한국게임학회 논문집
제1권 제1호 2001년 11월

DirectX Graphics기반 게임용 3D 렌더링 최적화에 대한 연구

우석진*, 김경식**
* 호서대 산업경영벤처 학원 컴퓨터게임학과 석사 dol93@freechal.com
** 호서대 게임공학전공 교수, kskim@office.hoseo.ac.kr

A Study on an Optimization of 3D Rendering for Games using DirectX Graphics

Seok-Jin Woo* and Kyung-Sik Kim**

Abstract

DirectX Graphics plays the role of hardware independent 3 dimensional drawing interface for 3 dimensional video display. However the APIs in DirectX Graphics provide not only small improvement in velocity in the lowest level but also unstable performance of velocity according to their usages. In this paper, we present the structure of 3D game engine of efficient performance and describe functions and implementational features of game engines for an efficient 3D rendering in the environment of DirectX Graphics.

KeyWord *: DirectX Graphics, DirectX SDK, DirectDraw, Direct3D*

1. 서 론

DirectX 8.0 에서부터 DirectDraw와 Direct3D가 통합되어 DirectX Graphics가 되었다. DirectX Graphics의 특징은 사용하기 쉬워진 인터페이스와 최신 그래픽 하드웨어 지원 그리고 세이더(shader)와 같은 기능이다[1].

DirectX Graphics의 주된 역할은 장치 독립적인 방식으로 3D 비디오 디스플레이 하드웨어를 위한 3D드로잉(Drawing) 인터페이스이다[2-4]. 그러나 DirectX Graphics에서 제공하는 API는 하위레벨(low level)에서 약간의 속도 향상 기능만을 제공할 뿐 오히려 사용 방법에 따라 커다란 속도 성능의 차이를 보인다. 특히 DirectX SDK (software development kit)에 제공되는 예제들은 가독성을 높이기 위해 속도최적화가 거의 수행되어져 있지 않기 때문에 이를 게임에 직접 응용하는 것은 엔진의 성능을 떨어뜨릴 수 있다.

본 논문에서는 DirectX Graphics측면에서 효율적인 성능을 낼 수 있는 3D엔진의 구조를 제시하고 DirectX Graphics 기반에서 3D 렌더링을 효율적으로 수행하기 위한 상위 레벨(high level)의 게임 엔진 기능, 구현상 특성에 관한 연구 내용을 기술하고자 한다.

게임 엔진의 속도 최적화 기법에서 DirectX Graphics은 철저한 상태 기계(state machine)구조라는 점을 고려하여 상태를 최소한으로 변경시킴으로써 기본적인 최적화를 수행한다. 이를 위해 변환과 클리핑(clipping)을 행하는 그래픽 파이프라인 (graphic pipeline) 처리과정에서 발생할 수 있는 작업을 최대한 배치(batch)화 시키는 것이 중요하다. 그 다음으로는 차세대 그래픽 카드 구조에 최적으로 수행될 수 있는 자료구조로 데이터를 설계하여야 한다는 것이다. 거기에 차세대 그래픽카드에 최적화하기 어려운 동적

렌더링을 보안하기 위해서 AGP(Accelerated Graphics Port) 메모리를 사용하여 보다 효과적인 성능을 발휘할 수 있다.

본 연구에서 제시한 DirectX Graphics측면에서의 3D 렌더링 최적화 기법을 사용하여 3D게임을 제작할 경우 높은 그래픽 연출 속도성능 향상 효과를 얻을 수 있을 것으로 기대한다.

2. DirectX Graphic기반 효율적 3D엔진구조

2.1 재질 그룹화

DirectX는 상태 기계(state machine)로 되어 있어서 한 번 상태를 정해 놓으면 그 상태를 계속 유지하고 있다. 그러나 상태를 변화하는 과정에서 많은 성능 저하가 발생하므로 그것을 최소화 하기위해 상태 변화를 기반으로 그룹핑 렌더링을 해야 한다(그림 1).

렌더링 과정에서 상태를 변화시키는 것 중에 가장 큰 기준이 되는 것은 재질이다. 따라서 3D 객체를 생성하는 과정에서 객체의 재질을 기준으로 같은 재질 즉, 상태가 같은 것끼리 그룹핑을 수행함으로써 이를 개선할 수 있게 되는 것이다. 그리고 이 과정은 실시간으로 수행하는 것이 아니라 객체를 생성하는 과정에서 미리 수행하여 불필요한 반복 계산과정을 줄일 수 있다.

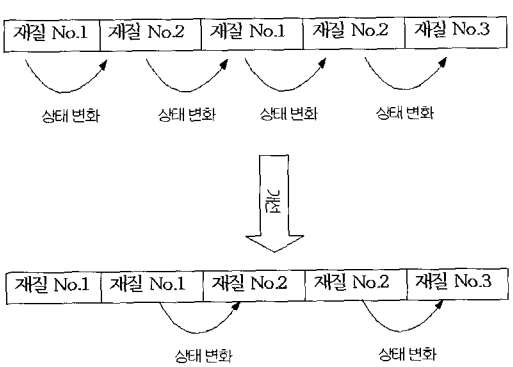


그림 1. 재질 그룹핑(material grouping)

2.2 변환과 조명(T&L)에 최적화된 정점 버퍼

3D 렌더링 엔진은 일반적으로 3D 그래픽 지오메트리 파이프라인(geometry pipeline)이라는 일련의 처리과정(그림 2)을 수행하는데 이 과정에 있어서 가장 많은 계산이 요구되는 부분이 변환(transform), 조명(light) 그리고 클리핑(clipping) 과정이다.

대부분의 3D 그래픽 가속기(3D graphic accelerator)는 클리핑을 기본적으로 지원해주고 있고, 차세대 그래픽 카드들에는 T&L과정역시 하드웨어적으로 처리해 주고 있다.

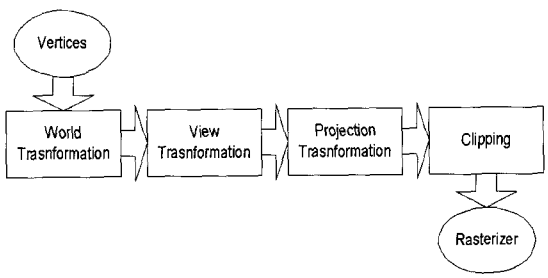


그림 2. 그래픽 파이프라인(graphic pipeline)

차세대 그래픽 카드의 데이터 최적화된 정점 버퍼를 데이터를 구성하는 것이 중요하다. 정점 버퍼는 그래픽 파이프라인에서 사용되는 정점들이 담겨진 메모리를 말하며 사용용도에 따라 일정한 순서로 필요한 것만 조합해서 정정 버퍼의 형태를 직접 만들 수 있으며, 데이터 액세스(access)과정에서 읽기는 빠르지만 쓰기는 상당히 느리다는 특징을 갖고 있다.

따라서 T&L에 사용되는 정점 버퍼는 한 번 쓰기를 해두고 될 수 있으면 변형시키지 않아야만 최적의 속도를 낼 수 있다. 이러한 정점 버퍼를 정적 정점 버퍼라 하고 게임에서는 정점버퍼의 내용이 변형되지 않는 객체를 표현하는데 사용된다. 이와 반대되는 개념이 동적 정점 버퍼인데 이는 실시간 즉, 렌더링 과정에서 매 프레임마다 읽기와 쓰기가 수행되어지는 정점 버퍼를 말하고, 실시간 정점버퍼의 내용이 변경되는 부분에 사용된다. 따라서 객체의 정점 버퍼의 내용이 변경되지 않는 정점 버퍼는 모두 T&L에 최적화 시킴으로써 T&L 가속기에 최대한의 이득을 얻을 수 있게 되는 것이다.

2.3 동적 정점 버퍼의 성능 향상을 위한 AGP의 사용

AGP(Accelerated Graphics Port) 인터페이스는 1997년 Intel Pentium II프로세서와 함께 소개된 것으로, 비디오카드가 메인 시스템 메모리에 고속으로 접근할 수 있게 하는 메커니즘을 제공한다(그림 3). AGP는 초기 최대 전송률이 266MB/s였지만, AGP4X로 알려져 있는 현재 사양에서는

최대 전송률이 1,067MB/s이다. 이는 일반 PC 시스템에 흔히 쓰이는 표준 PCI버스보다 8배나 빠른 것이다 [5]. DirectX Graphics에서 이러한 수행을 하기 위해서는 IDirect3DDevice8 인터페이스의 CreateVertexBuffer() 메서드를 호출 할 때 D3DUSAGE_DYNAMIC과 D3DUSAGE_WRITEONLY용법 플래그들을 포함시켜야 한다. 앞의 플래그는 프로그램이 정점 버퍼 안의 데이터를 주기적으로 변경할 것임을 알려주는 역할을 하며, 뒤의 플래그는 버퍼를 읽지는 않고 쓰기만 할 것임을 알려준다. 그리고 버퍼를 기본 메모리 풀(D3DPOOL_DEFAULT)에 할당하도록 지정하면 정점버퍼는 AGP메모리 안에 할당된다.

시스템 메모리는 기본적으로 일반적인 응용 프로그램들에 의해 쓰이기 위해서 어떤 응용프로그램들이 실행되고 있는지, 그리고 메모리 할당들이 구체적으로 어떻게 일어났는지에 따라 상당한 메모리 단편화가 일어나게 된다. 그래픽 카드가 AGP버스를 통해 접근할 메모리 할당을 요청했을 경우, 실제로는 메인 메모리 상의 여러 개의 비연속적인 조각들이 할당 될 수도 있다. 그 조각들을 하나의 연속적인 메모리 블록처럼 보이게 하기 위해서, 그래픽 카드와 칩셋(프로세서와 메인 메모리 사이의, 또한 그들과 PCI버스 사이의 통신을 제어)은 그래픽 주소 재 매핑 테이블(graphics address remapping table, GART)이라는 것을 사용한다. GART에 의해 재미핑되는데 쓰일 수 있는 메모리의 양은 시스템마다 다르나, 일반적으로 시스템 메모리의 절반이 최대 한계이다.

그래픽 카드는 정기적으로 메모리에 접근할 경우 무엇보다 중요한 것은 성능이므로 AGP 버스에 의해 접근되는 메모리가 디스크에 스와핑(swapping)되지 않게 하는 것이 중요한데, 이를 위해 AGP 메모리에 "캐시 불가"라는 플래그를 설정한다.

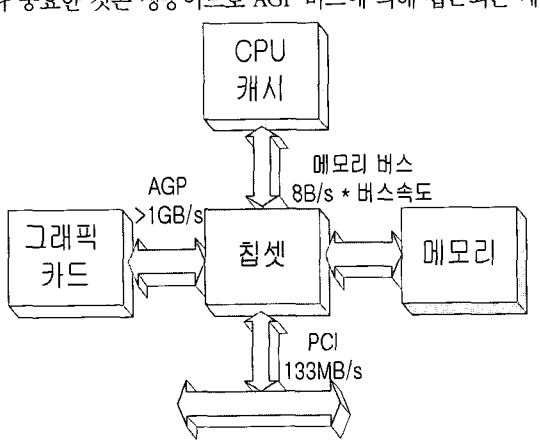


그림 3. PC Architecture of AGP 4x

3. 3D 렌더링 최적화

3.1 조명

3D 엔진에서 제공되는 조명의 종류에는 Ambient, Directional, Point, Spotlight, Specular 등의 조명이 있는데, 열거된 순서대로 점점 많은 계산이 요구되어 진다. 따라서 사용자가 원하는 조명효과를 위해서는 열거 순서대로 활용하여 복잡한 계산을 줄여주고, Point 조명과 Spot 조명은 광원의 범위 값이 존재하기 때문에 꼭 필요한 크기의 영역 값을 사용하여 계산을 최소화 시켜준다.

게임에서 가장 많이 사용하는 조명이 Point 조명인데 실제 DirectX Graphics을 통해서 가속기 지원 하에 사용할 수 있는 조명의 개수는 몇 개 되지 않는다. 가속기의 지원 여부에 따라 많아야 32개 정도인데, 게임에서 3개 이상을 사용하면 상당한 속도 저하 현상이 발생된다. 하지만 게임에서 사용되는 조명의 수는 수백, 수천 개가 등장할 수 있는데 이러한 문제를 해결하기위해서는 화면에 보이는 객체들을 기반으로 그 중에 영향을 받는(가장 가까운) 조명만 처리하는 것으로 이러한 문제를 해결 할 수 있고 객체에 영향을 주는 조명의 최대 개수는 3개를 넘지 않게 처리해서 속도 저하 현상을 막을 수 있다.

3.2 텍스처

텍스처의 크기가 작을수록 상태 변경이 적을수록 재질 상태 변경 속도가 빨라진다. 앞에서 살펴본 것처럼 재질에 따라 그룹핑을 수행하는 과정에서 작은 크기의 여러 텍스처를 사용하는 경우 작은 조각 하나하나가 모두 다른 재질 값을 갖는다. 이러한 문제를 해결하기위해서는 한 장에 여러 개의 텍스처를 한꺼번에 담아두어 텍스처의 상태 변경을 줄여 속도를 높일 수 있다. 그리고 대부분의 3D 가속기가 정사각형 텍스처만 지원하므로 텍스처의 모양을 정사각형을 사용하고, DirectX Graphics은 256X256 크기의 텍스처에 최적화되어 있으므로 실제로 좋은 성능을 얻기 위해서는 텍스처의 모양과 크기는 256X256을 사용한다.

텍스처는 3D게임에서 가장 많은 자원을 소비하는 부분이

기도 한데, 이를 위한 해결방안으로 텍스처를 캐시 메모리에 담아 처리하거나 가속기의 압축텍스처의 지원 여부에 따라 압축 텍스처를 사용할 수도 있다. 압축 텍스처를 사용할 경우 텍스처 메모리의 크기를 8배까지 줄일 수 있다.

3.3 Z 버퍼

Z 버퍼는 렌더링 되어질 객체의 깊이 정보(Z value)를 픽셀단위로 저장하기위한 그래픽 메모리 영역인데 그려질 픽셀의 Z 값을 Z 버퍼의 Z값과 비교하여 그 픽셀의 앞에 또는 뒤에 있는지의 여부를 판단하여 픽셀을 그리게 된다.

Z 버퍼를 사용할 때 폴리곤을 화면 앞에서부터 뒤로 그리게 되면 Z 버퍼와 비교만 할 뿐 실제 Z 버퍼와 화면이 갱신되질 않아 엔진 성능을 향상시킬 수 있다. 따라서 화면에 그려질 객체를 Z값을 기준으로 하여 정렬(sort)한 다음에 렌더링을 수행하게 되는데 오히려 폴리곤 단위의 정렬연산은 더 많은 계산이 요구되기 때문에 그룹핑 단위의 객체를 기준으로 정렬하면 최적의 효과를 얻을 수 있다.

Z 버퍼는 매 프레임마다 가장 먼 값으로 갱신되어져야만 올바른 수행을 할 수 있는데 만약 화면 전체가 배경이나 바닥같이 가장 멀리 보이는 객체에 의해 가려질 경우 Z 버퍼를 지울 필요 없이 가장 Z값이 큰 객체를 그릴 때 Z 버퍼 비교 연산을 하지 않고 그리게 되면 자동적으로 가장 먼 값으로 Z 버퍼가 채워지기 때문에 성능향상 효과를 볼 수 있다.

3.4 동적 메모리할당의 제거

상위레벨의 API를 설계/제작 하는 과정에서 프레임단위의 new와 delete등을 사용하여 동적으로 메모리를 할당하여 처리하는 경우가 많다. 이러한 구조는 개발하기는 쉽지만 메모리의 단편화(fragmentation)가 발생하고 메모리 블록의 재배치작업이 수행되기 때문에 게임 수행 성능이 나빠지게 된다 [6]. 이러한 문제에 대한 해결책은 프레임 기반 메모리(frame-based memory)를 사용함으로써 메모리 할당 시간을 줄이고 단편화를 없앨 수 있다 [7]. 프레임 기반의 메모리를 사용할 경우 데이터의 양이 클 경우에는 캐싱(Caching) 알고리즘을 사용하여 한정된 메모리 크기를 극복할 수 있다.

3.5 실험 결과

본 논문에서 최적화 실험 테스트로 사용된 데이터는 Face 수가 약 800개 정도 되는 데이터로 펜티엄2 400MHz에 Geforce2MX에서 실험한 결과, 데이터 100개(즉 Face 80,000) 경우 해상도 640X480, 16 bit color환경에서 약 48FPS (Frame Per Second)의 결과가 나왔다. 조명도 Point Light 1개만 사용하였고 정적 버텍스 버퍼를 사용한 결과이다.

4. 결 론

본 논문에서는 DirectX Graphics측면에서 효율적인 성능을 낼 수 있는 3D엔진의 구조를 제시하였고 DirectX Graphics기반에서 3D 렌더링을 효율적으로 수행하기 위한 상위 레벨(high level)의 게임 엔진 기능, 구현상 특성에 관한 연구 내용으로서 DirectX Graphics적 측면의 최적화의 기본인 상태 변경의 최소화, 재질 단위의 그룹핑 렌더링의 이점, Z버퍼 기법, 차세대 그래픽 카드의 성능과 동적 메모리처리를 위한 AGP메모리 활용, 메모리 단편화를 줄이기 위한 정적 메모리의 사용과 이의 한계를 극복하기위한 캐싱의 사용 등을 기술 하였다.

물론 여기서 제시한 것들은 게임 기획에 따라 다른 방식으로 최적화 될 수 있을 것이다. 앞으로는 GeforceIII, X-Box 등에서 제공되는 다양한 정점 셰이더(vertex shader)와, 픽셀 셰이더(pixel shader) 등의 기능이 추가적으로 개발되어져야 하며, 이러한 차세대 게임기에 적합한 엔진구조와 기능들이 일반화되어져서 엔진 자체를 제작하는데 많은 시간을 보내기보다는 게임 자체를 만드는데 모든 자원이 투자되었으면 하는 바람이다.

참고문헌

[1] DirectX 8 實踐programming, 工學社, 2001. 4.

[2] Adrian Perez, Dan Royer, 이주리 역, Advanced 3D Game Programming with DirectX, 민프레스, 2001. 3.

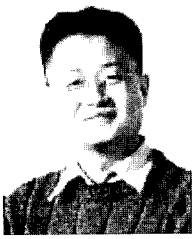
[3] Jone DeGoes, 3D Game Programming with C++, Coriolis, T3entertainment, 2001. 9.

[4] Todd Barron, Multiplayer Game Programming, Prima Tech, 2000.

[5] Dean macri, "Game Developer-Fast AGP Writes", June. 2001.

[6] Mickey Kawick, " Real-Time Strategy GAME PROGRAMMING Using MS DirectX", WORDWARE Press. may 2000.

[7] Mark Deloura외/류광 역, "Game programming GEMS", 정보문화사, 2001.01.



우 석 진

1999 년 호서대학교 컴퓨터 공학과(학사)
2001 년 호서대학교 컴퓨터 게임과(석사)
1999 년~2000 년 ㈜지디온 게임 개발팀 선임연구원
2000 년~2001 년 ㈜팬택네트 게임 개발팀 연구원
2001 년~현재 ㈜에이치씨아이 연구개발본부 연구원
관심분야: 3D 게임 엔진, 프로그램 분석&설계, 게임 기획

김 경 식

982년 서울대학교 전산기공학과 (학사)
1984년 서울대학교 전산기공학과 (석사)
1990년 서울대학교 컴퓨터공학과 (박사)
1984년~1991년 한국전자통신연구원 선임연구원
1991년~현재 호서대학교 컴퓨터공학부 게임전공주임
관심분야: 게임디자인, 게임프로그래밍, 게임제작전반