오브젝트 전처리에 의한 광선추적법 속도 개선

김범수, 황선태
국민대학교 컴퓨터 학부

Improvement of Ray-Tracing Performance by Using Object Pre-Processing.

Bumsoo Kim, Suntae Hwang
School of Computer Science, Kookmin University

요 약

3D 그래픽의 실사 이미지를 표현하기 위해 사용되는 광선추적법은 알고리즘상 빛과 물체의 충돌을 검사하는데 대부분의 시간이 소모된다. 보다 빠른 광선추적법을 수행하기 위해선 빛과 물체의 충돌 검사시간을 줄이는 것이 요구된다. 이를 위해 본 논문에서는 디자인 시 설정된 오브젝트를 바로 랜더링 하는 방법보다 오브젝트와 카메라의 거리, 오브젝트와 월드의 상대적 크기를 고려한 바운딩 볼륨을 설정하도록 전처리해 빛과 물체의 충돌 검사 시간을 줄임으로서 광선추적법 알고리즘을 빨리 수행하는 방법을 제시한다.

1. 서론

현재의 3D 그래픽은 그 기술적 발달로 인하여 실사와 같은 정교한 이미지 표현이 가능해진지 오래되었다. 실사와 같은 이미지를 표현하기 위해 사용하는 주된 랜더링 방법은 광선추적법과 Radiosity를 혼용하는 것인데, 특히 반사체나 굴절체의 표현을 위해 광선추적법이 사용된다. 광선추적법의 경우 빛과 물체의 반사, 굴절, 그림자를 가장 정교하게 표현할 수 있다는 장점에도 불구하고 연산양의 과다로 현재의 일반적 컴퓨터 시스템에서의 광선추적법의 사용은 매우 느리다. 이러한 현상은 컴퓨터 처리 속도의 증가로 점차 나아지고 있으나, 고전적 광선추적법 알고리즘을 개선한 처리 속도 증가여지는 많이 남아있다.
본 논문의 목표는 광선추적법 연산의 대부분을 차지하는 빛과 물체의 충돌 검사 시간을 줄이는 것에 중점을 두고 있다. 본 논문에서는 3D 오브젝트를 랜더링할 카메라의 위치, 월드의 크기를 고려한 오브젝트 스페이스에서 적절한 바운딩 볼륨을 미리 계산하고 랜더링 시 이를 사용해 빛과 물체의 충돌검사에 소요되는 시간을 줄이는 방법을 제시한다.

2. 고전적 광선추적법 방법

2.1 원리

광선추적법은 전적으로 빛과 물체 표면의 상관 관계를 추적하는 과정이다. 광선추적법은 관찰자의 사점에서 출발한 한 줄기 빛(Ray)이 물체 표면상의 한 점에 충돌한다면, 그 점에서의 광원에 의한 직접 조명과 반사광에 의한 간접조명 그리고 표면 특성에 의한 빛의 변화요소가 조합된 빛의 값을 획득해 화면상에 표시하게 된다. 이러한 일련의 과정에서 광선추적법연산의 대부분은 빛이 어느 물체의 어느 표면(Surface)에 충돌하였는가를 계산하는 것이다.
광선추적법의 연산 과정은 화면상의 모든 화소에서 수행되며 빛이 물체와 충돌한다면 설정된 충돌 횟수를 만족 시키지 못할 때 충돌 점에서 다시 빛을 쏜 것으로 계산해 최종 충돌점의 빛의 값을 얻는다. 따라서 고전적 광선추적법 알고리즘은 방향성과 시작점을 갖는 빛과 그 빛의 관점에서 보이는 월드의 모든 물체를 차례로 비교하므로 하나의 빛과 물체들의 충돌을 따지는 모듈로 구현이 가능하며 물체 표면에서 반사된 빛은 구현된 모듈을 재귀호출 함으로써 해결된다. 광선추적법의 재귀적 추적 과정의 슈도 코드는 다음과 같다.

```
prodedure TraceRay(StartPoint, Direction: Vector, Depth: Integer,
                   LocalColor:Color)

if Depth>MaxDepth then Color = GlobalColor
else
begin
{Intersect ray with all 오브젝트s and find intersection point}
if(no intersection) then Color = BackGround-Color
else
begin
LocalColor = {contribution of local color model at intersection point}
{Calculate direction of  reflected ray}
TraceRay(Intersection-point, reflected-direction, Depth+1, reflected-color)
{Calculate direction of  transmitted ray}
TraceRay(Intersection-point,transmitted-direction,Depth+1, transmitted-color)
{Combine all of colors( reflected, local, transmitted )}
end
end
end
```

그림.1 고전 광선추적법 알고리즘의 슈도 코드

2.2 문제점

광선추적법은 빛과 표면의 충돌을 알아내는데 필요한 수학적 연산을 계산하는데 대부분의 시간을 소비하며 관찰자의 시각으로부터 출발한 대부분의 빛은 물체 표면과 충돌하지 않고 공간 속으로 직행한다.
광선추적법에서 논의되는 문제로 그림 2와 같은 오브젝트를 생각할 수 있다. 충돌점을 찾기 위해 오브젝트의 모든 Surface의 평면 방정식에 빛의 직선 방정식을 대입하여 알아낸 충돌점의 좌표를 사용해 충돌점의 Surface포함 여부를 알아낼 수 있으나, 연산량을 줄이기 위해 편의상 그림2의 구와 같은 바운딩 볼륨을 설정하게 된다.
이 바운딩 볼륨은 오브젝트에 맞추어진 구나 직 육면체로 설정하는 것이 보통이다. 광선추적법에선 오브젝트를 구성하는 Surface와 빛의 충돌을 직접 계산하기 보다는 우선 보다 큰 단위의 바운딩 볼륨과의 충돌을 검사하고 충돌이 확인된 경우 오브젝트의 모든 Surface와의 충돌점을 계산하게 된다. 그림 2에서 관찰자의 시점에서 쏘아진 빛의 대부분은 바운딩 볼륨인 구와는 충돌하지만 오브젝트와 충돌하지는 않는다. 고전적 알고리즘의 단순한 바운딩 볼륨에 의한 오브젝트의 기술 구조는 충돌하지 않는 빛을 찾아내기까지 불필요한 연산을 수행하므로 충돌하지 않는 빛을 빨리 추출하기 위해 각 오브젝트에 적절한 다면체나 구의 바운딩 볼륨을 한 개 이상 설정하고 다수의 바운딩 볼륨에 의한 오버헤드를 최소화하는 구조로 오브젝트를 정의할 때 보다 빠른 광선추적법을 수행할 수 있다.

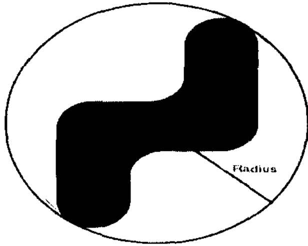

그림 2. 오브젝트와 구 바운딩 볼륨

3. 바운딩 볼륨 생성 기법

전처리는 오브젝트의 모양을 분석해 광선추적법을 하기위한 구조로 오브젝트 구조를 설정한다. 일반적인 3D 오브젝트를 간략히 나타내면 다음과 같은 구조를 갖는다.

```
Struct  Tag3DOBJ
{
    int Vertices              오브젝트를 구성하는 점 좌표 개수
    int Surfaces              오브젝트를 구성하는 폴리곤 총수
    VERTEX  *VertexList       오브젝트를 구성하는 점 좌표 배열
    SURFACE  *FaceList        오브젝트를 구성하는 폴리곤 배열
    Float      Radius         바운딩 볼륨의 반지름
}
```

위의 구조는 바운딩 볼륨을 오브젝트의 중심에서 가장 먼 점까지의 거리를 Radius로 하는 구를 미리 계산 함으로써 기본적인 전처리를 하고 광선추적법을 할 수 있다. 이러한 구조는 그림 2의 예와 같이 오브젝트를 포함하지 않는 구의 부피가 대부분으로, 바운딩 볼륨인 구에 빛이 충돌하더라도 오브젝트에 충돌하지는 않는다.
이러한 바운딩 볼륨의 설정은 광선추적법에 효율적으로 사용할 수 없으며, 오브젝트의 모든 폴리곤을 포함하며 빈공간을 최소화 할 수 있는 다면체 바운딩 볼륨의 설정이 요구된다. 오브젝트를 분석해 바운딩 볼륨을 설정할 때의 오버헤드로 랜더링 시간이 바운딩 볼륨을 설정하지 않을 경우보다 오래 걸릴 수 있으나, 전처리에 의해 미리 계산된 바운딩 볼륨은 연속적인 동영상 랜더링 시 재사용 가능하다. 미리 계산된 오브젝트의 바운딩 볼륨은 하나의 장면을 랜더링할 때 보다 동영상이나, 자유 카메라에 대한 광선추적법에 유리하다. 바운딩 볼륨을 전처리한 오브젝트의 구조는 다음과 같다.
간략화를 위해 위치, 방향, 재질의 종류 등은 생략하였다.

```
Struct TagBoundVolume
{
    int    Type        바운딩 볼륨의 종류(구, 피라미드, 직육면체 등)
    int    Vertices    바운딩 볼륨을 구성하는 점 좌표 수
    int    Surfaces    바운딩 볼륨을 구성하는 폴리곤 수
    int    iStartPoly  바운딩 볼륨에 포함되는 오브젝트 폴리곤 리스트
                       의 첫번째 인덱스
    int    iEndPoly    바운딩 볼륨에 포함되는 오브젝트 폴리곤 리스트의
                       마지막 인덱스
    float       Radius      바운딩 볼륨의 반지름(구)
    VERTEX   *VertexList  바운딩 볼륨을 구성하는 점 좌표 배열
    SURFACE  *FaceList    바운딩 볼륨을 구성하는 폴리곤 배열
}VOLUME
Struct Tag3DOBJ
{
    int Vertices           오브젝트를 구성하는 점 좌표 수
    int Surfaces           오브젝트를 구성하는 폴리곤 수
    int Volumes            오브젝트를 포함하는 바운딩 볼륨의 수
    VERTEX   *VertexList   오브젝트를 구성하는 점 좌표 배열
    SURFACE  *FaceList     오브젝트를 구성하는 폴리곤 배열
    VOLUME   *VolumeList   오브젝트를 포함하는 바운딩 볼륨의 배열
}
```

위와 같은 오브젝트구조를 사용해 표1의 플로우를 따라 전처리된 바운딩 볼륨을 설정한다.

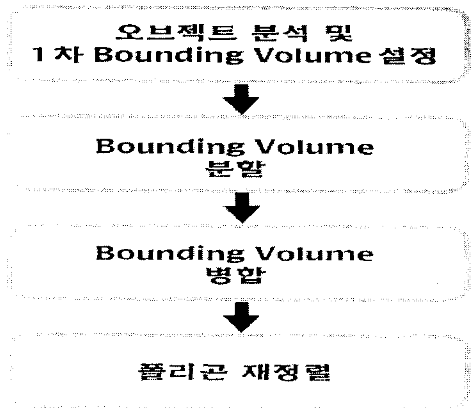

표1. 광선추적법을 위한 오브젝트 전처리 단계

3.1 오브젝트 분석 및 1차 바운딩 볼륨 설정

그림 3-1과 같은 3D 스페이스상의 오브젝트를 가정할 때, 바운딩 볼륨을 설정하기 위해 그림 3-2와 같이 오브젝트 중심점으로부터 카메라까지의 거리, 월드상에서 오브젝트가 차지하는 크기를 고려해 분할할 최소 단위의 큐빅의 크기를 결정하고 오브젝트의 중심으로부터 X,Y, Z축 방향으로 순차적인 최소 큐빅을 오브젝트의 모든 폴리곤이 포함될 때까지 확장해 나간다. 최소 큐빅으로 분할된 오브젝트를 그림 3-3과 같이 바운딩 볼륨이 오브젝트의 최소 큐빅을 모두 포함할때 까지 오브젝트의 최외각 폴리곤을 따라 X, Y, Z축으로 순차적으로 이동하며 큐빅을 합쳐나가 직육면체의 1차 바운딩 볼륨을 설정한다.
그림3-1의 오브젝트는 그림 3-3처럼 3개의 1차 바운딩 볼륨을 갖게 된다. 1차 바운딩 볼륨의 최소 부피는 그림 3-2의 최소 큐빅의 부피와 같을 수 있다.

3.2 바운딩 볼륨 분할

1차 바운딩 볼륨에 포함된 폴리곤 수가 카메라와의 거리, 월드상에서 오브젝트가 차지하는 크기를 고려해 일정 수보다 많거나 1차 바운딩 볼륨의 빈공간이 많을 경우 1차 바운딩 볼륨을 분할한다. 분할된 바운딩 볼륨은 1차 바운딩 볼륨의 모든 폴리곤을 포함하도록 한다.

3.3 바운딩 볼륨 병합

1차 바운딩 볼륨과 1차 바운딩 볼륨이 분할된 바운딩 볼륨에 포함된 오브젝트의 폴리곤 수를 검사하여 바운딩 볼륨에 포함된 폴리곤 수가 오브젝트 중심점으로부터 카메라까지의 거리, 월드상에서 오브젝트가 차지하는 크기를 고려한 폴리곤 수보다 적을 경우 인접한 바운딩 볼륨을 합쳐 두 바운딩 볼륨을 포함하는 최소 부피의 직육면체로 바운딩 볼륨을 병합한다. 바운딩 볼륨 병합의 경우 병합된 바운딩 볼륨의 부피가 병합전의 바운딩 볼륨의 합 보다 일정 비율이상 크면 병합하지 않는다.

3.4 Polygon정렬

디자인 시 무작위로 배치되어 있는 오브젝트의 폴리곤 리스트를 각각의 바운딩 볼륨에 포함되는 순서로 재배치한다. 바운딩 볼륨의 경계에서 각 바운딩 볼륨에 공유되는 폴리곤은 중복시켜 배치한다. 표 1과 같은 과정을 거쳐 바운딩 볼륨이 설정되고 각 바운딩 볼륨에 해당하는 폴리곤을 정렬하면 빛과 오브젝트의 충돌검사 시 오브젝트의 폴리곤과 충돌했는지 검사하기 보다는 우선 설정된 바운딩 볼륨과 빛이 충돌 했는가를 판별하고, 충돌한 경우 해당 바운딩 볼륨에 포함된 폴리곤 리스트를 검색해 충돌한 폴리곤과 충돌 좌표를 찾아내게 된다.

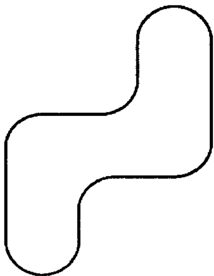

그림 3-1

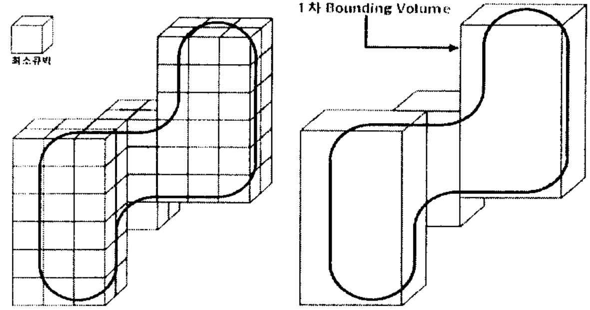

그림 3-2

그림 3-3

4. 결론

기존 광선추적법 알고리즘은 기본적인 바운딩 볼륨을 사용함으로서 빛과 오브젝트의 Hit-Testing을 하는 과정에서 충돌하지 않는 빛을 빨리 추출하지 못하는 단점을 갖고 있다. 본 논문에서는 기존 알고리즘의 이러한 단점을 보완하여 구조화된 오브젝트의 자료 구조를 제시하고 보다 빠른 Hit-Testing을 위한 바운딩 볼륨 생성 방법을 사용해 전처리된 오브젝트를 랜더링하는 방법을 제시하였다.
디자인 시 모델링된 오브젝트를 전처리 함으로서 전처리에 소요되는 오버헤드와 메모리 사용량의 증가에도 불구하고 전처리 데이터를 매 프레임마다 재 사용함으로서 연속적인 동영상의 랜더링이나 게임과 같은 실시간 랜더러에서 광선추적법을 보다 빨리 할 수 있을 것으로 기대된다. 향후 과제는 현재 제작중인 광선추적법 랜더러에 바운딩 볼륨 전처리를 응용하는 것이다.
동일한 오브젝트의 전처리를 하는 랜더러와 일반적인 랜더링 방법과 수행 속도를 비교한 결과를 토대로 더 빠른 광선추적법을 위한 알고리즘 개선에 대한 연구가 필요하다.

참고 문헌

[1] F.S.Hill.Jr: *Computer Graphics*, McMillan 1995
[2] Foley, van Dam, Feiner, Hughes: *Computer Graphics Principle and Practice*, 2nd Edition in C, Addison Wesley,1995
[3] Katherine Schowalter: *Radiosity*, John Wiley & Sons, 1994
[4] Craig A.Lindley: *Practical Ray Tracing in C*, Prentice Hall, 1992
[5] Watkins, Coy, Finlay: *Photorealism and Ray Tracing in C*, Wiley, 1992