Journal of Korea Multimedia Society Vol. 12, No. 2, February 2009(pp. 209-218)

다중 GPU 기반의 고속 삼차원 역전사 기법

이병훈[†], 이호[††], 계희원[†††], 신영길[††††]

요 약

삼차원 역전사(3D backprojection) 기법은 수백 장의 이차원 투영영상을 가지고 대상물의 공간적인 위치 파악이 가능한 단층 영상(tomography)을 생성하기 위해 사용되는 재구성 기법이다. 재구성 기법은 단층 영상을 구성하는 결과볼륨의 모든 화소로부터 각 화소 위치에 기여할 값을 이차원 투영영상에서 계산하여 얻어오기 때문에 결과볼륨이 커지거나 투영영상의 수가 증가하게 되면 전체 계산량은 상당히 증가하게 된다. 이러한 문제를 해결하기 위해 최근 범용 그래픽스 하드웨어(graphics processing unit: GPU) 기반의 고속 삼차원 재구성 기법이 연구되었으며 상당한 성능 향상을 가져왔다. 본 논문에서는 기존의 단일 GPU 기반의 삼차원 재구성 기법을 다중 GPU 기반으로 확장할 때 입력되는 투영영상 크기와 결과볼륨의 크기에 따라서 효율적으로 동작될 수 있는 두 가지 병렬 처리 구현 기법에 대해 제시하고 비교 분석한다. 제안한 병렬 처리 구현 기법은 투영영상을 입력 데이터로 간주하여 각 GPU가 모든 투영영상에 대해서 출력 데이터인 결과볼륨을 분할하여 생성하는 결과볼륨 분할생성 기법과 각 GPU가 투영영상을 분산적재하여 할당 받은 입력 데이터에 대한 결과볼륨을 출력한 후 각각의 출력 결과를 CPU에서 합하는 투영영상 분산적재 기법이다. 실험 결과, 결과볼륨의 크기가 GPU에 모두 할당할 수 있는 크기인 경우에는 결과볼륨 분할생성 기법이 더 좋은 성능을 보였고, 결과볼륨의 크기가 GPU 메모리보다 큰 경우에는 투영영상 분산적재 기법이 더 유리하였다.

Fast Multi-GPU based 3D Backprojection Method

Byeonghun Lee[†], Ho Lee[††], Heewon Kye[†††], Yeong Gil Shin[††††]

ABSTRACT

3D backprojection is a kind of reconstruction algorithm to generate volume data consisting of tomographic images, which provides spatial information of the original 3D data from hundreds of 2D projections. The computational time of backprojection increases in proportion to the size of volume data and the number of projection images since the value of every voxel in volume data is calculated by considering corresponding pixels from hundreds of projections. For the reduction of computational time, fast GPU based 3D back-projection methods have been studied recently and the performance of them has been improved significantly. This paper presents two multiple GPU based methods to maximize the parallelism of GPU and compares the efficiencies of two methods by considering both the number of projections and the size of volume data. The first method is to generate partial volume data independently for all projections after allocating a half size of volume data on each GPU. The second method is to acquire the entire volume data by merging the incomplete volume data of each GPU on CPU. The in-complete volume data is generated using the half size of projections after allocating the full size of volume data on each GPU. In experimental results, the first method performed better than the second method when the entire volume data can be allocated on GPU. Otherwise, the second method was efficient than the first one.

Key words: Backprojection(역전사), 3D Reconstruction(삼차원 재구성), GPU(그래픽스 하드웨어), Parallel Processing(병렬처리)

※ 교신저자(Corresponding Author) : 이호, 주소 : 서울특별시 관악구 관악로 599(151-744), 전화 : 02)880-1860, FAX : 02)886-7589, E-mail : holee@vplab.snu.ac.kr
접수일 : 2008년 11월 20일, 완료일 : 2009년 2월 2일
[†] 준회원, 서울대학교 컴퓨터공학부 박사과정
(E-mail : intellee@vplab.snu.ac.kr)
[††] 준회원, 서울대학교 컴퓨터공학부 박사
[†††] 정회원, 한성대학교 정보시스템공학과 전임강사
(E-mail : kuei@hansung.ac.kr)
[††††] 정회원, 서울대학교 컴퓨터공학부 교수
(E-mail : yshin@csc.snu.ac.kr)
※ 본 연구는 서울시 산학연 협력사업(10888), 지식경제부 중기거점 기술개발 사업(10028331-2008-23), 서울대학교 컴퓨터연구소(0421-20080066) 지원으로 수행되었음. 또한 한성대학교 교내연구비 지원으로 수행되었음

1. 서 론

컴퓨터 단층(CT) 삼차원 재구성 기법은 물체를 중심으로 측면에서 투영된 이차원 영상을 바탕으로 삼차원 볼륨 단층 영상을 얻어내는 기법으로 산업과 의료 분야에 널리 이용되고 있다. 얻어진 영상은 볼륨렌더링[1]을 통해 삼차원으로 가시화될 수 있어서 물체의 파괴나 침습 및 변형없이 내부를 관찰하는 것이 가능하다. 하지만 삼차원 재구성은 삼차원 볼륨의 모든 화소(voxel)로부터 모든 이차원 투영영상에 대해 현재 위치에 기여할 값을 가진 위치를 계산해서 가져와야한다. 따라서 결과볼륨의 크기와 투영영상의 수에 비례하여 시간 복잡도가 증가하기 때문에 결과볼륨의 크기가 커지거나 투영영상의 수가 증가하게 되면 전체 계산량은 상당히 증가하게 된다.

이러한 문제를 해결하기 위해, 기법 개선을 통한 시간 복잡도를 감소시키는 기법[2,3]이 연구되었다. 이 기법은 푸리에 변환을 이용하여 주파수 영역에서 재구성을 수행하는 기법으로 계산 복잡도를 $O(N^4)$에서 $O(N^3logN)$으로 감소시켰다. 하지만 이 기법은 고주파수 영역에서 보간할 때 인위적인 잡음(artifact)이 발생하고, 이 문제를 해결하기 위해서 과표본화(oversampling)를 하는데 이것이 계산 복잡도의 차수는 증가시키지 않지만 계수를 증가시켜 전체 계산량이 증가하게 된다.

위와 같은 문제를 해결하기 위해 삼차원 재구성은 공간 영역에서 병렬처리를 이용하여 성능을 향상시키는 방법으로 발전해왔다. 병렬처리를 통한 삼차원 재구성 성능을 향상시키는 방법으로 여러 개의 데이터를 하나의 인스트럭션으로 처리할 수 있는 장비를 이용하거나[4] 여러 대의 컴퓨터를 이용한 분산처리 기법 등이 제안되었다[5-7]. 이를 통해 한 번에 많은 데이터를 처리함으로써 상당한 성능 향상을 가져왔지만 이들 장비는 상당한 고가이고 문제없이 원활하게 운용할 수 있는 인력은 많지 않다.

한편 최근에는 범용 그래픽스 하드웨어(graphics processing unit: GPU)를 이용하여 경제적인 문제를 해결하고 성능을 향상시키는 기법이 제안되었다[8-11]. GPU는 여러 개의 셰이더(shader)로 구성되어 있고 각각의 셰이더는 동시에 연산이 가능하다. 또한 각각의 셰이더는 네 개의 색상 채널(red / green / blue / alpha)을 하나의 인스트럭션으로 처리할 수 있는 SIMD(Same Instruction Multiple Data) 연산을 지원하며 하나의 셰이더는 하나의 CPU와 비등한 산술 연산 성능을 낸다. Feldkamp, Davis, Kress (FDK)에 의해 제안된 역전사(backprojection) 기법[12]은 모든 화소가 서로의 결과에 영향을 주거나 받지 않기 때문에 병렬처리에 매우 적합하고 GPU는 이러한 특징에 매우 적합하다. 특히 투영영상으로부터 이선형 보간(bilinear interpolation)을 이용하여 원하는 값을 가져와야하는데 GPU의 텍스처 사상을 통해 이러한 보간을 매우 효율적으로 처리할 수 있다. 최근 GPU는 프로그래밍 가능한(programmable) 하드웨어라 하여 기존 GPU에 비해 훨씬 더 유연한 프로그램이 가능하고 CPU보다 빠른 연산 속도를 보여준다. 또한 개인용 컴퓨터의 메인 메모리보다 성능이 뛰어난 메모리를 탑재하고 있기 때문에 단순하지만 방대한 연산 측면에서 CPU보다 훨씬 뛰어난 성능을 보여준다. 뿐만 아니라 앞서 서술한 기존의 장비에 비해 훨씬 저렴하고 운용하는 것이 자유롭기 때문에 삼차원 재구성 분야와 관련하여 활발한 연구가 진행 중이다.

이러한 GPU의 성능을 극대화하기 위해 다중 GPU[13,14]를 이용하는 방법도 연구되고 있다. 확장 가능한 링크 인터페이스(Scalable Link Interface: SLI)[13]는 엔비디아(NVIDIA)의 기술로 두 개의 동일한 GPU가 텍스처 메모리 등을 공유하고 작업 부담을 분산하여 단일 결과를 출력하는 구조이다. 두 GPU는 SLI 브릿지(bridge)를 통해 출력 결과를 한 쪽 GPU로 전송하여 화면에 출력한다. SLI는 크게 두 가지 방식으로 작동하는데, 화면을 분할하여 각각의 GPU가 나눠서 처리하는 분할 프레임 렌더링(Split Frame Rendering: SFR)과 프레임 단위로 GPU가 번갈아가며 처리하는 교대 프레임 렌더링(Alternate Frame Rendering: AFR)이 있다. 크로스파이어(CrossFire)[14]는 에이티아이(ATI)의 기술로 SLI처럼 각 GPU가 텍스처 메모리 등을 공유하여 작업을 분담하여 성능을 향상시키는 기술이다. 크로스파이어는 분할 프레임 렌더링과 교대 프레임 렌더링 이 외에 수퍼 타일링(Super-tiling)이라는 기법이 있는데 이는 하나의 프레임을 작은 타일로 나눠 각각의 GPU가 타일을 분담하여 처리하는 방식이다.

하지만 SLI와 크로스파이어 모두 하드웨어 차원에서 작동하기 때문에 사용자 입장에서는 하나의

GPU를 사용하는 것과 동일하고 모든 GPU가 동일한 리소스를 적재하고 있어야하기 때문에 GPU의 수가 증가하여도 메모리의 이득이 전혀 없고 각각의 GPU를 개발자가 직접 제어할 수 없어 삼차원 재구성 측면에서는 효율성이 떨어진다.

본 논문에서는 다중 GPU를 이용하여 기존의 단일 GPU 기반의 기법보다 더 좋은 성능을 내고 GPU 수의 증가에 따른 메모리의 증가도 활용할 수 있는 효율적인 두 가지 병렬 처리 구현 기법을 제시하고 비교분석한다. 제안한 병렬 처리 구현 기법은 투영영상을 입력 데이터로 간주하여 각 GPU가 모든 투영영상에 대해서 출력 데이터인 결과볼륨을 분할하여 생성하는 결과볼륨 분할생성기법과 각 GPU가 투영영상을 분산적재하여 할당 받은 입력 데이터에 대한 결과볼륨을 출력한 후 각각의 출력 결과를 CPU에서 합하는 투영영상 분산적재 기법이다.

이 후 논문 구성은 다음과 같다. 2장에서는 관련 연구로 FDK 역전사 기법과 GPU 파이프라인에 대해서 간략하게 설명한다. 3장에서는 본 논문에서 제안하는 다중 GPU 기반의 기법을 소개하며 4장에서는 제안된 기법을 실험한 결과를 보여주고 5장에서는 결론을 맺는다.

2. 관련연구

2.1 FDK 역전사 기법

콘빔(cone beam) CT는 물체를 중심으로 감지기(detector)가 일정한 각도로 회전하면서 빔원(beam source)에서 발생하는 방사선이 물체를 투과하는 양을 감지기에서 감지하는 장비이다. 이를 다시 삼차원 데이터로 재구성하기 위해서 각 감지기로부터 빔원까지 진행하면서 물체 공간에 감지된 값을 누적해간다. 하지만 실제 구현에서는, 감지기로부터 빔원까지 교차되는 물체 공간의 좌표가 정수로 떨어지지 않아 계산이 불편한 점이 있다.

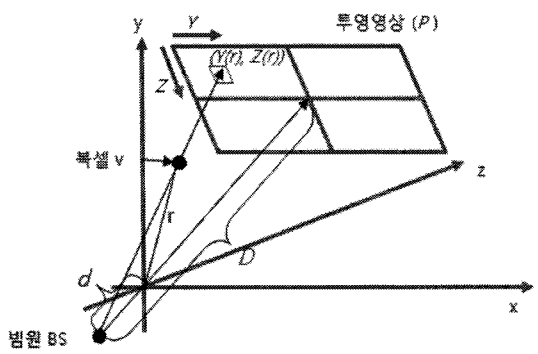


그림 1. FDK 역전사 기법

위의 문제를 해결하기 위해 FDK 역전사 기법[12]은 물체 공간의 각각의 위치에서 빔원의 위치를 고려하여 투영영상의 어느 위치의 값을 참조하여 현재 위치에 누적할지 계산하는 기법이다.

현재 화소에서 참조해야할 투영영상의 위치는 다음과 같이 구할 수 있다.

$$Y(r)=\frac{r_y}{d+r_x}D,\quad Z(r)=\frac{r_z}{d+r_z}D \tag{1}$$

$$\widehat{P_\Phi}(r)=P_\Phi(Y(r),Z(r)) \tag{2}$$

여기서 r는 삼차원 재구성 공간의 중심에서 임의의 화소의 위치를 나타내는 벡터이다. r와 빔원의 위치로부터 중심까지 거리(d), 빔원으로부터 투영영상까지의 거리(D)를 알면 해당하는 투영영상의 위치(P_Φ)를 구할 수 있고 이를 이용해서 원하는 값($\widehat{P_\Phi}$)을 얻어올 수 있다. 이와 같은 과정을 물체의 모든 화소와 모든 투영영상에 대해서 반복한다.

2.2 GPU 파이프라인

GPU는 그래픽 관련 응용 프로그램을 가속시키는 장치이다. GPU는 다수의 세이더 유닛(shader unit)으로 구성되어 있으며 각각의 세이더 유닛은 병렬적으로 작동한다. 또한 세이더 유닛 하나는 CPU와 비등하거나 우수한 산술 연산을 보이며 IEEE-754 32비트 실수 처리가 가능하여 매우 큰 성능 향상을 가능하게 한다[15].

GPU는 다각형, 선과 같은 도형 모델을 래스터 영상으로 변경하는데 이 과정을 GPU 파이프라인이라 한다. 그림 2에서와 같이 GPU 파이프라인[16]은 일반적으로 기하 처리, 래스터 변환, 프라그먼트(fragment) 연산의 세 단계로 나뉜다. 기하 처리 단계에서는 입력된 정점에 행렬을 적용하여 화면좌표계로 변환하고 정점 별 조명처리(per-vertex lighting)를 수행한다. 래스터 변환 단계에서는 투영된 프리미티브를 프라그먼트로 분해하는 단계이다. 래스터 변환은 프로그램이 가능하지 않고 하드웨어에서 이루

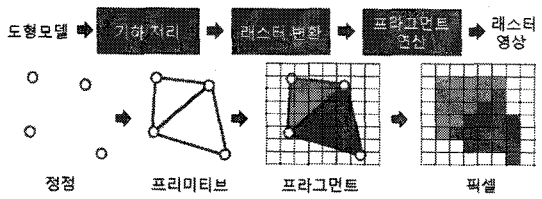


그림 2. GPU 파이프라인

어진다. 래스터 변환을 통과한 프라그먼트는 깊이 테스트, 스텐실 테스트 등의 테스트를 거쳐서 최종적으로 통과한 프라그먼트가 프라그먼트 연산 단계로 들어가게 된다. 이를 잘 활용하면 불필요한 연산을 줄일 수 있다. 프라그먼트 연산 단계에서는 생성된 프라그먼트가 최종 픽셀에 반영될 것인지를 결정하고 이전 픽셀과 합성 등이 이루어지는 단계이다.

이러한 GPU 파이프라인에 정점 셰이더(vertex shader)와 프라그먼트 셰이더(fragment shader)를 도입하여 파이프라인을 프로그램하는 것이 가능하게 되고(programmable hardware) 고수준의 셰이딩 언어(HLSL: high level shading language)가 제공되면서 개발이 용이해지고 더 복잡한 계산과 다양한 효과를 구현할 수 있다.

FDK 역전사 기법에서, 물체 공간의 모든 화소는 완전히 독립적으로 연산되기 때문에 동시에 연산이 가능하며 이점을 이용하여 GPU 파이프라인 중 프라그먼트 단계에서 다수의 화소에 대한 연산이 병렬적으로 이루어질 수 있다. 이러한 특징 때문에 FDK 역전사 기법은 GPU 파이프라인에 매우 적합하여 GPU 파이프라인을 이용하면 크게 성능을 향상시킬 수 있다. 또한 하나의 프라그먼트는 네 개의 색상 채널(red / green / blue / alpha, RGBA)을 출력할 수 있는데 이 점을 이용하여 한층 더 깊은 병렬처리가 가능하다.

3. 다중 GPU 기반 고속 삼차원 역전사 기법

FDK 역전사 기법은 투영영상을 입력 데이터로 하고 결과볼륨을 출력 데이터로 한 기법으로 볼 수 있다. 따라서 다중 GPU를 이용한다면 입력 데이터를 분산하거나 출력 데이터를 분할하여 수행할 수 있다. 물론 GPU의 메모리가 충분하여 입력 데이터와 출력 데이터를 모두 할당할 수 있으면 그것이 가장 이상적이겠지만 현실적으로 불가능하다. FDK 역전사 기법은 다음과 같이 표현될 수 있다.

$$S_r = \sum_{n=0}^{N_P-1} W(r)\hat{P}_\Phi(r) \tag{3}$$

식 (1)과 식 (2)에서 설명한 바와 같이 $\hat{P}_\Phi$는 투영영상에서 얻고자하는 값이고 r는 현재 화소 위치 벡터이다. $W(r)$는 r에 대한 가중치이며 N_P는 투영영상의 수이다. 이를 통해 결과볼륨에서 위치가 r인 화소의 값은 S_r 가 된다. 즉, 모든 투영영상에 따른 결과를 누적하여 S_r를 구하게 된다.

본 논문에서는 투영영상을 입력 데이터로, 결과볼륨을 출력 데이터로 간주하여 각각을 분산 또는 분할하는 두 가지 기법을 제안한다. 첫 번째 기법은 결과볼륨을 각 GPU에서 분할 생성하는 기법으로 r의 z 성분인 r_z를 기준으로 분할하여 S_r를 분할 생성하는 기법이고, 두 번째 기법은 투영영상을 각 GPU에 분할 적재하는 기법으로 N_P를 분할 적재하여 각각의 GPU로부터의 출력 결과 S_r를 생성한 후 CPU에서 산술적으로 합하는 기법이다. 두 가지 기법 모두 결과볼륨은 이차원 텍스쳐 스택으로 구성한다. (그림 3)

3.1 결과볼륨 분할생성 기법

본 기법은 순차적으로 입력 데이터인 투영영상을 한 장씩 받아 출력 데이터인 결과볼륨을 분할생성하는 기법이다(그림 4). 입력 데이터인 투영영상 P는 N_P장이 있고 출력 데이터인 결과볼륨 S는 N_S장으로 구성된다.

$$S_r^{g_n} = \sum_{n=0}^{N_P-1} W(r)\hat{P}_\Phi(r) \tag{4}$$

결과볼륨을 z축 방향으로 분할생성할 메모리를 각 GPU에 할당한다. $S_r^{g_n}$은 n번째 GPU에 할당될 결과이

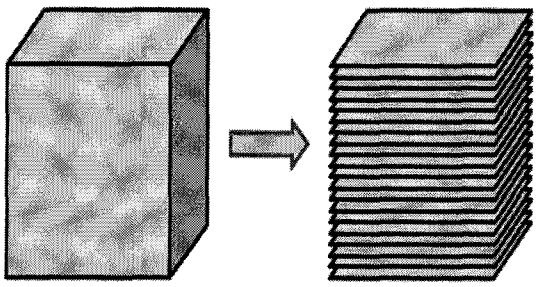


그림 3. 결과볼륨을 저장을 위한 이차원 텍스쳐 스택

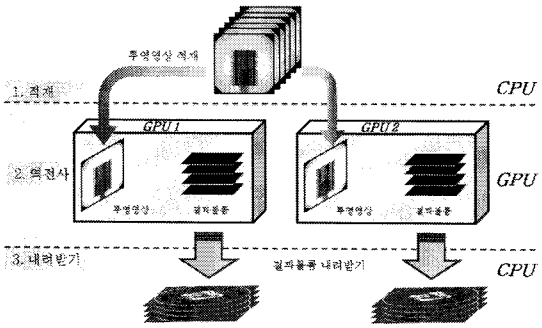


그림 4. 결과볼륨 분할생성 기법

다. 모든 GPU에 메모리 할당이 마치면 누적연산을 수행하기 위해 할당된 메모리를 모두 0으로 초기화 한다. 이후 모든 투영영상에 대해 투영영상 위치 등의 기하 정보를 갱신해가며 식 (4)에 따라 역전사 과정을 수행하여 누적한다. 모든 투영영상에 대한 연산이 완료되면 각각의 GPU로부터 CPU의 메인 메모리로 결과볼륨을 내려 받는다.

본 기법은 결과볼륨을 분할하여 생성하기 때문에 GPU의 수가 증가하는 만큼 성능 향상을 기대할 수 있다. 하지만 모든 GPU에 동일한 투영영상을 적재해야하기 때문에 GPU 수만큼 적재에 따른 부담이 증가하고 GPU에 결과볼륨 크기 이상의 메모리가 확보되어야 한다.

3.2 투영영상 분산적재 기법

본 기법은 투영영상을 각 GPU에 분산 적재하여 수행하는 기법(그림 5)으로 다음과 같이 표현된다.

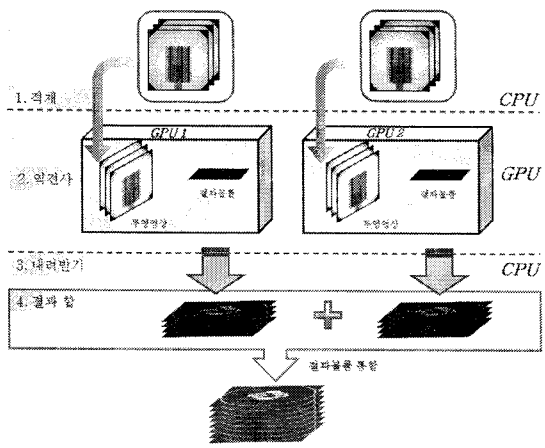


그림 5. 투영영상 분산적재 기법

$$S_r^{g_n} = \sum_{n=P_S(g_n)}^{P_E(g_n)} W(r)\widehat{P}_\Phi(r) \tag{5}$$

$$S_r = \sum_{n=0}^{N_g-1} S_r^{g_n} \tag{6}$$

$P_S(g_n)$은 현재 GPU g_n이 분담하는 시작 투영영상을 의미하고 $P_E(g_n)$은 마지막 투영영상을 의미한다. 각 GPU는 투영영상을 분산하여 적재하고 결과볼륨을 슬라이스 단위로 출력하기 위한 이차원 텍스처(S_{result})를 생성한다.

모든 텍스처의 적재 및 생성이 완료되면 역전사 단계로 넘어간다. 역전사 단계에서는 각 GPU에 적재된 투영영상의 역전사 결과를 S_{result}에 출력한다. 이 과정이 완료되면 각 GPU로부터 S_{result}를 메인 메모리로 내려 받고 각 결과값을 CPU에서 더하여 결과볼륨의 첫 번째 슬라이스가 완성된다(식 6). 역전사 연산이 교환법칙이 성립하기 때문에 단순히 합치는 연산이 가능하다. 이와 같은 과정을 기하 정보를 갱신해가며 결과볼륨 전체에 대해 반복한다.

본 기법은 결과볼륨 슬라이스를 하나씩 생성하고 메인 메모리로 바로 내려 받기 때문에 단일 GPU에서도 결과볼륨의 크기가 커지더라도 적용이 가능하다. 하지만 결과를 CPU에서 더하기 때문에 GPU의 수가 증가하거나 결과볼륨의 크기가 커지면 그만큼 결과를 통합하는 시간이 증가한다.

4. 실험 및 결과

실험에 사용된 그래픽카드는 NVIDIA GeForce 9800 GX2 512MB×2로 하나의 보드에 두 개의 그래픽카드가 장착된 장치이다. DirectX 10.0으로 개발하였다. 본 실험은 다중 GPU를 이용하여 단일 GPU의 경우보다 성능을 향상시키는 것을 목적으로 하기 때문에 GPU 상에서의 최적화 과정은 고려하지 않았다. 투영영상은 모두 필터를 적용한 결과[17]를 사용하였고 크기 정보는 표 1과 같다.

표 1의 실험 데이터를 가지고 제시한 결과볼륨 분할생성 기법과 투영영상 분산적재 기법을 단일 GPU와 다중 GPU를 이용하여 각각 적용하고 측정된 계산 시간을 비교 분석하였다. 측정된 계산시간은 객관성을 위해 동일 데이터에 대해 각각 10회를 수행하여 이들의 평균치를 기재하였다. 먼저, 데이터1을 가지

표 1. 실험 데이터

	투영영상	결과볼륨
데이터1	1000×1000×200	512×512×512
데이터2	1000×1000×400	512×512×512
데이터3	1000×1000×200	512×512×1024
데이터4	1000×1000×200	1024×1024×1024
데이터5	1000×1000×400	1024×1024×1024

표 2. 다중 GPU의 성능 향상 실험 (데이터1)

기법	결과볼륨 분할생성 (Cube)			
	준비	적재 및 역전사	내려받기	전체
단일	0.105	2.079	0.753	2.937
다중	0.149	1.292	0.626	2.067
기법	투영영상 분산적재 (Projection)			
	준비	적재 및 역전사	내려받기	전체
단일	0.084	0.284	2.582	2.950
다중	0.097	0.328	2.063	2.488

고 수행한 결과는 표 2와 같다.

실험 결과를 그림으로 나타내면 그림 6과 같다. 그림 6에서 괄호 안의 C는 결과볼륨 분할생성 기법을 뜻하고 P는 투영영상 분산적재 기법을 뜻한다. 그림 6에서 볼 수 있듯이 다중 GPU 기반 기법이 단일 GPU 기반 기법보다 좋은 성능을 보이는 것을 알 수 있다. 한 가지 눈여겨 볼 것은 단일 GPU 기반에서는 두 기법이 비슷한 성능을 보이나 다중 GPU 기반에서는 결과볼륨 분할생성 기법이 더 큰 성능 향상을 보인다는 것이다. 결과볼륨 분할생성 기법은 약 1.42배 향상되었고 투영영상 분산적재 기법은 약 1.18배 향상되었다. 이는 다중 GPU 기반 투영영상 분산적재 기법에서는 단일 GPU 기반에서와 달리, 결과 슬라이스를 각 GPU에서 생성 즉시 CPU의 메모리로 내려 받고 결과를 합치는 과정에서 모든 GPU가 멈춰있기 때문인 것으로 예측된다. 이로 인해 GPU의 수가 많아질수록 성능 향상을 저하시킬 수 있는 요인이 될 수 있다.

두 번째 실험은 더 많은 투영영상을 가진 데이터2를 이용하여 동일한 실험을 수행하였더니 표 3과 같은 결과를 얻었다.

실험에 사용된 그래픽카드 메모리 크기가 512MB이고 투영영상의 크기가 800MB인 관계로 단일 GPU 기반 투영영상 분산적재 기법은 실험이 불가능하였다. 하지만 첫 번째 실험을 통해 미루어보면 결과볼륨 분할생성 기법과 비슷한 결과가 나올 것을 예상할 수 있다. 이와 같이 가정하였을 때, 다중 GPU 기반에서 결과볼륨 분할생성 기법이 좋은 성능을 보였다. 이 역시 데이터1을 이용한 실험에서 말한 것과 같은 이유로 보인다.

세 번째 실험은 데이터3을 이용해 결과볼륨의 크기가 큰 경우에 대한 실험으로 결과는 표 4와 같다.

표 3. 다중 GPU의 성능 향상 실험 (데이터2)

기법	결과볼륨 분할생성 (Cube)			
	준비	적재 및 역전사	내려받기	전체
단일	0.126	3.884	0.870	4.880
다중	0.158	2.285	0.509	2.952
기법	투영영상 분산적재 (Projection)			
	준비	적재 및 역전사	내려받기	전체
단일	실험 불가			
다중	0.109	0.574	3.010	3.693

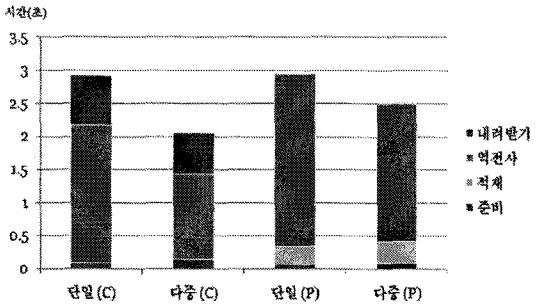


그림 6. 다중 GPU의 성능 향상 실험 (데이터1)

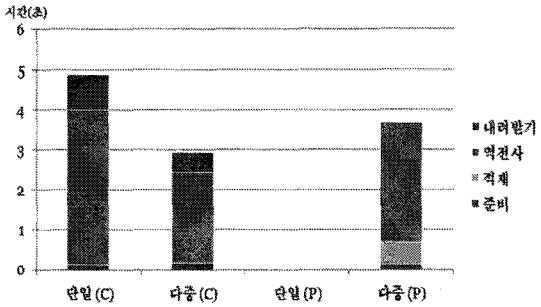


그림 7. 다중 GPU의 성능 향상 실험 (데이터2)

표 4. 다중 GPU의 성능 향상 실험 (데이터3)

기법	결과볼륨 분할생성 (Cube)			
	준비	적재 및 역전사	내려받기	전체
단일	실험 불가			
다중	0.164	2.179	1.270	3.613
기법	투영영상 분산적재 (Projection)			
	준비	적재 및 역전사	내려받기	전체
단일	0.083	0.276	5.035	5.394
다중	0.103	0.326	4.038	4.467

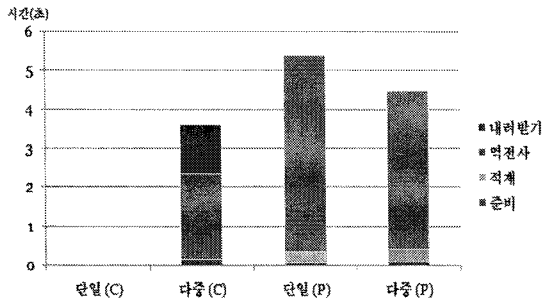


그림 8. 다중 GPU 성능 향상 실험 (데이터3)

결과볼륨의 크기가 512MB이기 때문에 단일 GPU 기반의 결과볼륨 분할생성 기법은 실험이 불가능하였다. 이번 실험에서도 다중 GPU 기반의 경우 결과볼륨 분할생성 기법이 더 우수한 것으로 나타났다. 지금까지 실험을 토대로 했을 때 다중 GPU 기반에서는 결과볼륨 분할생성 기법이 성능 면에서 우수한 것으로 나타났다. 이는 앞서 말한 대로, GPU가 쉬지 않고 서로 간섭 없이 연속적으로 작동하는 것이 가능하기 때문이다.

하지만 결과볼륨 분할생성 기법은 GPU에 결과볼륨을 모두 생성할 수 있는 메모리가 확보되어야 효율적으로 작동한다. 예를 들어, 데이터4와 같이 결과볼륨의 크기가 커서 GPU에 할당이 불가능하다면 결과볼륨을 분할하여 일부를 생성하여 CPU로 내려 받는 작업을 반복해야한다. 이 때, 결과볼륨을 분할한 만큼 투영영상이 반복되어 적재되기 때문에 그만큼 성능 저하가 발생할 수 있다. 반면에 투영영상 분할적재 기법은 결과볼륨의 크기와 상관없이 투영영상이 단 한번만 적재되기 때문에 앞서 말한 문제가 발생하지 않는다. 이는 표 2와 표 4의 결과를 통해 알 수 있다. 데이터3을 두 개로 분할하여 단일 GPU 기반

표 5. 결과볼륨 크기에 따른 두 기법 간의 성능 차이 비교 (데이터4)

기법	결과볼륨 분할생성 (Cube)			
데이터	준비	적재 및 역전사	내려받기	전체
1	0.105	2.079	0.753	2.937
4	0.105	2.079×8	0.753×8	22.656
기법	투영영상 분산적재 (Projection)			
	준비	적재 및 역전사	내려받기	전체
1	0.084	0.284	2.582	2.950
2	0.088	0.285	19.892	20.265

결과볼륨 분할생성 기법을 수행하면 표 2에서의 결과보다 대략 두 배가 걸릴 것으로 예상된다. 계산 복잡도를 계산하면, (준비 단계 시간) + [(적재 및 역전사 단계 시간)+(내려 받기 단계 시간)]×2 만큼의 시간이 들 것이고 이를 수치적으로 산술하면 0.105 + (2.079 + 0.753) × 2 = 5.769초이다. 이는 표 4의 단일 GPU 기반 투영영상 분산적재 기법의 5.394초보다 큰 시간이다. 이 차이는 결과볼륨의 크기가 커질수록 증가할 것이다. 이는 데이터4를 이용한 실험에서 다음과 같이 나타났다.

표 5의 결과는 모두 단일 GPU 기반에서 수행한 결과이다. 결과볼륨 분할생성 기법의 데이터4 실험은 산술적인 예상치이다. 실험 결과 표 5와 같이, 투영영상 분산적재 기법이 앞서 예상한대로 결과볼륨 분할생성 기법에 비해서 표 4보다 더 큰 차이로 좋은 결과를 보였다. 이를 통해 투영영상 분산적재 기법은 동일한 투영영상으로부터 그래픽카드의 메모리보다 큰 결과볼륨을 출력할 때 결과볼륨 분할생성 기법보다 유리한 것을 알 수 있다.

다음 실험은 투영영상과 결과볼륨이 모두 하나의 GPU에 할당이나 적재가 되지 않을 만큼 큰 데이터를 적용한 실험이다.

데이터5는 투영영상이 800MB이고 결과볼륨이 2GB이기 때문에 둘 중 어느 것도 하나의 그래픽카드에 할당이 될 수 없고, 다중 GPU 기반 결과볼륨 분할생성 기법에서도 두 개의 그래픽카드 메모리(1GB)보다 결과볼륨이 크기 때문에 실험이 불가능하고 다중 GPU 기반 투영영상 분산적재 기법만 실험이 가능하다. 표 3에서의 경우와 비교해봤을 때 거의 정비례로 수행시간이 증가하였다.

표 6. 큰 투영영상과 결과볼륨을 이용한 실험 (데이터5)

기법	결과볼륨 분할생성 (Cube)			
	준비	적재 및 역전사	내려받기	전체
단일	실험 불가			
다중	실험 불가			
기법	투영영상 분산적재 (Projection)			
	준비	적재 및 역전사	내려받기	전체
단일	실험 불가			
다중	0.103	0.596	38.462	39.161

그림 9는 제안한 다중 GPU기반 삼차원 재구성 기법을 적용하여 재구성된 임의의 단층 영상이고 그림 10은 재구성된 단층 영상들을 볼륨 렌더링 기법을 통하여 삼차원으로 가시화한 결과이다.

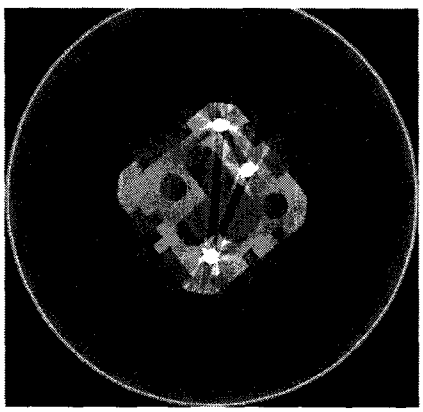


그림 9. 삼차원 재구성 단면 결과

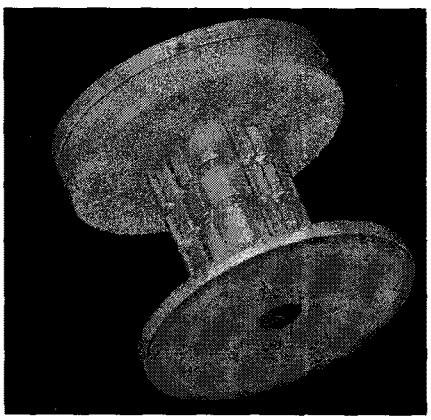


그림 10. 볼륨 렌더링 결과

5. 결론 및 향후 계획

본 논문에서는 다중 GPU 기반의 고속 역전사 기법을 적용하기 위해 두 가지 병렬처리 구현 기법에 대해 제안하였다. 결과볼륨을 분할하여 각 GPU에 할당한 후, 투영영상을 한 장씩 적재하면서 역전사를 수행하여 결과볼륨을 생성하는 구현기법과 투영영상을 각 GPU에 분산적재하여 각 GPU에서 할당 받은 입력 데이터에 대해 슬라이스 단위로 역전사를 수행한 부분 결과볼륨을 CPU에서 각각 내려 받은 후에 이 부분 결과볼륨들을 통합하여 최종 결과볼륨을 생성하는 구현 기법이다. 두 기법 모두 단일 GPU 기반의 경우보다 좋은 성능을 보였으나 결과볼륨 분할생성 기법의 경우 결과볼륨의 크기에 따라 약 1.4배에서 1.65배 정도 성능 향상을 보였고, 투영영상 분산적재 기법은 약 1.2배의 성능 향상을 가져왔다. 이는 투영영상 분산적재 기법에서는 결과볼륨 슬라이스를 합치는 과정에서 모든 GPU가 멈춰있기 때문에 결과볼륨 분할생성 기법이 우수한 성능을 보이는 것으로 추정된다. 하지만 결과볼륨의 크기가 1024^3과 같이 큰 결과볼륨을 생성해야 하는 경우 512^3의 크기인 결과볼륨을 생성할 때보다, 결과볼륨 분할생성 기법이 약 7.71배 수행시간이 증가한 반면 투영영상 분산적재 기법에서는 약 6.87배 수행시간이 증가하였다. 이는 결과볼륨 전체를 그래픽카드 메모리에 생성할 수 없기 때문에 결과볼륨 분할생성 기법을 적용하기 위해서는 결과볼륨을 그래픽카드 메모리에 적재할 수 있는 크기로 분할하여 생성하고 삼차원 재구성 과정을 분할한 수만큼 반복해야하기 때문이다. 따라서 결과볼륨의 크기가 그래픽카드 메모리보다 큰 경우에 결과볼륨이 커질수록 투영영상 분산적재 기법이 결과볼륨 분할생성 기법에 비해 좀 더 빠른 수행시간을 기대할 수 있다

향후 연구는 앞서 언급된 바와 같이 투영영상의 크기가 GPU 메모리를 넘어서는 경우와 본 논문에서 제시한 방법과 달리 각 GPU가 동일한 위상을 가지지 않고 다른 역할을 분담하여 부담을 조절하는 방법으로 효율을 극대화시킬 수 있는 방안에 대해서 고찰하고자 한다.

참 고 문 헌

[1] M. Levoy, "Display of Surfaces from Volume

Data," *Computer Graphics and Applications*, Vol.8, No.5, pp. 29-37. 1988.

[2] C. Axelsson and P.-E. Danielsson, "Three-Dimensional Reconstruction from Cone-Beam Data in $O(n^3 \log n)$ Time," *Physics in Medicine and Biology*, Vol.39, pp. 477-491, 1994.

[3] S. Basu and Y. Bresler, "An $O(n^3 \log n)$ Backprojection Algorithm for the 3D Radon Transform," *IEEE Transaction on Medical Imaging*, Vol.21, No.2, pp. 76-88, 2002.

[4] S. Buttler and M. I. Miller, "Maximum a Posteriori Estimation for SPECT Using Regularization Techniques on Massively Parallel Computers," *IEEE Transaction of Medical Imaging*, Vol.12, No.1, pp. 84-89, 1993.

[5] J. J. Fernandez, J.R. Bilbao-Castro, R. Marabini, J.M. Carazo, and I. Garcia, "Grid Computing in Structure Determination of Biological Specimens by Electron Microscope Tomography," *Lecture Notes in Computer Science*, Vol.2970, No.2004, pp. 171-181, 2004.

[6] J. R. Bilbao-Castro, J. M. Carazo, "Performance of Parallel 3D Iterative Reconstruction Algorithms," *12th Euromicro Conference on Parallel, Distributed and Network-Based Processing*, pp. 96-102, 2004.

[7] D. B. Keesing, J. A. O'Sullivan, D. G. Politte, B. R. Whiting, "Parallelization of a Fully 3D CT Iterative Reconstruction Algorithm," *Biomedical Imaging: Nano to Macro, 3rd IEEE International Symposium on*, pp. 1240-1243, 2006.

[8] F. Xu and K. Mueller, "Accelerating Popular Tomographic Reconstruction Algorithms on Commodity PC Graphics Hardware," *IEEE Transactions on Nuclear Science*, Vol.52, No.3, pp. 654-663, June 2005.

[9] N. Neophytou, F. Xu, and K. Mueller, "Hardware Acceleration vs. Algorithmic Acceleration: Can GPU-based Processing Beat Complexity Optimization for CT?," *SPIE Medical Imaging 2007*, 2007.

[10] K. Chidlow and T. Moller, "Rapid Emission Volume Reconstruction," *Volume Graphics Workshop*, pp. 15-26, 2003.

[11] B. Cabral, N. Cam, and J. Foran, "Accelerated Volume Rendering and Tomographic Reconstruction Using Texture Mapping Hardware," *Symposium on Volume Visualization*, pp. 91-98, 1994. Vol.2970, No. 2004, pp. 171-181, 2004.

[12] L. A. Feldkamp, L. C. Davis, and J. W. Kress, "Practical Cone Beam Algorithm," *Journal of the Optical Society of America A: Optics, Image Science, and Vision*, Vol.1, No.6, pp. 612-619, 1984.

[13] *Wikipedia*, http://en.wikipedia.org/wiki/Scalable_Link_Interface.

[14] *Wikipedia*, http://en.wikipedia.org/wiki/ATI_CrossFire.

[15] I. Buck, "Data parallel computing on graphics hardware," *Graphics Hardware Panel Talk*, 2003.

[16] D. Blythe, "The Direct3D 10 System," *ACM Transactions on Graphics*, Vol.25, No.3, pp. 724-734, 2006.

[17] A. Lukin, "Tips & Tricks : Fast Image Filtering Algorithms," *GraphiCon'2007*, 2007.

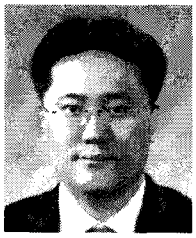
이 병 훈

2005년 2월 성균관대학교 정보통신공학부 학사

2007년 2월 서울대학교 컴퓨터공학부 석사

2007년 3월~현재 서울대학교 컴퓨터공학부 박사과정

관심분야 : 컴퓨터 그래픽스, 하드웨어 기반 렌더링, 삼차원 재구성



이 호

2000년 8월 숭실대학교 전자전지정보통신공학부 학사
2002년 8월 숭실대학교 정보통신공학과 석사
2002년 10월~2003년 3월 (주)인피니트테크놀로지 연구원
2009년 2월 서울대학교 컴퓨터공학부 박사
관심분야 : 영상정합, 의료영상처리, 컴퓨터 그래픽스, GPGPU

계 희 원

1999년 2월 서울대학교 전산과학과 학사
2001년 2월 서울대학교 컴퓨터공학부 석사
2005년 8월 서울대학교 컴퓨터공학부 박사
2006년 1월~2007년 3월 서울대학교 컴퓨터연구소 연구원
2007년 4월~2007년 8월 인하대학교 연구교수
2007년 9월~현재 한성대학교 정보시스템공학과 전임강사
관심분야 : 볼륨 가시화, 실시간 렌더링, 대용량 영상처리

신 영 길

1982년 2월 서울대학교 계산통계학과 학사
1984년 2월 서울대학교 계산통계학과 석사
1990년 2월 미국 USC 전산학과 박사
1990년 2월~1992년 2월 경북대학교 전자계산학과 전임강사
1992년 3월~현재 서울대학교 컴퓨터공학부 교수
관심분야 : 의료영상처리, 컴퓨터 그래픽스, 하드웨어기반 렌더링