한국컴퓨터그래픽스학회 Vol. 14, No. 3, P. 31~46

효율적인 피사계 심도 생성을 위한 두 가지 기법

서영선 임인성()
서강대학교 공과대학 컴퓨터공학과
{zard17@grmanet.sogang.ac.kr, ihm@sogang.ac.kr}

Two Efficient Methods for Generating Depth-of-Field

Young-Seon Suh Insung Ihm
Department of Computer Science & Engineering, Sogang University

요 약

피사계 심도란 카메라에서 초점이 맞은 지점을 전후로 초점이 맞은 것과 유사한 선명도를 가지는 영역을 말한다. 이 영역을 벗어난 물체들은 아웃 포커싱 되어 흐릿하게 보이게 된다. 렌더링 한 이미지에 피사계 심도 효과를 내게 되면 그렇지 않은 이미지에 비해 훨씬 사실적인 영상을 얻을 수 있다. 렌더링에서 피사계 심도 효과를 내기 위한 기존 연구는 크게 두 가지로 나눌 수 있다. 먼저 분산 광선 추적법은 한 픽셀에 대해 주어진 렌즈 영역을 샘플링 하는 방법이다. 이 방법은 샘플링 횟수가 많아질수록 노이즈가 없는 정확한 영상을 얻게 되지만 그러기 위해서는 시간이 매우 많이 걸린다. 반면 후처리 방법은 광선 추적법 등을 이용하여 얻은 핀홀 카메라 이미지와 깊이 정보를 이용하여 피사계 심도 효과를 내는 것인데 분산 광선 추적법 만큼 정확하지는 않지만 훨씬 빠른 속도로 유사한 결과를 얻을 수 있어 많은 연구가 진행되어 왔다. 그러나 후처리 방법은 핀홀 카메라에서 생성된 이미지만을 이용하므로 실제 렌즈에서 보이는 모든 정보를 알 수 없다는 한계가 존재한다. 본 논문에서는 이러한 두 가지 형태의 방법을 각각 발전시켜 후처리 기반 피사계 심도 계산의 정확도를 높이는 방법과 분산 광선 추적법의 계산 속도를 향상 시키는 방법을 제시한다.

Abstract

The depth of field is the range that the objects inside of this range treated to be focused. Objects that are placed out of this range are out of focus and become blurred. In computer graphics, generating depth of field effects gives a great reality to rendered images. The previous researches on the depth of field in computer graphics can be divided into two major categories. One of them is the distributed ray tracing that samples the lens area against each pixel. It is possible to obtain precise results without noise if enough number of samples are taken. However, to make a good result, a great number of samples are needed, resulting in an enormous timing requirement. The other approach is the method that approximates depth of field effect by post-processing an image and its depth values computed using a pin-hole camera. Though the second technique is not that physically correct like distributed ray tracing, many approaches which using this idea have been introduced because it is much faster than the first approach. But the post-processing have some limitations because of the lack of ray information. In this paper, we first present an improvement technique that corrects the previous post-processing methods and then propose another one that accelerates the distributed ray tracing by using a radiance caching method.

키워드 : 피사계 심도, 분산 광선 추적법, 복사 휘도 캐시, 후처리 방법.
Keywords : Depth of field, distributed ray tracing, radiance cache, post processing.

1. 서론

일반적으로 렌더링에 사용하는 핀홀 카메라(pin-hole 카메라)와 달리 현실에서 존재하는 카메라의 렌즈는 유한한 크기의 지름을 가진다. 그러므로 이러한 카메라에서는 피사계 심도(depth of field) 영역 안에 존재하는 물체들은 선명하게 나오고 그렇지 않은 물체들은 뿌옇게 흐려지는 효과가 나게 된다. 이처럼 실제 카메라로 촬영한 것과 같은 느낌을 주기 위해서는 이미지에 피사계 심도 효과를 내는 것이 필수적이다. 그러나 이 효과를 정확하게 계산하기 위해서는 굉장히 많은 시간과 메모리를 사용하는 방법이 필요하다. 또한 이 문제를 해결하기 위해 빠른 시간에 근사적으로 처리한 방법들은 정확하게 계산한 방법에 비해 몇몇 경우 상당히 품질이 낮은 결과를 생성한다.

정확성을 요구하는 렌더링을 할 때 대부분의 경우 한 픽셀 영역을 여러 개로 분할하는 수퍼 샘플링(super sampling) 방법을 사용한다. 분산 광선 추적법(distributed ray tracing)은 렌더링 시 부드러운 반사(soft reflection)나 부드러운 그림자(soft shadow), 피사계 심도, 모션 블러(motion blur) 등의 효과를 내기 위해 제안된 방법으로 핀홀 카메라에서 광선을 추적을 하는 개념을 확장하였다[1]. 이 방식에서는 렌즈의 여러 지점에서 출발한 광선을 각각 추적하고 그 결과를 합하여 이미지의 한 점을 구성한다. 실제 렌즈를 통해 빛이 굴절되는 것을 모방함으로써 이론적으로 정확한 결과를 얻게 되지만 연속된 영역인 렌즈를 이산적으로 샘플링 함으로써 오차가 발생하게 되고, 또한 렌즈 영역에 대해서 충분히 많은 샘플링을 하지 않게 되면 그 오차는 더욱 커지게 된다. 그러므로 이 방법은 렌즈 지름이 n배 증가하면 렌즈 영역에 대해 n^2 배의 샘플링을 수행해야 증가하기 전과 비슷한 수준의 결과를 얻을 수 있고, 이는 렌더링 시간이 그만큼 증가함을 의미한다. 또한, 렌즈의 위치와 크기 등의 설정이 조금만 바뀌어도 렌즈를 통해 지나가는 광선의 방향이 모두 새로 계산되기 때문에 그때마다 새로 전체 영상을 렌더링 해야 하는 문제가 있다.

누적 버퍼(accumulation buffer) 방법은 렌즈 영역 내에서 변하는 여러 시점에서 각각 렌더링을 하여 나온 결과들을 합하여 최종 영상을 생성한다[2]. 이 경우 분산 광선 추적법과 마찬가지로 실제에 근접한 영상을 얻기 위해서는 많은 시점에서 렌더링을 해야 하므로 시간이 많이 걸리게 된다.

위의 두 방법은 렌더링 하고자 하는 장면의 기하 복잡도(geometric complexity)에 종속적이다. 렌더링 해야 할 객체가 많아지고 복잡해질수록 샘플링 한 번 당 계산 시간이 늘어나게 되고, 많은 샘플링을 필요로 하는 피사계 심도 효과에 대해 매우 비효율적인 방법이다.

반면, Potmesil 등은 핀홀 카메라에서 렌더링 된 영상을 후처리하여 피사계 심도 효과를 내는 방식을 처음 제안하였다[3]. 이 방법의 기본 아이디어는 핀홀 카메라로 렌더링 한 이미지와 대응되는 깊이 정보를 이용하여 각 픽셀의 착란원(circle of confusion)을 구해서 그 픽셀이 흐리게 되는 정도를 계산한다. 후처리 방식의 가장 큰 장점은 계산 시간이 장면의 기하 복잡도에 종속적이지 않다는 것이다.

후처리를 통한 피사계 심도 계산은 크게 두 가지 방향으로 연구되었다. 먼저 순방향 매핑(forward mapping) 방법은 [3]의 방법을 이어받아 핀홀 이미지의 각 점에 대한 착란원을 구하고 계산된 착란원 크기에 의해 한 점은 여러 점들에게 자신의 색상을 나누어 준다. 이 과정을 모든 점에 대해 반복하여 새로운 결과를 얻는다. 올바른 결과를 얻기 위해 각 픽셀은 여러 픽셀로부터 얻은 색상을 뒤에서 앞으로 정렬해야 하며 높은 정밀도의 연산을 이용하여 혼합해야 한다. 또한 가려져 있는 영역에 대해 적절하게 색상을 분배하는 것도 어려운 작업이다[4]. 가려져 있는 영역의 문제는 [5]에 부분적인 해결책이 제시되었지만 속도가 느리고 여전히 분산 광선 추적법에 비해 부정확하다. 순방향 매핑 방법은 정렬하는데 드는 비용이 크고 GPU에서 높은 정밀도 연산을 지원하지 않는 문제가 있다. 또한 가려져 있는 영역 때문에 이미지 전체의 색상 에너지를 일정하게 유지하기 힘든 문제가 있어 실시간 응용 프로그램에 적합한 방법은 아니다.

한편 역방향 매핑(reverse mapping) 방법은 순방향 매핑 과정을 반대로 수행하는데, 각 점에서 자신의 색상을 나누어 주는 것이 아니라 자신의 착란원 크기에 의해서 다른 점들의 색상을 가져와 혼합한다는 차이가 있다. 이 방법은 순방향 매핑 방법과 비슷하지만 텍스쳐를 순차적으로 검색하는 GPU의 특성을 이용할 수 있는 방법이기 때문에 하드웨어 가속을 이용하여 순방향 매핑보다 빠른 시간에 영상을 생성하는 것이 가능하다[4,6]. 이 방법을 응용하여 핀홀 영상을 깊이에 따라 여러 개의 층(layer)으로 나누어서 각 깊이에 따라 일괄적으로 같은 크기의 커널을 사용하여 흐리게(blurring) 한 후 혼합하는 방법도 연구되고 있다. 역방향 매핑 방법은 물체가 흐려질 경우 앞에 있는 물체에 까지 흐려진 색이 흘러들어 오는 단점이 있어 이를 해결하기 위한 여러 방법들이 제시되었다[4,7]. 또한 여러 층으로 나누는 방법의 경우 여러 층에 걸쳐 연속된 물체에 불연속한 선 등이 나타나게 되는데 Barsky 등은 이것을 시간이 오래 걸리는 이미지 처리 방식으로 해결하였고[8], Kraus 등은 이를 발전시켜 GPU를 이용하여 빠른 시간에 불연속한 선을 제거하는 방법을 제안하였다[9]. 그러나 여전히 가려져 있는 영역에 대한 정보가 없음으로 인한 부정확한 결과에 대해서는 해결되지 않은 문제로 남아있다.

이처럼 핀홀 영상을 이용한 후처리 방법은 렌즈 위의 한 점에 대해서만 렌더링을 수행하면 되므로 계산 시간이 훨씬 적게 걸린다는 장점이 있지만 렌즈 모든 영역에서 보이는 점을 다 계산한 것이 아니므로 이론적으로 정확한 결과를 필요로 하는 경우에는 적합하지 못한 방법이다.

본 논문에서는 수퍼 샘플링과 후처리 방법의 단점을 보완하여 향상된 피사계 심도 영상을 생성하는 방법을 제안한다. 첫째로 핀홀 이미지에서 가려져 있던 물체들에 대한 영역을 찾아 그 부분을 추가적으로 렌더링 함으로써 기존의 후처리 방법보다 정확한 피사계 심도 결과를 얻는다. 두 번째 방법은 계산된 색상을 메모리에 저장해 놓고 유사한 지역의 교차점에 대해 저장된 색상을 이용하여 중복된 계산을 줄임으로써 분산 광선 추적법을 가속하는 방법을 제안한다.

2. 피사계 심도의 렌더링

2.1 컴퓨터 그래픽스에서의 피사계 심도

2.1.1 얇은 렌즈 모델

카메라로부터 거리를 달리해서 물체를 놓게 되면 어떤 부분은 초점이 맞고 어떤 부분은 초점이 맞지 않게 된다. 이때 물체의 초점이 맞아 보이는 범위를 피사계 심도라고 한다. 피사계 심도가 얕을 때 중경에 놓인 물체에 초점을 맞추면 전경과 원경은 초점이 맞지 않게 된다. 피사계 심도는 렌즈의 특성과 카메라로부터 초점이 맞는 지점까지의 거리에 의해 결정된다.

기하 광학에서는 Snell의 법칙을 이용하여 두 물질의 경계에서 빛이 굴절되는 각도를 계산한다[10]. 실제 카메라의 렌즈는 빛이 굴절되어 한 점에 모이도록 볼록하게 설계되어 있지만, 본 논문에서는 실제 렌즈처럼 유한한 크기의 지름을 가지지만 두께가 0에 수렴하는 얇은 렌즈(thin lens)를 가정한다[11]. 얇은 렌즈 모델에서 렌즈의 특성은 초점 거리(focal length)와 조리개 값(f-number)의 조합으로 나타낼 수 있다. 초점 거리는 렌즈가 무한대에 초점이 맞춰졌을 때의 렌즈와 필름 사이의 거리를 말하며 조리개 숫자는 렌즈를 통해 빛이 얼마나 들어가는가를 나타내 주는 표준적인 척도이다. 조리개 숫자가 낮을수록 렌즈 구경이나 조리개 구멍이 상대적으로 커진다. 얇은 렌즈를 사이에 두고 양쪽으로 빛이 모이는 점을 각각 R_1과 R_2라 했을 때 초점 거리 F에 대해 식 (2.1)이 성립하고 이를 얇은 렌즈 공식이라 한다.

$$\frac{1}{F} = \frac{1}{R_1} + \frac{1}{R_2} \tag{2.1}$$

R_1 값이 고정되었다고 가정하면 렌즈의 초점거리 F에 따라 R_2 값이 결정되므로 얇은 렌즈에서 피사계 심도를 계산할 때 렌즈의 고유한 특성으로 초점거리 값을 설정해주는 것이다.

2.1.2 착란원(circle of confusion)

렌즈를 통해 지나간 광선이 물체와 교차했을 때 교차점이 피사계 심도 밖에 존재할 경우 상평면(image plane)에 유한한 반경을 가지는 원의 형태로 나타나게 된다. 이것을 착란원이라 하는데 착란원의 반경이 작을수록 그 상의 선명도는 높아지게 된다.

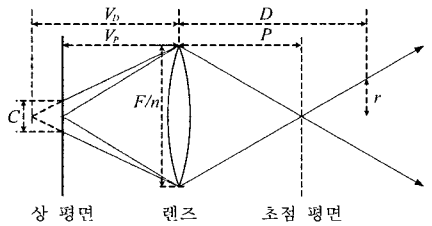


그림 2.1 착란원 크기의 결정

그림 2.1은 렌즈와 물체 사이의 거리에 따른 착란원의 크기를 도식화 한 것이다. 초점거리가 F이고 조리개 값을 n이라 할 때 렌즈의 지름은 $\frac{F}{n}$가 되며, P는 초점이 맞은 거리(focal distance), V_P는 이미지 평면을 의미한다. 여기서 P가 속하는 렌즈에 수직한 평면을 초점 평면(focal plane)이라고 하며 이미지 평면의 각 점에서 출발한 광선들은 이 평면 위의 한 점에서 모이게 된다. 이 때 얇은 렌즈 공식에 의해 V_P를 식 (2.2)와 같이 나타낼 수 있고, 같은 방법으로 렌즈로부터 거리가 D인 점이 초점이 맞는 거리 V_D는 식 (2.3)과 같이 표현할 수 있다.

$$V_P = \frac{FP}{P-F},\ P > F \tag{2.2}$$

$$V_D = \frac{FD}{D-F},\ D > F \tag{2.3}$$

이 때 착란원의 지름 C는 닮은꼴 삼각형의 비례 관계에 의해 표현할 수 있다(식 (2.4)).

$$C = |V_D - V_P|\frac{F}{n V_D} \tag{2.4}$$

2.2 분산 광선 추적법

상평면 위의 한 점 I에서 출발한 광선들은 그림 2.1에서 볼 수 있듯이 초점 평면의 한 점에서 모이며, 원뿔 모양으로 퍼져나간다. 이 때 거리 D에서의 원뿔의 반지름 r을 식 (2.5)로 나타낼 수 있고, 이것은 상평면 위에서 아래와 같은 R의 크기를 가진다.

$$r = \frac{1}{2}\frac{F}{n}\frac{|D-P|}{P} \tag{2.5}$$

$$R = r\left(-\frac{V_P}{D}\right) \tag{2.6}$$

식 (2.5)와 식 (2.6)을 비교하면 다음과 같은 관계를 도출할 수 있다.

$$R = \frac{C}{2}$$

그러므로 원뿔 내부의 점들은 상평면 위의 점 I를 포함하는 착란원을 가지며 원뿔 외부의 어떤 점도 점 I에 영향을 미치지 않는다는 것을 알 수 있다.

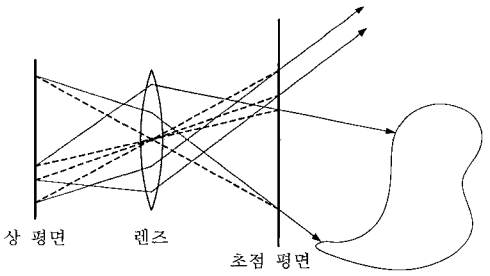


그림 2.2 분산 광선 추적법의 원리

분산 광선 추적법의 핵심 아이디어는 상평면의 출발한 하나의 광선이 피사계 심도와 부드러운 반사, 부드러운 그림자를 구하는데 모두 이용되므로 수퍼 샘플링 방법의 차원의 저주(curse of dimension)로부터 자유롭다는 것이다[1]. 본 논문에서는 그 중 피사계 심도에 대해서만 다루는 분산 광선 추적법을 설명한다(그림 2.2 참조). 분산 광선 추적법을 이용한 렌더링을 하기 위해서 사용자가 조절할 수 있는 변수는 렌즈의 반지름과 초점 평면의 거리이다. 먼저 상평면의 각 샘플점에서 렌즈의 중심 방향으로 발사된 광선과 초점 평면과의 교차점을 계산한다. 이제 상평면의 각 샘플점에 대하여 렌즈 평면의 한 점을 선택하는데 난수를 발생하여 이 점을 랜덤하게 선택함으로써 균일하게 선택할 때 나타나는 무늬 등을 제거할 수 있다. 상평면의 한 샘플점 I에서 출발한 광선은 렌즈 평면의 샘플점을 지나 초점평면 상의 교차점을 향해 날아간다. 이 교차점은 위에서 말한 원뿔 내부의 점 중 하나가 되고 이 점은 I에 영향을 미치는 점임을 위에서 보였으므로 분산 광선 추적법이 올바르게 동작함을 알 수 있다. 이런 식으로 계산된 광선들을 이용하여 기존의 재귀 광선 추적이나 전역 조명(global illumination) 계산 등을 수행하면 정확한 피사계 심도 효과를 낼 수 있다.

그러나 이러한 분산 광선 추적법은 샘플링 횟수가 충분하지 못하면 노이즈가 많이 발생하게 된다. 이론적으로 상평면의 한 점은 렌즈의 영역에서 들어온 빛들을 섞은 색을 가지게 되는데 렌즈 영역에 대해 너무 적은 샘플링을 하게 되면 결과에 오차가 생기게 된다. 예를 들어 상평면의 한 점당 한 번의 샘플링을 한다고 가정했을 때 한 점 I_i에서는 렌즈 윗부분을 샘플링하고 바로 옆의 점 I_{i+1}에서는 렌즈 아랫부분을 샘플링 할 경우 두 부분이 확연한 색상 차이를 보이게 되고 이것이 노이즈가 발생하는 원리이다. 그러므로 이러한 노이즈를 없애기 위해서 렌즈가 커질수록 더욱 많은 샘플링을 수행해야하고 렌더링 시간은 샘플링 횟수에 비례하여 늘어나게 되므로 만족스런 결과를 얻기 위해서는 상당히 많은 렌더링 시간이 필요하다.

2.3 계층적 깊이 영상을 이용한 후처리 방법

2.3.1 Barsky 등의 방법[8]

Barsky 등은 색번짐(color bleeding) 현상을 막기 위해 핀홀 영상을 여러 개의 부 영상(sub-image)들로 분해하여 각각에 대해 블러 필터(blur filter)를 적용하는 방법을 제안하였다[8]. 영상을 부 영상으로 분해할 경우에 부분적으로 가려진 점들이 부 영상 상에서 어두운 윤곽으로 나타나게 되는 문제가 있다. 또한 여러 깊이에 대해 연속된 물체의 경우 각 부 영상에 다른 크기의 블러 필터를 적용함으로써 연결 부분에 선 등의 부자연스러운 모양이 나타나게 된다. Barsky 등은 이러한 문제들에 대해 복잡한 이미지 처리 기법들을 사용하였으나 이 방법들은 그래픽스 하드웨어에 적용하여 가속하기에 적합하지 않고 속도가 느리다.

2.3.2 Kraus 등의 방법[9]

Kraus 등은 부 영상 기반의 심도 표현 기법을 하드웨어로 가속하여 Barsky의 방법을 발전 시켰다. Kraus 등이 제안한 방법은 컬링(culling), 나타남(disocclusion), 매팅(matting), 블러링(blurring), 블렌딩(blending)과 같이 몇 단계의 부분 계산으로 구성되어 있다[9](그림 3.3 참조). 각 부 영상 I_i^{sub}는 균일한 블러 반지름에 의해 흐려지게 되는데 이를 위해 각

부 영상의 블러 반지름 r_i를 정하고 이를 이용하여 깊이 좌표 z_i를 계산할 수 있다. 컬링 단계에서는 각 부 영상 I_i^{sub}에 대해 픽셀의 깊이 z가 z_{i-2}보다 작은 픽셀을 이 부 영상에 속하지 않는 전경 픽셀로 간주하고 투명한 검은색으로 칠한다.

나타남 단계에서는 컬링 단계에서 잘려진 부분을 채운다. 입력된 핀홀 이미지에서는 가려져 있었으나 렌즈 영역을 유한하게 확장할 경우 보이게 되는 부분이 존재하고 이 부분에 대한 정보를 알 수 없다. 그러므로 이 부분이 주변의 색과 비슷하다고 가정하고 주변의 색들을 보간하여 칠한다. 이 과정을 수행함으로써 나중에 블러링 계산이 적용되었을 때 가려져 있던 부분이 검은 테두리가 되어 나타나는 현상을 없앨 수 있다. 이 단계는 Strengert 등이 제안한 GPU(Graphics Processing Unit) 기반 피라미드 보간 방법을 이용하여 빠르게 수행하는 것이 가능하다[12].

매팅 단계에서는 부 영상들의 투명도를 결정한다. 해당 픽셀이 어떤 부 영상에 속하는지의 여부는 그 픽셀에 깊이에 따라 결정되므로 매팅 함수를 $w_i(z) \in [0,1]$로 나타낼 수 있으며 그림 2.3과 같이 정의된다. 매팅 함수 $w_i(z)$를 모든 색상 요소와 투명도 값에 곱함으로써 해당 픽셀이 부 영상 I_i^{sub}에 속하는 정도에 따라 색을 조절한다.

블러링 단계에서는 Kraus 등이 제안한 피라미드 방법을 사용한 4x4 크기의 박스 필터를 이용하여 각 부 영상을 블러링한다[13]. 블렌딩 단계에서는 이렇게 생성된 각 부 영상들을 뒤에서부터 앞으로 합쳐서 최종 결과 영상을 만들어 낸다.

이 방법은 비교적 적은 샘플링으로 생성된 핀홀 영상을 입력으로 받아 GPU 기반의 피라미드 이미지 처리 방식을 이용하여 빠른 속도로 피사계 심도 효과를 낼 수 있다. 그러나 핀홀 카메라 위치에서 가려져 있는 부분에 중요한 물체가 있을 경우에 이를 반영하지 못한다는 단점이 있다. 이 문제는 렌즈의 크기가 작고 피사계 심도가 넓을 경우에는 눈으로 구별할 수 없을 정도이지만 렌즈의 크기가 매우 커서 초점 평면 앞부분의 물체가 투명해지는 정도가 심해진다

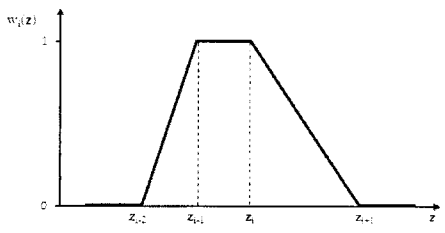


그림 2.3 매팅 함수의 정의

면 가려져 있던 물체를 나타내지 못하기 때문에 실제 카메라와 동떨어진 결과가 나온다(그림 2.4).

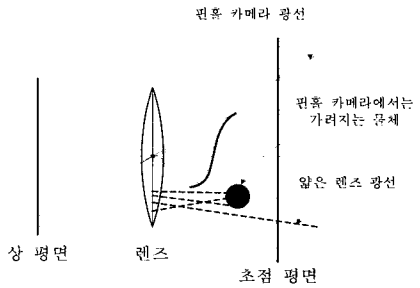


그림 2.4 후처리 방식의 문제점

3. 향상된 피사계 심도 렌더링 I : 후처리 기반 방법

3.1 가려짐(occlusion) 문제를 해결한 후처리 방법

기존의 후처리 방법은 가려져 있는 물체에 대한 충분한 정보를 포함하지 못하기 때문에 종종 부정확한 영상을 생성한다. 이를 보완하기 위해 먼저 실제로 렌더링을 수행하기 전에 가려져 있는 영역에 대해 찾아내고 이 영역을 추가적으로 렌더링 하여 후처리를 적용하는 방법을 생각해 볼 수 있다.

3.3.1 추가 영역의 계산 및 렌더링

먼저 2장에서 설명한 Kraus 등의 방법[9]을 이용하여 각 부 영상의 깊이 좌표를 구한다. 구해진 깊이 좌표들은 각각 추가 영역을 얻기 위한 과정의 앞 절단 평면(near clipping plane) 좌표가 된다. 이제 기존의 핀홀 영상을 얻는 것과 동일한 설정을 유지하면서 앞 절단 평면만을 바꿔가며 렌더링을 한다(그림 3.1). 이 때 실제로 정확한 색상을 구하는 것이 아니라 추가적으로 렌더링 해야 할 영역만을 얻으면 되므로 OpenGL 렌더링을 통하여 각 부 영상 I_i^{sub}에 대응되는 추가 영역 비교를 위한 임시 부 영상 I_i를 빠르게 렌더링하고 이렇게 얻어진 이미지들을 비교하여 추가 영역을 얻는다.

추가 영역의 비교는 뒤에 있는(깊이 값 z가 큰) 부 영상에서부터 앞에 있는 부 영상으로 순차적으로 수행하게 된다. 먼저 추가적으로 렌더링 할 영역에 대한 정보를 가지고 있는 매스킹 텍스쳐(masking texture)를 생성하고 모든 픽셀을 0으로 설정한다. 매스킹 텍스쳐는 각 부 영상에 대해 추가적으로 렌더링 할 부분은 1, 이미 렌더링 한 적이 있어서

렌더링을 하지 않아도 될 영역을 0으로 설정되어 필요한 부분만 렌더링 할 수 있게 만든다. 추가 영역을 렌더링 할 때 매스킹 텍스쳐를 입력으로 받아 1로 설정되어 있는 부분만 렌더링 한다. L_0는 기존의 후처리 방식에서 입력 이미지로 받는 핀홀 이미지로써 앞 절단 평면을 충분히 앞에 두어 렌더링 한 이미지이다. 임시 부 영상 L_i에 대하여 먼저 L_0와 비교하여 두 이미지에 동일하게 나타나는 부분은 무시하고, 겹치지 않고 L_i에 새로 나타난 영역을 1로 설정한다. 이와 겹치지 않고 L_i에서 새로 나타난 영역은 핀홀 카메라에서 가려져 있던 영역이므로 새로 렌더링 해야 할 부분으로 간주한다.

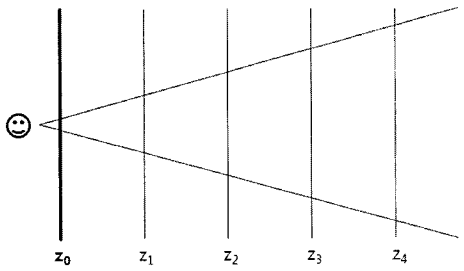


그림 3.1 깊이 좌표를 이용한 절단 평면의 설정

두 번째로 L_{i+1}과 L_i를 비교한다. 추가 영역의 계산은 뒤쪽의 부 영상으로부터 완료되므로 자신보다 뒤에 있는 부 영상에서 이미 새로 발견한 영역을 중복해서 추가적으로 렌더링 할 필요가 없다. 그러므로 자신의 뒤에 있는 이미지와 비교하교 동일하게 존재하는 영역에 대해서 매스킹 텍스쳐

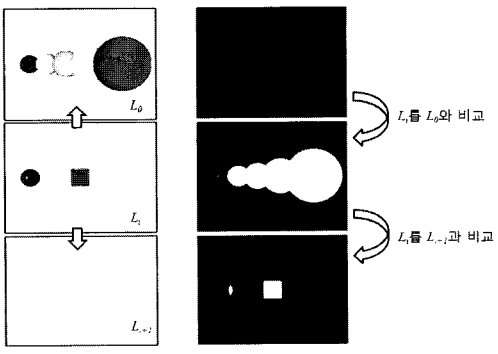


그림 3.2 L_i에서 추가적으로 렌더링 할 영역의 계산. 왼쪽은 절단 평면으로 잘라서 렌더링 한 이미지이고 오른쪽은 매스킹 텍스쳐임. 오른쪽 가장 하단에 위치한 이미지의 흰색 부분이 i 번째 부 영상에 대해 추가적으로 렌더링 해야 할 영역이다.

의 해당 픽셀을 다시 0을 설정해주는 과정이 필요하다.

이런 식으로 n개의 부 영상에 대해 n개의 매스킹 텍스쳐를 생성하는 작업이 끝났다면 이를 이용해서 각 부 영상의 추가영역들을 렌더링 한다. 이 때 렌더러는 추가 영역을 구할 때와 같은 앞 절단 평면을 사용하며 각 추가 이미지 A_i에 대해 대응하는 매스킹 텍스쳐의 픽셀의 값이 1인 부분에서만 샘플링을 수행하여 색상을 계산하고 0인 부분에 대해서는 아무런 계산도 하지 않는다.

3.1.2 후처리 과정

이제 생성된 추가 이미지들과 기존의 핀홀 이미지, 그리고 깊이 영상을 이용하여 후처리 방법을 적용한다. 2장에서 설명하였듯이 Kraus 등의 방법[9]에서 해당 픽셀이 어떤 부 영상에 속하는 정도를 계산하는 매팅 과정에 들어가기 전에 현재 부 영상의 전경에 해당되는 부분을 잘라내는 과정인 컬링과 잘라내진 부분을 주변 색들로 채우는 나타남 계산을 수행한다.

본 논문에서는 컬링과 나타남 과정을 추가된 이미지들로 채우는 과정으로 대체하였다. n개의 부 영상이 존재한다고 가정 할 때 각 부 영상 I_i^{sub}에 대해 픽셀의 깊이 가 보다 작은 픽셀을 추가이미지의 픽셀로 채운다. 이 때 앞에서 중복되는 렌더링을 막기 위해 뒤에 있는 이미지와 비교하여 매스킹 텍스쳐를 생성했기 때문에 추가이미지 A_i만으로는 I_i^{sub}가 필요로 하는 모든 정보를 알 수 없다. 그러므로 가장 뒤에 존재하는 추가 이미지인 A_{n-1}로부터 시작하여 A_i까지 뒤에서 앞으로 차례대로 픽셀을 채워 나간다. 결과적으로 전경 픽셀이 잘려나간 부분에 주변의 색 대신 실제로 가려져 있던 부분을 채움으로써 실제 물체 배치에 좀 더 근접한 결과를 얻을 수 있다. 이후의 연산은 기존 연구의 방법을 그대로 따른다.

3.2 실험 결과

이 절에서는 본 논문에서 소개한 후처리 방법을 사용했을 때의 실험 결과를 보여준다. Kraus 등의 방법[9]과 본 논문에서 제안한 방법을 각 단계 별로 비교하여 결과를 비교해 보았다. 또한 추가적으로 렌더링 하는데 걸리는 시간 및 후처리 시간의 비교를 통해 추가 영역을 렌더링 하는 방법의 유용성을 알아본다. 이 절에서 설명되는 모든 실험은 Intel Core2 CPU 6600 2.4 GHz의 CPU 와 3GByte의 메모리를 장착한 컴퓨터에서 진행되었고, GPU는 NVIDIA Geforce 8800 GTX를 이용하였다. 비교를 위해서 세 가지 장면에 대해 실험을 수행하였다. 먼저 'Garfield with sphere' 장면은 렌더링

된 이미지가 320x240의 크기를 가지고, 'Color bars' 장면은 300x300, 마지막으로 'Color spheres' 장면은 320x240의 크기를 가진다. 이 장면들은 모두 핀홀 카메라에서 가려져 있던 중요한 부분이 유한한 크기를 가진 렌즈에서 보이는 경우의 예이다. 입력된 핀홀 이미지와 추가로 렌더링 된 이미지들은 모두 픽셀 당 2x2개의 샘플링을 수행하여 생성하였다.

3.2.1 Kraus 등의 방법[9]과의 비교

먼저 Kraus 등의 방법[9]은 컬링과 나타남 단계에서 가려진 영역에 대해 주변의 색상들을 이용하여 채운다(그림 3.3). 반면 추가 영역을 구하는 방법에서는 각 부 영상의 깊이 좌표에 대해 앞 절단 평면을 설정하여 새롭게 렌더링 한 부분들을 이용하여 가려진 영역을 채운다(그림 3.4). 그림 3.5 - 3.7의 결과 영상을 비교해 보면 가려진 영역에 대해 추가로 렌더링 한 결과가 피사계 심도의 물리적 성질을 좀 더 반영하여 핀홀 카메라에서 가려져 있던 뒷부분의 정보가 이미지에 나타남을 알 수 있다.

두 방법의 렌더링 시간을 비교한 결과가 표 3.1에 나와 있다. 추가 영역의 계산과 후처리 방법은 GPU를 이용하여 가속되어 실시간에 처리된다. 추가 영역을 렌더링 하는 시간은 가려진 영역의 크기와 렌더링 방법에 따라 달라지는데 이 예제에서는 가려진 영역이 작고 간단한 광선 추적법을 사용하였기 때문에 그리 많지 않은 시간이 걸렸다. 후처리를 이용한 방법은 블러링을 렌더링이 끝난 후에 처리하므로 분산 광선 추적법을 이용하는 경우보다 훨씬 적은 샘플링을 사용해도 무방하다. 그러므로 시간이 많이 걸리는 전역 조명 효과를 분산 광선 추적법을 이용해 피사계 심도 효과를 내는 것보다 계산 시간이 부담이 되는 정도는 아니다.

3.3 후처리 방법의 한계

이 장에서 제안한 추가 영역을 구하여 렌더링 하는 방식은 기존 방법 보다 어느 정도 연산 시간의 증가로 납득할만한 결과를 얻는다. 그러나 이 방식은 여전히 정확한 피사계 심도의 계산이라 할 수 없다.

첫 번째 문제는 가려져 있던 영역을 추가적으로 계산하였으나 각 영역이 렌즈 전체 영역 중 얼마만큼의 영역에서 보이는가를 정확히 알 수 없다. 같은 깊이에 존재하는 물체라도 렌즈 영역에서 보이는 비율에 따라 선명도가 달라지는데 위의 방식으로는 그런 세밀한 효과를 표현할 수 없다.

두 번째 문제는 위에서도 언급한 피라미드 이미지 처리 방식의 문제이다. 피라미드 이미지 처리에서는 블러 크기를 2의 배수씩으로 밖에 증가시키지 못하므로 어느 영역에 속하는 모든 착란원들이 같은 블러 크기로 블러링 된다. 그에 따라 결과의 정확성이 떨어지고 일정 크기보다 렌즈가 커지면 이미지가 의도한 것 보다 심하게 흐려져서 상당히 뭉개지는 결과를 초래한다. 이러한 문제점들은 렌즈의 크기가 작거나 초점이 맞은 물체 앞에 가리는 영역이 없다면 눈에 띄게 나타나지 않는다. 그러나 렌즈의 크기가 커지고 가려진 영역이 많을 수록 이 방법의 부정확성이 증가하게 된다. 하지만 이러한 문제점들에도 불구하고 애니메이션 제작 시 비교적 빠른 속도로 피사계 심도를 계산해주기 때문에, 어느 정도 오차가 허용이 되는 경우(예를 들어, 빠르게 지나가는 영상의 경우) 유용하게 쓰일 수 있을 것이다.

4. 향상된 피사계 심도 렌더링 II : 분산 광선 추적 기반 방법

4.1 복사휘도 캐시(radiance caching)를 이용한 분산 광선 추적법의 가속

3장에서 제안한 방법에 비해 시간은 오래 걸리지만 보다 정확한 결과를 내는 방법을 제안한다. 현존하는 피사계 심도 렌더링 알고리즘 중에서 물리적으로 렌즈 모델에 가장 근접한 결과를 내는 것은 분산 광선 추적법이다. 그러나 분산 광선 추적법은 앞에서 언급했듯이 너무 많은 렌더링 시간이 걸리는 문제가 있다. 그러므로 분산 광선 추적법의 정확성은 유지하면서 속도를 줄일 수 있는 방법이 필요하다.

그림 4.1은 상평면 전 영역에 대해 렌즈 위의 한 점을 지나는 광선 방향의 범위를 나타낸다. 그림으로 알 수 있듯이 렌즈 여러 곳에서 출발한 광선들과 물체의 교차가 서로 거

표 3.1 후처리 방법의 렌더링 시간 비교 (단위: 초)

모델	Garfield with sphere		Color bars		Color spheres	
방법	Kraus 등의 방법[9]	본 논문의 방법	Kraus 등의 방법[9]	본 논문의 방법	Kraus 등의 방법[9]	본 논문의 방법
핀홀 이미지 생성	66.86	66.86	171.00	171.00	27.3	27.3
추가 영역 계산	-	0.01	-	0.01	-	0.01
추가 영역 렌더링	-	19.76	-	242.37	-	10.2
후처리	0.06	0.08	0.05	0.07	0.07	0.09
합	66.92	86.71	171.05	413.45	27.37	37.6

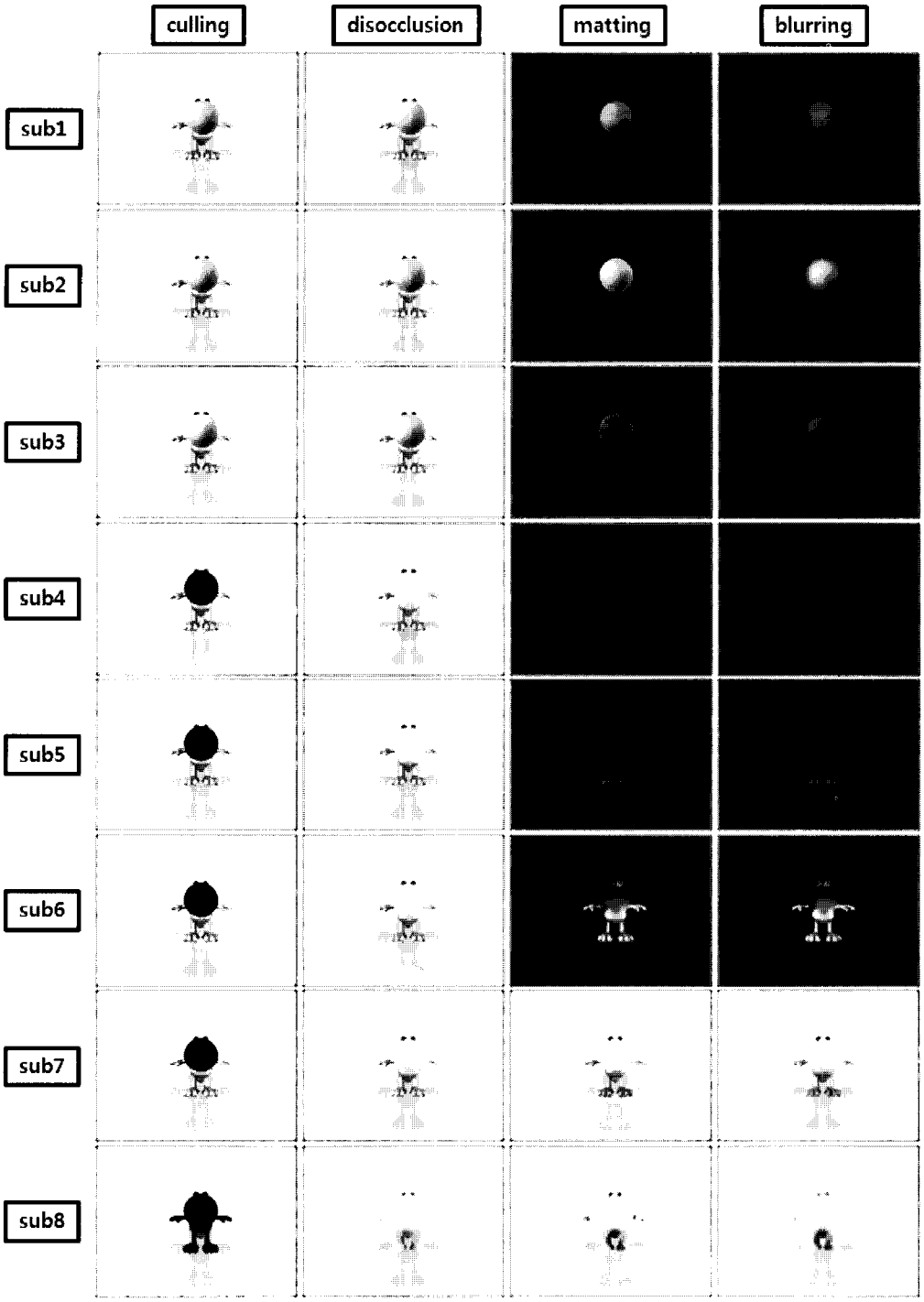


그림 3.3 Kraus 등의 방법[9]

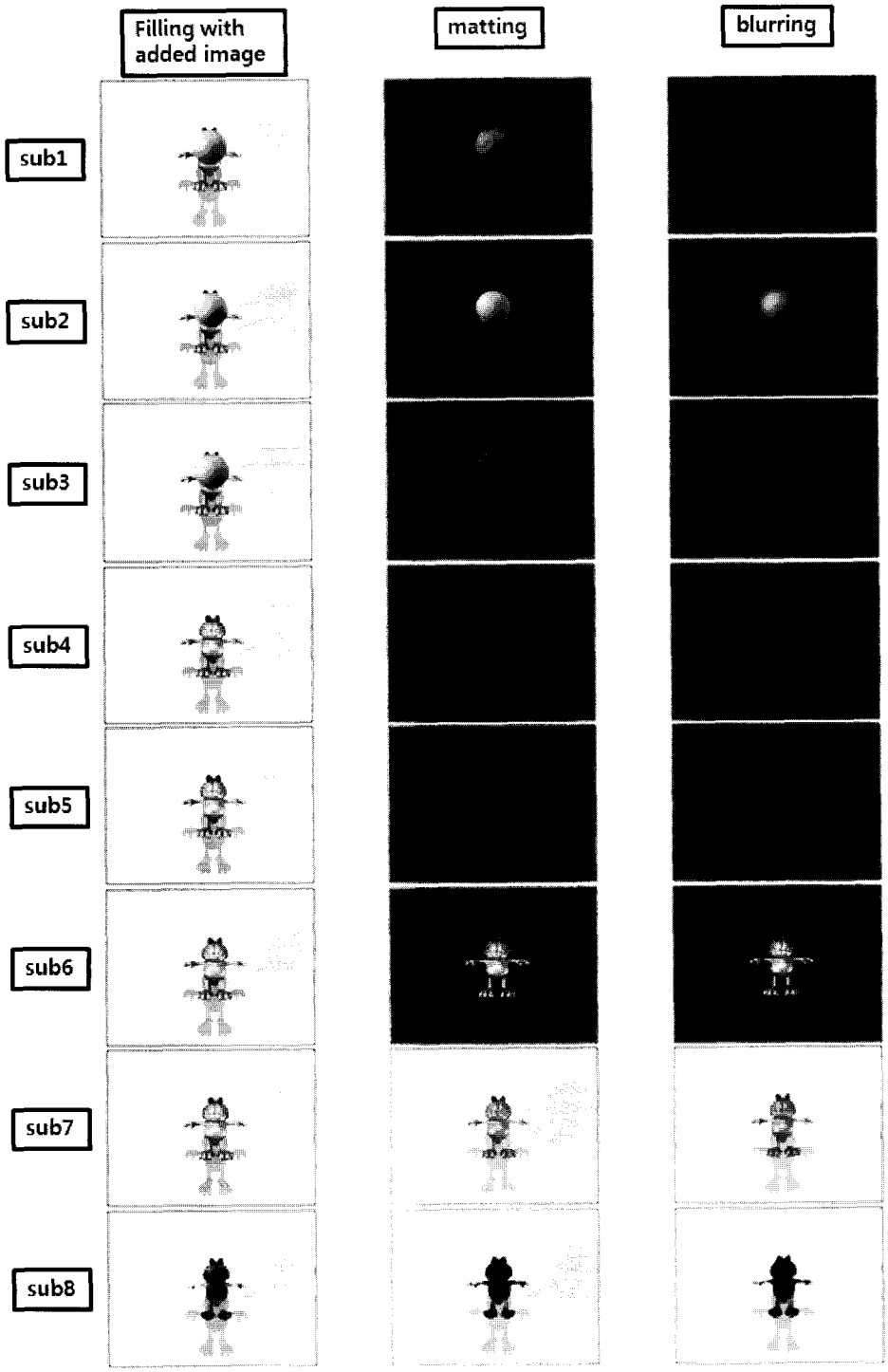


그림 3.4 추가 영역을 구하는 방법

그림 3.5 Kraus 등의 방법[9]과의 비교 (Garfield with sphere). 왼쪽이 Kraus 등의 방법이고 오른쪽이 제안된 방법을 사용하여 생성한 영상임. 오른쪽의 경우 Garfield의 얼굴과 팔 부분에 피사계 심도가 더 정확히 생성되고 있음을 알 수가 있다.

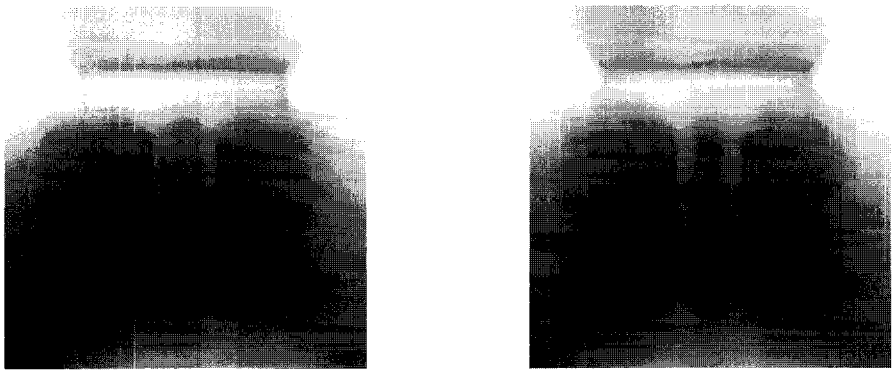


그림 3.6 Kraus 등의 방법[9]과의 비교 (Color bars). 왼쪽이 Kraus 등의 방법이고 오른쪽이 제안된 방법을 사용하여 생성한 영상임. 제안된 방법을 사용할 경우 가운데 기둥 부분의 피사계 심도가 더 정확하게 생성이 된다.

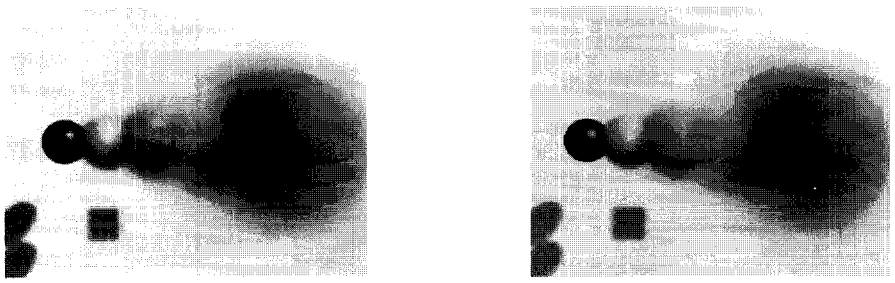


그림 3.7 Kraus 등의 방법[9]과의 비교 (Color spheres). 왼쪽이 Kraus 등의 방법이고 오른쪽이 제안된 방법을 사용하여 생성한 영상임. 제안된 방법을 사용할 경우 두 번째 구 부분에 오차로 인하여 가려졌던 육면체가 희미하게 보이게 된다.

의 비슷한 교차점에서 일어나는 경우가 많은데 분산 광선 추적법에서는 이 때마다 그 지점에서 새로 색상을 계산하여 그 값을 상평면의 픽셀에 더한다(그림 4.2). 그러나 포톤 매핑(photon mapping) 등의 전역 조명 방법을 사용하면 교차점 하나의 색상을 계산하는데 굉장히 많은 연산이 필요하게 된다. 렌즈의 여러 위치에서 한 장면을 보는 피사계 심도 계산의 특성 상 비슷한 위치의 색상 계산이 많이 일어나게 되므로 중복되는 교차점의 계산을 최소화하여 전체 렌더링 시간의 감소를 꾀하는 방법을 생각해 볼 수 있다.

4.1.1 자료 구조와 알고리즘

본 논문에서는 충분히 가까운 거리에 이미 계산된 색상이

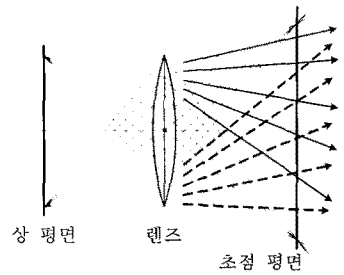


그림 4.1 렌즈 위의 여러 점에서의 광선의 범위

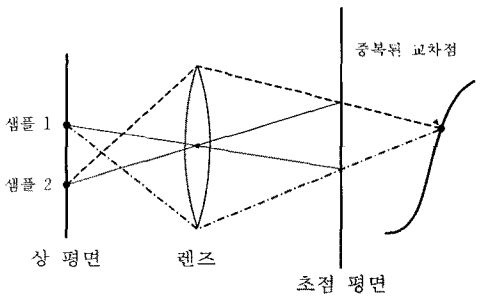


그림 4.2 중복 계산되는 교차점

있으면 새로운 색을 계산하는 대신 기존에 계산해 놓은 값을 가져오는 방식으로 분산 광선 추적법을 가속한다. 이를 위해 새로 계산된 색상과 위치를 저장하고 이들을 팔진 트리(octree) 자료구조에 넣어서 위치에 따라 빠르게 검색할 수 있게 한다.

먼저 분산 광선 추적법과 같은 방식으로 렌즈에서의 샘플링 위치와 방향을 결정하여 광선을 생성한다. 이 광선이 물체와 어떤 지점 p에서 교차하면 먼저 팔진 트리를 탐색하여 p 근처에 이미 계산된 샘플이 있는지를 검사한다. 만약 적당한 거리에 계산된 값들이 존재한다면 그 값들을 보간한 색을 가져오고, 근처에서 적당한 값을 찾지 못하면 새로운 색상을 계산하여 가져옴과 동시에 팔진 트리에 샘플을 추가해준다.

Algorithm 1 캐시 알고리즘

```
if(hitSomething)
    if(!InterpolateRadiance(isect.P,&R))
        R = FindRadiance()
        factor1 = CalculateFactor1()
        factor2 = CalculateFactor2()
        factor3 = (factor2 > min_factor2)?factor2:min_factor2
        factor = factor1 * factor2
        OctreeAdd(RadianceSample(R, isect.P, factor))
    return R
```

4.1.2 샘플의 범위 계산

저장된 샘플을 사용할 수 있는지의 여부를 결정하기 위해 새로 계산된 샘플을 계산할 때 그 샘플이 영향을 미칠 범위를 정하여 함께 저장한다. 이 범위는 샘플의 깊이와 초점 평면에 의해 결정되어 계산된다. 여러 가지 설정이 가능하겠지만 여기서는 정확도를 높이기 위해 촘촘하게 샘플링 한다. 최종적으로 생성될 이미지의 y축 길이를 y_{Res}라 하고 상평면의 y축 길이를 i_{height}라 하면 교차점 p의 깊이 D와 렌즈로부터 상평면까지의 거리 V_P를 이용하여 최종 이미지의 한 픽셀이 해당 깊이에서 어느 정도 크기를 가지는지 알 수 있다(식 (4.1)).

$$factor_1 = \frac{i_{height}}{y_{Res}} \frac{D}{V_P} \tag{4.1}$$

$factor_1$의 범위를 넘어가면 최종이미지에서 한 픽셀을

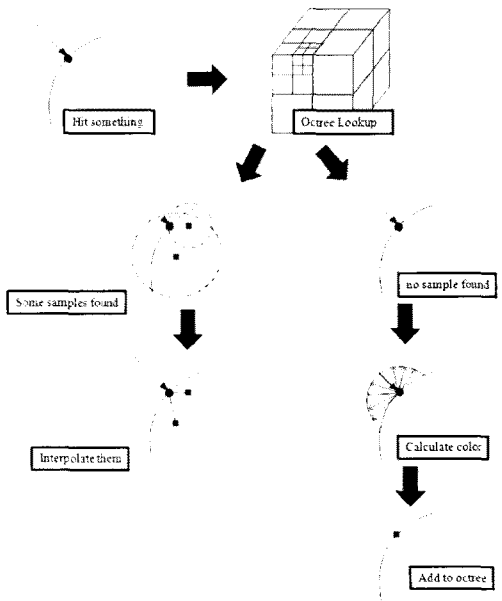


그림 4.3 캐시 알고리즘 요약

넘어가는 것과 마찬가지이므로 색을 가져올 때 오차가 생길 위험이 크다 할 수 있다. 그러므로 초점이 맞은 선명한 부분은 샘플이 영향을 미칠 반경을 $0.5factor_1$으로 두어 한 픽셀 내에서만 영향을 미치게 조정한다. 반면 초점이 맞지 않아 흐리게 되는 부분은 상대적으로 자세한 렌더링이 필요하지 않으므로 선명한 부분보다 성기게 샘플을 저장하여 렌더링 시간과 메모리를 절약할 수 있다. 이를 위해 깊이 D에서의 착란원의 크기를 구하여 $factor_2$라 한다. 여기서 P는 초점 평면을 나타낸다(식 (4.2)).

$$factor_2 = R\frac{|D-P|}{P} \tag{4.2}$$

$factor_2$의 크기가 일정 크기 이상일 때 $factor_1$과 $factor_2$를 곱한 값을 샘플의 범위로 정하며 일정 크기보다 작아서 선명하게 보이는 지점은 사용자에 의해 미리 설정된 $factor_2$의 하한값을 $factor_1$에 곱해 샘플의 범위로 사용한다.

4.1.3 샘플의 검색

팔진 트리를 탐색하여 광선과 교차하는 노드에서 샘플을 발견하면 그 샘플을 쓸 수 있는지 여부를 검사한다. 본 논문에서는 샘플의 사용 여부를 결정하기 위해 두 가지 테스트를 수행한다. 먼저 현재 검색된 i번째 샘플 p_i에 대해 p_i의 법선 벡터 n_i와 교차점 p의 법선 벡터 n을 비교해서 둘의 방향이 기준 보다 크게 다르면 색의 차이가 클 확률이 높으므로 p_i는 탈락한다. 두 번째로 교차점 p와 샘플의 위치 사이의 거리를 계산하여 위에서 정해준 샘플의 범위 안에 들어오면 그 샘플을 사용하기로 하고 아니면 탈락한다. 이런 식으로 교차하는 모든 샘플에 대해 사용 여부를 결정하여 색상과 가중치를 더해서 마지막에 더해진 색상을 더해진 가중치로 나눠 최종적으로 보간된 색상을 구한다. 식 (4.3)은 오차 값 ϵ_i를 계산하는 식이고 식 (4.4)는 오차 값을 이용해서 가중치 w_i를 구하는 식이다. 여기서 p와 n은 각각 교차점의 위치와 법선 벡터를 의미하고 p_i, n_i, $d_{\max}$ 는 i번째 샘플의 위치, 법선 벡터, 범위를 의미한다.

$$\epsilon_i = \frac{\|p-p_i\|}{d_{\max}(n \cdot n_i)} \tag{4.3}$$

$$w_i = \{1-\epsilon_i(p,n)\}^2 \tag{4.4}$$

이 때 샘플의 찾은 개수에 따라 보간한 값을 사용할 지의 여부를 결정한다. 난반사가 일어나는 지역의 경우 광선이 들어온 방향에 상관없이 비슷한 색상을 반사시키므로 한 개의 샘플로도 유사한 값을 구할 수 있다. 그러므로 검색된 샘플이 한 개 이상이면 사용한다. 그러나 정반사가 일어나는 지역은 광선의 방향에 따라 색상이 달라지고 거울과 비슷한 성질을 가지는 물체는 광선이 들어온 방향에 따라 다른 위치의 값을 가져오므로 한 개의 샘플만을 사용하여 보간하면 의도치 않은 결과가 나올 수 있다. 이 경우 적당히 많은 수의 샘플이 모이기까지 기다리고 기준치 이상이면 보간 값을 사용해도 된다.

4.2 실험 결과

이 절에서는 본 논문에서 소개한 복사 휘도 캐시 방법을 사용했을 때의 실험 결과를 보여준다. 분산 광선 추적법과의 시간 비교를 통해 얼마나 많은 속도 향상이 일어났는지를 보여주고 몇 가지 조절 가능한 변수를 이용하여 정확도를 조절해 본다. 또한 앞 장에서 제안한 후처리 방법과의 정확도를 비교한다. 이 절에서 설명되는 모든 실험은 Intel Core2 CPU 6600 2.4GHz의 CPU 와 3Gbyte의 메모리를 장착한 컴퓨터에서 진행되었고 GPU는 NVIDIA Geforce 8800 GTX를 이용하였다. 비교를 위해서 세 가지 장면에 대해 실험을 수행하였다. 먼저 'Cornell box' 장면은 렌더링 된 이미지가 300x300의 크기를 가지고, 'Garfield with sphere' 장면은 320x240, 마지막으로 'Gnome closeup' 장면은 320x240의 크기를 가진다.

4.2.1 분산 광선 추적법과의 비교

먼저 캐시를 사용하지 않은 분산 광선 추적법과 캐시를 사용하여 가속한 방법을 비교하여 실험하였다. 이 때 정확한 결과를 위해 $factor_2$의 하한값은 0.5를 사용하였다.

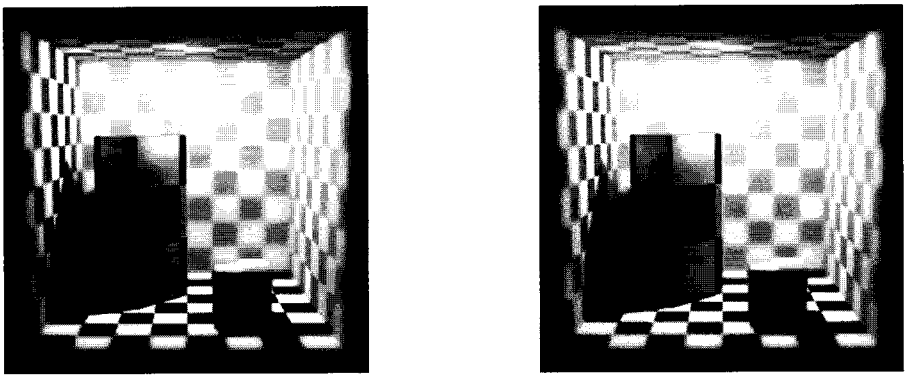
두 방법을 이용해서 렌더링 한 결과는 각 모델에 대해 그림 4.5, 그림 4.6, 그림 4.7과 같다. 그림들을 비교해보면 두 방법을 사용하여 렌더링 한 결과가 거의 유사함을 알 수 있다. 표 4.1은 각 장면에 대해 두 방법의 렌더링 시간과 캐시를 사용함으로써 증가되는 속도를 나타낸 표이다. 'Gnome closeup' 장면과 'Garfield with sphere' 장면은 각각 약 4배와 약 5.2배의 속도가 향상되었다. 'Cornell box' 장면의 경우 상

표 4.1 캐시 사용에 따른 렌더링 시간 비교 (단위: 초)

모델	캐시 미사용	캐시 사용	속도 증가(배)
Gnome closeup	4617	1171	4.0
Garfield with sphere	438	85	5.2
Cornell box	7063	308	22.9

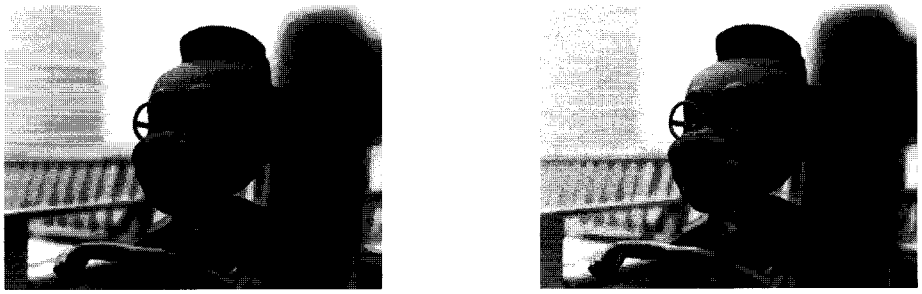
(a) 캐시 미사용 (b) 캐시 사용

그림 4.5 분산 광선 추적법과의 비교 (Garfield with sphere)



(a) 캐시 미사용 (b) 캐시 사용

그림 4.6 분산 광선 추적법과의 비교 (Cornell box)



(a) 캐시 미사용 (b) 캐시 사용

그림 4.7 분산 광선 추적법과의 비교 (Gnome closeup)

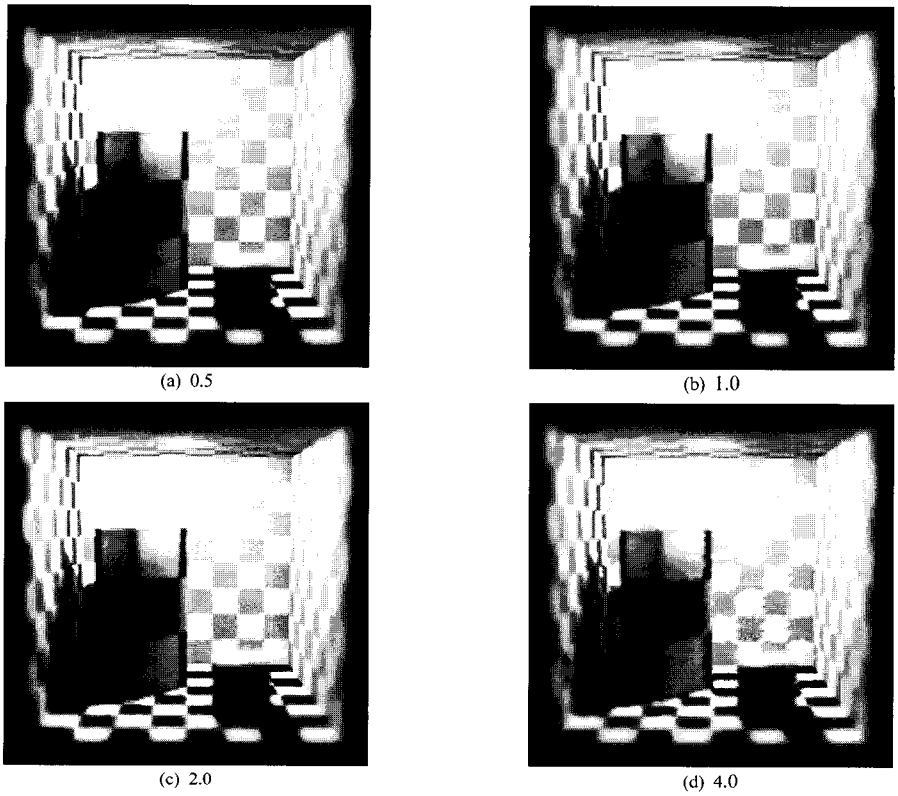
대적으로 작은 공간에 물체들이 모여 있어 캐시의 효과가 극대화 될 수 있는 장면으로 약 22.9 배의 속도 향상을 보였다.

4.2.2 $factor_2$의 하한값에 따른 비교

두 번째로 $factor_2$의 하한값의 변화에 따른 변화를 실험하였다. 위에서 설명했듯이, 새로 계산된 샘플의 범위를 구

표 4.2 하한값에 따른 렌더링 시간 비교 (단위: 초)

$factor_2$의 하한값	시간(초)
0.5	307
1.0	218
2.0	192
4.0	181



(a) 0.5 (b) 1.0

(c) 2.0 (d) 4.0

그림 4.8 $factor_2$의 하한값에 따른 결과

하기 위해서 $factor_1$과 $factor_2$를 곱하는데 이 때 계산된 $factor_2$가 하한값보다 작다면 그 값 대신 하한값을 이용하여 샘플의 범위를 구한다. 그러므로 $factor_2$의 하한값이 작을수록 촘촘하게 샘플이 저장되어 보다 정밀한 결과가 나옴을 예상할 수 있다. 그림 4.8은 'Cornell box' 장면에서 $factor_2$의 하한값을 두 배씩 증가시키면서 그에 따른 결과의 변화를 보여준다. 예상과 같이 $factor_2$의 하한값이 0.5일 경우 가장 정확한 결과가 나왔으며 두 배씩 증가할 때마다 결과의 정확도가 눈에 띄게 감소하는 것을 볼 수 있다. 표 4.2는 그림 4.8의 각 결과에 대해 걸린 렌더링 시간을 나타내는 표인데 하한값이 커질수록 샘플의 범위가 늘어나므로 계산 시간이 줄어듦을 알 수 있다.

4.2.3 후처리 방법과의 비교

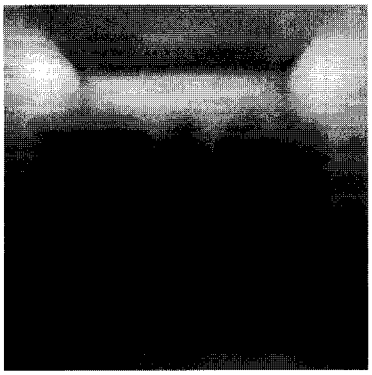
후처리 방법은 가려져 있는 물체에 대한 정보를 포함하지 않고 있거나 3장에서 제안된 방법으로 가려져 있는 물체에 대해 추가 렌더링을 하더라도 핀홀 이미지와 추가된 부분들이 렌즈 전체에서 얼마나 보이는지를 알 수 없으므로 정확한 결과가 나올 수 없다. 반면 복사 휘도를 캐시하는 방법은 속도는 오래 걸리지만 정확하게 피사계 심도 효과를 낼 수 있다. 위에서 언급한 방법들에 대해 최종적으로 비교한 그림 4.9와 표 4.3에서 알 수 있듯이 복사 휘도 캐시 방법은 후처리 방법으로 처리하기 힘든 가려진 영역과 흐리게 되는 부분의 정도를 세밀하게 표현하여 기준으로 삼은 분산 광선 추적법에 매우 근접한 빠른 결과를 얻는다.

표 4.3 최종 결과 비교 (단위: 초)

방법	시간
분산 광선 추적법	8825
캐시 방법	630
Kraus 등의 방법[9]	171
향상된 후처리 방법	413

5. 결론

본 논문에서는 컴퓨터 그래픽스에서 피사계 심도를 렌더



(a) 분산 광선 추적법

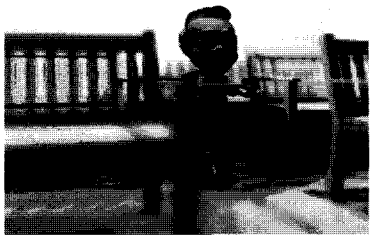
(b) 캐시 방법

(c) Kraus 등의 방법[9]

(d) 향상된 후처리 방법

그림 4.9 최종 결과 비교 (Color bars)

(a) 캐시 미사용 (3868초)



(b) 캐시 사용 (550초)

그림 5.1 정반사 하는 물질에 캐시를 적용한 예

링 하는 방법을 발전시켜 두 가지 방법을 제시하였다. 한 가지 방법은 기존의 후처리 기반 피사계 심도 렌더링 방법에서 가려진 영역에 대한 정보의 부족으로 잘못된 결과가 나옴을 보완하기 위해 가려진 영역을 찾아 이 부분을 추가적으로 렌더링 하여 기존의 후처리 방식에 끼워 넣는 방법을 제안하였다. 이 방법으로 기존의 후처리 방식보다 좀 더 정확한 결과를 얻을 수 있었지만 여전히 이론적으로 정확한 방법인 분산 광선 추적법에 비교할 때 흐려짐이 심하게 발생하며 가려지는 정도가 부정확한 결과를 얻는다. 두 번째로는 기존의 분산 광선 추적법의 방법을 이용하되, 중복된 계산이 많이 발생하는 피사계 심도 계산의 특성 상 복사휘도를 캐시하여 비슷한 영역의 색상을 중복적으로 계산하지 않고 이미 저장된 값을 보간하여 사용한다. 이 방법을 통해 장면의 특성에 따라 5 - 20 배 정도의 속도 향상을 이룰 수

있었다.

향후 연구 주제로는 4장의 복사 휘도 캐시 방법 발전시키는 방법들을 생각해 볼 수 있다. 먼저 캐시된 정보를 보존하여 카메라의 위치가 약간 이동할 경우 미세하게 변화하는 장면에 대해 이전 프레임에서 캐시된 정보를 이용하여 연속된 여러 장면을 효율적으로 렌더링 하는 방법을 고려해 볼 수 있다. 또한 본 논문의 캐시 방법은 난반사하는 물체에 대해서만 고려하였지 때문에 정반사하는 물체에 대해 캐시할 경우 왜곡된 결과를 생성하게 된다(그림 5.1). 그러므로 정반사 하는 성질의 물체에 대해서 효과적인 처리 방법을 생각하여 정반사 시에도 캐시를 사용할 수 있는 방법을 연구할 수 있다. 마지막으로 캐시의 오차 측정 방법을 향상하여 캐시 효율을 높이는 방법을 연구할 수 있을 것이다.

감사의 글

본 연구는 지식경제부 및 정보통신연구진흥원의 IT신성장 동력핵심기술개발사업(2006-S-045-03, 기능 확장형 초고속 렌더러 개발)의 일환으로 수행되었습니다.

참고 문헌

[1] R. L. Cook, T. Porter, L. Carpenter. Distributed ray tracing. ACM SIGGRAPH Computer Graphics, 18(3): 135-145, 1984.

[2] P. Haeberli, K. Akeley. The accumulation buffer : hardware support for high-quality rendering. ACM SIGGRAPH Computer Graphics, 24(4): 309-318, 1990.

[3] M. Potmesil, I. Chakravarty. A lens and aperture camera model for synthetic image generation ACM SIGGRAPH Computer Graphics, 15(3): 297-305, 1981.

[4] J. Demers. Depth of field: A survey of techniques. GPU Gems, pages 375-390, Addison-Wesley Professional, 2004.

[5] M. Shinya. Post-filtering for depth of field simulation with ray distribution buffer. In Proceedings Graphics Interface '94, pages 59-66, 1994.

[6] P. Rokita. Fast generation of depth of field effects in computer graphics. Computers & Graphics, 17(5): 593-595, 1993.

[7] J. D. Mulder, R. van Liere. Fast perception-based depth of field rendering. In VRST ï00: Proceedings of the ACM Symposium on Virtual Reality Software and Technology, pages 129-133, 2000.

[8] B. A. Barsky. Vision-realistic rendering: Simulation of the scanned foveal image from wavefront data of human subjects. In APGV'04: Proceedings of the 1st Symposium on Applied Perception in Graphics and Visualization, pages 73-81, 2004.

[9] M. Kraus, M. Strengert. Depth-of-Field Rendering by Pyramidal Image Processing. In Proceedings EG 2007, 26(3): 645-654, 2007.

[10] K. B. Wolf. Geometry and dynamics in refracting systems. European Journal of Physics, 16: 14–20, 1995.

[11] J. E. Greivenkamp. Field Guide to Geometrical Optics. SPIE Field Guides vol. FG01, page 14, SPIE, 2004.

[12] M. Strengert, M. Kraus, T. Ertl. Pyramid methods in GPU-based image processing. In Proceedings Vision, Modeling, and Visualization 2007, pages 169-176, 2006.

[13] M. Kraus, M. Strengert. Pyramid filters based on bilinear interpolation. In Proceedings GRAPP 2007, pages 21-28, 2007.