석사학위논문

고차 보간을 이용한 고화질 볼륨 가시화를 위한 GPU 기반의 가속화 기법

2018년

한 성 대 학 교 대 학 원

정 보 시 스 템 공 학 과

정보시스템공학전공

신 용 하

석사학위논문
지도교수 계희원

고차 보간을 이용한 고화질 볼륨 가시화를 위한 GPU 기반의 가속화 기법

GPU-based Acceleration Technique for
High-Resolution Volume Visualization Using
High-Order Interpolation

2017년 12월 일

한성대학교 일반대학원

정보시스템공학과

정보시스템공학전공

신 용 하

석사학위논문
지도교수 계희원

고차 보간을 이용한 고화질 볼륨 가시화를 위한 GPU 기반의 가속화 기법

GPU-based Acceleration Technique for
High-Resolution Volume Visualization Using
High-Order Interpolation

위 논문을 공학 석사학위 논문으로
제출함

2017년 12월 일

한성대학교 일반대학원

정보시스템공학과

정보시스템공학전공

신 용 하

신용하의 공학 석사학위 논문을 인준함

2017년 12월 일

심 사 위 원 _____(인)

심 사 위 원 _____(인)

심 사 위 원 _____(인)

국 문 초 록

고차 보간을 이용한 고화질 볼륨 가시화를 위한 GPU 기반의 가속화 기법

한 성 대 학 교 일 반 대 학 원
정 보 시 스 템 공 학 과
정 보 시 스 템 공 학 전 공
신 용 하

볼륨 가시화의 샘플링 단계에서는 일반적으로 선형 보간이 사용된다. 보통 선형 보간은 좋은 화질의 영상을 보여주지만 혈관, 신경관 같은 정밀 가시화의 경우 기대에 미치지 못하는 영상이 출력된다. 따라서 의료영상 소프트웨어는 높은 화질의 영상이 필요하다. 본 연구는 볼륨 가시화 샘플링 단계에서 B 스플라인 기반의 삼차 보간을 수행한다. 기존의 B 스플라인은 근사 함수의 특징을 가지고 있어 제어점을 지나지 않는다. 따라서 제어점을 이동하여 재생성 된 곡선이 기존의 제어점을 지나는 B 스플라인 보간법을 사용하였다. 이러한 삼차 보간은 선형 보간에 비해 매우 높은 연산량을 요구하기 때문에 가속화 기법을 필요로 하게 된다. 일반적으로 가시화 단계에서 불필요한 연산을 줄여 속도를 향상하기 위해 빈공간 도약 가속화 기법을 적용한다. 빈 공간을 판단하기 위해서는 각 블록에 대한 최대값을 계산해야 하지만 선형 보간과 달리 삼차 보간은 곡선이기 때문에 정확한 최대값을 구하기 어렵다. 다만 B 스플라인은 볼록포 성질이 있어 제어점의 값을 최대값으로 사용할 수 있다.

하지만 B 스플라인 제어점을 이용하여 구한 최대값은 보수적인 범위로 계산돼 빈공간 도약 효율이 낮아 렌더링 속도가 떨어지게 된다. 본 연구에서는 B 스플라인 곡선을 조각적 베지어 곡선으로 분해하여 타이트한 최대값을 얻어 렌더링 속도를 개선한다. 또한 결합전송을 위해 GPU 병렬처리에서 메모리 참조를 효율적으로 수행하는 블록, 스레드 구조를 제안하였다. 그 결과로 삼차 보간 기반의 고화질 볼륨 가시화가 기존의 B 스플라인 기반 빈공간 도약에 비해 2배 이상의 성능이 향상된다.

【주요어】 의료영상 가시화, 볼륨 가시화, 실시간 렌더링, B spline 보간, GPU 렌더링.

목 차

표 목 차

그 림 목 차

제 1 장 서 론

제 1 절 볼륨 가시화

볼륨 가시화는 삼차원의 볼륨 데이터를 이용하여 이차원의 영상을 얻는 렌더링 기법이다(Levoy, M., 1988; Engel, K., 2006). 환자가 CT 또는 MRI 등을 촬영하면, 수백 장의 인체 단면 영상이 복셀(voxel)이라고 불리는 스칼라(scalar)값의 3차원 배열 형태로 생성되며 이를 하나의 볼륨 데이터라고 한다. 배열의 크기가 수백 MB에서 수 GB에 이를 정도로 매우 많은 메모리를 차지하게 된다. 이토록 많은 단면 영상들을 환자 및 임상의가 하나씩 관찰하여 진단하기 어렵기 때문에, 의료영상 시스템에서는 볼륨 가시화를 이용하여 다양한 효과를 가진 영상을 생성한다.

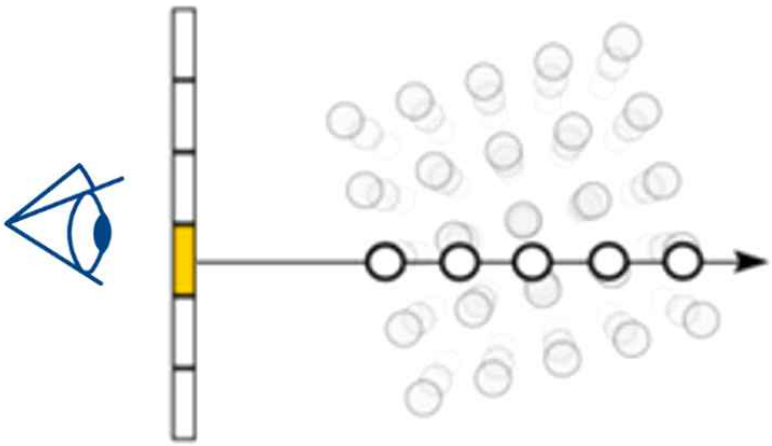
볼륨 가시화의 각 파이프라인 단계인 분류, 조명, 샘플링 등에서 영상의 화질 향상을 얻을 수 있으며, 본 논문은 각 파이프라인 단계 중 샘플링 과정에서의 화질 향상에 주목한다. 볼륨 데이터는 정수 좌표에 데이터가 존재하는 배열 형태이다. 회전이나 이동과 같은 변환 시 정수 좌표 사이 위치에서 값을 획득하는 샘플링 과정이 필수적이다. 이는 일반적인 영상처리와 과정이 같다. 보통 선형 보간은 하드웨어의 가속을 지원받을 수 있어 많이 사용되나, 최근 하드웨어의 발달로 더 좋은 화질을 생성하는 고차 보간의 사용이 증가하고 있다(계희원, 2011; Sigg, C. and Hadwiger, M., 2005).

고차 보간은 잘 알려진 그래픽스의 곡선을 사용한다. 볼륨 데이터에 저장된 복셀의 밀도 값을 제어점으로 파악하고, 중간 위치의 밀도 값은 곡선을 이용한 보간을 계산하여 추정하는 방법이다. 고차 보간은 연산량이 높기 때문에 볼륨 가시화에 사용하기 위해서는 필수적으로 가속화 알고리즘을 적용해야 한다.

볼륨 가시화 방법 중 VR은 MRI, CT와 같은 전산화 단층 촬영 장치로 인체 내부에 대한 단면 영상을 연속적으로 얻어낸다. 이를 인체의 3차원적 구조로 재구성함으로써, 내시경 카메라로 보는 것과 같은 가상의 입체 영상을

만들어낼 수 있다. 또 다른 방법인 MIP는 영상을 구성하는 각 픽셀에 대해 관찰 방향으로 가상적인 광선을 발사하여, 광선에서 일정 간격마다 볼륨 데이터 내부의 밝기값을 얻고 이 중 최대값을 보여주는 기법이다.

볼륨 가시화의 대표적인 방법은 광선 투사법(ray casting)이다(Levoy, M., 1988). 광선 투사법은 영상을 구성하는 각 화소에서 광선이 출발하여 육면체 형태의 볼륨 데이터를 관통하며 얻은 밀도값들을 사용자가 정의한 전이함수를 통해 광학 성분으로 변환 후, 이를 적분하여 화소의 색상을 결정한다. 이때 적분을 계산하기 위해 근사적으로 수치해법을 통해 광선의 일정 거리마다 이산적으로 밀도값을 구하는 방법을 사용한다.

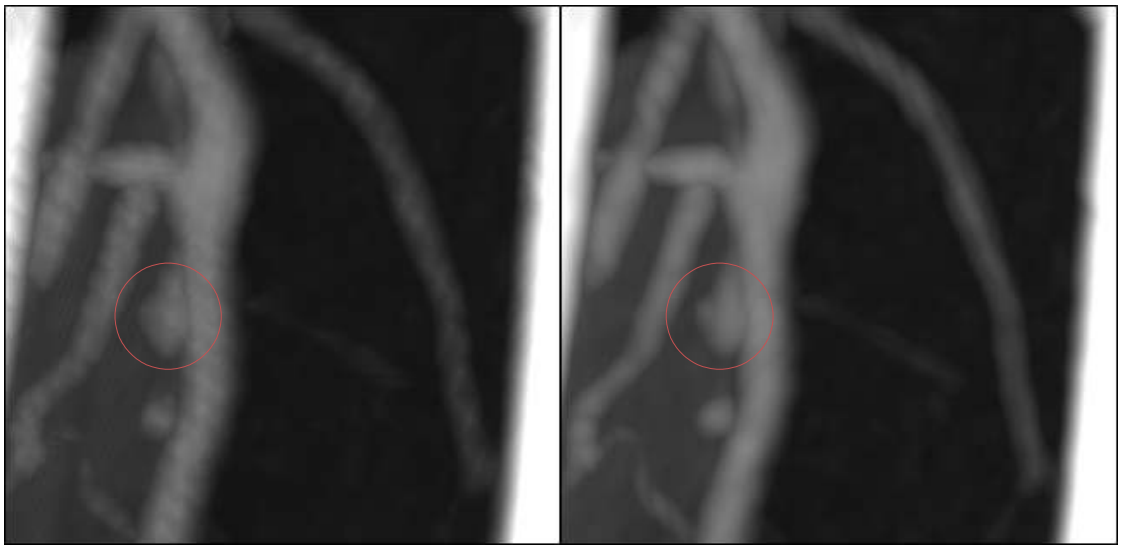


[그림 1-1] 광선 투사법.

제 2 절 샘플링 보간법

의료 데이터의 경우 정밀한 렌더링 영상이 필요하며, 샘플링 방법이 영상의 화질을 결정한다. 샘플링 간격을 촘촘하게 하는 슈퍼 샘플링을 사용하면 더 좋은 화질의 영상을 얻게 된다. 슈퍼 샘플링은 영상의 해상도를 증가시키는 영상영역 슈퍼 샘플링이 주로 사용되며, 샘플링 비율이 커지게 되면 일반적으로 사용하는 선형 보간 샘플링으로는 화질 향상이 어렵다. 따라서 이런 경우 샘플링 커널 함수를 선형이 아닌 고차 함수를 사용하여 더욱 좋은 화질을 얻

을 수 있다(EMeijering, E. H., et al, 2001; Hadwiger, M., et al, 2003; Unser, M., 2002). [그림 1-2]와 같이 조영된 혈관의 구조를 관찰하는 경우, 확대가 많이 되기 때문에 선형 보간에 비해 고차 보간의 영상이 매우 우수하다.



[그림 1-2] 선형 보간과 고차 보간 영상 비교.

제 3 절 연구 내용

앞 절에서 설명한 고화질의 영상을 생성하려면 연산량과 메모리 참조가 많아지기 때문에 오랜 시간이 걸리게 된다. GPU를 사용하더라도 대화적 처리가 불가능하다. 일반적으로 사용하는 선형 보간은 주어진 두 값을 직선으로 연결하고, 두 값의 가중 합으로 중간값을 얻는 방법이다. 삼차원 공간에서 선형 보간은 2^3=8개의 데이터에 가중치를 부여한 합으로 계산할 수 있으며, 고차 보간은 4^3=64개의 데이터에 가중치를 부여한 합으로 계산되어 연산량과 메모리 전송량이 대폭 증가하게 된다.

본 논문은 GPU기반으로 고차 보간의 고화질 영상을 생성하는 볼륨 렌더링을 수행한다. 그 과정에서 속도를 대폭 향상하는 두 가지 새로운 방법을 제안한다. 우선 대표적인 가속화 기법인 빈공간 도약을 고차 보간을 이용한 렌

더링에 적용하고, 이를 개선하여 효율적인 도약 비율을 보이고자 한다. 또한 GPU 기반의 렌더링 시 관찰 방향이 변경될 때, 병렬 처리 묶음의 최적 구조를 동적으로 계산하여 효율적으로 GPU 메모리 접근하는 방법은 제안한다.

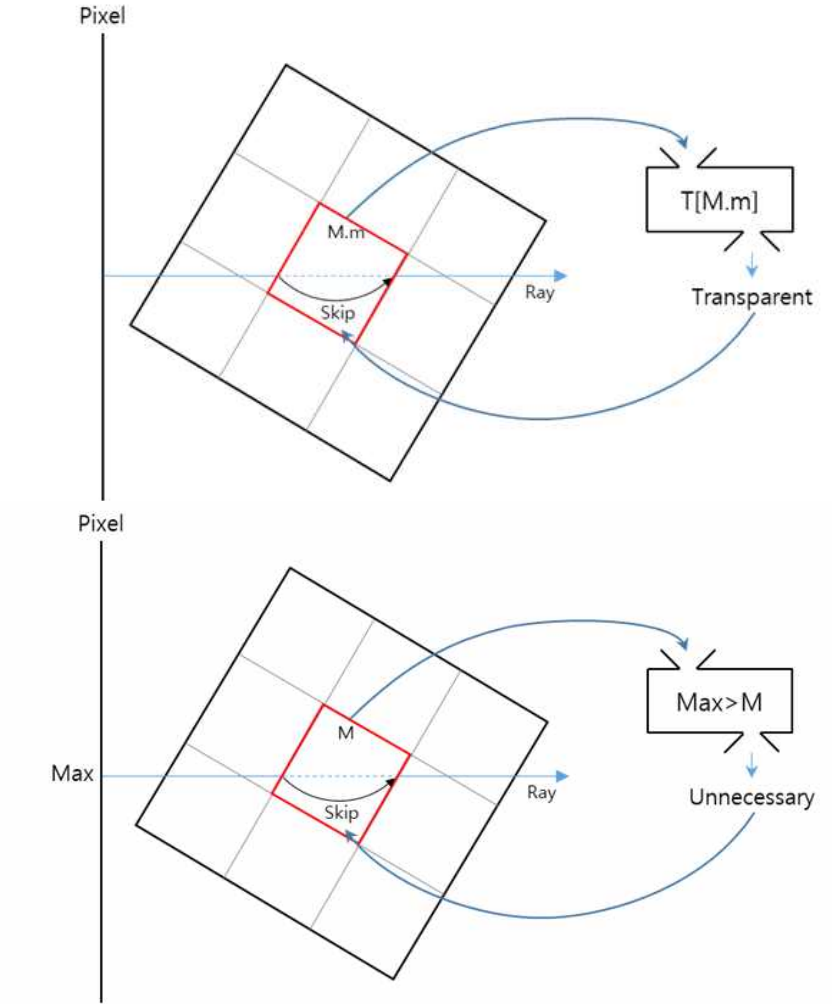
제 4 절 논문의 구성

본 논문은 다음과 같이 구성되어 있다. 먼저 2장에서는 빈공간 도약, B 스플라인 보간 등에 대해 본 연구와 관련된 기존 연구를 설명하고 3장에서 렌더링 수행 시 불필요한 영역을 효과적으로 판단하는 방법을 설명하고 4장에서 GPU의 메모리 참조 효율을 높이는 방법을 설명한다. 5장에서 실험 결과를 보이고 마지막으로 6장에서 결론을 맺는다.

제 2 장 관련 연구

제 1 절 빈공간 도약

볼륨 데이터에서 출력 영상에 반영되지 않는 불필요한 부분을 제거할 수 있으면 가시화 속도가 향상된다. 대표적인 가속 알고리즘인 빈공간 도약은 투명한 영역의 볼륨 데이터를 미리 찾아내서 저장해두고, 렌더링 시 투명한 부분의 연산을 생략하는 방법이다(Li, W., et al, 2003). VR의 경우에는 데이터의 투명도 판별은, 사용자가 입력한 전이함수를 통해 결정한다. 샘플링을 수행한 위치 x에 대한 볼륨 데이터의 밝기값 $d = V(x)$를 구하고, 이를 투명도로 변환하는 $a_d = T(d)$를 계산하면 해당 위치에서의 투명도, 즉 알파값를 계산할 수 있다. 볼륨 데이터의 어느 영역에서 존재하고 있는 밝기값의 최소값 m과 최대값 M을 안다고 가정하자. 만약 조건 $m < d' \leq M$를 만족하는 모든 d'에 대해서 $a'_d = T(d') = 0$이라면, 그 영역은 모두 투명한 것으로 파악되어 가속화가 가능하다. 또한 누적 합계 표를 $S_d = \Sigma a_k$로 정의해 놓으면 $S_{m-1} = S_m$의 여부로 해당 영역의 모든 점의 투명함 여부를 구할 수 있다. 그러므로 전처리 과정에서 볼륨의 모든 영역별로 최소값과 최대값을 미리 구하여 저장해 두면 빈공간 도약을 수행할 수 있다.



[그림 2-1] 빈공간 도약 방법.

본 연구에서 사용한 렌더링 기법인 MIP의 경우 투명한 영역이 존재하지 않는다. 따라서 MIP 렌더링 과정에서 실시간으로 누적된 광선의 최대값과 블록의 최대값을 비교하는 연구가 발달하였다. 광선의 진행 중에 현재까지 누적된 최대값이 현재 처리해야 할 블록의 최대값보다 크다면, 해당 블록을 생략하여도 최대값은 변화가 없고 영상에도 손실이 없다[그림 2-1].

[표 1-1] 도약 코드

if $max_{block} < max_{accumulation}$ then skip block else process block

제 2 절 B 스플라인 근사

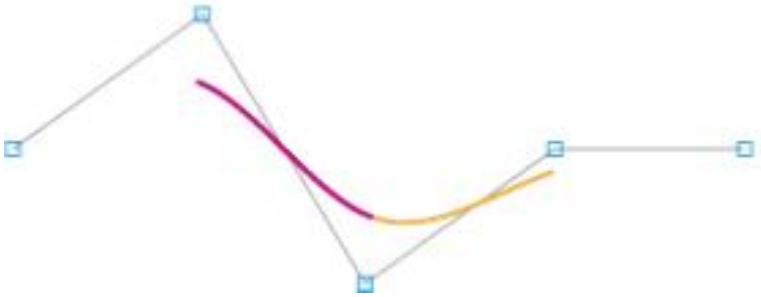
제어점을 이용하여 곡선을 생성하는 여러 방법은 매우 잘 알려져 있고, 그 중 삼차 B 스플라인의 식은 다음과 같다(Farin, G., 2014).

$$S_i(t) = [t^3\ t^2\ t\ 1]\frac{1}{6}\begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix}\begin{bmatrix} P_{i-1} \\ P_i \\ P_{i+1} \\ P_{i+2} \end{bmatrix} \qquad (1)$$

매개변수 t에 변함에 따라 4개 제어점의 가중치의 합으로 표현된다. 즉, 매개변수 t에 따라 점들의 가중치가 변하면서 곡선을 그리게 된다. 본 연구에서는 제어점 4개를 입력으로 받는 3차 B 스플라인에 대해 다룬다. 예를 들어, S_i는 P_{i-1}, P_i, P_{i+1}, P_{i+2}의 점을 입력으로 하고, S_{i+1}은 P_i, P_{i+1}, P_{i+2}, P_{i+3}을 입력으로 받는다. 각각의 S_i, S_{i+1}은 조각적으로 부드럽게 연결되는 특징이 있다. B 스플라인은 모든 $S_i(0) = (1P_{i-1} + 4P_i + 1P_{i+1})/6 \neq P_i$이므로 제어점의 1 : 4 : 1 : 0로 가중합으로 표현되어, 주어진 제어점을 지나지 않는 특징을 가지고 있다.

이를 볼륨 가시화에 적용하면, 사용자가 정의한 불투명도 전이함수의 역할로 인해 볼륨 데이터에는 분명히 존재하는 값이 관찰되지 않는 상황이 발생한다. 예를 들어 [그림 2-2]에서 가장 높은 점의 밝기가 10인 경우를 가정하자. 사용자가 $T(d) = 1$(단, $d \geq 10$), $T(d) = 0$(단, $d < 10$)으로 정의한 후, 렌더링을 수행하면, 실제로 샘플링된 결과(곡선의 높이)는 모두 10보다 작으므로 모든 점이 투명하게 판단되어 최종 영상에 반영되지 않는다. 이러한 문제로 B 스플라인은 오버블러(overblur)한 특성으로 비판이 된다. 실제 측정값이 존재하는 지점에서 B 스플라인으로 얻은 값이 실제 측정값과 다르기 때문이다. 즉, 볼륨 데이터의 격자점 (i,j,k)좌표에서 B 스플라인 근사를 수행하면 $Bspline(i,j,k) \neq vol[i][j][k]$이 된다. 때문에 의료 영상의 경우 B 스플라인 근사법은 매우 조심스럽게 적용해야 한다. 기존의 연구는 이러한 문제를 가진 B 스플라인 근사를 그대로 사용하는 경우가 존

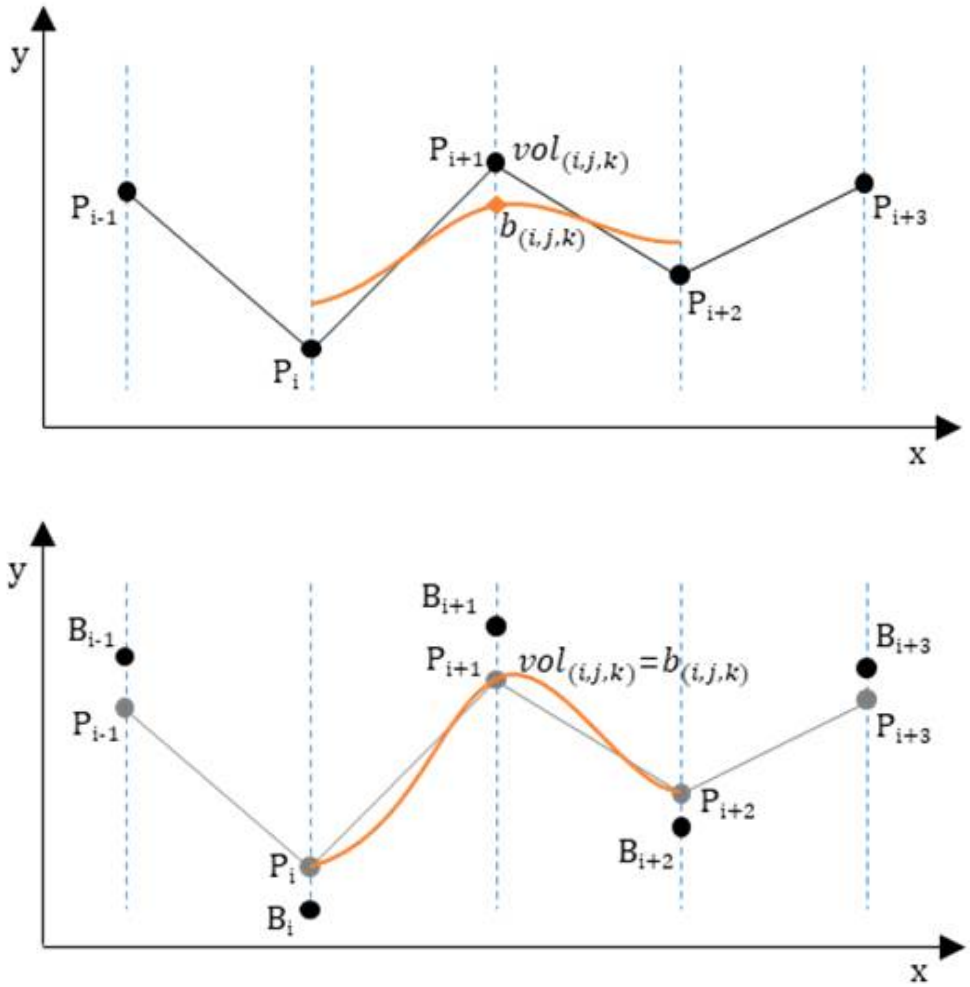
재하는데, 관련 연구의 경우, 주어진 볼륨 데이터를 B 스플라인 근사를 그대로 이용하여 샘플링을 수행 후, 그 결과값을 카디널 보간으로 재처리한다(Lee, B., et al, 2010). 또한 다른 관련 연구에 따르면 B 스플라인의 특성인 로우패스 필터링이 노이즈를 감소시켜 사용된다고 하며, 후술하는 B 스플라인 보간과 화질을 비교하였다(Mihajlovic, Z., et al, 2004).



[그림 2-2] B 스플라인 근사.

제 3 절 B 스플라인 보간

B 스플라인 보간이란 B 스플라인 근사를 응용하여, B 스플라인을 이용하지만, 제어점을 모두 지나는 보간의 역할을 수행한다. B 스플라인 근사의 문제를 해결하기 위해 제어점을 적절하게 이동시키고, 이 이동한 제어점으로 B 스플라인 곡선을 그렸을 때, 곡선이 이동 전의 제어점을 지나게 하려고 한다. 이 방법을 B 스플라인 보간이라고 한다. [그림 2-3]에서 보듯이, 제어점 P_i를 이동하여 제어점 B_i를 생성하고, B_i를 이용해 B 스플라인 곡선을 그리게 되면 원래 주어진 제어점 P_i를 모두 지나게 되는 원리이다.



[그림 2-3] B 스플라인 보간.

그렇다면, 주어진 제어점을 이용해 새로운 제어점의 위치를 구하는 방법이 필요하다. 주어진 점 P_i를 이용하여 B_i를 구하려면, 각 i에서 방정식을 만족하는 B_i를 생성하면 된다.

The values for c[k] should be calculated such that the equation is fulfilled for all integers k while β is B-spline basis of degree 3:

$$f[k]=\Sigma_{(k'\in Z)}c[k']\beta(k-k'),\quad \beta(x)=\begin{cases}0, & (2\le|x|)\\ \frac{1}{6}(2-|x|)^3, & (1\le|x|<2)\\ \frac{2}{3}-\frac{1}{2}|x|(2-|x|), & (|x|<1)\end{cases} \tag{2}$$

이를 식으로 표현하면 식 (2)와 같으며, 관련 문헌과 같이 방정식을 풀 수 있다(Ruijters D. and Thévenaz P., 2010).

제 4 절 볼륨 데이터의 B 스플라인 보간

앞선 3 절에서는 데이터를 일차원으로 가정하였으나 볼륨 데이터에 적용하려면 이를 삼차원으로 확장해야 한다. 즉, 삼차원 제어점 배열 B_i를 생성하고 이를 B 스플라인 기반의 삼차원 근사를 수행하면, 해당 곡선은 기존의 볼륨 데이터값을 지나는 보간의 역할을 수행해야 한다. 먼저 설명을 간단히 하기 위해 이차원 영상에 대해 예를 들었다. 주어진 $width \times height$ 크기의 영상에 대해, $(width+2)\times(height+2)$크기의 메모리 배열을 할당한다. 여기에 축 x방향으로 제어점을 생성하는 작업을 $height$회 반복하여, $(width+2)\times height$크기의 제어점을 생성한다. 이후, 이를 다시 입력으로 하여 y축 방향으로 제어점을 생성하는 작업을 $width+2$회 반복하면 이차원에 대한 제어점 구성이 완료된다. 삼차원 볼륨 데이터는 이를 z축에 대해 추가로 수행하면 된다. 원본 볼륨 데이터의 크기를 $Vol_x \times Vol_y \times Vol_z$라고 한다면 최종적으로는 각 축의 길이가 모두 2씩 늘어난 $(Vol_x+2)\times(Vol_y+2)\times(Vol_z+2)$의 B 스플라인 보간 된 볼륨 데이터가 생성된다.

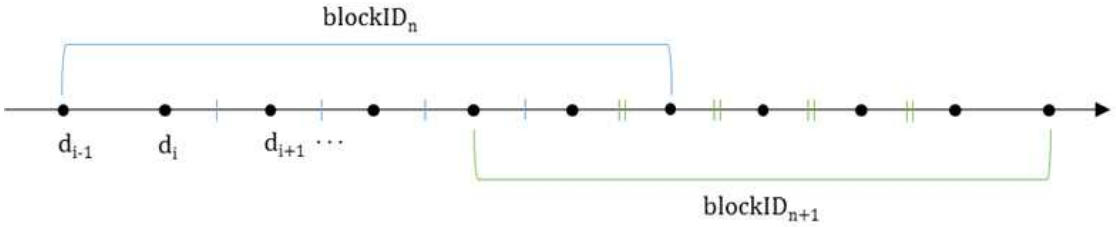
제 5 절 B 스플라인을 이용한 공간 도약

2장 1절에서 서술한 빈공간 도약 알고리즘을 수행하려면 블록 내부의 샘플 가능한 최대값을 계산해야 한다. 선형 보간과는 달리 일반적으로 삼차 곡선 또는 곡면과 입체에서 최대값을 정확하게 계산하는 것은 매우 복잡하여 시간이 오래 걸린다. 이를 간단하게 제어점을 지나는 곡선으로 설명한다. [그림 2-3]에서 각 제어점은 복셀을 의미하며 제어점의 높이(y)값이 복셀의 밝

기값을 의미한다.

제어점의 밝기 보간 된 Red spline에서 높이의 최대값을 계산해야 한다. Red spline은 기존 제어점 P_i를 통과 하지만 P_i의 값으로 곡선의 최대값을 계산할 수 없다. 즉, $\max(Red\,spline) \neq \max(P_{i-1}, P_i, P_{i+1}, P_{i+2})$이다. 정확한 곡선의 최대값을 계산하려면 방정식을 풀어 계산해야 한다. B 스플라인뿐만 아니라, 카트뮬-롬(catmull rom), 카디널 스플라인도 마찬가지로, 공간상의 볼륨 데이터서 정확한 최대값을 계산하려면, 64개의 항이 포함된 9차 방정식을 풀어야 하므로 복잡하다.

정확한 최대값을 계산하는, 다시 말해 곡선 및 곡면이 존재하는 영역을 완벽하게 포함하는 경계 상자의 범위를 알고 있는 경우 빈 공간을 파악할 수 있다(계희원, 2011). 하지만 볼륨 가시화에서 공간 도약을 하려면 정확한 최대값이 필요한 것은 아니다. 예를 들어 $max_{approx} \geq$ max(Red spline)인 max_{approx}를 [표 1-1]의 max_{block}으로 사용하면, 도약 비율이 줄어들어 성능은 하락하지만, 화질은 손상되지 않는다. 따라서 max_{approx}>max(Red spline)이며, $|max_{approx} - max(Red\ spline)|$인 max_{approx}를 얻으면 화질손실 없이 가속화를 수행할 수 있다. 관련 연구에서는 B 스플라인의 볼록포(convex hull)특성을 이용하여 max_{approx}:=max(B_{i-1}, B_i, B_{i+1}, B_{i+2}) ≥ max(Red spline)를 이용하였다.



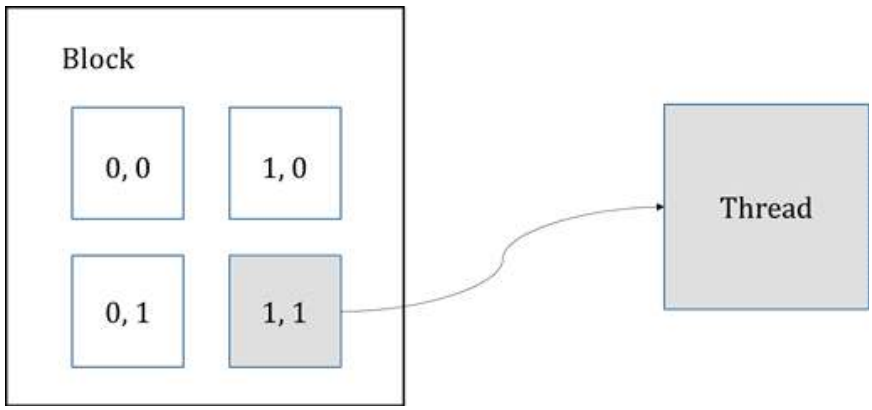
[그림 2-4] 블록의 최대 최소 생성.

이때, 전처리 과정에서 최대값을 계산하는 연산량은 다음과 같이 계산된다. 한 블록의 size가 s로 정해지면 한 블록에 s개의 셀(cell)이 존재하고 s+3개의 제어점이 필요하다. [그림 2-4]는 s=4인 경우이고 인접한 두 블록은 총 3개의 제어점을 공유하게 된다. 한 블록 안에 7개의 제어점으로 4개의 구간

에서 B 스플라인을 생성한다. 3차원 데이터의 경우 각 블록에 대해 $(b+3)^3$개의 복셀에 대해 크기 비교를 수행해야 하며, 전체 전처리 시간은 #of blocks $\times$ $(b+3)^3$회 = O(volume_size / $s^3 \times (s+3)^3$)의 비교가 수행된다.

제 6 절 CUDA 블록, 스레드

CUDA의 구조는 병렬로 2단계인 블록과 스레드(thread)로 되어 있다. 동일한 블록에 속한 스레드는 메모리를 공유할 수 있으며(shared memory), 동일한 분기 조건을 만족하면 성능이 향상되므로(분기 발산) 블록과 스레드를 잘 나누는 것이 중요하다. CUDA의 기본적인 실행단위는 워프(warp)이다. 블록 내의 32개의 스레드를 매핑하여 한 구조로 묶어 동일한 명령을 수행하는 것이 워프이다. 같은 워프에 속한 스레드는 같은 순간에 동일한 소스코드를 실행한다. 따라서 한 블록의 스레드 개수는 32의 배수가 되어야 효율이 높다.



[그림 2-5] CUDA 블록과 스레드.

스레드별 픽셀 간 데이터 교환이 없기 때문에 보통 한 스레드가 픽셀 한 개를 담당한다. 기존에는 단일 광선에 대한 최적화가 우선시 되었다. 그러나 GPU 렌더링 시 각 SM이 32개의 스레드, 즉, 한 워프를 동시에 처리하기 때문에 광선 다발을 모두 고려하는 최적화가 중요하다. 다시 말해 블록이 어떤 모양의 픽셀을 담당하는가가 중요하다. 이에 따라 워프에 따른 메모리 액세스

패턴이 변화되기 때문이다. 기존 연구에서는 볼륨의 x, y, z축 각 회전각을 기준으로 15도 구간별로 블록, 스레드 구조를 동적으로 선택하였다 (Sugimoto, Y. et al, 2014). 이는 일률적으로 같은 크기로 구간을 나누기 때문에 경계부분의 효율이 좋지 않게 된다.

제 3 장 조각적 베지어 공간 도약

제 1 절 베지어 제어점 생성

2장 5절에서 설명한 B 스플라인의 볼록포(convex hull)특성을 이용하여 계산한 경계상자는 [그림 2-2]의 곡선에서 볼 수 있듯이 매우 보수적으로 느슨한 경계값을 생성한다. 그 결과 실제로 투명한 블록들이 불투명할 수 있다고 오인하게 된다. 그로 인해 블록 내부에서 고차 샘플링을 여러 번 계산하여 성능이 하락하게 된다. 만약 영상의 오류가 없도록 내부에 실제로 존재하는 데이터를 완전하게 포함하는 되도록 작은 범위의 최대값을 계산할 수 있으면 빈공간 도약 기법의 효율이 증가하고 성능이 향상될 것이다.

따라서 우리는 B 스플라인 곡선을 조각적 베지어 곡선으로 쪼개어 더욱 타이트한 경계박스를 얻어 빈공간 도약 효율을 상향시키는 방법을 제안한다. 아래는 설명을 간단히 하기 위해 일차원으로 예를 들었다.

$$
\begin{aligned}
B(t) &= [t^3\ t^2\ t\ 1]\frac{1}{6}\begin{bmatrix}-1 & 3 & -3 & 1\\ 3 & -6 & 3 & 0\\ -3 & 0 & 3 & 0\\ 1 & 4 & 1 & 0\end{bmatrix}\begin{bmatrix}B_{i-1}\\ B_i\\ B_{i+1}\\ B_{i+2}\end{bmatrix}\\
&= [t^3\ t^2\ t\ 1]\begin{bmatrix}-1 & 3 & -3 & 1\\ 3 & -6 & 3 & 0\\ -3 & 3 & 0 & 0\\ 1 & 0 & 0 & 0\end{bmatrix}\begin{bmatrix}-1 & 3 & -3 & 1\\ 3 & -6 & 3 & 0\\ -3 & 3 & 0 & 0\\ 1 & 0 & 0 & 0\end{bmatrix}^{-1}\frac{1}{6}\begin{bmatrix}-1 & 3 & -3 & 1\\ 3 & -6 & 3 & 0\\ -3 & 0 & 3 & 0\\ 1 & 4 & 1 & 0\end{bmatrix}\begin{bmatrix}B_{i-1}\\ B_i\\ B_{i+1}\\ B_{i+2}\end{bmatrix}\\
&= [t^3\ t^2\ t\ 1]\begin{bmatrix}-1 & 3 & -3 & 1\\ 3 & -6 & 3 & 0\\ -3 & 3 & 0 & 0\\ 1 & 0 & 0 & 0\end{bmatrix}\frac{1}{6}\begin{bmatrix}1 & 4 & 1 & 0\\ 0 & 4 & 2 & 0\\ 0 & 2 & 4 & 0\\ 0 & 1 & 4 & 1\end{bmatrix}\begin{bmatrix}B_{i-1}\\ B_i\\ B_{i+1}\\ B_{i+2}\end{bmatrix}\\
&= [t^3\ t^2\ t\ 1]\begin{bmatrix}-1 & 3 & -3 & 1\\ 3 & -6 & 3 & 0\\ -3 & 3 & 0 & 0\\ 1 & 0 & 0 & 0\end{bmatrix}\begin{bmatrix}(B_{i-1}+4B_i+B_{i+1})/6\\ (2B_i+B_{i+1})/3\\ (B_i+2B_{i+1})/3\\ (B_i+4B_{i+1}+B_{i+2})/6\end{bmatrix} = Bezier(t)
\end{aligned}
\tag{3}
$$

기존의 B 스플라인 제어점을 이용하여 새로운 베지어 제어점을 생성하게 되면 기존의 B 스플라인으로 생성한 곡선과 베지어로 생선한 곡선은 동일한 곡선임을 알 수 있다. 이때 베지어 제어점으로 생성한 경계박스는 더욱 타이트한 최대값 범위를 가지게 된다. 이렇게 타이트한 범위로 생성된 도약 테이블을 이용해 빈공간 도약을 수행 시 불필요한 샘플링에 대한 도약비율이 높아져 렌더링 성능 향상을 기대할 수 있다.

$$Bezier_i(x,y,z,w) = ((B_{i-1} + 4B_i + B_{i+1})/6, 2(2B_i + B_{i+1})/3, \\ (B_i + 2B_{i+1})/3, (B_i + 4B_{i+1} + B_{i+2})/6)$$
$$Bezier_i(w) = Bezier_{i+1}(x) \quad (4)$$

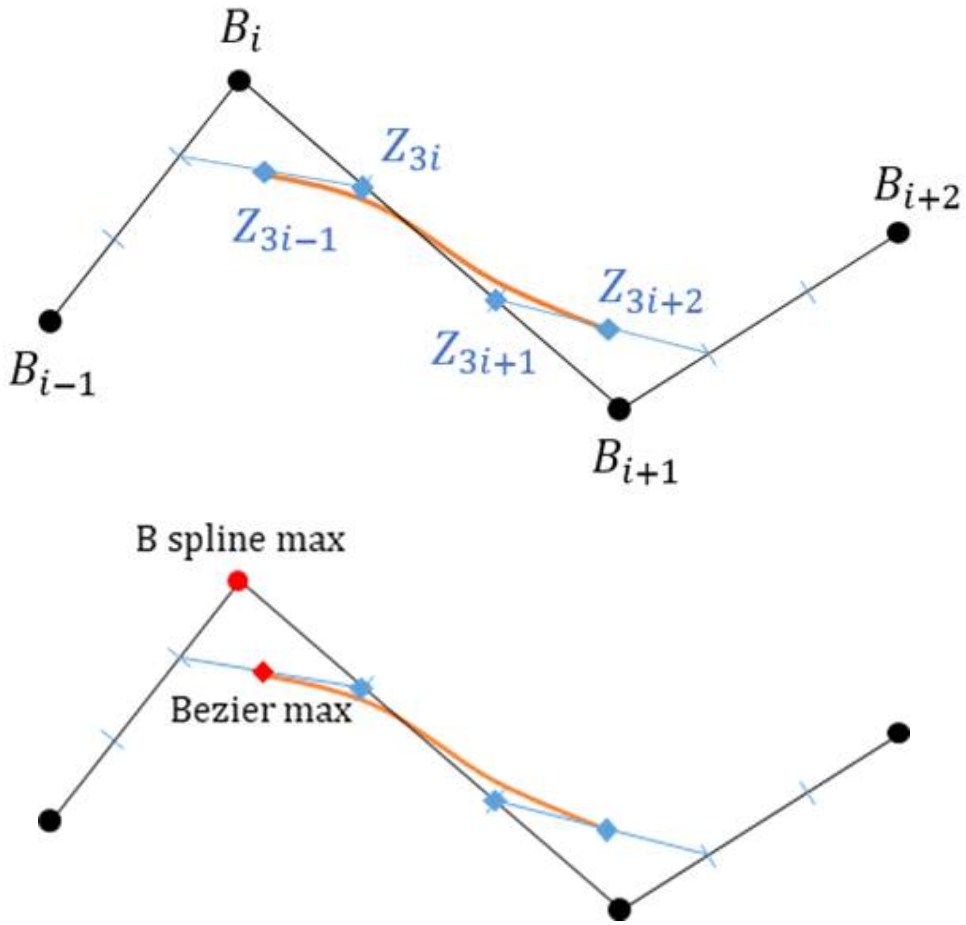
1차원 데이터의 B 스플라인 제어점을 이용해 베지어 제어점을 생성하는 식은 다음과 같다. B 스플라인 제어점 4개, B_{i-1}, B_i, B_{i+1}, B_{i+2}를 이용하여 베지어 제어점 4개, Z_{3i-1}, Z_{3i}, Z_{3i+1}, Z_{3i+2}생성 한다.

B 스플라인의 경우 한 셀이 4개의 제어점을 가지고 인접한 셀끼리 3개의 제어점을 공유하지만 베지어 제어점의 경우 한 셀이 4개의 제어점을 가지는 것은 동일하나 인접한 셀끼리 한 개의 제어점만을 공유한다. $Z_{3i+2} = Z_{3(i+1)-1}$ 이기 때문에 처음 셀의 경우 4개의 제어점을 가지며 인접한 셀이 하나 추가될 때마다 3개의 추가적인 제어점이 필요하게 된다. 그러므로 원래의 셀이 n개라면 B 스플라인은 n+3개의 제어점을 가지지만 조각적 베지어로 분할하게 되면 제어점이 약 3배가량 증가하여 4+3(n−1)개, 즉 총 3n+1개의 제어점으로 증가하게 된다. 삼차원 블록의 경우는 $(3n+1)^3$개의 제어점에 대해 최대값을 계산하면, 그 블록에 대한 처리가 완료된다.

볼륨 데이터 전체에 대해 베지어 제어점을 생성하려면, 기존의 볼륨 데이터에 비해 x, y, z 축으로 각각 3배 가량 제어점이 증가하여 총 3^3=27배의 메모리가 필요하다. 예를 들어 512×512×552 크기의 원본 볼륨 데이터가 있다면, 셀의 개수는 511×511×551개이고, B 스플라인으로 생성된 제어점은 514×514×554개이다. 그리고 변환된 베지어 제어점은 (1+511×3)×(1+511×3)×(1+551×3) = 1534×1534×1654개의 크기가 된다.

원 볼륨 데이터의 크기가 증가하면 모든 베지어 제어점을 저장하는 방법

은 메모리 용량이 부족해질 수 있다. 본 연구는 베지어 제어점의 일부를 생성하고 저장하고 생성된 제어점을 폐기하는 방식을 반복하여 문제를 해결한다. 베지어 제어점은 최대값을 구하기 위한 임시값이며, 제어점 전체가 동시에 필요치 않다. 1534×1534×1654 크기의 메모리를 한 번에 할당하는 대신 한 셀의 크기인 4×4×4 크기의 메모리만 할당하여, 셀을 순차적으로 루프를 돌아 블록 데이터 정보만 저장하여 메모리 부족을 방지한다.

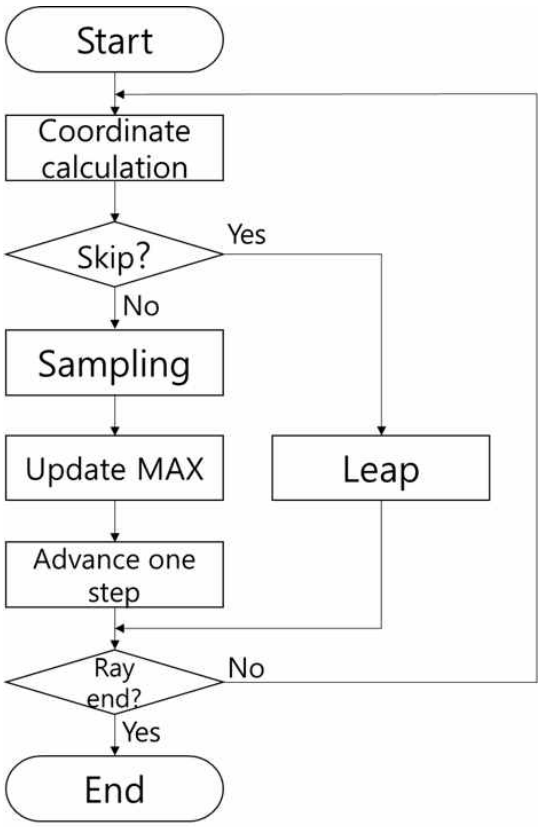


[그림 3-1] 베지어 제어점을 이용한 최대 최소 생성.

[그림 3-1]에서 보듯이 B spline max ≥ Bezier max 이므로, 본 연구에서 제안한 방법을 이용하면 빈공간 도약의 빈도가 증가하여 렌더링 속도가 향상된다.

제 2 절 렌더링 - 공간도약 방법

각 광선별 렌더링 순서는 [그림 3-2]와 같다. 광선이 시작하면 현재 좌표를 계산하고, 도약이 가능한지 확인하여 가능한 경우 그 영역을 건너뛴다. 도약이 불가능한 경우에는 샘플링을 수행하여 누적 MAX를 갱신시킨 후 한 칸을 전진한다. 전진한 광선이 볼륨 데이터의 끝에 도달했다면 한 광선의 처리가 종료된다.



[그림 3-2] B 광선 투사법 흐름도.

앞서 말했듯이 MIP 볼륨 렌더링은 광선에 시작지점에서 방향벡터 쪽으로 레이를 전진시켜 샘플링을 수행하게 된다. 이때 샘플링 해온 새로운 데이터와 기존의 레이를 진행하면서 저장된 최대 밝기값을 비교 연산하여 누적 최대밝기를 추출한다. 따라서 일정 영역의 밝기값의 최대값을 미리 알고 있다면 지금까지 저장된 누적 최대값보다 작을 시 불필요한 연산으로 판단하여 그 영

역의 샘플링을 건너뛰어 렌더링 시간의 효율을 상향 시킬 수 있다.

예를 들어 일정 영역 내의 존재하는 밝기값 D_1, D_2, … , D_n 중 미리 계산하여 최대값인 D_{max}를 알고 있다고 하자. 지금까지 광선을 진행하여 누적해온 최대값인 ray_{max}가 $ray_{max} > D_{max}$일 경우, 영역 내의 모든 데이터에 대해 max연산을 수행해도 누적 ray_{max}의 변화가 없다. 따라서 그 영역에 대해선 스킵을 해도 데이터 손실이 전혀 발생하지 않게 된다. 유효한 데이터가 존재하는 것으로 판단되는 $ray_{max} < D_{max}$일 경우만 샘플링을 수행하고 아닐 경우에는 그 영역을 스킵하여 불필요한 연산을 제거할 수 있다.

제 3 절 최적화

1) 초기값 생성

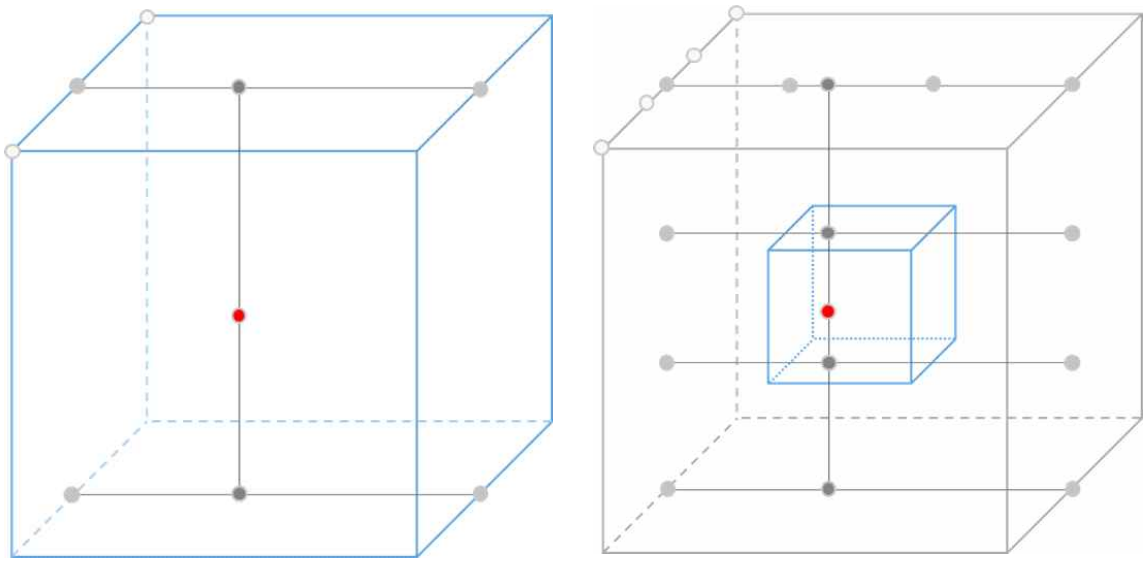
MIP 볼륨 렌더링은 누적 최댓값을 추출하는 작업이다. 때문에 광선 진행 방향으로의 샘플링을 순차적으로 하지 않아도 문제가 되지 않는다.

예를 들어 한 광선에 대한 각 샘플값을 d_0, d_1, d_2, …, d_n이라고 하자. max연산은 결합법칙 max(max(a, b), c) = max(a, max(b, c))과 교환법칙 max(a, b) = max(b, c)이 성립하기 때문에 다음과 같은 계산이 가능하다. Output = max(max(d_0, d_1)..., d_n)) = max(d_0..., max(d_{n-2}, max(d_{n-1}, d_n))) 등으로 순서를 다르게 계산하여도 결과값은 동일하다. 게다가, 샘플링 데이터 중 하나 d_i에 대해서 Output = max(d_i, max(max(d_0, d_1)..., d_n))으로 d_i를 먼저 계산하여도 값이 변하지 않는다. 3장 4절의 광선 도약에 따르면, 큰 값을 먼저 만나게 될 때 효율이 높아지므로, 높은 값으로 추정되는 d_i를 먼저 선택하여 연산순서를 변경하면 성능이 향상된다.

의료영상 데이터의 특성상 볼륨 중심부에 값(밀도)이 큰 뼈나 근육조직이 존재하므로, 이에 착안하여, 광선과 볼륨이 만나는 선분의 중심에서 d_i를 추출하여 먼저 계산하였다. 한편, d_i의 위치를 더 잘 계산하면 효율을 더 높일 수 있으나, d_i를 계산하는 시간과 상충(trade-off)관계이므로, 본 논문은 d_i는 볼륨 중심에서 가져오는 것으로 간단하게 처리하였다.

2) 효율적인 샘플링

삼차 보간의 샘플링은 많은 연산과 메모리 참조가 필요하다. 일차원 상의 B 스플라인 계산은 식 (1)을 전개하여 식 (5)와 같이 쓸 수 있다. 이 식을 3차원 공간에 적용하려면 [그림 3-3]과 같이 x 방향으로 16회, y 방향으로 4회, z 방향으로 1회, 총 21회의 B 스플라인 계산이 요구된다.



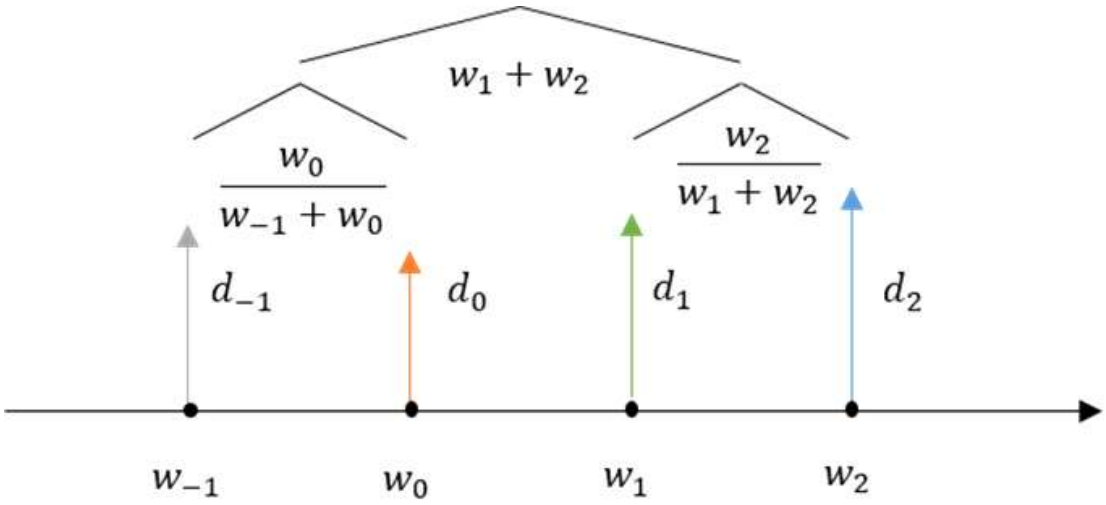
[그림 3-3] 선형 보간과 삼차 보간.

이 식을 각 복셀(제어점)에 대한 가중 합으로 보면, 총 64회의 데이터를 참조하여 계산해야 한다.

$$q_i(t) = \frac{1}{6}(p_i(1-t)^3 + p_{i+1}(3t^3 - 6t^2 + 4) + p_{i+2}(-3t^3 + 3t^2 + 3t + 1) + p_{i+3}t^3) \quad (5)$$

다만, B 스플라인은 제어점에 대한 가중치가 모두 양수이다. 이점에 착안하여 성능을 향상시키는 방법이 제안되었다(C. Sigg and M. Hadwiger, 2005, M. Hadwiger, et al, 2001). 설명의 편의를 위해 일차원 데이터로 예를 들면, 각 제어점의 값 d_{-1}, d_0, d_1, d_2에 대해 가중치 w_{-1}, w_0, w_1, w_2가 계산되어 있다고 가정한다. 이때, GPU의 텍스처 선형보간 엔진이 매우 빠르

다면, [그림 3-4]와 같이 $w_0/(w_{-1}+w_0)$과 $w_2/(w_1+w_2)$의 위치에서 일차원 텍스처 샘플링을 수행하고 이를 이용하여 가중합을 계산한다.



[그림 3-4] 선형 보간을 이용한 B 스플라인 근사.

그 결과 식 (6)의 수식이 유도된다. B 스플라인 근사 방법이 기존의 4번 데이터를 얻어오는 과정이 필요했는데, 이것이 텍스처 연산 2번으로 대체되게 된다. 두 번의 텍스처 연산의 결과에 각각 $(w_{-1}+w_0)$과 (w_1+w_2)의 가중치를 곱한 후 더하면 동일한 결과를 얻을 수 있다. 다만 이 식은, 각 가중치가 양수일 경우에만 성립한다. 따라서 카디널 스플라인에는 적용할 수 없다.

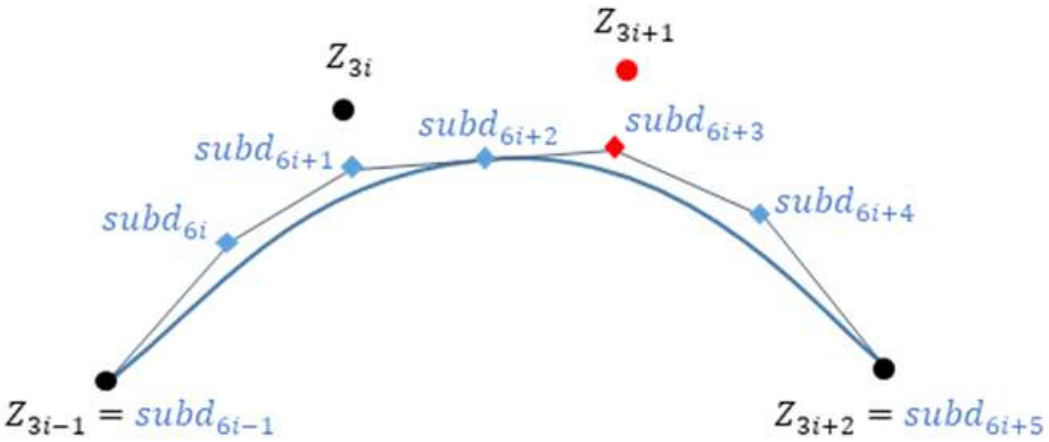
$$D=(w_{-1}+w_0)\left(\frac{w_0}{w_{-1}+w_0}d_{-1}+\frac{w_{-1}}{w_{-1}+w_0}d_0\right)+(w_1+w_2)\left(\frac{w_2}{w_1+w_2}d_1+\frac{w_1}{w_1+w_2}d_2\right) \quad (6)$$

그러나 결국 텍스처 연산도 GPU 내부에 두 번의 데이터 참조이므로 본질적으로 효율이 증가한다고 볼 수 없다. 그렇지만 GPU 하드웨어의 텍스처 보간 설계가 매우 효율적이라면 이 방법은 효과가 있다. 최근 GPU는 기존과 달리 일반 데이터의 메모리 참조에도 L2 캐시 메모리가 사용되고 있다. 따라서 실제 성능의 향상을 직접 확인해 보아야 한다. 이상의 방법을 본 연구에서는 삼차원으로 확장하였다. 총 64번의 볼륨 데이터의 값을 획득 대신 8번의 삼차원 선형 보간을 통한 결과의 가중합으로 연산하게 된다. 이후 나오는 실

험결과에서는 효율적인 샘플링을 모두 적용하여 실험하였다.

3) 서브디비전

앞서 본 연구에선 B 스플라인 곡선을 조각적 베지어 곡선으로 분해하여 타이트한 경계상자를 얻어 빈공간 도약 시 효율을 향상시켰다. 베지어 곡선의 특징은 하나의 베지어 곡선을 두 개 이상의 베지어 곡선으로 분리할 수 있다는 점이다. 반복적으로 분해해 나갈수록 [그림 3-5]와 같이 경계점이 타이트한 범위를 가지게 되어 빈공간 도약의 효율을 향상시킬 수 있다. 예를 들어 아래 [그림 3-5]에서 보는 것과 같이 곡선에서 베지어 제어점의 최대값은 Z_{3i+1}인 것에 비해 서브디비전을 수행한 제어점의 최대값은 $subd_{6i+3}$으로 실제 곡선의 최대값에 더 가까워짐을 볼 수 있다.



[그림 3-5] 베지어 서브디비전.

[그림 3-5]에서는 1차원의 예를 들었고, 4개의 제어점으로 표현되는 하나의 곡선이 인접한 두 개의 곡선으로 분리된다. 베지어 제어점 4개, $Z3_{i-1}$, Z_{3i}, Z_{i+2}, Z_{i+3}을 이용하여 서브디비전 베지어 제어점 7개를 다음과 같이 생성한다.

$$subd_{6i-1} = Z_{3i-1}, \quad subd_{6i} = \frac{Z_{3i-1} + Z_{3i}}{2}, \quad subd_{6i+1} = \frac{Z_{3i-1} + 2Z_{3i} + Z_{3i+1}}{4},$$
$$subd_{6i+2} = \frac{Z_{3i-1} + 3Z_{3i} + 3Z_{3i+1} + Z_{3i+2}}{8}, \quad subd_{6i+3} = \frac{Z_{3i} + 2Z_{3i+1} + Z_{3i+2}}{4},$$
$$subd_{6i+4} = \frac{Z_{3i+1} + Z_{3i+2}}{2}, \quad subd_{6i+5} = Z_{3i+2} \tag{7}$$

하나의 경계점은 공유하므로 2×(4−1)+1=7개의 제어점으로 표현할 수 있다. 만약 이를 다시 한번 분해한다면, 4×(4−1)+1=13개의 제어점으로 표현되는 4개의 곡선 조각으로 분해된다.

본 논문에서는 이를 삼차원으로 확장한다. 한 셀은 4^3개의 제어점으로 표현되며, 이를 분해하여 7^3개의 제어점을 가진 2^3=8개 micro−cell 영역으로 분리하였다. 이를 더 분해하여 13^3개의 제어점으로 분해할 수도 있으나, 메모리 사용량과 연산 시간을 감안하여야 한다.

구현은 CUDA를 이용하여 병렬처리되고 데이터에 대해 단 한번만 계산하면 되므로 시간이 오래 걸리지 않는다. 다만 볼륨 데이터 크기의 7^3배에 해당하는 메모리를 소모하므로 메모리 관리에 주의해야 한다. 이에 따라 본 연구는 일정 영역의 데이터에 처리를 재사용하여, 메모리 소모를 절약하였다.

제 4 장 GPU를 이용한 개발

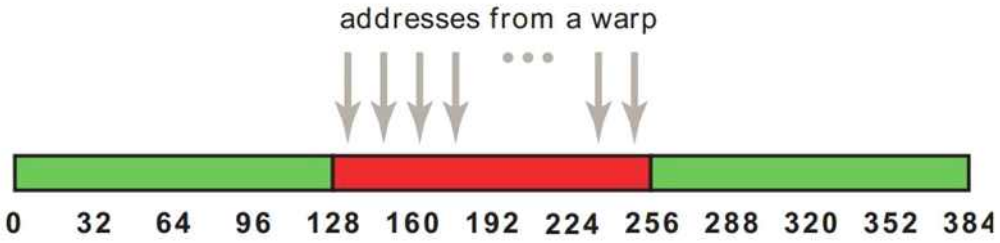
제 1 절 GPU 메모리 액세스 패턴

3차원에서 선형 보간은 8개의 점을 이용하여 7번의 선형 보간을 수행한다. 반면에 삼차 보간은 64개의 점을 이용하여 총 27회 보간을 수행한다. 삼차 보간을 CPU로 처리하면 너무 오랜 시간이 소요되므로, 본 연구는 GPU기반으로 실험을 했다. 최근 볼륨 가시화에 GPU를 사용하는 추세이며, OpenGL 혹은 DirectX의 셰이더 프로그램, OpenCL, CUDA 등의 선택이 있다. 본 연구는 편리한 개발환경, API의 사용 편의성, 문서화의 우수성 등을 고려하여 CUDA를 선택하였다. 병렬화 단계를 설정하기 위해, CUDA의 각 스레드가 볼륨 가시화의 각 광선을 담당한다. 광선 간에는 정보 전달이 없어 독립적인 병렬처리에 유리하다. CUDA는 3D 텍스처를 지원하며 3D 텍스처에 대한 샘플링도 하드웨어적으로 수행된다. 다만 샘플링은 선형 보간과 최근접 화소 보간만을 지원하기 때문에 B 스플라인 기반의 삼차 보간은 별도의 프로그래밍을 필요로 한다.

CUDA 메모리는 기본적으로 CPU보다 빠른 메모리 액세스 속도를 제공하지만 수백 개의 코어에서 동시에 데이터를 읽고 쓰면 데이터 병목현상이 발생하게 된다. 이러한 병목현상을 완화하기 위해 글로벌 메모리 액세스 결합 조건을 충족시켜줘야 한다.

글로벌 메모리 액세스 결합 조건은 아키텍처 버전에 따라 조금씩 변경되었으며 초기 GT200 및 이전의 GPU에서는 조건이 매우 까다로웠다. fermi 이후 많이 완화되어 한 워프 내의 스레드가 요구하는 데이터에 대한 주소가 동일한 크기로 정렬된 128byte 범위 내에 들어가면 글로벌 메모리 액세스 결합 조건을 충족시키게 되어 단일 메모리 트랜잭션만으로도 워프 내 모든 스레드가 요구한 데이터 읽기가 가능하다. 또한 완벽히 128byte 범위 내에 들어가지 못하더라도 스레드 방향으로 데이터가 인접하게 나열되어 있으면 최소한의 메모리 트랜잭션으로 연산이 가능하다. 반면에 최악의 경우인 워프의

모든 스레드가 각각의 물리적으로 떨어진 위치에 접근할 경우 하나의 기억 장치 참조 명령 처리를 위해서 총 32번의 메모리 참조가 필요할 수도 있다. 메모리 발산(memory divergence)을 최소화하고 CUDA 프로그램 성능을 높이기 위해서는 메모리 액세스 패턴을 고려해야 한다.



[그림 4-1] CUDA 메모리 결합전송.

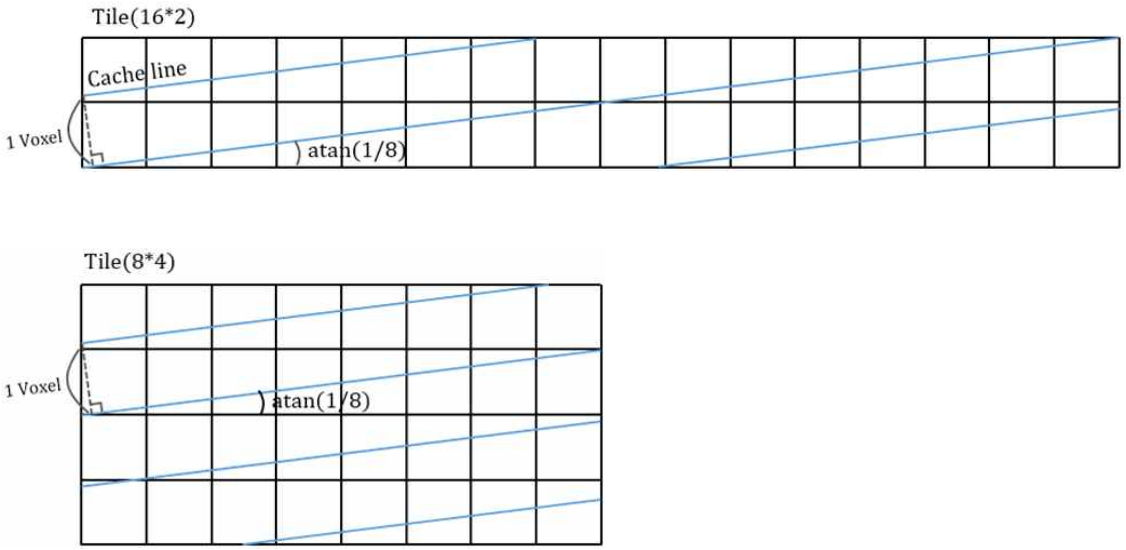
볼륨 렌더링에서 글로벌 메모리 액세스 패턴을 생각해보자. 기본적인 CUDA기반 볼륨 랜더링에서 한 워프 내의 32개의 스레드들은 데이터를 공유하지 않고 각자 독립적으로 명령어를 수행한다. 이러한 스레드들은 명령을 처리하기 위해 볼륨 데이터에 메모리 액세스를 요구하게 된다. 이때 워프 내의 모든 스레드가 볼륨 데이터의 물리적 메모리 순서인 x축 방향 오름차순(cache line)으로 요구하게 되면 단 한 번의 메모리 참조만으로 기억 장치 참조 명령(memory reference instruction)을 처리할 수 있다. 따라서 각 광선별로 스레드가 담당하기 때문에 인접한 레이끼리 같은 캐시 라인에 접근하게끔 스레드, 블록 구조를 잡아 주어야 한다.

제 2 절 GPU 메모리 액세스 최적화

본 연구에서는 영상 픽셀의 가로축인 크로스 벡터 방향과 볼륨 데이터의 물리적으로 메모리 배열순서인 x축의 사잇각을 통해 효율적인 스레드, 블록 구조를 제안한다. 한 워프의 총 32개의 스레드는 1×32, 2×16, 4×8, 8×4, 16×2, 32×1의 비율로 총 8가지 방법으로 선택할 수 있다. 또한 렌더링 시 스레드의 세로/가로의 비율과 크로스 벡터와 볼륨의 x축이 이루는 θ의 tan

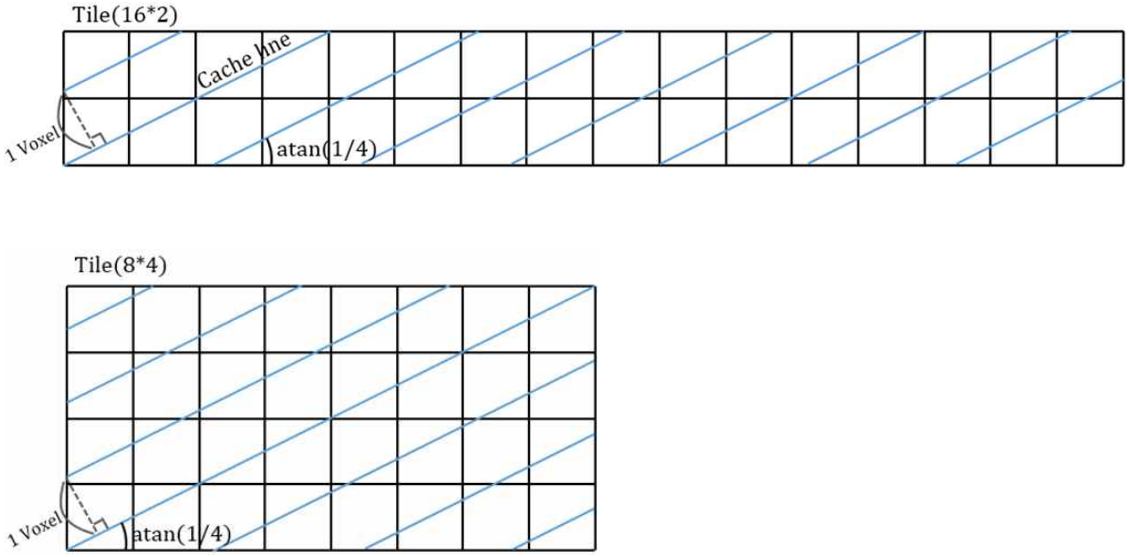
θ는 메모리 액세스 패턴과 밀접한 관련이 있으므로 θ값에 따라 스레드, 블록을 선택하여 렌더링을 수행하게 된다.

하나의 블록이 다루는 픽셀 영역을 가정하여 타일(tile)이라고 부르자. 예를 들어 2×6은 가로 16, 세로 2 크기인 직사각형이다. 우리가 따지는 모든 경우 타일 면적은 32로 동일하다. 볼륨 데이터의 x축에 나란한 선분 방향으로 데이터를 읽으면 효율적이다. 따라서 적은 수의 선분을 이용하여 타일을 칠하면 효율적이다. 그래서 동일 면적의 타일이 있을 때, 가로와 세로의 길이를 구하는 문제가 된다.



[그림 4-2] 블록 타일과 캐시 라인 1.

16×2 타일의 스레드 구조는 종횡비가 8:1이고, 8×4 타일의 스레드 구조의 종횡비는 2:1이다. [그림 4-2]에서 보다시피 픽셀의 크로스 벡터와 볼륨의 x축이 이루는 각도가 atan(1/8)를 이룰 때, 8×4 타일은 4개의 캐시 라인을 포함하는 반면 16×2 타일의 경우는 3개의 캐시 라인을 포함하여 랜더링 시 메모리 참조를 줄일 수 있다.



[그림 4-3] 블록 타일과 캐시 라인 2.

마찬가지로 [그림 4-3]에서는 픽셀의 크로스 벡터와 볼륨의 x축이 이루는 각도가 atan(1/4)를 이룰 때, 8×4 타일이 16×2 타일보다 적은 수의 캐시 라인을 포함하게 되어 8×4 타일을 선택 시 메모리 참조를 줄여 렌더링 속도 향상을 기대할 수 있다. 또한 8×4 타일과 16×2 타일의 선택의 기준인 경계 값은 기하평균을 통해 구하였으며 예시에서는 $\{(1/16)\times(1/2)\}^{0.5}=1/4$이 된다. 모든 타일에 대하여 위와 같은 방법으로 값을 계산해보면 아래표와 같이 나오게 된다.

[표 4-1] tanθ에 따른 타일 선택 ($0\leq\theta\leq\pi/2$)

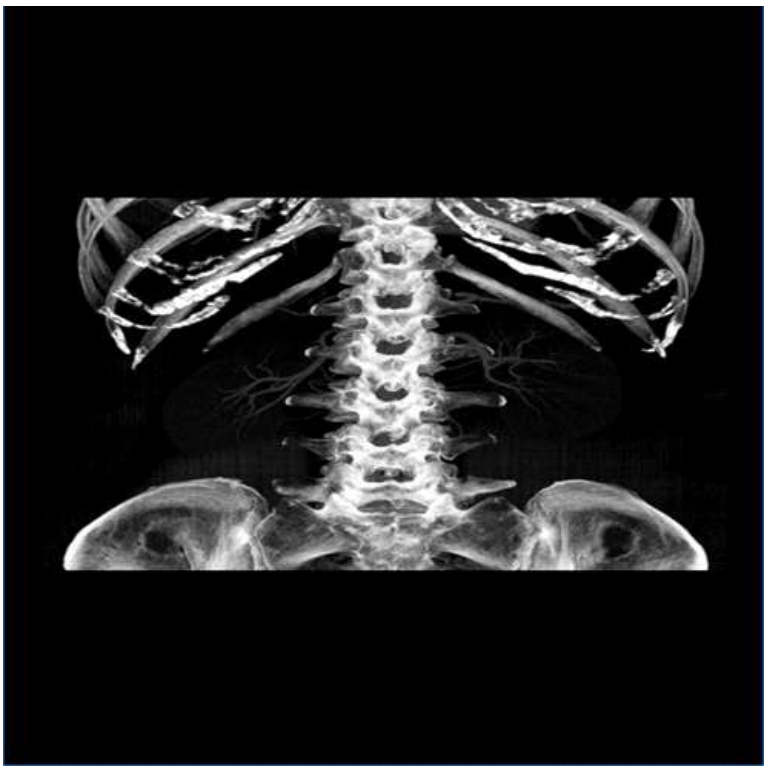
thread/block 구조(tile)	최적의 tanθ값	경계 θ값	선택 조건(degree)
1×32	1/32		θ≤3.58
	1/16	atan(1/16)	
2×16	1/8		θ>3.58 && θ≤14.04
	1/4	atan(1/4)	
4×8	1/2		θ>14.04 && θ≤45.00
	1	atan(1)	

8×4	2		θ＞45.00 && θ≤75.96
	4	atan(4)	
16×2	8		θ＞75.96 && θ≤86.42
	16	atan(16)	
32×1	32		θ＞86.42

제 5 장 실험 결과

제 1 절 결과 영상

본 연구는 window 10 운영체제에서 Microsoft Visual Studio 2010과 Cuda6.5버전을 사용하였다. 하드웨어는 Intel i5 6세대 CPU, 8GB RAM, GeForce GTX 960 GPU이다. 출력 영상의 픽셀크기는 1024×1024이며 무채색(gray scale)으로 출력하였다. 실험은 크게 2가지로, 2절은 빈공간 도약 시 B 스플라인 기반 도약과 베지어 기반 도약비교가 있으며 3절에는 스레드, 블록 선택 알고리즘으로 되어있다.



[그림 5-1] ABDOMEN 가시화 영상.

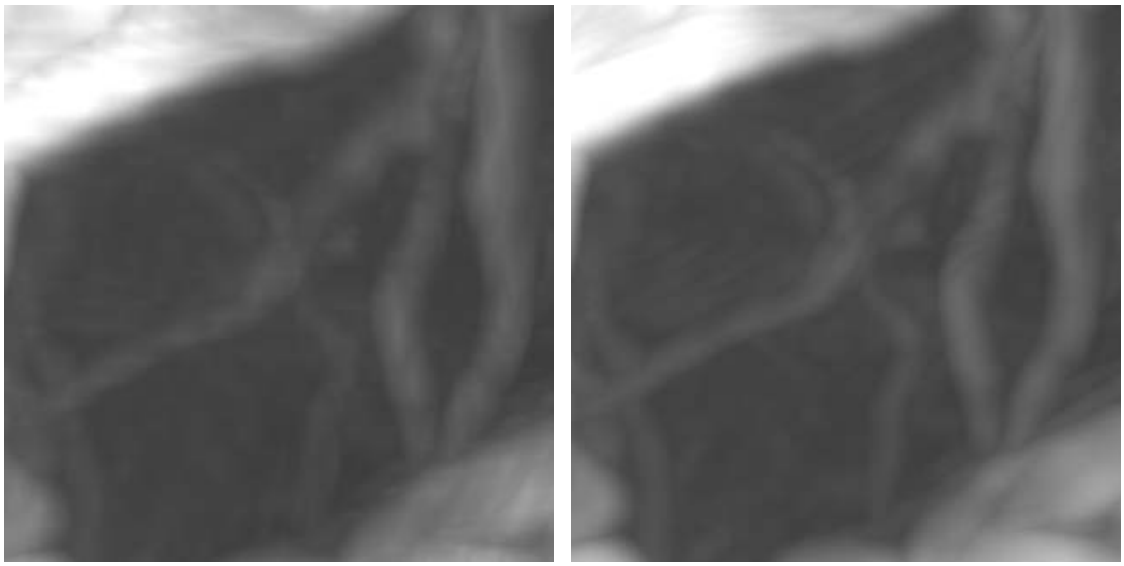
[그림 5-2] HEAD 가시화 영상.

[그림 5-3] LOWER 1 가시화 영상.

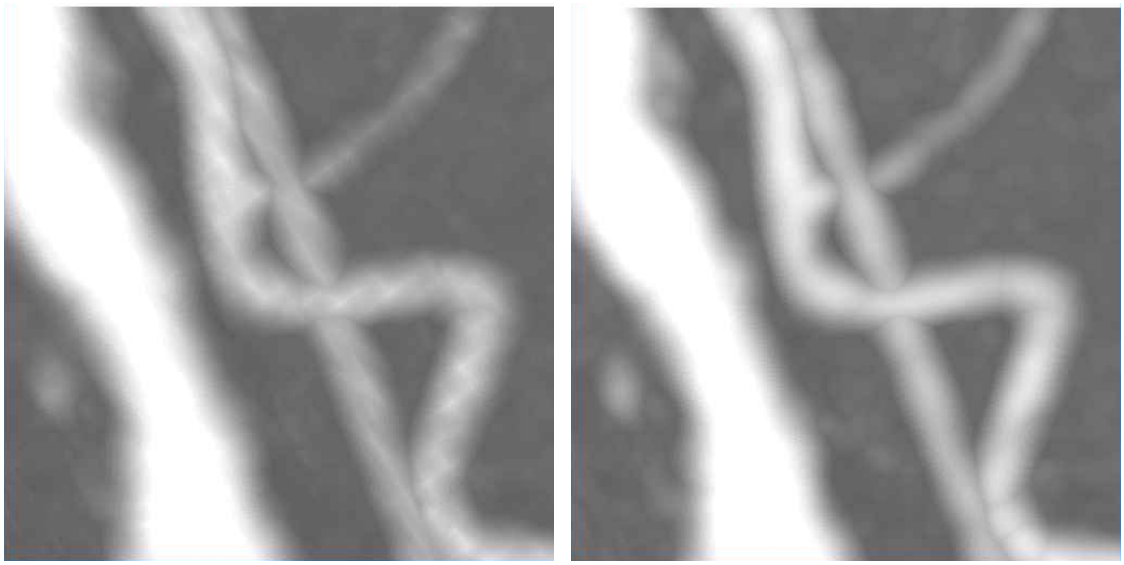
[그림 5-4] LOWER 2 가시화 영상.

[그림 5-5] LUNG 가시화 영상.

[그림 5-6] RIVER 가시화 영상.



[그림 5-7] 선형 보간과 삼차 보간 비교 결과 영상 1.



[그림 5-8] 선형 보간과 삼차 보간 비교 결과 영상 2.

결과 영상을 보면 골격부 형태와 연조직(soft tissue)이 잘 보존되어 있는 것을 볼 수 있다. [그림 5-7]과 [그림 5-8]은 선형 보간과 삼차 보간의 결과 영상을 확대하여 도시하였다. 왼쪽은 선형 보간이며 오른쪽은 삼차 보간이다. 선형 보간을 사용하여 샘플링하여 렌더링 된 영상의 경우는 혈관의 모습이 사각형으로 각진 모습이 뚜렷하게 나온다. 그에 반해 오른쪽의 삼차 보간을 이용하여 샘플링을 수행한 렌더링 영상은 각진 부분 없이 부드럽게 표현이 돼 있음을 볼 수 있다. 또한 B 스플라인 근사가 아닌 B 스플라인 보간을 이용하여 샘플링을 수행했기 때문에 B 스플라인의 오버 블러한 현상을 방지하여 원 볼륨 데이터에 왜곡이 발생하지 않음도 확인할 수 있다.

제 2 절 빈공간 도약 결과

2절에서의 모든 실험은 볼륨 데이터의 z축을 기준으로 180도를 회전시키며 볼륨의 중심을 바라보도록 렌더링을 수행하였다. 또한 빈공간 도약 블록의 크기는 4이다.

1) 베지어 빈공간 도약

[표 5-1] 빈공간 도약 시 성능 향상 결과

Data 512*512 (depth)	빈공간 도약 없음		B 빈공간 도약		Bezier 빈공간 도약	
	속도 (ms)	도약 비율	속도 (ms)	도약 비율	속도 (ms)	도약 비율
ABDOMEN_01 (277)	616.47	0	279.23	0.5268	163.03	0.7057
ABDOMEN_02 (110)	235.74	0	111.86	0.5008	59.20	0.7162
HEAD (552)	1294.49	0	586.33	0.5317	299.22	0.7248
LOWER_01 (583)	1342.89	0	625.94	0.5352	324.19	0.7206
LOWER_02 (582)	1327.12	0	597.47	0.5675	408.16	0.6820
LIVER_01 (370)	898.92	0	365.15	0.5706	194.75	0.7354
LIVER_02 (370)	759.36	0	330.38	0.5637	214.85	0.7266
LUNG_01 (528)	1304.03	0	452.33	0.5881	266.04	0.7421
LUNG_02 (528)	1191.70	0	462.55	0.5753	266.61	0.7336

[표 5-1]은 여러 볼륨 데이터를 가시화한 평균 렌더링 속도와 빈공간 도약 비율이다. 도약 비율은 광선이 진행하면서 만나는 모든 블록에 대한 스킵한 블록에 비이다. 빈공간 도약을 적용하지 않으면 매우 느린 렌더링 성능을 보여준다. B 스플라인 기반의 빈공간 도약 수행 시 약 2배 이상의 렌더링 속도 향상이 있으며 도약 비율 역시 블록의 절반 이상을 스킵한다. 베지어 기반

의 빈공간 도약의 경우 B 스플라인 기반의 도약에 비해 약 30%이상의 속도와 도약 비율 향상이 있음을 볼 수 있고 빈공간 도약을 적용하지 않은 방법에 비해 약 4배가량 성능 향상이 있다.

2) 초기값 설정

[표 5-2] 초기값 설정 시 성능 향상 결과

Data_512*512 (depth)	Bezier 빈공간 도약		Bezier 빈공간 도약(초기값)	
	속도(ms)	도약비율	속도(ms)	도약비율
ABDOMEN_01 (277)	163.03	0.7057	132.76	0.8032
ABDOMEN_02 (110)	59.20	0.7162	48.48	0.8085
HEAD (552)	299.22	0.7248	207.93	0.8156
LOWER_01 (583)	324.19	0.7206	274.38	0.8121
LOWER_02 (582)	408.16	0.6820	392.39	0.7543
LIVER_01 (370)	194.75	0.7354	156.80	0.8287
LIVER_02 (370)	214.85	0.7266	146.56	0.8205
LUNG_01 (528)	266.04	0.7421	223.04	0.8224
LUNG_02 (528)	266.61	0.7336	231.41	0.8240

[표 5-2]는 3장에서 서술한 초기값을 적용한 렌더링을 비교했다. 볼륨의 중간에서 먼저 샘플링 한 뒤 렌더링을 수행하여 빈공간 스킵 비율을 높이는 방법이다. 광선 투사법 시작 시 높은 밀도값으로 기대되는 볼륨의 중심을 먼저 샘플링하고 렌더링을 수행했기 때문에 도약비율이 약 10% 증가하는 것을 볼 수 있다.

3) 베지어 서브디비전 빈공간 도약

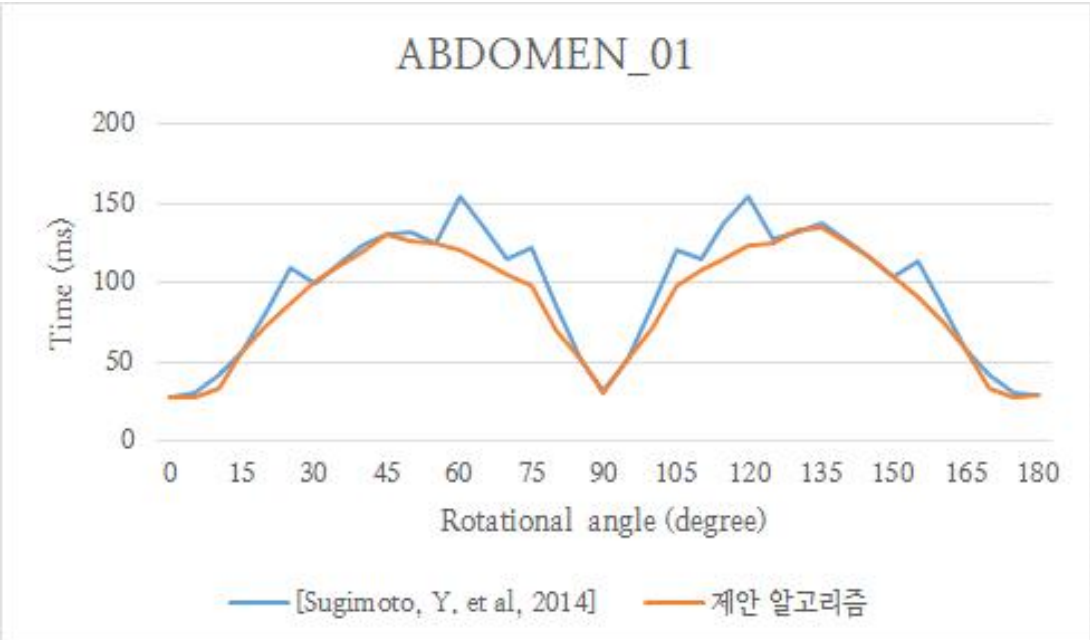
[표 5-3] 서브디비전 기반 도약 시 성능 향상 결과

Data_512*512 (depth)	Bezier 빈공간 도약			Subdivision 빈공간 도약		
	속도 (ms)	도약 비율	m/M 비율	속도 (ms)	도약 비율	m/M 비율
ABDOMEN_01 (277)	132.76	0.8032	0.4111	115.66	0.8232	0.3811 0.9207
ABDOMEN_02 (110)	48.48	0.8085	0.4483	43.01	0.8245	0.4242 0.9418
HEAD (552)	207.93	0.8156	0.4393	189.13	0.8284	0.4154 0.9407
LOWER_01 (583)	274.38	0.8121	0.3994	243.72	0.8305	0.3657 0.9063
LOWER_02 (582)	392.39	0.7543	0.4025	331.36	0.7834	0.3650 0.8887
LIVER_01 (370)	156.80	0.8287	0.4664	134.23	0.8439	0.4376 0.9348
LIVER_02 (370)	146.56	0.8205	0.4645	127.14	0.8372	0.4351 0.9331
LUNG_01 (528)	223.04	0.8224	0.4107	189.44	0.8406	0.3813 0.9216
LUNG_02 (528)	231.41	0.8240	0.4132	190.86	0.8426	0.3827 0.9193

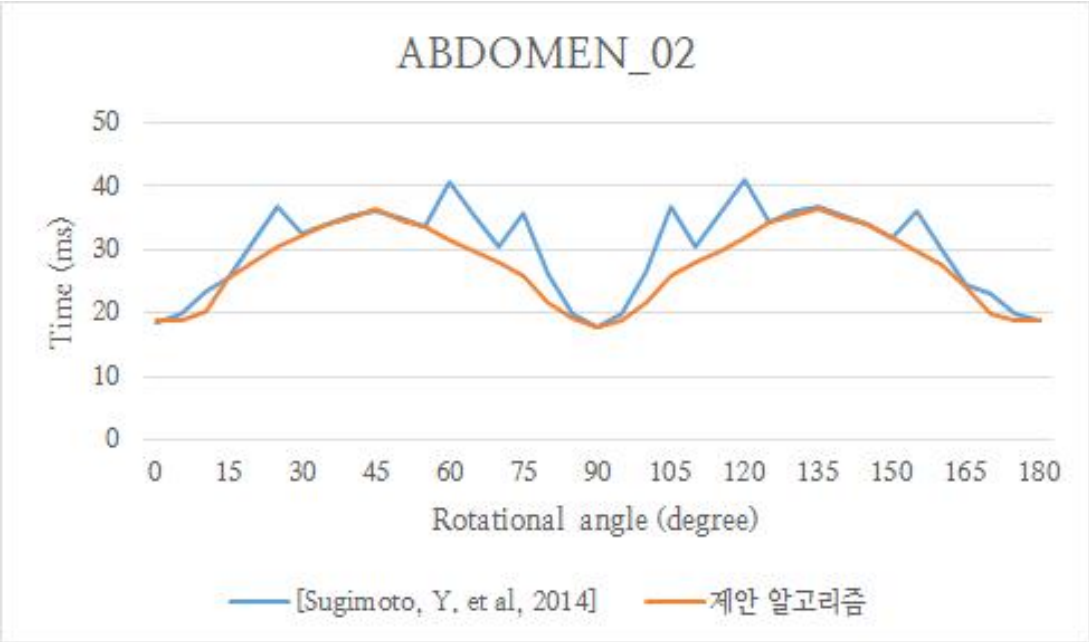
마찬가지로 3장에서 서술한 하나의 베지어 곡선을 두 개의 베지어 곡선으로 분리하여 빈공간 도약을 수행한 결과이다. 표에 나온 m/M 비율은 B 스플라인의 제어점으로 구한 min, max 차이를 1로 기준으로 하여 베지어 제어점으로 구한 min, max의 차를 상대적인 비율로 나타낸 것이다. [표 5-3]을 보면 베지어의 경우 B 스플라인에 비해서 최소값과 최대값의 차이가 절반 이하로 줄어든 것을 볼 수 있다. 서브디비전의 경우는 두 개의 m/M 비율이 나타나 있는데 전자는 B 스플라인 기준의 비율이며 후자는 베지어 곡선 기준에 대한 비율이다. 이 비율에서도 알 수 있듯이 한 번의 서브디비전을 수행 시 약 10% 정도의 성능 향상을 기대할 수 있다. 여러 번 서브디비전을 수행하여 더욱더 타이트한 경계박스를 얻을 수 있지만 서브디비전을 수행할 때마다 효율은 떨어지며 메모리 사용량도 고려하여 수행해야 한다.

제 3 절 CUDA 스레드, 블록 비교

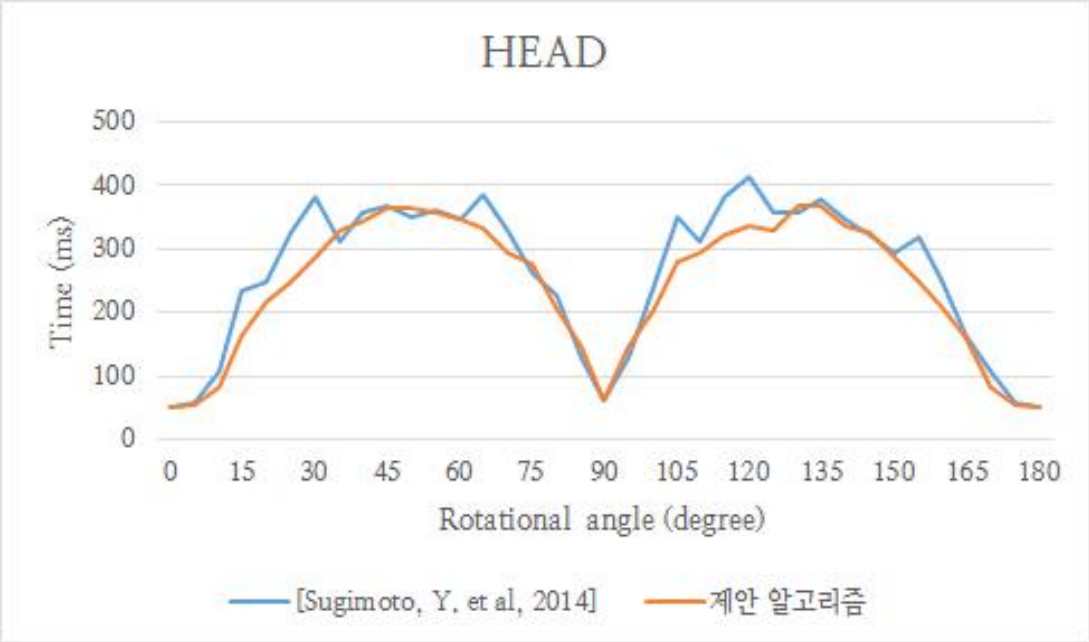
기존연구[Sugimoto, Y. et al, 2014]와 본 연구에서 제안한 알고리즘의 결과 그래프이다. xy평면을 수직으로 바라보면서 z축 기준으로 180도를 15도 단위로 피벗 회전하여 렌더링 시간을 측정하였다.



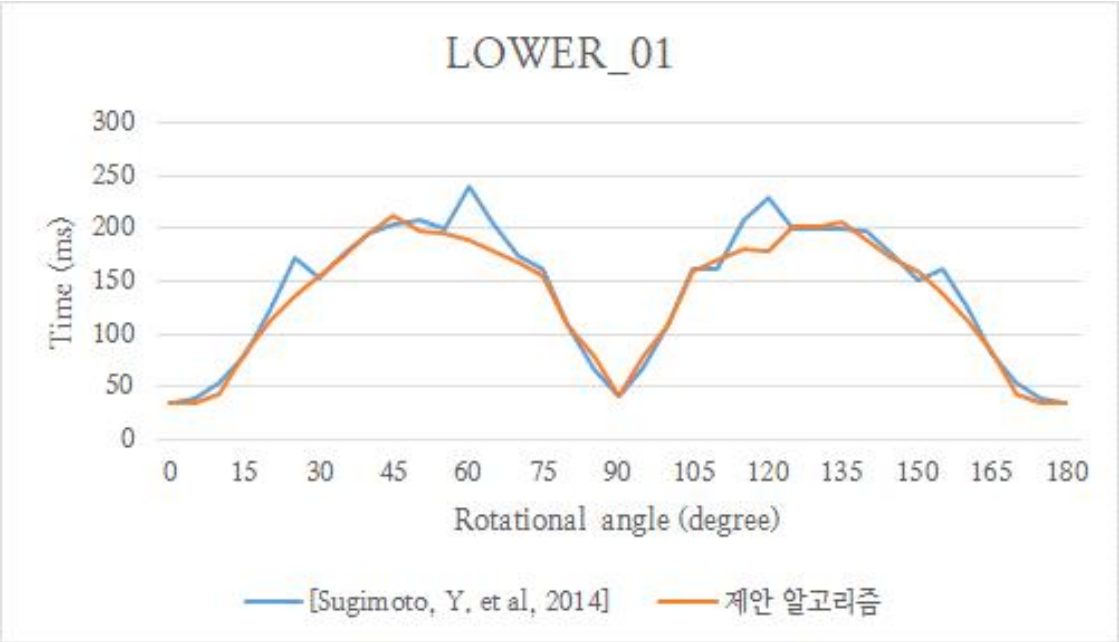
[그림 5-9] ABDOMEN 1 렌더링 속도 그래프.



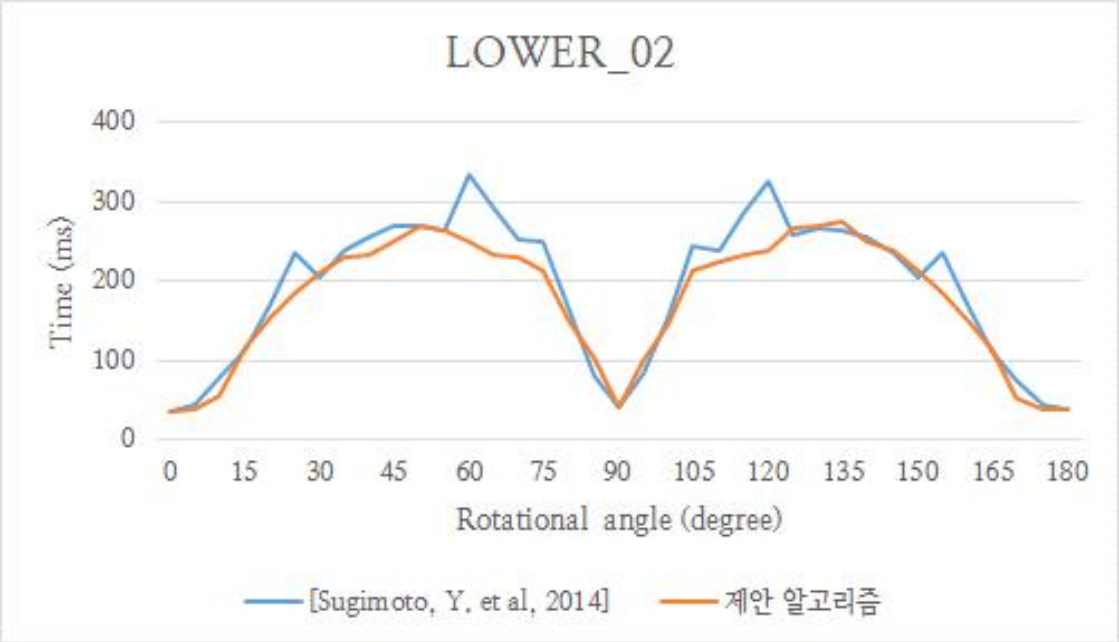
[그림 5-10] ABDOMEN 2 렌더링 속도 그래프.



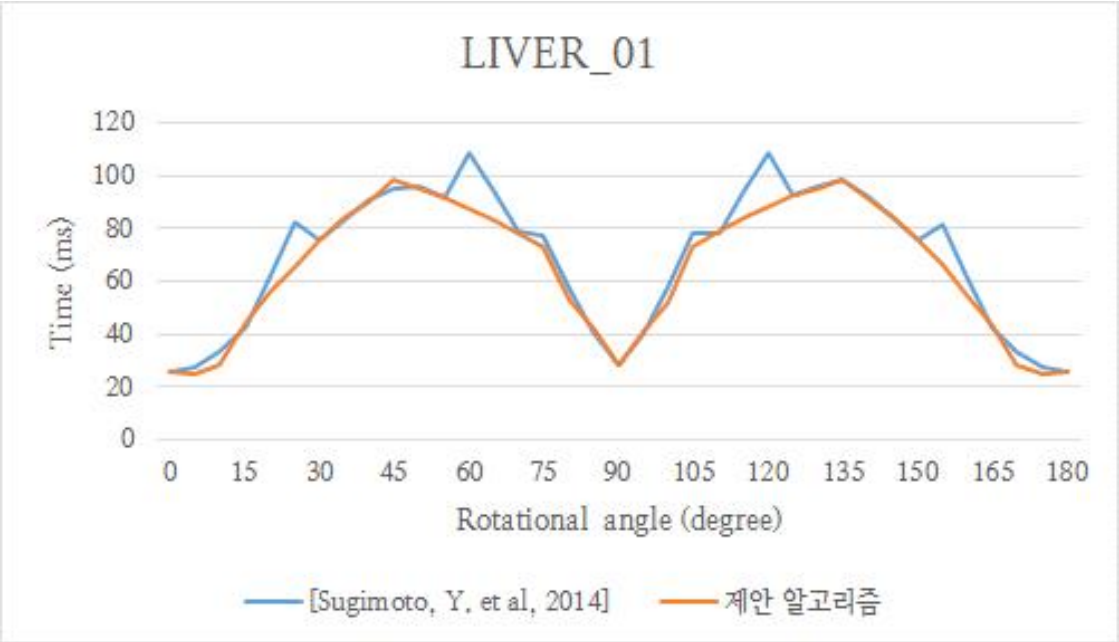
[그림 5-11] HEAD 렌더링 속도 그래프.



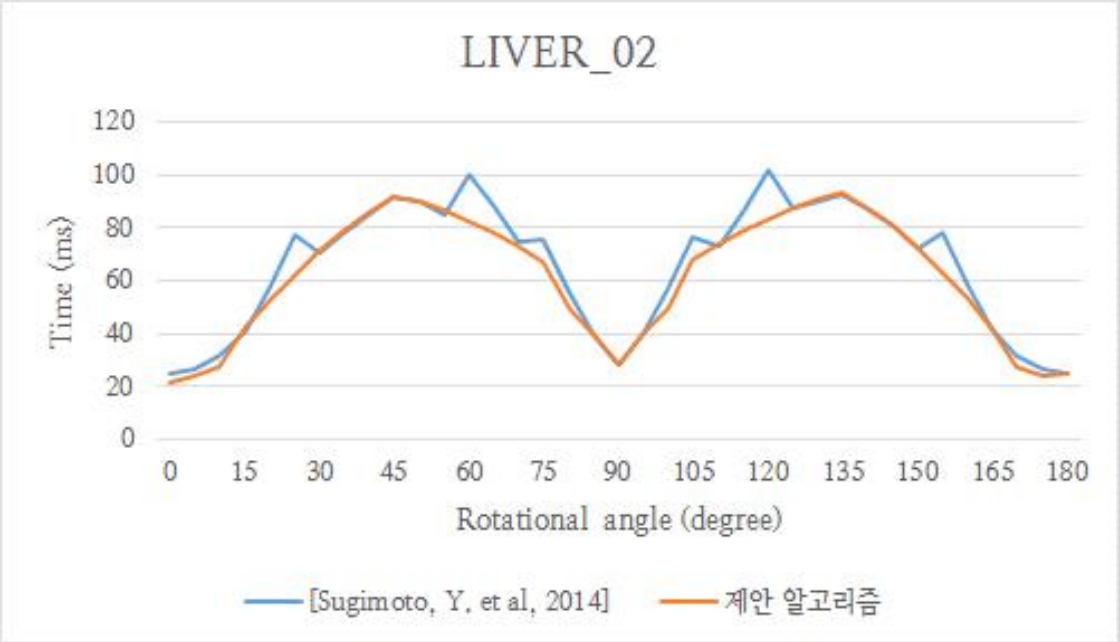
[그림 5-12] LOWER 1 렌더링 속도 그래프.



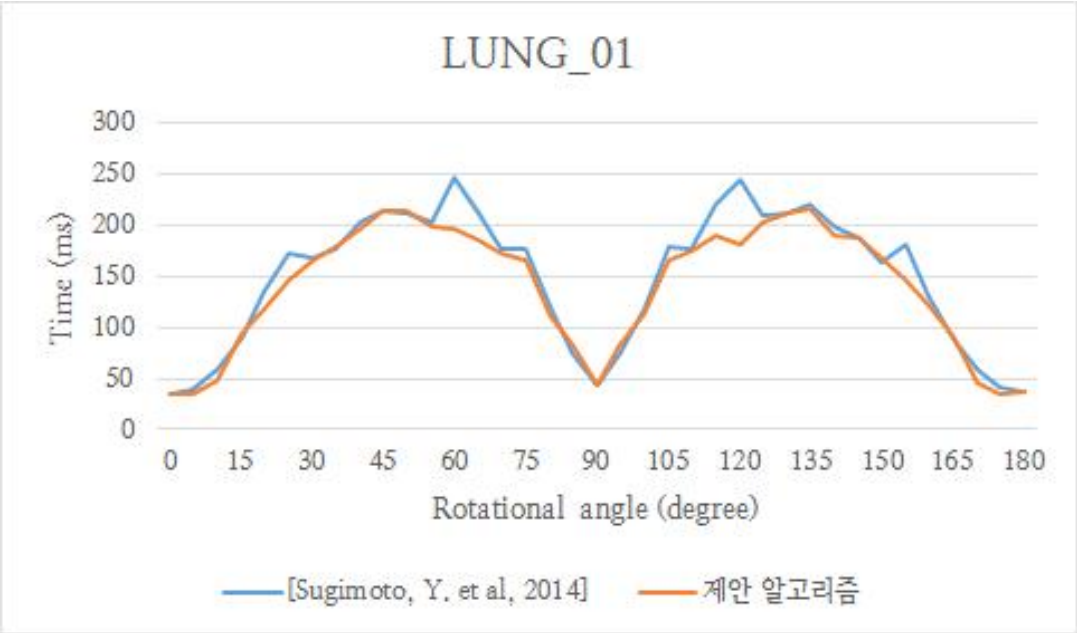
[그림 5-13] LOWER 2 렌더링 속도 그래프.



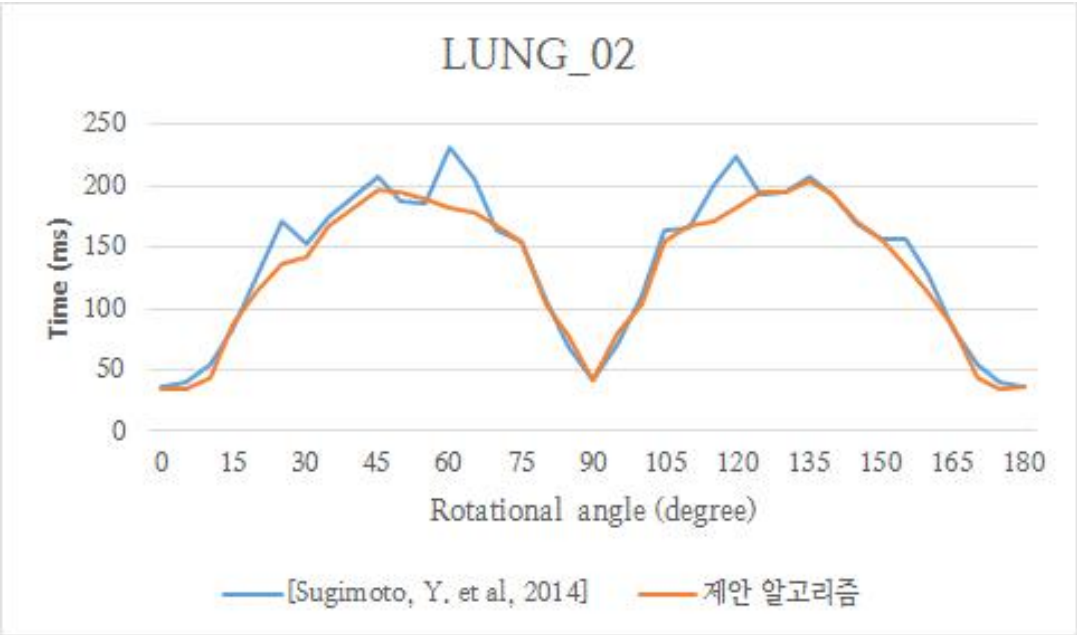
[그림 5-14] LIVER 1 렌더링 속도 그래프.



[그림 5-15] LIVER 2 렌더링 속도 그래프.



[그림 5-16] LUNG 1 렌더링 속도 그래프.



[그림 5-17] LUNG 2 렌더링 속도 그래프.

기존 연구의 경우 15도 간격을 스레드 구조를 선택의 기준으로 하여 일률적으로 적용하였다. 따라서 15도 배수마다 비효율이 발생하여 렌더링 시간이 증가하여 그래프가 튀는 것을 볼 수 있다. 본 연구에서는 블록, 스레드와 캐시 라인을 고려하여 최소한의 메모리 참조로 볼륨 렌더링을 수행하는 타일 구조를 제시하였다. 제시한 방법은 15도 경계 부분에서도 그래프가 완만한 곡선을 그리는 것을 확인할 수 있다. 이는 최적의 효율임을 나타내고 있는 것이다. 하지만 동적 블록 스레드 구조 선택은 영상의 가로축과 볼륨의 x축과 밀접한 연관이 있기 때문에 렌더링을 볼륨의 yz평면을 바라보게 수행하면 스레드 구조와 상관없이 낮은 메모리 참조 효율이 나타나게 되는 문제점을 안고 있다.

제 6 장 결 론

본 연구는 B 스플라인 보간 기반의 고화질 볼륨 가시화 알고리즘을 제안하였다. 기존의 B 스플라인이 보간 함수가 아닌 근사 함수인 점을 인지하고, 제어점을 이동하여 데이터 왜곡 없는 B 스플라인 보간 방법을 사용하였고, 결과적으로 고화질 볼륨 가시화 영상을 렌더링 할 수 있었다. 또 연상량이 많은 고차 보간을 이용하는 고화질 볼륨 가시화의 속도가 느린 점을 향상시키기 위해 크게 2가지 가속화 기법을 제안하였다.

대표적인 볼륨 가시화 가속화 기법인 빈공간 도약 알고리즘을 적용을 위해 B 스플라인의 블록포 성질을 이용하여 최대값을 얻었고, 이를 이용하여 빈공간 도약을 수행하였다. 또한 B 스플라인을 베지어 곡선으로 변환하여 영상의 손실이 없는 타이트한 경계 박스를 생성해 빈공간 도약의 효율을 개선했다. 더 나아가 베지어 곡선을 조각적 베지어 곡선 조각으로 분해하는 서브디비전을 통해 더욱 높은 효율의 빈공간 도약을 수행 하여 우수한 렌더링 속도을 얻었다.

두 번째로 GPU 병렬처리 기반의 프로그래밍 시 성능에 큰 영향을 미치는 메모리 액세스 패턴을 고려하여, 워프 내의 스레드가 데이터의 물리적 주소값을 인접하게 요구하도록 영상의 가로 방향인 크로스 벡터와 볼륨의 x축이 이루는 사잇각으로 스레드 구조를 선택하는 알고리즘을 제안하였다. 이를 통해 메모리 참조 비효율을 줄여 더 우수한 렌더링 속도를 얻었다. 그 결과로 상용 GPU를 이용하여 대화적 속도의 고화질 볼륨 가시화가 가능하였다.

참 고 문 헌

1. 국내문헌

계희원. (2011). 카디널 보간을 이용한 효율적인 고화질 볼륨 가시화.『한국 멀티미디어 학회 논문지』, 14(3), 339-347.

신용하, 계희원. (2016). B 스플라인 보간을 이용한 고화질 볼륨 가시화.『한국 컴퓨터 그래픽스 학회 논문지』, 22(3), 1-9.

2. 국외문헌

Engel, K., Hadwiger, M., Kniss, J., Rezk-Salama, C., & Weiskopf, D. (2006). Real-time volume graphics. CRC Press.

Farin, G. (2014). Curves and surfaces for computer-aided geometric design: a practical guide. Elsevier.

Fung, W. W., Sham, I., Yuan, G., & Aamodt, T. M. (2007). Dynamic warp formation and scheduling for efficient GPU control flow. In Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture, 407-420.

Hadwiger, M., Huaser, H., & Möller, T. (2003). Quality issues of hardware-accelerated high-quality filtering on pc graphics hardware.

Hadwiger, M., Theußl, T., Hauser, H., & Gröller, E. (2001). Hardware-accelerated high-quality filtering on PC hardware. In Proceedings of Vision, Modeling, and Visualization, 105-112.

Keys, R. (1981). Cubic convolution interpolation for digital image processing. IEEE transactions on acoustics, speech, and signal processing, 29(6), 1153-1160.

Kruger, J., & Westermann, R. (2003). Acceleration techniques for GPU-based volume rendering. In Proceedings of the 14th IEEE Visualization VIS2003, 38.

Lee, B., Yun, J., Seo, J., Shim, B., Shin, Y. G., & Kim, B. (2010). Fast high-quality volume ray casting with virtual samplings. IEEE Transactions on visualization and computer graphics, 16(6), 1525-1532.

Levoy, M. (1988). Display of surfaces from volume data. IEEE Computer graphics and Applications, 8(3), 29-37.

Levoy, M. (1990). Efficient ray tracing of volume data. ACM Transactions on Graphics (TOG), 9(3), 245-261.

Marsalek, L., Hauber, A., & Slusallek, P. (2008). High-speed volume ray casting with CUDA. In Interactive Ray Tracing, RT 2008. IEEE Symposium, 9-10.

Meijering, E. H., Niessen, W. J., & Viergever, M. A. (2001). Quantitative evaluation of convolution-based methods for medical image interpolation. Medical image analysis, 5(2), 111-126.

Mihajlovic, Z., Budin, L., & Radej, J. (2004). Gradient of B-splines in volume rendering. In Electrotechnical Conference, 2004. MELECON 2004. Proceedings of the 12th IEEE Mediterranean, 1, 239-242.

Narasiman, V., Shebanow, M., Lee, C. J., Miftakhutdinov, R., Mutlu, O., & Patt, Y. N. (2011, December). Improving GPU performance via large warps and two-level warp scheduling. In Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture, 308-317.

Ruijters, D., & Thévenaz, P. (2010). GPU prefilter for accurate cubic B-spline interpolation. The Computer Journal, 55(1), 15-20.

Shen, R., & Boulanger, P. (2007). Hardware-accelerated volume rendering for real-time medical data visualization. Advances in Visual Computing, 801-810.

Sigg, C., & Hadwiger, M. (2005). Fast third-order texture filtering. GPU gems, 2, 313-329.

Sugimoto, Y., Ino, F., & Hagihara, K. (2014). Improving cache locality for GPU-based volume rendering. Parallel Computing, 40(5), 59-69.

Unser, M. (2002). Splines: a perfect fit for medical imaging. Progress in Biomedical Optics and Imaging, 3(22), 225-236.

ABSTRACT

GPU-based Acceleration Technique for High-Resolution Volume Visualization Using High-Order Interpolation

Shin, Yong-Ha

Major in Information System Engineering

Dept. of Information System Engineering

The Graduate School

Hansung University

Linear interpolation is usually used a basic sampling method for volume visualization. This method generates good images but in the case of precision visualization such as blood vessels and neural tube it is inferior to our high expectation. Therefore, medical image software needs high quality image. In this paper, B spline based tri-cubic interpolation is used for the re-sampling step. The conventional B spline is an approximate function and does not cross the control points. So we moved the control points and the curve crosses the original control points. Such a cubic interpolation needs a much higher computational complexity than linear interpolation and therefore requires an acceleration technique. In general, we apply the empty space leaping acceleration method to improve speed by reducing unnecessary operations in the

rendering step. The maximum value for each block must be calculated to determine the empty space. However, the cubic interpolation is a curve, it is difficult to obtain an accurate maximum value. The B spline has convex hull, so the value of the control point can be used as the maximum value. However, the maximum value obtained using the B spline control point is conservative. Consequently, the empty space leap efficiency is low and the rendering speed is reduced. In this paper, we decompose the B spline curve into a divisioned Bezier curve to obtain a tight maximum and improve the rendering speed. Also, proposed a block and thread structure that efficiently performs memory references in GPU parallel processing through memory coalescing. As a result, tri-cubic interpolated volume rendering is possible in interactive speed.

KEYWORD: Medical Image Rendering, Volume Visualization, Real-Time Rendering, B Spline Interpolation, GPU Rendering.