실시간 렌더링의 속도 향상을 위한 소프트웨어적 기법

한영민° 황석민 성미영
인천대학교 컴퓨터 공학과
{ymhan°, hsm0477, mysung}@incheon.ac.kr

Software Method for Improving the Performance of Real-time Rendering

Youngmin Han° Seokmin Hwang MeeYoung Sung
177 Dowhadong, Namgu, 402-749 Incheon, South Korea

요 약

일반적인 렌더링 방식은 응용→기하→래스터화로 진행되는 렌더링 파이프라인 상에서 진행된다. 그래픽 카드의 발전으로 기하 단계의 연산을 GPU가 담당함에 따라 CPU의 연산을 줄여 CPU가 많은 연산을 할 수 있게 되었다. 그러나 이 같은 분배로 인해 CPU와 GPU가 서로 끝나기를 기다리는 병목현상이 발생하게 되었다. 이러한 병목 현상은 효율적인 렌더링을 저해하는 요인이다. 본 연구의 목적은 CPU와 GPU의 병렬처리 과정에서 발생하는 병목현상을 줄여 실시간 렌더링에서 그래픽 출력을 더욱 빠르게 하는데 있다. 이를 위해 본 논문에서는 그래픽 출력 과정 중 CPU 와 GPU 사이에서 하드웨어적으로 처리되고 있는 동기적 처리 과정을 소프트웨어적인 기법을 이용하여 비동기적으로 처리함으로써 성능을 향상시킬 수 있음을 말하고자 한다.

1. 서론

일반적인 렌더링 방식은 응용→기하→래스터화로 진행되는 렌더링 파이프라인 상에서 진행된다. 과거에는 파이프라인의 응용단계의 연산과 기하 단계의 연산은 CPU(Central Processing Unit)가 담당하였고 래스터화 단계의 연산은 그래픽 카드가 담당하였다. 그러나 그래픽 카드의 발전으로 기하 단계의 연산을 GPU(Graphics Processing Unit)가 담당함에 따라 CPU 측이 연산을 적게 하여 CPU가 더 많은 연산을 할 수 있게 되었다 [1],[2]. 그러나 이러한 연산의 분배로 그래픽 출력을 위한 일련의 프로세싱(응용, 기하, 래스터화)이 서로 다른 장치에 나뉘어 행해짐으로써 두 장치가 서로 상대편의 연산이 끝나기를 기다리는 병목현상이 발생하게 되었다. 그래픽 출력의 효율적인 렌더링을 저해하는 병목현상을 효율적인 병렬화를 통하여 유휴시간을 줄여 렌더링 속도를 향상시키는 방법을 제안한다. 또한 병목현상이 발생할 수 있는 장치를 동적으로 체크하여 보다 향상된 그래픽 출력을 제공하는 방법에 대해 논의한다.

이런 논의를 위해 2장에서는 관련연구, 3장에서는 본 논문에서 제안하는 방법, 4장에서는 유효성을 확인하는 실험내용, 마지막으로 5장에서는 결론 및 향후 과제에 대해 기술하겠다.

2. 관련연구

본 논문에서 제안하는 기법은 CPU쪽 연산을 이용하여 그래픽 출력속도를 높이는 것이 목적이다. 이런 목적을 가지고 있는 기법에는 뷰 프러스텀 컬링과 쿼드 트리 알고리즘이 있다. 하지만 이런 기법들은 GPU 연산량 자체를 직접 줄일 수 있는 반면, 본 논문에서 제안하는 기법은 이런 알고리즘들을 적용하는 일련의 과정에서 발생할 수 있는 병목현상 때문에 생기는 유휴시간을 줄일 수 있다는 차이점이 있다.

본 연구는 한국 산업기술평가원(ITEP)의 지원으로 인천정보산업진흥원(IIT)을 통하여 수행되었다.

3. 렌더링 병렬처리

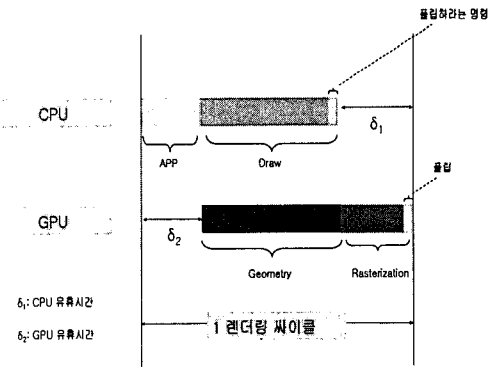

그림 1 일반적 CPU와 GPU의 병렬처리와 발생가능한 유휴시간

그림 1에서 보는 바와 같이 GPU 연산은 CPU 측의 백버퍼(back buffer)와 프런트 버퍼(front buffer)를 바꾸어 주는 플립연산을 끝으로 한 싸이클을 마치게 되는데[4] 일반적으로 이 플립연산이 끝날 때까지 CPU는 GPU를 기다리게 된다. 따라서 본 논문에서는 그림 1에서 보이는 것처럼 두 장치에서 나타날 수 있는 병목현상을 파악하여 소프트웨어적으로 병렬처리를 해 줌으로써 더 빠르고 효율적인 그래픽 출력을 제공하고자 한다. 본 논문에서 제안하는 알고리즘은 그림 2와 같다.

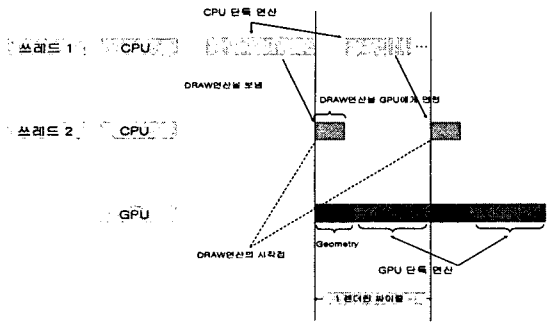

그림 2 본 논문에서 제안하는 방법

우선 응용 단계의 연산(CPU 연산)을 크게 APP, CULL, DRAW의 3단계로 나누었다[5]. APP 단계와 CULL 단계는 CPU만 사용하는 연산으로, DRAW는 CPU와 GPU 모두를 사용하는 연산으로 정의하고자 한다.

그림 2에서처럼 본 논문에서 제안하는 방법에서는 CPU 측 연산을 두 개의 쓰레드로 수행하게 하였다. 첫 번째 쓰레드는 APP와 CULL을 하는 쓰레드로서 두 번째 쓰레드로부터 플립 완료 이벤트를 받을 때까지 멈춰 있도록 구성하였다. 두 번째 쓰레드는 기존의 연산들 중 DRAW 연산을 실행하도록 구성하였다. 두 번째 스레드는 DRAW 연산에 대한 처리 명령을 이벤트로 받으면 실제 DRAW 명령을 GPU쪽으로 내려주는 역할을 하고 이런 연산이 끝났을 때는 종료 이벤트를 첫 번째 쓰레드 쪽으로 전송해주는 역할을 한다. 그림 2와 같이 구성하였을 경우에 그림 3, 그림 4, 그림 5 와 같은 세 가지 현상이 나타날 수 있다.

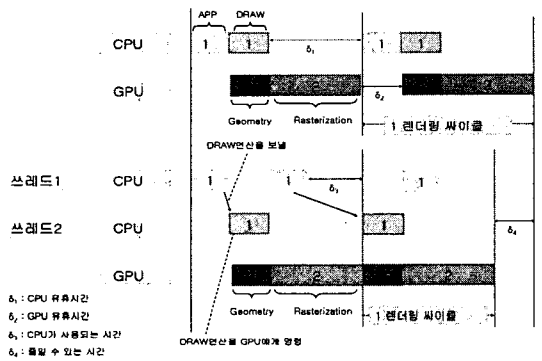

그림 3 GPU가 병목인 경우

그림 3은 GPU 연산 시간이 CPU 연산 시간보다 긴 경우로, 이런 상황이 발생한다면 GPU쪽에 병목현상이 생기는 것을 의미한다. 이러한 상황에서 기존의 방법으로 렌더링을 한 하나의 예를 보면, 한 프레임을 렌더링 하는데 걸리는 시간이 4 단위 시간이 걸리고, CPU 쪽에서는 2 단위 시간이, GPU쪽에서는 1 단위 시간이 아무런 연산도 하지 않고 대기 하고 있는 것을 볼 수 있다.

본 논문에서 제안 하는 방법을 사용하면 한 프레임을 렌더링 하는데 걸리는 시간이 3 단위시간이 되어 기존의 방법으로 렌더링 한 경우보다 1 단위시간을 줄여 전체적으로 렌더링 속도를 빠르게 해주는 결과를 얻을 수 있다. 또한 GPU쪽의 유휴시간은 제거하고, CPU쪽의 유휴시간은 1 단위시간 동안에 CPU가 쉬지 않고 다른 연산을 할 수 있어 1단위시간을 줄이는 효과가 있었다.

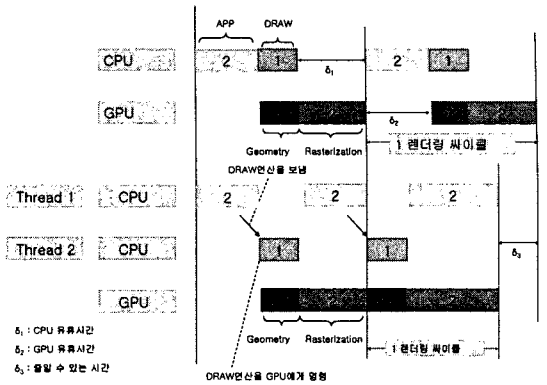

그림 4 이상적인 경우

그림 4는 플립하는데 걸리는 시간과 CPU 단독연산 시간이 같은 이상적인 경우로서 유휴시간이 전혀 발생하지 않는 경우이다.

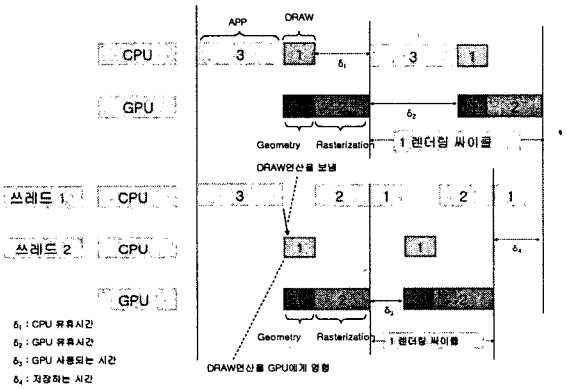

그림 5 CPU가 병목인 경우

그림 5는 플립하는데 걸리는 시간이 CPU 단독시간보다 짧아 CPU쪽에 병목현상이 발생하는 경우이다. 한 프레임을 렌더링 하는데 걸리는 시간이 6 단위시간이 걸렸으며, CPU쪽에서는 2단위시간이, GPU쪽에서는 3 단위시간이 대기하고 있는 것을 볼 수 있다. 그러나 본 논문이 제안하는 방법을 사용하면 한 프레임을 렌더링 하는데 4 단위시간이 되어 2 단위시간을 줄여 전체적인 렌더링 속도를 빠르게 할 수 있었다. 또한 CPU쪽의 유휴시간은 제거하고, GPU쪽은 1 단위시간 동안 더 많은 연산을 할 수 있는 효과가 있었다.

위의 경우에서 보는 것처럼 만약 매 프레임마다 병목현상이 발생하는 장치와 병목시간을 알 수 있다면 병목현상이 발생하지 않는 장치에 병목시간만큼의 연산을 추가로 할당하여 효율적인 그래픽 렌더링이 가능하게 할 수 있다.

4. 실험내용

본 절에서는 3절에서 소개한 방법에 대해 실험을 통하여 그 효과를 확인해 보고자 한다. 실험을 위하여 그림 1의 일반적인 경우와 그림 2의 제안하는 기법을 적용한 경우를 측정 할 수 있는 프로그램을 각각 작성하였다. 실험환경은 표 1과 같다.

표 1 실험 환경

CPU	Pentium4-2.54Ghz
RAM	1.00GB
VGA	ATI RADEON 9700 PRO
LIBRARY	DIRECTX 9.0[2]
개발 환경	VC++ 6.0
사용 언어	C++
모드	1280*944
사용색	32비트 트루 컬러

CPU 연산은 Sleep() 함수를 사용하여 이 함수의 인자 값을 조정함으로써 연산의 처리에 소요되는 CPU 시간의 변화량을 나타내었고, GPU 연산에 대하여는 그려지는 삼각형의 개수를 조정함으로써 연산의 변화량을 나타냈다. 동일한 CPU 시간과 GPU 시간을 설정한 후에 일반적인 방법으로 렌더링 하는 경우와, 본 논문에서 제안하는 방법인 DRAW 전담 쓰레드를 사용하여 렌더링 경우의 각각에 대하여 작성한 두 개의 프로그램을 실행시켜 FPS(Frames Per Second)를 측정해 보았다. 실험은 일정한 수의 삼각형 그리기(800,000개, 1,000,000개, 1,500,000개, 그리고 2,000,000개)에 대하여, 즉 정해진 GPU 연산량에 대하여 CPU 연산량을 0ms, 10ms, ···..100ms 등으로 단계별로 변화시켜 이때 변하는 프레임 수를 측정하는 방법으로 진행하였다.

그림 6, 7, 8, 9에서 보는 바와 같이, DRAW 전용 쓰레드를 사용한 경우 렌더링 되는 프레임수가 증가하는 것을 볼 수 있었다. 위의 네 가지 실험 모두에서, CPU 연산량이 늘어남에 따라 GPU 병목 상황에서 이상적인 상황을 거쳐 CPU 병목 상황으로 전환됨을 볼 수 있었다. 또한 삼각형의 개수를 늘릴수록 프레임수가 떨어지는 시점이 늦춰짐을 확인할 수 있었는데, GPU 쪽 연산이 증가하여 CPU 쪽 유휴시간이 더 많이 생길 수 있기 때문에 나타나는 현상으로 볼 수 있다.

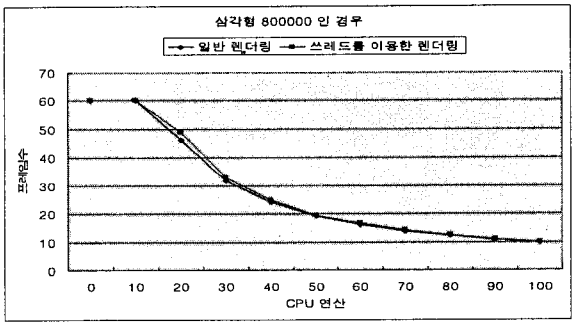

그림 6 렌더링하는 삼각형 수가 800,000개인 경우

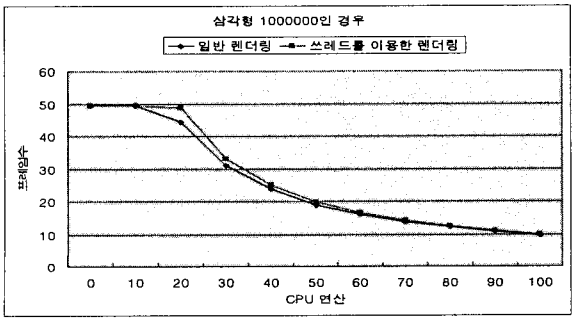

그림 7 렌더링하는 삼각형 수가 1,000,000개인 경우

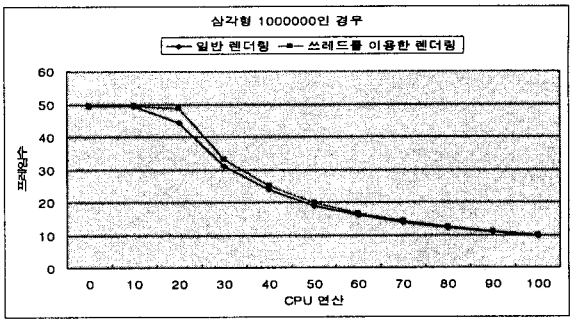

그림 8 렌더링하는 삼각형 수가 1,500,000개인 경우

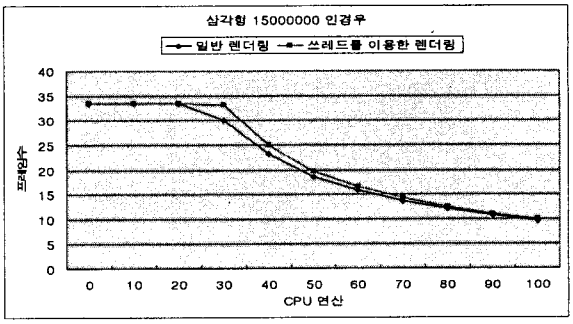

그림 9 렌더링하는 삼각형 수가 2,000,000개인 경우

5. 결론

본 논문에서는 CPU와 GPU의 병렬처리 과정에서 발생하는 병목현상을 줄여 실시간 렌더링에서 그래픽 출력을 빠르게 하는 소프트웨어적인 비동기적 처리 기법을 제안하였다. 렌더링을 위한 CPU 연산과 GPU 연산의 병렬처리에 있어서 CPU쪽의 연산을 순수 CPU 작업(APP와 CULL)을 하는 쓰레드와 GPU 제어 연산(DRAW)을 제어하는 쓰레드로 나누어 두 가지 연산을 비동기적으로 수행하는 방법을 제안하였고, 실험을 통하여 제안하는 방법이 그래픽 출력 과정에서 나타날 수 있는 CPU와 GPU 장치의 병목현상을 줄일 수 있음을 확인하였다. 향후에는 그림 4와 같은 이상적인 상황이 발생하는 GPU 연산량과 CPU 연산량의 비율을 계산하여, 실시간으로 GPU 연산량에 따른 CPU 연산량의 최적치를 유지하도록 조절함으로써 더욱 빠른 실시간 렌더링 효과를 가져오게 하는 연구를 진행하고자 한다.

참고문헌

[1] Akenine-Moller, T., Haines, E.: Real-Time Rendering. 2nd edition, A K PETERS (2002)

[2] Gray, K.: The Microsoft DirectX 9 Programmable Graphics Pipeline. Microsoft Press (2003)

[3] Wimmer, M., and Wonka, P.: Rendering Time Estimation for Real-Time Rendering. In proceedings on Eurographics Symposium on Rendering 2003 (2003) 118-129

[4] Cox, Michael, Sprague D., Danskin., Ehlers., Hook B., Lorensen B., and Tarolli G.: Developing High-Performance Graphics Applications for the PC Platform. In Course 29 notes at SIGGRAPH 98 (1998)

[5] Rohlf, J., Helman, J.: IRIS Performer: A High Performance Multiprocessing Toolkit for Real-Time 3D Graphics. In proceedings on SIGGRAPH 94 (1994) 381-394