고속 광선 추적법을 위한 멀티프로세서에서의 부하분산방식

권 오 봉†

요 약

광선 추적법(Ray-tracing)은 매우 사실적(photo-realistic)인 영상을 생성하나 처리시간이 방대하여 고속화가 과제로 되어 있다. 광선 추적법의 고속화는 알고리즘의 개선에 의한 방법과 멀티프로세서를 이용한 방법이 있다. 멀티프로세서를 이용하여 광선 추적법을 고속화하기 위해서는 효율적인 부하분산이 필요하다. 본 논문에서는 기존의 광선 추적법의 부하분산법과 멀티프로세서의 스케쥴링 기법에 대하여 고찰한 다음에 이것들을 이용한 부하분산방식을 제안하고, 제한한 방식을 멀티프로세서 상에서 구현하고 평가한다. 평가 결과는 제안한 방식이 동적부하분산법의 부하 불균형 문제를 해결하고, 정적 부하 분산법 중에서는 스캔라인 방식과 도트 방식이 효과적인 부하 분산을 보이고 있다.

Load Distribution Strategies for High Speed Ray-Tracing on Multiprocessors

Ou-Bong Gwun†

ABSTRACT

Ray-tracing algorithm can synthesize photo-realistic image, but its computational cost is very high. Fast image synthesis based on ray-tracing is one of the most important topics in computer graphics. There are two methods for high speed ray-tracing: One is based on algorithm, and the other is based on multiprocessor. We need balanced load distribution to utilize multiprocessor for high speed ray-tracing. First this paper discusses various load distribution and scheduling of multiprocessor for high speed ray-tracing. Then this paper proposes load distribution strategies based on them, implements and evaluates it on multiprocessor. The experment results show that the proposed method can solve the unbalanced load problem of dynamic load distribution, and scan line method and dot method among a kind of static load distribution strategies disperse the load efficiently.

1. 서 론

3차원의 물체모델을 컴퓨터 안에 만들어 이것을 2차원 스크린에 투영하여 시뮬레이트하는 3차원 컴퓨터 그래픽스(3D CG)는 에니메이션(animation), 게임, 광고 등 많은 분야에 사용된다. 3D CG의 처리는 크게 물체를 컴퓨터 내부에 만드는 모델링과 이것을 스크린에 영상화하는 렌더링으로 나누어지는데 일반적으로 모델링 작업은 주로 인간에 의해서 이루어지나, 렌더링은 컴퓨터에 의해서 처리되어 빠른 응답성을 요구한다. 렌더링의 기법으로서는 퐁세이딩(Phong shading), 광선 추적법(Ray-tracing), 레디오시티(Radiosity) 등이 있으나 광선 추적법은 알고리즘이 간단하고, 생성된

※ 본 연구는 정보통신부의 정보통신 우수 시범학교 지원사업에 의하여 수행되었습니다.
† 정 회 원 : 전북대학교 컴퓨터과학과 교수
논문접수 : 1998년 12월 8일, 심사완료 : 1999년 4월 13일

영상이 사실적(Photo-realistic)이기 때문에 컴퓨터 그래픽스 커뮤니티에서 많은 주목을 받고 있다. 즉, 광선 추적법은 단일화된 알고리즘으로 음영, 반사, 투명감, 그림자 등의 광학적인 효과를 모두 표현할 수 있어 생성된 영상이 아주 사실적이며, 폴리곤 메쉬, 음함수 등의 여러 가지 모델링 기법도 쉽게 적용할 수 있다[9]. 그러나, 광선 추적법은 최소한 화면의 해상도에 대응하는 광선을 추적하여야 하기 때문에 계산량이 많아 단순한 영상을 생성하는데도 많은 시간이 걸린다. 이러한 문제점 때문에 가상 현실감, 실시간 에니메이션 등의 고속성을 필요로 하는 분야에서는 광선 추적법을 사용하기 어렵다[7]. 광선추적법의 이러한 문제점을 해결하기 위하여 병렬 컴퓨터, 또는 알고리즘에 의한 고속화 기법이 제안되었는데 대표적인 것은 다음과 같다.

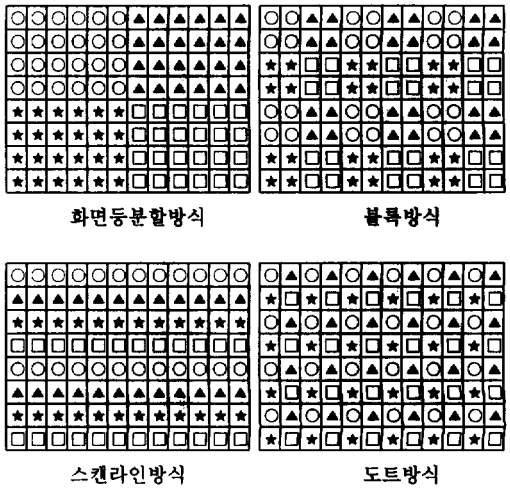
① 화면(image space)분할방식을 이용한 멀티프로세서 처리[3]
② 물체공간(object space)분할방식을 이용한 멀티프로세서 처리[1]
③ 옥트리(octree)를 이용한 고속물체탐색[2]
④ 경계공간(boundary space)을 이용한 고속물체탐색[8]

이러한 방식 중 본 논문에서는 화면분할방식을 이용한 멀티프로세서 처리에 대하여 논한다. 물체공간분할방식을 이용한 멀티프로세서 처리에서는 물체공간(object space)을 분할하여 분할된 부분공간(subspace)을 각 요소 프로세서에 분배함으로 각 요소 프로세서는 일부의 모델링 데이터만을 갖고 광선 추적처리를 할 수 있다. 이러한 이유로 물체공간분할방식을 이용한 프로세서 처리에서는 메모리 공간을 효율적으로 이용할 수 있는 장점은 있으나, 요소 프로세서에 걸리는 부하를 미리 예측할 수 없어 부하 분산이 어려운 단점이 있다. 이것에 비하여 화면분할방식을 이용한 멀티프로세서 처리에서는 동적 및 정적 부하분산기법을 이용하여 각 요소 프로세서의 부하가 균등하게 되도록 부하분산을 할 수 있다[3]. 본 논문에서는 광선 추적법의 처리에 병렬 컴퓨터를 효과적으로 사용할 수 있도록 화면분할방식 기반의 한 부하분산법을 제안하고 이것을 MIMD(Multiple Instruction stream Multiple Data stream) 방식의 富士通 AP-1000 멀티프로세서 위에서 평가, 고찰하였다. 본 논문의 구성은 다음과 같다. 우선 2장에서 광선 추적법의 부하 분산에 대하여 조사하고, 다음 3장에서 광선 추적법의 부하분산에 이용할 수 있는 여러 가지 멀티프로세서의 부하 스케줄링 방식에 대하여 고찰한다. 이어서 4장에서는 3장의 스케줄링 방식을 이용하여 AP-1000 멀티프로세서 위에서 여러 가지 부하분산방식에 관한 실험 및 평가를 한다. 그리고 5장에서는 4장의 실험을 바탕으로 하여 광선추적법을 위한 한 분하분산법을 제안하여 평가, 고찰하고 마지막으로 6장에서 결론을 내린다.

2. 광선 추적법의 부하분산법

화면분할방식의 부하분산법은 크게 정적 부하 분산법과 동적 부하분산법 두 가지로 나눌 수 있다. 정적 부하 분산법은 처리에 앞서 부하를 관리하는 호스트 컴퓨터(host computer)가 각 요소 프로세서(PE ; Processor Element)가 담당할 화면 영역을 정하여 한꺼번에 각 요소 프로세서에 할당하면 요소 프로세서가 주어진 영역을 처리하고 종료하는 방식이다. 동적 부하분산법은 호스트 컴퓨터가 우선 각 요소 프로세서가 처리할 기본적인 영역을 할당하고 그 다음의 영역을 계산이 끝난 요소 프로세서에게 순차적으로 할당하는 방식이다.

정적 부하 분산법은 다음과 같은 종류가 있는데 이것을 (그림 1)에 표시한다[10].



(그림 1) 정적 부하 분산법(요소프로세서 4개)

① 화면등분할방식

화면을 요소 프로세서 개수에 맞추어 같은 크기로 분할하여 요소 프로세서에 할당한다. 요소 프로세서의 개수를 P, 픽셀수를 $N_x \times N_y$ 라고 한다면 각 요소 프로세서가 처리하는 픽셀수는

$$\frac{N_x \times N_y}{P}$$

가 된다.

② 블록방식

화면을 몇 개, 또는 수십개 픽셀 단위의 같은 크기의 사각형 영역(block)으로 분할하여 요소 프로세서에 할당한다. 부하를 균등 분배하기 위해서는 사각형 영역의 개수를 요소 프로세서의 개수보다 많게 할 필요가 있다. 요소 프로세서는 주어진 사각형 영역의 광선처리를 한다.

③ 스캔라인방식

화면을 주사선(스캔라인) 단위로 분할하여 각 라인을 요소 프로세서에 할당한다. 라인의 수가 요소 프로세서 개수 보다 많을 때는 라인을 몇 개씩 건너뛰어 요소 프로세서에 할당함으로서 부하가 한 요소 프로세서에 집중하는 것을 방지한다.

④ 도트(dot)방식

픽셀 단위의 부하 분산법으로 픽셀을 몇 개씩 건너뛰어 요소 프로세서에 담당시켜 부하가 한 요소 프로세서에 집중하는 것을 방지할 수 있으나 오버헤드가 큰 것이 단점이다.

일반적으로 화면등분할방식, 블록방식, 스캔라인방식은 물체공간에 프리미티브(primitive)가 균일하게 분포되어 있을 경우에는 각 요소 프로세서의 부하가 균등하게 분산하나 한곳에 집중되어 있을 경우에는 부하의 불균형을 초래한다. 이에 비해 도트방식은 물체 공간 내에 프리미티브가 집중되어 있어도 요소 프로세서의 부하가 균형을 유지하나, 그래픽스 고유의 성질인 집중도(coherence)를 이용할 수 없어 처리 시간이 증가하고 앤티에일리어싱(anti-aliasing) 처리가 어려워지는 단점이 있다.

광선 추적법은 광학 효과를 3차원 공간에서 시뮬레이트하기 때문에, 하나의 화면을 렌더링 하는데 필요한 계산량은 가시면의 반사, 투과, 굴절등의 속성에 의존하여 같은 물체공간이라도 시점(view point)에 따라 계산량이 다르다. 따라서 정적으로 요소 프로세서의 부하가 균등하도록 부하를 분산시키는 것은 어렵다. 동적 부하분산은 호스트 프로세서가 처리가 끝난 요소 프로세서에 대해서 순차적으로 화면 영역을 할당하는 방법으로 요소 프로세서의 부하 처리 상태를 관찰하면서 부하를 할당하기 때문에 부하를 균등하게 분산하는 것이 비교적 쉽다. 그러나, 처리 중에 요소프로세서와 호스트 컴퓨터간에 통신이 계속하여 필요하기 때문에 통신 오버헤드(overhead)가 증가한다. 동적 부하분산법은 정적 부하 분산법과 동일하게 동적 블록방식, 동적 스캔라인방식, 동적 도트방식의 3 가지가 있으나 동적 도트방식은 통신 오버헤드가 너무 커서 그다지 사용되지 않는다.

3. 멀티프로세서의 부하 스케쥴링

3.1 순환문의 스케쥴링

광선 추적법에서 화면의 픽셀에 대응하는 각각의 광선 처리는 서로 의존 관계(dependency)가 없고, 처리해야할 광선의 수가 화면의 해상도(resolution) 이상으로 많기 때문에 광선을 처리 단위로 하는 MIMD 형 멀티프로세서 병렬처리에 적당하다. 광선 추적법은 순환문으로 기술할 수가 있는데 병렬 처리의 효과를 높이기 위해서는 병렬성을 최대한도록 이끌어 낼 수 있도록 이것을 분할하여 각 요소 프로세서에 할당하여야 한다. 광선 추적법의 주함수(main function)가 되는 순환문은 각 이터레이션(iteration) 간에 의존 관계가 없어 비교적 스케쥴링이 용이하다. 광선 추적법의 주함수는 다음과 같이 화면의 모든 픽셀에 대해서 같은 처리를 반복하는 순환문이다.

```
for ( 화면의 모든 픽셀에 대해서 ) {
        광선과 물체와의 교차 계산
        if ( 교점이 존재하면 ) {
                교점의 휘도를 계산한다.
        }
        else {
                배경색을 계산한다.
        }
}
```

이 순환문은 픽셀단위로 병렬성이 존재하기 때문에 위에서 설명한 광선 추적법의 부하 분산법과 순화문의 스케쥴링방식을 함께 이용하여 요소 프로세서간에 부하가 균형이 잡히도록 부하 분산을 시킬 수 있다.

순환문의 스케쥴링방식은 다음과 같이 3종류 4 방식으로 분류된다.

1) 정적 청크방식(Static Chunking)

요소 프로세서에 부하를 할당하는 단위를 청크라 하는데 요소 프로세서의 개수를 P, 이터레이션의 회수를 N 이라고 하면 각 요소프로세서에 할당되는 이터레이션을 단위로 하는 청크크기는 다음과 같다.

$$\frac{N}{P}$$

이 방식은 각 프로세서가 할당받은 부하를 처리하는 편차가 큰 것이 특징이다.

2) 고정길이 청크방식(Fixed-Size Chuncking)

청크수를 요소 프로세서의 개수 보다 많게 하고 청크크기를 임의 값으로 고정하여 각 요소 프로세서에 할당한다. 최적인 청크크기를 결정하기가 어려운 것이 단점이다.

3) 가변길이 청크방식

각 요소 프로세서에 할당하는 청크크기를 다르게하여 요소 프로세서의 부하가 균등하게 할당되도록 하는 방식으로 다음의 2 방식이 있다.

① GSS(Guided Self-Scheduling)

문헌[6]에서 제안하고 있는 동적 스케쥴링의 한 방식으로 통신 시간 때문에 모든 프로세서가 동시에 시작할 수 없는 점을 고려하여 먼저 처리를 시작하는 요소 프로세서에게 부하를 나중에 처리를 시작하는 요소 보다 많이 할당하는 방식이다. 이 방식에서는 쉬고 있는 프로세서(idle processor)에게 할당할 청크크기를 다음과 같이 계산한다.

$$R_0 = N, \quad R_{i+1} = R_i - G_i$$
$$G_i = \lceil \frac{R_i}{P} \rceil = \lceil (1 - \frac{1}{p})^i \frac{N}{P} \rceil$$

여기서 각각의 기호는

P : 요소 프로세서의 개수

N : 순환문의 전체 이터레이션 수

G_i : 다음의 쉬고 있는 프로세서에 할당되는 이터레이션 수

R_i : 나머지 이터레이션 수

를 의미한다.

② 팩토링(Factoring)

정적 스케쥴링의 한 종류로 실행시간이 다른 이터레이션 간의 차이를 줄이기 위해서 문헌[5]에서 제안한 방식이다. 요소 프로세서 개수와 같은 수의 청크를 배치(batch)로 구성하여 각 요소 프로세서에 할당하면 요소 프로세서가 배치내의 청크를 처리한다. 팩토링에서 변수는 배치의 이터레이션 수로 각 요소 프로세서는 모두 같은 수의 이터레이션 $\frac{N}{P}$를 처리한다. 각 요소 프로세서에 할당되는 청크크기는 다음의 식에서 구한다.

$$R_0 = N, R_{j+1} = R_j - PF_j$$
$$F_j = \lceil \frac{R_j}{xP} \rceil$$
$$= \lceil (\frac{1}{2})^{j+1} \frac{N}{P} \rceil$$

여기서 각각의 기호는

P : 요소 프로세서의 개수

N : 모든 이터레이션 수

R_j : 나머지의 이터레이션 수

F_j : 다음의 쉬고 있는 프로세서에게 할당할 이터레이션 수

x : 부하의 특성과 관련된 지수로 본 실험에서는 각 이터레이션의 처리 시간이 같다고 보아 2로 계산하였음.

을 의미한다.

<표 1>에 요소 프로세서의 개수를 8, 순환문의 이터레이션 수를 1000으로 하였을 때의 각 방식의 청크크기를 보인다.

〈표 1〉 청크크기의 예

방 식	N = 1000, P = 8
정적 청크방식	125 125 125 125 125 125 125 125
고정길이 청크방식	K K K K K K K K …
GSS	125 110 96 84 74 55 48 42 …
Factoring	125 125 125 125 62 62 62 62 …

3.2 화면 분할방식의 요소 프로세서 부하 스케쥴링

광선 추적법에 있어서 각 이터레이션을 한 개의 픽셀에 대응하는 광선 처리로 볼 수 있기 때문에 순환문의 스케쥴링방식을 화면 분할방식에 그대로 적용할 수 있다. 다음은 순환문의 스케쥴링방식을 이용하여 요소 프로세서에 화면 영역을 할당하는 방법에 대하여 고찰한다.

① 정적청크방식

이방식은 이터레이션 수를 여러 요소 프로세서에게 모두 같게 정적으로 할당하기 때문에 정적부하분산의 모든 방식(화면등분할방식, 블록방식, 스캔라인방식, 도트방식)에 적용할 수 있다. 즉, 분할된 화면, 스캔라인 등에 청크를 대응시키면 된다. 단, 화면등분할방식에 적용할 경우에는 부하가 어떤 요소 프로세서에 집중할 수 있어 요소 프로세서 간의 부하 불균형 문제를 생각하여야 한다.

② 고정길이청크방식

고정길이청크방식은 픽셀을 동적으로 분할할 수 있는 모든 방식(블록방식, 스캔라인방식, 도트방식)에 적용할 수 있다. 단, 도트방식은 통신 오버헤드가 너무 커질 우려가 있기 때문에 피하고 블록방식, 스캔라인방식을 선택하는 것이 바람직하다.

③ GSS

가변 길이의 동적할당이기 때문에 화면분할방식이나 블록방식에는 적용하기는 어렵다. 그러나 스캔라인 방향으로 가변 길이의 청크를 생성할 수 있기 때문에 스캔라인 단위로 할당할 수 있다. 가변 도트방식에서는 처리가 끝나감에 따라 한 개의 픽셀만의 처리가 많아지기 때문에 통신 오버헤드가 증가할 수 있다.

④ 팩토링

GSS와 같은 이유에서 스캔라인방식에 적용할 수 있다. 단, 청크의 할당은 정적 할당이다.

4. 평가 및 고찰

4.1 평가 방법

위에서 설명한 여러 가지 화면 분할방식의 요소 프로세서 부하 스케쥴링에 대한 성능을 비교하기 위해서 AP-1000 멀티프로세서를 이용하여 평가 영상의 광선 추적법 처리 시간을 측정하였다. 스크린의 해상도는 512×512로 하였고, 평가 영상은 Haines가 제안한 SPD 모델중[4] balls와 mount를 사용하였는데 이것을 (그림 4.1), (그림 4.2)에 보인다. 측정값은 정적부하분산법에서는 광선추적처리 시간만을 대상으로 하였고, 동적부하분산법에서는 광선 추적처리 시간에 통신 시간을 합한 것으로 하였다.

부하 분산 방식은 다음의 각 방식을 사용하였다. 각 방식에서 청크크기(청크가 갖는 픽셀수)는 다음과 같이 정하였다.

정적부하분산

① 화면 등분할방식 : 요소 프로세서가 담당하는 픽셀수는 65,536(PE 4개), 16,384(PE 16개), 4,096(PE 64개)

② 정적블록방식 : 64 픽셀/청크

③ 정적스캔라인방식 : 512 픽셀/청크

④ 도트방식 : 픽셀/청크

⑤ 팩토링방식 : 청크크기는 가변이고, 청크수는 68(PE 4개), 240(PE 16개), 832 (PE 64개)

동적부하분산

① 동적블록방식 : 4,096 청크×(64 픽셀/청크)

② 동적스캔라인방식 : 512 청크×(512 픽셀/청크)

③ GSS방식 : 청크크기는 가변이고, 청크수를 각각 160 (PE 16개), 565(PE 64개)

평가 항목은 다음과 같다.

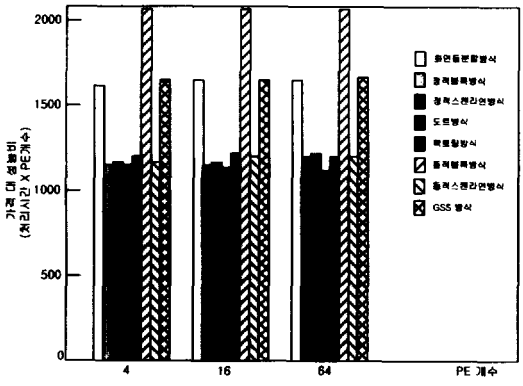
- 가격대 성능비의 비교 : 요소 프로세서의 개수를 4, 16, 64개로 변화시켰을 때 각 부하분산방식의 가격대 성능비를 비교한다.
- 요소 프로세서간의 부하 편차 : 각 부하분산방식을 이용하여 광선 추적법을 실행시켰을 때 요소 프로세서 종료 시간의 편차를 측정한다.
- 통신 오버헤드의 영향 : 동적부하분산으로 광선 추적법을 처리하였을 때의 호스트 컴퓨터와 요소 프로세서간의 통신 시간을 측정하여 통신 오버헤드의 영향을 조사한다.

4.2 평가 결과

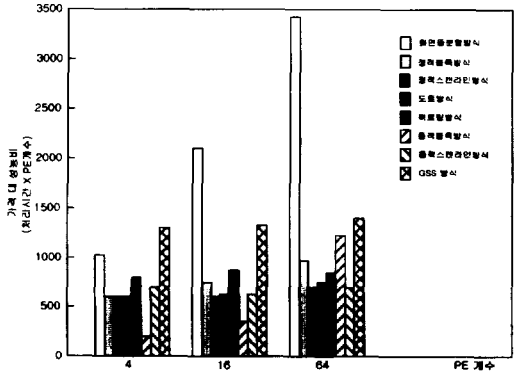
4.2.1 가격 대 성능 비의 비교

각 평가 영상의 가격 대 성능 비의 결과를 (그림 2.1)(balls), (그림 2.2)(mount)에 보인다. 세로축은 가격

대 성능 비를 표시하는데 이것은 식 : (최대 처리시간) ×(요소프로세서 개수)에서 구하였다. 가로축은 요소 프로세서의 개수를 표시한다.



(그림 2.1) 가격 대 성능비(balls)



(그림 2.2) 가격 대 성능비(mount)

- balls

(그림 2.1)에 평가 영상 balls의 영상 생성 시간을 보인다. 각 방식이 모두 요소 프로세서의 개수가 늘어남에 따라 대체로 성능도 향상된다.

- mount

(그림 2.2)에 평가 영상 mount의 영상 생성 시간을 보인다. 화면등분할방식, 정적블록방식, 동적블록방식의 성능이 좋지 않다. 특히 화면등분할방식은 요소 프로세서의 개수를 4배 증가시켜도 성능은 2배 정도만 증가하였다. 그리고 동적 블록방식은 요소 프로세서의 개수가 64개가되자 성능이 급격히 떨어졌다.

- 정적부하분산

정적 스캔라인방식, 도트방식, 팩토링방식에서 좋은 결과를 얻었다. 이 세 방식은 요소 프로세서 개수가 증가함에 따라 성능도 향상되었다. 그러나 화면등분할방식 및 정적블록방식은 요소 프로세서의 개수가 증가해도 성능이 향상되지 않았는데 이것은 요소 프로세서간의 부하 편차 때문이라 생각된다. 즉, 전체 처리 시간은 항상 가장 늦게 끝나는 요소 프로세서의 처리 시간으로 되기 때문에 어떤 한 프로세서의 처리가 늦으면 아무리 다른 프로세서의 처리가 빨라도 의미가 없다. 따라서 정적부하분산에서는 요소 프로세서의 수가 증가할수록 부하를 균등하게 분산시키는 것이 중요하다.

- 동적부하분산

동적스캔라인방식, GSS방식은 요소 프로세서의 개수가 증가함에 따라 성능이 향상한다. 그러나 동적 블록방식은 요소 프로세서의 개수가 64개일 경우에 성능이 급격히 저하한다. 이 원인은 블록방식이 다

〈표 2.1〉 요소 프로세서간의 부하편차(balls)

단위 : 초

	4			16			64		
부하분산방식	최대치	최소치	편차	최대치	최소치	편차	최대치	최소치	편차
화면등분할	399.64	183.29	54.1 %	104.54	25.40	75.7 %	26.29	3.73	85.8 %
정적블록	285.49	285.07	0.1 %	71.80	70.87	1.3 %	19.01	16.82	11.5 %
정적스캔라인	286.19	284.42	0.6 %	72.61	70.42	3.0 %	19.08	16.83	11.8 %
도트	285.36	285.33	0.01 %	71.41	71.36	0.07 %	17.90	17.85	0.3 %
팩토링	303.8	238.9	21.4 %	76.3	52.4	31.4 %	19.3	12.9	33.0 %
동적블록	512.47	511.99	0.09 %	128.30	127.82	0.29 %	32.16	31.80	1.12 %
동적스캔라인	295.57	295.50	0.02 %	74.42	74.27	0.2 %	19.11	18.75	1.9 %
GSS	417.69	229.95	44.9 %	104.69	63.43	39.4 %	26.21	17.31	33.9 %

〈표 2.2〉 요소 프로세서간의 부하편차(mount)

단위 : 초

	4			16			64		
부하분산방식	최대치	최소치	편차	최대치	최소치	편차	최대치	최소치	편차
화면등분할	264.12	35.93	86.4 %	132.26	7.51	94.3 %	53.70	1.74	96.8%
정적블록	153.48	150.70	1.8 %	43.79	28.80	34.2 %	15.22	2.76	81.9%
정적스캔라인	152.11	151.58	0.4 %	38.18	37.53	1.7 %	10.03	9.06	9.7%
도트	152.07	151.84	0.1 %	38.47	37.69	2.0 %	10.40	8.65	16.7%
팩토링	204.2	98.9	51.6 %	53.2	19.5	63.4 %	13.9	4.8	65.3%
동적블록	44.72	44.65	0.2 %	20.77	20.64	0.6 %	19.51	19.16	1.8%
동적스캔라인	156.89	156.61	0.2 %	39.28	39.01	0.7 %	10.17	9.77	4.0%
GSS	327.28	62.09	81.0 %	84.28	17.63	79.0 %	22.64	6.01	73.4%

른 방식과 비교하여 호스트 컴퓨터와 요소 프로세서 간에 통신 회수가 증가하였기 때문이라 생각된다. 동적블록방식을 이용한 광선추적처리시간은 <표 3.1>, <표 3.2>에서 알 수 있듯이 통신시간이 차지하는 비율이 상대적으로 높다. 즉, 성능이 떨어진 원인이 전체처리 시간에 비하여 통신 시간의 비율이 높아졌기 때문이다. 동적부하분산에서는 성능 향상을 위하여 통신의 오버헤드를 삭감하는 것이 무엇보다 중요하다.

4.2.2 요소 프로세서 간의 부하 편차

요소 프로세서간의 부하 편차를 다음과 같이 정의하였다.

$$\text{부하편차} = \frac{(\text{최대처리시간}) - (\text{최소처리시간})}{(\text{최대처리시간})} \times 100$$

위의 정의에 따른 부하 편차를 <표 2.1>, <표 2.2>에 표시하고 각 부하분산법에 관하여 부하 편차에 관하여 고찰한다.

- balls

  평가 영상 balls의 부하편차를 <표 2.1>에 보인다. balls 영상의 물체(object) 구성은 복잡하고 프리미티브의 수도 비교적 많으나, 화면 중앙을 중심으로 공간 내에 물체가 대칭으로 존재하기 때문에 각 방식 모두 부하 편차가 작다. 특히 정적 도트방식에서 부하 편차가 작고 비교적 부하가 균등하게 분산되었음을 알 수 있다. 그러나 화면등분할방식 및 GSS 방식은 비교적 편차가 크다.

- mount

  평가영상 mount의 부하편차를 <표 2.2>에 보인다. mount 영상은 ball 영상보다 부하 편차가 심하다.

〈표 3.1〉 통신 오버헤드(balls)

	4		16		64	
부하분산방식	최대치	최소치	최대치	최소치	최대치	최소치
동적블록	1.14 %	1.04 %	1.39 %	1.10 %	2.47 %	0.99 %
동적스캔라인	0.37 %	0.32 %	1.65 %	0.63 %	5.91 %	0.93 %
GSS	0.08 %	0.002 %	0.53 %	0.005%	8.00 %	0.02 %

〈표 3.2〉 통신 오버헤드(mount)

	4		16		64	
부하분산방식	최대치	최소치	최대치	최소치	최대치	최소치
동적블록	11.74 %	11.60 %	53.66 %	50.21 %	85.71 %	85.40 %
동적스캔라인	0.76 %	0.53 %	0.93 %	0.41 %	3.10 %	1.08 %
GSS	0.40 %	0.22 %	2.70 %	0.01 %	29.05 %	0.02 %

이것은 mount 영상이 많은 처리 시간이 필요한 투명물체 부하를 가지고 있는데 이것이 불균등하게 분산되었기 때문이라 생각된다. 그리고 팩토링 및 GSS방식도 비교적 부하 편차가 크다.

o 정적부하분산

정적부하 분산에서는 정적스캔라인방식 및 도트방식이 편차가 작다. 정적스캔라인방식 및 정적도트방식의 경우 인접하는 픽셀의 부하가 크게 차이가 나지 않는 그래픽스의 고유한 성질인 응집성(coherency) 때문이라 생각된다. 정적블록방식 및 팩토링 방식에서는 요소 프로세서의 개수가 증가할수록 편차도 커진다. 이것은 양방식 모두 각 청크의 부하에 편차가 생기기 때문이라 생각된다. 특히 팩토링방식의 경우 초기 단계에 생성되는 배치의 청크에 부하 편차가 많이 생기기 때문이라 생각된다. 단 정적 블록방식은 화면등분할방식과 도트방식의 중간에 위치하는 것으로 볼 수 있기 때문에 블록내의 픽셀수가 적으면 부하 편차도 작아지리라 예상된다.

o 동적부하분산

동적부하분산에서는 처리시간에 요소 프로세서의 부하 처리 상황을 검출하여 부하를 분배하기 때문에 부하 편차가 작게 나타났다. 그러나, 평가영상 mount에서는 GSS방식의 편차가 크게 나타났는데 이것은 할당되는 청크의 크기와 평가영상의 복잡도와 관련이 있기 때문이라 생각된다. GSS방식은 할당되는 청크내의 픽셀수가 변하기 때문에 초기 단계의 청크내 픽셀수는 종료 시간의 청크수와 비교해서 상당히 많다. 그리고 많은 처리 시간이 필요한 투명체가 장면의 한곳에 모아져있다. 이러한 이유로 투명체의 처리가 일부의 요소 프로세서에 집중되어 부하 편차가 크게 되었다고 생각된다.

4.2.3 통신 오버헤드의 영향

동적부하분산법은 호스트 컴퓨터와 요소 프로세서가 메시지를 교환하면서 광선처리를 수행하기 때문에 전체 처리 시간에 대한 통신 시간의 비율이 크다. 이 비율을 측정하기 위하여 호스트 컴퓨터와 요소 프로세서 간의 통신 시간을 측정하였다. 측정값은 호스트 컴퓨터와 요소 프로세서의 전체 처리 시간에 대한 통신 시간의 비율 중 최대값과 최소값을 표시하였다.

o balls :

모든 방식에서 통신 오버헤드가 작다.

o mount :

동적스캔라인방식은 비교적 통신시간이 차지하는 비율이 작으나 동적블록방식 및 요소 프로세서의 개수가 64개 일 때의 GSS방식이 통신 시간이 차지하는 비율이 높다. 요소 프로세서의 개수가 적을 때는 각 요소 프로세서의 처리시간이 비교적 크고 통신 시간이 작기 때문에 통신 오버헤드가 성능에 미치는 영향은 작다. 그러나 요소 프로세서가 증가하면 광선추적 처리에 필요한 시간은 짧게되어 상대적으로 통신시간의 비율이 높아 성능은 향상되지 않는다. 이러한 대표적인 예가 동적블록방식으로 mount 모델을 처리한 경우이다. <표 2.2> 및 <표 3.2>를 살펴보면 요소 프로세서가 64개인 경우의 광선처리에 필요한 시간은 16개 경우의 $\frac{1}{4}$ 정도이나 통신 시간이 광선처리시간 이상으로 증가하였기 때문에 성능이 향상되지 못하였다.

블록방식과는 대조적으로 스캔라인 방식에서는 통신시간이 차지하는 비율이 작기 때문에 모든 처리에 필요한 시간이 요소 프로세서 수가 증가함에 따라 짧아지고 있다.

따라서 동적 부하분산을 이용하기 위해서는 요소 프로세서에 할당하는 픽셀수와 통신 회수를 고려하여야 한다. 단 통신 오버헤드를 작게하기 위해서는 요소 프로세서에 할당하는 픽셀수를 많게 하여야 하나 너무 많게 하면 각 요소 프로세서의 처리 시간 편차가 커지기 때문에 통신 시간과 광선 추적 시간을 고려하여 최적한 픽셀수를 계산하여야 한다.

5. 제안된 방식의 제안 및 평가

5.1 제안된 방식의 제안

우리는 위의 실험 결과를 분석하여 다음과 같은 결과를 얻었다.

o 정적부하분산에서는 요소 프로세서 개수가 많아지면 부하 편차가 크게된다.

o 동적부하분산은 처리중에 부하가 처리되는 상태를 검출하여 부하 편차를 작게 할 수 있다.

현재 컴퓨터 하드웨어의 가격은 하락하고 있고, 장

래에는 이러한 경향이 더욱 가속되리라 생각되어 많은 요소 프로세서에 효과적인 부하분산법의 개발이 중요하다. 정적부하분산은 일반적으로 부하예측이 어려워 요소 프로세서 개수가 많아지면 부하를 균등하게 분산시키기가 어렵다. 그러나, 동적부하분산은 부하의 처리 상황에 맞추어 요소 프로세서에 부하를 할당하기 때문에 균등하게 부하분산을 할 수 있다. 그러나 동적부하분산은 통신 오버헤드의 문제가 있다. 통신 오버헤드를 감소시키기 위해서는 요소 프로세서에서의 처리 상황 및 부하의 특성을 고려하여 각 프로세서에 부하를 할당하여야 한다. 여기에서는 광선추적법의 성질을 고려하여 다음과 같은 부하 분산 방침을 정하였다.

① 부하를 고려하여 요소 프로세서가 처리할 청크크기(필셀 수)를 결정한다. 부하를 픽셀수를 적게한 전처리에 의해 측정하여 처리시간이 긴 투명물체에 해당하는 부하는 픽셀수를 적게 하여 요소 프로세서에 할당하고 반대로 배경에 해당하는 부하는 픽셀수를 많게 하여 요소 프로세서에 할당한다. 즉, 청크크기를 다르게 하여 요소 프로세서가 호스트 컴퓨터에 대하여 경합하는 것을 완화시킨다.

② 초기단계에서 너무 부하가 불균등하게 할당되어 편차가 크게 나는 것을 방지하기 위해서 초기 단계 청크크기를 GSS방식 혹은 팩토링방식보다도 작게 한다.

③ 청크크기가 일정 값 이하가 되면 1 주사선에 포함되어 있는 픽셀수를 청크크기로 한다. 이것이 전체의 청크수를 적게하고 호스트와의 통신회수를 줄이는 역할을 한다.

위의 방침에 따라 제안된 방식에서는 GSS방식과 팩토링방식을 절충하여 요소 프로세서에 할당하는 픽셀수를 다음과 같이 정하였다.

① $M_k > N_x$이 될 때 다음 식에 의해 요소 프로세서에 할당하는 픽셀수를 계산한다.

$$R_0 = N, R_{k+1} = R_k - P \times M_k$$
$$M_k = \lceil \frac{1}{4} \frac{R_k}{P} \rceil$$

② $R_k < N_x$이 될 때까지 다음 식을 반복하여 사용한다.

$$R_{k+1} = R_k - N_x$$
$$M_k = N_x$$

③ $R_k = N_x$일 때

$$M_k = R_k$$

여기에서

$N = N_x \times N_y$: 화면내의 전체 픽셀수

N_x : 화면내의 가로 방향의 픽셀수

N_y : 화면내의 세로 방향의 픽셀수

P : 요소 프로세서의 개수

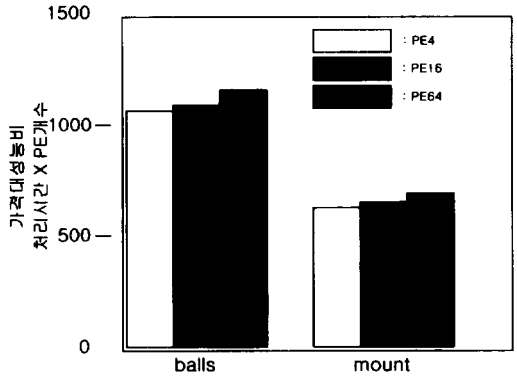
M_k : 쉬고 있는 프로세서에 할당하는 픽셀수

R_k : 나머지 픽셀수

를 의미한다.

5.2 제안된 방식의 평가 결과

위에서 제안된 방식을 사용하여 광선추적처리를 하였을 때의 가격 대 성능비, 부하 편차, 통신 오버헤드를 측정하여 이것을 (그림 3) 및 <표 4>에 표시하였다. 4장의 여러 방식 보다 다음과 같은 면에서 개선되었다.



(그림 3) 제안한 방식의 가격 대 성능비

- 가격 대 성능비

(그림 3)에 가격 대 성능비를 보였다. (그림 3)은 기존의 부하 분산 방법과 비교하여 평가 영상 2개가 모두 요소 프로세서의 개수에 비례하여 처리 시간이 감소한 것을 알 수 있다. 요소 프로세서의 부하 편차와 통신오버헤드가 작아졌기 때문이라 생각된다.

- 요소 프로세서의 부하 편차

<표 4.1>에 제안된 방식의 각 평가 영상에 대한 요소 프로세서의 부하 편차를 표시하였다. 평가 영상

〈표 4.1〉 제안된 방식의 부하편차

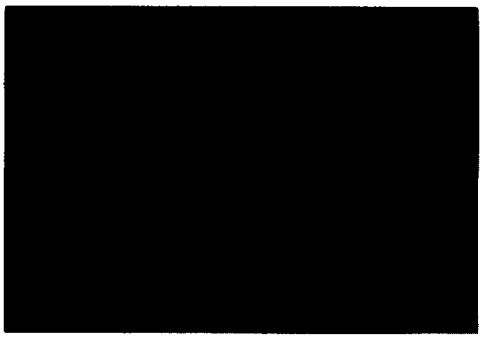
단위 : 초

	4			16			64		
부하부산방식	최대치	최소치	편차	최대치	최소치	편차	최대치	최소치	편차
balls	285.51	285.01	0.17 %	71.92	71.09	1.15 %	18.87	17.91	5.1 %
mount	154.29	154.11	0.11 %	38.65	38.41	0.6 %	9.93	9.57	3.6 %

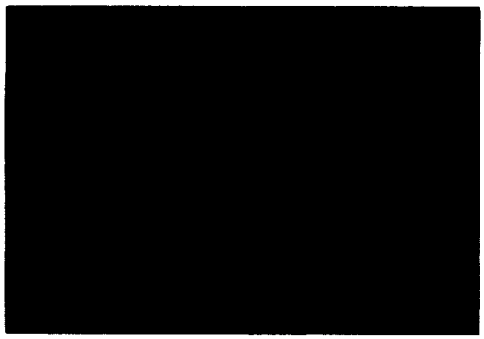
mount에 대하여 GSS방식, 팩토링방식, 제안된 방식을 비교하여 보면 제안된 방식이 부하 편차가 작은데 이것은 제안된 방식에서 제안한 초기 단계 청크 크기를 작게하여 부하를 균등히 하는 효과가 입증된 것이라고 생각된다.

o 통신 오버헤드

호스트 컴퓨터와 요소 프로세서간의 통신시간이 전체 처리에 차지하는 비율을 (그림 4.2)에 표시하였다. 제안된 방식에서는 처리시간에 비하여 통신시간이 차지하는 비율이 작아 통신 오버헤드를 줄일 수 있는 것을 알았다.



(그림 4.1) 평가 영상 balls



(그림 4.2) 평가 영상 mount

〈표 4.2〉 제안된 방식의 통신 오버헤드

	4		16		64	
부하분산방식	최대치	최소치	최대치	최소치	최대치	최소치
balls	0.02%	0.01%	0.09%	0.05%	3.44%	1.11%
mount	0.07%	0.03%	0.11%	0.09%	1.89%	0.40%

6. 결 론

본 논문에서는 멀티프로세서를 이용하여 광선추적 처리를 고속화하는데 효과적인 부하분산방식을 제안하고 이것을 멀티프로세서 AP-1000 위에서 평가하였다. 제안된 방식을 제안하기에 앞서서 기존의 여러 부하분산 방식을 평가하였는데, 이 과정에서 다음과 같은 것을 알았다.

정적부하분산에서는 정적 스캔라인방식과 도트방식이 성능이 상당히 좋으나 화면등분할방식 및 정적블록방식은 요소 프로세서간의 부하편차가 크고 가격 대 성능비도 좋지 않았다. 따라서, 정적부하분산에서는 인접하는 청크 부하의 차가 없도록 화면 분할을 하는 것이 중요하다는 것을 알았다. 동적부하분산에서는 무엇보다도 통신의 오버헤드가 성능에 미치는 영향이 큰 것을 확인하였다. 통신 오버헤드를 줄이는 가장 단순한 방법은 호스트 컴퓨터와 요소 프로세서간에 통신회수를 줄이는 것이나 이것은 부하 편차를 초래하여 처리시간을 증가시킬 수 있다. 따라서 통신회수와 처리시간을 함께 고려하여 청크크기를 결정하여야 한다.

멀티프로세서 AP-1000에서 최적의 동적부하분산방식을 결정하기 위해 광선 추적법을 실행하면서 부하편차 및 통신시간을 측정하였다. 결과를 분석하여 다음 방침을 정하였다.

o 초기단계에서 생성되는 청크크기를 GSS 및 팩토링 방식보다 작게하여 픽셀수가 많은 부하를 분산하여 부하가 균등하게 할당되도록 한다.

o 청크내의 픽셀수가 어느 정해진 값 이하로 되면 1 주사선에 포함되어 있는 픽셀수를 청크의 픽셀수로 한다. 이것에 의해 전체의 청크수를 작게하고 호스트와의 통신회수를 줄인다.

위의 방침에 의해서 고안된 동적부하분방식의 부하 편차와 통신 시간을 측정하여 본 결과, 이 방식이 물체가 한곳에 집중해있는 평가 영상에서도 요소 프로세서간의 부하를 균등하게 하고 통신 오버헤드를 줄일 수 있는 것을 알았다.

향후 연구는 본 논문에서 제안한 부하분산방식이 멀티프로세서 AP-1000 이외의 다른 종류의 멀티프로세서에서도 적용가능한지를 검증하고, 이 알고리즘을 개선하여 다양한 멀티프로세서에서 사용할 수 있는 일반적인 부하분산 알고리즘을 제안할 예정이다.

참 고 문 헌

[1] Dippe, M., and Swensen, J. : An Adaptive Subdivision Algorithm and Parallel Architecture for Realistic Image Synthesis, Proc. SIGGRAPH'84, pp.149-158, 1984.

[2] Sung, K. : A DDA Octree Traversal Algorithm for Ray Tracing, EUROGRAPHICS '91, pp.73-85 Elsevier Science Publishers B.V.(North-Holland), 1991.

[3] Green, S, A., and Paddon, D, J. : Exploiting Coherence for Ray Tracing, IEEE Computer Graphics & Applications, Vol.9, No.6, pp.12-26, 1989.

[4] Haines, E. : A Proposal for Standard Graphics Environments, IEEE Computer Graphics & Applications, Vol.7, No.11, pp.3-5, 1987.

[5] Hummel, S., F. Schonberg, E., and Flynn, L. E. : FACTORING : A Method for Scheduling Parallel Loops, Commun. ACM, Vol.35, No.8, pp.90-101, 1992.

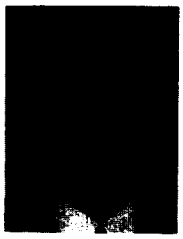
[6] Polychronopoulos, C., and Kuck, D. : Guided self-scheduling : A Practical Scheduling Scheme for Parallel Supercomputers, IEEE Trans. Comput. C-36, pp.1425-1439, 1987.

[7] Robert, A. Cross. : Interactive Realism for Visualization Using Ray Tracing, IEEE Proceedings '95 visualization. pp.19-25, 1995.

[8] Weghorst, H., Hooper, G., and Greenberg, D. P. : Improved Computational Methods for Ray Tracing, ACM Trans. Computer Graphics, Vol.3, No.1, pp.52-69, 1984.

[9] Whitted, T. : An Improved Illumination Model for Shaded Display, Comm. ACM, Vol.23, No.6, pp.343-349, 1980.

[10] 石井光雄 : 映像化 マシン, オム社, 1989年.



권 오 봉

e-mail : obgwun@moak.chonbuk.ac.kr

1980년 고려대학교 전기공학과 졸업(학사)

1983년 고려대학교 전기공학과(공학석사)

1992년 일본구주대학교 총합이공학연구과(공학박사)

1992년~1993년 일본구주대학교 정보공학과 조수

1994년~1995년 전북대학교 컴퓨터과학과 전임강사

1996년~현재 전북대학교 조교수

관심분야 : 컴퓨터 그래픽스, 사이언티픽 비주얼라이제이션, 병렬처리