Journal of Korea Multimedia Society Vol. 14, No. 8, August 2011(pp. 981-991)

GPGPU 환경에서 최대휘소투영 렌더링의 고속화 방법

계희원[†], 김준호[††]

요 약

최대휘소투영은 볼륨 렌더링의 한 기법으로, 의료영상을 판독하기 위해서 중요한 기능이다. 광선 투사법을 이용한 최대휘소투영 렌더링은 비교적 높은 화질의 영상을 생성하나 많은 연산을 요구한다. 본 연구는 그래픽 처리장치(GPU : Graphic Process Unit)에 일반 연산을 적용하는 GPGPU(General-purpose computing on Graphic Process Unit) 기술을 이용하여 최대휘소투영 렌더링의 속도를 향상시키는 방법에 관한 연구를 수행한다. 본 논문에서는 GPGPU를 수행 할 수 있는 프로그래밍 언어인 CUDA(an acronym for Compute Unified Device Architecture)를 기반으로 고속 광선 투사법을 구현하며, CUDA 환경에 적합한 가속화 방법을 제안한다. 구체적으로, 블록 기반 공간 도약 기법을 적용하여 불필요한 부분을 도약하고, 이분 이동법을 통해 블록 경계면의 탐색을 고속으로 수행하며, 초기 값 추정 알고리즘을 이용하여 공간 도약 확률을 향상시킨다. 이를 통해 화질 손실 없이 최대휘소투영 렌더링의 가시화 속도를 크게 향상시킨다.

Acceleration techniques for GPGPU-based Maxmum Intensity Projection

Heewon Kye[†], Jun-Ho Kim[††]

ABSTRACT

MIP(Maximum Intensity Projection) is a volume rendering technique which is essential for the medical imaging system. MIP rendering based on the ray casting method produces high quality images but takes a long time. Our aim is improvement of the rendering speed using GPGPU(General-purpose computing on Graphic Process Unit) technique. In this paper, we present the ray casting algorithm based on CUDA(an acronym for Compute Unified Device Architecture) which is a programming language for GPGPU and we suggest new acceleration methods for CUDA. In detail, we propose the block based space leaping which skips unnecessary regions of volume data for CUDA, the bisection method which is a fast method to find a block edge, and the initial value estimation method which improves the probability of space leaping. Due to the proposed methods, we noticeably improve the rendering speed without image quality degradation.

Key words: Volume Rendering(볼륨 렌더링), MIP(최대휘소투영), GPGPU(범용 그래픽스 연산), CUDA

※ 교신저자(Corresponding Author) : 계희원, 주소 : 서울특별시 성북구 삼선동2가 389 한성대학교 정보시스템공학과(136-792), 전화 : 02)760-8014, FAX : 02)760-5886, E-mail : kuei@hansung.ac.kr
접수일 : 2011년 3월 3일, 수정일 : 2011년 6월 13일
완료일 : 2011년 7월 12일
[†] 정회원, 한성대학교 정보시스템공학과
[††] 정회원, 한성대학교 정보시스템공학과
(E-mail: kamiru13@naver.com)
※ 본 연구는 한성대학교 교내연구비 지원과제임 또한 이 논문은 2010년도 정부(교육과학기술부)의 재원으로 한국연구재단의 기초연구사업 지원을 받아 수행된 것임(2010-0015641).

1. 서 론

볼륨 렌더링은 컴퓨터 단층촬영(CT : Computed Tomograpy)이나 자기공명영상(MRI : Magnetic Resonance Imaging) 등을 이용하여 얻은 삼차원 볼륨 데이터를 렌더링 하는 방법으로, 관찰 대상인 인체의 내부를 쉽게 알아 볼 수 있어 의료영상 분야에서 필수적인 기술이다. 그 중 최대휘소투영(MIP : Maximum Intensity Projection)은 볼륨 렌더링의 한 가지 방법으로, 볼륨 데이터를 관찰 방향으로 관통하는 광선을 가정하여, 광선 상에서 추출한 값 중 가장 높은 값을 선택하여 보여주는 렌더링 방법이다. 따라서 골격이나 조영된 혈관과 같은 높은 밀도 값을 가지는 조직을 쉽게 관찰 할 수 있다. 최대휘소투영의 결과 영상에는 깊이 정보가 누락되므로, 일반적으로 사용자는 관찰 방향을 자주 바꿔 가며 공간감을 획득한다. 따라서 빠른 속도의 렌더링 방법이 필요하게 된다.

최근 수년간 그래픽 처리장치(GPU : Graphic Process Unit)의 성능이 크게 증가하여 이를 이용한 가속화 방법이 꾸준하게 연구되고 있다. GPU 내부에는 간단한 연산을 수행할 수 있는 연산소자(computing element)가 수백 개 집적되어 있어서 병렬 연산을 효과적으로 수행할 수 있다.

그래픽 처리장치의 성능을 이용하는 방법은 크게 두 가지가 있다. 첫 번째는 OpenGL[1]이나 DirectX [2]와 같은 그래픽스 함수를 이용하는 것이다. 그래픽스 함수를 이용하는 기법은 고전적인 그래픽스 파이프라인의 자료 흐름을 따르기 때문에, 사용자가 구현하고 싶은 알고리즘을 정점 변환(vertex transformation), 질감 샘플링(texture sampling), 조명 효과(shading and illumination), 화소 연산(fragment processing) 등의 틀 안에서 표현해야 하는 어려움이 있었다. 따라서 자료의 정렬(sort)이나 가변길이 영상 압축과 같은 일반적인 연산은 구현되기 어려운 알고리즘이다.

두 번째는 GPU에 일반적인 연산을 시키는 범용 그래픽스 연산(GPGPU : General-Purpose computing on Graphic Processing)[3]을 이용하는 것이다. 2007년 CUDA(an acronym for Compute Unified Device Architecture)[4]로 대표되는 범용 그래픽스 연산의 등장으로, c언어와 비슷한 구조로 GPU를 이용한 병렬 프로그래밍을 수행할 수 있게 되었다. 범용 그래픽스 연산은 그래픽스 함수보다 유연하기는 하나 여전히 알고리즘 구현에 대한 제약이 있기 때문에, 구체적인 문제에 범용 그래픽스 연산을 적용하는 경우에 병렬화 요소에 대한 충분한 연구가 필요하다.

본 연구는 볼륨 렌더링의 가장 일반적인 방법인 광선 투사법(ray casting)을 이용하여 최대휘소투영을 CUDA를 이용하여 구현하고 이에 적합한 가속화 알고리즘을 제안한다. 본 논문의 구성은 다음과 같다. 2장에서는 기반이 되는 기술과 기존의 관련 연구를 소개하고, 3장에서는 본 논문에서 제안하는 CUDA기반의 최대휘소투영 가속화 알고리즘을 설명한다. 4장에서는 실험결과를 보이고 5장에서 결론을 내린다.

2. 기반 기술 및 관련 연구

2.1 광선 투사법

광선 투사법(ray casting)[5]은 볼륨 렌더링 기법 중의 한 가지로 결과 영상의 각 화소(pixel)마다 광선을 발사하여 광선이 지나는 주변의 복셀(voxel)들의 밀도 값을 참조하여 최종 밝기 값을 계산하는 방법이다(그림 1).

각 화소마다 발사되는 광선의 색상을 결정하기 위해, 그림 1의 B와 같이 광선은 일정 간격의 위치마다 복셀과 복셀 사이에서 밀도 값을 취하게 된다. 이때 현재 위치를 둘러싸는 격자의 꼭지점에 위치하는 주변 $8(=2^3)$개 복셀에서 값을 얻어와 삼선형 보간(trilinear interpolation)을 거쳐 현재 위치의 밀도 값을 계산한다. 이 때 광선이 진행하며 한 번의 밀도 값을 계산하는 과정을 샘플링(sampling) 이라고 하며 광선의 진행 간격이 짧을수록 샘플링 횟수가 많아져 결과 영상의 화질이 높아진다. 직접 볼륨 렌더링

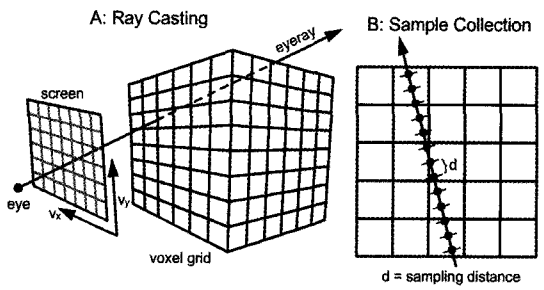


그림 1. 복셀 사이를 지나가는 광선과 샘플링을 취하는 거리[6]

(direct volume rendering)의 경우 각 광선의 샘플들에 대해 불투명도를 이용한 누적(alpha-blending) 연산을 수행[5]하여 광선의 색상을 결정하며, 본 연구와 같은 최대휘소투영 렌더링은 각 광선의 샘플들에 대해 최댓값을 계산하여 광선의 밝기로 삼는다. 이러한 광선 투사법은 렌더링 시간이 오래 걸리지만, 높은 화질의 영상을 생성하기 때문에 많은 가속화 기법들이 연구되고 있다.

2.2 관련 연구

이번 절에서는 볼륨 렌더링의 가속 방법을 CPU 기반 알고리즘과 GPU 기반 알고리즘으로 나누어 살펴본다. 먼저 CPU 기반 가속 방법을 알아보면, 먼저 메모리 참조를 효과적으로 하는 방법이 많이 연구되었다. 최대휘소투영 볼륨 렌더링에서는 상대적으로 연산이 간단하기 때문에, 전체 렌더링 시간은 연산(computation bound) 보다는 메모리 전송에 영향(memory bound)을 주로 받기 때문이다. 샘플링 위치를 적절히 움직여 삼선형 보간 대신 이선형 보간을 사용하여 샘플링을 수행하는 연구가 진행되었다[7]. 샘플 위치의 값을 획득하기 위해 8개의 복셀 대신 4개만을 사용하게 되므로 메모리 참조와 연산이 감소하게 된다. 그리고 광선 투사법 대신 쉬어-왑 분해 알고리즘[8]을 이용하여 메모리를 순차적으로 참조하되, 전치 연산을 이용하여 메모리 사용량을 줄이거나[9] 보간을 선택적으로 수행 [10]하는 방법이 연구되었다.

볼륨 렌더링은 대용량 볼륨 데이터에 비슷한 연산이 반복적으로 적용되는 구조이므로, 단일명령 복수데이터(single instruction multiple data)를 이용하여 대량의 데이터의 처리효율을 증가 시킬 수도 있다[9,10], 또는 멀티코어 병렬 CPU를 사용한 분산처리도 가능하다[11].

그 외에, 영상에 반영되지 않는 부분을 렌더링에서 제외하는 추려내기(culling)기법도 사용되는데, 계층적 가림 지도(hierarchical occlusion maps)를 이용하여 최대휘소투영에 적용한 사례도 존재한다[12]. 그리고 영상의 정확도에 여러 단계를 두어, 사용자의 조작이 있는 경우 낮은 화질의 영상을 출력하고 시간이 지나면 점진적으로 높은 화질의 영상을 출력하는 연구도 수행되었다[13]. 또한 데이터를 밀도 값에 대해 내림차순으로 정렬하고 각 밀도 값을 영상에 투영하여 렌더링하면 점진적 화질 개선이 일어나게 되어, 화질과 속도의 균형을 사용자가 편리하게 결정할 수 있다[14].

이러한 CPU 기반의 기법은 프로그래밍이 자유롭기 때문에 다양한 알고리즘을 손쉽게 적용할 수 있으며, 별도의 하드웨어가 필요하지 않다는 장점이 있으나 512×512×512 정도의 데이터 렌더링 속도가 최대 1-2 frame/sec 정도에 머물러 실시간 렌더링에 부적당하다는 단점이 있다.

한편, GPU 기반 가속 방법을 알아보면, GPU는 상대적으로 빠른 연산이 가능하나, 제한적인 명령어를 효율적으로 사용하여야 고속 렌더링이 가능하다. 따라서 CPU 기반 가속화 알고리즘을 GPU에 적합한 구조로 수정하여 적용하는 예가 많다. 그래픽스 API를 이용하는 방법으로는 블록 기반으로 메모리 참조 효율을 증가시키는 연구가 수행되었고, 추려내기를 효율적으로 하기 위해 가림 질의(occlusion query)를 적용하였다[15].

GPGPU는 GPU의 고속 메모리를 이용하며 다수의 처리장치들에 동시에 같은 명령을 내려 병렬 수행을 한다. 최근 연구에서는 GPGPU를 이용하여 빠른 속도로 볼륨 렌더링을 수행하는 방법에 대한 연구들이 진행되고 있다. GPGPU 언어인 CUDA를 이용하여 고속 병렬처리로 광선 투사법을 구현하였을 경우, 기존의 CPU기반의 광선 투사법에 비하여 속도 향상을 할 수 있다[16]. 또한 기존의 그래픽스 API와 함께 사용하여 GPGPU의 제한적인 기능을 보완한 방법들도 연구되었다[17]. CUDA 기반의 직접 볼륨 렌더링에서 기존의 블록 기반 최대-최소 8진 트리를 생성하여 가속화 하는 방법도 연구되었다[18].

본 연구에서는 범용 그래픽스 연산 개발 언어인 CUDA를 이용하여 광선 투사법을 구현하고 최대휘소투영 볼륨 렌더링에 적용시킬 수 있는 가속화 기법을 제안한다. 높은 화질을 요구하는 최대휘소투영 볼륨 렌더링에서 범용 그래픽스 연산을 적용시켜 화질과 속도를 모두 충족시킬 수 있다.

3. CUDA를 이용한 최대휘소투영 가속화 기법의 구현

3.1 CUDA 기반 광선 투사법의 개요

이번 절에서는 CUDA 기반 광선 투사법이 어떻게

설계되었는지 설명한다. GPU의 내부는 다수의 처리 장치가 동시에 동작하여 속도가 향상되도록 되어 있으며, 하나의 처리장치에서 실행되는 논리적인 프로그램의 단위를 쓰레드(thread)라고 한다. 따라서 하나의 쓰레드에 어떤 작업을 할당할 지, 쓰레드 외부에서 어떤 작업을 수행할 지 결정해야 한다. 설계 결과는 그림 2에 그림으로 나타내었으며 구체적으로 다음과 같다.

광선 투사법은 결과 영상의 매 화소마다 광선을 발사하여 화소의 최종 값을 결정하게 된다. 병렬로 수행되는 서로 다른 쓰레드들은 자신이 갱신하는 값을 다른 쓰레드와 공유할 경우 출력 값에 오류가 발생하므로[4], 각 쓰레드에 하나씩의 광선의 진행을 수행하도록 하는 것이 효과적이다.

GPU 외부에서는 공통 정보인 관찰자 정보와 볼륨 데이터를 GPU에 전달한다. 한편, 광선 투사법에서 정사투영을 구현할 경우, 각 광선의 진행방향은 모두 동일하다. 따라서 광선 진행 방향도 CPU에서 한번만 계산하여 GPU로 전달한다. 이후, 전달된 정보를 이용하여 병렬로 광선 투사법이 수행되고 그 결과는 출력을 위한 메모리에 복사되어, 최종적으로 영상이 출력된다.

GPGPU를 이용한 광선 투사법은 병렬 처리와 그래픽 처리 장치의 고속 부동 소수점 연산(floating-point operation) 성능에 의해 기존의 CPU 기반 광선 투사법에 비해 월등히 빠른 속도를 낼 수 있지만 실시간 영상 렌더링을 위해서는 별도의 고속화 알고리즘의 적용이 필요하다. 따라서 본 연구에서는 GPGPU 기반에서 적용 가능한 가속화 알고리즘을 제안한다.

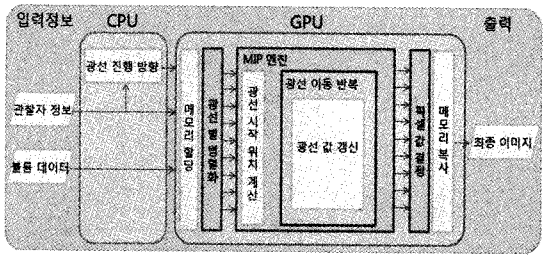


그림 2. CUDA 기반 최대위소투영 광선 투사법의 개요

3.2 블록 기반 공간 도약

볼륨 렌더링은 한 번의 샘플링을 위해 주변 8개의 데이터를 참조해야 하므로 많은 시간이 소요된다. 따라서 결과 화면에 영향을 끼치지 않는 부분을 미리 조사하여 건너뛰게 할 경우 시간을 많이 절약할 수 있다.

직접 볼륨 렌더링에서는 빈공간 도약 기법(empty space leaping)[18,19]을 사용하는데, 볼륨 데이터를 일정한 크기의 블록으로 나누고, 광선의 지나는 블록이 투명하다고 판단되면 해당 블록을 건너뛰는 방법이다. 한편, 최대휘소투영 볼륨 렌더링에서는 투명한 공간이 정의되지 않는다. 따라서 본 연구는, 현재까지 누적된 값보다 밀도 값이 낮은 구간은 결과에 반영되지 않음에 착안하여, 밀도 값이 낮은 블록을 건너뛰는 방법을 제시하였다.

그림 3에서 예를 들면, 광선이 두 번째 블록에 도달할 때, 누적된 값은 17이고 블록의 최댓값은 15라는 사실이 알려져 있다면, 두 번째 블록은 건너뛰어 가속화 할 수 있다. 본 연구는 블록의 최댓값을 선처리하여 별도의 공간에 저장하였다. 광선은 각 블록의 경계면을 지날 때마다 해당 블록의 최댓값과 자신의 현재 밝기 값을 비교하여 광선의 현재 밝기 값이 블록의 최댓값보다 클 경우 데이터를 참조하지 않고 다음 블록으로 이동한다. 이 방법은 기존 연구에서 CPU기반[12] 또는 그래픽스 API를 사용[15]하여 구현되었으며, 본 연구에서는 CUDA로 병렬화하여 각 쓰레드에 대해 도약을 수행하도록 새롭게 구성되었다.

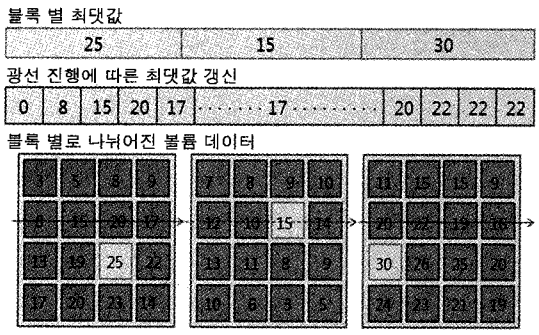


그림 3. 블록 기반 공간 도약 방법의 개요

3.3 이분 이동법을 이용한 효율적인 공간 도약

블록 기반 공간 도약에서 광선이 다음 블록의 시작 위치를 찾아내기 위해서는 광선-상자 교차 검사 기법(Ray-Box intersection)[19]을 이용한다. 광선-상자 교차 검사는 광선의 진행방향으로 현재 위치에서 블록의 경계면까지 거리를 구하는 방법이다. 광선

의 현재 위치를 *RayPosition*, 광선의 진행 벡터를 *vecStep* 이라고 했을 때 가장 가까운 앞쪽의 경계면을 찾기 위한 계산들은 다음과 같다.

1. 현재 블록의 *X*좌표 경계면 *X1*과 *X2*까지 도달하기 위한 이동 횟수 *StepX1*과 *StepX2*

$$StepX1 = (X1 - RayPosition.x)/vecStep.x$$
$$StepX2 = (X2 - RayPosition.x)/vecStep.x \quad (1)$$

2. 광선의 진행방향에 위치한 X좌표 경계면까지의 이동 횟수 *StepX*

$$StepX = \max(StepX1, StepX2) \quad (2)$$

3. 같은 방법으로 *Y*와 *Z* 좌표 경계면까지의 이동 횟수 *StepY*, *StepZ*를 구한다.

4. 가장 가까운 경계면 까지의 이동 횟수 *StepIntersection*

$$StepIntersection = \min(StepX, StepY, StepZ) \quad (3)$$

5. 다음 블록의 경계면에서 광선이 시작해야 될 위치 *NewRayPosition*

$$NewRayPosition = RayPosition + (vecStep * StepIntersection) \quad (4)$$

위와 같이 광선-상자 교차 검사 방법은 블록의 크기에 상관없이 경계면과 광선의 교차점을 찾을 수 있다는 장점이 있지만, 한 번의 계산에 연산 양이 많아 블록 이동 횟수가 많아 질 경우 속도 저하를 가져오게 된다.

따라서 본 연구에서는 블록을 이동하는 방법으로 이분 이동법(Bisection)을 제안한다. 이분 이동법은 블록을 도약하기 위해 광선의 이동거리를 크게 늘렸다가 다음 블록으로 넘어가게 되면 거리를 줄이면서 점점 경계면에 가까워져 가는 방법이다. 즉, 현재 위치에서 일정 거리를 앞서 살펴보고, 앞선 위치가 현재 위치와 같은 블록에 속해있다면 전진하며, 앞선 위치와 현재 위치가 다른 블록에 속하게 되면 거리를 반으로 줄여 반복하는 방법이다.

이분 이동법은 블록 크기와 관찰 방향에 따라서 경계면을 찾기까지 걸리는 시간이 약간씩 변화하나 광선-상자 교차 검사에 비해 계산이 매우 간단하다. 위치의 전진은 덧셈연산으로 간단하며, 이동 거리를 반으로 줄이는 연산은 광선의 진행 방향이 평행하므로 미리 계산해놓을 수 잇다. 그리고 현재 점이 속한 블록의 위치를 찾는 것은 미리 참조 테이블을 구성하여 간단히 파악할 수 있으므로 속도 향상을 얻을 수 있다. 그림 4는 광선-상자 교차 검사 기법과 이분 이동법의 구조를, 설명을 간단히 하기 위해, 2차원으로 나타낸 그림이다.

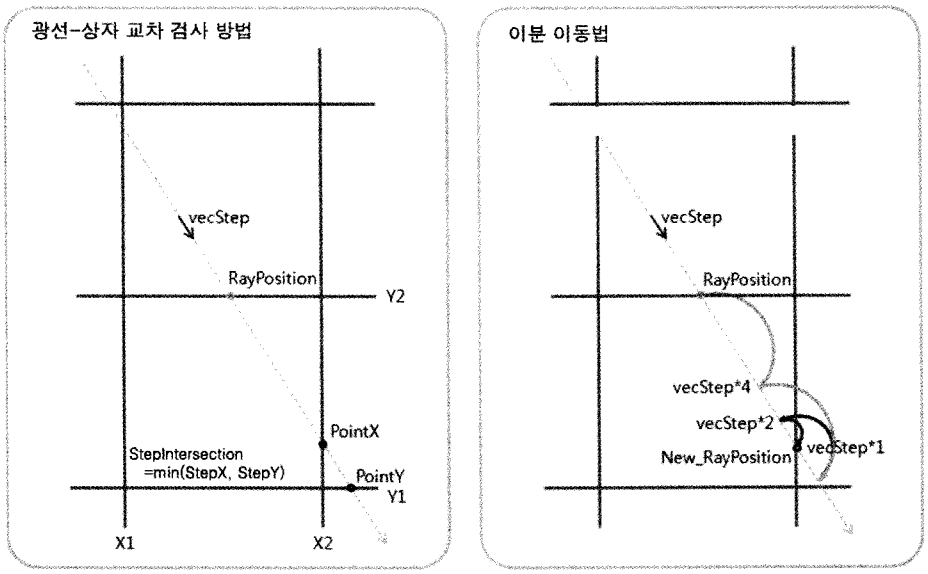


그림 4. 광선-상자 교차 검사 기법과 이분 이동법의 구조

3.4 광선 초기 값 부여를 이용한 공간 도약 비율 향상

블록기반 공간 도약 가속화 방법을 수행하려면, 현재 위치까지 누적된 광선의 밝기 값이 존재하며 충분히 큰 값이어야 공간 도약의 확률이 높다. 그러나 볼륨 데이터는 일반적으로 데이터의 중심에 중요한 정보가 위치하므로, 볼륨 데이터의 외곽 경계 부분은 상대적으로 낮은 값을 갖게 된다. 따라서 광선 진행의 초기에 누적된 밝기 값은 상대적으로 작아서, 블록 도약이 잘 일어나지 않는다. 본 연구는 이러한 문제점을 해결하기 위해, 광선 투사법을 실시하기 전에, 광선의 초기 값에 상대적으로 높은 밝기 값을 지정하는 방법을 제안한다. 제안 방법은 화질 저하가 전혀 없이 렌더링 속도를 향상시킬 수 있다.

앞에서도 언급했지만, 의료영상 볼륨 데이터의 경우, 데이터의 중앙에 밀도 값이 큰 데이터들이 위치하고 있다. 따라서 본 연구는 광선의 초기 값으로 광선이 데이터 중앙을 지날 때의 밀도 값을 이용한다. 예를 들어 한 광선에서 얻는 샘플 값이 차례대로 V_1, V_2, ..., V_n 이라고 가정한다. 이때, 본 연구는 광선의 누적 초기 값을 $M_{init} = V_{\lfloor n/2 \rfloor}$로 결정한다. 광선 시작부의 밀도 값은 작은 편이므로, 일반적으로 $M_{init} > V_1$, $M_{init} > V_2$ 등이 성립하여 공간도약 확률이 향상된다.

그리고, 제안 방법인 초기 값 부여 기법에 의해 화질이 저하되지 않으며 증명은 다음과 같다. 광선의 실제 최댓값을 $M = \max\{V_1, V_2, ..., V_n\}$이라고 정의하면, M_{init}은 광선 내부에서 얻은 값 이므로 $V_{\lfloor n/2 \rfloor} = M_{init} \leq M$ 이 성립한다. 한편, 광선의 최댓값 M이 존재하는 위치에 존재하는 블록 B를 가정할 수 있다. 광선의 최댓값 M이 존재하는 위치를 포함하는 블록 B의 최댓값을 M_B라고 하면, $M \leq M_B$가 된다. 따라서, 광선이 블록 B를 통과할 때, $M_{init} \leq M \leq M_B$ 가 되어 광선은 최댓값을 도약하지 않고, 샘플링을 수행하여 M을 찾게 된다. 따라서 광선의 최댓값 M이 누락되지 않고 화질이 보존된다. 여기에 대한 설명을 그림 5에 나타내었다.

영상의 초기 값을 0으로 할당하는 경우 대부분의 볼륨 데이터를 샘플링하며 진행해야 한다. 진행 과정에서 큰 값을 찾게 되면 그 이후는 블록 이동이 가능해진다. 한편, 영상의 초기값을 데이터 중심에서 얻게 되면 초기 영상의 값이 큰 값으로 저장되므로 블록 이동의 확률이 높아지게 되어 성능이 향상된다.

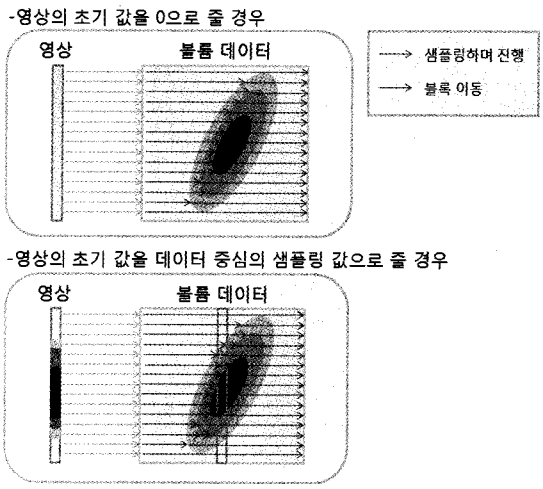


그림 5. 초기 값 부여에 따른 블록 도약 비율의 증가

3.5 제안 방법의 적용과 구현

본 연구에서는 최대휘소투영의 특성을 이용하여 밀도 값이 낮은 부분을 광선이 샘플링 하지 않고 건너 뛸 수 있도록 하는 공간 도약 기법을 적용시켰다. 본 연구가 제안한 세 가지 가속화 기법이 모두 적용된 결과는 그림 2를 확장하여 그림 6과 같이 설명된다.

본 연구는, 볼륨 데이터를 선처리 과정에서 일정한 크기의 3차원 블록으로 나누고 각각의 블록 내부의 최댓값을 계산하여 저장하고 GPU에 전달한다. 사용자 조작을 통해 광선의 진행 방향이 결정되면, 각각의 광선은 볼륨 데이터와 교차하는 선분 영역의 중심에서 밀도 값을 얻어 광선의 초기 값으로 추출한다.

이후 광선이 진행하며 만나는 블록마다 미리 저장된 블록별 최댓값과 현재 광선의 최댓값을 비교하여 광선의 최댓값이 블록의 최댓값보다 클 경우 다음 블록까지 공간을 도약한다. 이 때 다음 블록의 시작 위치로 이동하기 위한 방법으로 광선-상자 교차 검사기법 대신 이분 이동법이 적용되어 속도가 더욱 향상된다. 광선의 최댓값보다 블록의 최댓값이 클 경우에는 다음 블록을 만날 때 까지 주변 8개 복셀들을 보간하여 광선의 최댓값을 갱신하며 진행해 나간다. 최종적으로 각 광선의 최댓값들을 결과 영상의 픽셀 값으로 저장하여 사용자에게 출력한다. 상기 알고리즘은 CUDA를 이용하여 구현되었으며, 그림 6의 MIP 엔진으로 표시된 핵심 코드를 의사 코드(pseudo code)의 형태로 부록에 첨부하였다.

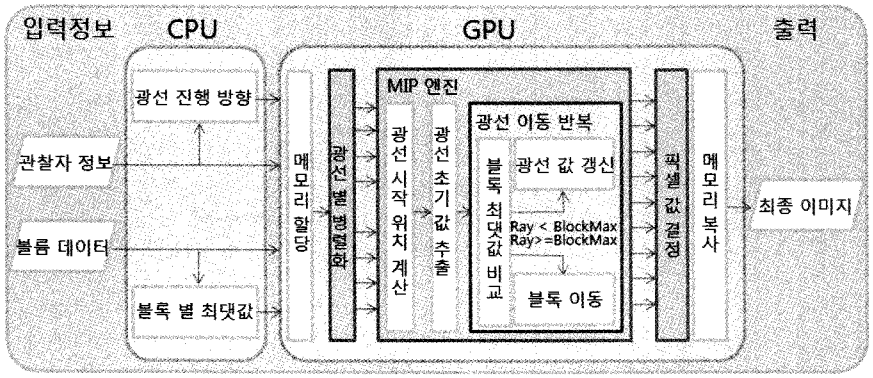


그림 6. 최대휘소투영 광선 투사법 가속화 적용의 개요

4. 실험 및 결과

본 연구에서는 그래픽 처리 장치의 빠른 연산속도를 이용해 볼륨 데이터를 렌더링하기 위하여 범용 그래픽스 연산을 이용하여 최대휘소투영을 수행하고, 범용 그래픽스 연산에 맞는 새로운 가속화 방법을 제시하였다. 이번 장에서는 3.2절에서 설명한 블록 기반 공간 도약을 적용시킨 실험결과와 3.3절에서 설명한 공간 도약 가속화를 적용시킨 실험결과를 먼저 비교해 본 뒤, 3.4절에서 설명한 초기 값 부여를 적용시킨 실험결과를 추가 비교하여 각 단계에서 가속화가 이루어졌음을 증명하도록 한다.

4.1 실험 환경

제안한 시스템은 Windows XP 32bit 운영체제에서 Visual Studio 2005를 이용해 MFC와 CUDA 기반으로 구현되었으며, 실험을 수행한 하드웨어 환경은 Intel Core2Duo CPU와NVdia의 Geforce 9800GTX +그래픽스 보드를 장착한 PC이다.

제안한 시스템의 타당성을 입증하기 위해 서로 다른 크기의 4가지 볼륨 데이터를 이용하여 관찰 시점을 바꿔가며 평균 렌더링 시간을 측정하였다. 사용된 데이터와 각 데이터의 해상도는 표 1과 같으며 결과 영상은 그림 7에 나타내었다. 본 연구에서 제안된 공간 도약은 최종 영상에 반영되지 않는 불필요한 부분에서만 일어나므로, 제안 방법으로 인한 화질 손실은 전혀 없다.

표 1. 각 실험용 볼륨데이터의 크기

데 이 터	크 기
Bighead (머리)	256 × 256 × 225
Dental (턱)	512 × 512 × 216
Chest (흉부)	512 × 512 × 256
Legs (다리)	512 × 512 × 512

그림 7. 각 실험용 볼륨 데이터의 최대휘소투영 결과 영상

4.2 실험 결과 및 분석

각 데이터별로 블록의 크기를 변경하며 렌더링 시간을 측정하여 그림 8에 그래프로 나타내었다. 그래프의 파란색 선이 블록 크기에 따른 렌더링 시간 측정 결과이며, 시간 눈금은 왼쪽의 세로축에 표시되었다. 블록의 크기가 커지면 렌더링 시간이 오래 걸리는 것을 확인할 수 있다.

한편, 블록이 렌더링 된 비율을 측정하여 주황색 선으로 도시하였다. 렌더링 된 비율은 (1 - 공간 도약 비율)로 측정되었으며 눈금은 각 그래프 오른쪽의 세로축에 표시하였다. 모든 데이터에서 확인할 수 있

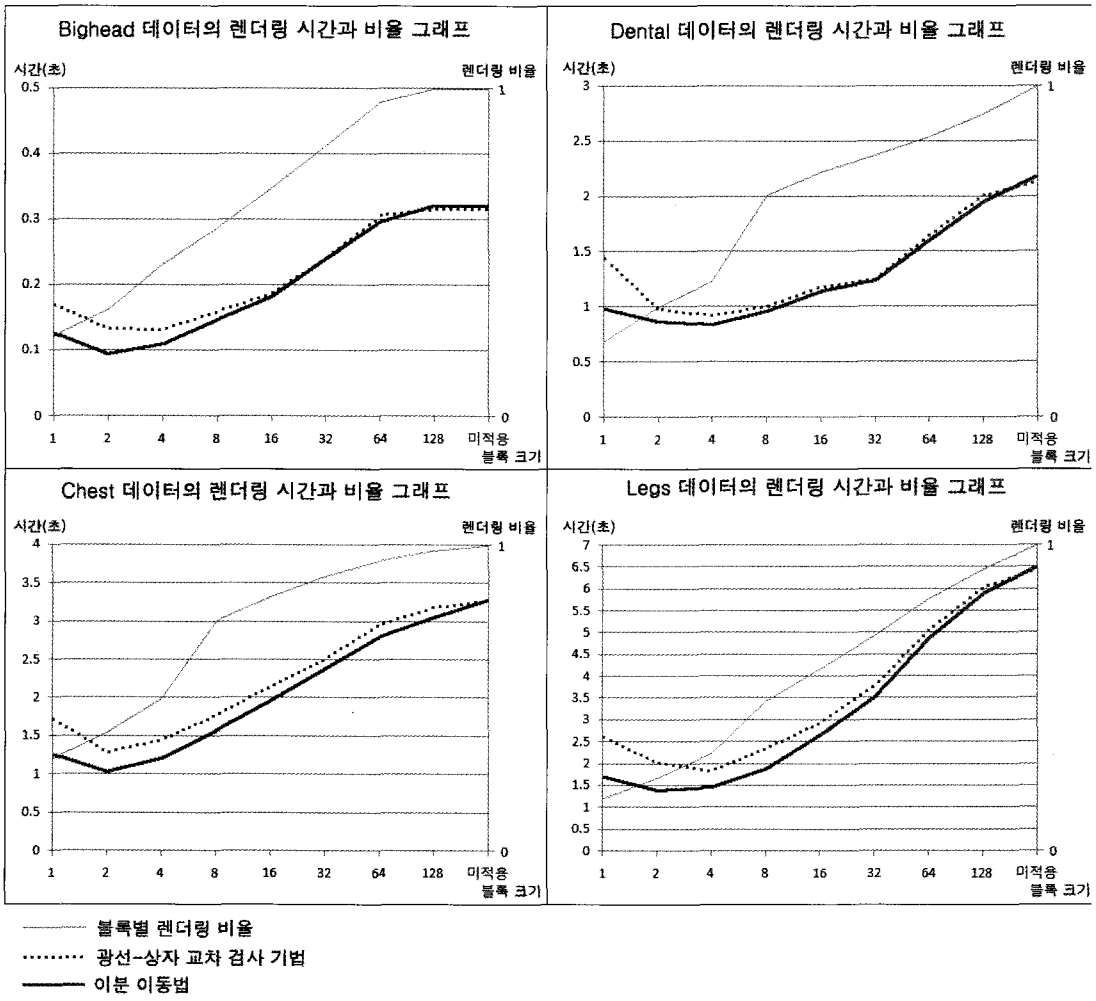


그림 8. 각 데이터의 블록 크기에 따른 렌더링 비율과 이동 방법에 따른 시간의 변화

듯이, 블록 크기가 클수록 렌더링 비율이 높아지고 블록 크기가 볼륨 데이터의 크기에 가까워지면 거의 모든 데이터를 렌더링 하고 있음을 확인할 수 있다.

그리고 본 연구의 두 번째 제안 방법인 이분 이동법을 추가로 적용하고 렌더링시간을 측정한 결과를 보라색 선으로 표시하였다. 특히 블록 크기가 작은 경우에, 기존의 광선-상자 교차 검사기법(파란색 선)에 비해 성능 향상이 두드러진 것을 확인할 수 있다.

이러한 실험 결과로서 얻을 수 있는 분석은 다음과 같다. 먼저, 블록 크기가 작을수록 공간 도약의 효율이 향상되고 그로 인해 렌더링 시간이 단축된다. 이는 직관과 일치하는 실험 결과로서, 예를 들어 블록 크기를 볼륨 데이터의 크기와 같게 하면 블록의 최댓값은 볼륨 데이터에서 존재하는 최댓값이 되어, 도약을 전혀 할 수가 없게 된다. 반면 블록 크기가 작은 경우 하나의 블록에 포함된 조직이 단순해지므로, 지역적으로 블록의 최댓값이 작아지는 효과가 있고 도약 확률이 높아진다.

한편, 블록 크기가 매우 작은 경우 렌더링 비율은 감소함에도 불구하고 렌더링 시간은 증가하는 모습이 관찰된다. 그 이유는 다음과 같이 설명된다. 고정된 크기의 볼륨 데이터에서 블록 하나의 크기가 작아지면 블록의 개수가 증가하여, 광선 당 공간 도약의 빈도가 증가하게 된다. 이때, 광선-상자 교차 검사기법을 여러 번 수행하게 되고, 공간 도약으로 인한 시간 이득을 광선-상자 교차 검사가 모두 상쇄하게 된다.

본 연구의 두 번째 제안 방법인 이분 이동법을 적용하면 공간 도약의 연산 시간을 감소시켜 앞의 문제

를 개선하게 된다. 그림 8에서 이분 이동법은 블록 크기가 작은 경우에 광선-상자 교차 기법에 비해 높은 성능 향상을 보인다. 이분 이동법으로 공간 도약을 할 경우 블록 크기가 2인 경우 가장 빠른 속도를 보였다.

광선 초기 값을 부여했을 경우 추가적인 성능 향상을 확인하고, 본 연구의 종합적인 성능 향상을 파악하기 위해, 렌더링 시간과 렌더링 비율을 표 2에 비교하였다. 측정 결과는 앞의 실험 결과에서 가장 빠른 속도를 보이는 블록 크기 2를 기준으로 하였다.

광선의 초기 값 부여 기법을 통해 렌더링 비율이 5% 정도 추가로 감소하였고, 더 많은 블록을 도약할 수 있었으며, 렌더링 시간의 단축은, Dental 데이터의 최저 5% (870.01/827.71 = 1.05)에서 Chest 데이터의 최고 38%(1032.86/743.81 = 1.38)까지 나타났다. 결과는 데이터 특성에 따라 변화가 심한데, 제안 방법은 광선 시작 부분의 밀도가 광선 중심부의 밀도에 비해 확연히 낮은 경우에 효과가 높은 방법이기 때문이다. 다만, 초기 값 추출의 비용이 광선 중심부의 값을 추출하는 것으로 간단하기 때문에, 모든 데이터에서 추가적인 속도 향상을 얻을 수 있었다.

종합적으로 본 연구에 의한 최대휘소투영의 성능 향상을 분석하면, 블록을 이용한 공간 도약 기법을 통해 2.18배(Dental)에서 3.16배(Legs), 평균적으로 2.5배의 속도 향상을 얻었다. 그리고 이분 이동법과 광선 초기 값 추정 기법의 효과를 더한 최종적인 속도 향상은, 2.57배(Dental)에서, 5.02배 (Legs), 평균적으로는 4배 정도의 성능 향상을 얻었다.

한편, CUDA를 사용한 본 연구의 성능을 CPU를 이용한 최적화 결과와 비교하기 위하여, 가속화된 광선 추적법[19]을 CPU만을 이용하여 구현하고, 속도를 측정하여 표 3에 나타내었다. 대략 4배 정도 속도가 향상되었으며, GPU의 대규모 병렬처리 기능이 효과적임을 확인할 수 있다.

5. 결 론

최근 하드웨어의 발달로 인해 그래픽스 가속장치를 이용한 볼륨 렌더링의 가속화에 대한 연구가 활발하게 진행되고 있다. 본 연구에서는 볼륨 렌더링의

표 2. 제안 방법을 통한 가시화 속도의 향상 결과

데이터		가속화 미적용 [17]	공간 도약 (블록 크기 2))	공간 도약 +이분 이동법	공간 도약+이분 이동법 +초기 값
Bighead	시간 (ms)	317.25	133.34	96.01	74.41
	상대속도	1	2.38	3.30	4.26
	렌더링 비율	100%	33.95%	33.95%	28.98%
Dental	시간 (ms)	2130.81	977.57	870.01	827.71
	상대속도	1	2.18	2.44	2.57
	렌더링 비율	100%	34.11%	34.11%	28.63%
Chest	시간 (ms)	3268.91	1291.85	1032.86	743.81
	상대속도	1	2.53	3.16	4.39
	렌더링 비율	100%	38.31%	38.31%	31.65%
Legs	시간 (ms)	6417.43	2030.66	1390.99	1278.69
	상대속도	1	3.16	4.61	5.02
	렌더링 비율	100%	23.15%	23.15%	20.47%

표 3. CPU와 GPU의 성능 비교

데 이 터	CPU[19] (ms)	제안방법 (ms)	속도향상 (배)
Bighead (머리)	332.45	74.41	4.47
Dental (턱)	2905.7	827.71	3.51
Chest (흉부)	3402.64	743.81	4.57
Legs (다리)	5349.22	1278.69	4.18

한 방법인 최대휘소투영을 범용 그래픽스 연산인 CUDA에 적합하게 구현하고 가속화 하는 방법을 제안하였다. CUDA를 이용한 볼륨 렌더링의 병렬화 설계를 보였으며, 세 가지 가속화 기법인 블록기반 공간 도약, 이분 이동법을 통한 광선도약, 광선 초기값 추정법을 제안하였다. 그리고 제안한 시스템의 타당성을 입증하기 위하여 서로 다른 네 가지의 데이터를 이용하여 시점을 바꿔가며 영상을 생성하고, 평균 시간을 측정하였다. 실험결과로 화질 저하 없이, 공간 도약을 적용시키지 않았을 경우와 비교하여, 평균적으로 4배 정도의 속도 향상을 얻었다. 이 결과는 최근 고해상도 대용량 볼륨 데이터의 고속 렌더링에 적합할 것으로 기대한다.

부록. CUDA 의사 코드

```
__global__ float TraceRay()
{
  // 쓰레드의 번호로부터 영상좌표를 획득
  x = __umul24(blockIdx.x, blockDim.x) +
threadIdx.x;
  y = __umul24(blockIdx.y, blockDim.y) +
threadIdx.y;

  // 광선의 방정식을 얻고, 광선 시작위치 계산
  float3 vRayStart = CoordTransform(x,y);
  float2 minmax = RayBoxIntrsc(vRayStart);
  if( NoHit(minmax) ) return;

  // 광선의 중심에서 초기값 추출
  float mid = (minmax.max + minmax.min) / 2;
  float3 pos = GetPos(vRayStart, mid);
  float current_max = tex3d(volume, pos);

  // 광선 이동 반복
  for(t=minmax.min; t<minmax.max; t=t+1) {

    // 현재 샘플좌표와 블록 정보 획득
    pos = GetPos(vRayStart, t);
    int blockID = GetBlockID(pos);
    // 블록 최댓값 비교
    if(B_max[blockID] < current_max ) {
      // 블록 이동
      for(;GetBlockID(GetPos(vRayStart,
t+1))==blockID; t=t+1);
      continue;
    }
    // 볼륨데이터에서 값을 얻어, 광선 값 갱신
    float den = tex3D(volume, pos);
    current_max = max(current_max, den);
  }
  return current_max;
}
```

참 고 문 헌

[1] OpenGL, http://www.opengl.org/

[2] DirectX, http://msdn.microsoft.com/en-us/directx/

[3] GPGPU.org, http://gpgpu.org/

[4] CUDA, http://www.developer.nvidia.com/

[5] M. Levoy, "Volume Rrendering Display of Surfaces from Volume Data," IEEE Computer Graphics and Application 1988, Vol.8, pp. 29-37, 1988.

[6] R. Zirbes, "Scientific Visualization: Volume Surface Rendering," http://johnrichie.com/V2/richie/isosurface/volume.html

[7] V. Pekar, D. Hempel, G. Kiefer, M. Busch, and J. Weese, "Efficient Visualization of Large Medical Image Datasets on Standard PC Hardware," Joint EUROGRAPHICS -IEEE TCVG Symposium on Visualization 2003, pp. 135-140, 2003.

[8] P. Lacroute and M. Levoy, "Fast Volume Rendering Using a Shear-Warp Factorization of the Viewing Transformation," In Proceedings of ACM SIGGRAPH 1994, pp. 451-457, 1994

[9] 계희원, "단일 명령 복수 데이터 연산과 순차적 메모리 참조를 이용한 효율적인 최대 휘소 투영 볼륨 가시화," 한국멀티미디어학회 논문지, 제12권 4호, 2009.

[10] L. Fang, Y. Wang, B. Qiu, and Y. Qian, "Fast Maximum Intensity Projection Algorithm Using Shear Warp Factorization and Reduced Resampling," Magnetic Resonance in Medicine, Vol.47, pp. 696-700, 2002.

[11] G. Kiefer, H. Lehmann, and J. Weese, "Fast Maximum Intensity Projections of Large Medical Data Sets by Exploiting Hierarchical Memory Architectures," *IEEE TRANSACTIONS ON INFORMATION TECHNOLOGY IN BIOMEDICIN*E, Vol.10, No.2, pp. 385-394, 2006.

[12] Benjamin Mora and David S. Ebert, "Low-Complexity Maximum Intensity Projection," *ACM Transactions on Graphics*, Vol.24, No.4, pp. 1392-1416, 2005.

[13] K.H. Kim and H.W. Park, "A Fast Progressive

Method of Maximum Intensity Projection," *Computerized Medical Imaging and Graphics*, Vol.25, pp. 433-441, 2001.

[14] L. Mroz, H. Hauser, and E. Groller, "Interactive High-Quality Maximum Intensity Projection," EUROGRAPHICS 2000, Vol.19, No.3, pp. 341-350, 2000.

[15] H. Kye and D.K. Jung, "Accelerated MIP Based on GPU Using Block Clipping and Occlusion Query," Computers and Graphics, Vol.32, No.3, pp. 283-292, 2008.

[16] L. Marsalek, A. Hauber, and P. Slusallek, "High-Speed Volume Ray Casting With CUDA," Interactive Ray Tracing, RT 2008. IEEE Symposium, pp. 9-10, 2008.

[17] A. Weinlich, A. Benjamin, S. Holger, K. Markus, and H. Joachim, "Comparison of High-Speed Ray Casting on GPU Using CUDA and OpenGL," High-Performance and Hardware-Aware Computing, pp. 25-30, 2008.

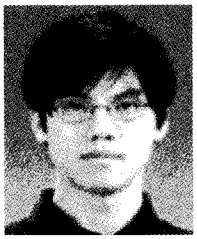
[18] 임종현, 신병석, "CUDA를 이용한 최대-최소 8진트리 생성 기법," 한국게임학회지 제9권 6호 pp. 191-196, 2009.

[19] M. Levoy, "Efficient Ray Tracing of Volume Data," *ACM Transactions on Graphics*, Vol. 9, No.3, pp. 245-261, 1990.

계 희 원

1999년 서울대학교 전산과학과 학사
2001년 서울대학교 전기컴퓨터 공학부 석사
2005년 서울대학교 전기컴퓨터 공학부 박사

2006년~2007년 서울대학교 컴퓨터연구소 연구원
2007년~현재 한성대학교 정보시스템공학과 조교수
관심분야 : 볼륨 가시화, 실시간 렌더링, 대용량 영상처리



김 준 호

2009년 한성대학교 정보시스템공학과 학사
2011년 한성대학교 정보시스템공학과 석사