꼭지점 및 픽셀 쉐이더를 이용한 3D 텍스쳐 기반의 빠른 볼륨 렌더링 기법

이중연
한국과학기술정보연구원 슈퍼컴퓨팅센터
e-mail:jylee@kisti.re.kr

3D Texture based Fast Volume Rendering using Vertex and Pixel Shaders

Joong-Youn Lee
Supercomputing Center, KISTI

요 약

PC 그래픽스 하드웨어의 급격한 발전에 따라 슈퍼컴퓨터 또는 여러 대의 컴퓨터를 이용한 병렬/분산 처리로나 가능하였던 실시간 볼륨 렌더링을 한대의 일반 PC에서 수행하려는 시도가 계속되고 있다. PC 그래픽스 하드웨어의 꼭지점 및 픽셀 쉐이더는 수치 계산에 최적화된 벡터 연산으로 빠른 볼륨 렌더링을 가능하게 하였을 뿐만 아니라 기존의 고정된 그래픽스 파이프라인에서 벗어나 사용자가 렌더링 과정에 개입하여 프로그래밍을 할 수 있도록 하였다. 본 논문에서는 이러한 그래픽스 하드웨어의 프로그래밍 기능 중 텍스쳐 좌표의 조작을 이용하여 다양한 종류의 볼륨 데이터를 빠르게 렌더링하고 픽셀 쉐이더의 여러 기능들을 이용하여 퐁 쉐이딩 연산, 이른 깊이 테스트, 팔진트리 텍스쳐등을 구현하여 고품질 영상을 실시간으로 얻고자 하였다.

1. 서론

여러 가지 과학적 가시화 기법 중 가장 중요한 위치를 차지하는 볼륨 가시화는 3차원이나 그 이상의 차원의 볼륨 데이터에서 의미 있는 정보를 추출해 내어 직관적으로 표출하는 가시화 방법으로 의료영상, 기상학, 유체역학 등 다양한 분야에서 활용되고 있다. 최근 대용량의 볼륨 데이터에 대한 고품질의 실시간 가시화에 대한 필요성이 급격히 증가하고 있으며 이를 원활하게 처리하기 위한 시도가 꾸준히 있어왔다. 한편, 최근 수년간 PC 그래픽스 하드웨어가 급속도로 발전하면서 전문적인 3D 그래픽스 워크스테이션에서나 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 혁명적이라고 할 정도로 PC 그래픽스 하드웨어의 성능을 한 단계 끌어올렸다. 본 논문에서는 이러한 PC 그래픽스 하드웨어의 꼭지점 및 픽셀 쉐이딩 기능을 이용하여 렌더링 영상의 손실없이 볼륨 데이터를 실시간으로 가시화하고자 하였다. 본 논문의 구성은 다음과 같다. 2장에서는 본 논문에서 구현한 GPU 기반의 볼륨 렌더링 기법에 대해 설명하고 3장에서는 실시간 렌더링을 위한 속도향상 기법을 소개한다. 4장에서는 실제 구현 결과와 렌더링 결과를 살펴보고 마지막으로 5장에서 결론을 맺는다.

2. GPU 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. 그래픽스 하드웨어의 텍스쳐 매핑 기능은 선형보간 또는 삼선형보간을 하드웨어적으로 매우 빠르게 처리할 수 있도록 해주는데, 텍스쳐 매핑을 이용한 볼륨 렌더링은 볼륨 데

이터를 텍스쳐로 간주하고 하드웨어 텍스쳐 매핑을 통하여 샘플링을 매우 빠르게 처리하도록 한다. 이렇게 텍스쳐 매핑을 이용하여 샘플링을 하기 위해서는 텍스쳐가 입혀질 도형이 필요한데 이를 대리 평면(proxy plane)이라고 한다. 보통 2차원 텍스쳐에 볼륨 데이터를 저장할 경우에는 대리 평면을 텍스쳐에 평행하도록 배치하고, 3차원 텍스쳐에 볼륨 데이터를 저장할 경우에는 영상평면(image plane)에 평행하도록 배치한다. 간편하게 대리 평면이 영상평면과 평행하도록 하기 위해서 그래픽스 하드웨어의 꼭지점 쉐이더(vertex shader)를 이용하였는데, 꼭지점 쉐이더에서는 각 꼭지점에 모델뷰 행렬(modelview matrix)과 투영 행렬(projection matrix)를 곱함으로써 각 꼭지점들에 대한 스크린 좌표를 계산하도록 한다. 여기서, 모델뷰 행렬에 회전 행렬을 곱함으로 해서 꼭지점이 실질적으로 회전하게 되는데, 대리 평면은 항상 영상평면에 평행으로 회전하면 안되기 때문에 꼭지점 쉐이더에서 꼭지점에 모델뷰 행렬을 무시하고 투영행렬만 곱해지도록 하였고, 대신 각 대리 평면의 꼭지점에 할당되는 텍스쳐 좌표에 모델뷰 행렬을 곱함으로써 볼륨 데이터가 회전하는 것처럼 보이도록 하였다.

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 있는데, 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 위해서 법선벡터를 복셀값과 함께 3차원 텍스쳐에 저장하였고 이 때문에 텍스쳐의 크기는 본래 볼륨 데이터의 3배 크기가 되었다.

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 블렌딩하도록하였으며 이에 따라 대리 도형을 그리는 순서 역시 시점에서 가장 먼 도형부터 가까운 순서대로 그리도록 하였다. 뒤에서 앞으로 블렌딩할 때에 사용되는 알파 블렌딩 연산은 아래의 식과 같은데, 이 식에서 C_i는 i번째 복셀에서의 색깔을나타내고 A_i는 i번째 복셀에서의 투명도를 나타낸다

$$C'_i = C_i + (1 - A_i)C'_{i+1}$$

수식 1. 컴포지팅 연산

3. 속도 향상 기법

3.1. 회전

볼륨 데이터를 회전시킬 때에 퐁 쉐이딩을 올바르게 적용시키기 위해서는 각 복셀의 법선 벡터들 역시 회전시켜야 한다. 이 경우 모든 프래그먼트(fragment)마다 법선 벡터를 회전시켜야 하므로 픽셀 쉐이더에서 매우 많은 연산을 처리하게 되어 렌더링 속도가 저하된다. 이러한 점을 해결하기 위해서 본 논문에서는 쉐이딩 계산 때에 복셀 및 법선 벡터는 회전시키지 않고 쉐이딩에 영향을 주는 다른 인자인 광원 및 시점을 회전 방향과 반대 방향으로 회전시켰다. 광원 및 시점은 프래그먼트에 독립적이므로 픽셀 쉐이더에서 계산할 필요없이 매 프레임마다 한번씩만 CPU로 계산해주면 되기 때문에 픽셀 쉐이더에 부하가 적게 걸리게 된다.

3.2. 빈공간 건너 뛰기

본 논문에서는 모든 복셀에 대해 픽셀 쉐이더에서 퐁 쉐이딩을 적용하였는데, 높은 수준의 렌더링 이미지를 얻기 위하여 특별히 반사 광원(specular light)을 포함한 완전한 퐁 쉐이딩을 사용하여 매우

복잡한 연산을 수행하도록 하였다. 이러한 이유로 전체 그래픽스 파이프라인 중 픽셀 쉐이더에서 병목 현상을 보여 전체 성능이 저하됨을 알 수 있었다. 이러한 단점을 극복하기 위해서 쉐이딩이 필요 없는 복셀에 대해서는 쉐이딩을 수행하지 않고 건너뛰도록 하여 전체 속도를 향상 시키고자 하였고, 이를 이중 패스 렌더링과 이른 깊이 테스트를 사용하여 구현하였다.

이른 깊이 테스트는 픽셀 쉐이더를 수행하기 전에 미리 깊이 테스트를 하여 테스트를 통과하는 프래그먼트(fragment)에 대해서만 픽셀 쉐이더를 적용하고 통과하지 못한 프래그먼트들은 모두 픽셀 쉐이더 작업을 생략하도록 하는 기법이다. 이러한 이른 깊이 테스트를 이용하여 빈 복셀에 대해서는 퐁 쉐이딩을 수행하지 않도록 하였는데, 이중 패스 렌더링 시 똑같은 대리 도형을 같은 위치에 두 번 그리도록 하고, 처음 그릴 때는 복잡한 퐁 쉐이딩을 적용시키지 않고 단순히 복셀 체크만 수행하여 그 복셀이 비었는지 여부만을 판단한다. 복셀이 비었을 경우에는 깊이 버퍼를 0으로 설정하여 두 번째 렌더링 시 깊이 테스트에서 건너뛰도록 하고 복셀이 비어 있지 않았을 경우에는 깊이 버퍼를 1로 설정하여 두 번째 렌더링 시 복잡한 퐁 쉐이딩을 수행하도록 하였다. 전체적인 알고리즘은 다음과 같다.

```
Firstpass:
    Draw proxy slice
    Check voxel value in pixel shader
    if (voxel value >= threshold) depth buffer = 1
    elseif (voxel value < threshold) depth buffer = 0
Secondpass:
    Draw proxy slice
    Apply Z test
    if (depth buffer == 0) skip pixel shader
    else do pixel shader
```

그림 1. 이른 깊이 테스트를 이용한 빈공간 건너뛰기 기법 알고리즘

4. 구현 결과

본 논문에서는 펜티엄4 2.4GHz, 1GB 메모리, 256MB 메모리의 GeForce 6800 GT 그래픽스 카드를 장착한 시스템을 이용하여 구현하였으며 쉐이더 모델 3.0을 사용하였다. 실험에 사용한 데이터는 표1과 같고 렌더링 영상은 그림2와 같다.

볼륨 데이터	볼륨 크기	텍스쳐 크기
Engine	256×256×110	256×256×128
Boston Teapot	256×256×178	256×256×256
CT Head	256×256×225	256×256×256
Aneurism	256×256×256	256×256×256

표 1. 각 볼륨 데이터의 크기

일반적인 그래픽스 하드웨어는 각 변의 길이가 2^n 인 텍스쳐만을 받아들이고 임의의 크기를 갖는 텍스쳐도 받아들이는 최신의 하드웨어라고 할지라도 최고의 속도를 내기 위해서는 텍스쳐의 각 변의 길이가 2^n이어야 하기 때문에 본 논문에서는 모든 볼륨 텍스쳐의 변의 길이를 2^n으로 하였다. 이 때문에 볼륨 데이터의 각 변의 길이가 2^n이 아닌 경우에는 그 변의 길이보다 큰 2^n 중 가장 작은 수의 길이를 갖는 크기로 텍스쳐를 생성하였다. 이 때문에 텍스쳐의 크기가 볼륨 데이터의 크기보다 약간 증가하였는데, 본 논문에서 사용한 Boston Teapot, CT Head, Aneurism 데이터의 경우, 실제 볼륨 데이터의 크기는 모두 다르지만 텍스쳐의 크기는 256×256×256로 모두 같게 되었다. 여기서 각 데이터의 특징을 보면, Engine과 CT Head의 경우 대부분의 영역이 데이터로 차 있지만, Boston Teapot과 Aneurism 데이터의 경우에는 복셀값이 0인 빈 공간이 많은 데이터이다.

표2는 각 볼륨 데이터에 대한 기본 렌더러와 속도향상 기법을 적용한 렌더러의 속도 차이를 보여준다. 256×256과 512×512 해상도를 가지는 윈도우에 256개의 대리 평면을 사용하여 렌더링하였다. 먼저 기본 렌더러의 경우, 데이터의 종류에 따른 렌더링 속도의 차이가 거의 없는 것을 알 수 있다. 렌더링 데이터의 특성에 거의 관계없이 일정한 속도를 보였으며 픽셀 개수가 정확히 4배 차이인 256×256 해상도와 512×512 해상도의 렌더링 속도 차도 약 3.5배로 4배에 근접한 차이를 보임을 알 수 있다. 256×256×128 크기를 갖는 Engine 데이터의 경우 두 배의 크기를 갖는 다른 데이터에 비해 속도가 그다지 빠르지 않았는데, 이는 그래픽스 메모리에서 수용 가능한 크기의 텍스쳐에 대해서는 그 크기에 큰 상관없이 비슷한 속도를 내기 때문으로 판단된다.

속도 향상 기법을 적용한 렌더러의 경우 비슷한 크기의 데이터라 할지라도 그 속도는 매우 다양함을 알 수 있다. 빈 공간이 많은 Aneurism과 Boston

그림 2. 각 볼륨 데이터의 렌더링 결과 (왼쪽에서부터 Engine, Boston Teapot, CT Head, Aneurism)

볼륨 데이터	Engine		Boston Teapot		CT Head		Aneurism	
윈도우 크기	256×256	512×512	256×256	512×512	256×256	512×512	256×256	512×512
기본	15.14	4.17	14.24	4.15	13.98	4.11	14.32	4.15
속도향상	46.48	16.08	51.22	19.61	28.68	9.77	62.95	23.38
렌더러 속도 차이	3.07	3.85	3.59	4.72	2.05	2.37	4.39	5.63

표 2. 각 볼륨 데이터에 대한 윈도우 크기 및 렌더러 종류에 따른 렌더링 속도 (frames/sec)

Teapot의 경우 512×512 해상도의 경우 23.38과 19.61의 초당 프레임 수를 보여 각각 5.63배와 4.72배의 성능 향상을 가져왔다. 반면, CT Head와 Engine 데이터의 경우 빈공간이 상대적으로 많지 않아 각각 2.37배와 3.85배의 속도 향상을 보였다.

윈도우 크기에 따른 속도향상률을 보면, 윈도우 크기가 커질수록 속도향상률이 높아지는 것을 알 수 있는데, 이는 건너 뛸 수 있는 픽셀 수가 4배로 늘어났기 때문인 것으로 판단된다. 또한 속도향상 기법을 적용하였을 경우 윈도우 크기 차이에 따른 렌더링 속도차가 2.6~2.9 배로 기본 렌더러의 3.5배에 비해 낮아 윈도우 크기가 커짐에 따른 렌더링 속도 저하가 상대적으로 낮음을 알 수 있는데, 이 또한 윈도우 크기가 커질수록 건너뛰는 픽셀 수가 증가하였기 때문으로 보인다.

5. 결론

본 논문에서는 PC 그래픽스 하드웨어를 이용하여 볼륨 데이터를 이미지 손실없이 고품질을 유지하면서 빠른 속도로 렌더링하였다. 이는 픽셀 쉐이더의 프로그래밍 기능을 이용한 완전한 퐁 쉐이딩으로 가능하였다. 그러나 이로 인하여 픽셀 쉐이더에 많은 연산이 필요하게 되어 속도 저하가 있었다. 이러한 점을 극복하기 위하여 쉐이딩이 필요없는 복셀에 대해서는 퐁 쉐이딩을 적용시키지 않도록 하였는데, 이중 패스 렌더링 및 이른 깊이 테스트를 이용하여 이를 가능하게 하였고, 이러한 알고리즘을 이용하여 일반 PC 컴퓨터에서 고품질의 이미지를 실시간으로 생성할 수 있었다. 향후에는 이른 광선 단절법을 GPU로 구현하여 보다 높은 성능 향상을 꾀하고 텍스쳐 압축을 통하여 보다 큰 크기의 볼륨 데이터를 렌더링하고자 한다.

참고문헌

[1] Akeley, "RealityEngine Graphics", Proceeding on SIGGRAPH 93 Conference, 109-116, 1993.

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

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

[4] Meissner, Guethe, "Interactive Lighting Models and Pre-Integration for Volume Rendering on PC Graphics Accelerators", Proceedings of Graphics Interface 2002, pp. 209-218, 2002.

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