브릭 정점을 이용한 GPU 기반 볼륨 광선투사법 가속화

채수평[O] 신병석

인하대학교 컴퓨터 정보공학과

cotnvud@naver.com, bsshin@inha.ac.kr

Accelerating GPU-based Volume Ray-casting Using Brick Vertex

Supyeong Chae[O] Byeong-Seok Shin

Department of Computer Science & Information Engineering, Inha University.

요 약

최근에 GPU기반의 볼륨 광선 투사법을 가속화하는 기법들이 많이 연구되고 있다. 하지만 이런 기법들은 CPU-GPU간 데이터 전송 시 병목 현상을 야기하고 계층구조를 표현하기 위한 추가적인 GPU 메모리 공간이 필요할 뿐만 아니라 불투명도 전이 함수가 변경되었을 때 실시간에 대응하지 못하는 문제점들이 발생할 수 있다. 본 논문에서는 이러한 문제점들을 해결하기 위해 GPU기반의 효율적인 빈 공간 도약 기법을 제안한다. 브릭(brick)안에 포함되는 복셀들의 최대 밀도 값을 하나의 정점에 저장하고 불투명도 전이 함수에 의하여 투명하다고 판별된 정점들을 기하 쉐이더에서 삭제한다. 이 정점들을 렌더링 시간에 기하 쉐이더의 입력 값으로 사용해 투명하지 않은 영역의 바운딩 박스를 만들어 광선이 효과적으로 진행하도록 한다. 생성된 정점들은 렌더링 중에 시점의 변화에 무관하게 사용할 수 있지만 불투명도 전이 함수가 변경되면 투명하지 않은 정점들을 다시 생성해야 한다. 이는 기하 쉐이더를 통해서 GPU안에서 고속으로 생성되기 때문에 대화식 처리가 가능하다. 제안하는 방법은 기존 광선 투사법의 결과와 동일한 영상을 생성하며 렌더링 속도는 기존의 방법에 비해 최대 10배 이상 향상되었다.

Abstract

Recently, various researches have been proposed to accelerate GPU-based volume ray-casting. However, those researches may cause several problems such as bottleneck of data transmission between CPU and GPU, requirement of additional video memory for hierarchical structure and increase of processing time whenever opacity transfer function changes. In this paper, we propose an efficient GPU-based empty space skipping technique to solve these problems. We store maximum density in a brick of volume dataset on a vertex element. Then we delete vertices regarded as transparent one by opacity transfer function in geometry shader. Remaining vertices are used to generate bounding boxes of non-transparent area that helps the ray to traverse efficiently. Although these vertices are independent on viewing condition they need to be reproduced when opacity transfer function changes. Our technique provides fast generation of opaque vertices for interactive processing since the generation stage of the opaque vertices is running in GPU pipeline. The rendering results of our algorithm are identical to the that of general GPU ray-casting, but the performance can be up to more than 10 times faster.

키워드: 볼륨 렌더링, 광선 투사법, 빈 공간 도약, 기하 쉐이더

Keywords: volume rendering, ray-casting, empty space skipping, geometry shader

투고일 : 2011.05.11 / 심사일 : 2차 2011.05.30 / 게재확정일 : 2011.07.19

1. 서론

볼륨 렌더링(volume rendering)은 3차원 볼륨 데이터를 가시화하기 위한 도구이다. 하지만 일반적으로 볼륨 데이터는 방대한 크기를 갖기 때문에 렌더링 시간이 오래 걸리는 단점이 있다. 직접 볼륨 렌더링 방법인 볼륨 광선 투사법(volume ray-casting)[1]이 가장 많이 사용된다. 볼륨 광선 투사법은 각 픽셀에서 광선을 발사하여 광선이 진행하는 구간의 볼륨 데이터를 샘플링하고 누적하여 결과영상을 만들어내는 방법이다.

최근 연산 능력이 매우 커진 그래픽 하드웨어는 3차원 이산 데이터를 기존의 방법에 비해 빠르게 가시화 할 수 있다. GPU를 활용한 볼륨 렌더링의 대표적인 방법인 GPU 볼륨 광선 투사법은 기존 CPU기반 방식을 프로그래밍 가능한 GPU의 장점을 이용하여 프레그먼트 쉐이더(fragment shader)에서 쓰레드마다 관측 광선을 투사하여 각 픽셀의 색상 값을 계산한다[2]. 볼륨 광선 투사법은 볼륨 렌더링 기법 중 가장 좋은 화질의 영상을 제공하지만 광선을 따라서 샘플링과 누적 연산이 많이 수행되므로 GPU를 활용하더라도 볼륨 데이터의 크기가 커지면 만족할 만한 속도를 기대하기 힘들다.

보편적으로 광선 투사법의 렌더링 속도를 향상시키기 위해서 옥트리와 같은 트리구조를 이용하여 볼륨 데이터의 빈 공간을 제거하는 방법이나 광선의 진행 단계에서 누적되는 불투명도 값에 제한을 두고 정해진 값에 도달하면 광선의 진행을 조기에 종료하는 방법이 사용되었다. 이 중에서 볼륨 데이터의 빈 공간을 제거하는 방법은 불투명도 전이함수(OTF: Opacity transfer function)에 의해 특정 복셀이 최종 영상에 기여하는 투명하지 않은 복셀인지를 판별하고 광선이 투명하지 않은 복셀들이 있는 부분으로만 진행되도록 하기 위해 기하구조나 계층구조를 필요로 한다. 따라서 렌더링 중에 OTF가 변하는 경우 이러한 기하구조나 계층구조를 재생성해야 한다. 비록 빈 공간이 제거된 데이터라 하더라도 CPU-GPU간의 데이터 전송 시에 많은 시간이 소요되므로 대화식 렌더링이 어렵게 된다.

본 논문에서는 이와 같은 문제점을 해결하기 위하여 기하 쉐이더(geometry shader)를 이용한 효과적인 빈 공간 도약(empty space skipping) 기법을 제안한다. 기하 쉐이더는 GPU 파이프라인 안에서 입력된 정점을 추가, 삭제, 수정할 수 있고, 처리한 데이터를 프레그먼트 쉐이더로 바로 전달하거나 정점 버퍼로 스트림 아웃(stream-out)하여 사용하는 것이 가능하다[3]. 기하 쉐이더의 스트림 아웃 기능을 이용하면 생성된 정점버퍼를 메인 메모리에서 가져올 필요가 없으므로 기존 방법들에서 발생했던 CPU와 GPU간의 병목현상이 발생하지 않는다. 본 논문에서는 볼륨 데이터를 사용자가 정한 크기의 브릭(brick)들로 나누고 특정 브릭 안에 포함된 모든 복셀들을 검사하여 최대 밀도 값을 구한 후 이것을 저장하는 브릭 정점(brick vertex)을 생성한다. 생성된 정점들은 기하 쉐이더의 입력 값이 되며 OTF를 적용하여 이중 투명하지 않은 정점들만을 생성한다. 생성된 투명하지 않은 정점들은 기하 쉐이더를 통해 바운딩 박스(bounding box)를 생성하고, 생성된 바운딩 박스들로만 광선이 진행되도록 한다.

제안한 방법은 빈 공간에 대한 샘플링을 최소화함으로써 기존 광선 투사법과 비교하여 효율적인 처리가 가능하다. GPU 파이프라인 안에서 투명하지 않은 브릭 정점들이 생성되므로 GPU와 CPU간의 병목현상이 없고, SIMD 병렬처리기능으로 고속 처리된다. 이처럼 빈 공간이 제거된 상태로 광선을 투사하면 샘플링 횟수가 줄어들어 각 샘플점마다 추가 연산이 필요한 법선벡터 계산이나 볼륨 데이터 대한 필터링 연산 시에 특히 유리하다.

2장에서는 GPU기반의 광선 투사법을 가속화하기 위한 방법 중 빈 공간을 제거하기 위해 기존에 제안된 방법들에 대해 살펴본다. 3장에서는 본 논문에서 제안한 기법에 대해 자세히 기술하고, 4장에서 실험 결과를 분석하고 5장에서 결론을 맺는다.

2. 관련 연구

GPU기반 볼륨 광선 투사법은 SIMD 병렬처리를 이용하여 고속 볼륨 가시화가 가능하다. 광선 투사법의 가속화 방법은 크게 두 가지로 분류된다. 샘플링이 필요 없는 빈 공간을 미리 계산하여 속도를 개선하는 빈 공간 도약 방법과 추적 광선의 종료 조건을 완화하여 렌더링 시간을 줄이는 광선 조기 종료(early ray termination)방법이다. 빈 공간 도약을 통한 가속화 기법은 육면체의 바운딩 박스를 깊이 텍스처(depth texture)에 투영하여 광선의 진행 방향과 시작점 그리고 종료 지점을 계산한다. 하지만 단순한 바운딩 박스를 이용하는 경우에는 빈 공간이 많은 큰 볼륨 데이터를 가시화 할 때 매우 비효율적이다. 이를 해결하기 위해 Hong은 옥트리를 이용한 효과적인 빈 공간 도약 기법을 제안하였다[4]. 이 방법은 볼륨을 일정 단위의 브릭으로 나누고 시점에 따라 옥트리를 생성하여 브릭들을 정렬한다. 정렬된 브릭들을 개체 순서(object

order)로 순회하면서 각각 4패스에 걸쳐 특정 브릭이 투명하지 않은지의 여부와 조기 광선 종료 가능성을 검사한다. 조건에 해당되는 브릭들에 대해서만 광선 투사법을 진행한 후 깊이 순서에 따라 누적시킨다. 이 방법은 각 브릭 당 4번의 렌더링 패스와 3번의 컨텍스트 스위칭(context switching)이 발생하여 렌더링 속도가 느리다. 또한 이 방법은 시점이 변경되면 브릭들을 다시 생성해야 하는 시점 종속적(view dependent)방법이라는 단점이 있다. 이와 유사한 자료구조 기반의 가속화 기법으로 Vincent는 kd-트리 기반의 광선 투사법을 제안하였다[5]. 이것은 전처리 단계에서 OTF를 참조하여 kd-트리구조를 생성하여 투명하지 않은 노드들만을 미리 걸러내는 방법이다. 하지만, 이 방법은 전처리 과정의 오버헤드 때문에 실시간 처리가 불가능하고 OTF가 빈번히 변하는 경우에는 적합하지 않다. 또한, kd-트리를 텍스처에 저장해야 하기 때문에 추가적인 메모리 사용이 불가피하다. Liu는 임의의 구형 기하 구조체를 사용한 가속화 기법을 제안하였다[6]. 이 방법은 블록(block)단위의 볼륨 데이터 구조를 2D텍스처에 저장하고 히스토 피라미드(Histo pyramid)라는 밉-맵(mip-map) 기반의 4진 트리(quard-tree)를 사용하여 그리고자 하는 유효 블록의 인덱스를 계산하는 방법이다[7]. 이 방법은 3D 볼륨 텍스처 외에도 블록 구조에 대한 3D텍스처 한 장과 히스토 피라미드를 저장하기 위해 추가된 3D텍스처 크기의 약 2배 크기에 해당되는 2D텍스처를 생성 하며, 유효 정점 수만큼의 2D 텍스처를 추가로 생성해야 한다. 그 후에 구형 기하 구조체를 그리기 위한 정점 버퍼를 생성한다. 이는 크기가 방대한 볼륨 데이터의 처리에서 GPU메모리의 낭비가 매우 크기 때문에 대용량 볼륨 렌더링에 적합하지 않다는 단점이 있다. 또한, 정점버퍼를 생성하는 단계가 메인 메모리에서 수행되기 때문에 생성된 정점데이터를 GPU로 전송하는 시간이 오래 걸린다.

최근에는 기하 쉐이더를 사용하여 빈 공간 도약 볼륨 광선 투사법에 대한 가속화 연구가 진행 되었다[8, 9]. Mensmann은 차폐절두체(occlusion frustum)라는 기하구조를 사용하였다[8]. 이전의 렌더링 결과에 의존하여 광선의 시작점 깊이를 얻고 이 시작점을 사용하여 기하 쉐이더에서 차폐절두체를 생성한다. 따라서 Mensmann의 방법은 광선의 시작점 깊이 값만을 사용하여 빈 공간을 제거하므로 효율적인 빈 공간 제거가 이루어지지 않으며 OTF가 변화 하면 시작점 깊이 값이 무의미 하므로 최소한 한번은 광선이 가장 기본적인 바운딩 박스 시작점 위치에서 투사되어야 한다. 이러한 성질 때문에 대화식 처리에 부적합 하다. Tatarchunk은 기하 쉐이더를 이용하여 마칭 큐브(marching cube)를 이용한 볼륨 데이터의 등위 표면(iso-surface) 생성을 가속하는 방법을 제안하였다[9]. 기존의 마칭 큐브[10]기법과 유사하지만 사면체를 사용하여 단순화시킨 사면체 기반 마칭 큐브[11]기법을 이용하여 등위 표면을 실시간으로 생성한다. 모든 복셀을 정점 데이터로 변환하면 투영 시에 구멍(hole)이 생긴다. 이를 해결하기 위해 정점들의 집합을 입력받아 기하 쉐이더에서 사면체로 이루어진 등위표면을 생성한다. 모든 복셀에서 등위 표면을 생성하고 이것을 바운딩 박스로 사용하여 광선 투사법을 진행하기 때문에 상당히 효율적인 빈 공간 도약이 가능하다. 하지만 마칭 큐브 방법을 사용하기 위해서 복셀들 각각을 정점으로 사용해야 하므로 큰 볼륨 데이터 처리에는 부적합하다. 또한 등위 표면 생성에도 많은 계산이 필요하기 때문에 OTF가 변경되는 경우는 대화식 처리가 어려운 단점이 있다.

3. 브릭 정점을 이용한 광선 투사법

제안하는 방법은 그림 1과 같이 전처리 단계와 렌더링 단계로 구성되며 처음 브릭 정점을 구성하는 전처리 단계를 제외하고 모든 단계는 GPU에서 수행된다. 전처리 단계에서는 복셀들을 사용자가 정한 크기의 브릭으로 나누어 정점으로 저장한다. 정점 집합은 GPU의 기하 쉐이더에서 OTF에 따라 투명하지 않은 복셀들을 대표하는 정점인지 판별하고 이를 스트림 아웃하여 유효한 정점 버퍼를 만든다. 이 정점 버퍼를 다시 입력받아 기하 쉐이더에서 큐브 형태로 변환 후, 래스터라이즈(rasterise)한다. 그 결과 시점을 기준으로 가장 가까이 있는 바운딩 박스 면들의 깊이 값과 가장 멀리 있는 바운딩 박스 면들의 깊이 값이 저장된 텍스처가 만들어 지고 광선 투사법 단계에서는 빈 공간이 아닌 영역에 대해서만 기존 방법을 적용하여 렌더링 한다.

3.1. 전처리 단계

전처리 단계에서는 렌더링 단계에서 쓰일 데이터를 만든다. 볼륨 데이터를 기하 쉐이더의 입력 값으로 사용하기 위해서는 정점의 형태로 가공한다. 볼륨 데이터를 사용자가 정한 크기의 브릭으로 나누어 해당 브릭 영역을 대표하는 정점 한 개(브릭 정점)로 저장한다. 브릭 정점은 브릭 안에 포함된 모든 복셀들의 밀도 값 중 최대 값 D_{max}를 저장한다. 브릭 정점의 x, y, z 필드에는 브릭 정점의 전역 좌표가, w 필드에는 D_{max}값이 저장된다.

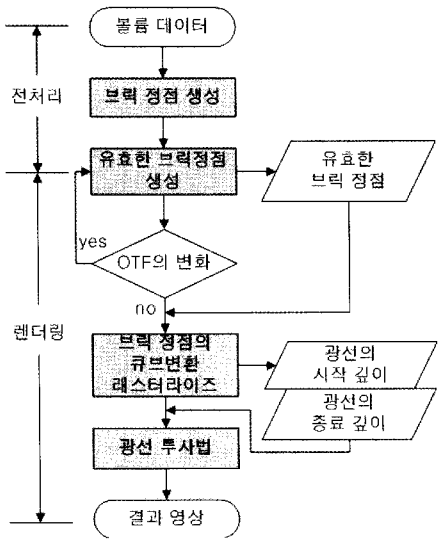

그림 1 본 논문에서 제안하는 방법의 처리 순서도. 전처리 단계와 렌더링 단계로 나누어진다.

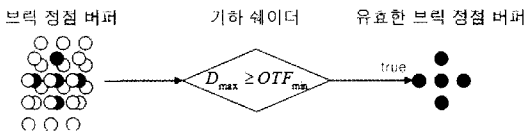

그림 2 기하 쉐이더를 이용한 유효 정점 버퍼 생성.

OTF에 의해 투명한 복셀로 정해지면 최종 픽셀값에 영향을 주지 않으므로 샘플링 할 필요가 없다. 따라서 전처리 단계에서 생성한 브릭 정점 중에서 D_{max}값이 투명 영역에 해당되면 그 브릭들을 제거한다. GPU의 파이프라인에서 기하 쉐이더를 이용한 스트림 아웃은 프레그먼트 쉐이더를 거치지 않고 바로 정점 버퍼의 생성이 가능하다. 브릭이 투명한지의 여부를 판별하는 것은 투명하지 않은 복셀의 최소 밀도 값 OTF_{min}을 기준으로 한다. 브릭 정점의 D_{max}값이 OTF_{min}보다 크거나 같은 경우 유효한 브릭 정점(valid brick vertex)이라고 판단하고 기하 쉐이더의 스트림 아웃기능을 이용하여 정점 버퍼에 저장한다(그림 2). 이 과정은 OTF가 변경 될 때마다 한 번씩만 수행되는데 병렬처리가 가능하므로 고속 수행되어 렌더링 시간에는 거의 영향을 주지 않는다. 또한 GPU 파이프라인 안에서 수행되므로 메인 메모리와 정보 교환을 필요로 하지 않기 때문에 GPU와 CPU사이의 병목 현상이 발생하지 않는다.

3.2. 렌더링 단계

3.2.1. 바운딩 박스 생성

볼륨 광선 투사법을 수행하기 위해서 광선이 샘플링을 시작하는 위치의 깊이 값(시작 깊이)과 종료되는 위치의 깊이 값(종료 깊이)를 알아야 한다. 전처리 단계에서 얻어낸 유효한 브릭 정점 각각은 기하 쉐이더를 통과하면서 원래의 브릭 크기에 해당하는 육면체의 정점들이 추가로 생성된다. 이 육면체들은 볼륨 데이터에서 OTF에 의해 불투명하다고 판단된 데이터들의 영역이므로 유효한 데이터 영역을 표현하는 바운딩 박스가 된다. 이렇게 생성된 육면체는 전처리 단계와 다르게 스트림 아웃하지 않고 바로 깊이 테스트(depth test)와 선별과정(culling)을 거쳐 프레그먼트 쉐이더에서 래스터라이즈 된다. 광선 투사법을 수행하기 위해 필요한 정보는 진행될 광선의 시작 깊이 값과 종료 깊이 값에 대한 정보이다. 이는 시점으로부터 해당 화소에 그려질 바운딩 박스의 가장 가까운 면의 깊이 값(nearest depth)과 가장 멀리 있는 면의 깊이 값(farthest depth)을 통해 얻을 수 있다.

유효한 바운딩 박스에서 시점으로부터 가장 멀리 있는 면의 깊이 값을 얻기 위해 큐브들을 깊이 값에 따라 정렬하거나 깊이 필링(depth peeling)을 하는 것은 비효율적이다. 본 논문에서는 이러한 과정을 간단하게 하는 방법을 제안한다. 스크린 좌표계에서 z값을 반전시켜 래스터라이즈하면 가장 후면의 깊이 값이 가장 가까운 값으로 바뀌기 때문에 바운딩 박스들 중에서 시점으로부터 가장 멀리 있는 면에 해당하는 깊이 값을 구할 수 있다. 이 과정은 깊이 테스트가 시작되기 전 단계인 기하 쉐이더 단계에서 육면체에 해당하는 정점을 생성할 때 계산된다.

3.2.2. 광선 투사

GPU기반의 광선 투사법은 바운딩 박스에서 광선의 시작 깊이 값과 종료 깊이 값을 얻고 광선의 진행 방향을 결정한다. 가장 기본적인 GPU기반 광선 투사법은 하나의 바운딩 박스를 사용하여 광선을 진행하는 것이다[2]. 본 논문의 방법은 3.2.1에서 얻은 유효한 바운딩 박스들로부터 빈 공간이 제거된 상태의 광선 시작 깊이와 종료 깊이를 이용하여 광선 투사법을 진행한다. 기본적인 GPU기반 광선 투사법에서 광선은 그림 3의 $\overline{AD}$와 같이 진행된다. 하나의 바운딩 박스만을 사용하기 때문에 광선의 진행 경로에 유효하지 않은 값들이 샘플링 된다. 제안하는 방법에서 광선은 $\overline{BC}$로 진행되며 빈 공간을 생

략하고 시작점 B에서부터 종료점 C까지만 광선이 진행되게 된다. 빈 공간이 제거된 광선의 시작 깊이 값과 종료 깊이 값은 기하 쉐이더를 통해 렌더링 시간에 생성되어 그려지는 유효한 브릭 정점 바운딩 박스들의 깊이 값 텍스처를 이용한다.

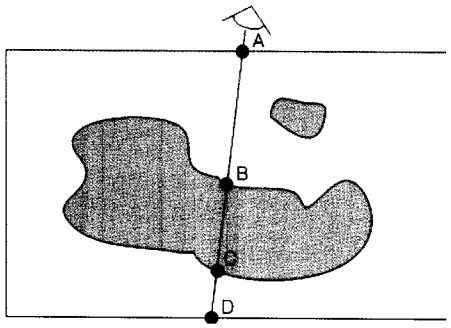

그림 3 기존 방법과 제안하는 방법에서의 광선 진행경로 비교. 외곽은 기본 광선 투사법의 바운딩 박스, 짙은 음영 구역은 볼륨 데이터, 옅은 음영 구역은 빈 공간이 제거된 바운딩 박스이다. $\overline{AD}$는 기본 광선 투사법에서 광선의 진행경로이고 $\overline{BC}$는 본 논문에서 제안한 방법 광선 진행 경로를 나타낸다.

4. 실험 결과

실험에 사용한 방법들은 3GHz의 AMD Phenom II X2 CPU와 1GB의 비디오 메모리를 가진 ATI Radeon HD 5870 그래픽 카드가 장착된 PC에서 실행하였다. DirectX 11을 사용하였으며 결과영상의 해상도는 1024×1024이다. 본 논문에서 제안한 방법이 적용된 광선 투사법과 비교 대상인 기존의 기본적인 광선 투사법에는 광선이 진행하며 누적하는 값의 알파 값이 특정한 경계 값에 도달하면 광선의 진행을 종료하는 조기 광선 종료 알고리즘[1]이 적용되어 있다.

제안하는 방법에서는 한 장의 3D 볼륨 텍스처, 유효한 브릭들을 감싸는 바운딩 박스의 광선 시작 깊이값과 종료 깊이값이 저장된 2D텍스처 두 장, 그리고 OTF를 위한 1D 텍스처 한 장이 사용된다. 또한 브릭 정점과 유효 브릭 정점을 위한 저장하기 위해 두 개의 정점 버퍼를 사용한다.

표 1은 몇 개의 볼륨 데이터를 다양한 크기의 브릭으로 테스트 한 결과이다. 본 논문에서는 2^3, 4^3, 8^3, 16^3, 32^3의 브릭들을 사용하여 실험하였다. 표 1에 대한 결과 영상은 그림 4(a)와 그림5에 나타나 있다.

표 1. 기본적인 광선투사법과 제안한 방법의 속도비교

실험용 볼륨 데이터	크기 (voxel)	브릭 크기 (voxel)	기본적인 광선 투사법(A) 의속도 (fps)	제안한 방법(B)의 속도 (fps)	배수 (A/B)
Head	256^3	2^3	96	27	0.28
		4^3		119	1.24
		8^3		204	2.125
		16^3		209	2.177
		32^3		168	1.75
Foot	512^3	2^3	39	10	0.26
		4^3		42	1.08
		8^3		104	2.67
		16^3		130	3.33
		32^3		113	2.89
Engine (interior)	512^3	2^3	32	84	2.62
		4^3		279	8.72
		8^3		422	13.19
		16^3		402	12.56
		32^3		302	9.44
Engine	512^3	2^3	41	10	0.24
		4^3		63	1.54
		8^3		218	5.32
		16^3		301	7.34
		32^3		226	5.51

표 2. OTF변경에 따른 유효한 브릭 정점 재생성 시간(512^3 볼륨)

브릭크기 (voxel)	2^3	4^3	8^3	16^3	32^3
시간 (*msec*)	25	0.377	0.32	0.315	0.309

Engine(interior) 볼륨 데이터(512^3)의 경우 브릭의 크기가 8^3일 때 하나의 바운딩 박스를 사용한 기본적인 GPU 광선 투사법과 비교해 13.19배의 속도차이를 보이며 Foot 데이터(512^3)의 경우 브릭의 크기가 16^3일 때 3.33배의 속도 증가 효과를 얻었다. 같은 크기의 데이터지만 이러한 속도 차이가 발생하는 것은 빈 공간이 많을수록 광선의 진행 구간이 짧아져 속도가 빨라지기 때문이다. 브릭의 크기가 작을수록, 래스터라이즈되는 삼각형 매쉬가 많아지기 때문에 깊이 비교와 선별작업에 걸리는 속도가 증가하게 된다. 따라서 적당한 브릭의 크기를 선택하여 사용하는 것이 중요하다. 표 1에 의하면 일반적으로 브릭의 크기가 8^3이나 16^3 인 경우에 가장 효과가 좋은 것을 알 수 있다.

그림 4의 위쪽 이미지와 가운데 이미지는 각각 광선의 시작점 깊이와 종료점 깊이로 얻은 볼륨 텍스처 좌표를

RGB 색상 값으로 나타낸 것이다. 그림 4의 (b)는 기본적인 볼륨 광선 투사법의 진행 과정을 나타낸다. 하나의 바운딩 박스를 사용하기 때문에 그림 3에서의 광선 진행 선분 $\overline{AD}$처럼 의미 없는 볼륨 공간에서도 광선이 진행하게 되어 불필요한 샘플링을 많이 한다. (a)는 제안된 방법에서 광선의 진행 과정을 나타낸다. 빈 공간이 제거된 유효한 바운딩 박스들을 생성하여 광선의 시작 깊이와 종료 깊이가 실제 결과에 영향을 주는 복셀들에 맞게 만들어진다. 따라서 광선이 진행하는 구간이 줄어들게 되어 전체 렌더링 프로세스가 가속화 된다.

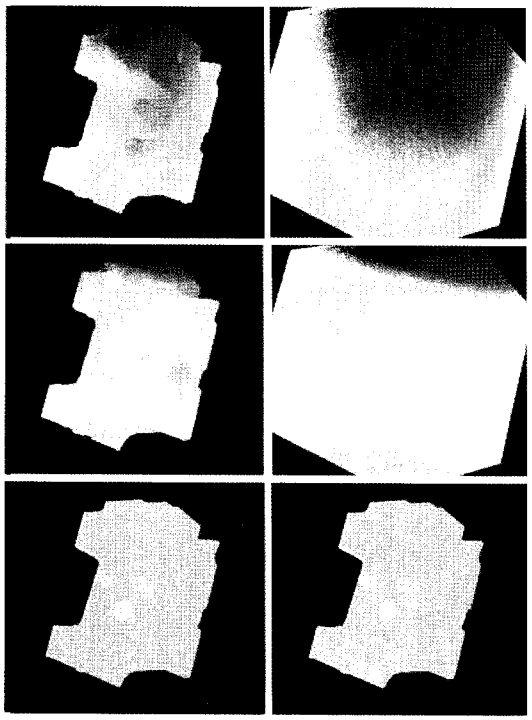

그림 4 (a) 제안된 방법의 광선 진행과 실험 결과(4^3브릭 크기, 62 fps), (b) 기본적인 광선 투사법의 광선 진행과 실험 결과. 위에서부터 광선의 시작 깊이, 광선의 종료깊이, 결과영상. (a)와 (b)를 적용하여 렌더링한 결과 이미지는 같다.

표 2는 Engine 볼륨 데이터(512^3)을 사용했을 때 브릭 크기에 따라서 OTF가 변경될 때 유효 브릭 정점을 재생성하는 시간을 나타낸다. 2^3의 브릭 크기를 사용하였을 때 브릭 정점 버퍼로부터 유효한 브릭 정점을 추출하여 정점 버퍼에 저장하는데 소요되는 시간은 25*ms*이고, 이것 보다 큰 브릭들은 약 0.3*ms*가 소요된다. 2^3브릭 크기를 사용하면 GPU에서 유효한 브릭 정점을 추출하기 위해 처리해야 할 정점은 256^3개가 된다. 이는 실험에서 사용한 GPU의 허용 쓰레드 한계를 초과하기 때문에 4^3이상의 브릭을 사용했을 때보다 훨씬 더 많은 시간이 걸린다. 실험 결과에 의하면 렌더링 중에 OTF가 변경되어도 렌더링 속도에 영향을 줄 만큼 큰 시간이 걸리지 않는다는 것을 알 수 있다. 특히 4^3이상의 브릭 크기를 사용한다면 속도 저하가 거의 없다. 이것은 브릭 정점이 기하 쉐이더에서 사용하기에 적합한 형태이므로 유효한 브릭 정점을 생성하는 작업에 매우 적은 처리 시간만이 소요되기 때문이다.

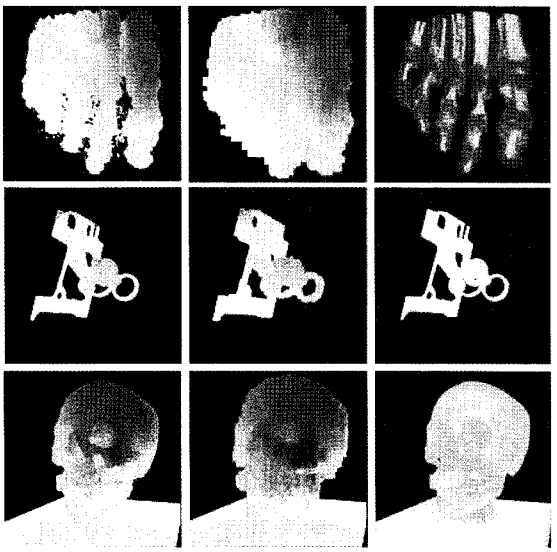

그림 5 (상) 좌측부터 표 1의 Foot 데이터에 브릭 크기를 4^3, 16^3로 했을 때의 결과, (중) Engine(interior) 데이터에 브릭 크기를 2^3, 8^3로 했을 때의 결과, (하) Head 데이터에 브릭 크기를 2^3, 8^3로 했을 때의 결과.

5. 결론

본 논문에서는 렌더링 속도를 향상시키기 위해 모든 과정이 GPU를 통해 수행되는 빈 공간 도약 기법을 제안하였다. 기존의 제안된 빈 공간 도약 기법들은 계층 구조에 의해 추가적인 텍스처 정보를 필요로 하지만 본 논문에서 제안된 방법은 계층 구조를 위한 텍스처를 필요로 하지 않는다. OTF에 변화가 있을 때, 새로 생성하는 유효한 정점들이 GPU 파이프라인 안에서 병렬로 처리되기

때문에 속도가 빠르며 기존의 방법들처럼 CPU와 GPU사이의 병목 현상을 유발하지 않는다. 따라서 대화식 속도의 실시간 볼륨 렌더링이 가능하다.

감사의 글

이 논문은 2011년도 정부(교육과학기술부)의 재원으로 한국 연구재단의 지원을 받아 수행된 연구임(No. 2011-0015779).

참고 문헌

[1] K. Engel, D. Weiskopf, C. Rezk-salama, J. Kniss, and M. Hadwiger, "Real-time volume graphics," AK Peters, 2006.

[2] J. Kruger, R. Westermann, "Acceleration tech-niques for GPU-based volume rendering," *In Pro-ceedings IEEE Visualization*, pp. 287-292, 2003.

[3] S. Patidar, S. Bhattacharjee, J. Singh, and P. Narayanan, "Exploiting the shader model 4.0 architecture," *Technical Report IIIT Hyderabad*, 2006.

[4] W. Hong, F. Qiu, and A. Kaufman, "GPU-based object-order ray-casting for large datasets," *In Volume Graphics*, pp. 177-185, 2005.

[5] V. Vidal, X. Mei, and P. Decaudin, "Simple empty-space removal for interactive volume rendering," *Journal of Graphics, GPU, and Game Tools*, 13:2, 21-36, 2008.

[6] B. Liu, G. J. Clapworthy, and F. Dong, "Accelerating volume raycasting using proxy spheres," *Computer Graphics Forum*, Volume 28, Issue 3, pp. 839-846, 2009.

[7] G. Ziegler, A. Tevs, C. Theobalt, and H. P. Seidel, "On-the-fly point clouds through histogram pyramids," *Vision, modeling, and visual-ization : proceedings*, November 22-24, 2006.

[8] J. Mensmann, T. Ropinski, and K. Hinrichs, "Accelerating volume raycasting using occlusion frustums," *In IEEE/EG International Symposium on Volume and Point-Based Graphics*, pp. 147-154, 2008.

[9] N. Tatarchunk, J. Shopf, and C. DeCoro, "Advanced interactive medical visualization on the GPU," *Journal of Parallel and Distributed Computing*, Volume 68, Issue 10, pp. 1319-1328, 2008.

[10] W. E. Lorensen, and H. E. Cline, "Marching cube: a high resolution 3D surface construction algorithm," *ACM SIGGRAPH Computer Graphics(Proceedings of SIGGRAPH 87)*, Volume 21, Issue 4, pp. 163-169, 1987.

[11] P. Shirley, and A. Tuchman, "A polygonal approximation to direct scalar volume rendering," *SIGGRAPH Computer Graphics*, Volume 24, Issue 5, pp. 63-70, 1990.

〈저자소개〉

채수평
- 2010년 인하대학교 컴퓨터공학부 학사
- 2010년~현재 인하대학교 컴퓨터공학부 석사
- 관심분야 : 볼륨 그래픽스, 실시간 렌더링

신병석
- 1990년 서울대학교 컴퓨터공학과 학사
- 1992년 서울대학교 컴퓨터공학과 석사
- 1997년 서울대학교 컴퓨터공학과 박사
- 2000년~현재 인하대학교 컴퓨터정보공학과 부교수
- 관심분야 : 볼륨 그래픽스, 차세대 컴퓨팅, 실시간 렌더링