Journal of Korea Multimedia Society Vol. 9, No. 1, January 2006(pp.41-50)

PDA 플랫폼을 위한 적응형 Z-버퍼 알고리즘

김효철[†], 김대영[††]

요 약

이 연구는 현재 툴 제공 업체 수준에서 지원이 미미한 PDA 플랫폼의 3D 그래픽 소프트웨어 엔진의 실효성을 개선시키고 그래픽 엔진의 핵심 부분인 래스터라이저의 성능 향상을 목적으로 진행되었다. 상대적으로 약한 프로세싱 파워를 가진 플랫폼에서 소프트웨어로 3D 그래픽 엔진을 구현하는 것은 많은 문제점이 있으나, 우리는 현재 널리 사용되고 있는 깊이정렬 알고리즘과 Z-버퍼 알고리즘의 장점을 취하고 문제점을 보완하여 적응형 Z-버퍼 알고리즘을 구현하고 여러 가지 PDA 플랫폼들을 사용하여 실험하였다. 새로운 알고리즘의 속도는 두 알고리즘의 중간 정도로 나타났으며, 깊이정렬 알고리즘과는 달리 순서 역전에 따른 오류가 발생하지 않음을 확인하였다.

An Adaptive Z-buffer Algorithm for PDA Platform

Hyo Chul Kim[†], Dae Young Kim[††]

ABSTRACT

This paper aims to improve the efficiency of a 3D-graphic software engine in a PDA platform and the performance of a rasterizer. There are many problems in implementing a software engine in a mobile platform, due to its relatively weak processing power. Taking the advantages and complementing weaknesses of the depth-sort algorithm and the Z-buffer algorithm, we implemented an adaptive Z-buffer algorithm and proved its performance on several PDA platforms. Our algorithm was evaluated to have midway speed compared with two conventional algorithms. It also removed reversal errors in comparison with the depth-sort algorithm.

Key words: Mobile 3D Graphic Engine(모바일 3차원 그래픽 엔진), Depth-sort Algorithm(깊이-정열 알고리즘), Z-buffer Algorithm(Z-버퍼 알고리즘), Rasterizer(래스터라이저)

1. 서 론

휴대폰이나 PDA 플랫폼들을 겨냥한 모바일 분야의 신기술들의 눈부신 발전과 더불어 모바일 컨텐츠 제작 기술도 동영상과 같은 단순 멀티미디어 서비스는 물론 모바일 3D 그래픽 엔진을 활용한 3D 게임, 3D 채팅이나 아바타 서비스 등과 같은 현실감을 강조한 컨텐츠들이 속속 출현하고 있다. 초기의 빈약한 처리 속도로 관심을 끌지 못했던 분야에서도 모바일 플랫폼들에 대한 하드웨어 설계 및 성능 향상 노력에 힘입어 데스크탑 PC 수준에 근접하는 3D 컨텐츠 구현이 가능하게 되었고, 가상 쇼핑몰이나 박물관과 같은 가상 현실 컨텐츠 서비스도 머지않아 개시될 것으로 전망된다. 3D 게임 분야에서도 이러한 경향은 뚜렷하게 나타나는데, 3D 그래픽 전용 칩을 내장한 게임 전용폰들이 속속 출시되고 있으며 인기를 끌고 있다. 그러나 PDA 분야에 대한 3D 그래픽 지원 노력은 여러모로 소홀한 편이다. 이것은 스마트폰과 같이

※ 교신저자(Corresponding Author) : 김효철, 주소 : 대구광역시 달서구 신당동 700(704-703), 전화 : 053)589-7593, FAX : 053)589-7595, E-mail : khc@km-c.ac.kr
접수일 : 2005년 9월 28일, 완료일 : 2006년 1월 20일

[†] 정회원, 계명문화대학 컴퓨터인터넷학부
[††] 계명문화대학 멀티미디어학부
(E-mail : dyk@km-c.ac.kr)

기존의 휴대폰에 PDA의 기능을 통합한 슈퍼 휴대폰들의 출시로 전용 PDA 기기들에 대한 대중의 관심이 다소 떨어진 때문이기도 하겠지만, 아직은 휴대폰과 PDA는 고유의 영역에서 슈퍼 휴대폰들에 비해 현격한 성능 우위를 점하고 있는 것도 사실이다.

Pocket PC는 WinCE Family 중에서도 PDA를 위해 설계된 운영체제이며, PIMS(Personal Information Management System)가 기본이지만 간단한 GameAPI (=GAPI)도 포함되어 있다. GAPI는 데스크탑 PC의 게임 라이브러리인 DirectX와 유사한 배경으로 개발되었으며, 사무용 응용 프로그램에 최적화된 PDA에서 복잡하고 느린 GDI만으로는 게임 개발자들의 욕구를 만족시킬 수 없다는 현실적인 요구를 반영한 결과이다. 그러나 GAPI는 DirectX와 유사한 프로그래밍 방식, 간단함 등의 장점은 있지만, DirectX의 DirectDraw, DirectSound, DirectInput, Durect3D 등 다양한 지원과는 달리 10여 개의 DrextDraw, DirectInput에 해당하는 간단한 API만을 제공하여 개발자들에게 기초적인 지원을 제공하는 수준이었으며 더블 버퍼링과 같은 간단한 API 조차 포함되어 있지 않았다. 게임 개발자들은 최초 GAPI가 발표된 2000년 4월 이래로 획기적인 차기 버전에 대한 기대를 지속적으로 가져왔으나, Pocket PC 기반의 일반 PDA에서는 3D 그래픽을 위하여 사용할 수 있는 표준 라이브러리가 현재까지도 지원되지 않고 있다. 따라서 간단한 3D 그래픽이라도 구사하려면 모델링은 물론 래스터라이져를 포함한 그래픽 파이프라인 전체를 직접 제작하여 사용해야 하는 실정이다.

이 연구는 현재 툴 제공 업체 수준에서 지원이 미미한 PDA 플랫폼의 3D 그래픽 소프트웨어 엔진의 실효성을 개선시키고 그래픽 엔진의 핵심 부분인 래스터라이져의 성능 향상을 목적으로 진행되었다. 상대적으로 약한 프로세싱 파워를 가진 플랫폼에서 소프트웨어로 3D 그래픽 엔진을 구현하는 것은 많은 문제점이 있다. 현재 가장 널리 사용되는 Z-버퍼 알고리즘(Z-buffer algorithm)은 단순함과 정확성에 있어서는 다른 알고리즘에 비해 탁월하지만 속도면에서는 PDA에 적용하기에는 무리가 따른다. 다소 정확성을 희생하더라도 속도면에서 유리한 방법으로는 단순화된 페인터의 알고리즘(painter's algorithm)을 들 수 있다. 단순화된 페인터의 알고리즘은 특별한 상황을 고려하지 않고 랜더링이 필요한 모든 다각형의 면(face)들을 Z 값만을 기준으로 하향식으로 정렬하여 표시하므로 깊이 정렬 알고리즘(depth sort algorithm)이라고도 한다[1]-[3]. 우리는 이 두 알고리즘의 장점을 취하고 문제점을 보완하여 적응형 Z-버퍼 알고리즘(adaptive Z-buffer algorithm)을 구현하여 실험하였다.

다음 장에서는 모바일 3D 그래픽 엔진의 개요를 기술하였고 3장에서는 3D 그래픽 엔진의 핵심이 되는 래스터라이져 구현에 사용되는 깊이 정렬 알고리즘과 Z-버퍼 알고리즘에 대하여 설명하였다. 4장에서는 이 연구에서 제안하는 적응형 Z-버퍼 알고리즘을 기술하였다. 5장에서는 이 알고리즘들을 이용한 실험 결과들을, 6장에서는 결론에 대하여 언급하였다.

2. 모바일 3D 그래픽 엔진

모바일 3D 그래픽을 위한 엔진의 기본구조는 그림 1과 같이 전처리기(preprocessor)와 트랜스폼 파이프(transform pipe), 래스트라이져(rasterizer) 등으로 이루어진 그래픽 파이프라인(graphic pipeline)으로 구성되어 있으며, 전처리기를 통하여 생성된 3D 객체들의 데이터들을 그래픽 파이프라인을 통과

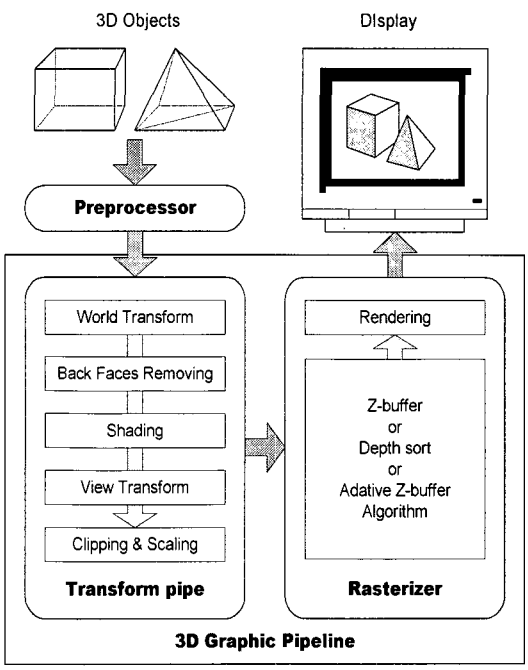

그림 1. 3D Graphic Engine

시키면서 궁극적으로 화면상의 픽셀(pixel)들로 전환시킨다[4-8]는 점에서 데스크탑 PC의 그것과 같다. 데스크탑 PC나 워크스테이션에서는 이러한 과정들이 그래픽 카드에 포함된 3D 하드웨어 가속장치(hardware accelerator) 및 드라이버 혹은 OpenGL, DirectX 등을 포함하는 다양한 그래픽 API들로 제공되고 있다. 그러나 PDA 환경에서는 3D 하드웨어 가속장치나 API들이 제대로 지원되지 않으며, 화면 크기 및 지원 색상, 저전력 및 저주파의 버스 사용으로 인한 제약으로 인해 데스크탑 PC에 비해 알고리즘 선택, 최적화 등이 더욱 요구된다. 심지어 그래픽 파이프라인에 포함되는 소프트웨어 모듈의 처리 순서조차도 최적화를 위하여 변경될 수 있다.

2.1 전처리기(preprocessor)

그리고자 하는 3D 객체(object)들을 정점(vertex)들과 이 정점들을 포함하는 삼각형 형태로 단순화시킨 면(face)들의 집합으로 구성된 그래프(graph) 형식의 데이터 구조로 표현하고, 오브젝트 컬링(object culling) 등의 처리 과정을 거친다. 전처리기의 성능은 적절한 데이터 구조와 기하 알고리즘의 선택에 의존적이다[9-11]. 그림 2는 정육면체 오브젝트에 대한 데이터 구조를 보인 것이다.

이 연구에서는 3D 모델러를 통하여 작성된 객체들을 임포트(import)하기 위한 파일 형식으로 표준 형식이라 할 만한 오토데스크(Autodesk™)의 DXF (Drawing Interchange File) 대신에 가상현실 분야에서 사용되는 REND386의 PLG(PoLyGon)를 사용하였다. PLG 형식을 사용하면 파서(parser) 부분을 간단하게 프로그래밍할 수 있으며, DXF 파일이 주어지더라도 변환기를 사용하여 손쉽게 형식 변환이 된다는 장점이 있다.

2.2 트랜스폼 파이프(Transform Pipe)

이 모듈에서는 객체의 정점들의 위치 값을 로컬 좌표계(local coordinates system)에서 월드 좌표계(world coordinates system)로, 다시 이것을 카메라 위치를 고려한 뷰 좌표계(view coordinates system)를 적용하여 계산한다. 이 과정중의 적절한 위치에서 가려진 면이나 정점들을 제거하는 후면 제거(back faces removing), 각 면의 법선 벡터(vertical vector)에 대한 광원의 효과를 계산하는 세이딩(shading) 및 원근의 효과를 극대화하기 위한 프로젝션(projection), 화면 모서리에서 잘린 부분들을 고려하여 삼각형을 재구성하는 클리핑(clipping), 스케일링(scaling) 등의 처리를 한다[6].

2.3 래스터라이저(Rasterizer)

래스터라이저는 3D 그래픽 엔진에서 가장 시간이 많이 소요되는 부분으로, 객체들의 삼각형 리스트와 트랜스폼 파이프에서 계산된 정점들의 색상과 좌표 값들을 이용하여 화면 좌표(screen coordinates)값과 색상을 계산하여 더블 버퍼(double buffer)에 저장한다. 이 때 생기는 병목현상은 특히 게임과 같은 응용에서 명백하게 나타난다. 이상적인 래스터라이져는 화면상에 표시될 픽셀들에 대한 계산을 한 번만 수행하는 것이겠으나, 면들이 겹쳐지는 경우와 같이 이미 계산한 픽셀 값을 다시 계산해야 하는 경우가 빈번하게 발생한다. 최종적인 픽셀 값은 카메라로부터의 거리(화면 깊이, Z-value)가 가장 가까운 삼각형에 대한 값이 된다.

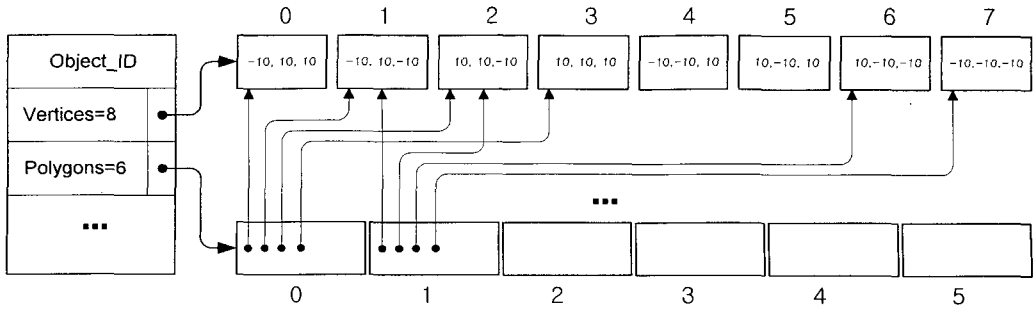

그림 2. 정육면체의 데이터 구조

가장 널리 사용되는 Z-buffer 알고리즘은 눈에 보이는 다각형 리스트 전체를 대상으로 평면방정식과 Z 보간법(interpolation method)을 이용하여 픽셀 단위로 전후 관계를 판단한다[11]. 깊이 정렬 알고리즘은 다각형 리스트 전체를 대상으로 다각형 단위로 전후 관계를 계산하는데, 실제로는 삼각형 단위로 대표 Z 값을 계산하고 이 값을 기준으로 삼각형들을 정렬하는 방식을 사용한다.

2.4 기타 요구사항들

PDA와 같은 모바일 플랫폼에서는 상대적으로 CPU 연산 능력이 부족하고 이동성을 강조한 저전력 설계의 메모리 속도 한계 등으로 인하여 요구사항들이 추가로 발생하게 된다. 모바일용 CPU들은 FPU (Floating Point Unit)를 가지고 있지 않기 때문에 소프트웨어적으로 실수를 N:M 형태의 고정소수점수로 변환하여 계산한 다음 다시 부동소수점수로 전환하는 방식을 사용하여야 한다. 이때 주어진 유효자리 숫자를 잘 고려하여 오버플로우(overflow)나 언더플로우(underflow)가 발생하지 않도록 적절한 비트를 할당하여야 한다[8]. 연산 속도를 빠르게 하기 위하여 나눗셈을 위한 역수표나 행렬 변환을 위한 삼각함수표를 미리 저장해 두어야 하며, 메모리 접근을 줄이기 위하여 래스터라이져과 같이 시간에 민감한 함수 내에서는 임시 변수의 사용을 줄이고 레지스터형 변수를 사용하여야 한다.

3. 깊이 정렬 알고리즘과 Z-버퍼 알고리즘

깊이 정렬 알고리즘과 Z-버퍼 알고리즘은 다음과 같은 특징을 가지며 현재 가장 널리 사용되는 알고리즘들이다.

3.1 깊이 정렬 알고리즘

화가들이 캔버스에 그림을 그릴 때 원거리의 객체들을 먼저 그리고, 근거리의 객체들을 그리게 되는데, 이러한 원리를 래스터라이져에 적용한 것이 페인터의 알고리즘(painter's algorithm)이다. 이 알고리즘의 원리는 모든 다각형들을 그들의 대표 Z 값을 기준으로 정렬한 다음, 가장 큰 Z 값을 갖는 다각형을 가장 먼저 그리고 가장 작은 Z 값을 갖는 다각형을 가장 마지막에 그린다는 것이다. 하나의 다각형을 대표하는 Z 값을 정할 때 존재하는 정점들의 최소, 최대 혹은 평균값을 취하는 방법들을 고려해 볼 수 있으나, 그림 3에서와 같이 이 세 가지의 방법들 중 어느 하나도 완전한 방법은 없다. 오류가 생긴 경우에는 대표 Z 값의 순서와는 역으로 다각형이 그려져야 바른 표현이 된다. 이 중에서는 평균 Z 값을 사용하는 경우가 보통 최선의 결과를 보여주므로 실험 과정에서 사용하였다.

완전한 페인터 알고리즘에서는 그림 3과 같은 문제점을 해결하기 위하여 모든 다각형의 쌍에 대하여 다음과 같은 다섯 개의 검사를 단계적으로 진행하여 순서 오류를 수정하여 올바른 결과를 얻는다[6].

- Z 범위 검사 : 다각형 정점들의 Z 범위가 오버랩(overlap)되면 다음 검사를 진행한다.
- X 범위 검사 : 다각형 정점들의 X 범위가 오버랩되면 다음 검사를 진행한다.
- Y 범위 검사 : 다각형 정점들의 Y 범위가 오버랩되면 다음 검사를 진행한다.
- 먼 다각형 검사 : Z 값이 더 큰 다각형 A가 다른 다각형 B보다 정말 멀리 위치하는지 검사한다. 이 검사는 A의 각 정점들을 B의 평면 방정식으로 검사한다. 결과가 틀리다면 다음 검사를 진행한다.
- 가까운 다각형 검사 : B가 A보다 가까이 위치하는지 검사한다. 이 검사 결과가 틀리다면 다각형은 명확히 틀린 순서이므로 순서를 바꾸어야 한다.

이러한 검사들은 정확성을 보장해 주기는 하지만 애매한 특수 상황도 많아서 실제로 구현된 프로그램은 모바일 플랫폼에서는 감당할 수 없을 정도로 느리고 혼란스러워진다. 예를 들어 위의 검사 결과로 순서를 바꾼 경우 현재의 다각형 쌍에 근접한 다각형들에 이 결과가 미치는 영향을 생각해 보라.

완전한 페인터 알고리즘 대신 단순한 Z 값 정렬만을 사용하는 깊이 정렬 알고리즘은 정확성을 보장하지는 못하지만 현존하는 래스터라이져 알고리즘 중 가장 빠르므로 특수한 응용에서는 사용할만한 충분한 가치가 있다.

3.2 Z-버퍼 알고리즘

Z-버퍼 알고리즘은 가장 널리 사용되는 방법으로 소프트웨어는 물론 대부분의 하드웨어 그래픽 가속기에서도 이 방법을 사용할 만큼 정확하고 이론적으

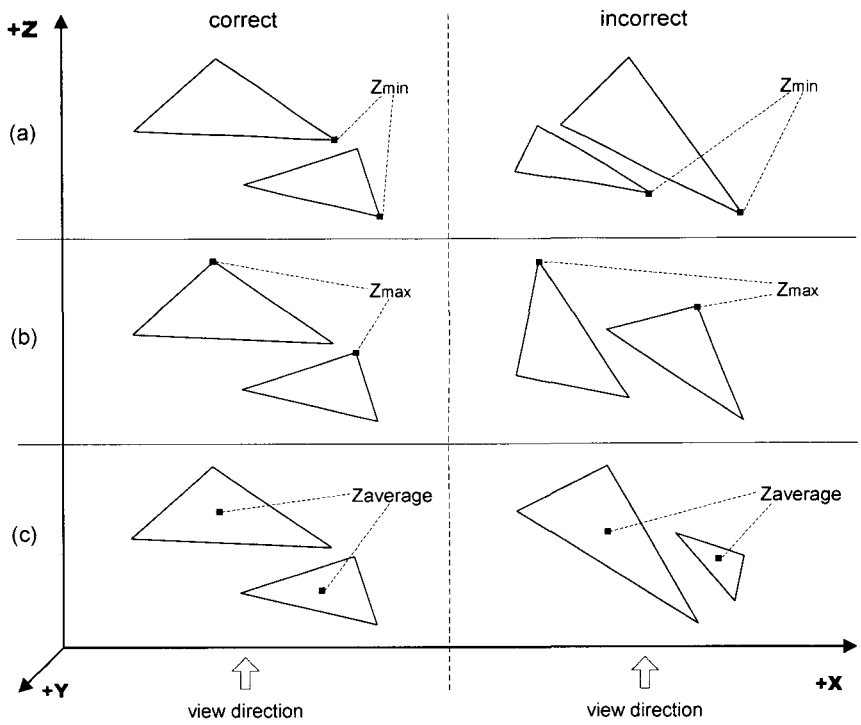

그림 3. 대표 Z 값의 선택 : (a) 최소 Z 값, (b) 최대 Z 값, (c) 평균 Z 값

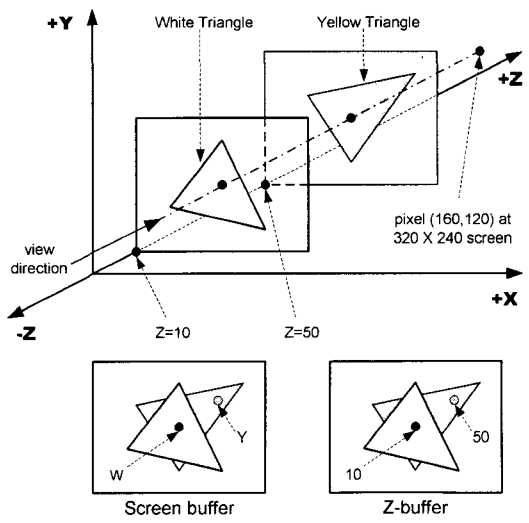

그림 4. Z-버퍼 알고리즘

로 완전한 방법이다. 그림 4와 같이 먼저 화면 버퍼와 동일한 크기의 Z-버퍼를 준비하여 아주 큰 값으로 초기화시키고, 각 다각형의 각 픽셀에 대한 Z 값을 평면방정식과 보간법을 이용하여 계산한다. 임의 순서(arbitrary order)로 선택된 다각형들에서 계산된 픽셀의 Z 값이 Z-버퍼에 현재 저장되어 있는 값보다 작다면 계산된 픽셀이 카메라와 더 가까운 것이므로 Z-버퍼의 값을 갱신하고 화면 버퍼의 동일한 위치에 픽셀 값을 저장한다. 모든 다각형에 대하여 이러한 처리를 반복하고 나면 정확한 최종 결과를 얻을 수 있다. 결국 Z-버퍼 알고리즘은 픽셀 단위의 깊이 정렬 알고리즘이며, 다각형은 결국 개개 픽셀 이하로 나누어질 수 없으므로 이론적으로 항상 정확한 랜더링 순서를 제공한다. 또한 페인터의 알고리즘에 비해 랜더링할 객체의 양에 덜 민감하게 실행된다.

모바일 플랫폼에서 소프트웨어로 구현된 Z-버퍼 알고리즘은 320×240 정도의 작은 화면 크기 덕분에 소요 메모리는 큰 문제가 되지 않지만, 모든 다각형에 대한 픽셀 단위의 계산량이 큰 부담이 된다.

4. 적응형 Z-버퍼 알고리즘

페인터의 알고리즘은 다각형들을 대표하는 Z 값을 기준으로 정렬하고 정확성을 보장해 주기 위하여 다섯 단계의 검사 과정을 거친다. 이 검사 과정은 Z

축에서의 순서가 틀릴 가능성이 있는 다각형들을 선별하여 일일이 순서 관계를 규명하는 수행을 하게 된다. 이 과정은 애매한 상황들을 모두 고려해야 하기 때문에 알고리즘이 혼란스러워지고 더불어 시간적인 부담을 극복하기가 어려워진다. 따라서 단순한 깊이 정렬 알고리즘이 설득력을 얻을 수 있었다. 그러나 그림 5와 같은 다각형의 교차 상황은 깊이 정렬 알고리즘은 물론 페인터의 알고리즘으로도 제대로 처리할 수 없으며, 주어진 면에 대한 픽셀 단위의 정렬 방식을 진행하는 Z-버퍼 알고리즘을 사용해야 한다.

이와 같은 이유들로 인하여 Z-버퍼 알고리즘의 사용은 필연적이라고 생각되지만 모든 다각형들에 대하여 Z-버퍼 알고리즘을 적용하는 것은 실험 결과 시간적인 부담을 모바일 플랫폼에서 감당하기가 어려운 것으로 나타났다.

그림 6과 같은 다각형 집합을 생각해 보자. 여기에서 모든 다각형들의 Y 범위가 오버랩된다고 가정한다. 삼각형 A와 B는 X 범위가 겹치지만 Z 범위가 겹치지 않으며, 삼각형 B와 C는 Z의 범위가 겹치지만 X의 범위는 겹치지 않는다. 삼각형 A와 C는 X와 Z 범위 어느 것도 겹치지 않는다. 삼각형 G는 A, B, C와 X 범위는 겹치지만 Z 범위는 전혀 관련이 없다. 반면에 삼각형 D와 F는 X, Y, Z 범위가 모두 겹치지만 정렬 순서는 바르며, 삼각형 E와 F는 X, Y, Z 범위가 모두 겹치고 정렬 순서가 틀려서 F가 먼저 그려지는 에러가 발생한다.

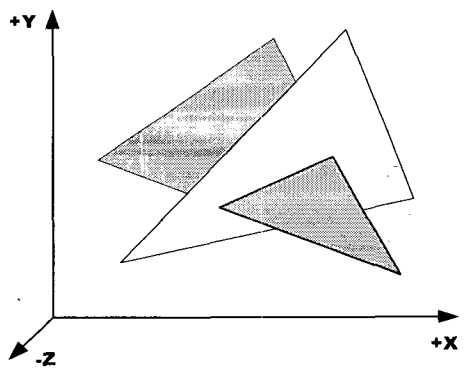

그림 5. 다각형의 교차

현재 랜더링되어져야 할 리스트에서 모든 다각형들이 Z-버퍼 알고리즘 연산의 대상이 되는 것은 아니다. 페인터의 알고리즘에서 상위 세 개 단계의 검사와 같이 X, Y, Z 3개의 축에 대한 정점들의 좌표 범위가 동시에 서로 겹치지 않는다면 다각형의 순서가 바뀔 가능성은 없다. 그림 6에서 삼각형 A, B, C, G는 순서 역전의 가능성이 없는 그룹에 속한다. 정점들의 범위 검사의 결과로 순서 역전 가능성이 있는 다각형 그룹에는 D, E, F가 속하며 순서를 정확하게 얻으려면 완전한 페인터의 알고리즘을 구현하는 방법을 사용하여야 한다. 물론 Z-버퍼 알고리즘을 이 그룹에 적용하면 바른 순서를 얻으려고 노력할 필요는 없다.

이 연구에서 제안하는 적응형 Z-버퍼 알고리즘의 골격은 우선 평균 Z 값을 이용하여 다각형들을 하향식 깊이 정렬을 실행한 다음, 순서 역전 가능성이 있는 다각형들을 전부 찾아낸다는 것이다. 결국 모든 다각형들은 순서 역전 가능성이 있는 그룹과 가능성이 전혀 없는 그룹으로 양분된다. 이 그룹들을 구분하기 위하여 별도의 연결 리스트(linked list)를 유지할 필요는 없으며, 하나의 비트 플래그 필드만으로

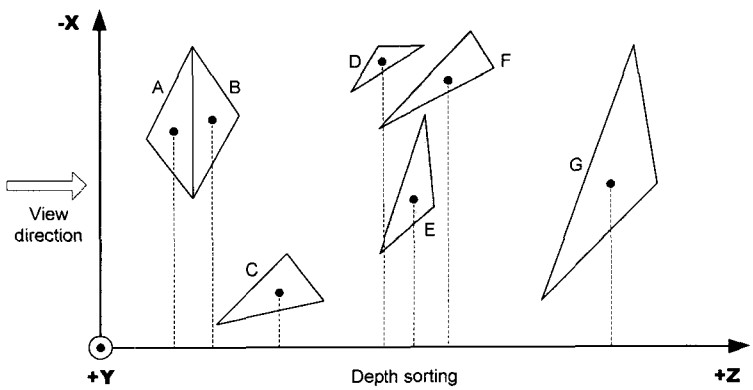

그림 6. 다각형 집합

충분하다. 최종적으로 Z 값으로 정렬된 다각형들을 순서대로 검사하여 역전 가능성이 있는 것으로 판정되었다면 Z-버퍼 알고리즘을 실행시키고 역전 가능성이 없다면 별다른 처리 없이 화면 버퍼에 직접 랜더링하면 된다. 개략적인 적응형 Z-버퍼 알고리즘은 다음의 그림 7과 같다.

순서 역전 가능성이 없는 그룹에 속하는 다각형들은 별도로 Z-버퍼에 다각형의 깊이 정보를 저장할 필요가 없으며, 깊이 정렬 결과에만 의존하여 화면 버퍼에 직접 랜더링된다. 그러나 순서 역전 가능성이 있는 다각형들은 별도의 논리적인 그룹을 형성하여 자신들끼리만 Z-버퍼에 깊이 정보를 저장하면서 동시에 화면 버퍼에 랜더링되도록 하면 된다. 이렇게 해도 이 그룹들과 역전 가능성이 있는 그룹들간에는 소그룹 단위의 정렬 순서는 틀리지 않으므로, 결국 화면 버퍼에는 올바른 순서로 그려지게 되는 것이다.

이 방법은 순수한 Z-버퍼 알고리즘만큼 명쾌하고 균등한 구조를 가지지는 않지만, 깊이 정렬 알고리즘의 신속성과 Z-버퍼 알고리즘의 정확성을 Trade-off시킨 알고리즘이라 할 수 있다. 여기에서 필요한 메모리는 Z-버퍼 알고리즘보다 많지 않으나 두 개의 그룹들이 객체의 상황에 따라 균일하게 분포되는 것

```
        . . .
 1 :    for all polygon P
 2 :        compute Zpaverage              //get average Z value of all vertices in P
 3 :        Poverlap = 0          //initialize flag of P overlapping
 4 :    next
 5 :    quick sort to polygon list PL by Zaverage
 6 :    for all polygon Pf          //Pf is First Polygon
 7 :        compute Xfmax, Xfmin // from all vertices in Pf
 8 :        compute Yfmax, Yfmin
 9 :        compute Zfmax, Zfmin
10 :        for Ps=Pf + 1 to Pl//Ps is Second Polygon. Pl is Last Polygon
11 :                compute Xsmax, Xsmin          // from all vertices in Ps
12 :                compute Ysmax, Ysmin
13 :                compute Zsmax, Zsmin
14 :                Xmax = max(Xfmax, Xsmax)
15 :                Xmin = min(Xfmin, Xsmin)
16 :                Ymax = max(Yfmax, Ysmax)
17 :                Ymin = min(Yfmin, Ysmin)
18 :                Zmax = max(Zfmax, Zsmax)
19 :                Zmin = min(Zfmin, Zsmin)
20 :                if ( (Xmax - Xmin + 1) < ((Xfmax - Xfmin + 1)+(Xsmax - Xsmin + 1)-1) &
21 :                    (Ymax - Ymin + 1) < ((Yfmax - Yfmin + 1)+(Ysmax - Ysmin + 1)-1)  &
22 :                    (Zmax - Zmin + 1) < ((Zfmax - Zfmin + 1)+(Zsmax - Zsmin + 1)-1) )
23 :                        Pfoverlap = Pfoverlap + 1
24 :                        Psoverlap = Psoverlap + 1
25 :                end if
26 :        next
27 :    next
28 :    for all polygon P
29 :        if Poverlap
30 :                call Draw_Polygon_by_Zbuffer();
31 :        else
32 :                call Draw_Polygon();
33 :        end if
34 :    next
        . . .
```

그림 7. 적응형 Z-버퍼 알고리즘

표 1. PDA 플랫폼 별 평균 fps

platform	processor	Z-buffer	depth sort	adaptive Z-buffer
IPAQ 3660	206MHz ARM SA1110	14.20	52.13	**27.83**
IPAQ 5550	400MHz INTEL XScale	21.10	59.37	**41.12**
HX 4700	624MHz INTEL PXA270	33.60	69.77	**52.60**

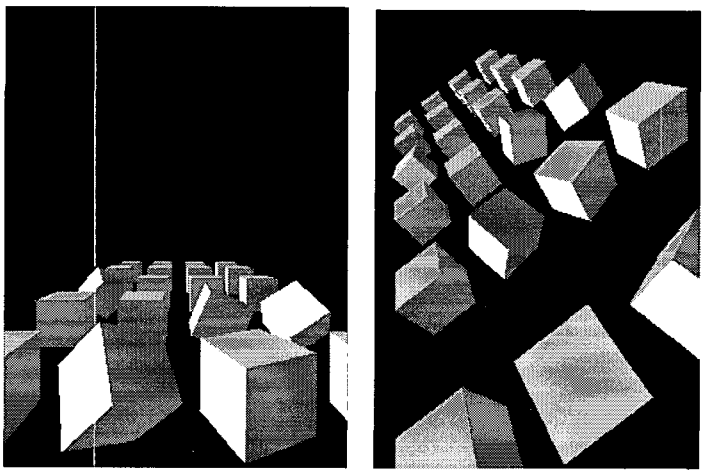

그림 8. 실행 화면 (240×320)

은 기대하기 어려우므로 프레임 단위의 계산량의 불균형이 생기게 된다. 이러한 불균형은 3D 그래픽의 역동성과 장면 복잡도(scene complexity)로 인해 어느 알고리즘에서도 자유로울 수 없으며, 프레임에 독립된 여러 개의 객체들이 등장할 경우 Z-버퍼 알고리즘보다 유리할 것으로 판단된다.

5. 실험 결과

실험을 위하여 Microsoft™사의 Embedded C++ version 3.0을 사용하여 프리프로세서, 트랜스폼 파이프 및 래스터라이저를 포함하는 전체 3D 그래픽 엔진을 구현하였다. 이 때 계산을 최소화하기 위하여 환경 조경과 직접 조명에 의한 발산(diffusion)만을 모델링하여 사용하였다. 또한 고정소수점을 사용하였으므로 복잡한 자리수 변환에 대한 부담을 덜기 위하여 카메라 근접 면에 대해 Z의 역수를 이용한 보간으로 왜곡 보정을 하지 않았으나 별도로 텍스쳐 매핑(texture mapping) 모듈은 구현하지 않았으므로 눈에 띄는 큰 왜곡은 발생하지 않는것으로 나타났다.

PDA 플랫폼은 IPAQ 3660, IPAQ 5550 그리고 HX 4700을 사용하였으며, 28개의 정육면체 객체를 이용하여 총 168개의 다각형을 대상으로 뷰 포인트(view point) 및 뷰 각도(view angle), 객체의 임의 회전 등의 변화를 다양하게 주면서 장시간 실행시키고 평균 fps(frames per second)를 구한 결과는 표 1과 같다.

그림 8은 실제 실행 화면을 나타낸 것으로 실험결과에 의하면 적응형 Z-버퍼 알고리즘은 깊이 정렬 알고리즘보다는 느리지만 정확성을 보장받을 수 있고, Z-버퍼 알고리즘에 비해서는 약 2배 정도의 빠른 속도를 보였다.

6. 결 론

이 연구의 목적은 그래픽 가속기는 물론 FPU와 같은 하드웨어적인 지원이 부족하고 그래픽 라이브러리의 지원도 미미한 PDA 플랫폼에서 3D 그래픽 소프트웨어 엔진을 구현하고, 엔진의 핵심 부분인 래스터라이저의 성능을 향상시켜 PDA에서도 3D 게임과 같은 그래픽 응용 프로그램을 개발할 수 있는 기반을 구축하는데 있다. 데스크탑 PC에 비해 프로세싱 파워와 메모리 접근성 등이 떨어지는 플랫폼에서

소프트웨어만으로 3D 그래픽 엔진을 구현하는 것은 많은 문제점이 존재한다.

현재 가장 널리 사용되는 Z-버퍼 알고리즘은 정확성에 있어서는 다른 알고리즘에 비해 탁월하지만 실험 결과에서 드러나듯이 속도 면에서는 현재 성능의 PDA에 적용하기에는 무리가 따르는 것으로 나타났다. 다소 정확성을 희생하더라도 속도 면에서 유리한 방법으로는 단순화된 페인터 알고리즘을 들 수 있다. 단순화된 페인터 알고리즘 즉, 깊이 정렬 알고리즘은 모든 다각형의 면들을 Z 값만을 기준으로 하향식으로 정렬하여 랜더링하므로 속도 면에서는 탁월하지만 면들의 전후 순서 역전으로 인한 오류를 피할 수 없다. 우리는 이 두 알고리즘의 장점인 정확성과 신속성을 Trade-off 시킨 적응형 Z-버퍼 알고리즘을 제안하고 구현하여 실험하였다. 적응형 Z-버퍼 알고리즘은 평균 Z 값을 이용하여 모든 다각형들을 하향식 깊이 정렬을 실행한 다음 순서 역전 가능성이 있는 다각형들을 전부 찾아내어 플래그에 표시를 해두고, 정렬된 다각형들을 Z 값이 큰 것부터 순서대로 검사하여 역전 가능성이 있는 것에 대해서는 Z-버퍼 알고리즘을 실행시키고 역전 가능성이 없는 다각형에 대해서는 화면 버퍼에 직접 랜더링하는 방법을 사용한다. 이 방법은 알고리즘 구조면에서는 순수한 Z-버퍼 알고리즘만큼 단순하고 명쾌하지는 않지만 약 2배 정도의 속도를 보여주었다.

적절한 텍스쳐 매핑 및 광원의 거울 반사, 앤티앨리어싱(anti-aliasing) 등에 대한 모듈 구현과 최근 3D 게임에 많이 사용되는 BSP(Binary Space Partition) 알고리즘에 대한 개선 연구는 추후에 진행할 예정이다. BSP 알고리즘은 재귀 알고리즘(recursive algorithm)을 통하여 다각형의 집합을 이원 트리 구조체로 전환하는 방식으로 렌더링 순서를 고속으로 결정할 수 있는 알고리즘이다[12]. 이 연구를 통하여 구현된 3D 그래픽 엔진을 스마트폰과 같은 플랫폼에 포팅하고 지속적으로 성능 개선에 대한 연구를 진행하여 급변하는 모바일 환경에 적합한 고성능 엔진 개발에 더욱 노력하고자 한다.

참 고 문 헌

[1] Hill S, "The lazy z-buffer," *Information Processing Letter*, No. 55, pp. 65-70, 1995.

[2] Slater M, Drake K, Davison A, Kordakis E, Billyard A, and Miranda E, "A statistical comparison of two hidden surface techniques : the scan-line and Z-buffer algorithms," *Comput Graph Forum*, No. 11, pp. 131-138, 1992.

[3] Newman WM, Sproull RF, *Principals of Interactive Computer Graphics (2nd edition)*, McGraw-Hill, 1979.

[4] Rogers DF, *Procedural Elements for Computer Graphics (2nd edition)*, McGraw-Hill, 1985.

[5] L Bishop, D Eberly, T Whitted, M FInch, and M Shantz, "Designing a PC Game Engine," *IEEE Computer Graphics and Applications*, Vol. 18, No. 1, pp. 46-53, Jan. 1998.

[6] A LaMothe, *Black Art of 3D Game Programming*, The Waite Group, 1995.

[7] K C Finney, *3D Game programming All in One*, Thomson Learning, 2004.

[8] B Hook, *Building a 3D Game Engine in C++*, John Wiley & Sons, 1995.

[9] Hertel S, Mehlhorn K, "Fast triangulation of the plane with respect to simple polygons," *Information and Control*, No. 64, pp. 52-76, 1985.

[10] Anders Kugler, "The Setup for Triangle Rasterization," *11th Eurographics Workshop on Computer Graphics Hardware*, pp. 1-10, Aug. 1996.

[11] E Lengyel, *Mathematics for 3D Game Programming and Computer Graphics (2nd edition)*, Charles river media, 2004.[1] Gordon D, Chen S, "Front-to-back display of BSP trees," *IEEE Comput Graph*, Appl 11, pp. 79-85, 1991.

[12] Gordon D, Chen S, "Front-to-back display of BSP trees," *IEEE Comput Graph*, Appl 11, pp. 79-85, 1991.

김 효 철

1987년 경북대학교 전자공학과 (공학사)
1989년 경북대학교 전자공학과 (공학석사)
2002년 경북대학교 컴퓨터공학과(공학박사)
1989년~1996년 국방과학연구소 (ADD) 근무
1996년~현재 계명문화대학 컴퓨터인터넷학부 교수
관심분야 : 멀티미디어 보안, 3D 게임, 임베디드 시스템

김 대 영

1983년 경북대학교 전자공학과 (공학사)
1985년 경북대학교 전자공학과 (공학석사)
1992년 경북대학교 전자공학과 (공학박사)
1985년~1986년 (주)금성전기 근무
1991년~현재 계명문화대학 멀티미디어학부 교수
관심분야 : 모바일 게임, 3D 영상 인식 등