# 3D 텍스쳐 매핑 하드웨어 하에서 법선 벡터 블렌딩을 이용한 가속화된 볼륨 렌더링

## (Accelerated Volume Rendering based on 3D Texture Mapping Hardware using Normal Blending)

윤 성 의 † 신 영 길 ††
(Sung-Eui Yoon) (Yeong-Gil Shin)

**요 약** 본 논문에서는 3D 텍스쳐 매핑 하드웨어(texture mapping hardware)하에서 OpenGL를 이용하여 빠른 추출(classification) 및 음영처리(shading)를 가능하게 하는 직접 볼륨 렌더링(direct volume rendering) 방법을 제안한다. 추출과정을 위해 lookup table을 통해서 볼륨 데이타의 밀도값(density)으로부터 불투명도(opacity)값을 얻어내고, 법선 벡터 블렌딩(normal blending)방법을 제안하여 볼륨 크기에 상관없이 최종 이미지에서만 음영 처리 연산을 수행한다. 본 논문에서 제시된 볼륨 렌더링의 전과정이 그래픽스 하드웨어(graphics hardware)에서 이뤄지며, 음영처리 연산의 복잡도 감소로 인하여 상호 대화적인 볼륨 렌더링이 가능하다.

***Abstract*** We present a new technique of volume rendering based on 3D texture mapping hardware, enabling fast classification and shading using OpenGL and extension. For classification, opacity is acquired from lookup table using original density value and shading is accomplished for only final viewport image regardless of volume size using a novel composition method, normal blending. Since all the above mentioned tasks are performed within graphics hardware and the time complexity of the shading operation is reduced, interactive direct volume rendering is possible.

## 1. 서 론

볼륨 렌더링은 CT, MRI, 3D 스캐너로부터 생성된 볼륨 데이타를 가시화 하는 기법을 말한다[1]. 그러나 볼륨 데이타의 방대함으로 인해 실시간 렌더링에는 많은 어려움이 따른다. 이러한 문제점을 효과적으로 극복하기 위한 한가지 방법으로 텍스쳐 매핑 하드웨어(texture mapping hardware)를 이용한 볼륨 렌더링이 연구 되어왔다[2-7]. 그러나 볼륨 데이타를 가시화 하는 중요한 요소인 추출(classification)과 음영 처리(shading)를 텍스쳐 매핑 하드웨어 하에서 동시에 구현하는 것은 최근까지 가능하지 않았다[2-6]. 이러한 문제를 해결하기 위한 시도로 Meissner[7]는 픽셀 텍스쳐(Pixel Texture[14])를 이용하여 이 두 가지를 그래픽스 하드웨어(graphics hardware)에서 수행하기 위한 방법을 제시하였다. 그러나 이 방법 또한 추출 및 음영 처리를 하기 위해 매 텍스쳐 샘플링 슬라이스(texture sampling slice)마다 픽셀 텍스쳐를 사용해야 함으로, 주 메모리(main memory)와 프레임 버퍼간에 과도한 데이타 이동이 존재하여 상호 대화적인 렌더링이 가능하지 않았다.

본 논문에서는 추출 및 음영처리를 3D 텍스쳐 매핑 하드웨어 하에서 가능하게 하는 법선 벡터 블렌딩(normal blending)이라는 새로운 혼합(composition) 방법을 이용한 볼륨 렌더링 방법을 제시한다. 이로 인해 추출 및 음영처리의 전 과정을 주 메모리로의 데이타 이동 없이 프레임 버퍼 내에서 가능하게 하였고, 또한 음영처리 연산을 볼륨 크기에 상관없이 최종 이미지에 대해서만 수행하도록 하여, 상호 대화적인 볼륨 렌더링이 가능하였다.

본 논문의 구성은 다음과 같다. 2장에서는 3D 텍스쳐 매핑 하드웨어를 이용하여 볼륨 렌더링 방법을 가속화하기 위한 기존의 연구에 대해서 간단히 알아보고, 3장

† 비 회 원 : 서울대학교 컴퓨터공학부
sungeui@cglab.snu.ac.kr
†† 종신회원 : 서울대학교 컴퓨터공학부 교수
yshin@cglab.snu.ac.kr
논문접수 : 2000년 7월 27일
심사완료 : 2001년 3월 14일

에서는 3D 텍스쳐 매핑 하드웨어를 이용하여 볼륨 데이타를 가시화 하는 기본 요소인 추출 및 음영처리 방법에 대해서 알아본다. 4장에서는 본 논문에서 제안하는 빠른 추출 및 법선 벡터 블렌딩을 통한 음영처리 방법에 대해서 살펴본다. 본 연구의 실험 결과를 5장에서 제시한 후 6장에서 결론을 맺는다.

## 2. 관련 연구

볼륨 렌더링 방법은 직접 볼륨 렌더링(directed volume rendering)과 간접 볼륨 렌더링(indirected volume rendering)으로 구분된다. 여기서 간접 볼륨 렌더링 방법은 마칭 큐브(Marching Cube[8])와 같은 방법을 이용하여 삼각형 정보를 얻어낸 후 이로부터 가시화를 하는 것이다. 이에 반해 직접 볼륨 렌더링은 볼륨 데이타를 직접 이용하여 가시화를 하는 방법으로, 이와 관련된 연구로는 [2-4, 9-11]등이 있다. 그러나 볼륨 데이타의 방대한 계산량으로 인해 소프트웨어적인 가시화 방법으로는 상호 대화적인 렌더링을 제공하기 힘들다.

현재까지 제시된 방법 중 쉬어-웝 분해(shear-warp factorization) 알고리즘[12]은 볼륨 데이타간의 연관성(coherency)을 이용하여 투명한 복셀(voxel)과 보이지 않는 픽셀을 건너뜀으로써 소프트웨어만으로 상호 대화적인 렌더링이 가능하도록 하였다. 그러나 Akeley[13]가 처음으로 3D 텍스쳐 매핑 하드웨어(특히 SGI RealityEngine[13])를 이용하여 볼륨 렌더링을 가속화 시킬 수 있다는 것을 언급한 이 후로 소프트웨어적인 렌더링 방법의 대안으로 3D 텍스쳐 매핑 하드웨어를 이용하여 볼륨 데이타를 가시화 하는 것이 효과적인 방법으로 인식이 되었다.

Cabral[2]은 $256^3$ 볼륨 데이타를 하나의 150MHz CPU를 가진 four Raster Manager SGI RealityEngine Onyx 상에서 텍스쳐 매핑 하드웨어를 이용하여 대화형 볼륨 렌더링이 가능하도록 하였다. 또한 Cullip 등[3]도 이와 비슷한 결과를 냈다. 하지만 이 방법들의 가장 큰 문제점은 볼륨 데이타에 대해서 음영처리 기능을 지원하지 못했다는 것이다. 이 점을 극복하기 위해 Van Gelder[4]는 볼륨 데이타의 추출과 음영처리를 위한 3-4 parameter lookup table을 제시하였는데, 불행히도 이 lookup table 대한 직접적인 하드웨어 지원이 가능하지 않았다. 그러므로 시각(viewing)이나 추출 파라미터(classification parameter)가 변경될 때마다 전체 3D 텍스쳐가 다시 만들어져야 한다는 단점이 있었다.

Westermann[6]은 텍스쳐 메모리에 밀도값(density value)과 그에 해당하는 법선 벡터를 저장하고, 그로부터 OpenGL과 extension의 여러 가지 기능을 이용하여 등위면(iso-surface)에서 추출과 음영처리를 가능하게 했다. 그러나 이 논문에서 제시된 방법은 볼륨 렌더링에는 적용되지 못했다. 이후 Meissner[7]가 픽셀 텍스쳐를 이용하여 그래픽스 시스템에서 볼륨 렌더링을 위한 추출 및 음영처리가 가능하도록 하였다. 그러나 픽셀 텍스쳐가 현재 glDrawPixels()에서만 구현 되어있고, 텍스쳐 샘플링 슬라이스마다 이 함수를 사용해야 함으로 프레임 버퍼와 주 메모리간의 과도한 데이타 이동이 발생하여 상호 대화적인 렌더링은 가능하지 않았다.

## 3. 3D 텍스쳐 매핑 하드웨어를 이용한 볼륨 렌더링

본 장에서는 표준 OpenGL과 extension을 이용하여 기존 연구에서 제시된 방법에 근거하여 볼륨 렌더링의 각 부분이 어떻게 이루어지는지, 또한 그래픽스 시스템에서 각 부분들이 어떻게 연결되는가를 기술한다.

### 3.1 슬라이싱(slicing) 및 텍스쳐 매핑

3D 텍스쳐에 볼륨 데이타를 저장하는 방식에는 $RGB\alpha$ 채널 각각에 미리 추출되어진 각 복셀의 색상($RGB$)과 불투명도($\alpha$)를 저장하거나[2, 3, 4, 5], CT나 MRI로부터 나온 밀도값(density)과 이로부터 구해진 법선 벡터를 저장하는 방식[6, 7]등이 있다.

3D 텍스쳐 매핑의 기본 아이디어는 볼륨 데이타를 $[0-1]^3$에서 정규화된 공간의 3D 텍스쳐로 보고, 이 텍스쳐 도메인 안에서 슬라이싱을 한 후 텍스쳐 매핑을 수행하는 것이다. 전체 과정을 자세히 기술하면, 먼저 텍스쳐 도메인 상에서 이미지 평면(image plane)에 평행한 평면을 시점 위치에서 멀리 있는 것부터 구한 후(슬라이싱), 이것들을 그래픽스 시스템으로 보낸다. 3D 텍스쳐 매핑 하드웨어는 삼각형의 각 점의 좌표를 3D 텍스쳐 좌표로 해석하고, 볼륨 데이타 안에서 삼선형 보간 하여 텍스쳐 샘플을 재구성한다. 마지막으로 각각의 텍스쳐 샘플링 슬라이스의 픽셀 값들을 프레임 버퍼에서 블렌딩 한다. 그림 1은 위에서 기술한 과정을 그림으로 나타내고 있다.

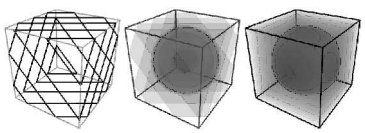


그림 1 볼륨 렌더링에 기반한 3D 텍스쳐 매핑에 적용되는 슬라이싱 방법[6]

### 3.2 추출

텍스쳐 데이타에 밀도값이 저장되어 있으면 3D 텍스쳐 매핑 단계 후에 밀도값을 lookup table에 통과시켜 해당 색상이나 불투명도 등을 구할 수 있다. 이 과정은 GL_TEXTURE_TABLE_SGI를 이용해 이루어진다.

### 3.3 등위면 상에서의 음영 처리

볼륨 데이타에서 구해진 등위면의 음영처리는 칼라 매트릭스(color matrix)(OpenGL 1.2의 Imaging subset의 일 부분)를 통해서 행해진다. 이것은 Westermann 등[6]에 의해 처음으로 제시되었다. 이 방법은 텍스쳐 데이타의 *RGB* 채널에 볼륨 데이타의 법선 벡터를 저장하고, $\alpha$ 채널에는 볼륨 데이타의 밀도값을 저장한다.

아래의 칼라 매트릭스는 난반사를 고려한 음영처리(diffuse shading)를 식으로 표현한 것이다. 여기서 $M_{rotation}$의 오른쪽 매트릭스는 보간 되어진 법선 벡터의 각 x, y, z요소( *RGB* 채널에 저장)의 범위를 [0,1]에서 [−1,1]로 변환하기 위해 필요하다. $M_{rotation}$의 왼쪽 행렬은 빛의 방향 벡터(light vector)와 법선 벡터의 내적을 구하기 위해 필요하다. 또한 여기서 $M_{rotation}$는 현재의 시각 행렬을 나타낸다.

$$M_{color} = \begin{bmatrix} L_x & L_y & L_z & 0 \\ L_x & L_y & L_z & 0 \\ L_x & L_y & L_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} M_{rotation} \begin{bmatrix} 2 & 0 & 0 & -1 \\ 0 & 2 & 0 & -1 \\ 0 & 0 & 2 & -1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

이렇게 칼라 매트릭스와 보간 되어진 법선 벡터를 곱한 후 post-color matrix scale and bias 연산( $k_d$를 곱하고, $k_aI_a$를 더함)을 이용하면 $I=k_d<L,N>+k_aI_a=k_dI_d+k_aI_a$인 난반사 음영 처리가 가능하다.

위에 기술한 모든 과정은 glCopyPixels()나 glDrawPixels()에 의해 수행 될 수 있는데, 두 개의 함수 중 glCopyPixels()는 프레임 버퍼에서 프레임 버퍼로 데이타가 이동하므로 더 효율적이다.

위에서 기술한 칼라 매트릭스를 통해서 음영처리를 정확히 실행하기 위해서는 $\alpha$ 채널이 1로 설정되어야 한다. 그러므로 반투명한 물체와의 블렌딩이 불가능하게 되어 등위면에서만 음영처리가 가능하다는 단점이 있다.

### 3.4 추출 기법과 음영처리 연산의 결합

앞에서 설명한 추출, 음영처리 기법이 3D 텍스쳐 매핑 하드웨어에서 일반적으로 행해지는 방법이다[6, 7]. 그러나 추출을 실행한 후에는 법선 벡터가 저장된 *RGB* 채널을 해당 색상으로 덮어쓰게 되면 더 이상의 음영처리가 불가능해진다. 또한 위에서 설명한 음영처리 방법은 반투명한 물체와의 블렌딩을 불가능하게 만든다.

3D 텍스쳐 매핑 하드웨어를 이용한 볼륨 렌더링에서의 추출과 음영처리는 Meissner[7]에 의해 그래픽스 하드웨어에서 가능해졌다. 이 방법도 Westermann[6]이 사용한 방법대로, 텍스쳐 데이타에 법선 벡터와 밀도값을 *RGBα* 채널에 저장한다. 여기서 Meissner는 $\alpha$ 채널이 1로 설정되는 것을 막기 위해 Pixel-Transfer-Operation을 사용하여 텍스쳐에 저장된 법선 벡터의 범위를 [−1,1]사이로 미리 만들어 준다. 그리고 난반사 음영처리 요소(diffuse shading term)를 구한 후에도 반투명이 가능한 추출 및 음영처리를 지원하기 위해서 아래와 같은 칼라 매트릭스를 사용했다.

$$M_{color} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ L_x & L_y & L_z & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} M_{rotation}$$

이후 보간된 샘플(법선 벡터와 밀도값으로 구성)을 칼라 매트릭스와 곱하면 다음과 같은 값을 얻을 수 있다.

$$M_{color} \begin{bmatrix} R \\ G \\ B \\ \alpha \end{bmatrix} = \begin{bmatrix} \alpha \\ <L,N> \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} D \\ I_d \\ 0 \\ 0 \end{bmatrix}$$

이후 남아 있는 밀도값($D$)과 난반사 음영처리 요소($I_d$)를 이용하여 추출과 난반사 음영처리를 실행할 수 있는데, 이 과정은 픽셀 텍스쳐(Pixel Texture[14])를 이용하여 가능하게 했다. 픽셀 텍스쳐는 버퍼에 저장된 *RGBα*의 값을 픽셀 텍스쳐의 색인으로 하는 lookup table 이다. 여기서는 밀도값($D$)과 난반사 음영처리 요소($I_d$)를 2차원 픽셀 텍스쳐 색인으로 지정한 후, 픽셀 텍스쳐의 각 항목을 아래와 같이 지정한다.

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = k_d(D) * I_d * \begin{bmatrix} R(D) \\ G(D) \\ B(D) \end{bmatrix} + k_a(D) * I_a * \begin{bmatrix} R(D) \\ G(D) \\ B(D) \end{bmatrix}$$

$$\alpha = \alpha(D)$$

여기서 glReadPixels()를 실행하여 프레임 버퍼의 내용을 주 메모리로 복사한 후 glDrawPixels()를 실행하여 이것을 lookup table인 픽셀 텍스쳐를 거쳐 추출과 음영처리를 행한 후 최종 렌더링 이미지만을 저장하는 다른 프레임 버퍼에 블렌딩 하면 되는 것이다.

그러나 현재 픽셀 텍스쳐는 SGI IMPACT 와 SGI Octane MXE에서만 가능하며, 더욱이 glCopyPixels()에서는 현재 구현되어 있지 않고, glDrawPixels()에서만 구현되어있으므로, 매 텍스쳐 샘플링 슬라이스를 구

한 후 추출, 음영처리하기 위해서 glReadPixels()와 glDrawPixels()를 행해야 함으로 전체 렌더링 시간을 높이는 가장 큰 요인이다. Meissner는 glReadPixels()와 glDrawPixels()에 소비되는 시간이 전체 렌더링 시간의 약 70% 이상 인 것으로 밝히고 있다[7].

## 4. 빠른 추출 및 음영처리 기법

본 논문에서는 Meissner[7]가 제시한 방법, 즉 매 샘플링 슬라이스마다 픽셀 텍스쳐를 이용하여 추출 및 음영처리를 동시에 실행한 후 프레임 버퍼에 블렌딩 하는 것과는 다르게 추출을 먼저 시행한 후 음영처리를 실행했다. 또한 매 샘플링 슬라이스마다 음영처리를 하는 것이 아니라 최종 이미지에서 단 한번만의 음영처리 연산을 행한다.

전체 단계를 간단히 살펴보면, 첫 번째 단계로 이미지 평면에 평행한 텍스쳐 평면을 시각 위치에서 멀리 떨어진 순으로 얻어낸다. 이렇게 얻어진 텍스쳐 샘플링 슬라이스는 삼선형 보간된 법선 벡터와 밀도값으로 구성된다. 다음 단계로 lookup table을 이용하여 추출을 행한다. 이 과정을 통해 밀도값으로부터 불투명도 값을 얻어낼 수 있다. 이후에 저장된 법선 벡터를 알파 블렌딩(alpha blending)을 시행한다. 최종 단계로 모든 텍스쳐 샘플링 슬라이스의 합성(composition)이 끝난 후 최종적으로 프레임 버퍼에 남아있는 합성된 법선 벡터(blended normal vector)를 가지고 단 한번만 음영처리 연산을 수행한다.

### 4.1 추출

본 논문에서는 밀도값에 따라서 *RGBα* 채널 모두를 변경하지 않고, $\alpha$(불투명도)값만을 얻어내어 반투명이 가능한 추출을 수행한다. 이때 법선 벡터(*RGB* 채널에 저장)가 그대로 유지되므로 이후의 음영처리가 가능하다.

### 4.2 법선 벡터 블렌딩(Normal Blending)

텍스쳐 매핑 하드웨어를 이용한 합성은 이미지 평면에 평행한 텍스쳐 샘플링 슬라이스를 구한 후 이것을 프레임 버퍼에 블렌딩 함으로서 이루어진다. 여기서 음영처리는 $I = k_d I_d + k_a I_a = k_d \langle L, N \rangle + k_a I_a$ $(k_d + k_a = 1)$을 행함으로써 구해지는데, 여기서 $I_d$는 난반사 음영처리 요소이고, $I_a$는 배경광 음영처리 요소이다. 또한 OpenGL에서 후 전진 방향 (back to front order) 블렌딩을 위한 식은 glBlendFunc (GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA) 이다. 이것을 식으로 나타내면 $I_{new_f} = I_s \alpha_s + I_f (1 - \alpha_s)$이다. 여기서 아래첨자 $s$는 새로이 들어오는 텍스쳐 샘플링 슬라이스이고, 아래첨자 $f$는 프레임 버퍼를 의미한다. 또한 아래첨자 *new_f*는 새로 갱신된 프레임 버퍼를 의미한다.

n번째 텍스쳐 샘플링 슬라이스를 프레임 버퍼에 후전진 방향으로(back to front order)로 블렌딩 하여 밝기(intensity)를 구하는 식을 나타내면 아래와 같다. (여기서 위 첨자 $n$은 n번째 텍스쳐 샘플링 슬라이스를 나타낸다.)

$$
\begin{aligned}
I_f^n &= I_s^n \alpha_s^n + I_f^{n-1}(1 - \alpha_s^n) \\
&= (k_d (L \bullet N_s^n) + k_a I_a)\alpha_s^n + \\
&\quad (k_d (L \bullet N_f^{n-1}) + k_a I_a)(1 - \alpha_s^n) \\
&= k_d L \bullet (N_s^n \alpha_s^n + N_f^{n-1}(1 - \alpha_s^n)) + \\
&\quad k_a I_a \alpha_s^n + k_a I_a (1 - \alpha_s^n) \\
&= k_d L \bullet (N_s^n \alpha_s^n + N_f^{n-1}(1 - \alpha_s^n)) + \\
&\quad k_a I_a \\
&= k_d L \bullet N_f^n + k_a I_a
\end{aligned}
$$

위 식에서 보는 것처럼 매 텍스쳐 샘플링 슬라이스의 법선 벡터로부터 난반사 음영처리 연산을 적용하여 구한 밝기를 블렌딩 하는 것과, 법선 벡터를 블렌딩 한 후 마지막으로 프레임 버퍼에 남은 합성된 법선 벡터로 한번만 음영처리 한 결과가 같음을 알 수 있다. 즉 매 텍스쳐 샘플링 슬라이스마다 음영처리를 할 필요 없이 최종적으로 합성된 법선 벡터로부터 한번의 음영처리만을 수행함으로 렌더링 시간을 단축시킬 수 있다.

### 4.3 음영 처리

본 논문에서 사용한 음영처리 방법은 Westermann[6]이 제시한 것과 비슷하게 진행된다. 그러나 $\alpha$ 채널이 1로 설정되는 것을 막기 위해 Pixel-Transfer-Operation (2를 곱하고, -1을 빼줌)을 사용하여 프레임 버퍼에 있는 합성된 법선 벡터의 각 요소 범위를 [0,1]사이에서 [−1,1]사이로 만들어준다. 이후 아래의 칼라 매트릭스를 프레임 버퍼에 있는 최종의 합성된 법선 벡터에 곱하면 난반사 음영처리 요소($I_d$)를 구할 수 있다.

$$
M_{color} = \begin{bmatrix} L_x & L_y & L_z & 0 \\ L_x & L_y & L_z & 0 \\ L_x & L_y & L_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} M_{rotation}
$$

이후 post-color matrix scale and bias 연산($k_d$를 곱하고, $k_a I_a$를 더함)을 이용하면 난반사 음영처리가 가능하다. 위에 기술한 모든 과정은 최종 프레임 버퍼의 내용을 자기 자신으로 다시 복사함으로 인해 이루어진다. 즉 이 과정은 glCopyPixels()(또는 glDrawPixels())

에 의해 이루어진다.

## 5. 구현 결과

본 연구는 2Mbyte 텍스쳐 메모리를 가진 SGI Indigo2 Impact 10000상에서 실험이 행해졌다. 여기서 텍스쳐 매핑 하드웨어의 메모리 공간보다 더 큰 볼륨 데이타를 렌더링하기 위해서는 주어진 볼륨 데이타에 대해 브릭킹(bricking)을 수행하여야 한다. 본 논문에서는 한 개의 브릭(brick) 크기를 64×64×64(실제 brick의 크기는 각 복셀당 *RGBα* 채널이 필요함으로 1Mbyte가 된다.)로 정한 후 이보다 큰 볼륨 데이타는 여러 개의 브릭의 집합으로 구성하였다. 본 연구에서 사용한 데이타는 표 1과 같고, 사용된 모든 데이타는 복셀당 8비트로 구성된다.

표 2는 각 실험 데이타에 대한 렌더링 시간을 나타낸 것이다. 단계 I는 3D 텍스쳐로부터 샘플링 슬라이스를 구하고, 추출 과정을 거친 후 프레임 버퍼에 블렌딩 하는 과정에 소비된 시간이고, 단계 II는 최종 프레임 버퍼에 남아 있는 합성된 법선 벡터로부터 음영처리 연산을 행하는 데에 걸린 시간이다. 또한 "렌더링 시간[7]"은 같은 조건하에서 Meissner[7]가 제시한 방법을 이용했을 때의 렌더링 시간이다.

표 1 본 연구에서 사용된 실험 데이타(텍스쳐 매핑 하드웨어를 이용하기 위해서는 데이타의 크기가 2의 멱수이여야 한다)

| 실험 데이타 | 볼륨 크기 | 브릭 개수 |
|---|---|---|
| small sphere | 64×64×64 | 1 |
| small brain | 128×128×128 | 8 |
| engine | 256×256×128 | 32 |
| big head | 256×256×256 | 64 |

표 2에서 나타난 바와 같이 단계 I는 볼륨 데이타 크기에 선형으로 비례하여 증가하는 것을 볼 수 있다. 이와는 반대로 단계 II는 실험 데이타 크기에 상관없이 일정한 속도를 가지는 것을 볼 수 있는데, 이것은 법선 벡터 블렌딩을 통해서 음영처리 연산을 단 한번만 행하기 때문이다. 그리고 단계 II가 상대적으로 차지하는 시간이 단계 I에 비해 작기 때문에 전체 렌더링 시간은 볼륨 데이타 크기에 선형으로 비례하여 증가됨을 알 수 있다. 또한 전체 렌더링 시간은 뷰포트(viewport)의 크기에 따라 선형으로 비례하여 증가하는 것을 볼 수 있다. 하지만 데이타가 큰 경우(engine, big head 데이타)에는 넓은 뷰포트에 대하여 좀더 빠르게 렌더링 되고 있는데, 예를 들어 bighead 데이타의 경우 뷰포트의 크기가 4배로 늘어났지만 전체 렌더링 시간은 2.5배만 늘어났다. 이것은 렌더링을 할 때 좀더 넓은 영역으로 텍스쳐 매핑하므로 텍스쳐 데이타에 대하여 캐시 적중률(cache hit ratio)이 증가되었기 때문이다.

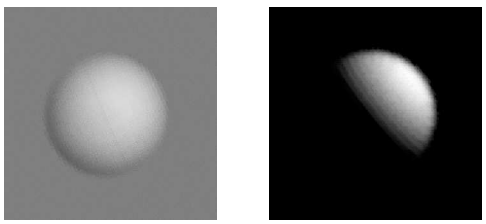
표 3은 실험적으로 얻어진 glCopyPixels()와 glReadPixels() + glDrawPixels()의 수행 시간을 비교한 것이다. 표에서 나타난 바와 같이 glCopyPixels()의 수행 시간이 glReadPixels () + glDrawPixels()의 수행 시간

표 3 glCopyPixels()와 glReadPixels() + glDrawPixels()의 수행 시간. (*msec*)

| 뷰포트의 크기 | glCopyPixels() | glReadPixels()+glDrawPixels() |
|---|---|---|
| 256×256 | 10.4 | 11.2 |
| 512×512 | 40 | 44 |

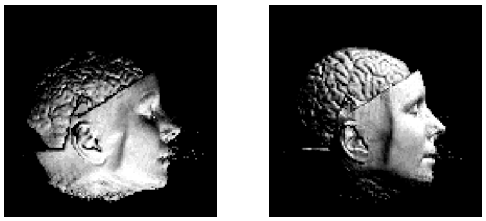
표 2 각 실험 데이터에 대한 렌더링 시간(*msec*)

| 실험 데이타 | 뷰포트의 크기 | 텍스쳐 샘플링 슬라이스의 개수 | 단계 I | 단계 II | 전체 렌더링 시간 | "렌더링 시간[7]" | 향상율 (배) |
|---|---|---|---|---|---|---|---|
| small sphere | 256*256 | 64 | 2 | 14 | 16 | 722 | 45.1 |
| small sphere | 512*512 | 64 | 3 | 43 | 46 | 2809 | 61.1 |
| small brain | 256*256 | 128 | 75 | 14 | 89 | 1312 | 14.7 |
| small brain | 512*512 | 128 | 148 | 42 | 190 | 4383 | 23.1 |
| engine | 256*256 | 128 | 311 | 13 | 324 | 1991 | 6.1 |
| engine | 512*512 | 128 | 541 | 44 | 585 | 4999 | 8.5 |
| big head | 256*256 | 256 | 634 | 15 | 649 | 4021 | 6.2 |
| big head | 512*512 | 256 | 1125 | 43 | 1168 | 10061 | 8.6 |



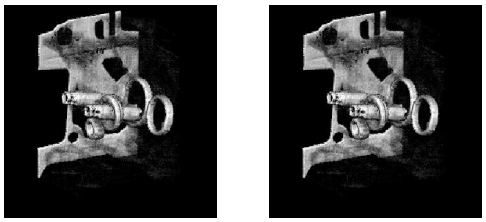
(a) (b)

(a)와 이로부터 렌더링 한 결과 (b)
그림 2 small sphere 데이타에 대한 합성된 법선 벡터



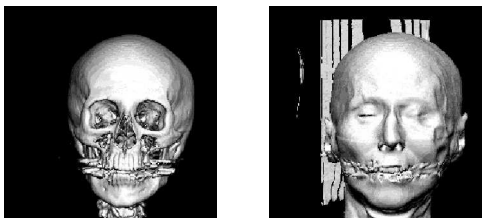
(a) (b)

그림 3 small brain 데이타를 서로 다른 광원 방향으로 렌더링한 이미지



(a) (b)

(a) 본 연구에서 제시된 방법으로 가시화 된 결과
(b) Meissner가 제시한 방법으로 가시화 된 결과
그림 4 engine 데이타에 대한 본 연구에서 제시된 방법과 Meissner[7] 방법과의 화질 비교.



(a) (b)

(a) 뼈 부분을 추출 한 후 렌더링한 결과
(b) 피부 부분을 추출 한 후 렌더링한 결과
그림 5 bighead 데이타를 다르게 추출한 결과

보다 좀 더 좋은 성능을 내는 것을 볼 수 있고, 또한 넓은 영역해 대해서 좀더 빠른 것을 알 수 있다.

Meissner[7]의 방법에 비해 본 논문에서 제시한 방법으로 인해 전체 렌더링 시간이 대략 6배에서 8배 정도의 향상이 있음을 볼 수 있다. Meissner가 제시한 방법에서는 매 텍스쳐 샘플링 슬라이스마다 픽셀 텍스쳐를 이용하기 위해 glReadPixels(), glDrawPixels()를 수행하여 프레임 버퍼와 주 메모리 사이에 과도한 데이타를 이동이 있는 것에 비해, 본 방법에서는 프레임 버퍼 안에서만 추출 및 음영처리를 행하고, 특히 음영처리는 최종 이미지에서 단 한번의 glCopyPixels()만을 수행함으로 이런 향상율이 얻어진 것이다.

그림 2는 small sphere 데이타를 본 논문에서 제시된 방법으로 렌더링을 수행한 결과이다. 그림 2(a)는 최종 프레임 버퍼에 있는 합성된 법선 벡터를 나타내고, 그림 2(b)는 대각선 방향으로 광원을 설정한 후 그림 2(a)의 합성된 법선 벡터로부터 음영 처리한 렌더링 결과이다.

그림 3은 small brain 데이타에 대하여 광원의 방향을 달리하여 생성된 이미지를 보여준다.

그림 4는 engine 데이타의 렌더링 결과이다. 그림 4(a)는 본 논문에서 제시된 방법으로 가시화 된 결과이고, 그림 4(b)는 Meissner가 제시한 방법으로 가시화 된 결과이다. 두 이미지의 결과가 똑같음을 볼 수 있다.

그림 5는 bighead 데이타의 렌더링 결과이다. 그림 5(a)는 뼈 부분만을 추출한 후 렌더링한 것이고, 그림 5(b)는 피부 부분을 추출한 후 렌더링 한 결과이다.

## 6. 결론 및 향후 연구 과제

본 논문에서는 3D 텍스쳐 매핑 하드웨어 하에서 빠른 추출 및 음영 처리를 지원하는 볼륨 렌더링 방법을 제시하였다. 본 논문에서 제시한 법선 벡터 블렌딩(normal blending) 방법을 통해서 추출 및 음영처리가 그래픽스 시스템에서 가능하며, 특히 음영처리 연산을 볼륨 데이타의 크기에 상관없이 최종 이미지에서 단 한번만 수행함으로서, 상호 대화적인 직접 볼륨 렌더링이 가능하도록 하였다.

향후 연구 방향으로는 OpenGL과 extension의 다른 여러 가지 기능을 이용하여 추출 과정이 좀더 확장되어도 법선 벡터 블렌딩이 현재의 그래픽스 시스템에서 가능하도록 하는 것이다. 또한 고 기능 그래픽스 시스템(high end graphics system)뿐만 아니라 일반 PC에 기반한 그래픽스 시스템에서도 3D 텍스쳐 하드웨어를 이용하여 볼륨 렌더링을 효율적으로 할 수 있는 방법을

개발할 예정이다.

## 참 고 문 헌

[1] Arie Kaufman, "Volume Graphics", IEEE Computer, pp.51-64, July 1993.

[2] B.Cabral, N. Cam, and J. Foran, "Accelerated Volume Rendering and Tomographic Reconstruction Using Texture Mapping Hardware", In 1994 Workshop of Volume Visualization, pp.91-98, Washington, DC, October 1994.

[3] T.J Cullip and U. Neumann, "Accelerating Volume Reconstruction with 3D Texture Mapping Hardware", Technical Report TR93-027, Department of Computer Science at the University of North Carolina, Chapel Hill, 1993

[4] A. Van Gelder and K. Kim, "Direct volume rendering with shading via three-dimensional textures", In Volume Visualization Symposium Proceedings, pp.23-30, San Francisco, CA, October 1996

[5] O. Wilson, A. Van Gelder, and J. Wilhelms, "Direct Volume Rendering via 3D Textures", Technical Report UCSC-CRL-9419, University of California, Santa Cruz, 1994.

[6] R. Westermann and T. Ertl, "Efficiently Using Graphics Hardware in Volume Rendering Application", In Computer Graphics, Proceeding of SIGGRAPH 98, pp169-177, Orlando, FL, August 1998.

[7] M. Meissner, U. Hoffmann, and W. Strasser, "Enabling Classification and Shading for 3D Texture Mapping based Volume Rendering using OpenGL and Extensions", Proceedings of the conference on Visualization '99, pp207-214, San Francisco, USA, 1999

[8] W.E Lorensen and H. E. Cline, "Marching cubes : A high resolution 3d surface construction algorithm", In Computer Graphics, Proceedings fo SIGGRAPH 87, pp163-169, 1987.

[9] M. Levoy," Display of surfaces from volume data", IEEE Computer Graphics & Application, 8(5) pp29-37, May 1988.

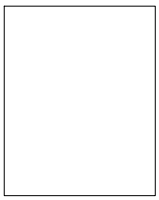
[10] L. Westover, "Footprint Evaluation for Volume Rendering, In Computer Graphics", Proceedings of SIGGRAPH 90, pp367-376, August 1990.

[11] R. A. Drebin, L. Carpenter, and P. Hanrahan, "Volume Rendering", Computer Graphics, 22 (4), pp65-74, August 1988.

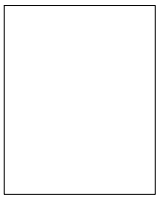
[12] P. Lacroute and M. Levoy, "Fast Volume Rendering Using a Shear-Warp factorization of the Viewing Transform", In Computer Graphics, Proceeding of SIGGRAPH 94, pp451-457, July 1994.

[13] K. Akeley, "RealityEngine Graphics", In Computer Graphics, Proceedings of SIGGRAPH 93, pp109-116, August 1993.

[14] Silicon Graphics Inc, "Pixel texture extension", Specification document, http::/www.opengl.org, 1996.



윤 성 의



신 영 길