# 범용 그래픽스 하드웨어 기반 여과후 역투사 최적화 기법에 관한 연구

## An Optimized GPU based Filtered Backprojection method

박종현, Jonghyun Park*, 이병훈, Byeonghun Lee**, 이호, Ho Lee***, 신영길, Yeong Gil Shin**

**요약** 삼차원 재구성 기법은 대상을 파괴하지 않고도 그 내부 구조의 공간적 해석을 가능하게 해주는 단층 영상을 생성해주기 때문에, 산업, 의료분야에서 널리 사용되고 있다. 최근 영상 장비의 성능 향상으로 고해상도의 CT 영상을 얻을 수 있게 되었으나, 대용량 데이터를 재구성하기 위해 많은 시간이 소요된다. 본 논문에서는 재구성에서 가장 많은 시간이 소요되는 여과와 역투사 과정을 범용 그래픽스 하드웨어를 사용하여 최적화하는 방법을 제안한다. 여과에서는 네 장의 영상을 압축하여 동시에 처리하는 기법을 적용하고, 역투사 과정에서는 깊이 테스트를 이용하여 계산량을 줄이는 방법을 사용한다. 제안된 방법으로 구현된 GPU 기반 프로그램은 OpenMP 를 사용하여 최적화 된 CPU 기반 프로그램에 비해 약 50 배 이상 속도가 향상 되었다.

**Abstract** Tomography images reconstructed from conebeam CT make it possible to observe inside of the projected object without any damage, and so it has been widely used in the industrial and medical fields. Recent advanced imaging equipment can produce high-resolution CT images. However, it takes much time to reconstruct the obtained large dataset. To reduce the time to reconstruct CT images, we propose an accelerating method using GPU (graphics processing unit). Reconstruction consists of mainly two parts, filtering and back-projection. In filtering phase, we applied 4ch image compression method and in back-projection phase, computation reduction method using depth test is applied. The experimental results show that the proposed method accelerates the speed 50 times than the CPU-based program optimized with OpenMP by utilizing the high-computing power of parallelized GPU.

***핵심어: Reconstruction, GPU, Filtering, FFT, compression***

본 연구는 서울시 산학연 협력사업(10888), 지식경제부 중기거점 기술개발사업(10028331-2008-23), 두뇌한국 21 지원 사업, 서울대 컴퓨터 연구소(0421-20080066) 지원으로 수행되었음.

*주저자 : 서울대학교 컴퓨터공학과 석사과정 e-mail: pjh_xp@cglab.snu.ac.kr

**공동저자 : 서울대학교 컴퓨터공학과 교수 e-mail: yshin@snu.ac.kr

**공동저자 : 서울대학교 컴퓨터공학과 박사과정 e-mail: intellee@cglab.snu.ac.kr

***교신저자 : 서울대학교 컴퓨터공학과 박사과정 e-mail: holee@cglab.snu.ac.kr

## 1. 서론

삼차원 재구성(Reconstruction)이란, 물체를 중심으로 일정 각도 간격으로 회전하며 얻은 투영 영상으로부터, 공간적 해석이 가능한 단층영상들을 생성하는 것을

말한다. 이렇게 생성된 삼차원 영상에 물체 내부를 보여주는 볼륨 렌더링(volume rendering)[1] 기법을 적용하면, 물체를 파괴하지 않고도 내부 구조를 정밀하게 관찰 할 수 있다. 이러한 이점으로 인해 삼차원 재구성 방법은 여러 산업, 의료 분야에 걸쳐 널리 사용되고 있다.

CT 영상의 다양한 활용성과 수요에 힘 입어 최근 더 크고 선명한 투영 영상을 얻을 수 있는 영상장비가 개발되고 있다. 영상장비로부터 얻은 데이터가 크고 정밀할 수록, 재구성 된 단층영상도 정밀해지지만, 재구성 과정의 소요시간이 기하급수적으로 증가하는 문제가 있다. 이에 더해, 산업, 의료 분야에서는 CT 영상을 실시간으로 재구성 하는 기술을 요구한다. 예를 들어 영상 유도 수술(image-guided surgical intervention), 또는 시변 데이터(time-varying data)를 이용한 실시간 진단(real time diagnosis)등의 기술이 개발되고 있는데, 이러한 것들은 실시간 재구성 기술을 전제로 한다.

이러한 필요에 따라 재구성 시간을 단축 시키기 위해 한편에서는 알고리즘 상에서 복잡도를 낮추는 방향으로 연구가 진행되고 있다. 그 결과 주파수 영역에서 재구성 연산을 수행하여, 시간 복잡도를 $O(n^4)$ 에서 $O(n^3 \log n)$으로 떨어뜨리는 알고리즘[2][3]이 발표되었다. 그러나 이 방법은 고주파 영역에서 발생하는 잡음(noise)을 줄이기 위해 과표본추출(over-sampling)을 하는 과정에서 오히려 전체 계산량을 증가시킨다.

다른 한편으로는 병렬처리 가능한 하드웨어를 활용하여 실행 속도를 향상 시키는 연구가 진행되어 왔다. 다수의 컴퓨터를 사용하여 분산 처리[4] 하거나, SIMD(Single Instruction Multiple Data)를 지원하는 고성능 CPU 를 사용하는 방법도 제안되었다. 이러한 방법에는 상당한 속도 향상이 있지만, 하드웨어의 비싼 가격 때문에 연구와 활용이 제한된다는 단점이 있다. 또한 일단 구현된 뒤에는 알고리즘 상의 보수가 어려워 유연성이 떨어진다는 치명적인 단점도 있다. 그래서 최근에는 비교적 저가의 범용 그래픽스 하드웨어(Graphics Processing Unit : GPU)를 이용하여 시간을 단축 시키는 연구[5-8]가 활발하게 진행되고 있다.

본 논문에서는 재구성 알고리즘 중에서 빠르고 간단하여 가장 널리 쓰이는 FDK 여과후 재구성 과정(Filtered Backprojection)[9]을 GPU 에서 구현하는 최적화된 방법을 제안한다. 재구성 과정에서 가장 많은 시간이 소요되는 두 가지 과정 중 하나인 여과처리(filtering) 단계에서는 네 장의 영상을 압축하여 동시에 처리하고, 또 다른 하나인 역투사(backprojection) 과정에서는 깊이 테스트를 이용하여 전체 연산량을 줄이는 방법을 사용하여 속도를 향상 시켰다.

2 장 에서는 그래픽스 파이프 라인(graphics pipe line)과 GPU 프로그래밍을 간략하게 설명한다.

3 장 에서는 여과 처리 단계에서 네 장의 실수 영상을 하나의 벡터(vector) 영상으로 압축하여 처리하는 방법과 역투사 과정에서 GPU 의 깊이 테스트 기능을 통해 불필요한 계산을 제거하여 속도를 향상 시키는 방법을 소개한다.

4 장에서는 제안한 GPU 기반 방법과 OpenMP 를 이용하여 최적화 시킨 CPU 기반 프로그램 등을 비교하여 성능을 평가하였다. 마지막 5 장에서는 소개된 가속 기법을 요약하고 끝을 맺는다.

## 2. 프로그램 가능한 GPU 파이프 라인

1 세대 GPU 의 경우, 렌더링(rendering)을 하기 위한 모든 단계가 최적화된 고정 파이프라인(fixed pipeline)으로 제공되었다. 그러나 2 세대 GPU 부터는 프로그래밍 가능한 파이프라인 (programmable pipeline)을 제공하여 렌더링 과정을 사용자가 조작하거나, 그래픽스 작업이 아닌 일반적인 연산 용도로 GPU 의 computing power 를 사용하는 것이 가능해졌다.

GPU 에서 실행되는 명령어 집합 (instruction set)을 쉐이더(Shader)라 한다. 쉐이더 언어(Shader language)는 쉐이더를 작성하기 위해 사용하는 프로그래밍 언어를 가리키며, 대표적으로 GLSL, Cg, HLSL 등이 있다. 사용자가 쉐이더 언어로 작성한 쉐이더 코드는 쉐이더 컴파일러(compiler)에 의해 컴파일 되고, GPU 메모리에 적재되어 사용자가 원하는 연산을 실행하게 된다.

쉐이더는 기본적으로 그래픽스 처리를 위한 것이므로, 기하처리(geometry processing), 래스터 변환(rasterization), 프래그먼트 연산(fragment processing)의 세 단계로 구성되는 파이프라인과 밀접한 관련이 있다. 버텍스 쉐이더(Vertex shader), 지오메트리 쉐이더(Geometry shader), 픽셀 쉐이더(Piexl shader), 이상의 3 가지로 나뉘는 쉐이더에서 전술한 파이프라인의 각 단계에서 수행할 작업을 정의한다. (쉐이더의 구분은 MicroSoft 사의 Direct3D 10(이하 DX10) 기준이며, 지오메트리 쉐이더는 DX10 에서 새로이 추가된 것이다.)

버텍스 쉐이더 코드가 실행되는 기하처리 단계에서는 사용자가 정의한 정점(Vertex)을 입력 받아, 화면 좌표계로 변환된 정점 좌표를 출력한다. 입력으로 받는 정점에는 좌표뿐만 아니라, 색상, 법선(Normal) 벡터나 그 밖에 사용자가 정의하는 정보를 추가로 포함 할 수 있다. 프로그래밍 가능한 파이프라인에서는 이와 같은 변환 과정을 정의하는 버텍스 쉐이더를 이용하여, 입력된 정점 좌표와 변환 행렬들의 단순 곱이 아닌, 사용자가 원하는 좌표와 정보를 가진 정점을 출력할 수 있다.

래스터 변환 단계는 기하처리의 결과 변화된 점들로 정의되는 도형(primitive)을 입력으로 한다. 이 단계에서는 벡터 형태(vector format)로 정의된 도형이 스크린에서 차지하는 픽셀(pixel)들의 집합, 즉 래스터 이미지(raster image)로 변환한다. 래스터 이미지를 이루는 픽셀들은 버텍스 쉐이더에 의해 변환된, 정의된 도형을 이루는 정점들로부터의 선형 보간(linear interpolation)된 정보를 담고 있다.

마지막으로 프래그먼트 연산과정은 이전 과정, 래스터 변환의 출력인 래스터 이미지들을 입력으로 받아 렌더 타겟(Render Target), 또는 백 버퍼(back buffer)라 불리는 메모리 공간에 최종 결과를 기록한다. 이 과정에서 래스터 이미지를 구성하는 개개의 픽셀들이 다른 래스터 이미지에 의해 가려지는지 판단하는 깊이 테스트(depth test)도 이루어진다. 이 때, 사용자 정의에 따라, 시점에서 먼 픽셀 값들을 버리거나, 또는 투명한 효과를 내기 위해 이전 값들과 합성이 이루어진다. 또한 미리 저장된 텍스쳐(texture)의 지정된 위치에서 값을 읽어와 결과에 반영하는 텍스쳐 매핑(texture mapping)도 수행된다. 픽셀 쉐이더에 이 단계에서 수행될 작업들을 정의하고, 미리 정의된 크기의 2D 또는 3D 배열 형태의 GPU 메모리 공간에 최종 결과를 저장할 수 있다. 그리고 나서 저장된 결과를 스크린에 시연(display)하는 대신, CPU 메모리로 복사해 오는 방법으로 범용 GPU 프로그래밍이 가능하다.

설명한 바와 같이, 파이프라인에서 이전 단계의 출력은 다음 단계의 입력이 된다. 이러한 관계는 기하처리에서, 래스터 변환, 프래그먼트 연산 순으로 한 방향으로만 성립한다. 따라서 FFT 와 같은 순차적인 작업을 수행하기 위해서는 매 단계마다 이전 단계에서의 결과를 텍스쳐로 지정하고, 다시 기하처리 단계에서 시작하는 불필요한 연산을 반복해야 한다. DX10 에서 추가된 지오메트리 쉐이더는 픽셀 쉐이더, 즉 프래그먼트 연산 과정을 거친 최종 결과를 다시 이전 파이프라인 단계를 모두 거치지 않고 입력으로 활용하는 효율적인 방법을 제시한다. 그러나 본 논문에서는 제안하는 방법은 지오메트리 쉐이더를 사용하지 않으므로 자세한 설명은 생략한다.

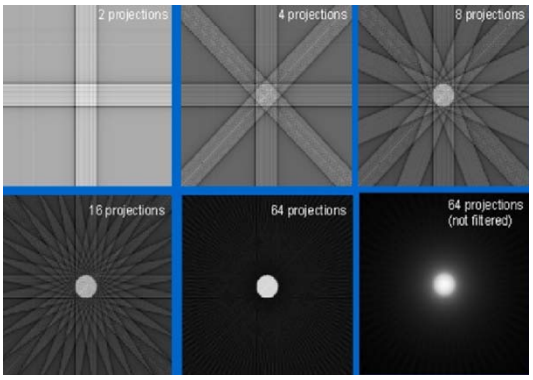
Intel 사의 최신 CPU 가 4 개의 코어(core)를 갖는데 반해, Nvidia 사의 비교적 최신 모델인 G80 계열의 GPU 는 128 개의 코어를 내장하고 있다. 이에 더해 각각의 코어 내부에서도 GPU 는, CPU 와 비교하여 볼 때, 프로그램의 실행을 제어하는 Control Unit 보다 데이터 연산을 담당하는 ALU(Arithmetic Logic Unit)에 더 많은 트랜지스터 (transistor)를 할당하는 구조로 설계되었다. 따라서 GPU 는 Control Unit 의 역할이 중요해지는 분기문(branch) 사용이 잦은 순차 프로그램에는 취약하지만, SIMD processor 로서 대형 데이터에 대한 반복 연산에서 CPU 에 구조적 우위를 갖는다. 또한 전술한 방법으로 범용 프로그래밍이 가능하기 때문에, 이 논문의 주제인 CT 삼차원 재구성에 훌륭한 해결책이 될 수 있다.

## 3.GPU 기반 삼차원 재구성

### 3.1 FFT 를 이용한 여과

1917 년 독일의 J. Radon[10]은 피사체를 중심으로 회전하며 얻은 투영 영상으로부터 피사체의 내부 구조를 보여주는 이차원 영상을 재구성 할 수 있음을 증명하는 논문을 발표하였다. 하지만 제한된 수의 투영 영상으로부터 재구성된 영상에서는 X 선의 감쇄가 크게 일어나는 특이점 근처에 방사형의 줄무늬(streak artifact)가 나타난다.

재구성 영상의 정확성과 이해도를 떨어뜨리는 이런 잡음(noise)를 제거하기 위해, FDK(Feldkamp, Davis and Kress) 역투사 알고리즘은 여과된 투영 영상을 바탕으로 삼차원 영상을 재구성한다. 아래 〈그림 3.1-1〉은 왼쪽 위부터 투영 영상의 개수가 두 개에서 예순 여섯 개까지 늘어감에 따라 재구성 영상이 선명해지는 것을 보여준다. 또한 투영 영상에 따라 줄무늬 형태의 잡음이 나타나는 것도 확인 할 수 있다. 두 번째 줄의 가운데 영상과 오른쪽 끝 영상은 여과를 거친 것과 그렇지 않은 것 사이에 선명함의 차이를 보여준다.




〈그림 3.1-1〉 투영 영상의 개수 변화와 여과에 따른 재구성 영상 화질 비교

일반적으로 공간영역(spatial domain)에서 정의된 $M \times N$ 크기의 영상을 여과할 경우 $O((M \times N)^2)$ 의 시간 복잡도가 요구되나 주파수 영역(frequency domain)에서 이를 수행할 경우 복잡도는 $O(M \times N)$ 으로 줄어든다. 주파수 영역에서 여과하기 위해, 공간 영역에서 정의된 투영 영상을 푸리에 변환(Fourier transform)을 사용하여 주파수 영역으로 변환한다.

푸리에 해석(Fourier analysis)은 모든 함수가 무한개의 정현파(sinusoidal wave)의 합으로 표현 할 수 있음을 보여준다. $N$ 개의 원소를 갖는 이산 함수(discrete function) $f(x)$ 의 경우도 연속함수와 마찬가지로, 같은 수의 원소를 갖는 이산 함수 $F(u)$로 정의될 수 있다. 이산 푸리에 변환식(Discrete Fourier Transform : DFT)은 동일한 문제 공간을 나타내는 두 함수 $F(u), f(x)$ 사이의 관계를 아래의 수식 (3.1-1), (3.1-2)와 같이 표현 한다. 푸리에 변환 식을 영상에 적용시키면, $f(x)$ 는 좌표에 따라 다른 색상(color), 밝기(brightness) 값을 갖는 영상 자체에, 샘플링(sampling) 수를 의미하는 $N$ 은 영상의 크기 즉 화소(pixel) 수에 대응된다.

$$F(u) = \frac{1}{N}\sum_{x=0}^{N-1} f(x) \cdot W_N^{xu} \qquad (3.1\text{-}1)$$

$$f(x) = \sum_{u=0}^{N-1} F(u) \cdot W_N^{-xu} \qquad (3.1\text{-}2)$$

$$W_N^{xu} = e^{-j2\pi xu/N} = cos(2\pi xu/N) - j \times sin(2\pi xu/N)$$

(3.1-3)

1965 년 Cooley 와 Tukey 는 이산 푸리에 변환을 효율적으로 계산하는 고속 푸리에 변환(Fast Fourier Transform : FFT) 알고리즘[11]을 발표 하였다. 고속 푸리에 변환 알고리즘은 이산 푸리에 변환을 작은 문제들의 합으로 나누어 중복되는 연산 결과를 재활용하는 동적 프로그래밍(dynamic programming) 방법으로 전체 연산량을 감소시킨다. 이러한 방법으로 고속 푸리에 변환은 이산 푸리에 변환의 시간 복잡도 $O(n^2)$ 을 $O(n \log n)$으로 낮추어 준다.

[Moreland, K., Angel, E. 2003][12]은 푸리에 변환은 복소수를 사용하지만, 일반적으로 영상은 실수 값만을 갖는 점에 착안하여 주파수 압축(frequency compression) 방법을 제안하였다. 즉, 일차원 고속 푸리에 변환을 위해 실수 영상을 복소수로 변환 할 때, 〈그림 3.1-2〉의 왼쪽과 같이 허수부를 '0' 으로 초기화 하는 대신, 오른쪽 그림처럼 또 다른 영상의 값을 넣어 하나의 복소수가 두 개의 실수 함수를 포함하게 하는 것이다.

고속 푸리에 변환은 선형(linear)이므로 압축된 영상은 주파수 영역으로 변환된 뒤 간단한 수식을 거쳐 각각의 결과로 분리 복원된다.

| Real | Image |
|---|---|
| f(0) | 0 |
| f(1) | 0 |
| f(2) | 0 |
| f(3) | 0 |
| f(4) | 0 |
| f(5) | 0 |
| f(6) | 0 |
| f(7) | 0 |

compression

| Real | Image |
|---|---|
| f(0) | g(0) |
| f(1) | g(1) |
| f(2) | g(2) |
| f(3) | g(3) |
| f(4) | g(4) |
| f(5) | g(5) |
| f(6) | g(6) |
| f(7) | g(7) |

〈그림 3.1-2〉

수식 (3.1-3)은 〈그림 3.1-2〉처럼 두 실수 함수를 하나의 복소수로 합성하는 것을 의미하며, 수식 (3.1-4)는 전 식에서 합성된 복소수를 이산 푸리에 변환 하는 것을 나타낸다. 수식 (3.1-5,6)에서 $F(u)r$ 과 $F(u)i$ 는 각각 $F(u)$ 의 실수부와 허수부를 가리킨다. 수식(3.1-3)과 (3.1-7)에서와 같이 $W_N^{x(N-u)}$는 $W_N^{xu}$의 켤레 복소수(pair of conjugate complex) 이므로 수식 (3.1-5)이 성립함을 쉽게 증명 할 수 있다. $u = 0$ 이거나 $u = N/2$ 인 경우, $H(u) = H(N-u)$ 이므로 수식 (3.1-6)이 성립한다.

$$h(x) = f(x) + j \times g(x) \quad (3.1\text{-}3)$$

$$H(u) = \frac{1}{N}\sum_{x=0}^{N-1}\{f(x) + j \times g(x)\}W_N^{xu} \quad (3.1\text{-}4)$$

$$when\ 0 < u < N,\ u \neq N/2 \quad (3.1\text{-}5)$$

$$F(u)r = \frac{1}{2}(H(u)r + H(N-u)r)$$

$$F(u)i = \frac{1}{2}(H(u)i - H(N-u)i)$$

$$G(u)r = \frac{1}{2}(H(u)i + H(N-u)i)$$

$$G(u)i = \frac{1}{2}(-H(u)r + H(N-u)r)$$

$$when\ u = 0\ or\ N/2 \quad (3.1\text{-}6)$$

$$F(u)r = H(u)r, \quad F(u)i = 0$$

$$G(u)r = H(u)i, \quad G(u)i = 0$$

$$W_N^{x(N-u)} = e^{-j2\pi x(N-u)/N} \quad (3.1\text{-}7)$$
$$= cos(2\pi x(N-u)/N) - j \times sin(2\pi x(N-u)/N)$$
$$= cos(2\pi xu/N) + j \times sin(2\pi xu/N)$$

FDK 역투사 알고리즘은 회전축과 평행한 방향으로만 여과를 적용하므로 일차원 푸리에 변환만을 사용한다. 변환된 영상에 수식(3.1-5,6)을 적용하면, 주파수 영역으로 변환된 영상을 공간 영역으로 변환하는 역 푸리에 변환 (Inverse Fourier Transform) 과정에서는 압축하지 않았을 때와 같은 메모리 공간과 연산량이 필요하다. 따라서 압축된 영상을 곧바로 여과하고 역 푸리에 변환한 결과에서 올바로 여과 처리된 각각의 영상을 분리해 낼 수 있다면, 일차원 푸리에 변환에서도 주파수 압축의 효과를 얻을 수 있다.

본 논문에서는 일련의 수식 (3.1-8)~(3.1-13)를 통해 분리 복원 없이 여과 후 역 푸리에 변환한 결과와, 압축 없이 두 개의 함수를 각각 처리한 결과가 같음을 보인다. 수식 (3.1-9,10)은 두 개의 함수 $f(x)$ 와 $g(x)$ 를 각각 여과한 결과를 보여준다. 압축된 영상을 여과한 결과의 실수부를 나타내는 수식 (3.1-13)과 수식 (3.1-10), 그리고 허수부에 해당하는 수식 (3.1-14)와 수식 (3.1-11)이 일치한다. 따라서 일차원 푸리에 변환만을 사용하는 여과에서는 압축된 영상을 분리 복원 없이 여과해도 원하는 결과를 얻을 수 있다.

$$F'(u) = F(u) \times filter(u) \quad (3.1\text{-}8)$$

$$G'(u) = G(u) \times filter(u)$$

$$H'(u) = H(u) \times filter(u) = F'(u) + jG'(u)$$

$$W_N^{-xu} = cos(2\pi xu/N) + j\,sin(2\pi xu/N) = R_u^x + jI_u^x$$

$$f'(x) = \sum_{u=0}^{N-1}\{F'(u) \times (R_u^x + jI_u^x)\} \quad (3.1\text{-}9)$$
$$= \sum_{u=0}^{N-1}[R_u^x F'(u)r - I_u^x F'(u)i + j\{R_u^x F'(u)i + I_u^x F'(u)r\}]$$

$$f'(x)r = \sum_{u=0}^{N-1}\{R_u^x F'(u)r - I_u^x F'(u)i\}$$

$$f'(x)i = \sum_{u=0}^{N-1}\{R_u^x F'(u)i + I_u^x F'(u)r\} = 0$$

$$g'(x) = \sum_{u=0}^{N-1}\{G'(u) \times (R_u^x + jI_u^x)\} \quad (3.1\text{-}10)$$

$$= \sum_{u=0}^{N-1}[R_u^x G'(u)r - I_u^x G'(u)i + j\{R_u^x G'(u)i + I_u^x G'(u)r\}]$$

$$g'(x)r = \sum_{u=0}^{N-1}\{R_u^x G'(u)r - I_u^x G'(u)i\}$$

$$g'(x)i = \sum_{u=0}^{N-1}\{R_u^x G'(u)i + I_u^x G'(u)r\} = 0$$

$$h'(x) = \sum_{u=0}^{N-1}[\{F'(u) + jG'(u)\} \times (R_u^x + jI_u^x)] \quad (3.1\text{-}11)$$

$$= \sum_{u=0}^{N-1}[\{R_u^x F'(u) - I_u^x G'(u)\} + j\{R_u^x G'(u) + I_u^x F'(u)\}]$$

$$= \sum_{u=0}^{N-1}\{R_u^x F'(u)r - I_u^x F'(u)i - R_u^x G'(u)i - I_u^x G'(u)r\} +$$

$$j\sum_{u=0}^{N-1}\{-I_u^x F'^{(u)}r + R_u^x F'^{(u)}i - I_u^x G'^{(u)}i + R_u^x G'(u)r\}$$

$$h'(x)r = \sum_{u=0}^{N-1}\{R_u^x F'(u)r - I_u^x F'(u)i - R_u^x G'(u)i - I_u^x G'(u)r\}$$

$$= \sum_{u=0}^{N-1}\{R_u^x F'(u)r - I_u^x F'(u)i\}$$

$$= f'(x)r \quad (3.1\text{-}12)$$

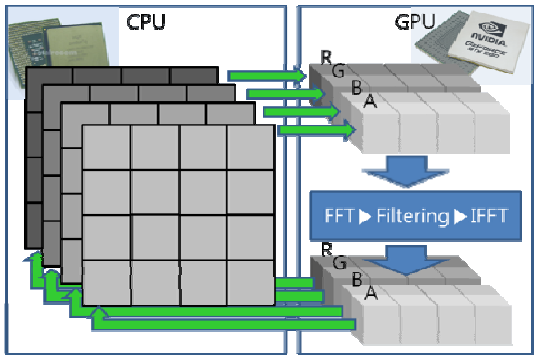
$$h'(x)i = \sum_{u=0}^{N-1}\{R_u^x G'(u)r - I_u^x G'^{(u)}i - I_u^x F'^{(u)}r + R_u^x F'^{(u)}i\}$$

$$= \sum_{u=0}^{N-1}\{R_u^x G'(u)r - I_u^x G'(u)i\}$$

$$= g'(x)r \quad (3.1\text{-}13)$$

GPU 에는 그래픽스에서 일반적으로 사용되는 네 채널, RGBA(red, green, blue, alpha)을 동시에 처리할 수 있는, 벡터 처리(vector processing) 하드웨어가 구현되어있다.

본 논문에서는 전술한 압축 방법과 GPU 의 이러한 장점을 이용하여, 각각의 색상 채널에 서로 다른 함수를 저장하면, 네 개의 영상을 동시에 여과 처리하는 방법을 제안한다. 각각의 색상 채널에 네 개의 함수를 저장하고, 이를 두 복소수로 다루어 일차원 푸리에 변환을 진행하는 방법이다. 즉 GPU 상에서는 한번에 두 개의 복소수 연산을 진행하지만, 실제로 각 복소수는 두 개의 함수를 압축한 것이므로 모두 네 개의 함수가 한번에 처리되는 것이다.

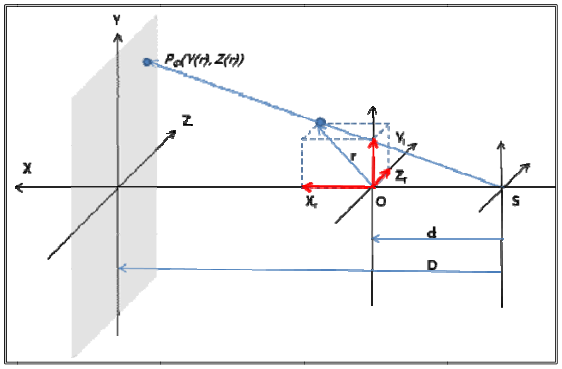



〈그림 3.1-3〉 GPU 를 이용한 4 채널 압축 FFT

## 3.2 FDK 를 이용한 역투사 기법

FDK 역투사 알고리즘은 피사체를 둘러싼 직육면체 형태의 공간을 나타내는 삼차원 배열로 된 역투사 공간(volume)을 정의한다. 그리고 모든 투영 영상으로부터 볼륨의 각 점(voxel)에 대응되는 지점의 값을 얻어와 누적시켜 나간다.

수식(3.2-1)은 역투사 공간의 한 점 $(X_r, Y_r, Z_r)$로부터 이에 대응 하는 투영 영상 위의 점$(Y(r),\ Z(r))$의 좌표를 구하는 과정을 나타낸다. $\theta$만큼 회전한 투영 영상의 경우, $(X_r, Y_r, Z_r)$을 $-\theta$ 만큼 회전하여 얻은 좌표$(X'_r, Y_r, Z'_r)$에 대응 하는 점을 구할 수 있다. (회전축은 역투사 공간의 중심$(0,0,0)$을 지나며, 〈그림 3.2-1〉에서는 Y 축과 같다. 따라서 변환된 좌표의 y 값은 변하지 않는다.) 이렇게 얻어온 값에 가중치를 적용하고 이 과정을 모든 투영 영상에 대하여 반복, 누적(accumulation)하면 역투사 공간 상의 한 점의 값을 얻을 수 있다. 수식(3.2-4)에서 0 에서 $2\pi$까지의 적분은 모든 투영 영상에 대해서 역투사 연산의 결과를 누적함을 의미한다.




〈그림 3.2-1〉 역투사 모델

FDK 알고리즘은 다음의 두 가지 감쇄(attenuation) 모델 가정한다. 첫째는 깊이 감쇄로, x-ray 가 물체를 통과할 때 물체 내부의 특이성에 관계없이 빔 소스로부터 감지기까지의 거리에 따라서만 규칙적으로 감쇄한다는 것이다. 두 번째는 원형 감쇄로 빔 소스로부터 멀어질수록 감지기에 측정되는 에너지의 세기가 감쇄한다는 것이다. 감쇄로 인한 오차를 보정하기 위해 FDK 역전사 알고리즘은 깊이 가중치(depth weight)와 원형 가중치 (circular weight)를 둔다. 수식(3.2-2)는 원형 가중치를 나타내고, (3.2-4)에서 $\frac{d^2}{(d+X_r)^2}$항은 깊이 가중치를 나타낸다.

$$Y(r) = \frac{Y_r}{d+X_r} \times D, \quad Z(r) = \frac{Z_r}{d+X_r} \times D \quad (3.2\text{-}1)$$

$$Cw\big(Y(r), Z(r)\big) = \frac{D}{\sqrt{D^2+Y(r)^2+Z(r)^2}} \quad (3.2\text{-}2)$$

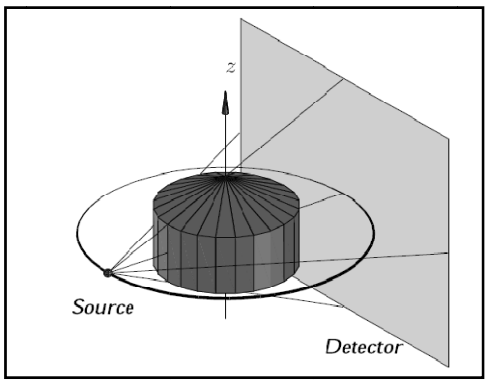
$$P'_\varphi(Y(r), Z(r)) = Cw\big(Y(r), Z(r)\big)P_\varphi(Y(r), Z(r))g(Y(r)) \quad (3.2\text{-}3)$$

$$f(r) = \frac{1}{4\pi}\int_0^{2\pi}\left\{\frac{d^2}{(d+X_r)^2}P'_\varphi(Y(r), Z(r))\right\}d\varphi \quad (3.2\text{-}4)$$

역투사 과정을 GPU 에서 실행시키는 데에는 두 가지 큰 이점이 있다. 첫째로, 투영 영상의 지정된 좌표에서

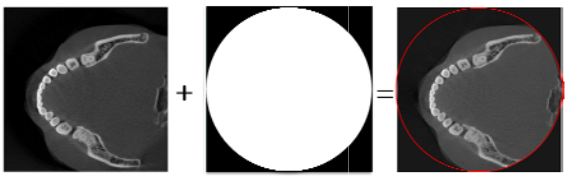
값을 읽어오는 과정(sampling)에서 계산 비용을 절감할 수 있다. 투영 영상은 정수형 좌표를 갖는 이차원 배열로 표현된다. 그러나 수식(3.2-1)으로 계산된 위치는 일반적으로 실수 값을 가지므로 정확한 값을 얻기 위해 이선형 보간(bilinear interpolation) 연산이 필요하다. 쌍일차 보간은 네 번의 메모리 읽기와 세 번의 실수 덧셈, 나눗셈이 필요한 비교적 고가의 연산이다. GPU 에는 그래픽스 작업에서 흔히 사용되는 텍스쳐 매핑(texture mapping)을 고속으로 처리 하기 위해 보간 연산 전용 하드웨어가 구현되어 있다. 따라서 실수 좌표를 사용하여 별도의 계산 비용 없이 보간 된 값을 직접 읽어올 수 있어 재구성 시간을 크게 단축 시킬 수 있다.

두 번째로 GPU 를 이용하면 역투사 공간에서 투영 영상에 대응 되지 않는 불필요한 공간을 쉽게 잘라낼 수 있다. 빔 소스에 의해 투영되는 공간은 삼차원 배열로 표현되는 직육면체가 아니라, 〈그림 3.2-2〉위 아래가 콘(cone)모양인 원기둥이다. 따라서 이 원기둥 바깥 공간에 대해서는 값비싼 역투사 연산을 수행할 필요가 없다.




〈그림 3.2-2〉빔 소스의 궤적[13]

GPU 에서는 래스터라이즈 단계에서 깊이 테스트(depth test)를 이용하여 역투사 공간상의 각 점에 대하여 연산이 필요한지의 여부를 쉽게 판별할 수 있다. 깊이 테스트는 하드웨어를 이용하므로 분기문을 사용하는 프로그램보다 더 효율적으로 불필요한 계산을 제거할 수 있다. GPU 에서의 깊이 테스트는 깊이 버퍼(depth buffer)를 참조하여 진행된다. 〈그림 3.2-3〉의 가운데 그림과 같이 유효한 공간과 잘라낼 공간을 구분하는 이진 영상을 만들고 이를 깊이 버퍼에 저장한다. 그 다음 역투사 연산 단계에서 이 버퍼를 참조하여 가운데 그림에서 검은색 영역에 해당하는 영역에 대한 역투사 연산을 제거함으로 전체 연산량을 20%가량 절감 할 수 있다.



〈그림 3.2-3〉 깊이 테스트를 이용한 단축된 역투사

## 4. 실험 결과 및 토의

본 논문에서 제안한 방법의 효과를 알아보기 위해 다음과 같은 조건에서 실험을 진행하였다.

| | |
|---|---|
| CPU | Intel core2 Quad Q6600@2.40 |
| RAM | DDR2 4GB × 2 |
| GPU | Nvidia GTX 280 (1GB) |
| OS | Window Vista 64bit |
| IDE | Visual Studio. Net 2005(C++) |

〈표 4.1〉 실험 환경

본 논문은 GPU 를 사용하여 대용량 데이터의 빠른 재구성을 목표로 한다. 실험에서 사용한 대용량의 투영 영상을 한번에 읽어 들이기 위해 1GB 의 메모리를 가진 GPU 가 필요하지만 이를 지원하는 모델이 거의 없기 때문에 다양한 모델에 대한 실험은 생략하였다.

| | |
|---|---|
| 투영 영상 크기 | 1000 × 1000 × 400 |
| 픽셀 당 바이트 | 2 byte |
| 재구성 영상 크기 | 512 × 512 × 512 |
| 픽셀 당 바이트 | 2 byte |

〈표 4.2〉 실험 데이터 명세

실험은 1)CPU 를 OpenMP 를 사용하여 최적화 시킨 경우와 2)GPU 만을 사용한 경우, 3)마지막으로 GPU 에 제안된 최적화 방법을 적용한 경우, 각각의 결과를 비교하는 방법으로 진행하였다. 〈표 4.2〉의 데이터에 대해 각각 10 번씩 수행한 결과의 평균을 기록하였다.

* 단위는 초(sec)

| 구분 | CPU | GPU | Opt. GPU |
|---|---|---|---|
| 여과 처리 | 75.107 | 2.507 | 1.097 |
| 역투사 | 350.223 | 3.722 | 3.028 |
| **총 계** | 449.517 | 6.864 | 5.013 |

〈표 4.3〉 비교 실험 결과

재구성 과정에는 여과 처리 과정과 역투사 외에도 여러 가지 과정이 필요하므로 실험 결과에서 여과 처리와 역투사에 소요된 시간의 합이 총계와 다르다. 〈표 4.3〉의 첫 번째 실험과 두 번째 실험을 비교해 보면 최적화 없이도 GPU 사용하여 나타난 월등한 성능 향상을 볼 수 있다. 두 번째와 세 번째 실험의 여과 처리 결과를 비교해

보면, 네 개의 채널을 이용한 압축 효과가 4 배의 속도 향상으로 이어지지 않았다. 그 이유는 GPU 에서 여과 처리를 진행하기 위해 CPU 메모리 상의 데이터를 GPU 로 옮기는 작업이 필요하기 때문이다. 또 고속 푸리에 변환의 특성상 매 단계 마다 GPU 와 CPU 간의 통신이 필요하므로 계산량의 변화가 소요시간의 변화와 정확히 비례하지 않는다. 하지만, 실험에서 메모리 복사에 드는 시간을 제외하고 측정하면 최적화 시킨 경우는 0.03 초, 최적화 시키지 않은 경우는 0.918 초로 30 배의 속도 차이가 발생한다. 이는 여과 처리 과정에서 CPU 와 GPU 간의 통신이 차지하는 시간 비용이 크고, 이 비용이 *1/4* 로 줄어 들었기 때문에 나타난 결과이다. 따라서 제안된 4 채널 압축 방법은 기존의 여과 처리 방법에 비해 월등한 성능 향상 효과를 가져온다.

역투사 과정에서는 최적화 시킨 방법이 예측한 바와 같이 20%정도의 성능 향상을 보였다. 이는 여과 과정과는 달리 메모리 복사 등의 비용보다는 역투사 연산 자체가 차지하는 비율이 지배적이기 때문이다.

## 5. 결론

본 논문에서는 기존의 삼차원 재구성 알고리즘을 GPU 상에서 구현하고 적용 가능한 가속 방법들을 제시하였다. 첫째로 일차원 고속 푸리에 변환을 사용한 여과 과정에서 네 장의 영상을 두 개의 복소수로 압축하고, 이를 GPU 의 RGBA 네 개의 채널을 이용하여 동시에 처리하는 방법이다. 두 번째로는 복잡도가 높아 가장 시간이 많이 걸리는 역투사 과정에서 깊이 테스트를 이용하여 불필요한 공간을 잘라내는 방법이다. 이러한 방법을 적용하여 기존의 FDK 알고리즘이 GPU 상에서 최적으로 동작할 수 있도록 구현 방법을 제시하고 비교 분석하였다. 그 결과 CPU 기반 프로그램 대비 약 50 배, 기존 GPU 프로그램 대비 50% 정도의 속도향상이 있었다.

## 참고문헌


[1] Drebin, R.A., Carpenter, L., Hanrahan, P., “Volume Rendering”, Computer Graphics,

[2] C. Axelsson, P. Danielsson, Three-dimensional reconstruction from cone-beam data in $O(N^3 logN)$ time, *Physics in Medicine and. Biology*, 39, 477-491, 1994.

[3] S. Basu and Y. Bresler, $O(N^3 logN)$ backprojection algorithm for the 3D Radon Transform, *IEEE Trans. on Med. Imaging*, 21(2), 76-88, 2002

[4] S. Butler and M. I. Miller, "Maximum a Posteriori estimation for SPECT using regularization techniques on massively parallel computers," *IEEE Trans. Med. Imag.*, vol. 12, no. 1, pp. 84-89, 1993.

[5] 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.

[6] 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.

[7] K. Chidlow and T. Moller, "Rapid emission volume reconstruction", Volume Graphics Workshop, pp. 15-26, 2003.

[8] 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.

[9] Feldkamp, L. A., L. Davis, and J. Kress (1984). Practical Cone-beam Algorithm.

[10] J. Radon “Uber die Bestimmung von Funktionen durch ihre Integralwerte langs gewisser Mannigfaltigkeiten” (1917) by J Radon”

[11] Cooley, James W., and John W. Tukey, 1965, "An algorithm for the machine calculation of complex Fourier series," *Math. Comput.* 19: 297 301.

[12] Kenneth Moreland , Edward Angel, The FFT on a GPU, Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware, July 26-27, 2003, San Diego, California

[13]Henrik Turbell “Cone-beam Reconstruction Using Filtered Backprojection” Linkoping Studies in Science and Technology Dissertation No. 67