리 접근과 계산을 방지하는, 각 부볼륨에 대한 공백 공간의 사전 판단 및 렌더링 생략 기법의 응용
3) 부볼륨들이 동일한 크기임을 활용한 저비용의 부볼륨간 가시순서 계산 및 효율적인 텍스쳐 좌표 계산 방법 제안
4) GPU기반 렌더링에 대한 효과적인 하드웨어 시뮬레이션 프레임웍 구축 및 제안

본 논문에서 제안하는 기법의 캐시 효율, 메모리 트래픽, 렌더링 시간 등 다양한 실험이 수행되었다. 전통적인 렌더링 방법에 비해 평균 11배의 대역폭 감소와 3배 빠른 렌더링을 가능하게 하여 제안하는 기법이 GPU기반의 볼륨 렌더링을 위한 효과적인 방법임을 보여주었다. 본 논문의 구성은 다음과 같다. 다음 장에서 관련연구에 관해 살피고, 3장에서 제시하는 렌더링 방법을 기술하고, 4장에서 실험 결과를 보여주며, 5장에서 결론을 맺는다.

2. 관련 연구

분할 렌더링 기법은 대용량의 볼륨 데이터를 처리하기 위해 다양하게 응용되어왔는데, 초기 레이 캐스팅(ray casting) 기반 전용 가속기[3]에서 자체 메모리 구조에 적합하도록 데이터를 저장하는데 사용하였다. TriangleCaster[4]는 기존의 3차원 텍스쳐 매핑기반의 하드웨어에 부가 유닛을 추가하여, 객체의 분할이 아닌 생성될 영상의 분할 방법을 이용하였다. 하지만 합성버퍼 및 다각형 생성기 등 하드웨어 유닛을 별도 제작해야 한다는 단점이 있었다.

볼륨 데이터에 대한 공간을 분할하는 대표적인 예는 옥트리, AMR(adaptive mesh refinement) 트리 등의 계층적 자료구조를 이용하는 방법이다. 이는 구현이 용이하며 LOD에 대한 표현이 가능하다는 특징 때문에 많은 연구에서 사용되었다[5, 6]. 또한, 볼륨 데이터의 크기를 줄이기 위하여 사용된 다양한 압축 기법들[7, 8]에도 각각 적합한 계층적 분할 방법들이 사용되었다.

3. 대역폭 효과적 볼륨 렌더링

제안하는 기법은 옥트리를 생성하는 전처리, 텍스쳐 매핑과 블렌딩을 수행하는 렌더링의 두 가지 단계로 구성된다(그림 1). 본 장에서는 이에 대해 구체적으로 살펴본다.

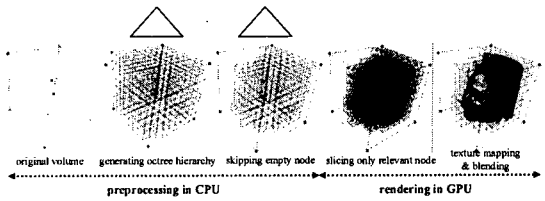


그림 1 옥트리 기반 볼륨 렌더링 기법

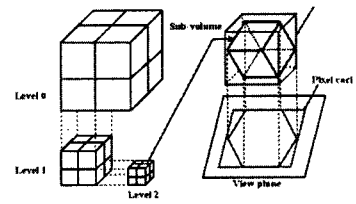


그림 2 부볼륨 크기 결정

3.1 옥트리 생성

제안하는 기법의 첫 단계는 옥트리 계층을 생성하는 전처리 단계이다. 옥트리는 영상 처리나 고체 모델링 분야에서 주로 응용되었고 레이 캐스팅 및 3차원 텍스쳐 기반의 볼륨 렌더링에서도 다양하게 사용[5, 6, 7, 8]되었는데 이는 주로 LOD 및 압축을 위한 것이었다. 본 연구는 이를 이용하여 GPU 캐시 효율을 높여 메모리 대역폭 문제를 해결할 수 있도록 효과적으로 데이터를 분할하는 것에 초점을 맞추었다.

옥트리는 다음과 같이 생성된다. 각 축에 대해 위치상의 중간값을 기준으로 원본볼륨을 분할하며 8개의 부볼륨을 생성한다. 이 과정은 사전 정의된 레벨에 도달하게 될 때까지 재귀적으로 수행된다. 각 부볼륨은 계층의 리프 노드들에 할당되고 이 각 노드들은 부볼륨 내부의 최대/최소값을 저장한다. 이 값들과 전달함수(transfer function)에 정의된 투명도 값을 참조하여 공백 공간은 사전에 검출되어 효과적으로 제거된다.

3.2 옥트리 렌더링

렌더링 단계에서는 재귀적으로 옥트리를 탐색하며, 각 리프 노드의 부볼륨에 대해 3차원 텍스쳐 매핑 및 블렌딩을 수행하게 된다. 옥트리에 의한 분할은 객체 공간(object space)에서 수행되기 때문에 각 보조 영상들은 영상 공간(image space)에서 중첩될 수 있다. 따라서, 부볼륨들은 가시순서(visibility order)에 따라 렌더링되어져야 한다. 이러한 시점-일관적인 가시순서는 루트 노드 아래의 8개의 자식노드와 시점간의 발생가능한 모든 가시순서를 테이블에 저장하고 이를 참조하는 방법을 이용[9]하면 쉽게 구할 수 있다. 일단 8개의 자식 노드들에 대해 가시성에 기준한 방문 순서를 구하게 되면 하위노드들까지 재귀적으로 적용된다.

렌더링을 위해서는 부볼륨과 평행한 단면들과의 교점을 계산해야만 한다. 텍스쳐 매핑을 위해서는 이 교점들의 공간좌표와 텍스쳐 좌표가 계산되어져야 한다. 분할 기반의 볼륨 렌더링은 분할 방법에 상관없이 교점의 위치와 텍스쳐 좌표를 계산하는 오버헤드가 증가하는데, 이러한 문제를 해결하기 위해 [2]에서는 연산을 효과적으로 줄일 수 있는 점진적 계산(incremental computation) 방법을 제안하였다. 하지만, 이 경우에도 부볼륨의 크기가 서로 다르기 때문에 모든 부볼륨에 대해 동일한 계산을 수행해야만 한다.

우리는 교점의 계산을 줄이면서 CPU-GPU간의 트래픽을 줄일 수 있는 새로운 방법을 제안한다. 옥트리에 의해 분할된 부볼륨은 같은 크기이기 때문에 각 부볼륨 간의 간격은 동일한 성질을 이용했다. 가시 순서상 첫 번째 부볼륨에 대해서는 [2]의 방법으로 교점들을 생성한 후 이를 테이블에 저장한다. 이 후의 다른 부볼륨에 대해서는 첫 번째 부볼륨과의 좌표계상의 차이를 테이블에 저장된 교점들에 더하여 빠르게 구해낼 수 있다. 또한, 다각형과 텍스쳐 좌표 값의 비례적인 특징을 이용하였는데, GPU로 전송된 정점좌표를 이용하여 GPU의 버텍스 쉐이더에서 텍스쳐 좌표 계산을 수행함으로 트래픽을 반으로 줄일 수 있었다. 버텍스 쉐이더 내에서는 한 번의 매트릭스 곱셈만으로 쉽게 연산이 가능하다.

3.3 캐시 효율성

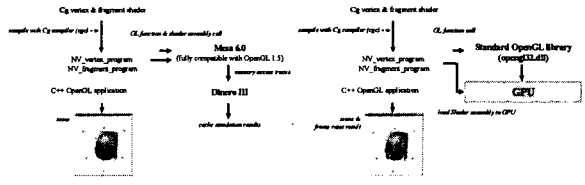
렌더링 성능에 영향을 주는 결정적인 요인은 부볼륨의 크기이다. 우리는 대역폭에 효과적인 렌더링을 위해 부볼륨의 크기가 부볼륨이 텍스쳐 캐쉬에 포함되고 텍스쳐가 매핑될 단면의 크기가 픽셀 캐시의 크기보다 작도록 선택하였다(그림 2). 이는 텍스쳐 캐시와 픽셀 캐시에 대한 접근 지역성을 최대로 보장하게 된다.

제안하는 분할 볼륨 렌더링은 텍스쳐 캐시 효율을 극대화할 수 있다. 3차원 텍스쳐 매핑은 삼선형 보간(tri-linear interpolation)을 수행하기 위해 메모리상의 x-, y-, z-차원 방면의 이웃을 포함한 8개의 텍셀을 읽어오게 된다. 볼륨 데이터가 텍스쳐 메모리로 적재될시, x-축을 기준으로는 연속적으로 저장되고, y-축을 기준으로는 x-축의 텍셀들의 하나의 행만큼 떨어져서 저장된다. z-축을 기준으로의 이웃 텍셀의 지역성은 상대적으로 더 많이 떨어지게 된다, x-축 텍셀들의 하나의 행에 y-축 텍셀들의 하나의 열의 곱만큼 떨어지게 되고, 이 경우 z-축 방향으로의 희박한 시간적 지역성(temporal locality) 때문에, z-축 방면의 이웃 텍셀의 접근 시 캐시에 연속적으로 미스가 발생하여 잦은 캐시 블록의 교환을 발생시키는 쓰레쉬(*thrash*)가 발생된다. 하지만, 부볼륨 렌더링은 쉽게 일반적인 텍스쳐 캐시에 포함되어 처리될 수 있고, 보간 시 중가된 지역성 때문에 캐시 적중률을 높일 수 있게 된다.

마찬가지로, 제안하는 방법으로 픽셀 캐시의 효율도 높일 수 있다. 전통적인 렌더링 방법의 단면은 전 화면에 걸쳐 존재하고, 블렌딩 시 단면상의 각 픽셀에 대해서 한 번의 읽기와 쓰기를 수행한다. 따라서 특정한 같은 픽셀을 다시 방문하기 전에 전 화면에 걸친 연속적인 다른 픽셀들을 처리해야만 하고, 결과적으로 공간적 지역성(spatial locality)이 떨어져 캐시 미스율이 급격히 증가하게 된다. 이에 비해 분할된 부볼륨의 영상들의 크기가 픽셀 캐시의 크기에 포함되도록 설정된다면, 최초의 단면에 대한 강압적 미스(compolsary miss) 이외의 모든 단면들에 대한 캐시 적중이 보장된다. 따라서 한 프레임을 생성하기 위한 프레임 버퍼로의 접근은 부볼륨 수만큼으로 줄어들어 캐시 효율을 높일 수 있게 된다.

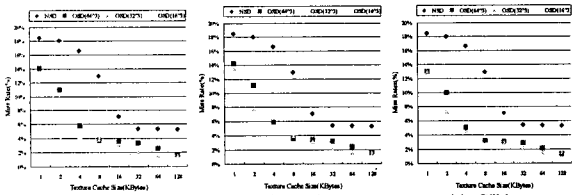
4. 실험 결과

우리는 소프트웨어 시뮬레이션과 하드웨어 가속의 두 가지 방법으로 실험을 수행하였다. 첫째, 캐시 효율 및 메모리 트래픽을 측정하기 위한 효과적인 소프트웨어 시뮬레이션 환경을 구축했다(그림 3a). 이 환경은 세 가지 모듈로 구성된다. 1) *OpenGL, 쉐이더 어셈블리 및 C++로 구현된 옥트리 기반 볼륨 렌더러.* 2) *수정된 Mesa 그래픽 라이브러리.* Mesa[10]는 OpenGL 그래픽 라이브러리를 완벽하게 소프트웨어로 구현한 것으로, 우리는 실행 시 메모리 접근 트레이스를 생성할 수 있도록 Mesa를 수정하였고 이는 표준 OpenGL 라이브러리(opengl32.dll)를 치환하여 1)의 렌더러를 완벽하게 소프트웨어적으로 수행한다. 3) *트레이스-기반 캐시 시뮬레이터인 DINERO III* [11]. 이를 이용해 2)의 Mesa에서 생성된 메모리 접근 트레이스들을 입력받아 캐시 성능이 평가된다. 현재의 GPU는 픽셀 캐시는 컬러와 깊이 캐시가 분리된 형태로 존재하기 때문에 각각에 대해 실험을 수행하였다. 텍스쳐, 컬러 및 깊이 캐시는 모두 fully associative, 블록 크기는 32B로 설정하였고, 1KB에서 128KB 사이의 캐시 크기를 실험에 사용하였다. 둘째, 우리는 제안하는 기법의 렌더링의 성능 향상을 측정하기 위해 실제 GPU 하드웨어 가속 환경에서도 실험을 수행하였다(그림 3b). 1)은 별도



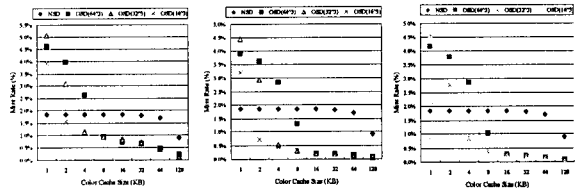
(a) 소프트웨어 시뮬레이션 (b) 하드웨어 가속 실험

그림 3 제안하는 기법의 실험 환경



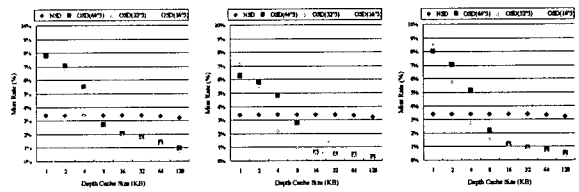
(a) Lobster (b) Engine (c) CTdata

그림 5 NSD와 OSD의 텍스쳐 캐시 미스율 비교



(a) Lobster (b) Engine (c) CTdata

그림 6 NSD와 OSD의 픽셀(컬러) 캐시 미스율 비교



(a) Lobster (b) Engine (c) CTdata

그림 7 NSD와 OSD의 픽셀(깊이) 캐시 미스율 비교

의 수정 없이 그대로 GPU에서 동작된다. 이는 Pentium IV 2.80GHz, 512MB RDRAM 주메모리, AGPx8의 버스, 256MB 그래픽 메모리를 장착한 NVIDIA GeForce FX 5900의 GPU에서 실험되었다.

테스트 데이터 셋은 Lobster(256x256x56), Engine(256x256x110) CTdata(256x256x256)를 사용하였다(그림 8). 해상도는 512^2이며 단면의 갯수는 모두 200장으로 설정하였다. 본 장에서는 전통적 비분할 방법(Non-SubDivided, NSD)과 제안하는 옥트리 기반 분할 방법(Octree-based SubDivided, OSD)의 성능 비교를 기술한다.

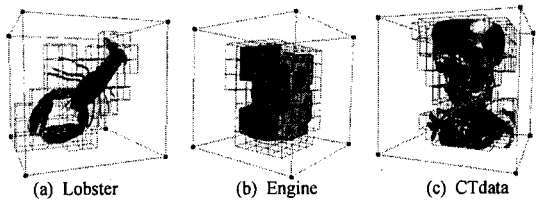
4.1 캐시 효율성

그림 5는 NSD와 OSD에 대한 텍스쳐 캐시의 미스율 결과 비교이다. NSD의 경우 캐시크기가 8KB 이하일 때는 쓰레쉬로 인해 10% 이상의 미스가 발생했고, 그 이상부터 미스율이 급히 줄어들지만 32KB 이상부터는 약 5%의 캐시 미스 이하로는 줄어들지 않았다. 이에 비해 OSD의 경우는 캐시크기가 8KB 이상일 때 세 테스트 데이터 셋 모두 미스율은 4% 이하를 기록했다. 부볼륨이 캐시에 포함되어 초기의 강압적 미스 이외에는 모든 텍셀 접근에 대해 캐시 히트를 기록했다.

그림 6과 그림 7을 보면 컬러과 깊이캐시의 경우도 전체적으로 상대적으로 캐시 지역성 증가에 따라 미스율이 감소함을 알 수 있다. NSD의 경우 단면과 단면 사이의 동일한 위치의 픽셀에 대한 재접근 이전에 현재 단면의 전 화면에 걸친 모든 픽셀을 처리해야 하므로 다른 픽셀에 대한 무수한 메모리 접근이 발생한다. OSD의 경우 상대적으로 지역성이 증가하므로 캐시 크기에 따라 미스율이 감소하게 된다. 부볼륨의 크기가 16^3와 32^3인 경우 컬러 및 깊이 캐시의 크기가 8KB이상일 때 상대적으로 미스율이 크게 감소하고, 그 이상의 경우는 거의 미스율이 0에 가깝게 떨어진다.

4.2 대역폭

우리는 또한 렌더링 시 캐시의 미스로 인해 발생하는 텍스쳐 메모리 및 프레임 버퍼에 대한 대역폭 비교를 수행했다. 총 내부 GPU 대역폭은 텍스쳐 캐시 미스로 인해 텍스쳐 메모리에서 읽은 데이터의 양 + 픽셀 캐시 미스로 인해 프레임 버퍼에서 읽고 쓴 데이터의 양으로 계산된다. OSD의 경우 분할에 따라 추가적으로 정



(a) Lobster (b) Engine (c) CTdata

그림 8 제안하는 기법으로 렌더링된 테스트 볼륨 데이터

점이 생성되어 CPU와 GPU 사이에 부가적인 트래픽이 발생한다.

NSD의 경우 평균 2GB 이상의 트래픽이 발생하였으나, OSD의 경우 공백 영역에 대한 렌더링 생략과, 캐시 적중률 증가에 따라서 대역폭은 캐시 크기에 따라 최소 70MB에서 최대 650MB 정도에 머물렀다. 최상의 경우(Lobester, 캐시 크기 128KB) NSD 에 비해 상대적으로 29배 정도의 대역폭 감소효과를 가져왔고, 평균 11배의 대역폭이 감소하였다. 그에 비해, OSD의 추가적인 정점으로 인해 생성된 트래픽은 최악의 경우에도(Engine, 16^3 부볼륨) 약 8MB 정도 증가하는데 이는 현재의 AGPx8 버스 대역폭(이론상 2GB)에 그리 큰 영향을 주지 않는다.

4.3 렌더링 성능

우리는 제안하는 방법을 통해 얼마만큼의 렌더링 성능향상을 가져왔는가를 알아보기 위해 실제 GPU 상에서도 렌더링 실험을 수행하였다(그림 3b). 표 1는 NSD와 OSD의 렌더링 시의 초당 프레임 비율을 보여준다. 평균적으로 렌더링은 OSD가 NSD에 비해 3.7배 가속되었고, 특히 Lobster의 경우는 데이터의 공백영역이 상당부분 포함되어 있기 때문에 이를 생략하여 상당부분 대역폭을 삭감했기 때문에 최대 6.7배까지 빠른 렌더링을 가능하게 하였다.

표 1 NSD와 OSD에 대한 렌더링 프레임 비율 비교

Data Set	Size	NSD	OSD(32^3)	Speedup	OSD(16^3)	Speedup
Lobster	256x256x56	4.4	25.7	**5.9**	29.1	**6.7**
Engine	256x256x110	4.6	11.1	**2.4**	12.2	**2.6**
CTdata	256x256x256	4.7	10.4	**2.2**	12.5	**2.7**
Average				**3.2**		**3.7**

5. 결 론

본 논문에서는 3차원 텍스쳐 기반의 볼륨렌더링의 기본적인 성능의 한계를 살펴보았고, 이를 해결하는 대역폭에 효과적 볼륨 렌더링을 제안하였다. 이는 GPU 캐시에 적합한 부볼륨 크기 선택과 공백 공간의 효과적인 생략을 활용하여 전통적인 비분할 볼륨렌더링의 대역폭 요구를 대폭 감소시켰다.

본 기법은 적은 크기의 부볼륨을 처리하기 때문에 게임에서 사용되는 실시간 렌더링의 볼류메트릭 라이팅, 화염, 구름과 같은 3차원 볼륨 효과에도 효과적으로 적용 가능할 것으로 예상된다.

참고문헌

[1] A.V.Gelder and K.Kim, "Direct Volume Rendering with Shading via Three-Dimensional Textures," In *Proc. of Volume Visualization 1996*, pp. 23-30, 1996.

[2] W. Li, K. Mueller, and A. Kaufman, "Empty Space Skipping and Occlusion Clipping for Texture-Based Volume Rendering," In *Proc. of Visualization 2003*, pp. 317-324, 2003.

[3] H. Pfister, J. Hardenbergh, J. Knittel, H. Lauer, and L. Seiler, "The VolumePro Real-Time Ray-Casting System," In Proc. of ACM SIGGRAPH, pp. 251-260, August 1999.

[4] G.Knittel, "TriangleCaster: extensions to 3D-texturing units for accelerated volume rendering," In *Proc. of Workshop on Graphics Hardware 1999*, pp. 25-34, 1999.

[5] I.Boada, I.Navazo, and R.Scopigno, "Multiresolution Volume Visualization with a Texture-Based Octree," *The Visual Computer*, Vol. 17, No. 3, pp. 185-197, 2001.

[6] R. Kahler, M. Simon, and H. C. Hege, "Interactive Volume Rendering of Large Sparse Data Sets Using Adaptive Mesh Refinement Hierarchies," IEEE Transactions on Visualization and Computer Graphics, Vol. 9, No. 3, pp. 341-351, 2003.

[7] S.Guthe, M.Wand, J.Gonser, and W.Straßer, "Interactive Rendering of Large Volume Data Sets," In *Proc. of Visualization 2002*, pp. 53-60, 2002.

[8] J. Schneider, and R. Westermann, "Compression Domain Volume Rendering," In *Proc. of Visualization 2003*, pp. 293-300, 2003.

[9] 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.

[10] *Mesa Graphic Library*, http://www.mesa3d.org

[11] *Dinero III*, http://www.cs.wisc.edu/~larus/warts.html