Journal of Korea Game Society 2013 Apr; 13(2): 99-110
http://dx.doi.org/10.7583/JKGS.2013.13.2.99

컴퓨터 그래픽스

다중 깊이 영상을 이용한 볼륨-표면 혼합 가시화

계희원
한성대학교
kuei@hansung.ac.kr

Intermixing Surface and Volume Visualization Using Layered Depth Images

Heewon Kye
Hansung University

요 약

컴퓨터 게임에 볼륨 가시화 기법이 적용되면서, 하나의 화면에 표면 데이터와 볼륨 데이터를 혼합하여 가시화하려는 요구가 발생하고 있다. 최신 그래픽스 하드웨어의 범용 연산 기능을 사용하면 혼합 가시화를 수행할 수 있으나, 컴퓨터 게임은 저사양 하드웨어에서도 동작해야 하는 경우도 있어 혼합 가시화를 수행하기 어렵다. 본 연구는 DirectX 9.0 기반의 범용 하드웨어에서 볼륨-표면 혼합 가시화를 수행하는 방법을 제안한다. 우선, 표면 데이터를 가시화하여 다중 깊이 영상을 생성하는 방법을 제안한다. 이때, 생성 시간을 단축하는 깊이 복잡도 축소 방법을 제안한다. 이후, 생성된 다중 깊이 영상을 이용하여, 볼륨-표면 혼합 가시화를 수행한다. 혼합 가시화 과정에서 표면 데이터와 볼륨 데이터 사이의 좌표계 변환 방법과 혼합 가시화의 가속화 방법을 제안한다. 이를 통해 볼륨-표면 혼합 가시화를 효율적으로 수행할 수 있다.

ABSTRACT

As volume rendering has been applied for computer game, the visualization of volume data with surface data in one scene has been required. Though a hybrid rendering of volume and surface data have been developed using the GPGPU functionality, computer games which run on low-level hardware are difficult to perform the hybrid rendering. In this paper, we propose a new hybrid rendering based on DirectX 9.0 and general hardware. We generate the layered depth images from surface data using a new method to reduce the depth complexity and generation time. Then, we perform the hybrid rendering using the layered depth images. In the rendering process, we suggest a new method to transform the coordinate system from a surface coordinate to a volume coorinate and propose an accelerated rendering technique. As the result, we can perform volume-surface hybrid rendering in an efficient way.

Keywords : volume visualization(볼륨 가시화), volume-surface hybrid rendering (볼륨-표면 혼합 가시화), layered depth images (다중 깊이 영상), graphics hardware(그래픽스 하드웨어)

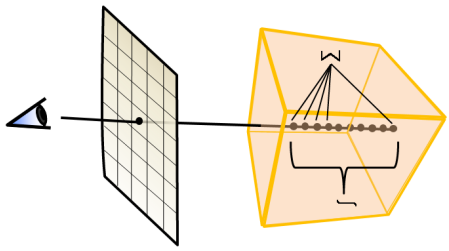
Received: Dec. 10, 2012 Revised: Mar. 14, 2013
Accepted: Apr. 02, 2013
Corresponding Author: Heewon Kye(Hansung University)
E-mail: kuei@hansung.ac.kr

ISSN: 1598-4540 / eISSN: 2287-8211

Ⓒ The Korea Game Society. All rights reserved. This is an open-access article distributed under the terms of the Creative Commons Attribution Non-Commercial License (http://creativecommons.otg/licenses/by-nc/3.0), which permits unrestricted non-commercial use, distribution, and reproduction in any medium, provided the original work is properly cited.

1. 서 론

컴퓨터 게임의 가시화 기술은 끊임없이 발전하고 있으며, 사용자의 요구에 부응하여 다양한 효과를 추가하고 있다. 그 중 하나로 볼륨 가시화 기법[1]을 게임에 적용하려는 요구가 높다. 볼륨 가시화(volume visualization)는 [Fig. 1]과 같이 삼차원 입방체 형태의 볼륨 데이터를 가시화 하여 영상을 생성하는 방법으로, 과학 시뮬레이션 등에서 주로 사용되는 기법이다. 볼륨 가시화는 의료 수술 시뮬레이션과 같은 교육용 게임에 주로 사용되며, 구름, 연기와 같은 특수효과 표현에도 사용된다.



[Fig. 1] Volume rendering

볼륨 가시화에 사용되는 볼륨 데이터는 부피 내부가 채워진 반투명한 가스 모델이다. 따라서 일반 게임에서 사용되는 다각형 표면 데이터(surface data)와 가시화 방법이 다르다. 일반적으로 [Fig. 1]과 같이 영상의 각 화소(pixel)를 통과하는 가상의 광선을 생성하고 광선상의 색상 정보를 종합하여 화소의 색상을 결정한다.

[Fig. 2]에 대략의 알고리즘을 제시한 바와 같이, 광선의 색상을 초기화하고 광선상의 표본 위치를 광선 방향으로 조금씩 진행하며 표본 위치에서의 색상을 누적한다. 흔히 영상 순서 렌더링(image order rendering)또는 광선 추적법(ray casting)이라고 불리는 이 기법은 게임에서 사용하는 표면 가시화(surface rendering) 기법과 매우 다르다. 특히 볼륨 모델과 표면 모델을 혼합할 때, 앞뒤 순서가 어긋나지 않도록 깊이 정렬(depth sorting)을 주의해야 한다.

```
for each pixel
  position = Get_start_position(pixel)
  ray_col = (0,0,0,0) // ray color, black
  while position is inside volume
    color = Get_sample_color(position)
    ray_col = alpha_blending(ray_col, color)
    position = position + ray direction (A)
  write ray_col to image buffer
```

[Fig. 2] Ray casting algorithm

표면 데이터가 불투명한 도형으로만 구성되어 있다면, 깊이 정렬 문제는 쉽게 해결된다[2]. 먼저, 표면 데이터를 가시화(polygon rendering)하여 깊이 버퍼(depth buffer)를 생성한다. 그리고 볼륨 가시화 단계에서 깊이 버퍼를 참조하며 혼합 가시화한다. [Fig. 2]의 (A)단계에서, 위치가 깊이 버퍼에 저장된 표면의 깊이를 넘는 순간, 표면 색상을 반영한다. 그 결과 볼륨 데이터와 불투명한 표면을 오류 없이 혼합 가시화 할 수 있다. 한편, 하나의 광선이 볼륨 데이터를 관통하는 동안 광선 진행을 n번 수행 하였다면, O(n)회의 추가 비교 시간이 필요하다.

표면 데이터에 반투명한 도형이 있으면 깊이 정렬은 어려운 문제가 된다. 예를 들어, 하나의 광선과 교차하는 중첩된 다각형이 d개라면 비교는 O(dn)회로 증가한다. 이 문제를 해결하기 위해 다양한 방법의 관련연구가 제안되었다.

고전적인 방법으로, 볼륨 데이터를 표면 데이터로 변환[3]하여 표면 가시화만을 수행하는 방법이 제안되었다. 이 방법은 볼륨 가시화의 장점인 부피감을 얻지 못하는 단점이 있다. 역으로, 표면 데이터를 볼륨 데이터로 래스터 변환(rasterization)하여[4] 볼륨 가시화만을 수행하면, 래스터 변환 과정에서 에일리어싱(aliasing) 현상이 발생하며, 회전이나 확대 등의 변환이 매우 불편하다는 점이 단점이다. 따라서 이후 각 데이터의 특징을 보존하며 혼합 가시화하는 연구가 진행되었다.

예를 들어, 관찰자로부터 매우 가까운 영역만을 볼륨 가시화한 다음, 해당 영역에 겹치는 표면 데이터를 가시화한다. 이후 조금 더 먼 영역을 볼륨

가시화하고 다시 해당 영역을 표면 가시화 한다. 이 과정을 반복해 혼합 가시화를 수행한다[2,5]. 그러나 이 방법은 영역의 판별이 복잡하며, 매번 깊이 영역을 갱신해야 하는 번거로움이 존재한다. 관찰자의 시점에 따라 거리가 변화하기 때문이다.

한편, 표면 데이터의 정렬된 깊이 값을 각 화소별로 모두 알 수 있다면 혼합 가시화를 수행할 수 있다. 볼륨 가시화 단계에서 표면의 깊이 값을 순차적으로 고려하여 혼합 가시화할 수 있으므로, 비교는 O(dn)회에서 O(d+n)회로 감소한다. 그러나 다각형을 정렬하는데 필요한 사전 시간인 최소 O(d log d)회가 추가로 필요하다. 최신 그래픽스 하드웨어(GPU)의 범용 계산 기능(GPGPU)를 사용하면 깊이 정렬을 빠르게 수행할 수 있는데[6,7], 모든 사용자가 GPGPU가 포함된 하드웨어를 사용하지는 않는다는 점을 감안하면 사용이 제한적이다.

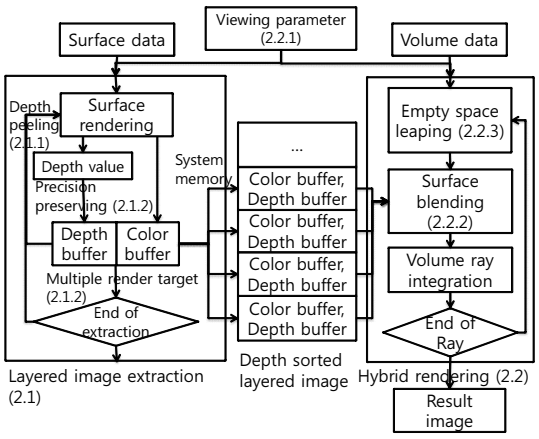
본 연구는 혼합 가시화를 위해, 다중 깊이 영상(layered depth images)을 생성하는 방법을 제안한다. 제안 방법은 깊이 정렬에 $O(d^2)$회의 시간이 소요되어 기존의 O(d log d)회 보다 느리다. 그러나 이 방법은 저사양의 GPU에서도 실행 가능하기 때문에 GPGPU[6,7] 에서만 동작하거나 속도가 느린 CPU에서 동작[8]하는 기존 방법에 비해 폭넓게 사용하기에 좋은 선택이다[9]. 여기서 본 연구의 목적은 일반적으로 사용되는 사양의 GPU에서 원활하게 동작하며, 구형의 GPU에서는 느리지만 혼합 가시화 기능이 오류 없이 수행되는 것이다.

또한 본 연구는 생성된 계층적 깊이 영상을 이용하여 볼륨 데이터와 표면 데이터의 혼합 가시화를 수행하는 방법을 제안한다. 이때, 볼륨 데이터와 표면 데이터간의 좌표계를 일치하는 방법과 가시화의 효율을 향상시키기 위한 빈공간 도약 기법(empty space skipping)을 제안한다[10]. 이를 이용하여 혼합 가시화에 필요한 비교 횟수를 O(d+n)회에서 O(d + log n)회로 향상시킨다.

이후 2장에서 연구 방법을 설명하고 3장에서 실험 결과 보이며 4장에서 결론을 맺는다.

2. 연구 방법

본 연구는 표면 데이터를 추출하여 깊이 정렬된 다중 깊이 영상을 생성하고 그 결과를 이용하여 혼합 가시화를 수행하는 단계로 이루어진다. 간략한 시스템의 개요를 [Fig. 3]에 제시하였다. 그림의 각 숫자는 본문에서 설명되는 절을 의미한다.



[Fig. 3] System overview

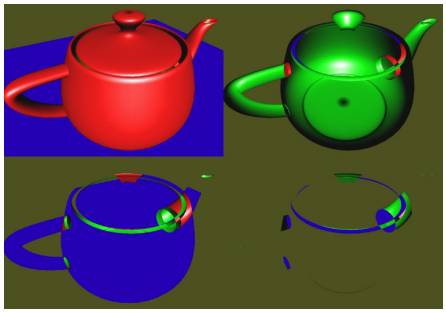
2.1 깊이 박피를 이용한 다중 깊이 영상

이번 절에서는 볼륨-표면 혼합 가시화를 위해 표면 데이터를 가시화하여 다중 깊이 영상을 생성하는 방법을 설명한다. 관련연구로서 기본이 되는 깊이 박피 알고리즘을 설명하고 이를 이용하여 다중 깊이 영상을 생성하는 방법에 대해 제안한다.

2.1.1 깊이 박피 알고리즘

깊이 박피(depth peeling)는 깊이 계층(depth complexity)이 있는 장면에서 각 단계가 진행할 때 마다 시점에서 가까운 영상을 단계적으로 추출하는 기법[11]이다. 일반적으로 한 번의 표면 가시화를 수행하면 시점에 가장 가까운 표면만이 보인다. 새롭게 표면을 그리면서 직전 단계에 저장된 깊이보다 먼 표면만을 그리면 시점에서 두 번째로

가까운 표면만이 보이게 된다. 이를 반복하면 중첩된 표면 영상을 한 겹씩 제거하며 영상을 생성하는 효과를 얻을 수 있다.



[Fig. 4] Depth peeling illustration [11]

예를 들어 [Fig. 4]와 같이 각 단계(pass)를 진행할 때 관찰자로부터 가까운 순서대로 표면의 색상과 거리 정보(layer)가 추출된다. 매 반복마다 모든 다각형을 그려야 하므로 화소 당 연산 횟수는 $O(d^2)$회가 된다.

그러나 깊이 박피법은 이론적인 방법이라 실제 구현에는 추가 고려가 필요하다. 깊이 박피법은 한 번의 가시화에 두 번의 깊이 검사가 필요하다. 실제 GPU는 한 번의 깊이 검사만을 지원하기 때문에, Everritt은 깊이 값을 알파 채널(alpha channel)에 저장하고 알파-검사(alpha test)를 이용하여 추가의 깊이 검사를 대체하는 방법을 제안[11]하였는데, 알파 채널의 정밀도(8 비트)가 통상적인 깊이 값의 정밀도(23 비트)에 비해 부족한 점이 문제로 지적된다.

2.1.2 다중 깊이 영상의 생성

본 연구는 추가의 깊이 검사를 텍스처와 프로그램 가능한 세이더 연산을 이용하여 수행한다. 깊이 박피의 이전 단계에서 추출된 깊이 값을 저장하고, 다음 단계에서 물체의 가려짐을 판별하는 조건으로 사용한다.

초기화 단계에서, 영상의 화소와 동일한 해상도의 텍스처를 생성한다. 가시화 단계에서는 이전 레이어(layer)까지 박피된 깊이 값을 색상 값으로 변환하여 텍스처에 저장한다. 이후 가시화 단계에서 텍스처에 저장된 색상 값을 다시 깊이 값으로 역변환하여, 현재 저장된 깊이보다 가까운 물체를 가시화에서 제거한다.

```
for each fragment in polygons
  if(fragment.depth < in_texture[pos].depth)
    culling // depth peeling
  if(fragment.depth > depth_buffer[pos])
    culling // depth test
  out_texture[pos].depth = fragment.depth
  depth_buffer[pos] = fragment.depth
  frame_buffer[pos] = fragment.color
copy from out_texture to in_texture
```

[Fig. 5] Depth peeling using MRT

이를 구체적으로 설명하면 [Fig. 5]와 같다. 먼저, 각 다각형이 가시화되면서 각 화소(fragment)의 깊이 값이 생성된다. 화소의 깊이 값이 지금까지 누적된(in_texture) 깊이보다 가까운 경우, 이전 단계에서 이미 추출 되었다고 판단하여 제거된다. 이후 일반적인 깊이 검사를 통해 가장 가까이 있는 화소가 영상에 누적되며, 자동적으로 깊이 버퍼(depth_buffer)와 색상 버퍼(frame_buffer)가 갱신된다.

다음 단계의 깊이 박피를 위해, 누적된 깊이 버퍼가 갱신되어야 한다. 일반적인 GPU는 텍스처의 읽고 쓰기가 동시에 발생할 수 없으므로 본 연구는 별도의 텍스처(out_texture)를 만들어 값을 저장한다. 가시화가 끝난 후 저장된 값을 누적 텍스처(in_texture)에 복사하여 다음 단계의 준비를 완료한다.

한 단계의 가시화가 끝나면 표면의 색상과 깊이 값을 시스템 메모리에 출력한다. 두 정보를 동시에 출력하기 위해 본 연구는 다중 렌더링 대상 기법(multiple render target, MRT)을 사용한다. 이 기법은 하나의 다각형(polygon)을 여러 출력 표면(surface)에 각각 다른 효과로 가시화하여, 가시화가 끝나면 복수의 영상을 얻는 기법이다.

본 연구는 DirectX의 프로그램 가능한 세이더(programmable shader)를 사용하여, [Fig. 6]과 같이 깊이 값을 색상 값으로 변경하여 저장하였다. 기존 Everritt의 방법이 8 비트 정밀도에 불과한 반면, 본 연구는 각 채널별 8 비트의 RGB에 저장하여 24 비트의 정밀도를 보장하였다. 깊이 버퍼의 범위는 [0, 1]이지만, 저장할 때는 볼륨 가시화를 위해 복셀(voxel) 단위로 크기변환 되어, [0, 10000] 범위로 증가한다. 이때, 정수부를 저장하기 위해 R과 G색상을 이용하였고, 소수부를 저장하기 위해 B색상을 이용하였다.

```
float4 Depth2RGB(float depth)
{
   float4 rgb = 0;
   float high_bit = floor(depth/256.0);
   rgb.r = high_bit / 255.0;
   float low_bit = floor(depth-high_bit*256.0);
   rgb.g = low_bit / 255.0;
   rgb.b = depth- floor(depth);
   return rgb;
}
```

[Fig. 6] Precision preserving using color channel

참고로 DirectX 세이더 문법에서 각 색상은 [0, 1] 범위의 실수가 [0, 255] 범위의 8 비트 정수로 변환된다. 따라서 색상으로 변환할 때, 256이 아닌 255로 나누어야 올바른 결과를 얻는다. 한편, 응용 프로그램에 따라 필요한 정수부의 정밀도가 다른데, [Fig. 6]의 나눗셈에 해당하는 값을 변경하면 간단히 대응이 가능하다.

이상 설명한 MRT 및 세이더 프로그래밍 방법은 DirectX 9.0c 에서 지원하는 세이더 버전 2.0에 포함되어 있는 기능이므로, 2005년 이후 생산된, 현재 대부분의 사용자가 소유한 하드웨어 환경에서 실행이 가능하다.

2.1.3 불투명한 도형의 깊이 복잡도 축소

기존 깊이 박피 알고리즘은 전체 데이터를 깊이 박피 하는 것이 목적이다. 그러나 본 연구는 혼합 가시화가 목적이므로 불투명한 표면 뒤에 가려지는 다른 표면 데이터는 보이지 않으니 무시해도 좋다.

이 착상을 구현하기 위해, 간 단계마다 불투명한 물체만 골라 반투명한 물체보다 먼저 가시화하였다. 이는 DirectX의 알파 테스트 기능을 이용하여 쉽게 구현 가능하다. 깊이 버퍼가 불투명한 물체로 초기화되고, 반투명한 물체를 이후에 그리며 깊이 박피를 수행한다.

깊이 박피 단계에서 가려지는 반투명한 물체는 GPU의 깊이 검사가 동작하여 자연적으로 연산에서 생략된다. 게다가 가려지는 반투명한 다각형의 수가 감소하는 효과가 있기 때문에 깊이 영상의 깊이 복잡도가 감소하는 효과가 있다. 그 결과로 수행 시간과 메모리 사용량을 절감할 수 있다.

2.2 다중 깊이 영상을 이용한 가시화

이번 절은 앞 절에서 생성한 다중 깊이 영상을 이용하여 혼합 가시화를 수행하는 방법을 설명한다. 먼저 볼륨과 표면 데이터의 좌표계 일치 방법에 대해 설명하고, 구체적인 혼합 가시화 방법을 설명한다. 마지막으로 공간 도약을 이용한 가속화 방법을 제안한다.

2.2.1 볼륨-표면 데이터의 좌표계 일치

볼륨-표면 혼합 가시화를 수행하려면 두 데이터 간의 좌표계가 일치해야 한다. 볼륨 가시화의 좌표계는 볼륨 데이터의 해상도인 복셀 단위로 설정되는 것이 일반적이다. 예를 들어 볼륨 데이터의 크기가 512×512×300 이라면 참조 가능한 볼륨 좌표계의 좌표는 (0, 0, 0) - (511, 511, 299)가 된다.

그러나 표면 데이터의 좌표를 볼륨 좌표계에 맞춰 생성하기는 어렵다. 복셀의 크기가 데이터마다 다르므로, 볼륨 데이터가 변경되면 그에 따라 표면 데이터의 좌표가 변경되어야 하기 때문이다. 따라서 볼륨과 표면 데이터에 공통적으로 적용할 수 있는 좌표계 구성이 필요하다.

본 연구는 많은 수의 볼륨 데이터를 차지하는 의료영상 데이터의 특성에 주목하였다. 표준 의료영상 데이터(DICOM) 파일의 헤더 정보에는 각 복셀의 크기 정보가 포함된다. 따라서 밀리미터 단위의 표준 좌표계를 도입하고, 볼륨 데이터와 표준 좌표계, 표면 데이터와 표준 좌표계 사이의 관계를 정의하였다. 두 관계를 분석하여 최종적으로 볼륨 데이터와 표면 데이터의 좌표계를 일치시켰다. 즉, 표면 데이터의 좌표는 볼륨의 좌상단을 기준으로 밀리미터 단위의 거리를 예상하여 결정하였다.

표면 데이터의 가시화는 DirectX를 사용하므로, 물체 변환(world transform), 관찰자 변환(view transform), 투영 변환(projection transform)의 표준화된 그래픽스 파이프라인을 이용하였다. 본 연구에서 물체 변환, 관찰자 변환, 투영 변환을 결정하는 방법은 다음과 같다.

먼저, 물체 변환의 초기 값은 단위행렬을 유지하고 사용자 입력에 따라 변경할 수 있도록 하였다. 이로서, 사용자가 표면 데이터의 위치를 자유롭게 변경할 수 있게 된다. 관찰자 변환은 DirectX 또는 OpenGL 등에서 제공하는 LookAt 개념을 차용하였다. 관찰자 위치는 볼륨 데이터의 좌상단, 관찰 방향은 광선 진행 방향으로 설정하였다. 투영 변환은 볼륨 가시화의 윈도우와 표면 가시화의 절단면을 일치시켜야 한다. 볼륨 가시화의 윈도우 크기를 관찰 파라미터를 분석하여 밀리미터 단위로 변환한다. 이후, 윈도우 영역이 표면 가시화의 절단면 범위인 [-1, 1]크기가 되도록 크기변환 및 평행이동 요소를 결정하여 투영 변환을 완료한다.

투영 변환 및 뷰포트(viewport) 변환에 대한 식을 (eq. 1)과 같이 정리할 수 있다. 단, 영상의 크기 *width*, *height*는 픽셀단위의 크기와 밀리미터 단위의 크기를 모두 저장하고 있으며, 평행투영을 기준으로 투영 변환의 깊이 *depth*는 밀리미터 단위만 저장하고 있으면 된다.

정리하면, 사용자는 밀리미터 단위 좌표로 물체 변환을 입력한다. 회전 등의 관찰자 변환은 마우스 등의 사용자 입력을 통해 관리하고, 좌표계의 일치는 DICOM 헤더에 저장된 화소의 밀리미터 단위의 크기와 모니터 윈도우의 화면 크기를 이용하여 제안 시스템이 투영 변환과 뷰포트 변환을 자동으로 계산하여 좌표계를 일치시켰다.

$$Viewport = (width.pixel, height.pixel)$$

$$M_{proj} = \begin{pmatrix} \frac{2}{r-l} & 0 & 0 & 0 \\ 0 & \frac{2}{t-b} & 0 & 0 \\ 0 & 0 & \frac{2}{z_f - z_n} & 0 \\ \frac{l+r}{l-r} & \frac{b+t}{b-t} & \frac{z_n}{z_n - z_f} & 1 \end{pmatrix} \quad \text{(eq. 1)}$$

$$\begin{pmatrix} l = 0, r = width.mm \\ \text{단,} \quad t = 0, b = height.mm \\ z_n = -depth.mm, z_f = depth.mm \end{pmatrix}$$

2.2.2 혼합 가시화 방법

본 연구에서 제안하는 혼합 가시화 단계는 다음과 같다. 먼저 앞 절에서 생성된 좌표계를 바탕으로 표면 데이터를 가시화한다. 이후, 볼륨 가시화를 수행할 때, 각 픽셀에서 생성된 광선은 일반적인 광선 추적법(ray casting) 기반의 볼륨 가시화 [12]를 수행한다.

```
for each pixel
  position = Get_start_position(pixel)
  ray_col = (0,0,0,0) // ray color, black
  layer_num = 0
  while position is inside volume (A)
    color = Get_sample_color(position)
    ray_col = alpha_blending(ray_col, color)
    if Is_across(position, layer_num)
      ray_col = alpha_blending
                (ray_col, LDI(layer_num))
      layer_num += 1 // next layer
    position += ray direction
  write ray_col to image buffer
```

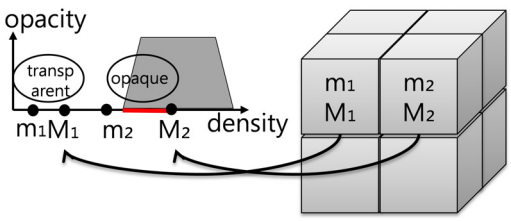
[Fig. 7] Hybrid rendering algorithm

앞서, [Fig. 3]에 나타내었듯이, 시스템 메모리에

저장된 색상 버퍼와 깊이 버퍼를 그대로 이용하여 혼합 가시화를 수행한다. 광선을 진행하면서 다중 깊이 영상의 깊이 값을 통과하게 되면, 해당 다중 깊이 영상 화소의 색상과 투명도를 광선에 반영하여 혼합 가시화를 수행한다. [Fig. 2]의 광선 추적법 알고리즘을 확장하여 제안된 혼합 가시화 알고리즘을 [Fig. 7]에 나타내었다. (단, LDI는 깊이 정렬된 다중 깊이 영상의 참조를 의미한다.)

2.2.3 지연된 표면 결합을 이용한 효율적인 빈공간 도약

볼륨 가시화 속도를 향상시키기 위한 대표적인 방법 중 하나는 빈공간 도약 알고리즘을 적용하는 것이다. 빈공간 도약은 볼륨 데이터의 투명한 지역 정보를 가시화 전에 미리 저장하여 두었다가 볼륨 가시화 단계에서 투명한 부분의 처리를 건너뛰는 가속화 방법[13]이다.



[Fig. 8] Empty space leaping method [14]

[Fig. 8]은 볼륨 데이터를 8개의 블록으로 구분하고 각 블록 영역의 최솟값 m과 최댓값 M을 미리 계산해 저장한 모형이다. 일반적으로 사용자는 관찰하고 싶은 밀도 영역을 그림과 같이 사다리꼴 형태로 정의하는데, 밑변의 범위를 [d_1, d_2]와 같은 값으로 입력한다. 이때, 어떤 블록에 대해 다음 식이 성립한다면, 그 블록의 내부는 불투명한 부분이 없어 모두 투명하므로 가시화를 생략할 수 있다.

$$\{x|d_1 \le x \le d_2\} \cap \{x|m \le x \le M\} = \varnothing$$

예를 들어 [Fig. 8]에서 첫 번째 블록의 최대, 최소 범위 [m_1, M_1]은 사다리꼴 구간과 교집합이 없으므로 가시화에서 생략한다. 이런 블록 자료구조를 미리 생성하여 광선 진행 중 투명한 영역을 빠르게 처리할 수 있다. 블록을 계층적인 트리 구조로 상세화 할 경우[13], [Fig. 7]의 (A)영역에 표시된 반복 횟수가 O(n)회라면 트리를 이용하여 가속화된 반복 횟수는 O(log n)회로 감소하게 된다.

그런데 빈공간 도약은 볼륨 가시화를 위해 제안된 방법이므로, 혼합 가시화 직접 적용하기는 어렵다. 빈공간 도약을 혼합 가시화에 적용하면, 투명한 지역에 속한 표면 데이터를 생략하는 문제가 발생하기 때문이다.

```
for each pixel
  position = Get_start_position(pixel)
  ray_col = (0,0,0,0) // ray color, black
  layer_num = 0
  while position is inside volume
    if Is_transparent_region(position)  (A)
        position += Skip_length(position)
        continue the while loop
    while Is_over(position, layer_num)
      ray_col = alpha_blending
          (ray_col, LDI(layer_num))  (B)
      layer_num += 1 // next layer
    color = Get_sample_color(position)
    ray_col = alpha_blending(ray_col, color)
    position += ray direction

  while Is_over(position, layer_num)
    ray_col = alpha_blending
        (ray_col, LDI(layer_num)) (C)
  write ray_col to image buffer
```

[Fig. 9] Hybrid rendering acceleration

본 연구는 이 문제를 해결하기 위해 표면 결합을 지연하는 가속 방법을 제안한다. 이 방법은 빈공간을 만나면, 표면데이터 정보를 일단 무시하고 도약하여 효율을 향상한다. 대신 도약을 마치고 유효한(불투명한) 첫 번째 볼륨 데이터를 만날 때, 그동안 무시하고 통과한 모든 표면을 뒤늦게 누적하여 반영한다. 볼륨 데이터의 빈공간만을 도약했

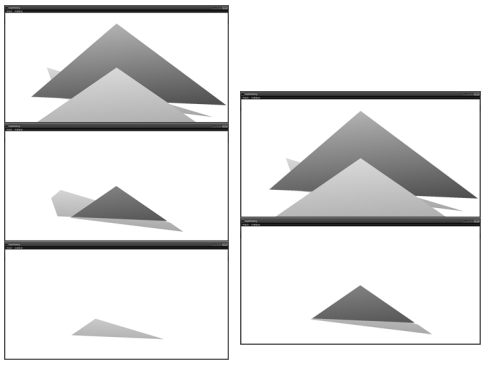
으므로, 뒤늦게 표면 정보만을 누적해도 영상의 화질에는 차이가 없게 된다.

[Fig. 9]에 제시한 바와 같이, 투명한 지역은 [Fig. 9](A)와 같이 바로 건너뛴다. 그리고 불투명한 영역을 처리하기 직전, 무시한 표면 데이터의 색상을 [Fig. 9](B)와 같이 누적하여 오류 없는 영상을 생성한다. 이때 공간 도약의 효율이 그대로 유지되므로 비교 횟수가 기존의 O(d+n)회에서 O(d + log n)회로 감소하여 속도가 향상된다. 참고로, 광선의 종료시 표면 데이터의 누적 검사를 [Fig. 9](C)와 같이 추가로 수행해야 한다.

3. 실험 결과

본 연구는 DirectX 9.0 SDK를 이용하여 구현하였고 세이더 모델 2.0을 사용하였으므로, DirectX 9.0c를 지원하는 기존 하드웨어에서 실행 가능하다. 시간 측정은 일반적인 사양으로 인텔 i5 CPU를 장착한 PC에서 수행하였으며, 그래픽스 장치는 nVidia의 Geforce GTX 570을 사용하였다.

이번 장에서는 먼저 다중 깊이 영상(2.1절)의 생성 결과를 먼저 보이고, 이를 이용한 혼합 가시화 결과(2.2절)을 보인다. 마지막으로, 가속화 방법을 이용한 성능 향상(2.2.3절) 결과를 제시한다.



(a) (b)

[Fig. 10] Layered images of surface data

우선, GPU를 이용하여 표면 데이터를 가시화하였고, 그 결과로 얻은 다중 깊이 영상을 시스템 메모리로 전송하였다. 세 개의 반투명한 삼각형을 생성하고, 깊이 박피 알고리즘을 이용하여 각 단계에 대해 영상을 추출하였다. 그 결과 영상이 [Fig. 10]으로 생성되었다. [Fig. 10](a)에 제시된 바와 같이 각 표면이 순차적으로 추출된다. 투명도 누적은 렌더링 단계에서 일어나므로 출력 영상은 누적되지 않는다.

한편 2.1.3절에서 설명한 바와 같이 불투명한 도형이 있는 경우 깊이 복잡도를 줄일 수 있다. 기존의 데이터에 중간의 어두운 도형만을 완전히 불투명하게 변경하고, 다시 다중 깊이 영상을 추출하여 [Fig. 10](b)에 나타냈다. 불투명한 도형 뒤에 가려진 도형은 생성될 필요가 없으므로, [Fig. 10](a)에 비해 깊이가 한 단계 축소된다.

[Table 1] Creation time of layered image (ms)

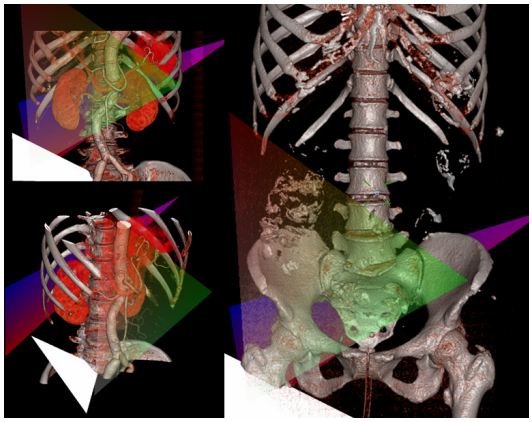
Resolution	(a) depth images	(b) depth reducing
700×410	13	9
1400×820	48	32

다음으로, 다중 깊이 영상 생성에 소요되는 시간을 측정하여 [Table 1] 에 나타내었다. 두 종류의 화면 해상도를 이용하여 실험을 수행하였고, 깊이 복잡도 축소 기법이 포함되지 않은 경우(A)와 포함된 경우(B)를 분리하여 시간을 측정하였다. 실험 결과, 해상도의 화소수가 4배 증가하면 시간 또한 4배 증가하여, 화소에 비례하는 시간이 소요되었다. 그리고 제안한 깊이 복잡도 축소 방법을 이용하여 세 단계의 출력 영상(A)을 두 단계로 축소(B)하였을 때, 전반적으로 1.5배의 성능 향상을 얻었다.

앞의 결과로, 다중 깊이 영상의 생성 시간은 시스템 메모리로 출력하는 화소 수에 비례하는 것으로 파악된다. 일반적으로 GPU의 가시화 성능은 대단히 우수하여 가시화 자체는 매우 빠르게 수행된다. 대부분의 수행 시간은 결과 영상을 시스템 메모리로 전송할 때 소비된다고 짐작(bandwidth

bottleneck)할 수 있다.

결과적으로 고해상도 영상의 경우 32ms가 소요되었는데, 고해상도 볼륨 가시화가 1초 이상 소요되는 작업임을 감안하면, 제안 방법을 적용했을 때 추가적인 시간 부담은 3% 이내가 되어 실용적으로 사용하기에 적당하다. 한편, 내장형 그래픽스 하드웨어를 장착한 노트북에서 시간을 측정한 결과, 저해상도에서 35ms가 소요되었다. 따라서 실행 시간이 오래 걸리는 깊이 복잡도가 큰 데이터나 저사양의 GPU의 경우, 화질 손실을 감수하고 화면 해상도를 약간 줄이면 수행 시간을 조절할 수 있다.



[Fig. 11] Results of hybrid rendering

다음으로, [Fig. 11]에서 혼합 가시화된 결과 영상을 표시하였다. 세 개의 삼각형을 표면 가시화하였으며, 그중 하나는 불투명한 흰색으로 표현하였다. [Fig. 11]의 왼쪽은 콩팥(kidney) 혈관 데이터의 혼합 가시화 결과이다. 관찰 방향을 다르게 하여 가시화를 수행한 결과, 표면 데이터의 위치가 올바르게 표현되고 있다. 반투명한 삼각형과 불투명한 삼각형의 깊이 순서가 올바르게 표현되어 있으며, 인체 데이터와의 깊이 관계도 오류 없이 표현되어 있다.

볼륨 데이터를 골격(bone) 데이터로 변경하여 혼합 가시화한 결과를 오른쪽에 보였다. 삼각형의 좌표가 고정된 위치의 원점(인체의 오른발 앞쪽)을 중심으로 보존된다. 동일한 표면 데이터가 서로 다른 볼륨 데이터에서 실제 크기(밀리미터 단위)에 맞도록 표현되어 있는 것을 관찰할 수 있다.

이를 통해, 표면 데이터 디자이너는 원점을 기준으로 모델을 생성할 수 있으며 단위는 밀리미터를 기준으로 하기 때문에, 수술 도구나 임플란트 모형과 같은 표면 데이터의 표준 모형을 제작하기 편리하다.

[Table 2] Rendering time (ms)

Dataset	existing method	proposed method
kidney	1826	1421
bone	1430	1198

마지막으로 혼합 가시화 영상을 생성하는데 소요된 시간을 [Table 2]에 제시하였다. 광선 추적법의 가시화 시간은 보통 화면 해상도에 비례한다. 따라서 위아래로 길게 표현된 골격 데이터는 가시화 시간이 상대적으로 짧게 측정되었고, 화면을 넓게 채우는 콩팥 데이터가 시간이 길게 측정되었다.

제안 방법인 지연된 표면 결합을 이용하면 빈공간 도약을 효율적으로 수행하게 된다. 그 결과로 20-30% 정도의 속도 향상을 얻었다.

4. 결 론

본 연구에서는 새로운 볼륨-표면 혼합 가시화 방법을 제안하였다. 표면 데이터를 이용하여 다중 깊이 영상을 생성하였고, 깊이 정렬된 다중 깊이 영상을 볼륨 가시화에서 이용하여 혼합 가시화를 수행하였다.

다중 깊이 영상을 생성할 때, 구형 하드웨어에서도 동작할 수 있도록 고려하였다. 깊이 박피 알고리즘을 이용하여 다중 깊이 영상을 생성하였으며 DirectX 9.0 기반의 구형 하드웨어 기능만을 사용하면서도 다중 렌더링 대상 기법을 활용하여 24비트 정밀도를 확보하고 깊이 정렬을 달성하였다는

것에 의의가 있다.

그리고 생성된 다중 깊이 영상을 생성하여 볼륨-표면 혼합 가시화를 수행하였다. 볼륨 데이터와 표면 데이터의 좌표계를 일치시키기 위해 표준 좌표계를 도입하였다. 그리고 가시화 속도를 향상시키기 위해 빈공간 도약 알고리즘을 적용하는 경우, 표면 데이터의 누락을 효율적으로 방지하는 지연된 표면 결합 방법에 대해 제안하였다.

실험 결과로서 표면 데이터를 이용한 혼합 가시화 영상이 오류 없이 생성되는 것을 확인하였으며, 앞으로 다양한 형태와 포맷의 표면 데이터를 이용하여 사용자 경험을 증가시킬 수 있을 것으로 예상한다. 본 연구의 한계로서, 깊이 박피 알고리즘이 깊이 복잡도와 화면 해상도의 곱에 비례하는 시간이 소요되므로, 깊이 복잡도가 높은 장면의 다중 깊이 영상의 생성 시간이 상대적으로 오래 걸린다는 문제가 있다. 따라서 깊이 복잡도가 높은 다각형 위주의 게임 보다 볼륨 가시화가 주된 의료 시뮬레이션 게임 등에 적합할 것으로 보인다. 향후 각종 가속화 알고리즘[15,16]을 이용하여 더 빠른 속도의 가시화를 수행하고자 한다.

ACKNOWLEDGMENT

This Research was financially supported by Hansung University.

REFERENCES

[1] K. Engel, M. Hadwiger, J. Kniss, C. Rezk-Salama, and D. Weiskopf, “Real-time volume graphics”, AK Peters, 2006.

[2] K. Kreeger and A. Kaufman, “A. Mixing translucent polygons with volumes”, IEEE Visualization’99. pp. 191-198, 1999.

[3] Lorensen W.E. and Cline, H.E., “Marching Cubes: A High Resolution 3D Surface Construction Algorithm,” Computer Graphics, Vol. 21, No. 4, pp. 163-169, July 1987.

[4] A. Kaufman, “An Algorithm for 3D Scan-Conversion of Polygons”, Proc. Eurographics ‘87, Elsevier Science Publishers B.V., pp. 197-208, 1987.

[5] I. Boada and I. Navazo, “3D texture-based hybrid visualizations”, Computers & Graphics, Vol. 27, pp. 41-49, 2003.

[6] B. Liu, L. Wei, and Y. Xu, “Multi-Layer Depth Peeling via Fragment Sort”, TechReport MSR-TR-2006-81, Microsoft Research, 2006.

[7] B. Kainz, M. Grabner, A. Bornik, S. Hauswiesner, J. Muehl, and D. Schmalstieg, “Ray casting of multiple volumetric datasets with polyhedral boundaries on manycore GPUs”, ACM Transactions on Graphics Vol. 28. 2009.

[8] W. Chen, W. Cai, E. Firle, M. Wang, and Q. Peng, “A Hardware Assisted LDI Building Algorithm with Application to Hybrid Volume Rendering”, Proceedings of the Seventh International Conference on Computer Aided Design and Computer Graphics, pp. 225-232, 2001.

[9] H. Kye, “A LDI building method for rendering polygon and volume data”, 2012 spring Korea Game Society conference proceeding, pp. 275-278, 2012.

[10] H. Kye, “Intermixing surface and volume visualization using layered depth image”, 2012 fall Korea Game Society conference proceeding, pp. 209-212, 2012.

[11] C. Everitt. “Interactive order-independent transparency”, Technical Report, NVIDIA Corp., 2001.

[12] J. Kajiya and B. Herzen, “Ray tracing volume densities,” Proc. of SIGGRAPH‘84, pp. 165-174, 1984.

[13] M. Levoy, “Efficient Ray Tracing of Volume Data,” ACM Transactions on Graphics, Vol. 9, No. 3, pp. 245-261, 1990.

[14] H. Kye, “Efficient high quality volume visualization using cardinal interpolation”, Journal of Korea Multimedia Society, Vol. 14, No. 8, pp. 981-991, 2011.

[15] E. Choi and B. Shin, “Interlaced Scanning Volume Raycasting”, Journal of Korea Game

Society, Vol. 9, No. 4, pp. 89-96, 2009.
[16] S. Yoo, E. Lee, and B. Shin, “Acceleration of GPU-based Volume Rendering Using Vertex Splitting”, Journal of Korea Game Society, Vol. 12, No. 2, pp. 53-62, 2012.

계 희 원 (Kye, Heewon)

1999년 서울대학교 전산과학 학사
2005년 서울대학교 컴퓨터공학 박사
2007년-현재 한성대학교 정보시스템공학과 조교수

관심분야 : 실시간 가시화, 대용량 영상 처리