RGBA 데이터 압축을 이용한 병렬 볼륨 렌더링 가속 기법†

김형래[0] 이원종 김정우 박우찬 한탁돈
연세대학교 컴퓨터과학과
kimhr@cs.yonsei.ac.kr[0], {airtight, mosh, chan, hantack}@kurene.yonsei.ac.kr

Accelerating Parallel Volume Rendering by RGBA Data Compression

Hyung-Rae Kim[0] Won-Jong Lee Jung-Woo Kim Woo-Chan Park Tack-Don Han
Dept. of Computer Science, Yonsei University

요 약

볼륨 렌더링은 물체의 겉면만이 아니라 내부에 있는 모든 3차원 데이터를 이용해서 렌더링 하는 기법이다. 따라서 기존의 폴리곤 렌더링에선 불가능했던 물체 내부에 대한 표현이 가능하기 때문에 과학, 의료 분야등 물체 전체에 대한 데이터 처리가 필요한 곳에서 많이 쓰이고 있다. 하지만 이러한 볼륨데이터의 크기는 일반적으로 1024^3Bytes 이상이기 때문에 기존의 단일 그래픽 가속기로는 메모리 크기나 연산 능력면에서 처리하기에 한계가 있다. 따라서 본 논문에서는 이런 기가급 볼륨데이터를 처리하기 위한 병렬 볼륨 렌더링 구조를 제시하고, 전송된 부분 이미지 합성을 위한 블렌딩 순서를 결정하는 시점 추적 (*point-tracking*)기법과 네트웍에 의한 성능저하를 최소화 할 수 있는 '프레임간 유사성(frame-to-frame coherency)을 이용한 RGBA데이터 압축기법'을 제안한다.

1. 서 론

볼륨렌더링은 물체의 겉면뿐만 아니라 내부에 있는 데이터까지 처리하는 렌더링 기법이다. 과거의 경우 볼륨데이터는 광선이 통과하는 곳에 위치한 복셀들을 렌더링하는 기법인 레이캐스팅[1]으로 렌더링했지만, 하드웨어적인 지원을 받을 수가 없었기 때문에 렌더링시 시스템에 많은 부하가 요구되어 고성능 워크스테이션에서만 렌더링이 가능했었다. 하지만 최근 들어서는 이 기법 대신 하드웨어 가속지원을 받을 수 있는 3차원 텍스쳐 기법을 많이 사용하고 있다. 3차원 텍스쳐 기법은 기존의 볼륨데이터 내에 여러장의 슬라이스를 생성한 후, 여기에 슬라이스에 따른 3차원 텍스쳐 데이터를 생성해 맵핑함으로써 렌더링을 하는 방법이다[2]. 이런 볼륨렌더링에서 쓰이는 데이터의 양은 기존의 폴리곤 기반의 렌더링에서 사용되는 데이터보다 훨씬 크다. 1024^3샘플, 샘플당 8bit 데이터의 경우 1024*1024*1024 Bytes, 즉 1GBytes이상의 양을 갖게 되는데 이는 현재 가장 빠른 그래픽 가속기의 메모리의 8배에 달하는 양이다. 현재 그래픽 가속기가 많은 발전을 이루었다고 해도 메모리 대역폭이나 메모리의 한계, 연산량의 증가 등으로 인해 이러한 데이터의 처리는 불가능하다. 따라서 이런 데이터의 처리를 위해서 데이터를 분할처리 할 수 있는 클러스터기반의 렌더링이 사용된다. 데이터를 작게 나눌 경우, 한 컴퓨터에 주어지는 데이터의 양이 작아지기 때문에 메모리의 한계와 연산능력에 의한 한계를 극복할 수 있다. 그리고 각 노드의 그래픽 시스템만 최신의 것으로 바꾸면, 시스템 전체의 성능도 같이 향상되므로 저렴한 비용으로 계속적인 사용이 가능한 장점도 있다. 따라서 현재 기가급의 볼륨렌더링에 대한 연구는 이런 클러스터 환경을 이용한 병렬 볼륨 렌더링기법이 각광을 받고 있다[3][4].

병렬 볼륨 렌더링 기법은 네트웍을 통해 연결된 여러 대의 렌더링 서버가 각각에 해당하는 부분 볼륨 데이터를 클라이언트의 신호에 따라 처리하여 생성된 이미지를 전송하면, 클라이언트에서 최종적인 이미지를 합성하는 방법이다. 본 논문에서는 병렬 볼륨 렌더링을 위한 구조를 보이고, 정확한 볼륨 이미지 합성을 위한 '시점 추적(point-tracking)기법'과 제한된 네트웍의 대역폭을 효율적으로 사용할 수 있는 '프레임간 유사성(frame-to-frame coherency)에 의한 RGBA데이터 압축 기법'을 제안한다.

2. 병렬 볼륨 렌더링

병렬 볼륨 렌더링에는 분할된 데이터가 쓰이게 된다. 2.1절에서는 볼륨데이터의 분할에 대해서 다루고, 2.2절에서는 이를 바탕으로한 병렬 볼륨 렌더링 구조를 제안한다.

2.1 볼륨 데이터의 분할

볼륨 데이터는 CT촬영기나 3차원 스캐너에 의해서 생성된다. 작은 것은 128^3sample의 크기부터 1024^3sample 크기 이상의 것까지 있다. 현재의 최신 PC 그래픽 가속기는 256^3정도의 볼륨데이터는 무리없이 처리할 수 있다[5]. 하지만 최근 사용되는 3차원 텍스쳐 기법을 사용하기 위해선 모든 텍스쳐 데이터를 그래픽가속기의 텍스쳐 메모리에 읽어들여야 하는데 1024^3이상의 기가바이트 볼륨데이터의 경우, 최신 그래픽 가속기라 하더라도 128MBytes의 텍스쳐 메모리로는 모든 데이터를 읽어들이는 것은 불가능하기 때문에 데이터를 분할하여 처리하여야 한다. 1024^3sample을 8개의 정방형 볼륨데이터로 나누면 하나당 512^3sample의 크기를 가지게 된다. 각 sample당 8bit로 가정하면 단일 그래픽 가속기가 읽어야할 텍스쳐 데이터의 양은 각각 128MBytes이 되기 때문에 구현이 가능하다.

2.2 제안하는 클러스터 기반 병렬 볼륨 렌더링 구조

클러스터 환경의 컴퓨터는 각각이 서로 독립적인 연산을 담당하기 때문에 많은 데이터를 보다 효율적으로 처리할 수 있다. 기가바이트 이상의 큰 볼륨데이터의 경우, 이를 각각의 클러스터 노드가 렌더링할 수 있을 만큼 작게 나누어 처리하면, 텍스쳐 메모리 한계에 의한 문제를 해결할 수 있다. 그러나 지금까지의 병렬 볼륨 렌더링 성능은 512^3크기의 데이터를 5프

†본 연구는 한국과학기술평가원의 '99 국가 지정연구실 과제 (2000-N-NL-01-C-133)로 수행되었음

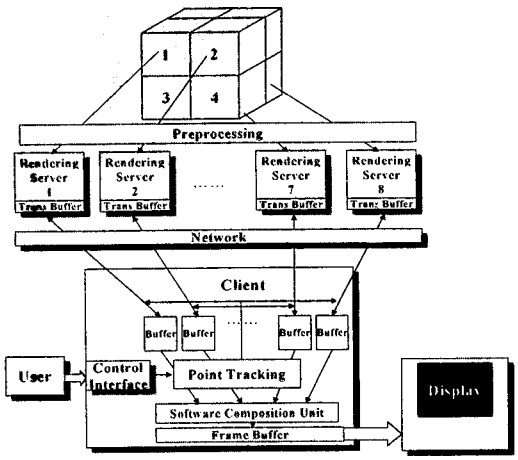

그림 1 제안하는 구조

레임 정도로 렌더링하는게 한계였다[3][4]. 이는 하드웨어의 한계와 하드웨어의 병목점을 효율적으로 해결하지 못한 결과이다. 따라서 최신 하드웨어를 이용하고, 일어날 수 있는 여러 병목점을 해결할 경우, 보다 높은 성능을 낼 수 있다.

그림 1은 제안하는 병렬 볼륨 렌더링 구조를 나타낸 것이다. 이 구조는 1024^3의 데이터를 초당 30프레임으로 처리하는 것을 목표로 설계되었다. 각 렌더링 서버는 네트웍을 통해서 클라이언트와 연결되어 있다. 렌더링 서버는 512^3의 데이터를 초당 30프레임으로 처리할 수 있는 능력을 가지고 있고, 네트워크는 한 노드당 미리넷(192MBytes/sec) 이상의 전송속도를 가지고 있다고 가정한다. 클라이언트에서는 사용자의 조작에 의한 신호가 들어오면 그에 따른 시점 변환(viewing transformation)을 한 후, 각 렌더링 서버에 해당되는 시점 변환에 따른 정보를 보낸다. 렌더링 서버는 클라이언트에서 전송되어 온 값을 바탕으로, 가지고 있는 부분 볼륨데이터에 대한 렌더링을 수행한다. 렌더링이 완료되면 렌더링 서버는 네트웍을 통해서 생성된 이미지를 클라이언트로 보낸다. 클라이언트는 전송되어 온 이미지들을 각 서버별로 존재하는 저장 버퍼로 받아서 합성한 후 최종 이미지를 디스플레이에 보여주게 된다. 최종 이미지를 합성할 때는 이미지의 순서가 중요한데 이를 위해서 시점 추적 기법을 사용한다.

네트웍을 통해 이미지를 전송해야 하므로, 네트웍에 의한 지연을 감안하여, 클라이언트는 각 렌더링 서버에서 전송되는 이미지를 임시로 저장하는 한 프레임 크기의 저장 버퍼를 두게 된다. 한 프레임을 저장하고 두번째 프레임이 전송되어 왔을 때 렌더링을 시작하므로, 최종적으로 보이는 이미지는 바로 전 프레임의 이미지이다.

3. 알고리즘

볼륨렌더링의 경우 생성된 이미지는 눈에 보이는 순서대로 블렌딩을 해야 정확한 이미지가 생성되므로, 이를 위해서 여러대의 렌더링 서버에서 전송되어온 이미지를 가시 순서에 맞게 블렌딩하는 시점 추적 기법을 3.1절에서 제안한다. 그리고 병렬렌더링 환경은 특성상 각 클러스터에서 초기 데이터를 읽어들이는 부분과 최종적으로 생성한 이미지를 클라이언트에 보내는 부분에서 네트웍의 제한된 대역폭에 의한 병목점이 존재한다. 3.2절에서는 렌더링의 최종단계인 이미지 전송시 발생하는 병목현상을 최소화 시킬수 있는 방법인 '프레임간 유사성(frame-to-frame coherency)에 의한 RGBA데이터 압축기법'을 제안한다.

3.1 시점 추적(Point-Tracking)기법

볼륨 렌더링의 경우, 들어온 이미지는 가시 순서에 맞게 블렌딩

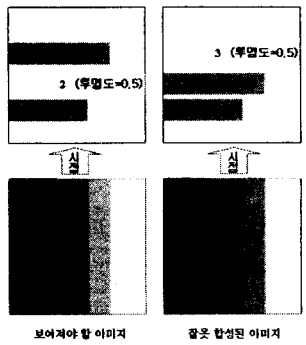

그림 2 이미지 합성의 예

을 수행해야 한다. 그렇지 않을 경우, 잘못된 이미지가 출력되게 된다. 그림 2의 왼쪽 그림은 올바른 순서로 들어온 것을 블렌딩했을 때이다. 하지만 블렌딩해야 할 이미지가 바뀐 순서로 들어오면 α값에 따라 오른쪽의 그림처럼 잘못된 이미지가 보이게 된다. 따라서 이 순서를 결정하기 위해서 순차적 블렌딩 기법[6]을 써야 한다. 하지만 이 방법은 단일 렌더링 서버에서 사용되게 제안한 방법으로, 이 방법을 병렬환경에 맞도록 변형시켜야 한다.

방법은 먼저 서브 볼륨 데이터를 가지고 있는 렌더링 서버 각각에 ID를 부여하고, 기준이 되는 렌더링 서버 ID(Point)를 정한다. 그 기준이 되는 Point의 위치가 사용자의 시점 변환에 따라 바뀌면, 바뀐 위치에 의해 나오는 순서대로 네트웍을 통해 전송된 이미지를 합성한다.

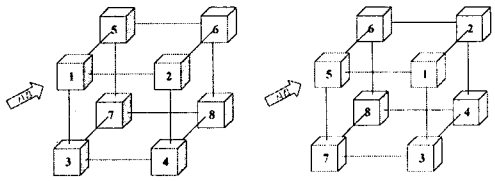

그림 3 시점 변환에 따른 순서변화

그림 3은 분산된 볼륨데이터가 각각의 렌더링 서버에 있는 것을 도식화 한 것이다. 그림과 같이 시점이 1번 렌더링서버에 가까운 곳에서 보고 있다면 전송되어온 이미지는 1→2→5→6→3→4→7→8의 렌더링 서버 순서로 합성된다. 다음에 사용자의 시점이 왼쪽으로 이동되면, 1이 2의 위치에 있다는 것을 벡터 연산에 의해 인식한 후, 5→1→6→2→7→3→8→4의 순서로 합성하게 된다. 이 순서는 눈에 보이는 순서대로 결정된다. 따라서 기준이 되는 point를 계속적으로 추적하면서, point의 위치와 현재 시점과의 위치만을 판단해 기준점을 정하면, 나머지 순서는 가시 순차적 기법에 의해서 결정된다. 결과적으로 1번의 변환 연산과 8번의 비교 연산에 의해서 합성될 이미지의 순서를 쉽게 정할 수 있다.

3.2 프레임간 유사성(frame-to-frame coherency)에 의한 RGBA 데이터 압축기법

각 렌더링 서버에서 처리된 이미지의 크기는 한 프레임당 약 1MBytes이다. 이런 데이터를 8대의 렌더링 서버가 초당 30장씩 보내기 위해서는 평균속도 240MBytes이상의 전송속도가 필요하다. 이는 현재 출시된 가장 빠른 상용 네트웍 'Myrinet'(평균속도 192MBytes/sec)[7]으로도 원활한 프레임의 구현이 불가능하다.

실시간으로 생성되는 장면에는 앞뒤 프레임별로 유사성을 가지고 있다. 또한 볼륨렌더링의 경우 데이터간의 의존도가 높아

렌더링된 부분 이미지 각각의 유사성도 높기 때문에 이미지를 압축할 경우 많은 효과를 볼 수 있다. 하지만 압축방법은 압축 효율과 압축을 위한 연산에 따른 상충관계가 존재하므로 실시간 처리가 가능한 적절한 방법을 사용해야 한다.

제안하는 방법은 먼저 압축을 위해서 먼저 각 RGBA채널별로 전 프레임과의 차를 계산하여 변하지 않은 값과 변한 값을 구분한다. 그 후 변하지 않은 값은 run-length-encoding 방법을 통해서 Zero Table에 넣고, 변한값은 그대로 전송 버퍼에 넣는다. 네트웍을 통해서 데이터를 전송할 때는, 이렇게 만들어진 전송 버퍼와 Zero Tabel을 같이 보낸다. 클라이언트에서는 받은 수신 버퍼 값을 조사해서 일반값이면 바로 저장하고, 0이 나오면 차례대로 Zero Table의 값을 인덱싱하여 불변값의 양을 읽어낸다. 그리고 불변값만큼 프레임버퍼를 건너뛴 후, 수신 버퍼에서 다음 값을 읽어들여서 프레임버퍼에 저장한다.

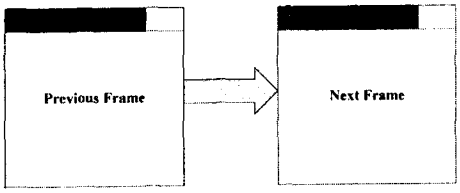

그림 4 프레임의 변화에 따른 픽셀 움직임

그림 4와 같이 6개의 픽셀이 다음 프레임에서 왼쪽으로 한 픽셀씩 움직이는 변화를 보였을 때, 이를 압축하지 않고 모두 보내면 6*4Bytes=24Bytes의 전송량이 필요하다. 하지만 이를 RGBA별로 압축하면 표 1과 같다.

표 1 압축에 의한 데이터 감소량

픽셀	1	2	3	4	5	6	전송버퍼	Zero Table
R	0	0	255	0	0	0	0 255 0	2 3
G	0	0	0	0	0	0	0	6
B	0	0	0	0	0	255	0 255	5
A	0	0	0	0	0	0	0	6
최종값							7Bytes	2*5=10Bytes

전송버퍼의 값은 인자하나당 1Byte를 차지하고, 테이블의 경우 2Bytes를 차지한다. 표에서 보듯이 압축 후 전체적으로 전송되어야 하는 양이 17Bytes가 됨을 알 수 있다. 이론적으로 대략 30%정도의 데이터 감소 효과가 나오는 것을 알 수 있다.

실제 실험은 RGBA값을 분리하지 않고 압축했을 때(ALL)와 각각의 채널을 분리해서 압축했을 때(RGBA)의 두가지 경우로 수행하였다. 결과는 그림5와 같다.

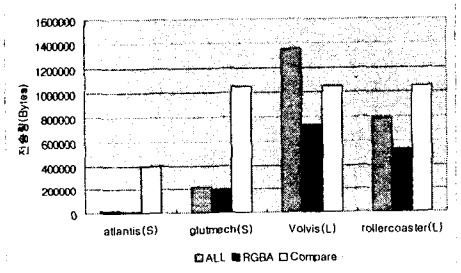

그림 5 실험 결과

그림 5의 가장 오른쪽에 있는 막대(Compare)는 압축되지 않은 원본데이터의 양을 나타낸다. 프로그램은 ATI사의 볼륨렌더링 예제 프로그램인 Volvis[5]와 일반적인 OpenGL-glut프로그램[8] 3가지를 사용하였다. 각각의 이미지는 그림 6에 나와있다. atlantis나 glutmech등은 비교적 화면의 움직임이 작은 영역에서 일어나는 프로그램(S)이고, Volvis나 rollercoaster등은 화면의 전영역에 걸쳐서 움직임이 일어나는 프로그램(L)이다. 위의 결과값에서 보여주듯이 제안하는 알고리즘을 쓸 경우 움직임이 적은 프로그램에서는 81~97%, 움직임이 큰 프로그램에서는 30~ 49%정도의 데이터량 감소효과가 있었다. 이는 이론적으로 예상한 30%를 웃도는 수치인데, 실제적으로 사용되는 프로그램에서 RGBA별로 많은 값이 중복되기 때문이라고 추측할 수 있다. 특히 볼륨렌더링 프로그램은 화면 전영역에 걸쳐서 이미지의 변화가 일어나지만, 제안한 알고리즘을 사용하였을 때 30퍼센트이상의 높은 압축율을 보이고 있다.

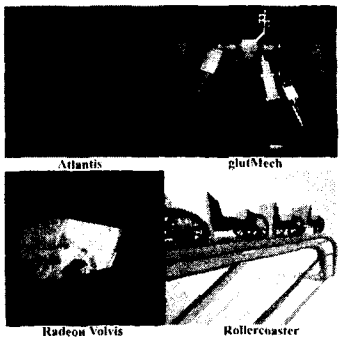

그림 6 프로그램 실행 화면

마지막으로 제안한 알고리즘을 쓸 경우, 최소 30%정도의 데이터 감소효과가 있기 때문에 1024^3의 볼륨데이터를 30프레임으로 구현하기 위해 필요한 240MBytes의 전송량을 168MBytes로 줄일 수 있다. 이는 최근에 사용되는 가장 빠른 네트웍의 속도(평균속도 192MBytes)를 밑도는 수치로서, 병렬볼륨렌더링에서 인터렉티브한 프레임율을 기대할 수 있다.

4. 결론 및 향후 계획

본 논문에서는 병렬볼륨렌더링의 구조를 보이고, 정확한 최종 이미지의 합성을 위한 '시점 추적 기법'과 제한된 네트웍에 의한 병목현상을 최소화하는 기법인 '프레임간 RGBA채널 유사성에 의한 압축기법'과 을 제안하였다. 현재는 제안한 구조에 대한 시뮬레이터를 제작 중에 있으며, 이를 이용할 경우, 기가급의 볼륨데이터를 필요로 하는 기상 관측 데이터처리, 인체 단층 촬영데이터 처리, 해수면의 흐름 데이터 처리 및 화산 폭발 데이터 시뮬레이션등의 여러 볼륨 애플리케이션을 저렴한 비용으로 처리할 수 있을 것으로 예상된다.

참고 문헌

[1] Kwan-Liu Ma, James S. Painter, Charles D. Hansen, Michael F. Krogh, A Data Distributed Parallel Algorithm for Ray Traced Volume Rendering, Parallel Rendering Symposium, pp15-22, 1993

[2] Frank Dachille, Kevin Kreeger, Baoquan Chen, Ingmar Bitter, and Arie Kaufman, High-Quality Volume Rendering Using Texture Mapping Hardware, SIGGRAPH / Eurographics Work shop on Graphics Hardware, pp 69-76, 1998

[3] Chandrajit Bajaj, Sangmin Park, Anthony Gene Thane, Parallel Multi PC Volume Rendering System, VolVis, 2002

[4] C. Rezk-Salama, K. Engel, M. Bauer, G. Greiner, T. Ertl, Interactive Volume Rendering on Standard PC Graphics Hardware Using Multi-Textures and Multi Stage Rasterization, SIGGRAPH /Eurographics Graphics Hardware Workshop, pp109-118, 2000

[5] ATI. Developer Relations. (Radeon Volvis). www.ati.com

[6] H.Ray and D.Silver, The RACE II engine for real time voume rendering, Proceedings of SIGGRAPH/ Eurographics Workshop, pp 129-136, 2000

[7] Myrinet overview. www.myri.com/myrinet/overview/index.html

[8] OpenGL-GLUT. www.opengl.org