PC Cluster 상에서의 병렬 광선 추적 알고리즘의 성능

임동익*, 이효종, 임범현
전북대학교 전자공학과

e-mail:{dilim, hlee, bhlim}@sel.chonbuk.ac.kr

Performance of Parallel Ray Tracing Algorithm on PC Cluster

Dong Ick Im, Hyo Jong Lee, Bum Hyun Lim
Dept of Electronics Engineering, Chonbuk National University

요약

광선 추적 기법은 컴퓨터를 활용하여 사진과 같은 고해상도의 영상을 얻어내기 위한 렌더링 기법 중 하나이다. 그러나 이 기법은 이미지를 생성할 때 각 점마다 시뮬레이션을 하여 계산해 내므로 점의 수에 따른 계산량이 증가되고 그에 따른 계산 시간이 많이 소요된다는 단점이 있다. 이러한 많은 계산량을 처리하기 위해 병렬처리 기법을 활용할 수 있다. 본 논문에서는 MPI(Message Passing Interface)를 이용한 병렬 광선 추적 기법을 제시하고 그러한 기법을 여러대의 PC를 이용한 PC Clustering 기법에 적용시켜봄으로써 복잡한 계산에 소요되는 시간을 단축시키고자 하였다. 또한 작업의 크기의 변화에 따른 작업 시간과 노드 수의 증가에 따른 속도 향상률을 알아보았다. 이러한 실험을 위해 병렬 프로그래밍 도구로는 Windows NT 기반의 MPICH를 사용하였고 노드의 수는 3대에서 30대까지 증가시켰다. 노드의 수가 증가함에 따라 렌더링에 소요되는 시간이 선형적으로 감소함을 알 수 있었다.

1. 서론

병렬처리란 여러 개의 프로세서를 동시에 이용해서 연산능력을 극대화하여 성능의 한계를 극복하기 위한 방법인데, CPU를 여러 개 사용함으로써 그 성능의 몇 배에 달하는 효과를 낼 수 있다. 물론 중대형 컴퓨터를 이용하면 되겠지만 비용적인 측면에서 무리가 있으므로 여러 대의 일반 컴퓨터를 병렬로 연결하여 Clustering 함으로써 적은 비용으로 높은 효과를 낼 수 있다. 레이 트레이싱과 같이 렌더링 시에 시간이 많이 소요되는 작업 시간을 획기적으로 단축시킬 수 있다.

레이 트레이싱(Ray Tracing)이란 단어의 뜻 그대로 광선 추적 기법이라고 한다. 일반적인 그래픽이 현실감이 떨어지는 이유는 빛에 대한 반사 등이 고려되지 않아서인데, 그래픽 영상에 좀더 현실감 있는 입체 영상을 제작하기 위해서 표면의 반사도나, 투명도, 빛의 굴절률 등을 지정하여 빛의 반사되는 과정을 추적하여 표현하는 것을 말한다. 이러한 광선 추적 기법에서는 시점과 벽, 대상 물체등의 요소를 지정해 주고 주어진 시점에서부터 출발한 빛이 어떠한 경로로 반사되고 흡수되고, 굴절하는지를 수학적으로 계산하여 그 값을 영상 데이터로 출력하게 된다. 눈에서 출발하여 각 화소별로 날아가는 광선의 방향 및 세기 계산은 서로 다른 화소간에는 서로 아무런 영향을 미치지 않는다. 이러한 부분은 충분히 병렬화가 가능한 부분이다.

광선 추적 기법은 엄청난 계산 시간을 요한다. 이는 첫째, 주어진 광선에 대해 그 광선이 처음으로 마주치게 되는 면을 찾아내야 한다. 둘째, 레이 트리의 하부 노드에 있는 모든 반사광, 투과광의 세기를 계산해야 상위 노드의 색을 결정할 수 있다. 셋째, 화면을 구성하는 모든 개별 화소에 대해 이러한 작업을 가해야 한다. 이러한 이유로 광선 추적 기법은 많은 계산 시간을 필요로 하고 또한 영상이 복잡한

물체들로 구성되었을 때에는 더 많은 계산 시간이 필요하게 된다.

이러한 계산 시간을 단축하기 위해 광선 추적 알고리즘이 이미 발표되었다[1,3]. 또한 광선이 이동하는 방향은 독립적으로 측정 가능하고 이러한 부분을 병렬 환경에서 추적하는 연구도 발표되었다[5,6,7,8].

한편 이 논문에서는 이러한 계산 시간 단축을 위해 병렬화한 병렬 광선 추적 알고리즘을 PC상에서 Clustering 기법을 이용하여 구현하였다. 슈퍼컴퓨터를 이용한 병렬처리를 할 수도 있지만 보다 저렴한 가격의 PC를 여러 대 활용함으로써 얻는 비용적인 측면의 절감 효과 또한 크고 보다 더 실용적이라는 판단하에 PC Clustering을 본 논문에서 활용하였다. 본 논문에서는 최대 30대의 PC상에서 MPI(Message Passing Interface)를 이용한 Windows NT용 MPICH를 이용하여 실험하였다. 30대의 PC를 3대에서 30대까지 점차적으로 노드의 수를 증가시키고 그에 따른 성능 향상을 측정하였다. 본 논문은 2절에서는 순차적 광선 추적 알고리즘에 대해서 기술하고, 3절에서는 병렬 광선 추적 알고리즘을 설명하였다. 4절에서는 실험 결과 값을, 마지막으로 5절에서는 결론을 기술하였다.

2. 순차적 광선 추적 알고리즘

광선 추적 기법이란 관찰자의 눈에 들어오는 광선을 찾아 그 광선의 색을 화면에 표시해주는 방법이다. 광선을 추적하는 방법에는 크게 2가지가 있는데, 하나는 광원이 내보내는 빛의 방향을 세분하여 추적하는 정방향 광선 추적법이고, 다른 하나는 관찰자의 눈에서 광원 방향으로 시선을 추적하는 역방향 광선 추적법이다. 정방향 광선 추적법은 빛이 지나가는 모든 경로를 추적하기 때문에 자연 현상을 정확하게 영상화 할 수 있다는 장점이 있는 반면에 반사되는 빛들이 산란되는 경우 계산 시간에 많은 낭비를 초래할 수 있다. 반면 역방향 광선 추적은 관찰자의 시야를 따라서 빛을 내보낸 실제 물체를 찾아내는 방법이다. 이 빛이 실제 물체들과 부딪히는 면을 계산하고, 부딪히는 경우 그 물체의 특성에 따라 반사, 굴절 및 투과된 빛들은 또 다시 다른 물체와 부딪히는지를 계산하고 이와 같은 과정을 반복한다. 이런 과정을 반복하다 보면 결국 빛의 세기가 임계값 이하로 약해지거나 빛이 어떤 물체와도 부딪히지 않을 때 그 화면에 대한 점의 빛 역 추적 과정은 종료된다. 이 방법을 화면상에 존재하는 모든 점에 적용해서 완전한 영상을 구성하게 된다. 일반적으로 광선 추적 기법은 정방향 광선 추적법이 아닌 역방향 광선 추적법을 의미한다.

3. 병렬 광선 추적 알고리즘

병렬 처리 기술 중 메시지 전달을 기반으로 하는 소프트웨어로는 크게 MPI(Message Passing Interface)와 PVM(Parallel Virtual Machine)이 있다. 역사적으로 볼 때 PVM이 먼저 개발되었고 워크스테이션의 네트웍에 맞게 설계 되었다. 반면 MPI는 그와는 달리 많은 하드웨어 판매자에 의해 지원되고 있으며 PVM보다 많은 기능을 제공한다. 한편 PVM의 경우 메시지를 전송하기 전에 pack(일종의 압축)함으로 서로 다른 이종간(heteromachine)에 전송이 자유롭게 이루어지는 특성이 있는 반면 MPI는 이러한 과정이 없으므로 이종간의 메지시 전송이 자유롭지는 않지만 MPI_xx라는 동일한 형(type)을 가짐으로 이종간에 전송도 가능해지고 있다. pack을 하지 않음으로써 그 만큼의 전송으로 허비되는 시간을 없애므로 시간을 단축할 수 있다.

광선 추적 알고리즘에서 광원에서부터 나오는 빛의 이동 경로들은 각각 독립적이기 때문에 이 부분을 병렬화 함으로써 계산 시간을 획기적으로 단축시킬 수 있다.

본 논문에서 사용된 MPI는 (그림 1)의 의사코드와 같이 Master 노드와 Slave 노드 사이에 메시지를 주고받음으로써 병렬처리를 하게 된다. Master 노드의 경우 작업을 관장하는 역할을 하게 되며 Slave 노드의 경우 직접적인 계산 등의 작업을 하게 된다. 병렬 광선 추적 알고리즘의 경우 (그림 1)에서 볼 수 있듯이 입력된 영상으로부터 Master 노드에서 영상의 정보를 획득하고 노드 수에 맞게 작업을 분류한다. 그리고 초기화 된 상태로 대기 상태에 있는 Slave 노드에 메시지를 보냄으로써 작업을 요청하게 된다. 그러면 작업을 요청 받은 Slave 노드들은 병렬화된 광선 추적 과정을 각각 맡아서 진행하게 되고 거기에서 산출된 결과값을 Master 노드에 다시 메지시를 통해서 보내게 된다. 그렇게 해서 모든 Slave 노드들로부터 다시 수집된 결과값을 종합함으로써 최종적인 이미지를 생성하게 된다. 이 과정에서 Master 노드는 전반적인 병렬 처리 순서를 정하고 Slave 노드들에게 일을 분배하는 역할을 하게 되며 Slave 노드들이 작업을 마치게 되면 그 결과값을 다시 수집하여 최종적인 이미지를 생성하게 된다.

반면 Slave 노드들은 Master 노드로부터 분류되어 넘겨진 작업을 수행하게 되며 그 결과 값을 다시 Master 노드에게 넘겨주는 역할을 하게 된다. 또한 이 과정에서 Master 노드는 만일 Slave 노드 중 작업이 마쳐진 상태의 노드에게는 다른 노드의 일처리 유무와 관계없이 다음 작업을 할당함으로써 보다 더 높은 성능을 보여주게 된다. 본 논문에서는 위의 알고리즘을 활용하여 노드 수를 3개에서 30개까지 증가시켰으며 그에 따른 성능 향상과 속도 증가율을 살펴보았다.

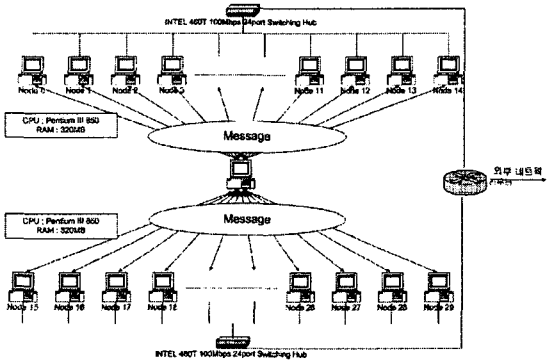
```
Process Program MPI Main
BEGIN
   Initialize MPI
   Initialize Variables
   Parse Program Option Lines

   if (Master Process)
       Initialize MPI Master Process
       Controlling the jobs

   if (Slave Process) {
      Initialize Slave Process
      while (end of frame)
         Rendering Frame assigned into each
                                  Slave Process
   }

   Terminate MPI
END
```

(그림 1) MPI 병렬화 부분 의사 코드

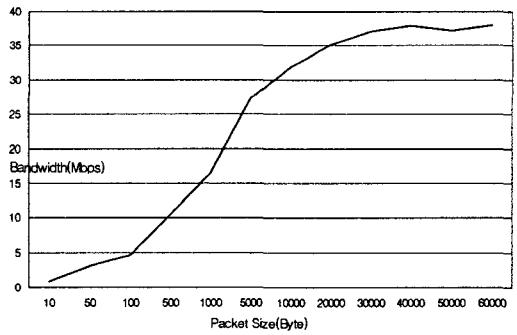
4. 실험 결과

본 논문의 실험을 위해서 (그림 2)와 같이 30대의 PC를 100Mbps의 초고속 Ethernet으로 Clustering하여 이용하였고 실험 결과 실제 PC 사이의 Bandwidth는 (그림 3)과 같이 약 40Mbps이다. 각 PC의 사양은 Pentium-III 850MHz CPU, 320MB Memory이다. 또한 병렬 처리 관련 프로그램은 MPI (Message Passing Interface)를 사용하여 메시지를 이용한 통신을 하였고 입력 이미지는 체스판이 사용되었으며 결과물은 1024*768의 크기에 24bit 해상도를 가진 영상이 생성되었다.

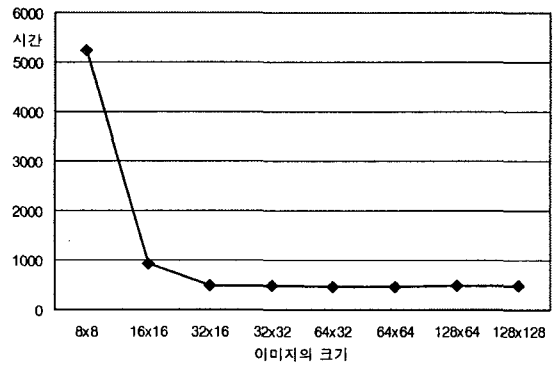
(그림 4)는 체스판을 Slave 노드에서 처리하는 영상의 크기에 변화를 주어가면서 그에 따른 작업 시간을 측정한 것이다. 노드는 10개를 활용하였으며 이미지의 크기를 8x8에서부터 128x128까지 증가시켜가면서 실험하였다. 분할 이미지의 크기가 작을 경우 매우 많은 작업 시간이 필요하게 되며 분할 이미지의 크기가 64x32일때 렌더링 시간이 가장 적게 걸림을 알 수 있다.



(그림 2) MPI 기반의 실험 환경



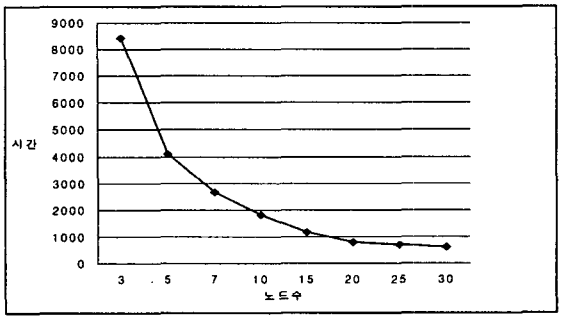
(그림 3) Bandwidth 측정 결과



(그림 4) 작업 크기에 따른 광선 추적 시간

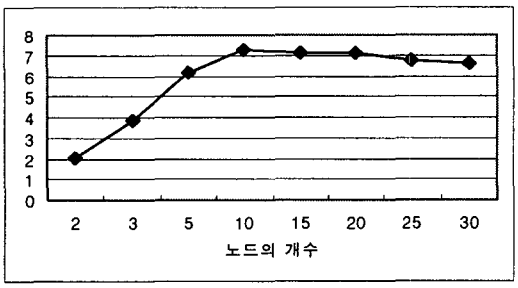
(그림 5)는 병렬 광선 추적 알고리즘을 이용하여 체스판을 만들어내는데 소비한 시간의 그래프이다.

Slave 노드수를 3개에서부터 30개까지 점진적으로 증가시켜가면서 실험을 진행하였고, 노드의 수가 증가함에 따라 계산 시간도 단축되었다. 노드의 수가 20개 정도까지는 현저한 계산 시간 단축이 있었지만 그 이후에는 시간 단축이 그다지 많이 되지는 않았다.

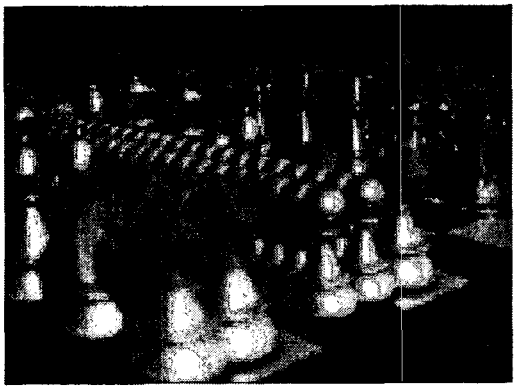


(그림 5) 노드수의 증가에 따른 작업 시간

(그림 6)은 노드 개수의 증가에 따른 속도 증가율을 나타낸 그래프이다. 노드의 개수가 증가함에 따라 향상률로 증가하다가 14개가 되었을 때는 오히려 속도 향상률이 떨어지는 결과를 보였다. 이는 통신량의 증가로 인해 렌더링시 성능 저하가 나타났기 때문이다.



(그림 6) 노드의 개수 증가에 따른 속도 향상률



(그림 7) 실험에 사용된 체스판

5. 결론

광선 추적 알고리즘은 많은 계산량을 필요로 한다. 때문에 많은 계산 시간을 병렬성을 이용해서 단축하고자 하는 연구가 많이 이루어졌고, 본 논문에서는 PC Clustering을 이용한 병렬성을 이용하고자 하였다. Master 노드와 Slave 노드 사이에서 전체 작업을 Master 노드가 조율하고 Slave 노드는 실제적인 계산 작업을 분산 처리함으로써 계산 시간의 단축을 꾀할 수 있었다. 또한 영상의 복잡도에 비해 너무 많은 노드를 이용해서 계산을 하는 것은 오히려 통신량의 증가로 인한 비효율적인 계산 시간을 보여주었다.

참고문헌

[1] A S Glassner. Space subdivision for fast ray tracing. IEEE Computer Graphics and Applications, vol.4, no 10. pp 15-22, Oct, 1984

[2] N Wilt. Object-Oriented Ray Tracing in C++. The Wait Group, 1994

[3] J Arvo and D Kirk. Fast ray tracing by ray classfication. ACM Computer Graphics, vol. 21, no. 4, pp. 55-64, Jul. 1987

[4] POV-Ray Team, Persistence of Vision Ray Tracer (Pov-Ray) Version 2.0 User's Documentation, Private Publication. 1993

[5] B Freisleben, D Hartmann, and T Kielmann. Parallel raytracing: a case study on partitioning and scheduling on workstation clusters. Conference on System Science, 1:596 605, 1997

[6] Chang-Geun Kwon, Hyo-Kyung sung, and Heung-Moon Choi. An implementation of a parallel raytracing algorithm on hybrid parall architecutre. IEEE Conference on Acoustics, Speech and Signal Processing, 3:1745 1748, August 1998.

[7]W Lefer. An efficient parallel ray tracing scheme for distributed memory parallel computers. Parallel Rendering Symposium, pages 77 80, August 1993

[8]S Gaudet, R Hobson, P Chilka, and T Cavert. Multiprocessor experiments for high-speed ray tracing. Transactions on Graphics, 3(7):151 179, July 1988.