3차원 텍스쳐 맵핑 및 텍스쳐 좌표 조작을 통한 대용량 볼륨 데이터의 효과적인 가시화 기법

이중연
한국과학기술정보연구원 슈퍼컴퓨팅센터
jylee@kisti.re.kr

Efficient Visualization Method for Large Volume Dataset using 3D Texture Mapping and Texture Coordinate Tweaking

Joong-Youn Lee
Supercomputing Center, KISTI

요 약

PC 그래픽스 하드웨어의 급격한 발전에 따라 과거 슈퍼컴퓨터 급에서나 가능하였던 대용량 데이터의 볼륨 렌더링을 일반 PC에서 수행하려는 시도가 계속되고 있다. 특히, PC 그래픽스 하드웨어의 꼭지점 및 픽셀 쉐이더는 기존의 고정된 그래픽스 파이프라인에서 벗어나 사용자가 렌더링 과정에 개입하여 프로그래밍을 할 수 있도록 하여 많은 각광을 받고 있다. 그러나 그래픽스 하드웨어의 텍스쳐 메모리의 크기보다 큰 볼륨 데이터의 가시화는 아직까지 충분히 빠르지 못하며 텍스쳐의 압축으로 인하여 영상 품질도 좋지 못하다. 본 논문에서는 이러한 그래픽스 하드웨어의 프로그래밍 기능 중 꼭지점 좌표 및 텍스쳐 좌표의 조작, 그리고 픽셀 쉐이더를 통한 퐁 쉐이딩 연산을 이용하여 그래픽스 하드웨어의 메모리 크기보다 큰 대용량 볼륨 데이터를 고품질로 가시화하였다.

1. 서 론

1.1. 연구 배경

여러 가지 과학적 가시화 방법 중 가장 중요한 위치를 차지하는 볼륨 가시화는 3차원이나 그 이상의 차원의 볼륨 데이터에서 의미있는 정보를 추출해 내어 직관적으로 표출하는 가시화 방법으로 의료영상, 기상학, 유체역학 등 다양한 분야에서 활용되어 지고 있다. 최근 대용량의 볼륨 데이터에 대한 고해상도 가시화에 대한 필요성이 증가하고 있으나 이를 원활하게 처리하기 위해서는 고가의 슈퍼컴퓨터를 이용해서 장시간 처리를 해야 하는 것이 현실이었다[2]. 한편, 최근 수년간 PC 그래픽스 하드웨어가 급속도로 발전하면서 전문적인 그래픽스 워크스테이션에서만 가능하였던 고품질, 고성능의 그래픽스 작업이 일반 PC에서도 가능하게 되었다. 특히 고정되어 있던 그래픽스 파이프라인을 사용자가 임의로 수정하여 사용할 수 있도록 한 꼭지점 및 픽셀 쉐이딩(vertex & pixel shading)기능은 가히 혁명적이라할 정도로 PC 그래픽스 하드웨어의 성능을 끌어올렸다.

1.2. 기존 연구

이와 같이 PC 그래픽스 하드웨어의 성능이 급격히 발전하면서 일반적으로 고성능의 슈퍼컴퓨터에서나 가능하던 빠른 볼륨 렌더링을 일반적인 PC에서 구현하려는 노력이 꾸준히 있어왔다. Akeley에 의해 3차원 텍스쳐 매핑을 이용한 볼륨 렌더링의 가능성[1]이 최초로 제기된 이후 이를 이용한 볼륨 렌더링에 대한 많은 연구가 계속되어왔고[4][10][11], PC 그래픽스 하드웨어의 성능이 발전하면서 PC 환경에서의 고성능의 볼륨 렌더링 연구 역시 꾸준히 진행되었다[5][9]. 이러한 노력에도 불구하고 텍스쳐 매핑을 이용한 볼륨 렌더링은 뚜렷한 한계를 보였는데, 볼륨 데이터의 크기가 텍스쳐 메모리의 크기를 넘지 못한다는 것이 그것이다. 이러한 단점을 극복하기 위하여 볼륨 데이터를 압축하고 계층구조로 표현하는 등 여러 가지 해결 방법이 제안되었지만 렌더링 속도가 늦어지고 영상 품질이 떨어지는 등 많은 문제점이 있어왔다[3][6][8].

2. 텍스쳐 매핑 기반의 볼륨 렌더링

2.1. 샘플링

볼륨 렌더링은 크게 3가지 과정으로 이루어지는데 샘플링(sampling), 쉐이딩(shading) 및 컴포지팅(compositing)이 그것이다. 볼륨 데이터에서 적절한 복셀(voxel)을 찾아오는 샘플링 과정은 매우 많은 시간을 필요로 하므로 이를 빠르게 처리하는 것은 주요한 연구 주제 중 하나였다. Lacroute 등은 쉐어-왑 분해(shear-warp factorization)를 이용하여 빠른 샘플링을 가능하게 하였다[7]. 그래픽스 하드웨어의 텍스쳐 매핑 기능은 선형보간(*linear interpolation*) 또는 삼선형보간(*tri-linear* interpolation)을 하드웨어적으로 매우 빠르게 처리할 수 있도록 해주는데, 텍스쳐 매핑을 이용한 볼륨 렌더링은 볼륨 데이터를 텍스쳐로 간주하고 하드웨어 텍스쳐 매핑을 통하여 샘플링을 매우 빠르게 처리하도록 한다. 이렇게 텍스쳐 매핑을 이용하여 샘플링을 처리하도록 하기 위해서는 텍스쳐가 입혀질 도형이 필요한데 이를 대리 도형(proxy geometry)라고 한다. 본 논문에서는 렌더링 속도를 향상시키기 위하여 대리 도형으로 평면(plane)을 이용하였으며 항상 영상 평면(image plane)에 평행하도록 하였다. 이를 위하여 그래픽스 하드웨어의 꼭지점 쉐이더(vertex shader)를 이용하였다. 꼭지점 쉐이더는 꼭지점의 이동을 가능하게 하는데, 볼륨 데이터의 회전 시에 모델뷰 행렬(modelview matrix)에 회전 행렬(rotation matrix)이 적용이 되더라도 대리 도형은 회전하지 않도록 꼭지점 쉐이더를 조작하였고 이러한 방법을 사용하여 대리 도형이 영상 평면과 평행하도록 하는데 추가적인 연산을 하지 않도록 하여 속도를 향상시켰다.

2.2. 쉐이딩

샘플링된 복셀들은 전이함수(transfer function)을 통해 색깔 및 투명도가 결정되고 여기에 쉐이딩을 통해 음영이 입혀지게 된다. 이러한 작업은 프로그래밍이 가능한 쉐이더(programmable shader)를 이용하여 수행된다. 전이함수는 종속 텍스쳐 (dependent texture) 기법을 이용하여 구현될 수 있

는데, 종속 텍스쳐란 한번 텍스쳐 매핑한 값을 텍스쳐 좌표로 활용하여 다시 텍스쳐 매핑을 하는 기법을 말한다. 본 논문에서는 2차원 텍스쳐를 이용한 2차원 전이함수를 사용하였는데, 복셀값과 복셀의 그라디언트(gradient) 크기를 인덱스로 하였다. 복셀의 그라디언트 크기가 크면 복셀값이 급격히 변화하므로 물질의 경계부분이라는 뜻이고 그라디언트 크기가 작으면 복셀값의 변화가 거의 없는 균일(homogeneous) 영역이라는 뜻이다. 일반적으로 물질의 경계면이 중요하게 인식되므로 이 부분의 투명도를 작게 하고 경계면이 아닌 균일 영역은 상대적으로 덜 중요하므로 투명도를 크게 하였다. 쉐이딩은 퐁의 조명 모델(Phong's illumination model)을 사용하였고, 그래픽스 하드웨어의 픽셀 쉐이딩(pixel shading) 기능을 이용하여 구현하였다. 퐁의 조명 모델을 계산하기 위해서는 모든 복셀마다 법선 벡터(normal vector)가 필요한데 이는 전방차이(forward difference) 방법을 이용하여 계산하였다. 이렇게 계산한 법선 벡터의 복셀값과 함께 3차원 텍스쳐에 저장되는데 텍셀(texel)의 RGB 영역에 법선벡터를, A 영역에 복셀값을 저장하도록 하였다.

2.3. 컴포지팅

컴포지팅은 그래픽스 하드웨어의 알파 블렌딩(alpha blending) 기능을 이용하여 수행한다. 이때 블렌딩 순서가 매우 중요한데 앞에서부터 뒤로 블렌딩하는 방법(front-to-back order)과 뒤에서부터 앞으로 블렌딩하는 방법(back-to-front order)이 있다. 본 논문에서는 뒤에서부터 앞으로 블렌딩하도록 하였으며 이에 따라 대리 도형(proxy geometry)를 그리는 순서 역시 시점에서 가장 먼 도형부터 가까운 순서대로 그리도록 하였다. 뒤에서부터 앞으로 블렌딩할 시에 사용되는 알파 블렌딩 연산은 아래의 식과 같은데, 이 식에서 C_i는 i번째 복셀에서의 색깔을 나타내고 A_i는 i번째 복셀에서의 투명도를 나타낸다.

$$C_i' = C_i + (1 - A_i)C_{i+1}'$$

2.4. 볼륨 데이터 회전

2.1.장에서 설명하였듯이, 대리 도형(proxy geometry)이 항상 영상 평면에 평행하도록 배치되므로 볼륨 데이터를 회전시키더라도 이것은 렌더링 영상에 전혀 반영되지 못하게 된다. 이러한 점을 해결하기 위하여 볼륨 데이터를 회전시킬 경우 모델뷰 행렬에 반영된 회전 행렬을 텍스쳐 좌표에 적용시켜서 텍스쳐가 회전하도록 하였다. 이러한 작업은 꼭지점 쉐이더에서 수행된다. 이때 텍스쳐 좌표가 [0, 1]의 영역에 있는데 그대로 회전할 경우 가운데를 중심으로 회전하지 못하므로 모든 텍스쳐 좌표에 0.5 만큼 빼주고 회전한 뒤 다시 0.5를 더하도록 하였다. 즉, 볼륨 데이터의 중심을 (0.5, 0.5) 만큼 이동하여 회전시킨 것이다.

볼륨 텍스쳐가 대리 평면(proxy plane)에 딱 맞게 매핑시킬 경우 텍스쳐를 회전하면 텍스쳐의 모서리 부분이 짤리게 된다. 이러한 점을 극복하기 위하여 텍스쳐 회전시에도 모든 텍스쳐가 대리 평면 안에 들어오도록 텍스쳐 좌표를 설정하였다.

그림1에서 보면 a_1의 텍스쳐 좌표는 0, a_2의 텍스쳐 좌표는 1이어야 한다.

$a_1a_2 : a_1a_3 = \frac{1}{\sqrt{2}} : \frac{1}{2} + \frac{1}{2\sqrt{2}} = 1 : a_3$ 이므로,

$a_3 = \frac{1}{2} + \frac{1}{\sqrt{2}}$ 가 된다.

마찬가지로, $a_0 = \frac{1}{2} - \frac{1}{\sqrt{2}}$ 이 된다.

따라서, 텍스쳐 좌표는 $\left[\frac{1}{2} + \frac{1}{\sqrt{2}}, \frac{1}{2} - \frac{1}{\sqrt{2}}\right]$ 이어야 한다.

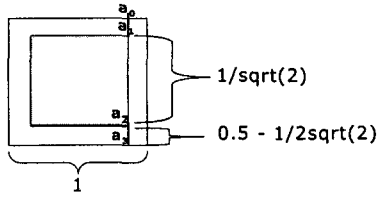

그림 1 볼륨 데이터 회전을 위한 텍스쳐 좌표

3. 텍스쳐 조작을 이용한 대용량 데이터 처리

텍스쳐 메모리 크기보다 큰 볼륨 데이터를 렌더링하는 방법은 여러 가지가 있지만 본 논문에서는 가장 일반적으로 사용하는 방법인 큰 볼륨 데이터를 작은 서브 볼륨 데이터로 나누어 처리하는 방법을 사용하였다. 이때 서브 볼륨의 크기는 구현하는 그래픽스 하드웨어가 처리할 수 있는 가장 큰 텍스쳐 크기로 한다. 예를 들어 512^3 크기의 볼륨 데이터를 최대 64MB의 텍스쳐까지 처리할 수 있는 그래픽스 하드웨어로 렌더링한다고 하면 256^3 크기의 서브 볼륨 8개로 나누어서 처리한다. 이렇게 나누어 놓은 서브 볼륨 데이터들은 같은 대리 평면에 각기 적절한 텍스쳐 좌표에 매핑된다. 이때 컴포지팅 시 정확한 순서로 알파 블렌딩 되어야 하므로 렌더링 시 각 프레임 마다 서브 볼륨과 시점간의 거리를 구하여 먼거리에 있는 서브 볼륨 순으로 렌더링하도록 하여 항상 뒤에서부터 앞으로 블렌딩되도록 하였다. 큰 볼륨 하나를 여러 개의 서브 볼륨으로 나누었기 때문에 텍스쳐 좌표를 이에 맞게 수정하여야 하는데, 이는 크게 두 가지 경우로 나눌 수 있다.

i) 텍스쳐가 좌표축의 시작 부분에 해당될 때

그림에서 보면 a_1의 텍스쳐 좌표는 0, a_2의 텍스쳐 좌표는 1 이므로 a_3의 텍스쳐 좌표는 2이다.

$a_1a_3 : a_1a_4 = \frac{1}{\sqrt{2}} : \frac{1}{2} + \frac{1}{2\sqrt{2}} = 2 : a_4$ 이므로

$a_4 = 1 + \sqrt{2}$ 이다.

마찬가지로, $a_0 = 1 - \sqrt{2}$ 이다.

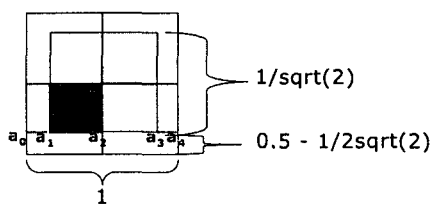

그림 2 대용량 데이터의 처리를 위한 텍스쳐 설정 1

ii) 텍스쳐가 좌표축의 끝 부분에 할당될 때

그림에서 보면 a_2의 텍스쳐 좌표는 0, a_3의 텍스쳐 좌표는 1 이므로 a_1의 텍스쳐 좌표는 -1 이다.

$a_2a_3 : a_2a_4 = \frac{1}{2\sqrt{2}} : \frac{1}{2} = 1 : a_4$ 이므로

$a_4 = \sqrt{2}$ 이다.

마찬가지로, $a_0 = -\sqrt{2}$ 이다.

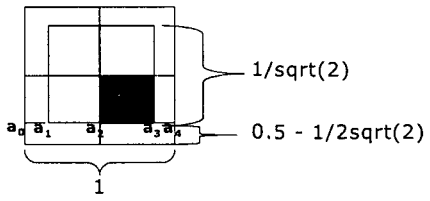

그림 3 대용량 데이터의 처리를 위한 텍스쳐 설정 2

4. 구 현 결 과

본 논문에서는 펜티엄4 2.4GHz, 1GB 메모리, 128MB 메모리의 GeForce FX 5900 그래픽스 카드를 장착한 시스템을 이용하여 구현하였다. 실험에 사용한 데이터는 다음과 같다.

데이터	크기	용량
bighead	256X256X225	14MB
engine	256X256X110	7MB
visible man	512X512X512	256MB

표 1. 실험에 사용한 볼륨 데이터의 종류와 크기

여기서 bighead와 engine은 복셀 크기가 8bit이고 visible man의 경우에는 12bit이다. 또한 bighead와 engine은 텍스쳐 메모리에 한번에 올리는 것이 가능하지만 visible man은 불가능하다. 이때문에 bighead와 engien은 그대로 텍스쳐 메모리에 올려서 렌더링하였고, visible man은 12bit를 8bit로 변환한 다음 서브 볼륨으로 나누어서 렌더링하였다. 결과는 그림 4 및 그림 5와 같다.

그림 4 렌더링 결과 영상 (왼쪽: bighead, 오른쪽: engine)

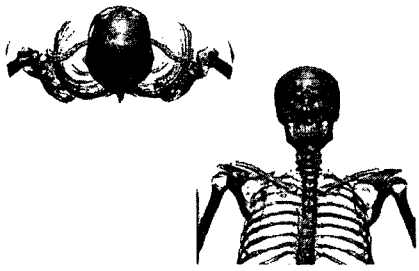

그림 5 렌더링 결과 영상 (visible man)

5. 결 론

본 논문에서는 PC 그래픽스 하드웨어를 이용하여 대용량 데이터를 이미지 손실없이 고품질로 렌더링하였다. 이는 데이터를 전혀 압축없이 렌더링하였기 때문에 가능하였다. 그러나 이로 인하여 텍스쳐 데이터를 매번 바꿔주어야 하고 또 대리 도형을 8배나 많이 사용함으로써 속도가 저하되었다. 특히 CPU와 GPU 사이의 대역폭이 병목으로 작용하여 렌더링 속도에 많은 영향을 미쳤다. 이를 해결하기 위해서는 대용량의 텍스쳐 메모리 또는 고성능의 대역폭을 제공할 수 있는 인터페이스가 요구되어 진다. 특히 텍스쳐 메모리의 용량은 한계가 있을 것이므로 결국 인터페이스의 속도가 매우 빨라져야 하는데 PCI Express 인터페이스는 이를 위한 좋은 대안이 될 것으로 생각된다.

6. 참고문헌

[1] Akeley : RealityEngine Graphics, SIGGRAPH 93 Conference Procedings, 109-116, 1993.

[2] Bajaj, Ihm, Koo, Park : Parallel Ray Casting of Visible Human on Distributed Memory Architextures, Data Visualization '99, 1999

[3] Boada, Navazo, Scopigno, : Multiresolution Volume Visualization with a Texture-Based Octree, The Visual Computer, 17(3), 185-197. Springer, 2001.

[4] Cabral, Cam, Foran : Accelerated Volume Rendering and Tomographic Reconstruction Using Texture Mapping Hardware, Symposium on Volume Visualization, 1994.

[5] Engel, Kraus, Ertl : High-Quality Pre-Integrated Volume Rendering using Hardware-Accelerated Pixel Shading, Eurographics/SIGGRAPH Workshop on Graphics Hardware, 2001.

[6] Guthe, Strasser : Real-Time Decompression and Visualization of Animated Volume Data, IEEE Visualization 2001, 2001.

[7] Lacroute, Levoy : Fast Volume Rendering Using a Shear-Warp Factorization of the Viewing Transformation. Computer Graphics, 28 (SIGGRAPH 94 Proceedings), 451-458, 1994.

[8] LaMar, Hamann, Joy : Multiresolution Techniques for Interactive Texture-Based Volume Visualization, IEEE Visualzation '99, pp 355-362, 1999.

[9] Rezk-Salama, Engel, Bauer, Greiner, Ertl : Interactive Volume Rendering on Standard PC Graphics Hardware using Multi-Textures and Multi-Stage Rasterization, EG/SIGGRAPH Workshop on Graphics Hardware, 2000.

[10] Van Gelder, Kim : Direct Volume Rendering with Shading via Three-Dimensional Textures. Symposium on Volume Visualization, pp 23-30, 1996.

[11] Westermann, Ertl : Efficiently using Graphics Hardware in Volume Rendering Applications, Computer Graphics (SIGGRAPH 98 Proceedings), pp 291-294, 1998.