Ray casting algo

Ray casting을 간단하게 하기 위해서 수많은 도형이 들어와 있을 때 어떻게 표현할 것인지.

방법론

1. 트리 형태의 도형 구조
2. 각 트리의 계층에서 존재하는 것처럼 도형의 특정을 계층 구조로 한다. Hit에 대한 연산을 단계적으로 반복해야 하지만, 한 형태에 곡률이 많아 하나의 도형을 만들 경우에 해당 부분을 크게 줄일 수 있다. 예를 들어 각 도형이 2개 이상의 삼각형을 가지고 있을 때 훨씬 효율적이다. 때문에 이를 사용해 볼 예정이다.
3. 가장 간단한 도형의 구조인 구형 혹은 타원형으로 만들어 해당 부분의 Hit가 있는지 확인한다.
4. 만약 해당 트리에 Hit가 발생할 경우 해당 트리 내부의 자식 노드를 찾는다.
5. 자식 노드 중 군집화 되어 있는 것 중 퍼스트 Hit를 찾는다.
6. 최종 자식 노드의 삼각형에 색과 음영 등을 뽑는다.
7. 과연 이 정도 했을 때 얼마나 속도 개선이 되나?

해당 방법론의 핵심

1. 각 노드에서 Hit box가 얼마나 보편화 할지?
   1. Hit box가 간단할 경우, 단계별 히트박스 계산은 되게 간단하게 할 수 있기에 속도면으로 큰 장점을 가져올 수 있다. 반면에 넉넉하게 히트박스를 했을 경우 복잡한 형태의 도형일 경우 실제로는 hit가 아님에도 hit라고 오류로 포함될 경우가 있으며, 특히 풀이 무성한 나무의 경우 해당 나뭇잎을 하나의 도형으로 만들 경우 여러 도형의 히트 여부를 물어야 하며, 퍼스트 히트인지, 점증적으로 보아야 해서 최악으로는 모든 행동을 하며, 그에 2^n-1만큼 추가로 하는 경우가 생길 수도 있다.
   2. Hit box가 복잡하게 할 경우 각각에 히트박스를 계산하는데 시간이 걸리기에 각각의 히트박스를 계산하는데 시간이 걸린다. 하지만 반복해서 계산해야 할 가능성이 낮기에, 반복 가능성은 낮다. 그렇지만 히트박스 계산에 오래 걸린다면, 삼각형으로 형태를 표현하는 것에 대해 장점이 사라지게 된다.
   3. 즉 해당 두 사이에서 어떻게 효율적으로 표현할지 고민해 봐야 한다. 하지만 중요한 것은 hit box의 경우 계산이 단순한, 직육면체, 구, 타원체 정도가 한계이다. 다른 것들의 경우 계산의 난이도가 있기에 넘어가야 한다.
   4. Hit box가 over하는 것에는 큰 문제가 없지만, hit box가 실제 도형을 포함하지 않을 경우에 즉 loss가 발생할 경우에는 큰 문제가 될 수 있기에 반드시 모든 도형을 감쌀 수 있어야 한다.
2. 트리 탐색에서 우선권을 어떻게 할 것인지?
   1. 트리의 탐색권한을 가져갈 때, 어떤식으로 우선권을 가져갈지는 중요하다.
   2. 시점에 따라 도형의 거리에 따라 트리의 순서를 바꾼다.
      1. 장점 : hit box의 overflow가 났다고 하여도, 전체 탐색할 필요 없이 트리의 순서대로 탐색을 하면 된다. 즉 한번 검색한 내용에 대해서 재검색을 하여, 확인할 필요가 없다.
      2. 단점 : 시점에 변화에 따라 지속적인 정렬이 있어야 한다. 즉 시점 전환이 매우 잦을 경우 해당 방법의 경우 매우 비효율적일 수가 있다. (단, 시점은 고정고정 있을 때, 신규가 생기는 것은 추가가 어려운 것은 아니다.)
   3. 트리의 순서에 변경 없이 매 검색상에서 트리의 순서에 변경이 없다.
      1. 장점 : 시점이 바뀌어도 탐색에 큰 문제가 없다.
      2. 단점 : 매번 hit miss가 날 때마다 반복적으로 탐색해야한다. 즉 최악의 경우 탐색은 시그마n이다.
   4. 트리 순서를 지속적으로 바뀔 때 거리 측정을 어떻게 해야할 지도 문제이다. 우선 순서에 대해서 가장 가까운 표면으로 거리를 만들지, 중심으로 거리를 해야할지
3. 삼각형으로 도형을 표현할 때, 삼각형의 모서리 부분을 어떻게 표현할지?
   1. 각 표면에서 삼각형으로 도형을 표현할 때, 모서리부분이 튀어나오기에 해당 부분에 하이라이트가 생기어 색이 이상하게 만들어질 수도 있다. 해당 상황에서 어떻게 하이라이트를 처리하고, 스무스한 도형을 만들 것인지.
   2. 특히 해당 표면이 반사 혹은 굴절이 일어나는 유리 재질일 때 어떻게 해당 부분을 표현하여, 무리 없는 상황을 만들지에 대해 고민이 필요하다.
      1. 해당 부분을 베지어 곡선으로 표현할 경우
         1. 베지어 곡선 공식 이용
      2. 클로소이드 곡선으로 표현할 경우
         1. 클로소이드 곡선 공식 이용
      3. 일반 곡선으로 표현
         1. 양끝점으로 호를 만든다.
4. 기본적 탐색에 어떤 기본도형으로 바운더리를 만들지?
   1. 구
      1. 하나의 큰 구로서 표현이 되기에 많은 손실이 일어난다.
      2. 계산이 매우 쉬워 간단하다.
   2. 타원
      1. 타원은 적극적인 도형을 표현할 수는 있으나, 계산과정이 복잡하여, 컴퓨팅 파워에 어려움을 준다.
   3. 직육면체
      1. 도형을 적극적으로 표현할 수 있으며, 계산 과정상에 큰 어려움은 없다. 하지만, 도형의 특성상 법선 벡터의 구성을 지속적으로 해야한다.
   4. 구 집합
      1. 기본적인 구의 집합으로 하나의 규모를 표현하여, 적극적인 표현은 가능하지만, 길고, 얇은 도형의 경우에는 계산상에 어려움이 있을 수 있다. 특히 구조가 복잡할 경우 너무나 많은 구집합이 필요하고, 이는 실질적으로 탐색 속도를 저해시킬 수 있다.
      2. 더하여, 긴 막대기의 경우 구집합으로는 여러 개의 집합으로 표현해야 하지만, 실질적인 도형은 긴 원통형 형태 하나이다. 때문에 오히려 바운더리로 만들었을 때 손해가 더 나타난다.
   5. 원기둥
      1. 원기둥의 추정은 직선에 거리를 추정한다. 하지만 거리의 최단거리를 구하기 위해서 직선을 뽑아서 서로 수직임을 증명하고 해당 점에서의 거리측정과 그 거리의 정도를 확인해야 한다.
      2. 두 직선에 수직 백터 꼬인 위치의 두 직선의 방향벡터의 외적(정사영)
   6. Box
      1. 크기의 손해를 보더라도 계산상의 손해를 감수할 필요가 없다. 다른 것들의 경우 곱셈을 필수적으로 해야하지만, box로 한다면, 그 계산은 비교만으로 정리가 된다. 총 6번의 비교를 하면, 해당 값을 구할 수 있으며, 계산의 리소스는 곱셈 하나보다 작다. 때문에 더 빠른 작업이 가능하다.
      2. 어셈블리에서 중수의 곱셈을 단순 더하기가 아닌 시프트와 덧셈으로 처리한다면, 얼마나 빨라질까?