GPU 기반 반투과 매체 렌더링의 향상 기법
(Enhancement Techniques for GPU-Based Rendering of Participating Media)

차 득 현 † 이 용 일 † 임 인 성 ††
(Deukhyun Cha) (Yong-il Yi) (Insung Ihm)

요 약 구름, 연기, 가스 등과 같은 반투과 매체(participating media)를 사실적으로 가시화해주기 위해서는 그 내부에서 빛이 진행하는 과정을 물리적으로 시뮬레이션 해주어야 한다. 이 과정은 볼륨 렌더링 방정식이라는 적분 방정식을 통하여 표현할 수 있으나, 이를 정밀하게 푸는 것은 상당한 계산 시간을 요한다. 최근 GPU의 성능 향상에 힘입어, 반투과 매체에 대한 고속의 렌더링 기법들이 소개되고 있으나, 아직도 해결해야할 문제들이 많이 남아있다. 본 논문에서는 기존의 GPU 기반의 반투과 매체 렌더러의 기능/성능 향상을 위하여 적용한 렌더링 기법들을 설명하고, 그러한 노력들이 어떠한 개선 효과를 달성하였는지 분석한다. 이러한 기법들은 추후 각종 디지털콘텐츠 제작에 있어 특수 효과 생성을 위한 고속의 사실적인 볼륨 렌더러 구축에 유용하게 적용될 수 있을 것이다.

키워드 : 반투과 매체, 볼륨 렌더링 방정식, 볼륨 포톤 매핑, GPU 가속, 발광 현상, 뷰 프러스텀 컬링, 필터링

Abstract In order to realistically visualize such participating media as cloud, smoke, and gas, the light transport process must be physically simulated inside the media. While it is known that this process is well described physically through the volume rendering equation, it usually takes a great deal of computation time for obtaining high-precision solutions. Recently, GPU-based, fast rendering methods have been proposed for the realistic simulation of participating media, however, there still remain several problems to be resolved. In this article, we describe our rendering techniques applied to enhance the performances and features of our GPU-assisted participating media renderer, and analyze how such efforts have actually improved the renderer. The presented techniques will be effectively used in volume renderers for creating various digital contents in the special effects industries.

Key words : Participating media, volume rendering equation, volume photon mapping, GPU acceleration, light emitting phenomena, view frustum culling, filtering

1. 서 론

연기, 구름 및 화염, 불, 폭발과 같은 반투과 매체(participating media)에 대한 렌더링은 특수 효과, 게임, 3D 애니메이션 등 다양한 그래픽스 응용에서 요구되는 사실적인 고품질 영상을 생성하는데 있어서 매우 중요한 요소이다. 그러나 반투과 매체에서의 빛의 효과는 물체 표면에서와는 달리 빛이 매우 복잡한 경로를 이동하며 산란(scattering), 흡수(absorption), 발광(emission)하는 과정을 통해 생성되는데, 이는 Kajiya의 논문에서 기술된 다음의 볼륨 렌더링 방정식(volume rendering equation)에 잘 나타나 있다[1].

$$L(x,\overrightarrow{w_o})=\int_0^s e^{-\tau(x,x')}\sigma_a(x')L_e(x')dx'$$

· 이 논문은 2010년도 정부(지식경제부)의 재원으로 SW 컴퓨팅 산업원천기술 개발사업(과제번호: 10035261-2010-01)의 지원을 받아 수행된 연구임

† 학생회원 : 서강대학교 컴퓨터공학과
seaboy7@sogang.ac.kr
asem@sogang.ac.kr
†† 종신회원 : 서강대학교 컴퓨터공학과 교수
ihm@sogang.ac.kr

논문접수 : 2010년 9월 9일
심사완료 : 2010년 10월 13일

정보과학회논문지: 컴퓨팅의 실제 및 레터 제16권 제12호(2010.12)

$$+\int_0^s e^{-\tau(x,x')}\sigma_s(x')\int_{\Omega 4\pi} p(x',\overrightarrow{w_o},\overrightarrow{w_i})L(x',\overrightarrow{w_i})d\overrightarrow{w_i}dx' + e^{-\tau(x,x+s\overrightarrow{w})}L(x+s\overrightarrow{w},\overrightarrow{w_o}) \quad (1)$$

이 볼륨 렌더링 방정식에서 볼 있는 것과 같이 반투과 매체에서의 빛의 효과를 사실적으로 시뮬레이션하기 위해서는 복잡한 계산을 필요로 한다. 그러므로 전역 조명(global illumination) 효과를 포함하는 고품질 영상의 효과적인 생성을 위해 반투과 매체에 대한 렌더링 계산의 가속이 크게 요구되고 있다.

최근 다수 코어(many-core)의 병렬 계산 하드웨어인 GPU(Graphics Processing Unit)를 사용하여 다양한 응용을 가속하기 위한 연구들이 활발하게 이루어지고 있다. GPU는 다면체 모델에 대한 렌더링 계산 파이프라인을 하드웨어로 가속하기 위하여 지속적으로 발전해 왔으나, 보다 다양한 계산을 가능하게 하기 위하여 OpenGL 쉐이더나 Cg 쉐이더 등을 통하여 파이프라인의 일부를 사용자의 프로그램으로 대체하여 계산할 수 있도록 확장되었다[2,3]. 또한 수년전부터 NVIDIA 사에서 제공하는 CUDA(Compute Unified Device Architecture)와 같은 프로그래밍 환경은 GPU의 3D 그래픽스 렌더링 파이프라인 용 인터페이스를 통하지 않고 사용자가 직접 접근할 수 있도록 함으로써, GPU를 범용의 병렬 계산이 가능한 하나의 보조 프로세서(coprocessor)로 활용할 수 있도록 하고 있다[4].

고품질 반투과 매체 렌더링 계산을 범용성이 확장된 GPU를 사용하여 가속화하기 위해서는 볼륨 렌더링 방정식에서 표현하고 있는 각 계산 요소를 효과적으로 병렬화 해주어야 한다. 이를 위해 OpenGL 쉐이더나 Cg 쉐이더 등과 같은 쉐이딩 언어를 사용하여 GPU의 다수 코어 및 고정 렌더링 파이프라인에서 제공하는 하드웨어 기능을 구동할 수 있으나, 보다 효과적인 렌더링 계산을 위해서 CUDA와 같은 범용 프로그래밍 환경을 사용하여 계산의 성능/기능을 향상시키는 것이 더 바람직하다.

본 논문에서는 OpenGL/Cg 기반의 반투과 매체 렌더러에 대하여, CUDA를 사용하여 기능/성능을 개선/확장하기 위해 적용한 렌더링 기법을 설명하고, 이를 통해 얻어진 성능 향상 결과를 실험을 통해 보인다. 본 논문에서 제안하는 렌더링 파이프라인은 Zhou 등[5]의 실시간 기법과 비교해 전처리 시간이 없고 반투과 매체의 형태나 광원의 위치, 성질 등 렌더링 환경에 제약 없이 고화질의 영상 생성을 위한 모든 계산을 수십 초에서 1분 안에 수행하는 특징을 가진다. 특히 특수 효과나 3D 애니메이션과 같은 실제 응용에의 실용적인 적용을 위하여 중요하게 요구되는 다음의 사항에 대한 가속화 연구를 수행하였다.

첫째, 기하물체에 대한 효과적인 처리를 통하여, 반투과 매체와 기하물체가 혼재하는 장면 데이터에 대한 렌더링 계산을 효과적으로 수행 하였다. 반투과 매체만 존재하는 경우에는 배경색을 가지는 빛이 반투과 매체로 입사함을 가정하고 시점방향으로 진행하는 과정을 시뮬레이션 할 수 있다. 그러나 카메라와 물체 사이에 반투과 매체가 존재하는 경우에는 카메라가 바라보는 물체 위 지점에서 출발한 빛이 반투과 매체로 입사하게 되며, 광원으로부터 출발한 빛이 반투과 매체로 입사하는 과정에서 주어진 물체에 의해 가려지는 경우도 발생할 수 있다. 결국 물체가 존재하는 장면에서는 현재 바라보는 방향, 혹은 빛이 진행하는 방향으로 물체가 존재하는지를 검사하고, 해당 물체 지점 및 그 지점에서 반사되어 나오는 빛의 정보를 효과적으로 계산하는 것이 필요하다. 본 논문에서는 최근 광선 추적법에서 대표적인 공간 자료구조로 사용되고 있는 kd-트리를 반투과 매체를 위한 본 GPU 렌더링 모듈에 적용하여 이를 가속화 하였다. 또한 반투과 매체를 통과하는 빛에 대한 산란, 흡수, 반사 효과를 렌더링 모듈에서 계산한 후 실제 물체 표면을 출발하는 빛의 정보와 합성하여 최종 영상을 생성하는 방법을 사용함으로써 물체 표면에 대한 조명 계산 알고리즘에 독립적인 특성을 가지도록 하였다.

둘째, 연기, 구름 또는 가스와 같은 매체와는 달리 화염이나 폭발처럼 스스로 발광하는 반투과 매체에 대한 사실적인 렌더링을 위하여, 볼륨 포톤 매핑(volume photon mapping) 기법을 확장한 물리 기반 발광 시뮬레이션을 GPU로 가속하는 확장 모듈을 개발하였다. 볼륨 데이터가 온도 속성을 가질 때, 이를 활용하여 고온의 매체의 발광현상을 효과적으로 표현할 수 있는데, 본 방법에서는 매 프레임마다 각 격자점의 온도의 함수로, 적절한 색깔을 가지는 포톤을 생성하여 기존의 포톤 집합에 추가함으로써, 특수 효과 제작에 있어 중요한 요소인 화염, 폭발과 같은 현상을 사실적으로 렌더링할 수 있도록 하였다.

셋째, 포톤 매핑 기반의 볼륨 렌더링 방법의 속도와 화질을 개선하기 위하여 간단하면서도 효과적인 최적화 기법들을 GPU 모듈에 적용하였다. 현재 카메라의 뷰-볼륨(view volume)안에 포함되는 영역에 대한 적응적인 렌더링 계산을 통해 전역 조명 계산 속도를 향상시켰고, 종종 렌더링 시간에 제약을 가지는 환경에서 포톤을 충분히 사용하지 못할 경우 발생하는 프레임간의 시간적 앨리어싱(temporal aliasing) 현상을 필터링 기법을 통해 감소시켰다.

본 논문의 구성은 다음과 같다. 2장에서는 먼저 배경이론에 대해 간략하게 설명하고 본 연구의 기여도를 보

인다. 3장에서는 반투과 매체 렌더링을 향상시키는 기법들에 대해 설명하며, 4장에서 이 기법들을 통해 렌더링된 결과들을 분석한다. 5장에서 결론 및 향후 연구 과제를 논의한다.

2. 연구 배경 및 기여도

2.1 볼륨 포톤 매핑 기법

포톤 매핑(photon mapping)은 방대한 계산을 요구하는 전역 조명 계산 문제의 실용적인 대안으로 Jensen이 제시한 방법으로서[6], 사실적 렌더링 분야에서 널리 쓰이고 있다. 이후 Purcell 등이 가속 방법을 제시한 후[7], GPU 상에서도 널리 구현되어 왔다.

볼륨 포톤 매핑은 물체 표면에서뿐만 아니라, 포톤과 장면 공간 내의 반투과 매체와의 작용을 묘사해주기 위한 방법으로서[8], 이러한 빛의 효과를 사실적으로 렌더링하기 위해서는 서론의 식 (1)에서와 같이 매우 복잡한 방정식을 풀어야 한다. 볼륨 렌더링 방정식이 표현하고 있는 반투과 매체에서의 물리 기반 전역 조명 렌더링 계산을 효과적으로 수행하기 위해 개발된 다양한 방법 중, 볼륨 포톤 매핑 기법은 근사화를 최소화하고 계산의 정확도를 높일 수 있는 대표적인 방법으로 널리 사용되고 있다.

볼륨 포톤 매핑 기법에서는 식 (1)을 수치적으로 계산하기 위하여 식 (2)와 같이 이산화 된 형태로 식을 정의한다[6,8]. 카메라가 바라보는 한 방향을 일정한 단위 영역으로 분할하였을 때, n번째 단위 영역을 투과한 빛이 n+1번째 단위 영역을 투과하면서 발생하는 반투과 매체에서의 전역 조명 계산을 나타내고 있다. 단위 영역 안에서는 광원으로부터 직접 입사하는 빛의 계산을 위해 또다른 입사 광휘(radiance) L_i의 계산이 필요하며, 간접적으로 산란되어 들어오거나 나가는 빛의 효과를 계산하기 위하여 방향에 대한 적분을 단위 방향에 대한 합으로 표현하고 있다.

$$L_{n+1}(x,\vec{w}) = \sum_{i=1}^{N} L_i(x,\vec{w_i})p(x,\vec{w_i},\vec{w})\sigma_s(x)\triangle x \qquad (2)$$
$$+\left\{\sum_{p=1}^{n} p(x,\vec{w_p},\vec{w})\frac{\triangle\Phi_p(x,\vec{w_p})}{\frac{4}{3}\pi r^3}\right\}\sigma_s(x)\triangle x$$
$$+e^{-\sigma_t(x)\triangle x}L_n(x+\triangle x\cdot\vec{w},\vec{w})$$

시점 방향에 대하여 분할 된 단위 영역들에 대한 계산과 광원으로부터 직접 입사하는 직접 조명의 계산은 분할 된 단위 영역들에 대한 계산 결과를 순차적으로 누적하는 형태가 되므로, 광선이 진행하는 방향을 일정한 혹은 적응적인 구간으로 샘플링하고 각 구간에서의 값을 누적하는 광선 진행(ray marching) 기법을 사용한다.

2.2 CUDA를 통한 GPU 프로그래밍

CUDA(Compute Unified Device Architecture)는 GPU의 하드웨어 자원을 직접 접근할 수 있도록 함으로써 범용 병렬 계산을 위한 GPU의 활용을 극대화하기 위해 NVIDIA 에서 제안한 솔루션이다[4]. 기존의 그래픽스 파이프라인에서의 프로그래밍을 위해 개발된 어셈블리 언어 수준의 픽셀 쉐이더나 고급 언어 형태의 GLSL[2], Cg[3]와 같은 쉐이딩 언어와는 달리 CUDA는 GPU의 하드웨어에 접근하고 쓰레드를 할당할 수 있는 함수들을 제공한다. 특히 SM(Streaming Multiprocessor) 단위의 상세한 쓰레드 관리가 가능한데 각 SM에 할당되어 있는 레지스터(register), 공용 메모리(shared memory) 등을 고려하여 쓰레드를 효과적으로 생성하거나 프로그램을 최적화할 수 있다. 또한 전역 메모리(global memory)에 자유로운 읽고 쓰기가 가능하며, 연산을 수행하는데 있어서 기존의 그래픽스 파이프라인을 통해 계산하는 과정에서 추가되는 불필요한 연산 없이 최적화된 프로그래밍이 가능하다. 그러나 기존에 그래픽스 파이프라인에서 관리되었던 부분들을 프로그래머가 직접 설정해 주어야 하고 연산에 비해 큰 비중을 차지하는 전역 메모리 접근을 효과적으로 줄이는 것이 중요하므로 최적화된 연산을 위해서는 주어진 GPU의 하드웨어 구조를 명확하게 파악하고 이에 최적화된 연산 과정의 및 쓰레드에 대한 자원 할당이 이루어져야 한다.

본 논문에서 제안하는 효과적인 반투과 매체 렌더링을 위한 확장 알고리즘을 병렬화하고 최적의 성능을 구현하기 위해서는 GPU 자원에 대한 범용 접근이 요구되므로 CUDA를 사용하여 각 모듈을 구현하였다.

2.3 kd-트리 탐색

3차원 공간에서 정의된 다각형(polygon) 데이터를 빠르게 탐색하기 위하여 BVH(Bounding Volume Hierarchy)[9], 8진트리(octree)[10], 그리드(grid)[11], 그리고 kd-트리(kd-tree)[12] 등의 다양한 공간 자료구조가 개발되었다. 특히 kd-트리는 정적인 장면에서 주어진 광선과 교차하는 다각형 위의 지점을 계산하는데 효과적으로 알려져 널리 사용되는 자료구조이다(그림 (1)). kd-트리는 BSP(Binary Space Partition) 트리의 한 종류로써, 공간을 x, y, z 어느 한 축에 대해서 수직인 평면으로 분할하는 것이 특징이다.

GPU의 성능이 발전함에 따라, kd-트리를 GPU에서

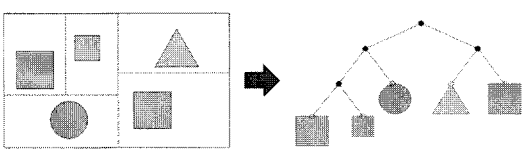


그림 1 kd-트리를 사용한 공간의 분할

구현하여 다양한 응용에 적용하고자 하는 연구가 진행되어 왔다. 그러나 kd-트리를 탐색하는 과정에서 스택(stack)을 사용함으로 인하여 발생하는 빈번한 메모리 접근을 효과적으로 줄이기 위하여 Backtrack[13], Restart[13], Short Stack[14] 등의 다양한 방법이 고안되었다. 본 논문에서는 다각형으로 이루어진 기하물체를 효과적으로 탐색하기 위하여 제한된 스택을 사용하는 Short Stack 알고리즘을 적용하였다. 해당 알고리즘은 처리해야 할 계산은 다소 증가하지만 사용 가능한 최적의 스택 깊이를 사용자가 선택함으로써 메모리의 접근이 성능에 중요한 영향을 미치는 GPU 구조에 적합한 알고리즘이라고 할 수 있다.

본 논문에서는 GPU의 제한된 공유 메모리를 사용하는 Short Stack 알고리즘을 CUDA로 구현하고 주어진 GPU 환경에서 최적의 스택 깊이를 설정하였으며, 해당 모듈을 볼륨 렌더링 모듈과 효과적으로 연동할 수 있도록 하였다.

2.4 본 논문의 기여도

본 논문에서는 CUDA를 통한 GPU 기반 고품질 반투과 매체 렌더링 시스템의 가속에 있어서 실용성을 증대할 수 있는 다음의 확장된 기능 및 알고리즘을 구현하였다.

1) kd-트리 탐색 알고리즘을 이용한 장면에 포함된 기하물체와 반투과 매체의 효과적인 상호작용
2) 사실적인 물리 기반 발광 현상의 가속을 위한 확장된 볼륨 포톤 매핑 알고리즘
3) 카메라 뷰-프러스텀(view frustum) 기반 적응적 단일/다중 산란 계산
4) 필터링 기법을 통한 볼륨 포톤 매핑 기법에서의 애니메이션 과정에서 발생하는 깜빡임 현상 제거 알고리즘

또한 볼륨 포톤 매핑 기반 반투과 매체 렌더링 파이프라인에 대한 확장 모듈로 구현된 위의 각 알고리즘에 대한 성능 평가를 수행하여 렌더링 효과대비 요구되는 추가 연산 시간을 분석하였다. 이를 통하여 본 논문의 GPU 기반 반투과 매체 렌더링 알고리즘이 실제 응용에서 얼마나 효과적으로 적용될 수 있는지를 보임으로써 향후 특수 효과나 3D 애니메이션 제작에 다양하게 응용될 수 있는 렌더링 프레임워크를 제안한다.

3. GPU 기반 반투과 매체 렌더링의 향상 기법들

3.1 kd-트리 기반 다각형 데이터 탐색을 통한 기하물체와의 효과적인 상호작용

본 논문에서 기반으로 하는 반투과 매체 렌더링 파이프라인은 그림 2에 나타나 있다. 전처리 과정에서 렌더링 과정에서 사용하는 자료구조를 사용자가 설정한 변수를 참조하여 정의하고 해당 메모리를 할당하며, 파티

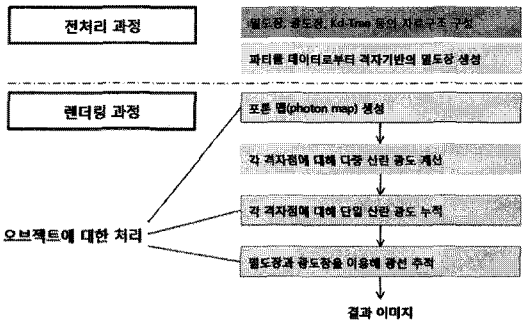


그림 2 반투과 매체 렌더링 파이프라인 및 기하물체 처리 부분

클 형태로 이루어진 반투과 매체 데이터에 대해서는 격자 형태의 밀도장(density field)으로 변환한다. 렌더링 과정에서는 볼륨 포톤을 추적하여 포톤 맵을 생성하고, 이를 사용하여 광휘 계산의 가속을 위해 격자 형태를 가지는 광도장(radiance field)에 다중 산란 광휘 정보를 캐싱한다. 그 이후에는 단일 산란 정보를 광도장에 누적하고 광휘 정보가 캐싱된 광도장과 밀도 정보를 가지고 있는 밀도장을 사용하여 각 픽셀에서 출발한 광선에 대한 광선 진행 계산을 수행함으로써 최종적으로 전역 조명 효과가 포함된 결과 이미지를 얻게 된다. 이 모든 과정이 GPU에서 이루어지며 필요한 장면 정보를 CPU로부터 읽어 들인 이후에는 GPU 메모리 접근만을 통해 계산이 이루어진다. 기반 렌더링 파이프라인을 구성하는 알고리즘에 대한 상세한 설명은 2008년 Son[15] 및 2009년 Cha 등[16]이 발표한 논문을 참고할 수 있다.

주어진 장면이 기하물체를 포함하게 되면 렌더링 파이프라인에서 포톤 맵 생성, 단일 산란 계산, 광선 추적에 대한 세 가지 모듈의 계산이 영향을 받게 된다(그림 2). 서론에서 설명하였던 것과 같이 기하물체에 대한 조명 계산은 독립적으로 수행된 후 합성될 수 있으므로 위의 세 가지 모듈에서는 주어진 광선이 기하물체와 만나는지와 만나게 되는 경우 해당 지점에서의 기하 정보 및 물체의 속성 정보에 대한 계산이 필요하다. 주어진 광선과 기하물체와의 교차 계산을 kd-트리를 사용하여 효과적으로 계산하게 되는데, 이때 각 모듈에서 요구되는 기하 정보 계산은 다음과 같다.

3.1.1 포톤 맵 생성 모듈

볼륨 포톤 매핑 기법에서는 다중 산란 정보를 표현하기 위해 광원에서 출발한 포톤들이 반투과 매체 안에서 산란하며 상호작용하는 지점들을 포톤 맵에 저장한다. 이 과정에서 산란 방향과 산란 거리를 확률적으로 계산하면서 포톤을 추적하게 되는데, 기하물체가 현재 추적되는 포톤 방향에 대한 산란 거리 안에 존재하는 경우에는 주어진 산란 거리만큼 진행하지 못하고 기하물체

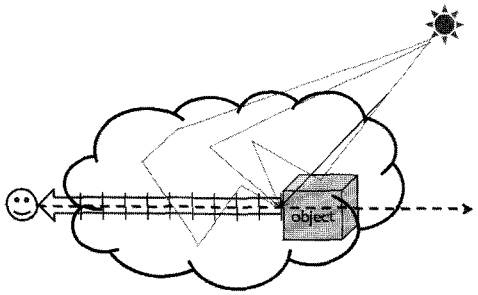


그림 3 포톤 산란 과정에서의 기하물체와의 상호작용

와의 상호작용을 통해 새로운 산란 방향 및 산란 거리를 계산해야 한다. 이를 위해 포톤 산란 과정에서 발생하는 모든 진행 구간 안에 기하물체가 존재하는지 검사하기 위해 광선-물체 교차 계산을 수행하고, 기하물체가 존재하는 경우에 포톤이 해당 물체와 만나는 지점에서의 기하 정보 및 BRDF와 같은 표면 속성 정보를 계산하게 된다. 이 정보를 통해 포톤이 계속 진행 할지의 여부와 산란 방향 및 거리를 확률적으로 계산한다(그림 3). 이 과정에서 물체와 상호작용하는 지점에서의 정보는 포톤 맵에 저장하지 않는다.

3.1.2 단일 산란 모듈

현재 샘플링된 지점에 대하여 광원으로부터 직접 입사하는 광휘 정보를 얻기 위해서는 광원으로부터 출발한 광휘가 반투과 매체를 투과하면서 얼마나 감쇠(attenuation)될 것인가를 계산해야 한다. 렌더링 방정식에서 정의된 단일 산란 부분이 나타내고 있는 내용이 바로 이것인데, 앞서 설명한 바와 같이 광도장에 이를 캐싱하기 위해 모든 격자 지점에서 광원으로의 광선 추적이 일어나게 된다. 감쇠 계산은 각 추적되는 광선을 균일한 혹은 적응적인 구간으로 샘플링한 후 각 샘플링 지점에서의 값을 누적함으로써 이루어지게 된다(그림 4). 그러나 주어진 격자와 광원 사이에 기하물체가 존재한다면 물체의 불투명도에 따라 추가 감쇠가 이루어져야 한다. 그러므로 모든 격자에서 출발하는 광선은 먼저 물체와의 교차 계산을 수행하고 물체와 교차하는 경우에는 해당 지점에서의 불투명도를 얻어오게 되며 불투명도가 100%인 경우에는 반투과 매체에 의한 감쇠 계산을 수행하지 않는다.

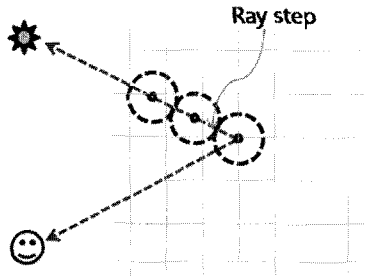


그림 4 광도장에서의 단일 산란 계산

3.1.3 광선 진행 모듈

광도장에 계산된 단일 산란과 다중 산란 값을 사용하여 반투과 매체를 지나 각 픽셀로 입사하는 최종 광휘를 계산하기 위한 광선 진행이 수행된다. 픽셀로부터 출발하는 광선을 균일한 혹은 적응적인 구간으로 샘플링한 후, 각 샘플링 구간에서의 단일/다중 산란 값을 계산하여 누적하며 각 누적된 값이 픽셀과 해당 지점 사이의 반투과 매체를 투과하면서 감쇠되는 정도를 밀도장의 값을 이용하여 함께 계산한다. 이 과정에서 광선 진행 방향으로 기하물체가 존재하는 경우에 기하물체 뒤쪽에서 입사하는 광휘는 기하물체의 불투명도에 영향을 받아 감쇠가 일어난다. 기하물체의 불투명도가 100%인 경우에는 뒤쪽에서 입사하는 광휘가 영향을 미치지 못하므로 광선 진행을 종료한다. 이를 위하여 각 광선 진행 방향으로 kd-트리를 이용한 광선-물체 교차 계산을 수행하게 되며 광선이 물체와 교차하는 경우 물체의 불투명도에 따라 감쇠 계산을 수행한다. 광선 진행을 통하여 각 픽셀에 최종적으로 반투과 매체를 통해 산란되어 들어오는 색깔 정보 및 반투과 매체로 인해 발생하는 불투명도를 계산하여 독립적으로 생성된 기하물체를 포함하는 배경 영상과 합성하게 된다.

3.2 온도 데이터를 이용한 발광 현상 구현

반투과 매체에 대한 렌더링에서 구름, 연기와 같은 형태뿐만 아니라 불, 화염과 같이 발광 효과를 포함하는 형태에 대한 효과적인 렌더링 기법이 크게 요구된다. 본 논문에서는 볼륨 포톤 매핑 기법을 확장하여 사실적인 발광 효과를 생성하는 Kang 등의 연구[17]를 볼륨 포톤 매핑에 기반한 파이프라인에 통합하여 GPU에서 가속 하였다.

불과 화염 등의 효과는 밀도가 존재하는 지점이 포함하고 있는 온도 정보를 적절한 색깔을 가지는 광휘 정보로 표현함으로써 얻을 수 있다. 기존의 볼륨 포톤 매핑 기법에서는 광원에서 출발하는 광휘를 계산하므로 광원이 가지고 있는 빛의 세기에 따라 포톤의 밝기가 결정되게 된다. 그러나 불과 화염 등은 스스로 연료를 태워 온도가 상승하고 이에 의해 빛 에너지가 생성되므로 발광 포톤맵에서는 각 지점에서의 온도에 따라 적절한 색깔을 가지는 적절한 개수의 포톤을 생성하게 된다. 그림 5는 온도(T)가 주어졌을 때, 파장(λ)을 따라 나타나는 복사(radiation)의 분포를 보여주는데, 이를 식 (3)과 같은 함수 형태로 나타낼 수 있다. 이 함수를 파장 구간 $\lambda_{\min}$과 $\lambda_{\max}$사이에서 적분함으로써, 온도에 따른 전체 발광 세기 Φ_{tep}를 계산할 수 있다.

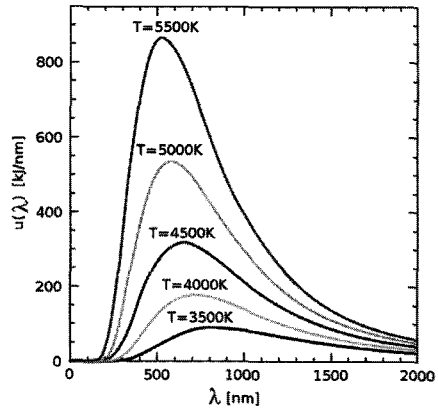


그림 5 파장에 대한 복사 분포도

$$\Phi_{tep}(T, \lambda_{\min}, \lambda_{\max}) = \pi \int_{\lambda_{\min}}^{\lambda_{\max}} E(\lambda, T) d\lambda \qquad (3)$$

물리 기반 시뮬레이션 혹은 사용자 입력으로 생성될 수 있는 온도 데이터는 3차원 격자에 이산화 된 형태로 표현될 수 있으며 발광 포톤맵 역시 3차원 격자 상에서 효과적으로 생성할 수 있다. 각 격자 안에서 주어진 온도에 따른 초기 포톤을 생성하고 발광 포톤맵에 저장한다. 저장된 초기 포톤들은 임의의 방향을 부여받아 밀도장안에서 산란하게 된다.

이와 같이 동작하는 발광 포톤맵 모듈을 CUDA를 사용하여 GPU에서 구현하고 반투과 매체 렌더링 파이프라인의 볼륨 포톤 매핑 모듈 부분에 추가하여 발광 현상을 효과적으로 생성할 수 있다. 초기 발광 포톤들이 산란되는 과정에서 기존의 볼륨 포톤 매핑 기법과 다른 점은 포톤들이 가지고 있는 빛의 세기가 각각 다르다는 것이다. 광원에서 출발한 포톤들은 광원의 밝기를 추적한 포톤 개수만큼 나누어 가지고 있으나 발광 포톤은 각 지점에서의 온도에 따라 그 값이 결정되기 때문이다. 그러나 산란 과정에서 기하물체와의 상호작용을 고려하는 부분은 볼륨 포톤맵의 산란 부분과 동일하다. 이렇게 산란된 발광 포톤맵은 볼륨 포톤맵과 함께 광도장에 계산 된 후 동일한 반투과 매체 렌더링 과정을 수행하여 최종적인 영상을 생성하게 된다.

3.3 뷰-프러스텀 기반의 적응적 단일/다중 산란

반투과 매체 안을 카메라가 이동하면서 바라보는 애니메이션과 같이 장면에서 밀도가 차지하는 크기에 비하여 카메라가 보고 있는 부분이 작은 경우에 단일/다중 산란을 계산하는 과정에서 불필요한 계산이 일어날 가능성이 높다. 광선 진행 과정에서 접근이 일어나지 않는 많은 광도장 지점에서 단일 산란 계산을 위한 광선 추적 계산이 일어나거나 다중 산란 및 발광 효과를 위한 포톤 저장과 수집 계산이 일어나야 한다.

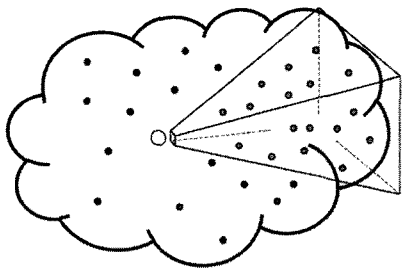


그림 6 뷰-프러스텀 기반의 적응적 포톤맵 생성. 붉은 색 점들은 산란 과정에서 상호작용이 일어났으나 저장하지 않은 지점들이다.

이러한 비효율을 제거하기 위하여 뷰-프러스텀 내부에 위치하는 광도장 격자에 대해서만 광원들에 대한 광선 추적 계산을 수행하고, 포톤 산란 계산 과정에서 뷰-프러스텀에 위치하는 포톤과 반투과 물체의 상호작용 지점에 대해서만 포톤맵에 정보를 저장한다(그림 6). 이때 적응적 광도장 계산을 위한 새로운 뷰-프러스텀을 설정할 필요성이 있다. 포톤 수집 계산에서 광도장 격자에 대하여 수집 반경내 있는 포톤들이 영향을 미치게 되므로 적응적 포톤맵을 생성할 뷰-프러스텀은 기존에 설정된 뷰-프러스텀의 각 평면에 대하여 수집 반경만큼 확대된 형태를 가진다. 또한 사용자가 설정하거나 전체 장면의 크기에 의해 설정된 뷰-프러스텀의 원평면(far plane)에 대해서 밀도장에 의한 불투명도 값 누적에 따라 광도 누적 계산이 필요한 실질적인 원평면이 계산될 수 있다. 본 논문에서는 각 픽셀에 대한 불투명도 값을 계산한 후, 불투명도가 100%가 되는 거리 중 가장 먼 거리 값과 설정된 원평면거리 중 작은 값에 새로운 원평면을 설정한다. 이렇게 설정된 뷰-프러스텀 안에 포함되는 광도장 격자에 대해서 단일/다중 산란 계산을 수행함으로써 장면에 분포하는 반투과 매체의 크기에 비해 뷰-프러스텀의 크기가 작은 경우에 적은 계산 비용으로 동일한 영상을 생성할 수 있으며 포톤 저장에 사용되는 메모리의 크기를 효과적으로 감소할 수 있다.

3.4 애니메이션의 깜빡임 현상 제거

포톤 매핑 기반의 렌더링 방법에서 생기는 문제점 중의 하나는, 프레임간의 깜빡임(flickering) 현상이다. 포톤 매핑 기법에서 포톤 산란을 계산하는 과정에서 포톤의 흡수 여부, 추적 방향 그리고 진행 거리의 계산에 확률적인 요소가 포함되므로 장면에 포함된 반투과 매체나 기하물체가 애니메이션되는 경우에는 생성되는 포톤맵의 프레임간 연속성이 떨어질 수 있다. 이는 렌더링된 영상에서 특정 지점들이 깜빡이는 문제로 나타나게 된다. 확률적으로 충분한 개수의 포톤을 생성하거나 다양한 난수 생성 기법들을 통해 이러한 현상을 줄일 수 있

다. 그러나 깜빡임을 완전히 제거할 수 있을 정도의 포톤을 사용하기에는 메모리 등의 자원이나 렌더링 시간의 제약으로 인하여 사용할 수 있는 포톤 개수에 한계가 있을 수 있고 이러한 경우에 난수 생성 기법들로도 문제를 완전하게 해결하기는 어렵다. Meyer 등[18]의 연구에서는 물체 표면에서의 간접 조명을 계산하는 과정에서 충분하지 못한 샘플링으로 인해 발생하는 노이즈를 줄이기 위하여 전체 애니메이션 이미지를 생성한 후 적절한 개수의 기본 함수(basis function)를 이용하는 이미지 기반의 필터링 기법을 사용하였다. 물체 표면에서의 간접 조명을 계산하면서 발생하는 노이즈에 비해 반투과 매체에서의 포톤 매핑 기법에서는 보다 높은 빈도(high frequency)의 노이즈일 가능성이 높으며, 광선이 추적되는 과정에서 각 샘플링 지점에서의 다중 산란 정보가 누적되므로 이미지 기반 방법으로 문제가 되는 부분을 찾기 어렵다.

본 논문에서는 광도장의 대응되는 격자에 대한 필터링을 통해 적은 개수의 포톤을 사용하면서 프레임 간 연속성을 증가하는 방법을 제안한다. 이를 위해 그림 7과 같은 형태로 필터를 사용하여 주어진 렌더링 프레임과 시간적으로 연속성이 요구되는 주변 프레임간의 다중 산란 광도를 스무딩(smoothing)한다. 본 논문에서는 주어진 프레임을 중심으로 앞, 뒤 프레임에 대하여 (0.25, 0.5, 0.25)의 가중치를 가지는 3개 프레임 필터링과, (0.1, 0.2, 0.4, 0.2, 0.1)의 가중치를 가지는 5개 프레임 필터링을 사용하였다. 5개 프레임의 경우 보다 넓은 시간 구간을 필터링함으로써 더 큰 스무딩 효과를 얻을 수 있다.

이를 위하여 각 필터링 방법에서 사용하고자 하는 개수만큼의 프레임에 대한 다중 산란 계산이 이루어져야 한다. 그러나 한번 계산되어 광도장에 저장된 다중 산란 결과는 다시 계산되지 않고 필요한 프레임만큼 유지된다. 이를 위하여 필터링에 사용되는 프레임 수만큼의 광도장을 렌더링하는 과정에서 유지해야 하므로 추가되는 메모리 공간이 필요하다. 최신의 GPU에서 지원되고 있는 것과 같이 메모리가 충분한 경우에는 필요한 광도장들을 GPU 메모리에 할당하여 최적의 성능을 구현할 수 있으나, 그렇지 않은 경우에도 약간의 추가 비용과 함께 CPU 메모리를 사용할 수 있다. 또한 전체 반투과 매체 렌더링 파이프라인에서 다중 산란 계산 모듈을 분리하여 미리 계산된 정보를 사용할 수 있도록 하여야 한다. 다음 장에서 필터링을 사용하지 않고 포톤의 개수를 늘려 렌더링을 수행한 경우와 필터링을 통해 렌더링을 수행한 경우에 대한 결과 비교를 통해 제안하는 필터링 기법의 성능을 분석하고 제한된 렌더링 환경에서 효과적으로 이용할 수 있도록 한다.

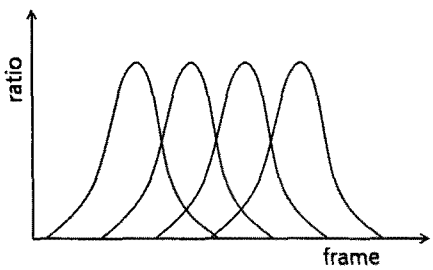


그림 7 애니메이션 프레임간 필터링 함수

4. 실험 결과 및 분석

이 장에서는 본 논문에서 제안한 GPU기반 확장 모듈 및 최적화 방법에 따라 렌더링 된 결과를 보여준다. 모든 실험의 성능 측정은, 3.0 GHz의 Intel Core 2 Duo CPU와 4GByte의 메모리, NVIDIA GeForce GTX 280 GPU를 장착한 컴퓨터에서 수행되었다.

4.1 kd-트리 기반 다각형 데이터 탐색

그림 8은 kd-트리 기반 다각형 탐색을 통해 기하물체가 포함된 장면에서의 반투과 매체 렌더링 결과를 보여준다. 세 번째 행의 렌더링 결과에서 단일 산란에서의

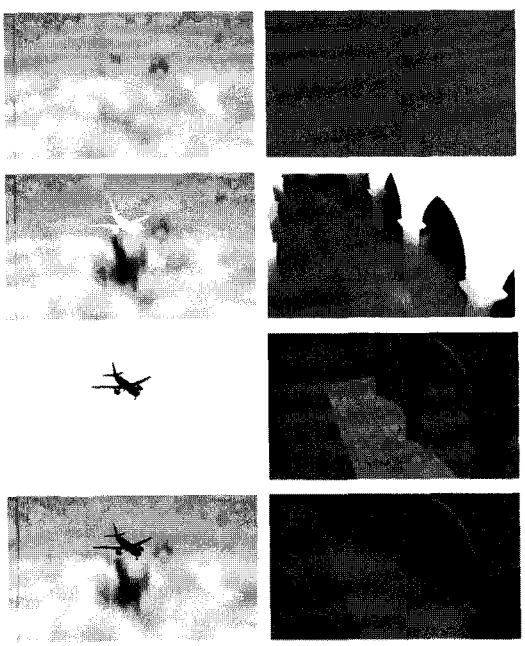


그림 8 기하물체가 포함된 구름(좌) 및 안개(우) 장면에 대한 렌더링의 예. 각 행은 위에서부터 순서대로 기하물체가 없는 경우, 기하물체가 포함된 반투과 매체, 기하물체의 렌더링 결과를 나타내며 마지막 행은 최종 합성 이미지를 보여주고 있다.

그림자 효과, 광선 진행 과정에서의 기하물체 처리 모습이 반투과 매체의 불투명도 및 색깔 정보로 표현된 것을 확인할 수 있다. 다중 산란에서의 포톤 추적 과정에서도 기하물체와의 상호작용이 적용되어 기하물체 표면의 재질 및 색깔 정보가 포톤 추적 과정에서 반영 되었다.

표 1은 그림 8의 렌더링 결과에 대하여 기하물체와의 상호작용이 포함된 경우의 렌더링 시간 증가를 보여주고 있다. 구름 장면이 포함하고 있는 기하물체는 3,592개의 삼각형으로 이루어져 있으며 안개 장면이 포함하고 있는 기하물체는 66,454개의 삼각형으로 이루어져 있다. 안개 장면이 보다 많은 수의 삼각형으로 이루어진 기하물체를 포함하고 있어, 렌더링 시간이 0.2% 증가한

표 1 기하물체 정보를 이용하여 렌더링했을 때의 계산 시간 비교(단위: 초)

	렌더링 시간	kd-트리 생성 시간	전체 시간
구름	5.82	-	5.82
구름+기하물체	5.83	0.33	6.16
안개	2.36	-	2.36
안개+기하물체	2.78	7.33	10.11

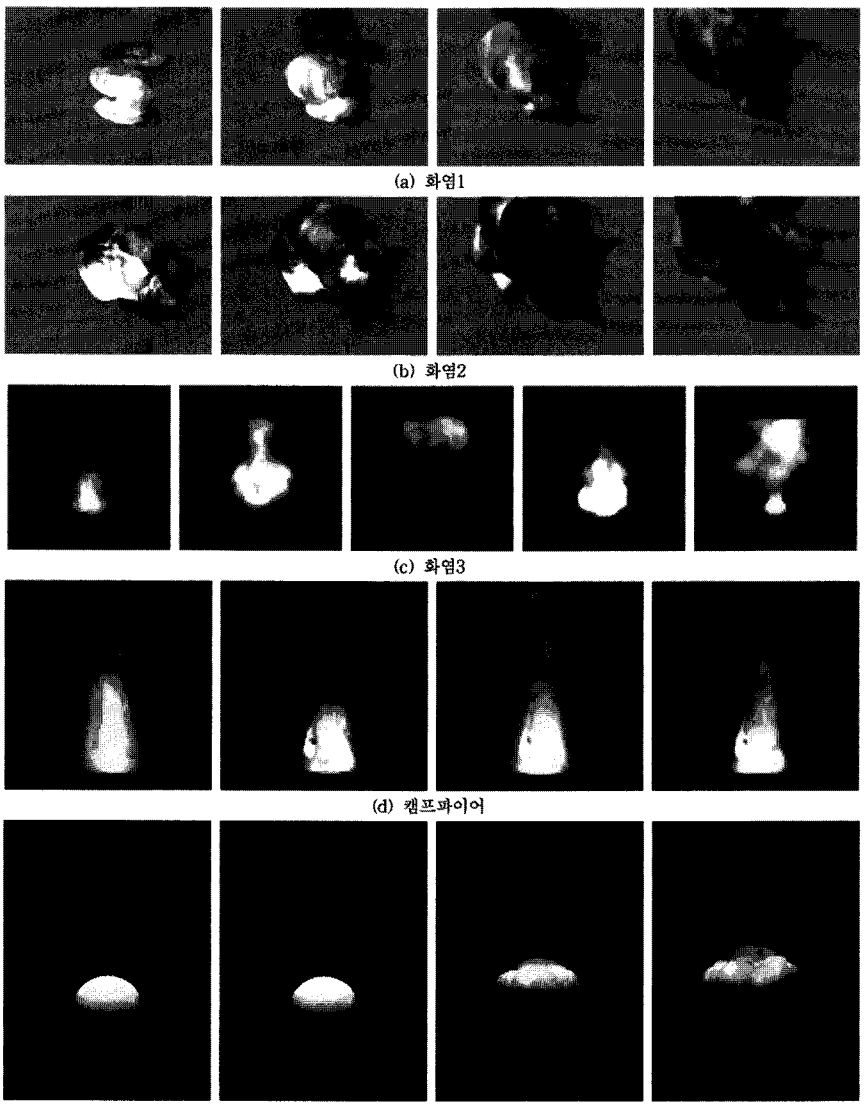


그림 9 GPU 기반의 발광 포톤맵을 사용한 발광 효과 렌더링의 예. 각 행은 주어진 장면에 대하여 선택된 애니메이션 프레임의 렌더링 결과를 나타낸다.

구름 장면에 비해 17.8%의 렌더링 시간 증가를 보인 것을 알 수 있다. 또한 다각형 개수가 증가할수록 kd-트리 생성 시간이 증가하는 것을 알 수 있다. 트리 탐색 성능은 매우 우수하지만 트리 생성에 드는 비용이 비교적 크다는 것이 kd-트리의 단점이라고 할 수 있다. 삼각형의 개수 증가에 따라 kd-트리 생성 시간은 O(N log N)의 복잡도로 더 늘어나게 되는데[19], 정적인 기하물체의 경우 전처리를 통해 kd-트리를 한번만 생성해 놓을 수 있다. 또한 동적인 기하물체를 포함하는 경우에도 일반적인 장면에서 수십초의 시간에 최종 영상을 생성할 수 있는 본 논문의 GPU 기반 오프라인 반투과 매체 렌더링 파이프라인에서 기하물체가 포함된 장면 생성을 위해 허용 가능한 비용이라고 할 수 있다.

4.2 발광 포톤맵을 이용한 발광 효과 생성

그림 9는 발광 포톤맵을 이용해 발광 효과를 생성한 결과 영상들이다. 데이터가 가지고 있는 온도 정보를 이용하여 사실적인 발광 효과를 표현하고 있는 것을 확인할 수 있다. 표 2는 각 장면에서 사용된 렌더링 정보 및 CPU와의 시간 비교를 보여주고 있다. 격자 해상도는 밀도장 및 광도장에 사용된 격자의 해상도를 의미하며, 발광 포톤 개수는 발광을 위해 생성하고 산란되어 발광 포톤맵에 저장된 포톤의 개수를 의미한다. CPU 렌더링 시간은 발광 포톤맵을 제안한 기존의 논문[17]의 내용을 구현한 렌더러를 사용하여 렌더링한 결과이고, GPU 렌더링 시간은 해당 내용을 GPU를 통한 가속 모듈로 구현한 본 논문의 렌더러를 사용하여 렌더링한 결과이다. 다수 코어를 가지는 GPU의 병렬 계산 환경에서 효과적으로 가속되어 주어진 5개의 장면에 대하여 CPU에서의 결과에 비해 평균 약 357배의 성능 향상을 보이는 것을 확인할 수 있다. 단 화염3 장면과 같이 격자 해상도가 작아 계산양이 적은 경우에는 상대적으로 성능 향상 정도가 작아 9.1배 정도의 향상이 있는 것을 확인할 수 있으며, 격자 해상도가 크고 사용된 포톤의 개수가 많은 화염2의 경우에는 890.2배 정도의 상대적으로 큰 성능 향상을 보이는 것을 확인할 수 있다. 또한 발광 포톤의 개수가 크게 증가하는 폭발 장면의 경우에 GPU에서의 렌더링 시간도 크게 증가하지만, CPU의 결과에 비하여 여전히 약 54.2배의 성능 향상을 보이는 것을 알 수 있다. 그러므로 온도 데이터에 대한 사실적인 발광 현상을 수십초 내의 시간에 빠르게 생성할 수 있으며, 계산양의 증가에 따라 그 성능 향상 정도가 증가하므로 계산이 복잡한 실제 영상 제작에서 효과적으로 사용될 수 있다.

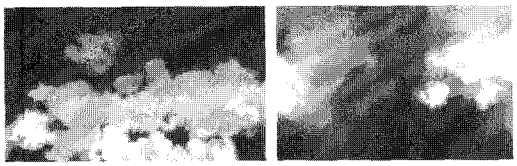
4.3 뷰-프러스텀 기반의 적응적 단일/다중 산란

본 논문에서는 뷰-프러스텀 기반의 적응적 단일/다중 산란 기법의 적용 결과 중, 렌더링 파이프라인에서 가장 큰 계산 시간이 요구되는 다중 산란 부분에 대한 결과를 보인다. 그림 11은 그림 10에서 볼 수 있는 것과 같이 구름 장면을 카메라가 이동하는 경우에 대하여 처리되는 포톤의 개수를 그래프로 나타내고 있다. 그림 10(a)와 같이 뷰-프러스텀 안에 구름의 많은 부분이 포함되는 경우와 그림 10(b)와 같이 뷰-프러스텀안에 구름의 일부분이 포함되는 카메라 애니메이션에 대한 결과로써, 뷰-프러스텀이 가장 많은 구름 영역을 포함할 때 최대 217,794개의 포톤을 저장하여 처리하고 있으며 가장 적은 구름 영역을 포함할 때 최소 2,313개의 포톤을 처리하고 있다. 총 600프레임으로 구성되어 있는 본 애니메이션 장면의 렌더링에 대하여 적응적인 포톤맵을 사용하지 않을 경우 매 장면 마다 약 250,000개의 포톤을 처리해야 하는데, 적응적인 포톤맵을 사용하게 되면 평균적으로 약 165,870개의 줄어든 포톤을 처리할 수 있다. 이는 전체적으로 28.5%의 렌더링 시간의 감소를 가져온다. 여기에 뷰-프러스텀 안에 들어오는 지점에 대해서 단일 산란을 적응적으로 계산하게 되면 뷰-프러스텀을 적용하지 않은 경우에 비해 34.1%의 렌더링 시간 감소를 가져온다(표 3). 그러나 이러한 단일/다중 산란 계산 비용의 감소에도, 밀도의 계산이나 포톤의 추적 계산은 전체 반투과 매체 영역을 대상으로 수행되므로 동일한 화질의 영상을 생성할 수 있게 된다.

이와 같이 대부분의 영상 제작에서 카메라나 장면의 애니메이션이 존재하고, 특히 구름, 연기, 안개 등의 반투과 매체의 경우에는 매우 넓은 영역에 대하여 모델링되는 경우가 많으므로 본 논문에서 제안하는 뷰-프러스텀 기반 적응적 단일/다중 산란 기법은 효과적으로 활용될 수 있다.

표 2 발광 현상 렌더링에 걸린 시간 (단위: 초)

	이미지 해상도	격자 해상도	발광 포톤 개수	렌더링 시간 (CPU)	렌더링 시간 (GPU)
화염 1	640×480	300×250×200	39,000	4155.57	6.20
화염 2	640×480	300×250×200	39,000	7148.25	8.03
화염 3	640×480	60×60×60	23,000	33.00	3.61
캠프 파이어	512×512	70×70×80	83,000	1291.39	7.78
폭발	600×800	80×100×80	444,000	1417.86	26.15



(a) 줌-아웃 (b) 줌-인

그림 10 뷰-프러스텀에 포함되는 장면의 크기가 다른 카메라 애니메이션 프레임에 대한 렌더링 결과

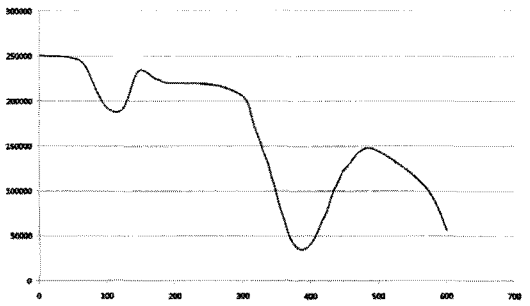


그림 11 뷰-프러스텀을 이용한 적응적 다중 산란에서의 포톤 사용 개수 변화. 가로축은 애니메이션 프레임, 세로축은 포톤의 개수를 나타낸다.

표 3 적응적 단일/다중 산란을 통한 렌더링 시간 비교 (단위: 초)

	기존의 방법	적응적 다중 산란	적응적 다중/단일 산란
렌더링 시간	26.10	18.66	17.19

4.4 필터링을 사용한 프레임간 깜빡임 현상 제거

3.4절에서 설명한 것과 같이 충분하지 못한 포톤으로 인하여 발생하는 애니메이션 프레임간 깜빡임을 제거하기 위하여 인접한 프레임들의 다중 산란 정보를 계산한 광도장을 생성하고 정의된 가중치를 사용하여 이들 광도장을 필터링한 다음 렌더링을 수행한다. 표 4에서는 그림 12에서와 같은 연기 애니메이션 장면에 대하여 필터링을 수행하지 않고 포톤을 늘린 경우와 각각 3개 프레임과 5개 프레임에 대한 필터링을 사용한 경우에 렌더링 시간을 비교하고 있다. 필터링에 사용된 가중치는 3.4절에서 설명한 것과 동일하게 적용되었다.

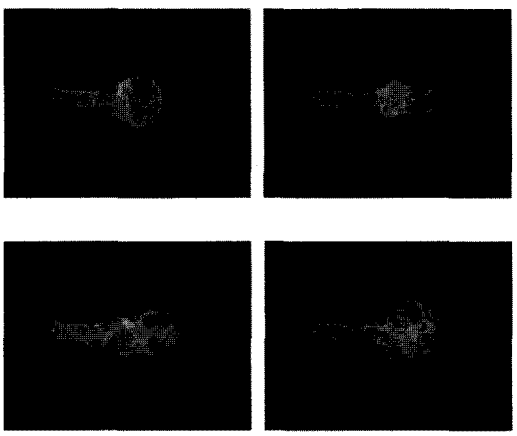


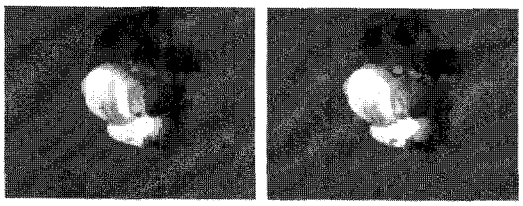
그림 12 깜빡임 테스트를 위한 연기 애니메이션

표 4 반짝임을 테스트하기 위해 포톤의 개수에 따라 렌더링에 걸린 시간 (단위: 초)

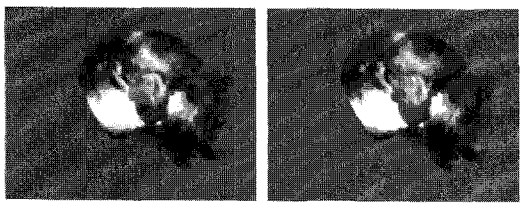
	필터링하지 않음	3개 프레임을 필터링	5개 프레임을 필터링
포톤 10,000	3.06	4.75(1.55x)	4.80(1.57x)
포톤 20,000	3.26	4.98(1.53x)	5.10(1.56x)
포톤 50,000	3.68	5.46(1.46x)	5.70(1.55x)
포톤 100,000	4.45	6.37(1.43x)	6.63(1.49x)
포톤 500,000	10.82	14.20(1.31x)	14.69(1.36x)
포톤 1,000,000	18.92	23.88(1.26x)	24.55(1.30x)

증가하는 포톤 개수에 대하여 3개의 필터링을 사용한 경우에는 평균 42% 정도의 렌더링 시간 증가가 있었으며, 5개의 필터링을 사용한 경우에는 평균 47% 정도의 렌더링 시간 증가가 있음을 확인할 수 있다. 또한 사용되는 포톤의 개수가 증가할수록 렌더링 시간 차이가 줄어드는 것을 볼 수 있다. 이때 중요한 것은 포톤의 부족에 의한 깜빡임이 언제부터 제거 되는가 하는 것인데, 주어진 장면에 대하여 필터링을 하지 않은 경우는 100,000개, 3개 프레임을 필터링 한 경우에는 20,000개, 5개 프레임을 필터링 한 경우에는 10,000개의 포톤을 사용한 시점부터 깜빡임이 보이지 않는 것을 시각적으로 확인할 수 있었다. 각 경우에 대한 렌더링 시간은 4.45초, 4.98초, 4.80초로써 필터링을 하지 않고 많은 개수의 포톤을 사용한 경우에 오히려 약간 빠른 시간에 렌더링을 수행 할수 있는 것을 볼 수 있다. 그러나 장면의 크기가 커지고 사용되는 포톤의 개수가 증가하게 되면 깜빡임을 제거할 수 있는 정도의 포톤을 사용하는데 한계가 발생할 가능성이 커지므로 필터링을 사용한 깜빡임 제거 기법이 의미를 가진다고 하겠다. 주어진 장면에서는 3개 프레임 필터링의 경우 20%의 포톤을, 5개 프레임 필터링의 경우 10%의 포톤만을 사용하고도 깜빡임이 제거된 애니메이션 영상을 얻을 수 있었으나 장면의 특성에 따라 변화는 있을 수 있다.

충분치 못한 포톤으로 인한 깜빡임 현상은 발광 포톤 맵을 사용하는 경우에 보다 심각하게 발생할 수 있는데, 폭발 등의 격렬한 효과에 의해 프레임간 밀도장의 움직임이 클 수 있고 온도 변화가 심해 생성되는 초기 포톤들의 정보 역시 크게 다를 수 있기 때문이다. 이러한 경우 필터링을 사용하여 깜빡임 현상을 효과적으로 제거할 수 있다. 그림 13은 화염1, 화염2 장면에 대하여 필터링을 수행하지 않은 경우와 3개 프레임을 사용하여 필터링을 수행한 경우에 대한 이미지를 보여주고 있다. 광도장의 필터링으로 인하여 약간의 스무딩 효과가 발생할 수 있으나 거의 동일한 영상을 얻을 수 있는 것을 확인할 수 있다. 각 영상에 대한 필터링은 3개 프레임에 대해 0.25, 0.5, 0.25의 가중치를 가지고 수행되었다. 화



(a) 화염1



(b) 화염2

그림 13 발광 장면에 대한 필터링 효과 비교. 필터링의 수행하지 않은 영상(좌열)과 3개 프레임을 사용한 필터링 수행 영상(우열)의 차이가 거의 없음을 확인할 수 있다.

염1 장면의 필터링을 적용하지 않은 프레임당 평균 렌더링 시간은 6.62초이고, 필터링을 적용했을 때에는 8.04초이다. 화염2 장면의 경우에는 각각 5.30초와 6.65초의 평균 렌더링 시간을 가진다. 두 장면이 각각 21%와 25%의 평균 렌더링 시간 증가를 통해 화염의 모양을 잘 보존하면서 애니메이션 프레임간의 깜빡임 현상을 크게 감소할 수 있다.

5. 결 론

본 논문에서는 GPU에서 가속되는 포톤 매핑 기반의 반투과 매체 렌더링 파이프라인을 확장하여, 실제 응용에서 크게 요구되는 기하 물체와의 상호작용, 발광 효과 생성, 뷰-프러스텀 기반 적응적 단일/다중 산란을 통한 렌더링 성능 향상, 필터링을 통한 제한된 포톤 개수 사용에 대한 깜빡임 현상 제거의 네 가지 GPU 기반 알고리즘을 제안하고 NVIDIA의 CUDA를 사용하여 구현하였다. 광선 추적 기반 기법에서 널리 사용되며 최근 GPU에서 효과적으로 구현되고 있는 kd-트리 구조를 사용하여 반투과 매체 렌더링 파이프라인 안에서 효과적인 기하 물체와의 상호작용을 구현하였으며, 발광 포톤맵을 사용한 발광 효과의 생성 알고리즘을 GPU에서 빠르게 구현하고 그 결과를 보였다. 뷰-프러스텀 기반 적응적 단일/다중 산란 및 필터링을 사용하여 메모리나 렌더링 시간 등이 제한적인 상황에서 보다 효과적으로 고품질의 영상을 생성할 수 있도록 하였으며 이를 분석하여 실제 응용의 적용에 참고할 수 있도록 하였다.

향후 연구 과제로 뷰-프러스텀뿐만 아니라 데이터의 특성을 반영하여 밀도장이나 광도장을 적응적으로 생성하고 이를 GPU에서 효과적으로 관리할 수 있는 자료구조를 개발하려고 한다. 또한 다양한 필터링 함수를 사용하여 보다 효율적으로 제한적인 상황에서의 포톤 매핑 기법의 깜빡임 현상을 제거할 수 있으리라 생각한다. 끝으로 기하 물체와의 상호작용에 사용한 kd-트리 구조의 생성을 GPU에서 수행하는 Zhou등의 방법[20]을 적용하거나 탐색 성능은 떨어지나 생성 속도가 빠른 BVH등의 자료구조를 혼용하는 방법을 적용할 수 있으리라 생각한다.

참 고 문 헌

[1] J. T. Kajiya and B. P. Herzen, "Ray tracing volume densities," *In Proc. of ACM SIGGRAPH'84*, pp.165-174, 1984.

[2] OpenGL, *OpenGL Shading Language*, http://www.opengl.org/documentation/glsl/, 2009.

[3] NVIDIA, *Cg Toolkit User's Manual*, 2009.

[4] NVIDIA, *CUDA Programming Guide 2.3 Manual*, 2009.

[5] K. Zhou, Z. Ren, S. Lin, H. Bao, B. Guo and H. Shum, "Real-time smoke rendering using compensated ray marching," In *Proc. of ACM SIGGRAPH'08*, pp.1-12, 2008.

[6] H. W. Jensen, *Realistic Image Synthesis Using Photon Mapping*, A. K. Peters, Ltd, 2001.

[7] T. Purcell, C. Donner, M. Cammarano, H. W. Jensen, and P. Hanrahan, "Photon mapping on programmable graphics hardware," ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware, pp.41-50, 2003.

[8] H. W. Jensen and P. H. Christensen, "Efficient simulation of light transport in scenes with participating media using photon maps," In *Proc. of ACM SIGGRAPH'98*, pp.311-320, 1998.

[9] S. M. Rubin and T. Whitted, "A 3-dimensional representation for fast rendering of complex scenes," In *Proc. of SIGGRAPH'80*, pp.110-116, 1980.

[10] A. S. Glassner, "Space subdivision for fast ray tracing," pp.160-167, 1988.

[11] A. Fujimoto, T. Tanaka, and K. Iwata, "Arts: Accelerated ray-tracing system," *Computer Graphics and Applications*, vol.6, no.4, pp.16-26, 1986.

[12] V. Havran, *Heuristic ray shooting algorithms*, Ph.D. Thesis, Department of Computer Science and Engineering, Faculty of Electrical Engineering, Czech Technical University in Prague, 2000.

[13] T. Foley and J. Sugerman, "Kd-tree acceleration structures for a gpu raytracer," In *Proc. of HWWS'05*, pp.15-22, 2005.

[14] D. R. Horn, J. Sugerman, M. Houston, and P. Hanrahan, "Interactive k-d tree gpu raytracing,"

In *Proc. of I3D'07*, pp.167-174, 2007.
[15] S. Son, *GPU-Assisted Acceleration Techniques for Photon-Mapping Based Volume Rendering*, Master's thesis, Department of Computer Science and Engineering, Sogang University, 2008.
[16] D. Cha, S. Son, and I. Ihm, "GPU-assisted high quality particle rendering," *Computer Graphics Forum (Eurographics Symposium on Rendering 2009)*, vol.28, no.4, 2009.
[17] B. Kang, I. Ihm, and C. Bajaj, "Extending the photon mapping method for realistic rendering of hot gaseous fluids: Natural phenomena and special effects," *Computer Animation and Virtual Worlds*, vol.16, no.3,4, pp.353-363, 2005.
[18] M. Meyer and J. Anderson, "Statistical acceleration for animated global illumination," *ACM Transactions on Graphics(ACM SIGGRAPH 2006)*, vol.25, no.3, pp.1075-1080, 2006.
[19] I. Wald and V. Havran, "On building fast kd-trees for ray tracing, and on doing that in O(N log N)," In *Proc. of the 2006 IEEE Symposium on Interactive Ray Tracing*, pp.61-69, 2006.
[20] K. Zhou, Q. Hou, R. Wang and B. Guo, "Real-time KD-tree construction on graphics hardware," *ACM Transactions on Graphics(ACM SIGGRAPH ASIA 2008)*, vol.27, no.5, pp.1-11, 2008.

차 득 현
정보과학회논문지 : 컴퓨팅의 실제 및 레터 제 16 권 제 2 호 참조

이 용 일
2008년 2월 서강대학교 컴퓨터공학과 졸업(공학사). 2010년 2월 서강대학교 컴퓨터공학과 졸업(공학석사). 2010년 3월~현재 엔메이즈 소프트 개발팀. 관심분야는 컴퓨터 그래픽스

임 인 성
정보과학회논문지 : 컴퓨팅의 실제 및 레터 제 16 권 제 7 호 참조