사실적 장면 표현을 위한 렌더링 기술 동향

The Recent Trends of Rendering Technologies for Realistic Scene Representation

장호욱 (H.W. Jang) 디지털액터연구팀 선임연구원
이인호 (I.H. Lee) 디지털액터연구팀 팀장

목 차

컴퓨터를 이용하여 실물이나 가상의 객체를 형상화하고 움직임을 부여하는 컴퓨터 그래픽스(CG)는 현실과 구분할 수 없을 정도의 기술의 발전과 함께 영화, 게임, VR, 산업용 디스플레이, 화상통신 등 디지털 콘텐츠 전분야에서 활용폭이 넓어지고 있다. 렌더링은 CG에서 장면의 추상적인 3차원 description을 2차원 영상으로 생성하는 과정을 말하며, 생성되는 영상의 품질 및 제작시간을 결정하는 중요한 요소로 인식되고 있다. 과거에 비해 전역 조명 및 셰이딩 기술이 발전하면서 극도로 사실적인 영상의 생성이 가능하게 되었으나 이를 렌더링하는 것은 많은 시간을 필요로 하고 있어, 사실적인 영상을 고속으로 렌더링하기 위한 연구가 진행되고 있다. 본 고에서는 CG 제작 과정에서 널리 사용되고 있는 주요 렌더러들의 기능들과 미래의 기술 발전 방향을 분석한다.

Ⅰ. 서론

컴퓨터 그래픽스는 인간이 상상할 수 있는 객체나 장면을 표현하고 실제 세계에서 실현이 불가능한 것을 나타내기 위한 기술로 가상 객체 및 장면의 형상을 만드는 모델링 기술, 객체를 현실 세계에서 보이는 모습으로 보이게 하는 렌더링 기술, 정지 이미지들을 연속적으로 빠르게 재생함으로써 장면이 실제로 움직이는 것처럼 보이게 하는 애니메이션 기술을 적용하여 가상의 영상을 생성한다. 렌더링 기술은 3차원 CG 오브젝트들을 형상과 위치, 광원과 시점 정보에 의해 2차원 영상으로 변환하면서 모델링된 장면에 실재감을 부여하는 과정을 말한다. 렌더링에서의 영상 생성은 일반적으로 3단계를 거쳐 이루어지는데, 첫번째 단계는 3차원상의 점, 선, 면 등을 2차원 평면에 투영하는 단계이며, 두번째 단계는 불투명 오브젝트의 면과 불투명 오브젝트 뒤에 있는 면을 차별화하여 앞에 보이는 면은 보이게 하고 앞의 물체에 가려지는 뒷면은 제거해야 한다. 세번째 단계는 오브젝트의 보이는 면에 대해서 음영을 표시하게 된다.

렌더링은 수작업에 의한 품질보완이 용이한 모델링이나 애니메이션 과정과는 달리 어떤 렌더러를 사용하는가에 따라 성능이 크게 좌우되며 이에 따라 생성되는 영상의 실재감과 품질이 결정된다고 할 수 있다. 사실적인 렌더링을 위해서는 그림자나 색상과 농도와 같은 3차원 질감이 요구되며 전역조명(global illumination) 처리를 필요로 하여 렌더링 작업에 막대한 처리 시간이 소요되고 있다. 또한 생성된 영상의 품질을 높이기 위해 렌더링된 영상을 손보아 다시 렌더링하는 경우가 많아 CG 작업중에서 가장 많은 작업 시간을 요구하고 있다.

본 고에서는 사실적 장면 표현을 위해 필요한 렌더링의 주요 기능들을 Ⅱ장에서 살펴보고 가장 널리 사용되고 있는 상용 렌더러들을 Ⅲ장에서 분석한다. 분석 렌더러들은 CG 제작업계 및 학계에서 가장 널리 사용되고 있는 RenderMan, Mental Ray, POV-Ray(Persistence of Vision Ray tracer) 및 3DS Max의 플러그인 렌더러로 사용되는 Final Render, Brazil Renderer, V-Ray를 대상으로 하였다. 마지막으로 Ⅳ장에서는 앞으로의 전망 및 결론을 맺는다.

Ⅱ. 주요 렌더링 기능

1. 은면/은선 제거

가. Z버퍼 알고리듬

1960년대 이후 수많은 은선 제거 기법들이 개발되었지만 그 중에서 가장 대표적인 은선 제거 기법이 바로 Z버퍼 알고리듬이다. Z버퍼 알고리듬은 각 면의 깊이 값을 별도의 버퍼에 임시로 저장하여 깊이 값 비교에 의해 최종적으로 어떤 면이 보일 것인지를 결정하는 방식으로 각 픽셀마다 폴리곤 테이블의 순서에 따라 각 오브젝트 표면의 깊이 값을 계산한 다음 계산된 깊이 값과 현재 Z버퍼에 저장된 값을 비교해서 새로 계산된 값이 시점과 가깝다면 프레임 버퍼를 갱신하고 그렇지 않다면 기존의 값을 그대로 사용하게 된다. 이 방식은 Z버퍼를 사용하기 위한 메모리가 별도로 필요하고 보이는 여부에 상관없이 모든 오브젝트를 고려해야 하므로 다소 비효율적인 단점과 함께 원리가 매우 단순하여 구현하기 쉽고 각 픽셀에 투영되는 면의 색과 깊이 값만 구하면 되므로 모든 유형의 오브젝트에 적용할 수 있으며 결과를 저장할 때 Z버퍼의 값을 같이 저장하면 각각의 오브젝트를 따로 렌더링해서 나중에 합성하거나 피사계 심도(depth of field) 효과를 주는 등의 작업도 가능해진다는 장점 때문에 널리 사용되고 있다.

나. 스캔라인 알고리듬

오브젝트를 구성하는 모서리 목록(edge table)과 폴리곤 목록(polygon table)을 이용하여 한 번에 한 줄씩 각 스캔라인과 만나는 면들을 골라내고 이 면들에 대해서만 보이는 면을 판정(visible-surface determination)하는 방식으로 하나의 스캔라인 위의 모든 픽셀에 대하여 각 표면의 깊이 값을 계산하

여 어떤 것이 가장 작은지를 결정한다. Z버퍼의 크기는 스캔라인 크기의 깊이 정보를 저장하는 정도만 요구되어 Z버퍼 알고리듬에 비해 요구되는 메모리가 상당히 줄어든다.

다. 워노크 알고리듬

한 화면상에서 출력해야 할 요소 자체를 얻기 위한 알고리듬으로 출력상에 많은 내용의 사실을 특색있게 유사한 형태로 그릴 수 있도록 한다. 이미지 공간 내에서 하나의 윈도를 고려하여 그 윈도가 비어있는지 혹은 윈도 내에 포함될 내용을 출력하기에 충분한 크기로 되어 있는지를 결정하고, 위의 두 가지 경우가 아니면 윈도를 4개의 동일한 부 윈도로 분할하는 과정을 반복한다.

라. Occlusion Culling

시점과 시점 방향 그리고 보여질 물체가 주어졌을 때, 눈에 보이는 부분만을 효과적으로 판단하여 그려질 영상을 만드는 데 사용되며 불필요한 데이터의 대부분을 Z버퍼링 이전 단계에서 미리 제거하여 처리되는 데이터 양을 줄이는 기법으로 실제 영상의 질에 영향을 미치지 않으면서, 실질적으로 처리하는 데이터의 상당량을 제거해 준다.

2. 표면 매핑(Surface Mapping)

3차원 오브젝트의 사실감을 부여하기 위해 오브젝트 표면에 재질을 입히는 과정으로 비트맵 이미지를 사용하는 텍스처 매핑을 비롯하여 디테일한 표현을 위한 다양한 매핑 기법들이 존재한다.

가. 텍스처 매핑(Texture Mapping)

나무나 천과 같은 질감을 표현하기 위해 표면에 미리 준비된 질감 데이터를 입히는 방식으로 일반적으로 비트맵 이미지를 사용하여 오브젝트 각 부분의 변수나 색상 값을 대치한다. 이때 사용되는 비트맵 이미지를 텍스처맵이라 부르며, 텍스처맵을 이용해서 오브젝트 표면 특성을 대체하기 위해서는 텍스처맵의 어느 픽셀이 오브젝트 표면의 어느 부분에 대응될 것인지를 결정해 주어야 한다. 이를 위해 오브젝트 표면에도 텍스처맵에 해당하는 좌표계를 설정해 주어야 하는데 오브젝트 표면에 설정되는 좌표계를 매핑 좌표계라 부른다. 매핑 좌표에는 일반적으로 사용되는 XYZ 좌표계 대신 UVW 좌표계가 사용되는데 UVW는 각각 XYZ축에 대응되며 매핑에는 일반적으로 2차원 맵만 사용하므로 UV 좌표계라고도 부른다.

나. 범프 매핑(Bump Mapping)

오브젝트 표면의 울퉁불퉁함을 표현하기 위한 매핑 방법으로 모델링으로 표현하기에는 너무 미세하게 거친 표면이나 울퉁불퉁한 모양이 애니메이션 되어야 하는 경우에 주로 사용된다. 오브젝트의 표면은 그대로 두고 표면 법선 벡터(surface normal) 값을 매핑 소스의 밝기 값에 따라 변화시켜 울퉁불퉁하게 보이는 표면을 나타내는 방식을 사용하는데, 표면 법선 벡터 값만이 바뀌었기 때문에 오브젝트 자체는 아무런 변화가 없어 오브젝트의 가장자리는 매핑 전과 동일하게 보이는 단점이 있다. 이런 현상은 범프 매핑 대신 오브젝트의 형태 자체를 매핑 소스의 밝기 값에 따라 바꿔버리는 방법인 변위 매핑(displacement mapping)을 사용하면 해결될 수 있다.

다. 절차적 매핑(Procedural Mapping)

사용자에 의해 주어지는 변수 값들을 바탕으로 프랙탈이나 노이즈같은 수학 함수에 의해 텍스처맵을 생성하여 오브젝트에 매핑시키는 방식으로 연기, 먼지, 비누, 거품, 침식 등의 매핑에 주로 이용한다. 이 방식은 다른 매핑 방식에 비해 몇 가지 장점을 가지고 있는데, 첫째, 함수에 의해 생성된 텍스처맵이 실제로 3차원 공간 전체를 채우고 있기 때문에 매핑 좌표라는 개념이 필요 없으므로 오브젝트 형태에 따라 매핑 이미지가 밀려서 매핑되는 문제가 발생하지

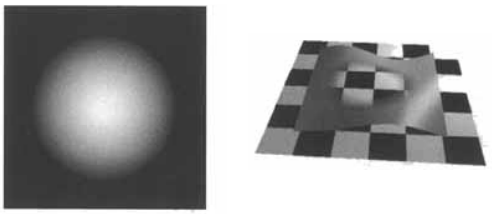
않고, 둘째, 무한대의 해상도를 갖기 때문에 고해상도 출력이 필요한 경우 비트맵 이미지를 사용하는 것보다 메모리가 적게 들며, 셋째, 비정형의 3차원 물체를 표현할 때 유용하게 사용될 수 있다.

라. 환경/반사 매핑(Environment/Reflection Mapping)

지역 조명 모델(local illumination model)만을 사용하는 렌더러에서는 반사(reflection)나 굴절(refraction)과 같이 반짝이는 오브젝트 상호간의 관계에 의해 생성되는 현상들을 표현할 수 없으므로 광선 추적법을 대신하여 반사와 굴절 현상을 표현하는 데 사용되어 왔다. 최근에는 하드웨어 사양이 높아져 과거보다 레이트레이싱 기능을 제공하는 렌더러가 많아졌지만 장면 내의 모든 오브젝트들의 반사를 정확하게 표현하는 것은 레이트레이싱으로 많은 시간이 걸리기 때문에 이를 대체하는 방법으로 여전히 많이 활용되고 있다. 환경 매핑은 시점이 바뀌거나 오브젝트가 움직이게 되면 오브젝트 특정 지점에서 반사되어 보이는 부분이 바뀌게 되므로 해당 오브젝트 표면의 색상도 바뀌는 특징을 갖고 있으며 환경에 매핑을 하므로 오브젝트에 별도의 매핑 좌표를 만들 필요가 없다. 환경 매핑을 구현하는 대표적인 방법은 천체와 같은 거대한 구에 이미지를 매핑하여 환경으로 사용하는 구형 환경 매핑과 반사면을 가진 오브젝트에 가상의 6면체를 만들어 환경을 구성하는 큐빅(cubic) 환경 매핑이 있다.

마. 투명 매핑(Transparency Mapping)

매핑 이미지의 명도 단계에 따라 객체의 투명도를 조절하는 방식으로 매핑 이미지의 흰색 부분은 투명하게, 검은색 부분은 불투명하게 처리하여 표현한다. (그림 1)은 앞쪽 공의 이미지를 뒤의 체크무늬 이미지에 적용한 투명 매핑 처리 결과를 나타낸다.



(그림 1) 투명 매핑의 예

3. 지역 조명 모델

지역 조명 모델은 빛의 작용을 계산함에 있어 사용자가 관심을 갖는 특정 표면과 이 표면을 직접 비추는 광원만을 고려하는 방식으로 광원, 표면, 시점 간의 관계를 고려해서 난반사 및 정반사되는 빛의 세기를 계산하여 물체 표면의 색상을 구하는 모델로 많은 모델들이 개발되어 있으나 비교적 간단한 계산에 의해 사실에 근접한 효과를 낼 수 있는 Phong 모델과 Blinn 모델이 가장 널리 사용되고 있다.

가. Phong 모델

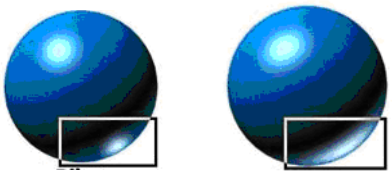
Phong이 개발한 가장 대표적이고 기본적인 지역 조명 모델로 난반사의 계산은 Lambert의 코사인 법칙을 이용하고 정반사의 계산에는 $\cos^n\alpha$을 곱하는 방법을 이용하였다. α는 반사 벡터 R과 시점 벡터 V 사이의 각도로서 이 방법을 사용하면 반사 벡터 R과 시점 벡터 V가 일치하는 지점을 최대값으로 해서 두 벡터 사이의 각도가 벌어질수록 정반사의 세기가 점점 줄어들게 된다. n은 사용자가 임의로 지정하는 숫자로서 하이라이트의 크기를 조절하는 역할을 한다. Phong 모델은 물체 표면에서 일어나는 물리적인 현상을 기초로 한 것이 아니므로 실제 상황과 다소 차이가 날 수 밖에 없으나 상대적으로 계산량이 적고 구현이 간단하며 최종적으로 얻어지는 결과물이 상당히 유사하게 보이기 때문에 기본적인 조명 모델로서 가장 널리 애용되고 있다.

나. Blinn 모델

Phong 모델보다 좀 더 사실적인 결과를 생성하기 위해 물리적으로 좀 더 올바른 조명 모델들을 개발하게 되는데 이와 같은 방법들이 바로 물리 기반

지역 조명 모델(physically based local illumination model)들이다. Blinn 모델은 최초로 개발된 대표적인 물리 기반 지역 조명 모델로서 정반사 특성을 계산함에 있어 Phong이 사용했던 $\cos^n\alpha$ 대신에 Torrence-Sparrow 모델을 사용한다. Blinn 모델이 Phong 모델과 비교해서 갖게 되는 실질적인 차이점은 정반사를 계산할 때 조명의 입사각에 따라 정반사 특성이 바뀌느냐 안바뀌느냐에 있다. 즉 Phong 모델에서는 입사각과 관계없이 항상 반사각 방향으로 정반사율이 높게 나오지만 실제 세계에서는 빛이 비스듬하게 입사할 경우 반사각보다 더 비스듬한 방향으로 정반사율이 높게 나오게 되는 off-specular peak 현상이 일어나는데 정오에 태양이 지면에서 거의 수직해 있을 때는 하이라이트가 별로 발생하지 않던 아스팔트 표면이 석양 무렵 태양 빛이 지면에 비스듬하게 입사할 때에는 훨씬 반짝여 보이는 것이 바로 이 현상 때문이다. (그림 2)에서와 같이 Blinn 모델은 off-specular peak 현상을 제대로 표현할 수 있으며, 조명이 뒤에서 비춰지는 경우에도 Phong 모델보다 사실적인 결과를 생성할 수 있다.

Blinn 모델을 비롯한 대부분의 물리 기반 지역 조명 모델들은 올바른 하이라이트를 얻기 위해 정반사 특성의 계산에만 이와 같은 방법을 적용할 뿐 난반사 특성은 Phong에서 사용했던 Lambert의 코사인 법칙을 그대로 사용하지만, 최근에는 Oren-Nayer 모델같이 난반사 특성을 계산하기 위한 모델들도 개발되고 있다.



(그림 2) Phong 모델과 Blinn 모델

4. 전역 조명 모델

지역 조명 모델만 가지고는 오브젝트간의 반사나, 굴절, 그림자 등과 같이 다른 오브젝트에 의해 반사되거나 다른 오브젝트를 투과, 굴절해서 오브젝트 표면에 영향을 미치는 빛에 의한 결과들을 재현할 수 없다는 문제가 있다. 이러한 문제를 해결하기 위하여 개발된 전역 조명 모델은 렌더링 과정에서 오브젝트 상호간의 관계도 모두 고려하기 때문에 오브젝트들 간의 상호 반사(interreflection), 굴절, 그림자 효과 등을 재현할 수 있어 훨씬 사실적인 이미지를 생성할 수 있으나 엄청난 렌더링 시간이 걸린다는 치명적인 단점을 가지고 있다. 최근에는 하드웨어 환경이 급속히 좋아지는데다 속도 개선을 위한 많은 방법들이 개발되어 활용이 점점 늘어나는 추세이다.

가. 광선 추적법(Ray Tracing)

광선 추적법은 눈에서부터 각 픽셀을 향해 광선(ray)을 방출한 다음 이 광선의 굴절, 반사 등을 계산해서 광선이 시작되었던 조명에 이를 때까지의 경로를 역추적해 나가고 이 과정을 통해 각 픽셀의 색상을 결정하는 렌더링 방법이다. 광선을 추적하는 과정에서 반사와 굴절이 되풀이하여 일어난다고 하여 재귀적 광선 추적법(Recursive Ray Tracing)이라고도 부르는데, 광선 추적 과정에서 각 오브젝트 간에 일어나는 완전 정반사 특성만을 고려하여 난반사 특성이라던가 불완전한 정반사 특성을 계산할 때에는 광선 추적법이 가능한 렌더러에서도 기존의 지역 조명 모델을 그대로 사용한다. 따라서 완전 정반사 특성이 없는 일반적인 오브젝트를 렌더링할 때에는 광선 추적법 렌더러나 지역 조명 모델만을 사용하는 스캔라인 렌더러나 결과에 별 차이가 없게 된다.

재귀적 광선 추적법이 거울 반사나 굴절, 그림자 등을 매우 사실적으로 표현해 줌에도 불구하고 부족한 것이 있으니 그것은 얻어지는 결과물들이 모두 지나치게 깨끗하고 완벽하다는 점이다. 실제 세계에서는 아무리 반짝반짝 하게 닦아 놓은 금속구라 하더라도 그 표면에 반사되는 주위 환경은 약간 흐릿하게 보이지만 재귀적 광선 추적법에서는 광선들이 반사될 때 완벽하게 100% 반사되어 아주 이상적인

거울 같은 또렷한 반사가 얻어지기 때문에 이와 같은 결과가 너무나 사실적이라 오히려 사실적이지 않은 느낌을 주게 된다. 이와 같은 문제를 해결하기 위한 분산 광선 추적법(Distributed Ray Tracing)에서는 한 줄로 이루어진 가느다란 광선 개념 대신 여러 광선을 골고루 뿌리고 그 결과를 종합하여 표면의 반사를 계산하여 부드럽게 뭉개지는 반사를 표현하게 되는데[1] 현재 사용되는 대다수의 광선 추적법 렌더러들은 이 방법을 기반으로 하고 있다.

빛이 유리나 물 같이 고르지 못한 표면을 갖는 매개체를 통과하는 과정에서 굴절되거나 반사되면서 어떤 무늬를 만드는 (그림 3)과 같은 코스틱(caustic) 현상은 분산 광선 추적법으로도 표현하지 못하는데 그 이유는 거울 반사가 전혀 없는 표면을 만나면 광선 추적 작업이 끝나 버리기 때문이다. 이와 같은 문제를 해결하여 코스틱 현상을 표현할 수 있도록 해주는 것이 이단계 광선 추적법(Two-Pass Ray Tracing)으로 먼저 광원으로부터 광선을 발사한 다음 광선이 벽이나 천장과 같은 난반사(diffuse) 표면에 닿을 때까지 추적하고 이렇게 추적된 각 광선의 에너지, 즉 코스틱 이미지를 난반사 표면상에 그려주는 첫번째 단계를 거치고 두번째 단계인 기존의 광선 추적법으로 최종적인 결과를 얻는 것이다. 이 방법은 눈과 광원 양쪽 방향으로부터 동시에 광선 추적 작업을 수행한다고 해서 양방향 광선 추적법(Bi-Directional Ray Tracing)이라고 부르기도 하고 눈으로부터 추적하는 기존의 방법과 반대 방향으로 작업이 진행된다고 해서 역방향 광선 추적법(Backward Ray Tracing)이라고 부르기도 한다.

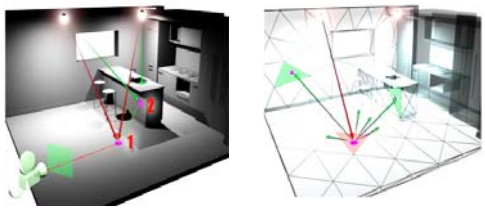
(그림 3) Caustic 현상

나. 래디오시티(Radiosity)

사실적 장면 표현을 위해서는 광원으로부터의 빛뿐만 아니라 오브젝트 서로 간에 주고 받는 빛의 영향도 고려해야 한다. 실세계에서는 직접 빛이 비춰지지 않는 곳에도 바닥이나 벽면 등 주위 오브젝트 표면에서 난반사된 빛이 다른 오브젝트 표면에 반사되어 특정 색상이 다른 오브젝트 표면에 영향을 주는 color bleeding 현상이 발생한다. 이와 같은 현상을 재현하기 위해 난반사 특성의 계산에 있어 주위의 다른 오브젝트들과의 관계를 포괄적으로 고려하는 방법으로 래디오시티 기법이 개발되었다.

래디오시티에서는 먼저 장면을 이루는 모든 표면을 패치(patch)라고 불리는 조각으로 나누고 광원에서 특정 패치로 또 이 패치로부터 다른 패치 그리고 또 다른 패치로 얼마만큼의 광 에너지가 전달되는지 그 양을 계산한다. 패치들간의 주고 받는 에너지의 흡수 및 반사 정도는 패치간의 기하학적 관계에 의해서 결정되고 이를 form-factor라고 한다.

표면을 나눈 후에는 각각의 조각들이 서로 얼마나 가까이 있는지, 어느 정도의 각도로 서로 마주보고 있는지를 확인해서 form-factor를 결정하게 되는데 두 개의 면이 매우 가까운 거리에서 서로 마주보고 있다면 1에 가까운 값이 되고 반대로 먼 거리에서 매우 벌어진 각도로 바라보기 때문에 서로 영향을 거의 미치지 않는다면 form-factor는 0에 가까운 값이 된다. 이렇게 결정된 form-factor 값을 이용해 표면들 서로 간에 주고 받는 빛의 양과 색을 계산함으로써 해당 표면의 색을 결정하게 되며, 이와 같은 알고리듬의 특성 때문에 빛을 받는 모든 패치들이 사실상 광원의 역할을 하게 된다. (그림 4)에



(그림 4) 광선 추적법과 래디오시티 계산

서와 같이 래디오시티는 광선 추적법과는 달리 현실 세계에서처럼 창문을 통해 들어오는 하나의 광원만으로도 실내의 구석구석까지 자연스럽게 밝아지는 매우 효과적이면서도 사실적인 결과물을 얻을 수 있다. 또한 난반사만을 고려하기 때문에 오브젝트들의 위치가 변하지 않으며 form-factor 값에 변화가 없기 때문에 시점 변화에 따른 렌더링 계산이 필요 없는 특징을 가지고 있어 제품 디자인 분야나 인테리어, 건축 시뮬레이션 분야에 활용도가 높다. 래디오시티의 단점은 사실적인 결과물을 얻기 위해서는 렌더링 시간이 다른 렌더링 방식에 비해 훨씬 늘어나게 되며, 난반사만 고려하기 때문에 반사, 굴절을 표현할 수 없어 대부분 광선 추적법을 함께 사용하는 하이브리드(hybrid) 형태의 렌더러 형태로 제공되는데, 먼저 래디오시티 기법을 이용해서 난반사 특성의 계산을 끝낸 다음 광선 추적법으로 반사와 굴절을 추가해서 마무리해야 한다.

다. Sub Surface Scattering

래디오시티와 유사한 빛 처리 과정을 수행하나 오브젝트의 재질까지 함께 고려한 방식이다. 일반적인 래디오시티 알고리듬은 동일한 정반사와 난반사 수치를 갖고 있으면 어떤 물질이라도 같은 방식으로 계산되나, sub surface scattering는 물질의 반투명성과 정확한 각도의 빛 반사를 계산하는 것으로 BSSRDF에 기반한 반투명 재질에 대한 사실적인 시뮬레이션 결과를 얻을 수 있다[2],[3]. (그림 5)와

(그림 5) BRDF를 이용한 얼굴 렌더링

(그림 6) BSSRDF를 이용한 얼굴 렌더링

(그림 6)은 각각 BRDF와 BSSRDF를 이용한 얼굴 렌더링 결과를 나타내며 BRDF는 BSSRDF 모델에서 sub surface scattering을 고려하지 않은 간략화된 모델이다. Sub surface scattering은 현재 Mental Ray, Brazil Renderer, V-Ray 등에서 기능을 제공하고 있다.

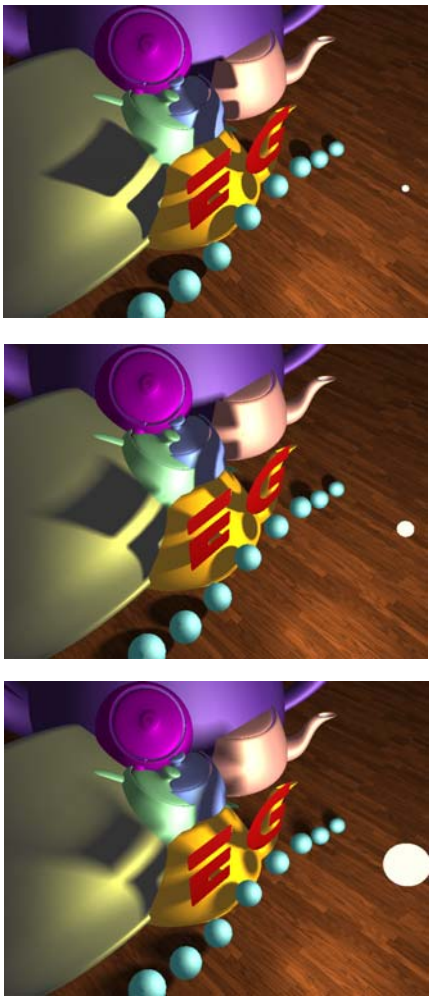
라. 포톤맵 광선 추적법

포톤맵을 이용한 광선 추적법(Photon Map Ray Tracing)의 경우도 두 단계의 광선 추적 과정을 거치는데, 광원으로부터 광자들을 방출한 다음 이 광자들이 반사되고 굴절되는 과정을 통해서 포톤맵을 생성하고 이를 이용해서 최종적인 결과를 얻는다. 일반적으로 작업의 효율을 위해 상호반사 확산(diffuse interreflection)을 재현하기 위한 글로벌 포톤맵, 코스틱을 재현하기 위한 코스틱 포톤맵, 반투과물질(participating media)을 재현하기 위한 볼륨 포톤맵을 별도로 생성한다[4]. 이 방법의 장점은 광선 추적법임에도 불구하고 상호반사 확산을 재현할 수 있다는 점과 렌더링 속도가 매우 빠르고 래디오시티에서처럼 메시(mesh)를 분할할 필요가 없으며 반 투과물질까지도 다룰 수 있어 깊이감 효과(volumetric effect)까지도 재현할 수 있는 등의 여러 가지를 들 수 있으며, 이런 장점들로 인해 최근에 가장 인기 있는 렌더링 기법이 되었다. Mental Ray가 이 기법을 사용하고 있으며 3D Studio MAX의 플러그인 렌더러인 Brazil Renderer, Final Render, V-Ray 역시 이 기법을 사용하고 있다.

5. 그림자 생성

렌더링된 영상의 사실감을 높이고 사용자가 공간적 관계를 결정하는 데 그림자가 중요한 역할을 담당하고 있으며, (그림 7)에서 보는 것과 같이 광원의 개수 및 위치에 따라 다르게 생성되어야 하기 때문에 해결이 어려운 분야로 간주되고 있다.

CG에서의 그림자는 빛을 전혀 받지 않는 영역인 umbra에서 빛을 받는 영역으로의 급격한 전환이 일어나는 하드 섀도(hard shadow)와 빛을 전혀 받지 않는 영역인 umbra 및 그림자 영역의 부드러운 전환이 일어나는 영역인 penumbra로 구성된 소프트 섀도(soft shadow)로 분류된다. 게임 등과 같이 영상의 사실감보다는 실시간성이 중요시되는 분야에서는 하드 섀도 생성을 사용하는데, 이를 위한 방법으로는 섀도 매핑(shadow mapping) 알고리듬과 섀도 볼륨(shadow volume) 알고리듬이 있다. 영상의 사실감이 중요시되는 분야에서는 소프트 섀도 생성을 주로 사용하며, penumbra wedge를 사용하는 알고리듬, back projection을 사용하는 알고리듬, 동시에 여러 섀도 맵을 사용하는 알고리듬 등의 여러 방식들이 제안되었다[5]. 그러나 현재까지 소프트-섀도 생성 알고리듬들은 너무 느리거나 앨리어싱이 발생하는 등의 단점을 가지고 있어 이를 개선하기 위한 연구가 활발히 진행되고 있으며 상용 렌더러에서도 계속적인 기능 개선을 시도하고 있다.

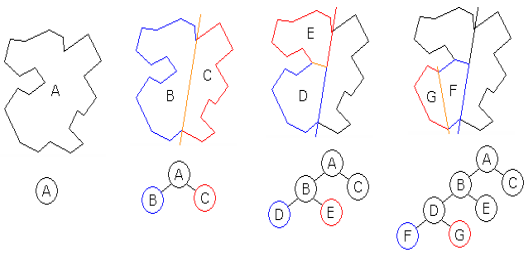


(그림 7) 광원 크기에 의한 그림자 변화

6. BSP

n차원 공간 장면을 n-1차원 하이퍼플레인(hyper-plane)으로 반복적으로 나누어 만드는 트리 구조로 충돌 감지(collision detection), 은면 제거, 레이트레이싱, 솔리드 모델링까지 다양한 분야에서 사용되고 있다. BSP 트리는 사용 목적에 따라 division 방식이 조금씩 달라지는데, 예를 들어 충돌 감지를 위한 BSP 트리는 각 파트가 개별적으로 충돌 감지될 수 있는 수준으로 나누어지며 렌더링을 위한 BSP 트리는 오브젝트들의 깊이를 측정하는 화가 알고리듬(painter's algorithm)이 사용될 수 있도록 각 파트가 convex subspace가 될 때까지 나누어진다. 개체나 관찰자의 위치가 변하는 모든 장면마다 화가 알고리듬을 적용하여 오브젝트를 다시 재정렬하는 것은 상당한 양의 계산을 필요로 하게 되지만 BSP 트리를 이용하면 장면에서 오브젝트의 상대적인 위치를 저장할 수 있기 때문에 관찰 시점이 바뀔 때마다 재정렬을 해줄 필요는 없어 렌더링 시간을 절약할 수 있다[6]. (그림 8)은 BSP 트리를 구성하는 과정을 나타내는 예이다.

BSP 트리 외의 다른 공간 분할 방법으로는 각 영역을 4개와 8개로 나누는 Quadtree와 Octree가 있



(그림 8) BSP 트리

는데 BSP 트리가 모든 n차원 공간에서 사용될 수 있는 반면 이들은 2차원과 3차원 공간에서만 사용될 수 있다.

7. HDRI 렌더링

이미지에는 LDRI와 HDRI의 두 가지 종류가 있는데 LDRI는 일반적인 비트맵 방식의 이미지를 말하며 JPEG, TIFF, BMP 등의 형식으로 픽셀 당 8비트인 256가지 모드의 색을 표현할 수 있다. HDRI는 단순히 색상을 저장하는 LDRI와 달리 각 픽셀들이 색상과 함께 밝기 강도 값을 저장하고 있어 이미지의 표현 폭이 아주 높아지게 되며, 이런 빛의 양 정보는 하나의 이미지에서 동일한 색상을 가진 픽셀간의 구별을 가능하게 만들어 준다. 예를 들어 하얀색의 종이와 하얀색의 태양광을 구분할 수 있는 픽셀의 에너지 정보가 각각의 픽셀에 함께 저장된다. HDRI 렌더링은 CG가 가지는 궁극적 목표인 실세계와 구분할 수 없는 사실적인 렌더링을 위한 기술로 실세계 조명 정보를 저장하고 있는 래디언스 맵(radiance map)을 조명 정보로서 사용하고 래디언스 렌더링 시스템의 전역 조명 알고리듬을 이용하여 가상 물체를 사실감 있게 렌더링한다. 현재 중급 이상의 상용 렌더러에서는 모두 HDRI 렌더링 기능을 제공하고 있다.

8. 분산 렌더링

네트워크상의 여러 대의 컴퓨터나 멀티 프로세스 컴퓨터의 여러 개의 프로세스를 사용하여 작업량을 분산함으로써 전체 렌더링 시간을 줄이는 기능으로 네트워크의 속도와 렌더링을 수행하는 모든 컴퓨터의 성능을 최적화 할 경우 최고의 효과를 보인다. 분산 렌더링을 구축하기 위해서 먼저 렌더링 과정을 병렬화 시켜야 하는데 여러 프로세서 사이의 통신, 적절하지 못한 작업 분배에 의한 지연, 추가적인 계산 및 저장공간 요구와 같은 부작용이 발생할 수 있어 업무를 효율적으로 분배해야 한다. 렌더링 과정의 병렬화 종류는 다음과 같다.

- 함수 병렬성: 렌더링 과정을 여러 개의 함수로 나누고 하나의 프로세서를 각 함수에 배정하는 방식으로, 수행시간이 1/파이프라인 단계 수로 줄어들지만 파이프라인의 성능이 가장 느린 단계의 성능에 의해 결정될 위험성이 있으며 파이프라인의 단계 수에 의해 성능이 제한 받는 단점이 있다.
- 데이터 병렬성: 데이터를 여러 개의 스트림으로 나누어 각 아이템들이 동시에 렌더링되는 방식으로, 렌더링 파이프라인의 단계가 아닌 프로세서의 수에 따라 성능이 제약 받으며 각 프로세서 사이의 데이터로 전달되는 네트워크 성능이 중요한 요소가 된다. 데이터 병렬성은 프로세서의 수가 많을수록 좋은 성능을 얻으며 프로세서 수에 따라 병렬성이 가변적인 특성이 있다.
- 시간적 병렬성: 수많은 프레임을 만들어야 하는 애니메이션 제작에서 시간 간격으로 작업을 나누며 각 프로세서가 자신이 렌더링해야 할 프레임을 할당 받아서 작업을 수행하게 된다.
- 하이브리드 병렬성: 여러 병렬 방식을 같이 사용하는 방법으로 함수 단계들로 렌더링 프로세서를 나눈 다음 각 함수를 여러 개의 프로세서들에 할당하여 수행시킨다.

현재 POV-Ray를 제외한 대부분의 유명 렌더러들이 분산 렌더링 기능을 제공하고 있으며, POV-Ray의 경우에도 패치 소프트웨어 및 지원도구들을 통해 분산 렌더링을 구현할 수 있다.

Ⅲ. 렌더러 기술 동향

1. Render Man

Render Man은 PRMan과 RISpec의 두 가지 의미가 함께 사용되고 있는데 RISpec은 모델러와 렌더러간의 표준 프로토콜을 정의한 인터페이스이며[7], PRMan은 RISpec에 기반하여 개발된 사실적 장면 생성을 위한 렌더러이다. RISpec에 기반한 다른 렌더러들로는 BMRT와 Aqsis 렌더러를 들 수 있으며, 이들을 RenderMan Compliant 렌더러로 부르고 있다. PRMan은 Reyes 구조에 의한 렌더링 처리를 수행하는데, Reyes 구조는 복잡한 장면을 고품질로 비교적 빠른 시간에 처리하는 목표를 가지고 마이크로 폴리곤 기본 단위 처리를 하며 레이트레이싱보다는 텍스처 매핑 위주의 렌더링 과정을 수행하여 왔다[8]. PRMan의 초기 버전에서는 레이트레이싱을 비롯한 전역 조명 기능을 제공하지 않은 관계로 반사나 굴절 등의 효과를 자동적으로 렌더러 차원에서 만들어 주지 못하고 대체 솔루션 기술을 발전시켜 왔으나, V11가 발표되면서 전역 조명, 딥 섀도(deep shadow) 기능이 Reyes 구조에 통합되면서 좀 더 정확한 장면표현을 나타낼 수 있게 되었다. 이와 함께 Render Man은 셰이딩 언어를 제공하는데, 이 언어는 절차적 언어이며 확장성이 높고 C/C++ 형태로 프로그래밍하여 Render Man에 동적으로 링크할 수 있는 특성을 갖고 있으며 실사적 시각효과를 나타내는 요소를 가지고 있다[9]. 현재 현장에서 사용되는 대부분의 오프라인 렌더러는 어떠한 형태로든 셰이딩 언어를 지원하는데, Render Man 셰이딩 언어는 가장 확실하고 잘 알려진 셰이딩 언어로 평가 받고 있다.

2. Mental Ray

독일의 Mental Images사에서 개발된 레이트레이싱 렌더링 소프트웨어로 CAD/CAM 분야에서 독보적인 위치를 차지하면서 세계 굴지의 자동차 회사들의 제품 디자인의 곡선 표면을 수치적으로 정확하면서도 가장 뛰어난 품질로 렌더링하여 왔다. Mental Ray가 3D CG 분야에서 처음 모습을 드러낸 것은 1993년 하이엔드 CG 시장을 겨냥한 SoftImage사의 디지털 스튜디오 계획의 일환으로 Mental Ray를 SoftImage 3D에 내장되면서부터이다. 현재는 SoftImage 3D의 최신 모델인 SoftImage|XSI 외에 Maya, 3DS Max, Avid 3D 등의 다양한 소프트웨어와의 통합과 Macintosh, IBM-PC, SGI, Linux 등의 다양한 하드웨어 및 OS 플랫폼에서의 운용이 가능하다[10]. 빛의 사용에 중점을 두어 한줄기 빔이 아닌 액티브빔 효과를 사용하는 Render Man과의 기술 차별화에 노력하여 그 동안 Render Man이 독점하다시피 한 영화 제작프로세스에도 Render Man을 위협하는 경쟁자로 올라섰다.

3. POV-Ray

POV-Ray는 네트워크를 통하여 모인 자발적 프로그래머 모임에 의해 개발되고 있는 렌더러로, 1980년대에 만들어진 DKBTrace를 개선 및 확장한 프로그램이다. POV-Ray는 다양한 기하 구조 및 광원 지원을 비롯한 고급 기능과 방대한 라이브러리까지 제공되면서 무료로 제공된다. 셰이더 플러그인 제공 등의 Render Man이나 Mental Ray에 비해 다소 부족한 기능들은 방대한 third-party 지원을 이용해 보완할 수 있으며 다양한 패치 소프트웨어, 지원도구, 텍스처, 모델링 데이터, 튜토리얼들을 웹에서 손쉽게 발견할 수 있다[5]. 상업적 목적 이외의 POV-Ray의 소스 및 바이너리 코드는 무료 배포가 가능하여 현재 해외에서는 컴퓨터 그래픽스 수업의 교재로 POV-Ray를 활용하는 대학들이 늘어가는 추세이다.

앞 절에서 기술한 Render Man과 Mental Ray는 스캔라인 렌더링 기본 구조에 레이트레이싱을 비롯한 전역 조명 기능을 통합한 형태로 발전하여 왔으나 POV-Ray는 레이트레이서(ray tracer)에서 출발하여 다른 기능들을 확장하여 현재의 구조로 발달하였다. 또한 POV-Ray는 NURBS를 사용하는 다

른 대다수 모델러/렌더링 소프트웨어에 비해 수학적 정의를 사용하여 오브젝트를 나타내기 때문에 POV-Ray 파일들은 POV-Ray 스크립팅 언어의 실제 소스코드로 되어 있다. 이 코드는 프로그래밍 언어에서 일반적인 많은 특성들과 데이터 파일에서 비정형적인 특성(변수, 루프, 수학적 함수, 매크로 등)들을 포함하고 있어 단순 숫자보다 훨씬 추상적인 방식으로 사물을 기술할 수 있다. 이러한 이유 때문에 POV-Ray 파일들은 다른 렌더러에서 읽을 수 있도록 변환하기 어려우며 스크립팅 언어를 인터프리트(interpret)하는 변환 과정과 오브젝트의 수학적 표현을 삼각형으로 변환하는 tessellation 과정이 필요하다[11].

POV-Ray의 단점으로는 멀티-쓰레드로 운용하여 렌더링시간을 줄이기가 까다롭다는 점을 들 수 있다. POV-Ray에서 멀티-쓰레드 프로그래밍이 불가능하지는 않지만 다양한 범주의 플랫폼에서 포터블하게 사용되는 철학을 가진 POV-Ray의 특성상 간단히 해결할 수 있는 문제는 아니다. 일반적으로 POV-Ray 렌더링 엔진은 싱글-쓰레드로 동작하므로 듀얼 펜티엄 프로세서에서 POV-Ray를 운용하면 각 CPU 파워의 50% 정도만 사용하게 된다. POV-Ray를 멀티프로세서에서 돌려 렌더링 시간을 줄이려면 여러 카피의 POV-Ray를 운용하는 방식을 사용해야 한다. 또한 3D 비디오 카드를 사용하더라도 POV-Ray의 속도가 빨라지지 않는데, 일반적으로 3D 비디오 카드는 광선 추적법 알고리듬을 위해 디자인되어 있지 않고 폴리곤 메시를 읽어 들이고 이들을 위한 스캔라인 렌더링 가속화를 위한 목적으로 많이 사용되고 있다. 이에 따라 POV-Ray의 렌더링 속도를 높이기 위해서는 광선 추적법 과정에 필요한 많은 수의 부동 소수점 계산 시간을 줄이기 위해 빠른 FPU를 사용하는 것이 최선의 방법으로 간주되고 있다. POV-Ray의 공식 버전은 단일 호스트에서 렌더링하는 구조로 되어 있으나 비공식 패치인 PVM 패치와 관련 도구로 개발된 POV-Anywhere는 여러 시스템에 분산하여 렌더링할 수 있는 기능을 제공하고 있다.

4. Brazil Renderer

Brazil Renderer는 BLUR사의 Ghost Renderer 개발자들이 독립하여 만든 렌더러로 오랜기간 동안 베타 테스트를 통해 문제점을 찾고 해결하면서 성숙한 단계로 발전하였다. Brazil Renderer는 전역 조명 계산에서 몬테카를로 엔진(Monte Carlo Engine)을 채택하고 있는데 이 엔진은 완전한 물리적 기반의 정확한 계산 능력 때문에 빛의 행동을 가장 근접하게 접근할 수 있는 방식이나 다른 전역 조명 엔진에 비해 훨씬 많은 메모리와 계산 시간을 필요로 한다. 이에 따라 Monte Carlo 방식에서 난수에 대한 계산법을 보완하여 개발된 Quasi Monte Carlo 방식을 주로 사용하고 있으며 이 방식은 V-Ray를 비롯한 다른 렌더러에서도 채택하여 사용되고 있다. 최신 버전의 Brazil Renderer에서는 스킨 셰이더(skin shader)도 새롭게 추가되었으며 셰이더 옵션 중에는 SSS의 3단계 깊이에 대한 부분과 cell structure라고 부르는 피부 조직에 대한 조정도 가능하다. 또한 브라질 렌더러에는 BCM이라 부르는 전용 카메라가 있는데 다양한 렌즈 타입을 제공하고 있으며 360의 구형 이미지를 위한 환경 매핑, Orthographic View, VR을 위한 파노라마 기능을 제공한다. 이와 함께 화면상의 오브젝트가 실사처럼 공간감이 느껴지도록 정교하고 자연스럽게 그림자가 뿌려지도록 하는 소프트 섀도와 영역 섀도(area shadow) 기능을 제공하고 있다[12]. 경쟁 제품들인 Final Render와 V-Ray와 비교하면 렌더링 시간은 가장 많이 걸리지만 특유의 독특한 색감과 탁월한 계산 능력 때문에 가장 훌륭한 전역 조명 결과물을 만든다는 사용자들의 평가이다.

5. Final Render

Final Render는 3DS Max와의 결합 정도에 따라 stage-0, stage-1이 있으며 Maya용 stage-2가 현재 개발중에 있다. Stage-0는 스캔라인 렌더러를 전면에 두는 구조로 되어 있으며 Stage-1은 semi-external 렌더링 시스템으로 되어 있으며 추가 기능

들은 stage-1의 서비스 팩 1/2를 통해 제공하고 있다. Final Render는 전역 조명을 위해 Quasi Monte Carlo, Hyper GI, FR Image의 3가지 엔진을 탑재하고 있으며 초기 stage-0부터 발전시켜온 FR Image를 핵심 렌더링 엔진으로 사용하고 있다. Hyper GI는 래디오시티 방식으로 초기에 모델에 면을 정해준 수치만큼 분할하여 분할 면에 조명 정보를 함축하여 근거리의 면끼리 빛에 대한 정보를 계산하는 방식으로 Brazil Renderer의 주력 렌더링 기법인 Quasi Monte Carlo는 빛에 대한 시뮬레이션을 가장 사실적으로 계산한다고 평가되는 방식이다[13]. Final Render는 다른 렌더러에 비해 다양한 옵션과 기능들이 있어 약간 복잡하다는 평을 듣고 있으며 셰이더의 사용 편이성과 다양한 재질 표현이 좋아 인테리어 장면에서 성능이 탁월하다. 또한 분산 렌더링 성능이 경쟁 렌더러에 비해 가장 우수하다는 평을 듣고 있다.

6. V-Ray

2002년 이전에는 베타 테스트 버전으로 무료로 배포되어 개발되어 왔으며 2002년 2월에 정식으로 상용화가 되었다.

V-Ray의 핵심 렌더링 엔진은 irradiance map으로 이것은 물체에 직접 맵을 만들어 전역 조명을 표현하는 렌더링 방식으로 렌더링 작업을 시작하기 전에 각각의 픽셀마다 전역 조명 샘플을 만들어 블록으로 계산한 후 전역 조명 샘플에 저장하여 빠르게 결과물을 얻는 방식으로, 필요한 부분에만 irradiance map 계산을 하여 보다 빠르고 정확한 결과물을 만들어 준다. V-Ray는 다른 고급 렌더러와 비슷한 기능들을 제공해주는데 포톤 매핑, 코스틱, 스페셜 카메라, 3D 모션 블러 등을 지원한다[14]. V-Ray의 장점은 경쟁 렌더러들에 비해 렌더링 시간이 빠른 것과 보다 단순하고 적응력이 높은 구조인 것, 높은 품질의 샤도를 빨리 계산할 수 있다는 점을 들 수 있으며 사용자층이 가장 넓다는 점을 들 수 있다. 단점으로는 셰이드 서비스가 경쟁 렌더러들에 비해 부족하다는 평가를 받고 있다.

Ⅳ. 결론

고품질의 사실적인 영상 생성을 위한 렌더링의 중요성이 부각되면서 다양한 렌더링 기법 연구와 개발이 진행되고 있다. 렌더링 과정은 수작업으로 성능 개선이 거의 불가능하여 사용자들이 사용하는 렌더러 기능에 의해 렌더링 품질이 좌우되고 있다. 또한 렌더링은 CG 영상 콘텐츠 제작뿐 아니라 VR, 게임, 산업용 디스플레이, 영상 콘텐츠 스트리밍 서비스, 화상통신 등의 다양한 응용분야에서 활용 가능한 핵심 기술로 간주되고 있다. 외국 유명 콘텐츠 제작사들은 고유한 장면 표현을 위해 상용 렌더러와 함께 자체적으로 개발한 렌더러를 함께 사용하고 있어, 상용 렌더러에서 부족한 기능을 보완하고 있으나 국내에서는 렌더러 개발 노하우가 없어 콘텐츠 제작 시 외국 렌더러에 의존하고 있는 실정이다.

상용 렌더러들의 기능은 각자의 고유한 특성을 가지고 있던 몇 년 전과 달리 최근에는 비슷하게 수렴되고 있는 상태이다. 예로 과거 Mental Ray 등 일부 렌더러에서만 제공되던 전역 조명 기능을 보면 현재 본 문서에서 분석한 모든 렌더러에서 제공되고 있으며 Maya나 3DS Max 등의 특정 CG 통합 도구와 연계되어 왔던 렌더러들이 다른 CG 통합 도구까지 지원 폭을 넓혀가고 있다.

기존 상용 렌더러들은 실시간과 고품질의 두 가지 목적을 한꺼번에 얻기 위한 노력을 계속하고 있으며 하드웨어 성능의 발전과 함께 렌더링 속도도 급속히 빨라지고 있으나, 실제와 구별할 수 없을 수준의 사실적 영상을 생성하기 위해서는 아직도 상당한 렌더링 시간을 필요로 하고 있다. 이에 따라 극사실적 장면들을 렌더링해야 하는 영화 제작 시에는 수백~수천 개의 렌더링 호스트들을 연결한 렌더팜 시스템을 이용하여 제작시간을 줄이는 노력을 하고 있다. 그러나 렌더팜을 구축하는 것은 상당한 비용을 부수적으로 요구하므로 적은 비용으로 고속 렌더링 작업을 수행할 수 있는 고속 렌더러에 대한 요구가 증대하고 있다.

약어 정리

BMRT	Blue Moon Rendering Tools
BRDF	Bidirectional Distribution Function
BSP	Binary Space Partitioning
BSSRDF	Bidirectional Surface Scattering Distribution Function
CG	Computer Graphics
FPU	Floating Point Unit
HDRI	High Dynamic Range Image
LDRI	Low Dynamic Range Image
PRMan	Photo Realistic Render Man
PVM	Parallel Virtual Machine
RISpec	The RenderMan Interface Specification

참 고 문 헌

[1] R.L. Cook, T. Porter, and L. Carpenter, "Distributed Ray Tracing," *ACM Computer Graphics*, Vol.18, July 1984, pp.137-145.

[2] Kari Puli and mark Segal, "Fast Rendering of Subdivision Surfaces," *Proc. of the 7th Eurographics Workshop on Rendering*, 1996, pp.61-70.

[3] http://graphics.uscd.edu/~henrik/images/subsurf.html

[4] Henrik Wann Jensen, "Realistic Image Synthesis Using Photon Mapping," AK Peters, 2001.

[5] Ulf Assarsson & Tomas Akenine-Moller, "A Geometry-based Soft Shadow Volume Algorithm Using Graphics Hardware," *ACM Transactions on Graphics*, Vol.22, No.3, July 2003, pp.511-520.

[6] http://en.wikipedia.org/wiki/Binary_space_partitioning

[7] Steve Upstill, "The Renderman Companion," Addison-Wesley, 1990.

[8] Robert L. Cook, Loren Carpenter, and Edwin Catmull, "The Reyes Rendering Architecture," *ACM Computer Graphics*, Vol.21, No.4, July 1987, pp.95-102.

[9] http://renderman.pixar.com

[10] http://www.mentalimages.com

[11] http://www.povray.org

[12] http://splutterfish.com

[13] http://www.finalrender.com

[14] http://www.chaosgroup.com/software/vray/